Estratégia de Automação em Testes: Requisitos, Arquitetura e Acompanhamento de sua Implantação

Mozart Guerra Costa

Trabalho Final de Mestrado Profissional em Computação

Instituto de Computação
Universidade Estadual de Campinas

Estratégia de Automação em Testes: Requisitos, Arquitetura e Acompanhamento de sua Implantação

Mozart Guerra Costa

Fevereiro de 2004

Banca Examinadora:

- **Profa. Dra. Eliane Martins (Orientadora)**
  Instituto de Computação - Unicamp
- **Prof. Dr. Marcos Chaim**
  Escola de Artes, Ciências e Humanidades - USP
- **Prof. Dr. Ricardo Anido**
  Instituto de Computação - Unicamp
- **Profa. Dra. Cecília Baranauskas (Suplente)**
  Instituto de Computação – Unicamp

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Costa, Mozart Guerra

C823e Estratégia de automação em testes: requisitos, arquitetura e acompanhamento de sua implantação / Mozart Guerra Costa -- Campinas, [S.P. :s.n.], 2006.

Orientadora : Eliane Martins

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software – Testes. 2. Software – Qualidade. 3. Engenharia de software. I. Martins, Eliane. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Estratégia de Automação em Testes: Requisitos, Arquitetura e Acompanhamento de sua Implantação

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigido e defendido por Mozart Guerra Costa e aprovado pela Banca Examinadora.

Campinas, 23 de fevereiro de 2004

Profa. Dra. Eliane Martins
(Orientadora)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Computação na área de Engenharia de Computação

Aos meus amados pais

Agradecimentos

Agradeço especialmente a Karina, minha esposa que tanto amo, por toda a força me dada: incentivando-me desde o início, compreendendo a minha ausência devido à dedicação a este trabalho e também por contribuir com o mesmo através de várias considerações.

Aos meus filhos queridos, Daniel, Julia e Bruno, agradecer e desculpar-me por uma ausência ainda maior do que a que já somos forçados pela nossa vida profissional. Adoro estar com vocês, assim como já adoro estar com o bebê que está chegando.

Serei sempre agradecido a meus pais, Jandyra e Moisés, por toda a estrutura e afeto me dada quando adulto eu ainda não era, muitas vezes a base de sacrifícios. Deles, ainda hoje recebo incentivo para continuar estudando e crescendo.

Agradeço às minhas irmãs, Jaqueline, Cita e Mônica por todo carinho e amizade.

Agradeço a Denise Kato e a Raiffe pelo socorro de última hora e a Adilsom e Fábio pelos toques em vários momentos.

Agradeço a dedicação de Eliane, minha orientadora, sem ela eu talvez ainda estivesse dando voltas sem saber ao certo para onde ir. Sempre direcionando e apoiando-me com muita ética e profissionalismo.

Por fim agradeço a todos os colegas e amigos que direta ou indiretamente me ajudaram a construir este trabalho.

Resumo

O teste é uma atividade que ajuda ao atendimento da crescente demanda por uma maior qualidade do software. A automação traz a possibilidade de tornar o teste mais rápido e efetivo em encontrar erros, desde que se tenha atenção quanto ao que automatizar e como deve ser feita esta automação. Este trabalho final busca organizar diversos requisitos de automação em uma arquitetura composta por seis funcionalidades: planejamento dos testes, construção dos artefatos, execução, análise de falhas, medições e gerência de configuração dos artefatos de teste. São apresentados modelos que servem como guia para a automação, tais como: o relacionamento entre os artefatos, as ferramentas responsáveis pelos artefatos e as necessidades de integração entre as ferramentas. Além disso, são identificadas as necessidades levantadas supridas pelas ferramentas já existentes na empresa do campo de estudo deste trabalho e são relatadas as iniciativas para solucionar as necessidades não atendidas diretamente pelas ferramentas.

Abstract

Testing is an activity that helps to meet the increasing demand for better quality software. Automation can make testing faster and more effective in error identification, provided proper attention is given to what and how to automate. The objective of this dissertation is to organize several automation requirements in an architecture made up of six functionalities: test planning, artifact construction, execution, failure analysis, measurements, and set up management of test artifacts. It also introduces models that work as a guideline for automation such as artifact relationship, tools responsible for artifacts, and tool integration requirements. In addition, it verifies whether the needs in question have been addressed by tools already available at the company and highlights initiatives to solve whatever has not been directly achieved by these tools.

Índice

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução e motivação

Com o passar do tempo a tecnologia da informação se fez cada vez mais presente nas mais diversas atividades humanas, chegando a se tornar crucial para o funcionamento de algumas áreas, o que levou se exigir cada vez mais qualidade dos softwares desenvolvidos. Durante o desenvolvimento de um software erros podem ser inseridos em qualquer fase do projeto, desde o levantamento junto ao usuário, passando pela modelagem, até a construção do código. Estes erros podem ser oriundos de diversas questões técnicas, como por exemplo: uma programação mal feita, uma modelagem inadequada, a má comunicação entre as pessoas das áreas de informática e as pessoas das áreas usuárias ou mesmo dentro da própria área de informática. A atividade de teste destaca-se como crítica entre aquelas que têm como objetivo evitar que esses erros cheguem à produção ([Pre95]).

A indústria de software precisa atender a clientes cada vez mais exigentes com a qualidade dos produtos e se manter num mercado extremamente competitivo. A atividade de teste está entre as que mais consome tempo no desenvolvimento de software. É comum uma organização de software gastar 40% do esforço total do projeto em teste ([Pre95]). Por esta razão os testes precisam ser mais rigorosos em encontrarem falhas e consumir o menor tempo possível para permitir que se entregue os produtos mais rapidamente, utilizando-se menos recursos e com menos falhas.

Algumas medidas podem ser tomadas para obter testes melhores, mais rápidos e baratos. Podemos citar: sistematizar as atividades de testes, definir os papéis e responsabilidades vinculadas ao teste, antecipar a preparação dos testes já durante as fases de construção, conhecer as técnicas relacionadas ao tema, testar características diferentes do software nas diferentes fases de testes, estimar adequadamente os esforços para os testes, ter um controle efetivo das falhas encontradas e automatizar os processos de testes.

Muitas tentativas de automatizar os processos de testes falham, talvez por não considerar que a automação de testes é similar a um projeto de desenvolvimento de software e a

possibilidade de fracasso é grande se não tivermos uma boa definição de requisitos (para [Som03] “Requisitos correspondem à especificação do que deve ser implementado. São descrições de como o sistema deve se comportar ou as definições e propriedades e atributos de um sistema. Podem ser restrições no processo de desenvolvimento de um sistema"); de design; o uso de boas práticas em codificação; e a dedicação total do time de automação nesta tarefa ([Kan97], [Nag00], [Pet01a]).

A automação dos processos de testes deve focar naquelas atividades repetitivas, que feitas manualmente são sujeitas a erros, consomem muito tempo e exige dos testadores paciência redobrada para repetir os mesmos passos dos testes e conferir os resultados. O investimento feito em automação somente trará retorno se os testes automatizados forem repetidos diversas vezes. Também se deve considerar a automação parcial de um teste, por exemplo, automatizar a execução e deixar a verificação para ser feita manualmente ([Pet01b]).

A automação dos testes não pode ser implementada com sucesso se não existirem processos bem definidos para os testes ([Pet01b]). Além disso, os profissionais precisam conhecer as técnicas de testes, devem estar estabelecidos os documentos que devem ser preenchidos, ou seja, não se pode automatizar sem uma cultura razoável sobre testes.

A automação pode atingir diversas atividades do teste como, por exemplo, o planejamento dos testes, o desenvolvimento dos artefatos de teste, a execução dos testes, a análise de falhas, as medições dos testes e os versionamentos dos artefatos dos testes ([Eic96]).

Escolher adequadamente e adquirir ferramentas de testes não significa garantia de sucesso no processo de automação. O ponto fundamental é como será o uso destas ferramentas. Simplesmente disponibilizá-las e utilizá-las sem uma estruturação adequada não trará os benefícios esperados ([Nag00]).

1.1 Objetivos

Este trabalho tem como principal objetivo apresentar e acompanhar a implementação de uma estratégia de automação de testes para softwares em diversas plataformas: mainframe, cliente-servidor e aplicações web.

Estes estudos serviram para nortear o processo de implementação da automação de testes em uma empresa, que é o campo de trabalho deste estudo, que já possuía uma metodologia de

testes (chamada de MTS – Metodologia de Testes de Sistemas) com processos e templates definidos, profissionais treinados nas técnicas de testes e diversas ferramentas de testes já adquiridas.

A estratégia de automação de testes proposta é composta por uma arquitetura de automação, os requisitos para cada área de automação e por um workbench de testes que contém: os artefatos de testes, as ferramentas de testes com suas responsabilidade pelos artefatos e as integrações necessárias entre as ferramentas. Tudo isso terá como base análises feitas a partir da metodologia de testes existente, pois a automação deverá ser conduzida para facilitar os processos de testes já definidos e praticados.

Relatamos as ferramentas encontradas no campo deste estudo e como elas atendem ou não a alguns dos requisitos e integrações desejadas pela automação. Por fim, comentamos algumas implementações, encaminhamentos e planos futuros de tratamentos de algumas das pendências relacionadas à automação de testes.

1.2 Estrutura da Dissertação

No capítulo 2 é feita uma revisão bibliográfica em que são apresentados os principais conceitos relacionados ao teste de software: o objetivo, as técnicas, os critérios, princípios e estratégias do teste de software. É apresentada também uma visão geral sobre automação de teste de software, assim como as dificuldades, limitações, vantagens e sugestões de quando a automação é indicada e como deve ser conduzida.

A automação proposta será baseada nos processos de testes já definidos na metodologia de testes da empresa deste campo de estudo (MTS). No capítulo 3 apresentamos alguns dos principais conceitos da metodologia de testes e relacionamos alguns dos processos de testes e documentos definidos pela metodologia.

No capítulo 4 apresentaremos inicialmente uma arquitetura de automação de testes contendo as macro-funções de testes que devem ser automatizadas. Depois, serão relacionados os requisitos de automação agrupados pelas macro-funções de testes, estes requisitos têm como objetivo suprir as necessidades dos processos já existentes. Na segunda metade do capítulo 4 passamos a descrever modelos relacionados à automação. Estes modelos descrevem características básicas dos artefatos de testes e das ferramentas de automação, como: que

ferramenta vai atender a automação de cada artefato de teste e que integrações são necessárias entre as ferramentas.

No capítulo 5 descrevemos as ferramentas já disponíveis na empresa relacionadas ao teste de software e a que requisitos de automação do capítulo 4 serão responsáveis pelo atendimento. Relataremos o que é atendido das funcionalidades e das integrações desejadas.

No capítulo 6 apresentaremos as soluções práticas que foram encaminhadas para os requisitos de automação que as ferramentas não atendem. Serão abordadas soluções que já foram implantadas contendo uma avaliação se foi satisfatória, outras que estão em andamento e também decisões quanto a não implementar alguns requisitos da automação.

No capítulo 7 são apresentadas as conclusões e sugestões de continuidade do assunto.

Capítulo 2

Teste e automação de teste

Neste capítulo apresentaremos uma visão geral com os principais conceitos relacionados ao assunto teste. O padrão IEEE 829-1990 ([Iee90]) define os termos: defeito (*fault*, *bug*) é um passo, processo ou definição de dados incorreta; engano (*mistake*) é uma ação humana que produz um resultado incorreto; erro (*error*) é uma diferença entre o valor obtido e o esperado; e falha (*failure*) é uma produção de uma saída incorreta de acordo com a especificação. Neste trabalho, os termos engano, erro e defeito serão referenciados como defeito (causa); e o comportamento incorreto do programa será referenciado como falha (conseqüência).

Para Myers teste é o processo de executar um programa com o objetivo de encontrar defeitos ([Mye79]). Outra definição é a do padrão IEEE 829-1983 ([Iee83]), na qual teste é o processo de analisar um item de software para detectar as diferenças entre as condições existentes e as requeridas (isto é, erros) e avaliar as características do item de software: É uma definição mais maleável que a de Myers, pois utiliza o termo "item de software" no lugar de "programa" e acrescenta "avaliar as características do item de software". Item de software pode ser um programa, um componente, uma classe, um objeto, uma sub-rotina ou todo um aplicativo. Por "avaliar características do item de software" pode-se entender como medir e avaliar algumas características funcionais e não funcionais do software, como desempenho ou a carga máxima que o software pode atender.

Pressman aborda como projetar testes: o nosso objetivo é projetar testes que descobrem sistematicamente diferentes classes de erros e fazê-lo com uma quantidade mínima de esforços ([Pre95]). Ou seja, não é para desperdiçar esforços projetando testes que não irão descobrir novos erros, quanto mais enxuto os testes melhor desde que não se abdique de encontrar novos erros.

Estas definições centradas em que o objetivo do teste é o de descobrir erros estão corretas, porém poderíamos considerar que alguns testes não têm como principal objetivo descobrir erros, mas mostrar que o sistema atende aos objetivos para que fora construído. Esta diferença é descrita na definição: "Os testes para detecção de defeitos têm como finalidade expor defeitos

latentes em um sistema de software antes de o sistema ser entregue e isso contrasta com os testes de validação que se destinam a mostrar que um sistema cumpre com as suas especificações" ([Som03]).

Vamos contextualizar o teste, para Pressman o teste de software é um elemento de um aspecto mais amplo, que é freqüentemente referido como *verificação e validação* (V&V). Verificação se refere ao conjunto de atividades que garante que o software implementa corretamente uma função específica. Validação se refere ao conjunto de atividades que garante que o software construído corresponde aos requisitos do cliente. Estes dois conceitos são importantes na qualidade de software, nos quais os testes estão inseridos.

Em outras palavras, por Validação entende-se responder às questões: estou construindo um sistema que é o que esperam? Entendi perfeitamente o que o usuário espera deste sistema? O usuário sabe qual será o sistema que será entregue? Ou seja, o sistema está atendendo aos requisitos do usuário / cliente? A Verificação tenta determinar se estou corretamente construindo o sistema, se estou utilizando corretamente as técnicas de desenvolvimento de software como modelagem, especificação e programação.

Apesar de o teste desempenhar um papel extremamente importante em V&V, outras atividade são também necessárias. O teste oferece o último reduto no qual a qualidade pode ser avaliada e mais pragmaticamente os erros podem ser descobertos, porém a qualidade deve ser incorporada ao software durante todo o processo de engenharia de software. A aplicação adequada de métodos e ferramentas, as revisões técnicas formais efetivas e a gerência e a medição sólida, todas levam à qualidade que é confirmada durante o teste ([Pre95]).

As revisões ou inspeções de software analisam e verificam as representações do sistema, como o documento de requisitos, os diagramas de projeto e o código fonte do programa. Trata-se de uma técnica estática de V&V porque não requerem que o sistema seja executado ([Som03]). As revisões de software funcionam como uma espécie de filtro para o processo de garantia de software. Isto é, as revisões são aplicadas em vários pontos durante o desenvolvimento de software e servem para descobrir erros e defeitos que podem depois ser removidos ([Pre95]).

O teste de software é uma outra atividade de V&V que envolve executar uma implementação do software com os dados de teste e examinar suas saídas e seu comportamento operacional, a fim de verificar se está sendo executado conforme o esperado. Trata-se de uma

técnica dinâmica de V&V porque trabalha com uma representação executável do sistema ([Som03]).

Revisões e testes não são atividades concorrentes em V&V. Elas têm as suas vantagens e desvantagens, se complementam e devem ser utilizadas em conjunto no processo de verificação e validação ([Som03]).

2.1 Estratégia de Teste

Conhecer e aplicar as técnicas de testes não é suficiente para garantir que bons testes serão realizados; também é necessário sabermos se estas técnicas devem ser utilizadas em um projeto, além de quando, como e quem irá utilizá-las, em outras palavras precisa-se definir uma estratégia de teste para cada projeto de software. Uma estratégia deve definir quais as atividades, além de quem serão os envolvidos, quando e por quanto tempo serão realizadas estas atividades. Para que uma estratégia de testes tenha sucesso é necessário observar alguns aspectos que foram apontados por [Pre95], [Het87], [Som03] e [Mye79]:

Os requisitos não funcionais[1] devem ser especificados de forma mensurável. Estes requisitos, que tratam dos atributos de qualidade, devem ser quantificáveis e não ambíguos. Por exemplo, não se deve dizer que o sistema deve ter um bom desempenho. O que significa ter um bom desempenho? Neste caso, deve-se explicitar em números, talvez em um intervalo, o tempo de resposta que se considera bom e, se possível, também aquele que se considera aceitável. Com isso espera-se evitar a subjetividade na avaliação dos resultados dos testes.

Da mesma forma os requisitos funcionais[1] devem ser não ambíguos e possíveis de serem testados. Eles devem ser completos, precisos e testáveis, de tal maneira que seja possível elaborar o plano e a especificação dos testes. Para garantir a qualidade dos requisitos devem ser realizadas revisões técnicas formais sobre o documento de requisitos.

Os objetivos do teste devem ser enunciados explicitamente no plano de teste. Um plano de teste deve conter uma significativa quantidade de informações, tais como quem serão os responsáveis pelo teste, que tipos de testes serão realizados, quando e quantas horas serão

[1] Os termos requisitos funcionais e não funcionais serão definidos na seção 3.2.

dedicadas a cada tipo, o número de falhas previstas de serem encontradas, quantos ciclos de testes são previstos e a cobertura esperada.

O teste deve focar no uso real do produto. Nunca é demais lembrar que o software é feito para satisfazer as necessidades dos usuários e não dos desenvolvedores. Deve-se procurar entender como os usuários vão utilizar o produto e especificar casos de testes que sejam baseados neste uso.

Um outro fator para que uma estratégia de testes tenha sucesso é buscar sempre assegurar a qualidade e a estabilidade por construção, evitando falhas e minimizando os testes necessários. Devemos sempre pensar na possibilidade de construir um software que "perceba" quando uma falha está ocorrendo e tome alguma atitude. Por exemplo, num sistema de resgate de aplicações em um banco, podem ser inseridas linhas de código para verificar que caso o resgate seja maior que o dobro da aplicação inicial, suspenda o resgate, pois seria quase impossível uma valorização tão grande num determinado período de tempo.

Como dissemos, faz-se necessário realizar revisões ou inspeções de software antes dos testes. Não se pode esperar ter sistemas com qualidade realizando apenas testes para aferir sua qualidade. Também se faz necessária a realização de revisões desde o início do projeto, em todos os documentos. As revisões têm como características a possibilidade de poder encontrar vários erros ao mesmo tempo e, principalmente, permite que tais erros sejam encontrados em um momento próximo de quando foram inseridos. Deixar para localizar e corrigir estes erros somente no momento dos testes pode ser muito tarde. Quando se trata de erros de programação talvez não seja tão traumático achá-los nos testes, porém quando nos referimos aos erros motivados por um mau levantamento ou por um projeto mal concebido, descobri-los no momento dos testes pode significar ter de refazer muita coisa. As revisões também devem ocorrer nos documentos de testes como os que definem a estratégia de testes e o projeto de casos de testes.

Por fim, deve-se determinar o desempenho dos testes e melhorar o processo. Meça o desempenho (tamanho, prazo, esforço, qualidade e eficácia) dos testes, registre formalmente todas as falhas encontradas durante a realização dos testes. Estas informações serão úteis para melhorar o processo de testes e o processo de desenvolvimento.

Fases de Teste

Para [Pre95] os testes devem ser divididos em fases que possuem relação com as fases de desenvolvimento, a figura 2.1 demonstra a relação entre as fases de testes e as fases de desenvolvimento de software. Os testes devem começar nas unidades de um sistema, uma vez que temos a garantia de que as unidades funcionam corretamente, devem ser testadas as interfaces entre essas unidades, realizando a integração dos componentes. Posteriormente serão testadas as funcionalidades de mais alto nível que envolva diversas unidades, garantindo que o software atenda aos requisitos funcionais e não funcionais. Essa diretriz de iniciar os testes pelas unidades tem uma razão: não tem sentido testar diversas unidades em conjunto se não se tem uma mínima certeza de que estas unidades funcionam adequadamente sozinhas.

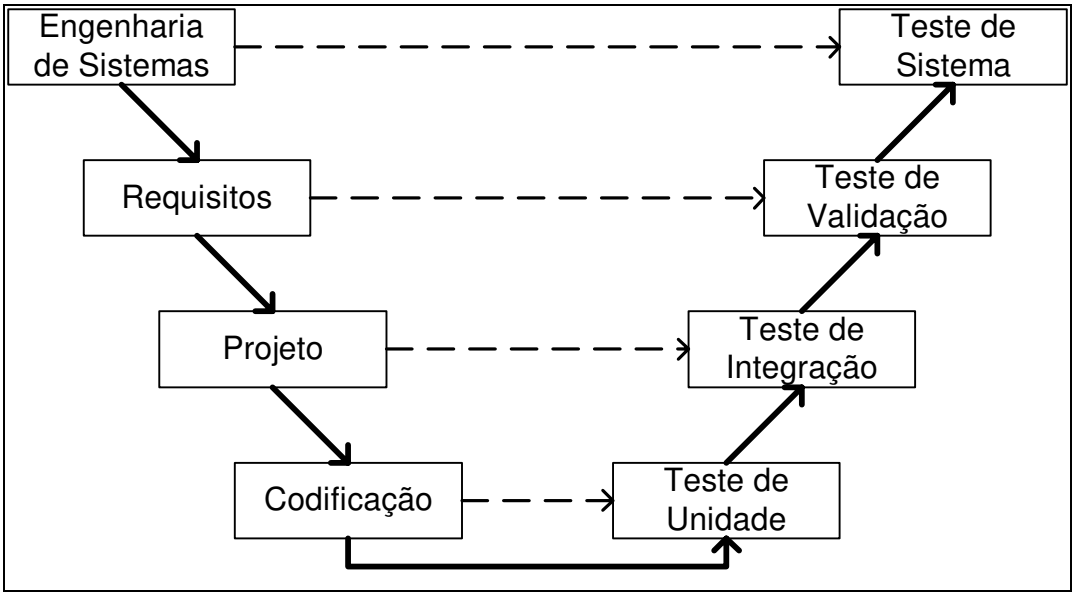


Figura 2.1: Fases de Testes e Fases de Construção

A primeira fase de teste é chamada de Teste de Unidade em que o teste é focado em cada módulo individualmente, não importando a sua interação com os outros módulos. Após os testes nas unidades temos a fase de Testes de Integração em que são testadas as interfaces entre os módulos na medida em que se constrói a integração entre eles. Existem algumas abordagens quanto à estratégia de integração: integrar todas as unidades ao mesmo tempo, o que é desaconselhável; integração incremental ascendente (*bottom-up*) em que os módulos são integrados na ordem de baixo para cima na estrutura hierárquica do sistema; integração

incremental descendente (*top-down*) em que os módulos são integrados de cima para baixo também segundo a hierarquia e a integração combinada que utiliza os dois métodos de integrações incrementais citados.

Durante os testes de unidade e de integração algumas funcionalidades de mais baixo nível foram testadas, porém os testes de unidade são mais focados na estrutura do código e os testes de integração na interface entre os módulos. Será na fase de Teste de Validação que o foco passa a ser testar as funcionalidades do sistema. Agora com os módulos devidamente testados e integrados podem ser examinadas as funcionalidades que envolvem vários deles.

A última fase de teste proposta na estratégia definida por Pressman é a fase de Teste de Sistema, que tem como objetivo explorar o comportamento do software num contexto mais amplo com outros elementos como pessoas, hardware e banco de dados. Nesta fase devem ser projetados casos de teste que busquem identificar a presença de falhas relacionadas à integração entre todos os elementos do sistema. Alguns tipos de testes sugeridos por [Bei84] são os testes de recuperação em caso de falha, os testes de segurança quanto à invasão imprópria, os testes de stress e os testes de desempenho.

A depuração é uma atividade que não é de teste, mas que ocorre em decorrência dos testes bem sucedidos e que, portanto, encontram falhas. Em cada fase de teste vai ocorrer a atividade de depuração, assim como as atividades de planejamento, preparação, execução e avaliação dos resultados dos testes.

Um tipo de teste muito comum é o teste de regressão que segundo [Het87] é um teste seletivo para verificar se as alterações efetuadas não provocaram efeitos colaterais adversos, ou para verificar se um sistema modificado continua atendendo às necessidades preestabelecidas. Assim, quando um sistema sofre uma alteração, deve-se fazer não só testes que verifiquem a alteração realizada, mas também testes que verifiquem se a alteração não provocou que outras partes do sistema deixassem de funcionar corretamente.

2.2 Técnicas e Critérios de Teste

O teste completo não é possível: a variação de caminhos possíveis dentro de um programa assim como a variação do domínio das entradas é muito grande, o que torna praticamente impossível cobrir todas as combinações de caminhos de um programa ou todas as combinações

de suas entradas ([Mye79]). No projeto de casos de teste devem ser utilizadas técnicas que permitam encontrar o maior número de falhas com um número mínimo de casos de teste, ou seja, não é para projetar um caso de teste que se espera encontrar a mesma falha que outro já encontra.

Existem diversas técnicas que auxiliam na especificação dos casos de testes, estas são divididas em dois grupos: os testes caixa branca e os testes caixa preta. Estes tipos de testes são complementares e é indicado o uso tanto de testes caixa branca como de testes caixa preta, pois os tipos de falhas encontrados por eles são de naturezas diferentes ([Roc01]).

Se o teste completo é impossível até que ponto se deve testar? Para isso temos o critério de teste, que é um predicado a ser satisfeito para se considerar a atividade de teste como encerrada. Dada uma técnica de teste temos, em geral, a ela associada diferentes critérios de teste. O critério de teste auxilia tanto na criação dos casos de testes através do fornecimento de diretrizes para a geração dos dados de entrada, como também na determinação se os testes executados são satisfatórios, ou seja, se satisfazem a um determinado critério de teste ([Roc01]).

2.2.1 Testes Caixa Preta

Os testes caixa preta ou testes funcionais são aqueles baseados nos requisitos, não interessando qual a estrutura interna do software. Os testes são projetados visando encontrar discrepâncias entre o que o software faz contra as suas especificações ([Pre95], [Mye79]).

O nome "caixa preta" advém do fato que não se considera como o software está construído, não se "enxerga" o que ele tem por dentro. Para a especificação das entradas e saídas esperadas dos testes é levada em consideração a especificação do software. Deve-se preocupar com as funcionalidades e características do software e não com os detalhes de implementação ([Bei90]). As técnicas de testes funcionais mais conhecidas são:

- **Métodos de testes baseados em grafo** ([Pre95], [Bei90], [Mye79]): A relação entre os principais objetos do sistema é desenhada em um grafo e casos de testes são projetados para mostrar que essas relações não estão sendo satisfeitas.
- **Particionamento de equivalência** ([Pre95], [Mye79]): Consiste em dividir as entradas de um programa em partições nas quais se espera, apesar de não se ter certeza absoluta, que o programa tenha um comportamento similar para todos os elementos pertencentes à mesma

partição. Casos de testes são projetados utilizando-se um elemento de cada partição, diminuindo assim a quantidade de casos de teste necessários.

- **Análise de valor limite** ([Pre95], [Mye79]): Esta técnica baseia-se na técnica de particionamento de equivalência, sendo que ao invés de selecionar qualquer elemento da classe de equivalência, os casos de testes são projetados utilizando-se os elementos pertencentes aos limites das partições, mais os elementos imediatamente inferiores e os imediatamente superiores a estes elementos limites. Isto é feito assim porque a experiência mostra que existe uma alta probabilidade de se encontrar falhas nestes limites.
- **Teste de comparação** ([Pre95]): Consiste em criar sistemas redundantes que seriam submetidos a conjuntos idênticos de casos de teste e a saída esperada de ambos deve ser a mesma. Esta técnica tende a ser utilizada em casos especiais para sistemas críticos devido ao custo elevado de redundar os sistemas.

2.2.2 Testes Caixa Branca

Os testes caixa branca são também conhecidos como testes estruturais e focam nas estruturas internas do software. Eles são projetados baseados no código do programa e têm como objetivo exercitar seus caminhos, condições, ciclos e estruturas de dados. Os testes caixa branca são mais utilizados nas fases iniciais do teste principalmente nos testes de unidades e em menor escala nos testes de integração ([Pre95]). As principais técnicas de testes caixa branca são:

- **Teste de caminho básico**: Esta técnica tem como objetivo garantir que todos os comandos de um programa tenham sido exercitados pelo menos uma vez. Para cumprir este objetivo com um menor conjunto de casos de teste possível faz-se uso do conceito de caminhos básicos, que é um conjunto mínimo de caminhos a partir do qual qualquer outro caminho pode ser derivado. A quantidade de caminhos básicos de um programa é dada pela complexidade ciclomática do programa, que mede a sua complexidade lógica. Para facilitar a identificação dos caminhos básicos e do cálculo da complexidade ciclomática utiliza-se um grafo de fluxo que mostra o fluxo de controle lógico de um programa. A complexidade ciclomática determina a quantidade máxima de caminhos básicos e também a quantidade máxima de casos de teste que deverão ser projetados para que todos esses caminhos sejam percorridos nos testes ([Pre95]).

- **Teste de condição**: Tem como objetivo examinar as condições lógicas contidas em um programa, para isso projeta-se casos de testes que têm como objetivo testar estas condições. O objetivo do teste de condição é encontrar erros não só nas condições do programa, mas também erros não relacionados às condições, isto porque um conjunto de casos de testes capaz de encontrar erros nas condições de um programa provavelmente também será efetivo em encontrar outros tipos de erros ([Pre95]).
- **Teste de fluxo de dados**: Esta técnica é baseada nas variáveis e seus comportamentos dentro de um programa. Temos diversas possibilidades de estratégias de fluxo de dados, a mais comum é a de definição-uso. Por cadeia definição-uso entende-se como um caminho entre uma definição e um uso de uma variável sem que haja outra definição no meio deste caminho. Numa estratégia definição-uso devemos preparar casos de testes que exercitem as cadeias definição-uso das variáveis de um programa ([Pre95]).

2.2.3 Critérios de Testes

Um critério de teste define qual é o conjunto de elementos requeridos do software que deve ser exercitado ([Roc01]). Um critério pode ser utilizado tanto para selecionar ou gerar um conjunto de casos de teste, neste caso chamado de critério de seleção ou de geração; como pode ser utilizado para avaliar a qualidade de um conjunto de casos de teste, chamado de critério de cobertura ou de adequação.

Quando se executa um conjunto de casos de teste pode-se avaliar a cobertura para determinar o percentual de elementos requeridos pelo critério de teste que fora exercitado por este conjunto. Caso não se alcance a cobertura desejada, explicitada no critério de teste, pode-se adicionar mais casos de teste ao conjunto de casos de teste para que alguns dos elementos ainda não exercitados sejam testados ([Roc01]).

Como exemplos de critérios associados às técnicas caixa branca podemos citar: que todas as instruções sejam executadas pelo menos uma vez; ou que cada nó, que cada arco e cada caminho do grafo do programa sejam executados ao menos uma vez; ou que pelo menos um caminho entre todas as associações definição-uso de cada variável seja executado pelo menos uma vez. Um exemplo de critério associado às técnicas funcionais é o de exigir que cada classe de equivalência seja exercitada por pelo menos um caso de teste. Vale observar que podem ser

utilizadas as técnicas funcionais para criar os casos de teste e utilizar um critério de cobertura relacionado à estrutura do software para determinar se foi alcançada a cobertura desejada.

2.3 Automação de Testes

Nos últimos anos ferramentas e técnicas têm melhorado a produtividade no desenvolvimento de software, com isto tem aumentado a pressão por uma maior produtividade também nos testes, a automação de teste é uma das maneiras de se conseguir isso ([Pet01a]). Nesta seção iremos introduzir o assunto automação de testes abordando as dificuldades, custos, erros e sugestões para que se tenha sucesso no tema. Não iremos entrar em detalhes para as questões técnicas de implementação da automação, isto será feito no capítulo 4.

Não é incomum a automação de testes não corresponder às expectativas em relação aos ganhos que se espera com a sua implantação. Embora acene com a possibilidade de nos levar a uma situação extremamente produtiva, a implementação de testes automatizados pode criar tantos problemas quanto resolvê-los ([Pet01b]).

Uma constatação é que um projeto de automação é tão suscetível a falhar quanto qualquer outro projeto de software. Eles freqüentemente falham mais porque muitas organizações não dedicam o mesmo cuidado e profissionalismo a automação como o fazem para os projetos de desenvolvimento de produtos ([Bac96], [Pet01b]). Não levam em consideração, por exemplo, a necessidade de estabelecer requisitos, de construir um projeto e de utilizar boas práticas de programação nos projetos de automação.

Para muitos autores como [Bac96] e [Kan97] essa falsa expectativa e a falta de uma maior dedicação nos projetos de automação seriam frutos dos mitos e falsas promessas feitas pelos vendedores de ferramentas de teste e por quem não conhece o suficiente de teste. Diz-se, por exemplo, que pessoas com pouca experiência em programação podem usar estas ferramentas para criar facilmente os testes e que mantê-los não é um problema.

A automação de teste é uma atividade que requer um esforço dedicado e não compartilhado com outras tarefas. Os responsáveis pela automação devem ter conhecimentos tanto de desenvolvimento como de teste ([Pet01a]). Para que se tenha sucesso é importante que se entregue a tarefa de automatizar os testes a programadores seniores ([Kan97]). Estas afirmações vão de encontro ao que geralmente se faz: entrega-se a tarefa de automação a estagiários ou a

pessoas que atuam na automação nos tempos que sobram de outras tarefas. É necessário que os profissionais conheçam bem tanto de desenvolvimento de software, já que o que vão fazer é realmente um desenvolvimento, como possuam também conhecimentos em testes, já que são as atividades de testes que se espera automatizar.

Assim como ocorrem em diversas outras áreas que se deseja automatizar, deve-se manter uma cuidadosa distinção entre a automação e o processo que ela automatiza ([Bac96]). A automação é uma boa idéia, mas para que dê certo é preciso pensar primeiro no teste e depois na automação. A automação é somente uma das várias estratégias para se alcançar um teste de software eficiente ([Bac96]). Entre outras medidas para aumentar a produtividade nos testes podemos citar: a determinação de um processo padronizado e a capacitação dos profissionais nas técnicas de testes.

Nem todas as atividades de teste devem ser automatizadas por dois motivos: o primeiro é que nem tudo pode ser automatizado, algumas das atividades de teste podem ser executadas por ferramentas como a execução e a avaliação dos resultados, outras só podem ser executadas por humanos como projetar os casos de testes e depurar o código fonte em caso de erro ([Kan00]). O segundo motivo é que, mesmo para aquelas atividades em que o uso de ferramentas é possível, não se deve sentir impelido a automatizar tudo do teste, deve-se procurar o que vai dar maior retorno. Por exemplo, pode-se automatizar a execução e deixar a verificação para ser feita manualmente, ou vice-versa ([Pet01b]). Se quiser ter sucesso, deve-se focar em obter rapidamente uma automação que possa ser utilizada várias vezes. Os testes automatizados custam de 3 a 10 vezes mais para criar, verificar e documentar, do que os testes manuais [Kan97].

Um caso de teste possui quatro atributos que descrevem a sua qualidade, ou seja, o quão bom é o caso de teste ([Few01]): a eficácia (que é a capacidade do caso de teste em encontrar erros), a exemplaridade (que é capacidade de encontrar diferentes tipos de erros), a economia (que são os recursos necessários para que seja executado, analisado e depurado) e a sua manutenibilidade (o quão fácil é a sua manutenção a cada evolução do software). A eficácia e a exemplaridade de um caso de teste independem se o teste será executado manual ou automaticamente. Quanto à economia um teste automatizado é mais caro nas primeiras execuções e depois tende a ser mais barato que o teste manual. Em geral, o caso de teste automatizado tende a custar mais para ser criado e mantido.

Para atenuar os custos da automação é importante identificar as situações em que ela é indicada. [Kan98] aborda este tema sugerindo um checklist a que se deve submeter um projeto candidato a automação. Dentre as questões a serem respondidas estão:

- A interface do usuário da aplicação está estável ou não?
- Espera-se vender este produto em múltiplas versões?
- Espera-se que o produto esteja estável quando liberar esta versão, ou deverá ter mais versões para correção de erros?
- Sua empresa desenvolve outros produtos de maneira similar? Existirão outras oportunidades de amortizar o custo do desenvolvimento da automação?
- Quão freqüentemente se recebem novas construções do software?
- Será necessário rastrear os casos de testes para trás até os requisitos e mostrar que cada requisito tem casos de testes associados?
- É sua empresa sujeita a auditoria ou inspeções por organizações que preferem ver extensivos testes de regressão?
- Que tipos de testes são realmente pesados em sua aplicação? Como a automação pode fazer estes testes mais fáceis de conduzir?

Uma vez decidido que os testes de um projeto serão automatizados, alguns cuidados precisam ser tomados. Como dissemos boa parte das tentativas de automação dos processos de testes falharam e [Kan98] relaciona os erros mais comuns que levaram ao insucesso muitas dessas iniciativas, eis alguns deles:

- Não subestime o custo da automação.
- Não espere ser mais produtivo em curto prazo.
- Não deixe de encontrar erros para automatizar casos de testes: lembrar que o objetivo do teste é encontrar erros, a automação deve ajudar neste objetivo e não se sobrepor a ele.
- Não automatize casos de testes simples demais: priorize os testes mais complexos de serem executados manualmente e mais propensos a achar erros.

- Não busque por 100% de automação: o ideal é mesclar testes manuais e automatizados.
- Não automatize scripts (programas que irão automatizar os passos da execução do teste) isoladamente, tente reaproveitar partes em comum: a reusabilidade de scripts é uma dos pontos chave da automação e tem de ser devidamente tratada.
- Não crie scripts de testes que serão difíceis de manter em longo prazo.
- Não esqueça de tratar a automação de testes como um autêntico projeto de programação, ou seja, são necessários requisitos, design, práticas de codificação, etc.
- Não "esqueça" de documentar o projeto: para a automação ser reutilizada deve estar devidamente documentado.

Para quem está iniciando, [Few01] propõe uma estratégia para aprender e evoluir na automação dos processos de testes:

- Comece com poucos, mas distintos testes que tenham a capacidade de encontrar diferentes tipos de erros.
- Automatize-os em uma versão antiga e estável do software, talvez o faça algumas vezes explorando diferentes técnicas e abordagens. Neste momento o objetivo é descobrir o que a ferramenta pode fazer e o quão fácil é implementar, analisar e manter diferentes testes a serem automatizados.
- Execute os testes numa versão posterior, mas ainda estável, do software para explorar as questões de manutenção. Isto pode levar a observar diferentes maneiras de implementar testes automatizados que evitem ou ao menos reduzam os custos de manutenção.
- Execute os testes numa versão não estável do software para que se possa aprender o que está envolvido na análise de falhas e explorar conhecimentos para fazer esta tarefa mais fácil e, portanto, reduzir o esforço de análise.
- Não adicione, sem critério, mais e mais casos de testes. Isso pode ocasionar uma desnecessária duplicação e um elevado custo de manutenção.

São muitas as sugestões que se encontra na literatura de como tirar o maior proveito da automação de testes, passamos a citar mais algumas delas.

Como geralmente a automação exige que os testes sejam executados primeiro manualmente, a maior parte dos erros é encontrada durante os testes manuais. Entretanto, pode-se aumentar o número de erros encontrados pela automação executando certos tipos de testes, tais como: rodar os mesmos testes através de diferentes versões do sistema operacional ou em diferentes configurações, rodar dois programas equivalentes para os mesmos testes, usar testes aleatórios para exercitar um código instrumentado na busca de situações estranhas ([Kan98]).

A maior parte das vezes não se terá retorno do investimento feito em automação dos testes na versão em que está criando a automação, pois os ganhos aparecerão quando ocorrer a repetição dos testes nas versões posteriores. Eis exemplos de como obter ganhos em curto prazo com a automação: automação do teste de fumaça (teste para verificar se existem condições mínimas de realizar os demais testes), automação dos testes de configuração (se a configuração do hardware está adequada) e automação do teste de stress (testes de altos volumes em curto espaço de tempo) ([Kan98]).

Para ser eficiente e manutenível é preciso primeiro desenvolver uma estrutura de automação para partes já estáveis: para tanto, selecionam-se produtos para automatizar garantindo que a manutenção será factível ([Kan98]). Ao se automatizar os casos de testes, é indicado que haja independência entre eles, cada caso de teste deve conter o necessário para rodar sem depender dos demais ([Pet01b]).

Pelo que vimos até agora podemos afirmar que, segundo diversos autores, a automação de testes vale à pena desde que não se criem expectativas irreais em torno dela e que se saiba quando, onde e como automatizar. Nem tudo é para ser automatizado, deve-se começar pequeno e não se pode esperar retorno imediato da automação. Deve-se dedicar seriamente à tarefa e entregá-la a profissionais experientes e com dedicação total. Não se pode acreditar que somente a automação será capaz de aumentar a produtividade dos testes, a definição de um processo padronizado de testes e a capacitação dos profissionais são outras iniciativas que devem preceder a automação. A automação pode tornar os processos de testes mais rápidos, desde que estes estejam estabelecidos e executados por profissionais capacitados.

Como trabalho relacionado temos [Cun04] que apresenta um sistema para automação da execução de teste funcional chamado AutoTest. O AutoTest implementa vários dos requisitos de

automação de teste que serão abordados neste trabalho e deixa de implementar algumas áreas como é o caso do planejamento de testes e dos testes estruturais. Na área da execução dos testes utiliza as arquiteturas orientadas a Palavras-Chave, Data-Driven e Record & Playback (estes termos serão apresentados na seção 4.2.3). Também facilita o processo de execução com a seleção de que testes executar e o acompanhamento da sua execução. Além disso, permite a execução de testes em paralelo, testes remotos e disponibiliza a visualização dos erros e das falhas após a execução do teste.

Fora a área de execução do teste, o Autotest possui funcionalidades relacionadas a outras áreas abordadas neste trabalho como é o caso da construção dos artefatos de teste onde os arquivos de dados para o teste podem ser facilmente criados e manipulados pelo projetista. Na análise das falhas permite o acesso a banco de dados para verificação das falhas e a fácil extensão dos pontos de verificação de se houve ou não uma falha. Também atende à gerência de configuração de testes com a integração com sistemas de controle de versão. Por fim, atende a requisitos da área de medições com relatórios como o das falhas encontradas nos testes.

2.4 Resumo do Capítulo

Neste capítulo definimos e contextualizamos o teste em relação às outras atividades de qualidade de software. Apresentamos uma estratégia de testes com as várias fases de testes. Relatamos algumas técnicas de testes caixa preta e caixa branca e os critérios associados a elas. Na segunda parte do capítulo introduzimos o assunto automação de testes, abordando as dificuldades, custos, erros e sugestões relacionadas ao tema.

Capítulo 3

Os Processos de Testes já Existentes

Neste capítulo iremos apresentar resumidamente os principais pontos e definições dos processos de testes existentes na organização que foi o campo de estudo deste trabalho. Os requisitos de automação que serão apresentados no capítulo 4 devem ter como objetivo suprir as necessidades oriundas destes processos de testes surge então a necessidade de apresentar estes processos. Não será apresentada toda a metodologia de testes (chamada MTS – Metodologia de Testes de Sistemas), mas aquilo que consideramos essencial para justificar a automação desejada.

Além das atividades relacionadas ao teste de software, a metodologia de testes MTS, apesar do que o nome indica, propõe que sejam executadas atividades de revisões nos produtos, documentos e códigos fontes durante o desenvolvimento do projeto.

3.1 As Fases da Metodologia MTS

A MTS segue o princípio do "V" que estabelece que para cada fase de construção no desenvolvimento de sistemas deve existir uma fase de teste correspondente ([Som03]). Por exemplo, aquilo que é definido nos levantamentos com o usuário será testado na fase de homologação e o que é especificado para os programas serão testados nos testes de unidade.

Um outro princípio utilizado pela metodologia é que os testes podem e é indicado que sejam planejados e projetados antes das fases de testes: uma vez que os requisitos estejam definidos, já podemos especificar os testes que devem ser realizados para validá-los. Não precisamos esperar a fase de testes para projetá-los, esta especificação pode ser feita antecipadamente, durante as fases iniciais do projeto em que os requisitos estão sendo construídos.

Unindo estes dois princípios podemos afirmar que aquilo que é feito em uma determinada fase de construção será testado numa fase de testes correspondente e estes testes podem ser planejados e preparados ainda na fase de construção. Por exemplo, o que é definido nos

levantamentos com o usuário será testado na fase de homologação e a especificação desses testes pode ser realizada ainda na fase de levantamento. Da mesma maneira, quando estamos especificando os programas já podemos especificar os testes de unidade que serão realizados para validar estes programas. No Planejamento de testes deve ser utilizado o documento Plano de Teste, que será apresentado na seção 3.4.1. A preparação dos testes pode ser feita no documento GRT – Gestão e Rastreabilidade dos Testes a ser apresentado na seção 3.4.2. e no documento Roteiro de Testes a ser apresentado na seção 3.4.3.

Vamos apresentar como a MTS define cada uma das fases de testes:

Fase de Teste de Unidade: Verificação de um componente de software através de testes caixa preta e caixa branca para verificar se o sistema satisfaz os requisitos detalhados[2] especificados.

Fase de Teste de Integração: Verificação das interfaces entre componentes de um software, através de testes caixa preta e caixa branca para verificar se o sistema satisfaz aos requisitos detalhados especificados.

Fase de Teste de Sistema: Fase de testes na qual é considerado um sistema integrado de hardware e software para verificar se o sistema satisfaz os requisitos de usuário, funcionais e não funcionais especificados. As funcionalidades testadas nesta fase são de um nível mais alto do que aquelas testadas nas fases de testes anteriores e por vezes envolve a utilização conjunta de dois ou mais sistemas. Estudamos também nesta fase concorrência, desempenho, segurança e volume.

Fase de Teste de Homologação: Fase de testes executada pelos usuários, na qual são verificados os requisitos de usuário e funcionais referentes aos critérios de aceitação para permitir ao cliente determinar se aceita o sistema ou não. É a validação de um software pelo comprador, pelo usuário ou por terceira parte, com o uso de dados ou cenários especificados ou reais. O teste de homologação deve testar basicamente o negócio, e não se ater em campos (estes foram feitos nos testes de unidade). Estes testes devem ser breves sem o intuito de encontrar erros de formatação, consistências de campos ou mesmo erros de definição de tamanhos.

Fase de Teste de Implantação: Fase de testes na qual são verificados os requisitos de negócio como, por exemplo, se com a implantação do sistema as vendas cresceram em 20%, se a agilização de um determinado processo foi alcançada. Verificam-se também os requisitos

[2] Os termos requisitos detalhados, funcionais, não funcionais, de usuário e de negócio serão definidos na seção 3.2.

referentes à diminuição do risco de solução de continuidade do negócio devido à implantação do sistema.

3.2 Testes Baseados em Requisitos

Nas definições das fases de testes da seção anterior pode-se observar que são propostos testes baseados em requisitos. Isto significa que o desenvolvimento dos projetos também deverá ser focado em requisitos. Iremos apresentar outra definição para requisito, além da já apresentada no capítulo1. De acordo com o Glossário da IEEE 610.12-1990 ([Iee90]) requisito é:

1. Uma condição ou capacidade necessária para um usuário resolver um problema ou alcançar um objetivo;
2. Uma condição ou capacidade que deve estar presente num sistema para satisfazer um contrato, padrão, especificação ou outro documento formal;
3. Uma representação documentada de uma condição ou capacidade de acordo com as definições (1) e (2).

Para a MTS os requisitos podem ser de diferentes tipos (requisito de negócio, de usuário, funcional, não funcional e detalhado) e define cada um deles conforme mostrado a seguir.

Requisitos de Negócio: São focados no entendimento do por quê o projeto ou sistema é necessário e como contribuirá para os objetivos do negócio. Correspondem a objetivos, metas ou “desejos” da área de negócios. São escritos na linguagem da área de negócios e podem conter gráficos, tabelas e diagramas.

Requisitos de Usuário: Diz o que o usuário estará habilitado a realizar com o sistema ou produto. São escritos sob o ponto de vista do usuário e podem conter "*use-cases*", cenários e processos de negócios. Os Requisitos de Usuário derivam dos Requisitos de Negócio e servem como base para os Requisitos Funcionais e Não-Funcionais.

Requisitos Funcionais: Determinam quais as funcionalidades que serão disponibilizadas pelo sistema ou produto. Especifica, por exemplo, as ações, processamentos, respostas, cálculos e relatórios. Os Requisitos Funcionais são base para os Requisitos Não-Funcionais e Detalhados.

Requisitos Não-Funcionais: Referem-se às qualidades, características e restrições dos Requisitos Funcionais e podem referir-se também aos Requisitos de Negócio ou de Usuário. São bases para os Requisitos Detalhados. Exemplos de tipos de Requisitos Não Funcionais: Operacionais (aparência, facilidade de uso); Desempenho (tempo de resposta); Recuperação (o que precisa ser feito antes e depois de uma falha); Disponibilidade (razão entre o tempo durante o qual um software se mantém operacional); Segurança (a segurança, a confidencialidade e a integridade do sistema); Testabilidade (funcionalidades que devem ser apresentadas para facilitar a preparação, execução ou evidenciação de testes); e Portabilidade (o sistema pode ser portado para outras plataformas).

Requisitos Detalhados: Como os Requisitos Funcionais e Não-Funcionais serão implementados. São escritos sob a ótica dos desenvolvedores e contém especificações detalhadas sobre a implementação do sistema. Servem de base para o projeto físico e a programação.

Requisito de Teste

Um conceito importante para a MTS é o de requisito de teste. Um requisito de teste diz em algumas linhas como o requisito será testado. Um requisito pode ser decomposto em alguns requisitos de teste. Quando da execução dos testes, são os requisitos de teste que são validados e verificados e conseqüentemente e indiretamente, os requisitos também. O requisito de teste deve ser registrado no documento GRT – Gestão e Rastreabilidade dos Testes, que será apresentado na seção 3.4.2. No capítulo 4 entraremos em mais detalhes sobre somo deve ser a implementação da automação dos requisitos de testes.

A partir do requisito de teste podem ser especificados casos de testes. Os casos de testes são detalhamentos do requisito de teste, é no caso de teste que definiremos os valores de entrada e saída esperada para o teste, os passos necessários para a sua execução e o que deve ser feito antes e depois da execução do teste. Por exemplo, para uma aplicação que permita saques entre R$10,00 e R$ 500,00 múltiplos de 10, um requisito de teste poderia ser “Realizar saques superiores a R$ 500,00”. Este requisito de teste poderia gerar um caso de teste com valor de R$ 510,00, cujo resultado esperado é “saque inválido”. Um outro requisito de teste poderia ser “Realizar saques entre R$ 10,00 e R$ 500,00 não múltiplos de 10”. Este requisito de teste poderia gerar um caso de teste como R$ 15,00. A metodologia permite que, caso se deseje, não se registre

os casos de teste; a execução e o controle se o teste passou ou falhou pode ser feita diretamente no requisito de teste.

Revisões técnicas nos Requisitos

Como dissemos, para a MTS os testes caixa preta devem ser baseados em requisitos, surge uma questão: e se os requisitos não estiverem bem definidos? Vamos projetar testes que validem os requisitos, então é necessário que se tenha um mínimo de garantia de que estes requisitos estão bem definidos. Por isso a metodologia de testes considera fundamental que se faça revisões técnicas dos requisitos. Além disso, a revisão dos requisitos permite localizar e remover defeitos mais cedo e a menores custos, em comparação com o custo de remover o mesmo defeito em momentos posteriores.

A importância do documento de requisitos sugere um maior rigor na sua revisão. A falta de clareza nos requisitos, ou pior, a falta deles, é uma das razões mais freqüentes de insucesso na construção de sistemas. Devem ser questionados os requisitos evitando que requisitos incompletos, ambíguos, não realizáveis e não testáveis migrem para as fases seguintes de desenvolvimento.

Os requisitos devem ser revisados por alguém que não o seu autor, e com bons conhecimentos em requisitos e das técnicas de revisões de requisitos. As atividades de revisões de requisitos devem ser executadas nas fases iniciais do projeto no momento em que são gerados.

Os problemas encontrados nos requisitos pelas revisões devem ser registrados e acompanhados até o encerramento. Este registro e acompanhamento devem ser feitos no documento RF – Registro de Falhas, que veremos posteriormente.

3.3 Atividades de Qualidade da Metodologia

Nesta seção iremos apresentar algumas das atividades de testes e revisões para três fases da metodologia de desenvolvimento de sistemas. Escolhemos estas fases por considerarmos que possuem atividades representativas para o objetivo deste estudo.

Fase 1 – Definições Iniciais

É a fase de compreensão do problema; possui as atividades abaixo.

- Elaborar o Plano de Teste preliminar e o Plano de Teste de Implantação completo;
- Elaborar o documento de Gestão e Rastreabilidade dos Testes preliminar;
- Revisar Requisitos de Negócio, de Usuário e de Qualidade;
- Derivar Requisitos de Testes correspondentes;
- Realizar a fase Definições Iniciais para as Soluções de Teste Automatizadas;
- Elaborar o Plano de Testabilidade do Sistema.

Fase 4 – Detalhamento Técnico

Esta fase corresponde ao projeto físico da engenharia de software.

- Elaborar o documento de Gestão e Rastreabilidade dos Testes completo;
- Construir Roteiros de Teste;
- Preparar Ambiente (Hardware/Software);
- Revisar Documentos gerados na fase;
- Realizar a fase de Detalhamento Técnico para as Soluções de Teste Automatizadas.

Fase 6 – Teste de Unidade

É a primeira das fases de teste; listamos abaixo as suas atividades.

- Revisar o Plano de Teste de Unidade;
- Executar os Testes de Unidade (manual/automatizados);
- Verificar a conformidade dos Testes executados com os testes planejados (cobertura);
- Elaborar Indicadores de Testes;
- Aprimorar as Soluções de Teste Automatizadas.

3.4 Documentos de Testes

Nesta seção apresentaremos resumidamente os documentos pertencentes à metodologia de testes MTS. Atualmente estes documentos estão implementados em planilhas Excel (*templates*) e um fruto deste trabalho será o de definir quais ferramentas serão responsáveis por implementá-los.

3.4.1 Plano de Teste

Como vimos no capítulo 2 um dos princípios do teste é que se deve enunciar explicitamente os objetivos do teste em um Plano de Teste. Podemos dizer que o Plano de Teste atua no campo estratégico definindo, entre outros, quem irá realizar os testes, quais os tipos dos testes mais indicados para o projeto, quando deverão ser realizados e qual o esforço será dedicado aos mesmos ([Som03]).

Cada projeto deve possuir um Plano de Teste e ter o preenchimento iniciado desde o início do projeto com o levantamento dos primeiros requisitos, atualizado na medida em que são detalhados os requisitos e finalizado até a fase de Projeto Lógico. Será utilizado durante todo o projeto e tem como principal benefício assegurar um processo substanciado de estimativa e negociação de prazos, esforços e qualidade dos testes.

Cada fase de teste, que vimos na seção 3.1, deve ter as suas atividades planejadas. Assim, o Plano de Teste é subdividido em planos que correspondem a cada fase de testes: Plano de Teste de Implantação, Plano de Teste de Homologação, Plano de Teste de Sistema, Plano de Teste de Integração e Plano de Teste de Unidade. Para cada fase de teste deve ocorrer o cadastramento das seguintes informações:

- As horas previstas para a preparação dos testes, sejam roteiros manuais, sejam scripts automatizados de testes.
- As horas previstas para as execuções manuais ou automatizadas dos testes.
- Os tipos de testes previstos como: teste de carga, de regressão, caixa branca e caixa preta.

- A quantidade prevista de requisitos de testes.
- A quantidade de falhas previstas de serem encontradas e o tempo necessário para suas correções.
- A quantidade de horas para a revisão de produtos intermediários, para a construção de soluções automatizadas para testes e soluções para o aumento da testabilidade.
- Recursos humanos alocados com seus papéis e responsabilidades. É preciso definir quem serão os responsáveis pelas atividades de planejamento, preparação e execução dos testes. Além de quem serão os revisores e os responsáveis pela construção das soluções de testes automatizadas e para o aumento da testabilidade.
- Os treinamentos necessários para que os profissionais alocados tenham os conhecimentos adequados para exercer as atividades de testes.
- Recursos de hardware e software necessários para as atividades de testes.

3.4.2 Gestão e Rastreabilidade dos Testes (GRT)

Na medida em que os requisitos funcionais e não funcionais são definidos podemos estimar melhor os prazos e esforços para testes, isto implica analisarmos cada requisito gerado e derivá-los em requisitos de testes, estimar para cada requisito de teste quanto tempo será gasto para preparar, executar, evidenciar e gerenciar o teste necessário. Este e outros procedimentos são feitos no documento Gestão e Rastreabilidade dos Testes ou GRT.

O GRT é o principal documento de gestão dos testes, é um mapa que contém os requisitos com os riscos associados, os requisitos de testes associados a cada requisito, a fase de teste em que serão testados os requisitos, as horas estimadas para preparação do roteiro manual ou de scripts automatizados, as horas para a preparação de extratores e as horas para a execução dos testes. Também será neste documento que ocorrerá o controle se o teste passou ou falhou para quando se opta por não derivar casos de testes para os requisitos de teste.

3.4.3 Roteiro de Testes (RT)

O roteiro de testes é o documento em que serão descritos os casos de testes. Ele guia o testador na execução do teste descrevendo as preparações (Ex: carga de uma base), os passos, os dados de entrada, os resultados esperados, os resultados obtidos, um indicativo se o teste passou ou falhou para cada ciclo de teste executado e o que deve ser feito após a execução dos testes (Ex: limpeza de bases). Traz como benefícios permitir reutilizar a inteligência e o esforço dedicado a testes e permitir que profissionais menos experientes e de menor custo possam executar e evidenciar os testes. O roteiro é o guia para o teste executado manualmente, quando automatizamos a execução do roteiro geramos o *script* de teste. O roteiro de teste possui as informações:

- Identificação única do roteiro.
- Responsável pela elaboração (especificação) do roteiro.
- Descrição do hardware e software requeridos para a execução dos testes (informação não obrigatória)
- Definição da fase em que os testes serão executados (Unidade, Integração, Sistema, Homologação ou Implantação).
- Informações sobre as pré-condições (*setup*) para a execução dos testes, por exemplo, a preparação das bases de dados.
- Informações sobre a execução dos testes: quantidade de ciclos de testes executados, total de falhas localizadas, esforço total em horas e para cada ciclo de teste: testador responsável pela execução, data de início e conclusão, esforço de tempo, falhas localizadas e número de condições de testes executadas.
- Informações sobre a finalização após a execução dos testes (*clean-up*), por exemplo, com a limpeza da base de dados.

Cada caso de teste pertencente ao roteiro deve possuir:

- *Setup* específico para o caso de teste, ao contrário do setup do roteiro que é comum a vários casos de teste (informação não obrigatória).

- Informações sobre como executar os testes: descrição do teste, condições e entradas para o teste, passo a passo e resultados esperados.
- Resultado do teste para cada ciclo de teste executado, gerado pelo módulo de execução dos testes.
- *Clean-up* específico do caso de teste (informação não obrigatória).

Quando se trata de testes caixa preta, os casos de testes são geralmente detalhamentos dos requisitos de testes, porém nem sempre é possível essa associação porque o uso de requisitos de testes é indicado pela MTS, porém não é obrigatório. Quando não existe o requisito de teste o caso de teste está relacionado diretamente ao requisito. Quando se trata de testes caixa branca os casos de testes não estão relacionados a requisitos.

Um roteiro descreve um ou mais casos de testes e, por sua vez, os roteiros podem ser agrupados em suítes de testes. Utiliza-se o roteiro como um meio de agrupar casos de testes que possuem objetivos em comum, por exemplo, podemos ter um roteiro contendo os casos de testes responsáveis pelos testes de homologação de uma determinada funcionalidade. De maneira análoga utilizam-se suítes de testes; por exemplo, podemos ter uma suíte contendo todos os roteiros de testes de homologação de um sistema.

3.4.4 Registro de Falhas (RF)

Um dos fatores importantes para um bom trabalho de testes e qualidade são o registro e acompanhamento das falhas encontradas durante o processo de desenvolvimento. Estão incluídas neste processo tanto as falhas encontradas pelos testes como as falhas encontradas pelas revisões dos produtos intermediários.

Para isso é proposto um sistema formal de coleta, rastreamento e acompanhamento de falhas detectadas durante o projeto através do documento Registro de Falhas (RF). O registro de falhas pode ser preenchido em qualquer fase do processo de desenvolvimento e testes de sistemas.

Ciclo de Vida das Falhas

O *template* RF trata os status possíveis que uma falha pode ter: Aberta, Em tratamento, Corrigida, Em reteste, Fechada, Adiada e Cancelada. As transições entre estes status devem obedecer a uma seqüência lógica e ao controle de permissão de quem pode tratar cada um deles.

O registro de falhas deve conter as seguintes informações:

- Identificador único
- Identificação do Projeto / Sistema / Módulo em que a falha ocorreu
- Tipo de registro: problemas encontrados na execução do programa ou problemas encontrados na revisão de documentos
- Tipo de documento revisado que gerou a falha: pode ser qualquer um dos documentos pertencentes à metodologia de desenvolvimento
- Versão do documento ou programa em que a falha foi encontrada
- Requisito associado, para falha encontrada na revisão de requisitos
- Caso de Teste associado, para falha encontrada nos testes
- Identificação de quem registrou a falha
- Descrição sucinta e detalhada da falha
- Fase da metodologia em que a falha foi identificada
- Fase da metodologia em que a falha foi originada
- Identificação do responsável pela solução
- Severidade ou impacto da falha: muito baixo, baixo, alto e muito alto
- Prioridade para a solução: dispensável, desejável, importante e imprescindível
- Solução proposta para a correção
- Tempo gasto para a correção
- Históricos dos acompanhamentos: cada tratamento que a falha receber deve ser registrado o status e a data

3.5 Resumo do Capítulo

Abordamos a metodologia de testes existente na empresa e que será alvo do processo de automação, definimos as suas fases e relacionamos algumas atividades. Conceituamos os tipos de requisitos, ressaltamos que serão base para os testes e que, portanto, devem ser submetidos a um processo de revisão. Por fim, apresentamos os documentos de testes: o plano de testes, o documento da gestão e rastreabilidade dos testes, o roteiro de teste e o registro de falhas.

Capítulo 4

Requisitos de Automação de Testes

Este capítulo irá apresentar os requisitos que uma abordagem em automação de testes deve atender. O primeiro e principal requisito é uma arquitetura de automação de testes. Inicialmente, abordaremos qual seria esta arquitetura e posteriormente iremos apresentar os requisitos de automação organizados de acordo com a arquitetura proposta. Estes requisitos são oriundos da metodologia de testes MTS, que foi apresentada no capítulo 3, e de diversos autores da literatura. Focaremos nas questões técnicas da automação de testes, deixando de lado os aspectos organizacionais, de custos e expectativas que foram discutidos na seção 2.3. Também neste capítulo apresentaremos um workbench[3] de testes contendo modelos que detalham as características dos artefatos de testes e das ferramentas responsáveis por eles.

4.1 Proposta de Arquitetura de Automação de Testes

Para que a automação de testes contemple as desejadas capacidades de reusabilidade dos artefatos de testes, tais como a facilidade de reuso de scripts ([Few01], [Kan97], [Kan00]) e a rastreablidade entre os artefatos como Requisitos, Casos de Testes, Execução e Falhas, é necessário que se construa uma arquitetura de automação eficiente para este fim.

Uma arquitetura de automação deve contemplar seis grandes funcionalidades objetivando tornar os testes mais ágeis e precisos. Deve haver uma clara separação entre as funcionalidades com o uso de componentes independentes e que ocorra a integração entre as ferramentas ([Eic96]). A seguir descrevemos estas funções:

- Planejamento dos Testes: Trata de informações relacionadas ao documento Plano de Testes apresentado na seção 3.4.1.

[3] Um workbench é um conjunto de ferramentas, reunidas com o objetivo de fornecer suporte a uma fase particular do processo de software (no nosso caso a fase de testes) ([Som03]).

- Desenvolvimento do Teste: Consiste da especificação e implementação da configuração do teste. Alguns artefatos desenvolvidos incluem casos de testes, roteiros de teste (apresentado na seção 3.4.3), suítes de testes, critérios de teste, geração de dados para teste e oráculos.
- Execução do Teste: inclui a execução do código e gravação de artefatos, como saídas do teste, registro (*trace*) da execução e status do teste.
- Análise de Falhas: Trata-se da verificação e documentação do comportamento da execução dos testes. Ou seja, é responsável por comparar o resultado encontrado contra o resultado esperado para determinar se o teste passou ou falhou. Em caso de falha esta deve ser registrada e acompanhada no documento Registro de Falhas apresentado na seção 3.4.4.
- Medições dos Testes: As medições dos testes não devem estar espalhadas nas diversas funções, mas concentradas neste módulo. Inclui, entre outros, a medição da cobertura dos testes e medições relacionadas às falhas nos testes.
- Gerenciamento da Configuração dos Artefatos de Teste: Inclui suporte para a persistência dos artefatos de teste, dos relacionamentos entre os artefatos e a preservação do estado de execução. Os artefatos devem ser armazenados em um repositório. Um repositório passivo atende a necessidade básica de armazenamento, porém é desejado um repositório ativo para suportar as relações entre os artefatos.

A figura 4.1 ilustra a arquitetura de automação com os seis módulos responsáveis pelas grandes funções e a passagem de dados entre eles. Ela é baseada na proposta de Peter Vogel acrescida do módulo de Planejamento dos Testes ([Vog03]).

Neste gráfico procuramos demonstrar a independência entre as funções, pois a passagem de dados entre elas é feita através do repositório de dados pertencente à funcionalidade do Gerenciamento da Configuração dos Artefatos de Testes. A exceção é entre os módulos de Execução dos Testes e de Análise de Falhas, pois a dependência entre eles é muito forte e praticamente o módulo de Análise de Falhas funciona como um módulo auxiliar ao de Execução dos Testes.

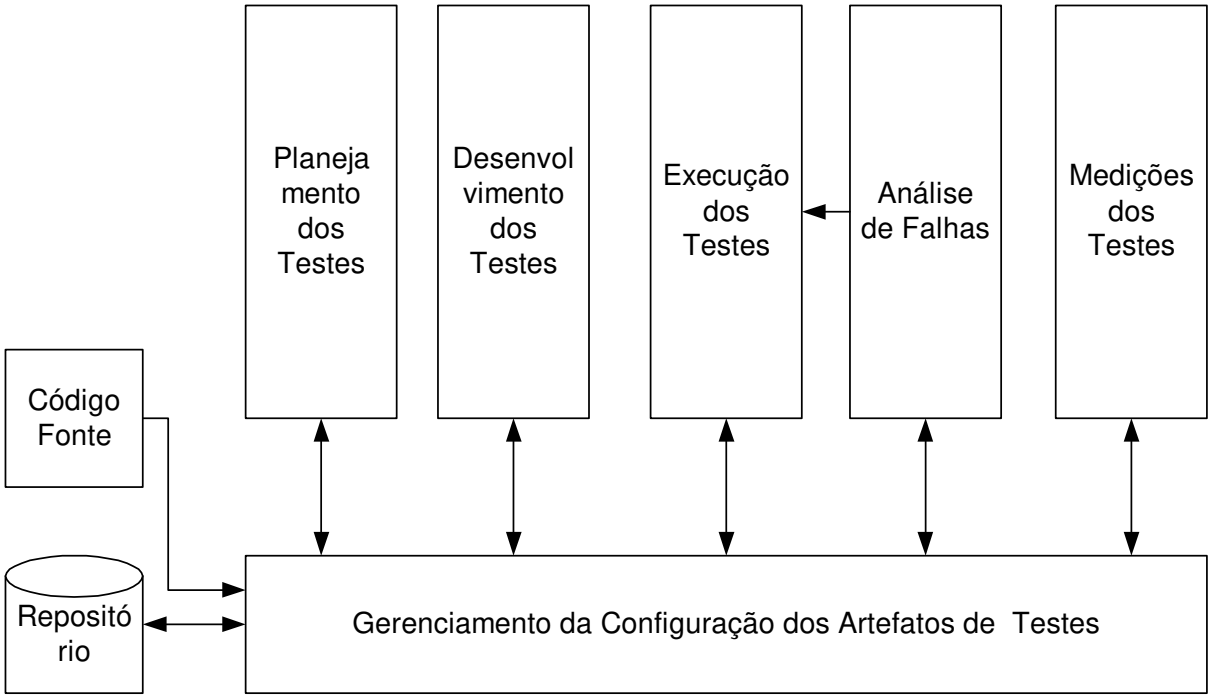


Figura 4.1: Arquitetura de Automação dos Testes ([Vog03])

4.2 Requisitos da Automação

Passaremos a apresentar os requisitos de automação do teste de software oriundos tanto da literatura como da metodologia de testes (MTS) apresentada no capítulo 3. Iremos identificar cada requisito de automação com um Identificador de Requisito (IdReq) e serão agrupados pelas seis grandes funções apresentadas. Objetivando uma maior clareza, a partir deste ponto utilizaremos os termos requisito de automação (quando se tratar de requisitos relacionados às necessidades de automação), requisito de teste (conforme definido na seção 3.2) e requisito de software (no lugar do termo requisito definido da seção 3.2).

4.2.1 Planejamento dos Testes

Nesta seção trataremos dos requisitos relacionados à automação do planejamento de testes. Em geral nas ferramentas, a atividade de planejamento é tratada de maneira estanque, isolada, sem relacionamento com as outras atividades. Porém, é necessário que o planejamento

dos testes esteja integrado com as outras funcionalidades e não ser tratado por uma ferramenta isolada ([Eic96]). Por exemplo, é necessário que a ferramenta de planejamento de testes forneça e receba dados da ferramenta responsável pelo planejamento do projeto de desenvolvimento em que os testes estão contidos (Identificador do Requisito - IdReq: 1.01). Como outro exemplo, é necessário que ocorra o retorno das ferramentas que serão utilizadas nas fases seguintes para as ferramentas de planejamento (IdReq: 1.02). Assim, quando estivermos especificando os roteiros de testes poderemos ter uma previsão mais concreta sobre a estimativa de tempo para execução dos mesmos. Esta informação deve retornar à ferramenta de planejamento para produzir estimativas mais atualizadas e realistas.

A ferramenta responsável pelo planejamento dos testes deve possibilitar o cadastramento das informações do documento Plano de Testes conforme descrito no capítulo 3 (IdReq: 1.03). Espera-se que algumas estimativas possam ser calculadas pela ferramenta de planejamento dos testes ([Met04]). A partir da quantidade de requisitos de software informados e através de parâmetros como a utilização de índices de mercado e de indicadores históricos da organização, a ferramenta estimaria: a quantidade de requisitos de testes, a quantidade de horas para teste, a quantidade de falhas previstas de serem encontradas e a quantidade de horas necessárias para corrigir as falhas. Caberia ao responsável pelo planejamento dos testes confirmar as estimativas sugeridas ou informar os números que julgar mais adequado (IdReq: 1.04).

4.2.2 Desenvolvimento do Teste

Aqui trataremos dos requisitos de automação relacionados à construção dos artefatos de testes. Por exemplo, para que o módulo de análise de falhas utilize um oráculo, o módulo que tratamos aqui deve construir este oráculo. Para uma melhor organização dividimos esta seção em três de acordo com o tipo do artefato a ser construído: os que relacionam requisitos de software e testes, os que denominamos de básicos e aqueles que serão utilizados como oráculos.

Requisitos da Automação do Relacionamento entre Requisito de Software e Teste

Na metodologia de testes em que este trabalho foi estruturado os testes funcionais são baseados em requisitos de software, como vimos no capítulo 3. Entre os objetivos da automação dos testes estão o de viabilizar o relacionamento entre testes e requisitos de software e a

implementação do conceito de requisito de teste (IdReq: 2.01). É indicado que o gerenciamento de requisitos de software e o gerenciamento de testes sejam feitos por uma mesma ferramenta ou que tenham uma forte integração ([Fro04]) (IdReq: 2.02). Mudanças nos requisitos de software devem ser percebidas na ferramenta que trata os testes, pois para cada requisito de software alterado teremos testes associados que precisam ser revistos (IdReq: 2.03).

Além da descrição do requisito de software, que é fundamental para a especificação dos testes (IdReq: 2.04), outra informação importante é o risco de falha associado ao requisito de software (IdReq: 2.05). Esta informação auxilia na priorização de que testes merecem mais atenção e na decisão de que falhas devem ser resolvidas primeiro. A informação de risco deve ser composta pela probabilidade de ocorrer alguma falha no requisito de software quando este estiver implementado e o impacto em caso de ocorrer uma falha ([Met04]). Na automação é necessário que a informação do risco do requisito de software esteja disponível na ferramenta de gerenciamento dos testes (IdReq: 2.06).

Usualmente, um requisito de testes irá originar casos de testes. Algumas vezes pode-se preferir não criar casos de testes por se considerar que o requisito de teste é suficiente para garantir a adequada execução do teste ([Met04]). Portanto, é desejável que a ferramenta de automação permita que um requisito de teste seja validado sem a necessidade de casos de testes (IdReq: 2.07).

É imprescindível que se garanta a rastreabilidade tanto entre os requisitos de software e os requisitos de testes (IdReq: 2.08), como entre requisitos de testes e os casos de testes a eles associados (IdReq: 2.09) e entre os casos de testes e requisitos de software (IdReq: 2.10). No modelo de dados dos artefatos de testes que será apresentado na seção 4.3.1 traremos mais detalhes sobre os relacionamentos entre estes artefatos.

É necessário que a ferramenta disponibilize as funcionalidades que são supridas pelo documento da MTS chamado de GRT – Gestão e Rastreabilidade dos Testes, conforme visto no capítulo 3 (IdReq: 2.11).

Requisitos da Automação do Desenvolvimento dos Artefatos Básicos de Testes

Cabe à automação de testes prover mecanismos que viabilizem a criação do que chamamos artefatos básicos do teste, que são os casos de teste (IdReq: 3.01), os roteiros de testes (cujas informações foram listadas no capítulo 3) (IdReq: 3.02), as suítes de teste (IdReq: 3.03), os

critérios de teste (IdReq: 3.04) e a geração de dados de teste. Para isso as ferramentas envolvidas devem disponibilizar editores e facilidades para tornar produtivo este trabalho.

Como dissemos no capítulo 3, a MTS propõe que os casos de testes sejam definidos e agrupados em roteiros de testes, que por sua vez se agrupam em suítes de testes. É obrigatório que a ferramenta exija o agrupamento dos casos de testes em roteiros e possibilite o agrupamento de roteiros em suítes de testes.

Os dados de entrada para o teste podem ser gerados de forma manual ou de forma automatizada. A automação deve contemplar a construção de geradores de dados de teste através do uso de utilitários e ferramentas. Esta geração automatizada pode ocorrer através da seleção de dados de um banco de dados (IdReq: 3.05), ou através da geração de dados aleatórios que satisfaça uma especificação baseada em uma gramática (vocabulário) definida para este fim ([Som03], [Ric94], [Bin00]) (IdReq: 3.06). Alguns estudos apontam para a automação da geração de dados para popular adequadamente um banco de dados a ser utilizado em um teste ([Cha00], [Dav00]) (IdReq: 3.07).

Requisitos da Automação do Desenvolvimento de Oráculos

Os oráculos são artefatos que têm como responsabilidade determinar se o resultado do teste passou ou falhou. Para isso deve ser composto por um gerador de resultados que determina as saídas esperadas para um caso de teste (IdReq: 4.01) e um comparador que compara as saídas esperadas contra as encontradas na execução do teste (IdReq: 4.02) ([Bin00]).

Diversos tipos de oráculos foram definidos por [Bin00]. Selecionamos aqueles mais adequados baseados na metodologia de testes existente (MTS) e em nosso campo de estudo (IdReq: 4.03):

- Julgamento Humano: O testador avalia se o teste passou ou falhou olhando a saída em tela, numa listagem ou em outra interface humana.
- Oráculo para Amostra Resolvida: Desenvolve os resultados esperados manualmente que podem ser incorporados ao caso de teste para serem avaliados automaticamente.
- Oráculo de Simulação: Grava o resultado esperado exato através de uma implementação mais simples do SUT (*System Under Test*, o SUT pode ser um

componente, uma classe ou toda a aplicação) capaz de simular as suas responsabilidades essenciais.

- Oráculo de Sistema Verdadeiro: Rodam-se alguns casos de teste sobre um sistema verdadeiro para gerar os resultados esperados. Por sistema verdadeiro entende-se aquele que se tem uma alta confiança sendo capaz de computar algumas das funções do SUT.
- Oráculo de Teste Paralelo: Similar ao oráculo de sistemas verdadeiro, difere que aqui todos os casos de testes são executados.
- Oráculo de Teste de Regressão: Neste caso os resultados encontrados para uma dada implementação são utilizados como resultados esperados quando esta implementação evolui.
- Smoke Test: Por ser geralmente o primeiro teste de uma aplicação que busca verificar se os demais testes podem ser realizados, usa as operações básicas para checar o ambiente. Não precisa verificar os resultados, apenas se terminou normalmente.
- Oráculo de código embutido: Assertivas e outros mecanismos são implementados no código do SUT para verificar algumas correções. Por exemplo, para uma aplicação bancária poderíamos implementar uma assertiva que verifica se o valor a ser resgatado não é maior que o dobro do valor aplicado.
- Oráculo de Reversão: Algumas funções, a minoria, podem ser revertidas. Quando isso é possível, devemos considerar a hipótese de construir oráculos que explorem essa lógica inversa.

Como ocorre em outras atividades de teste, é necessário que se faça um coerente uso do que se vai automatizar e o que se vai fazer manualmente, neste caso deve-se permitir que um mesmo caso de teste possua parte da avaliação manual e parte automática ([Bin00]) (IdReq: 4.04).

A automação dos comparadores pode ser feita através da programação tradicional ou, preferencialmente, através do uso de utilitários e ferramentas de manipulação de dados ([Bin00]) (IdReq: 4.05). É necessário que se permita o uso de soluções alternativas para a análise das saídas, por exemplo: que um processo da baixa plataforma (chamaremos de baixa plataforma

todas as plataformas que não são mainframe: cliente/servidor, Web, etc) possa chamar processos do mainframe que comparem dois arquivos ([Few01]) (IdReq: 4.06).

Deve-se garantir que a saída a ser comparada possa ter diversos formatos (IdReq: 4.07). Se a saída observável não for persistente, por exemplo, não esteja armazenada em um banco de dados, mas sim numa tela, devem ser feitos mecanismos para recuperá-la e salvá-la (IdReq: 4.08). Deve-se permitir que a comparação seja entre pequenas seções da tela e não a tela toda ([Kan98]) (IdReq: 4.09). Não podemos esquecer que também são necessários mecanismos que permitam uma avaliação do ambiente ([Bin00]) (IdReq: 4.10).

4.2.3 Execução dos Testes

Um dos principais requisitos da automação da execução dos testes é que esta deve ser feita através de Drivers de Testes ([Ric94] e [Vog03]) (IdReq: 5.01). Entendemos por Driver de Testes um componente que é responsável por coordenar os processos de execução dos testes. Cabe ao driver a responsabilidade de chamar o módulo responsável pela preparação das bases, recuperar os casos de testes de uma suíte e enviá-los um a um ao SUT. Após a execução do SUT, o driver deve chamar o módulo de análise de falhas para avaliar o resultado do teste como passou/falhou e no final chamar o módulo de limpeza.

Uma característica desejada é que o Driver de Teste copie todos os arquivos que serão utilizados pelo teste para um diretório de execução. Se ocorrer uma falha os arquivos usados e criados pelo teste devem ser guardados para facilitar o debug ([Vog93]) (IdReq: 5.02).

Arquiteturas para criação de Scripts de Testes

Apresentamos a seguir as arquiteturas de criação de scripts de testes que têm como fonte diversos autores como [Pet01b], [Kan00], [Nag00] e [Bin00]. É possível e, em muitos casos, esperada a utilização concomitante de mais de uma destas arquiteturas.

Talvez a arquitetura mais simples seja a utilização pura da ferramenta de *Capture & Replay* (ou *Record & Playback*). Nesta arquitetura são utilizados somente os recursos oferecidos pela ferramenta que basicamente é o de "filmar" uma ação humana interagindo com um aplicativo e reproduzi-la quando solicitado (IdReq: 5.03). Estas ferramentas possuem uma

linguagem em que são codificados os scripts que são programas que reproduzem ações humanas de navegação e entrada de dados.

Outra possibilidade é o desenvolvimento de scripts sem o auxílio de ferramenta. Neste caso os codificadores conhecem a linguagem da ferramenta de *Capture & Replay*, codificam um script e utilizam a ferramenta para reproduzir (IdReq: 5.04).

Uma maneira também utilizada é o desenvolvimento de scripts a partir de scripts gerados pela ferramenta. A ferramenta de *Capture & Replay* "filma" e gera os scripts, os codificadores alteram estes scripts e utilizam a ferramenta para a reprodução de uma maneira diferente do que foi "filmado" (IdReq: 5.05).

Um outro tipo de arquitetura é a biblioteca de scripts: os genéricos e os modificáveis. Na biblioteca de scripts genéricos os scripts comuns a várias aplicações, como o que trata do botão de sair, são disponibilizados em bibliotecas. Os responsáveis pela automação dos testes podem utilizar estes scripts em suas aplicações específicas (IdReq: 5.06). Nesta mesma linha temos as bibliotecas de scripts modificáveis: esqueletos de scripts estão disponíveis para serem recuperados e modificados pelas necessidades específicas das aplicações (IdReq: 5.07).

Uma arquitetura mais evoluída é a Arquitetura *Data-Driven* (IdReq: 5.08), na qual dados de entrada para o aplicativo são isolados do script, retirando-os do código e armazenando-os numa estrutura de dados externa. Os scripts são alterados para acessar estas estruturas e ler os dados armazenados ou recebê-los de quem invocou o script (o driver de teste).

Para muitos autores a arquitetura mais avançada é a dirigida a Palavras-Chave (IdReq: 5.09). Nesta arquitetura os scripts são gerados por um *framework* a partir da definição passo a passo de um caso de teste utilizando-se de um vocabulário (conjunto de palavras chave que definem os passos de um teste). Esta abordagem exige um investimento inicial maior devido à necessidade de se construir o *framework*, porém é recompensado por ser mais fácil de utilizar, manter e perpetuar. O vocabulário utilizado deve ser independente da ferramenta de *Capture & Replay* e próxima da linguagem dos testadores, sendo capaz de guiar também os testes manuais descrevendo as ações a serem tomadas. O vocabulário também deve ser independente do *framework*. Ou seja, quando definimos as instruções necessárias para testar uma aplicação em particular, poderíamos usar exatamente as mesmas instruções em qualquer implementação de *framework* capaz de testar esta aplicação (IdReq: 5.10).

Requisitos da Automação da Execução que independem de como são criados os Scripts

É necessário que se disponibilize ferramentas capazes de atuar como simuladores de dois tipos: o simulador alvo (IdReq: 5.11) que simula a máquina em que o programa deve ser executado e o simulador de interface com o usuário (IdReq: 5.12) que simula interações com múltiplos usuários ao mesmo tempo, útil para testes de carga e stress ([Som03], [Bin00]).

É esperado que seja possível escolher entre a execução de um caso de teste ou de uma suíte de testes. Quando executamos uma suíte deve ser provida a seleção de um subconjunto de casos de testes para execução, pois em algumas situações pode não ser interessante executar todos os casos de testes de uma suíte (IdReq: 5.13). Deve ser disponibilizada a funcionalidade de não executar os testes que são mais propensos à falhas, visando não desperdiçar recursos do ambiente com estes testes ([Vog93]) (IdReq: 5.14).

Quando uma suíte de testes é selecionada para execução, deve ser permitida a execução de casos de testes em paralelo ([Ric94]) (IdReq: 5.15). A execução em paralelo, independente se manual ou automatizada, pode fazer com que alguns testes provoquem efeitos indesejáveis em outros. Isso ocorre muitas vezes quando estes testes compartilham de recursos em comum, como por exemplo, bases de dados. Diversas iniciativas[4] buscam criar mecanismos que tragam soluções para esta questão.

Ao se executar uma suíte de testes, alguns casos de testes podem falhar e em um novo ciclo de execução pode ser conveniente executar o subconjunto dos casos de testes que falharam. Para isso devem ser permitidos o *restart*, a recuperação e a seleção de alguns casos de teste que falharam ([Bin00]) (IdReq: 5.16).

É necessário que a execução dos testes tenha o suporte e integração com uma ferramenta de depuração ([Bin00]) (IdReq: 5.17). Além desta integração, veremos na seção 4.3.6 que são necessárias também as integrações entre o módulo de Execução dos Testes com o módulo Gerador de Dados (IdReq: 5.18) e com a ferramenta de *Capture/Replay* (IdReq: 5.19).

Para facilitar a depuração e também para determinar o progresso dos testes é necessário que se gere um registro (*trace*) de execução ([Kan97], [Kan98]) (IdReq: 5.20). Para isso pode-se utilizar uma ferramenta que adicione código ao programa para gerar este registro da execução

[4] Permitir que cada sessão de teste possua sua própria versão do dado parece ser a solução mais apropriada. Para isso estuda-se o uso de versionamento de schema, de transaçõees longas ou da replicação de dados.

([Som03]). O registro da execução deve ser armazenado num repositório de onde serão recuperados pelo módulo de medições ([Eic96]) (IdReq: 5.21).

É importante permitir que partes do teste sejam executadas manualmente e partes de maneira automatizada (IdReq: 5.22). Cabe ao módulo de execução registrar informações para o roteiro de testes, tais como o testador responsável pela execução e a data de início e conclusão da execução dos testes (IdReq: 5.23). Deve também registrar para o caso de teste o correto resultado da sua execução: passou, falhou, não executou e não resolvido ([Pet01b]) (IdReq: 5.24). Para tanto o módulo de execução chama o módulo de análise de falhas.

4.2.4 Análise de Falhas

O módulo de análise de falhas é chamado pelo módulo de execução de testes para determinar o resultado do teste após a sua execução. Para isso pode utilizar os oráculos para gerar os resultados esperados (IdReq: 6.01) e atuar como comparadores para determinar se o teste passou ou falhou (IdReq: 6.02).

Caso o resultado encontrado não seja o mesmo que o esperado, houve a ocorrência de uma falha que deve ser registrada (com as informações listadas no capítulo 3) e devidamente acompanhada (IdReq: 6.03). Deve-se registrar o histórico (IdReq: 6.04), permitir o *workflow* com controle de quem tem permissão para o tratamento da falha em cada um dos seus status (IdReq: 6.05) e permitir consultas através das informações do registro de falhas (IdReq: 6.06), por exemplo, uma consulta às falhas que contenham na sua descrição um determinado termo ([Met04]).

O gerenciamento das falhas, também conhecido como *bug-tracking*, deve ter a capacidade de servir como um ambiente de aprendizado colaborativo e interativo (IdReq: 6.07). As falhas detectadas e resolvidas devem servir de aprendizado para a corporação. Para tanto é necessário que se disponibilize um FAQ (questões mais freqüentes) e que se utilize um repositório corporativo ([Shu96]). Com este repositório espera-se que se encontrem mais facilmente soluções para as falhas através de consultas a uma base de falhas conhecidas ([Vog93]).

A MTS determina que sejam feitas revisões dos produtos intermediários de um projeto e caso sejam encontrados problemas nas revisões, estes devem ser registrados. Espera-se que a

mesma ferramenta que registra e gerencia o ciclo de vida das falhas encontradas nos testes também o faça para as falhas encontradas nas revisões (IdReq: 6.08).

Um dos processos que merece atenção quanto às possibilidades de automação é o de revisão de código fonte. É desejado que se tenha uma ferramenta para esse fim, conhecida como analisador estático de código, que deve receber como entrada o código fonte e examiná-lo quanto a algumas características. O analisador estático não necessita que o programa seja executado, ele percorre o texto do programa e tenta encontrar possíveis defeitos e anomalias complementando os recursos de detecção de erros fornecidos pelo compilador de linguagem ([Som03]). No caso da empresa deste campo de estudo é prioritário um analisador de código para o mainframe nas linguagens estruturadas (IdReq: 6.09) e no SQL-DB2 (IdReq: 6.10)

Muitos dos indicadores relacionados a testes são resultantes do registro e gerenciamento das falhas. Apesar de estas informações serem geradas nesta funcionalidade de análise de falhas, a medição das falhas deve ser feita no módulo de medições que veremos a seguir. Além desta necessidade de integração do módulo de Análise de Falhas com o módulo de Medições (IdReq: 6.11), é necessária também a integração do módulo de Análise de Falhas com o módulo de Execução dos Testes (IdReq: 6.12), estas integrações serão vistas em mais detalhes nas seções 4.3.5 e 4.3.6.

4.2.5 Medição dos Testes

A automação de testes deve viabilizar a coleta de indicadores durante o processo de teste que avaliem e permitam melhorar a qualidade do software, do processo de desenvolvimento e do próprio processo de teste. A medição não pode ser acoplada às outras funções que geram as informações, mas sim no módulo específico de medições, para isso deve-se utilizar um repositório que permita a desejada independência ([Eic96]) (IdReq: 7.01).

Passamos agora a relacionar os indicadores desejados no processo de automação. Abaixo seguem os indicadores para falhas ([Met04]). Estes indicadores valem, em sua maioria, tanto para as falhas encontradas nos testes como também para as registradas nas revisões (IdReq: 7.02).

- Falhas abertas X falhas fechadas por impacto que a falha provocou
- Falhas abertas no dia X falhas abertas acumuladas

- Falhas fechadas no dia X falhas fechadas acumuladas
- Falhas por status em que se encontra
- Falhas por fase que foi originada e por fase em que foi encontrada
- Análise do desempenho dos testes: através da relação entre as falhas encontradas pela área de TI e as encontradas pelos usuários e clientes
- Tempo útil médio para conserto das falhas e tempo corrido médio decorrido entre a data de abertura da falha e a de fechamento
- Falhas distribuídas por programas ou módulos do sistema
- Falhas distribuídas por responsáveis em encontrá-las e em resolvê-las

Os indicadores do planejamento de testes estão relacionados ao interesse em se comparar o que foi planejado contra o que efetivamente ocorreu (IdReq: 7.03). Por exemplo, quando executamos os testes temos o tempo efetivamente gasto na sua execução e podemos compará-lo com o tempo planejado. Outros exemplos de comparação entre o previsto no planejamento e o ocorrido em fases posteriores são: o tempo para especificação dos testes, a quantidade de requisitos de testes e a quantidade de falhas encontradas ([Met04]).

Para a MTS é importante que se realimente o processo de planejamento com indicadores mais precisos sobre os de testes. Assim, quando um novo projeto for utilizar o processo de planejamento pode fazer uso de indicadores históricos sobre projetos que tenham similaridades com o atual (IdReq: 7.04).

Com relação aos indicadores da execução dos testes podemos citar a verificação de se o critério de testes foi adequadamente satisfeito ([Met04]) (IdReq: 7.05). Para isso é necessário, como abordamos anteriormente, que o módulo de execução instrumentalize o código fonte para produzir registros (*traces*) que permitam esta análise. Após a execução de uma suíte de testes ou de um caso de teste pode-se verificar, através do modulo de medições, onde e quanto o critério de testes foi adequadamente satisfeito. Cada caso de teste deve ser associado com os elementos do critério que satisfaz ([Ric94]) (IdReq: 7.06). Outras medições sobre a execução dos testes são: o total de condições de testes cobertas, a quantidade de ciclos de testes executados, o total de falhas localizadas e o esforço total em horas ([Met04]) (IdReq: 7.07).

4.2.6 Gerência de Configuração dos Artefatos de Testes

A gerência de configuração dos artefatos de testes é fundamental para uma automação completa e para garantir a reprodutibilidade do processo de teste. Durante os testes diversos e volumosos artefatos são produzidos. O ciclo de vida destes artefatos e os relacionamentos entre eles devem ser suportados pela gerência de configuração. Sem uma gerência de configuração eficientemente automatizada não é possível coordenar efetivamente os processos de testes ([Eic96]). Entre os requisitos de automação que a gerência de configuração deve endereçar estão a reusabilidade, a rastreabilidade e o versionamento dos artefatos.

A reusabilidade se trata da capacidade de armazenar, recuperar e reutilizar os artefatos de testes e seus relacionamentos, tais como: plano de teste, scripts, suítes de testes, casos de testes, critérios de testes, oráculos e resultados (IdReq: 8.01). A reusabilidade é fundamental para uma vantajosa automação do processo de teste. Por diversas oportunidades afirmamos a importância de uma arquitetura de automação que garanta componentes independentes e com funcionalidades claras, somente com isso e com uma eficiente gestão dos artefatos de teste pode-se alcançar a reusabilidade destes artefatos ([Ric94]). Entre os artefatos de teste que se destacam na necessidade de se garantir o reuso estão os scripts de teste. Diversos autores ressaltam a importância de se garantir a facilidade para reuso dos scripts ([Few01], [Kan97], [Kan00]).

Para se alcançar a reusabilidade é necessário o uso de um repositório ativo que permita o armazenamento e a recuperação dos artefatos de testes e de seus relacionamentos. Para a empresa do campo de estudo deste trabalho é importante um repositório corporativo que permita armazenar os artefatos de testes tanto do mainframe como os da baixa plataforma (IdReq: 8.02).

Também deve ser provido o relacionamento entre os artefatos de testes e os de análise ([Ric94]) (IdReq: 8.03). Ou seja, uma suíte de teste de unidade deve estar relacionada ao componente a que se propõe testar. Posteriormente, na seção 4.3.1 iremos apresentar um modelo com os relacionamentos entre os artefatos de testes.

É necessário que a gerência de configuração promova o versionamento dos artefatos de testes ([Few01]). Este versionamento deve também considerar os relacionamentos entre os artefatos. É necessário que se conheça e que seja facilmente recuperada a versão da suíte de teste que foi utilizada para qualificar determinada versão de um produto ([Vog93]). Ao se recuperar uma suíte de testes também devem ser recuperados outros artefatos como os dados de teste

utilizados ou a cobertura alcançada com a sua execução. A responsabilidade por conhecer, por exemplo, qual a versão da suíte de testes e de outros artefatos relacionados que foram utilizados para testar determinada versão de um componente não deve ser dos profissionais envolvidos ou das ferramentas isoladas, mas sim de um módulo independente que é este da gerência de configuração (IdReq: 8.04).

4.3 Workbench de Testes

Nesta seção apresentaremos um workbench de testes contendo os artefatos de testes e as ferramentas responsáveis pelos seus tratamentos. A vantagem de se agrupar ferramentas em um workbench é que elas podem trabalhar juntas para prover maior suporte que uma ferramenta isolada, os serviços comuns podem ser solicitados por outras ferramentas ([Som03]).

Este workbench é derivado dos requisitos de automação já apresentados neste capítulo trazendo mais detalhes através de modelos que demonstram: os artefatos de testes e seus relacionamentos entre si; as ferramentas responsáveis pelos artefatos e pelos relacionamentos entre os artefatos; e as integrações esperadas entre as ferramentas.

4.3.1 Modelo de Dados dos Artefatos de Testes

Na figura 4.2 apresentamos o MER - Modelo de Entidade e Relacionamento dos artefatos que são necessários aos testes. Incluímos também os artefatos que têm uma forte ligação com testes, como requisitos de software (chamado no modelo de Requisito), por exemplo. Através destes relacionamentos representamos muitos conceitos que estão por trás do assunto testes e que devem ser considerados na implementação da automação.

Para exemplificar como é a leitura deste modelo vamos utilizar o relacionamento entre Caso de Teste e Requisito. Encontramos do lado de Requisito o (0,1) que significa que um caso de teste pode ter nenhum requisito de software associado (isto é correto conceitualmente, pois pode se tratar de um caso de teste caixa branca) ou no máximo um. Na ponta do Caso de Teste encontramos o (0, N) que significa que um requisito de software pode não ter caso de teste associado e pode chegar a ter vários. Outro exemplo é o relacionamento entre Requisito e

Requisito de Teste, no lado de Requisito encontramos o (1,1) que significa que um requisito de teste tem de estar associado à pelo menos um requisito de software e no máximo a um, ou seja, todo requisito de teste deve estar associado a um e somente um requisito de software. Na ponta do Requisito de Teste temos o (0,N) que significa que um requisito de software pode não ter requisito de teste associado e pode chegar a ter vários. Também está representado que a um Requisito de Teste seja associado nenhum ou vários casos de teste e a um caso de teste nenhum ou um Requisito de Teste.

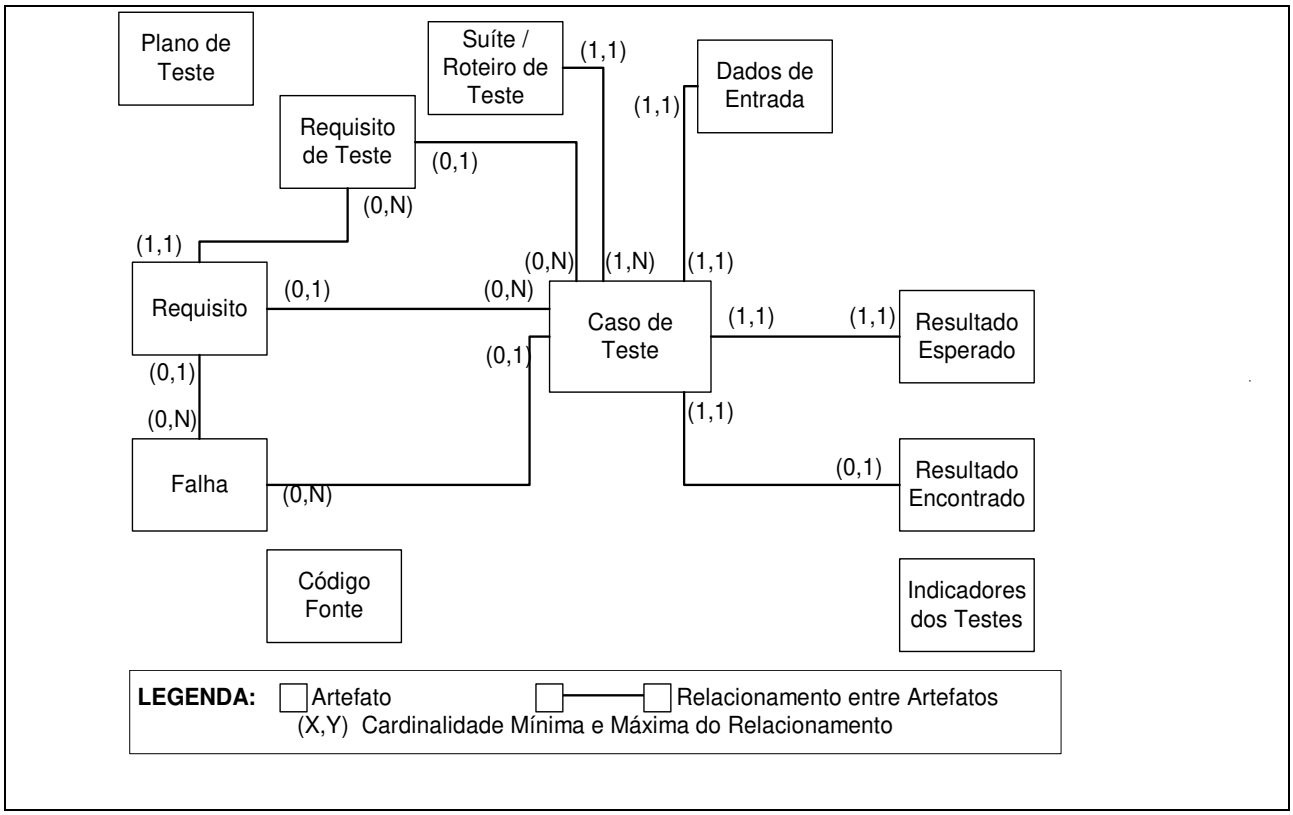


Figura 4.2: Modelo de Dados dos Artefatos de Testes

O plano de teste, apesar de tratar informações que serão úteis a diversas outras funcionalidades, não se relaciona com os demais artefatos. Quando dizemos que não se relaciona é no conceito de relacionamento utilizado na modelagem de dados. O fato do planejamento de teste ter de interagir com as demais funcionalidades deve ser resolvido através de processos e não na modelagem de dados. Raciocínio similar justifica porque a entidade Indicadores de Testes está isolada das demais entidades.

Relacionamos tanto os requisitos de software (representado como Requisito) como os requisitos de testes ao caso de teste. Poderíamos ter relacionado apenas o requisito de teste ao caso de teste. Se assim tivéssemos feito seria obrigatório para todo requisito de software ter pelo menos um requisito de teste associado, porém nem sempre isso será exigido dos desenvolvedores.

Resolvemos representar em uma mesma entidade tanto as falhas encontradas nos testes como as oriundas de revisões. Fizemos isso para reforçar que devem ter um tratamento similar e, se possível, por uma mesma ferramenta. Quando se tratar de uma falha encontrada nas revisões não vai ocorrer o relacionamento com o caso de teste e quando se tratar de uma falha encontrada nos testes não ocorrerá o relacionamento com requisito de software. Para simplificar também representamos em uma mesma entidade a suíte e o roteiro de teste.

4.3.2 Modelo das Ferramentas Responsáveis pelos Artefatos

Apresentamos na figura 4.3 um modelo que é uma evolução do modelo apresentado na seção anterior passando a considerar a automação de testes baseada nos requisitos de automação já discutidos na seção 4.2. Surgem agora as ferramentas responsáveis pela automação dos testes e alguns artefatos que apoiarão estas ferramentas, como é o caso do script de teste. Estas ferramentas assumem as grandes funções já relacionadas neste capítulo, por exemplo, para atender a funcionalidade de Análise de Falhas temos as ferramentas: Gerador de Resultados, Comparador, Gerenciador de Falhas e o Analisador Estático.

É importante ressaltar que este modelo é o que acreditamos ser o mais adequado para o nosso campo de estudo. Não se trata de uma única possível visão, pois mais de uma forma de representar poderia ser válida. Essas considerações também são válidas para os demais modelos apresentados neste capítulo.

A ferramenta central é o Gerenciador de Testes a quem cabe a responsabilidade pela criação e gerenciamento da execução do caso de teste, do roteiro de teste e da suíte de teste.

Além da responsabilidade de tratar os requisitos de software, colocamos para o Gerenciador de Requisitos a responsabilidade de tratar o requisito de teste e não para o Gerenciador de Testes. Esta opção foi porque acreditamos que deve ser ainda nas fases iniciais do projeto, quando da definição dos requisitos de software, que se deve pensar em como serão os testes que irão validar se aquele requisito de software foi implementado corretamente.

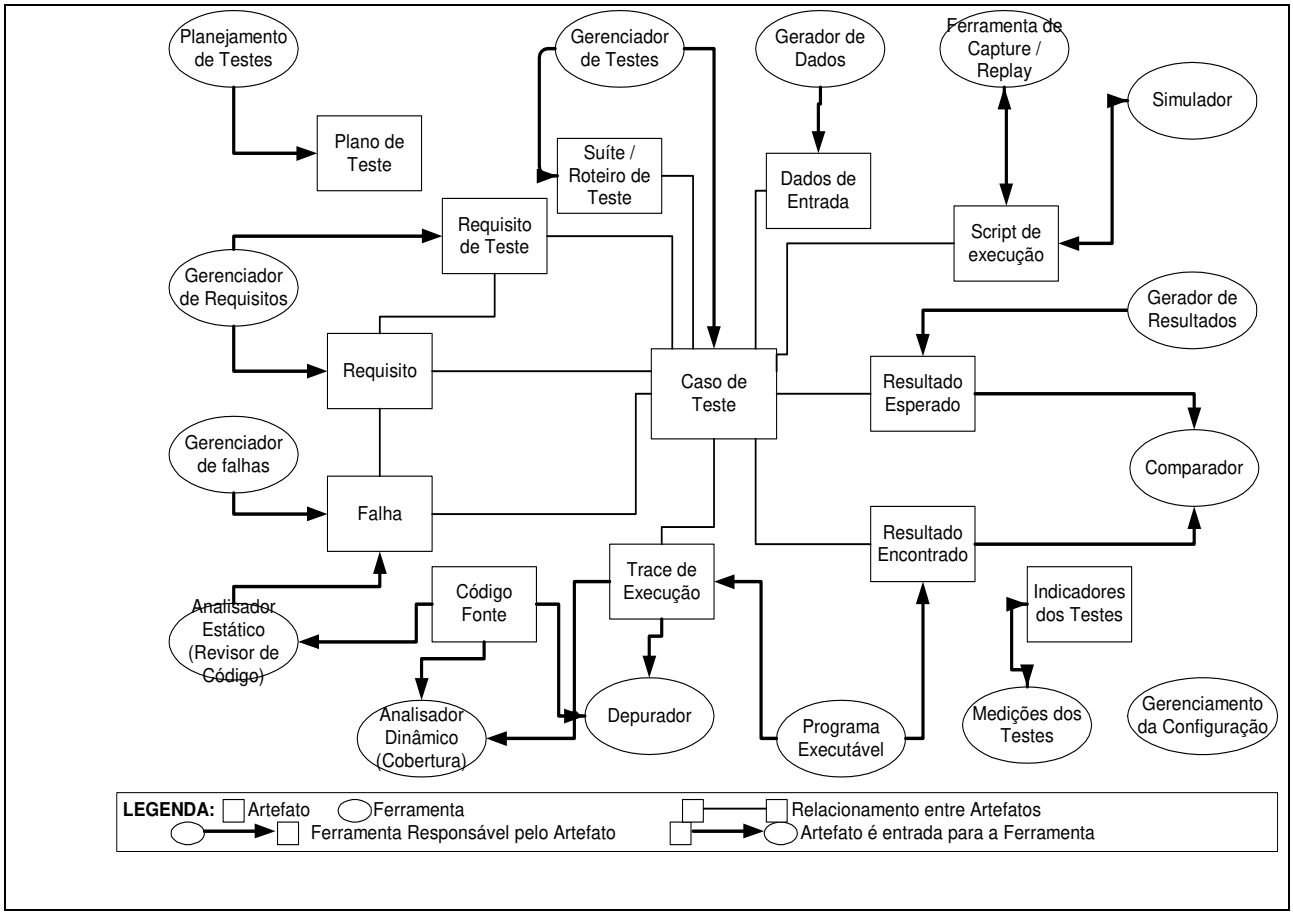


Figura 4.3: Ferramentas responsáveis pelos artefatos

Caberá ao Gerenciador de Falhas tratar o ciclo de vida tanto das falhas encontradas nos testes como das encontradas nas revisões. No caso de revisões, as falhas tanto podem ser cadastradas pelos revisores através da ferramenta de Gerenciamento de Falhas, como também poderão ser criadas diretamente pela ferramenta de Revisão de Código (Analisador Estático) a partir da análise do código fonte que recebe como entrada.

Como já foi dito, o Planejamento, as Medições e a Gestão dos Artefatos de Testes não devem ser tratados pelas diversas ferramentas que compõem o workbench de testes, mas sim por ferramentas específicas, que são as ferramentas de Planejamento dos Testes, de Medições dos Testes e de Gerenciamento da Configuração.

Em relação ao modelo apresentado na seção anterior, dois novos artefatos surgiram: o script de execução e o registro (*trace*) de execução gerado pelo programa executável instrumentado. O script de execução serve como entrada para as ferramentas de *Capture &*

Replay e para a de Simulação de múltiplos usuários. Ele foi relacionado ao caso de teste, porém poderia estar relacionado à Suíte / Roteiro de Teste, pois é comum que um mesmo Script de Execução seja capaz de conduzir a execução dos testes para todos os casos de testes de uma suíte / roteiro.

As funcionalidades do Oráculo serão atendidas pelas ferramentas Gerador de Resultados e Comparador. O resultado esperado gerado pela ferramenta Gerador de Resultados e o resultado encontrado gerado pelo programa executável são entradas para o Comparador determinar se o teste passou ou falhou.

O código fonte e o registro (trace) de execução gerado pelo programa instrumentado executável serão entradas para o Analisador Dinâmico determinar qual foi à cobertura de código alcançada na execução dos testes e entradas para o Depurador auxiliar na depuração quando ocorrem falhas durante os testes.

4.3.3 Análises dos relacionamentos entre os Artefatos

Nesta seção a partir da análise dos relacionamentos entre os artefatos definiremos que ferramentas são responsáveis por eles. Para exemplificar estas análises utilizaremos gráficos que são "recortes" do modelo apresentado na seção anterior.

Relacionamentos entre Caso de Teste, Requisito de Software e Requisito de Teste

Vamos analisar os relacionamentos entre os artefatos Requisito de Software (representado simplesmente como Requisito), Requisito de Teste e Caso de Teste, representados na figura 4.4 em conjunto com as ferramentas responsáveis por eles. Como a responsabilidade do tratamento do Requisito e do Requisito de Teste é do Gerenciador de Requisitos, caberá naturalmente a ele a responsabilidade pelo tratamento do relacionamento entre estes artefatos. Os relacionamentos entre Caso de Teste e Requisito e entre Caso de Teste e Requisito de Teste devem ser de responsabilidade do Gerenciador de Teste, já que o momento de fazer esta associação é na definição do Caso de Teste e a responsabilidade pela definição do Caso de Teste é do Gerenciador de Teste.

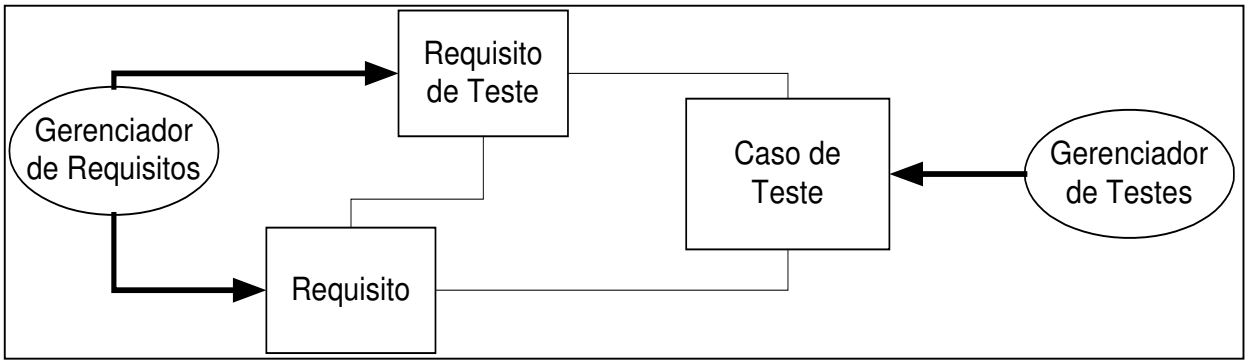


Figura 4.4: Subconjunto Requisito, Requisito de Teste e Caso de Teste

Representamos a responsabilidade que as ferramentas possuem sobre o relacionamento entre os artefatos da mesma maneira que fizemos com a responsabilidade das ferramentas pelos artefatos, conforme mostra a figura 4.5.

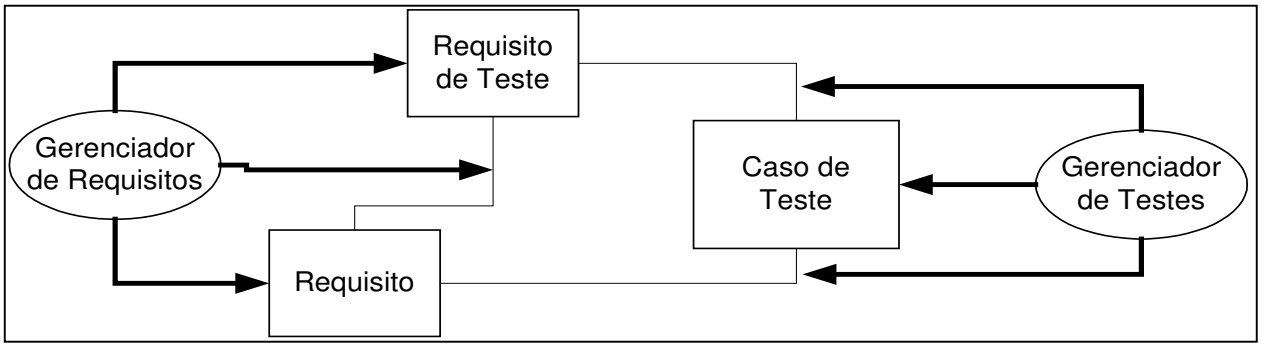


Figura 4.5: Ferramentas e os relacionamentos entre os artefatos

Relacionamentos entre Falha, Requisito de Software e Caso de Teste

Vamos analisar os relacionamentos entre Falha e Requisito e entre Falha e Caso de Teste na figura 4.6. O relacionamento entre Falha e Requisito deve ser de responsabilidade do Gerenciador de Falhas, pois é no momento do cadastramento da falha oriunda de revisões que deve ser feita a sua associação com a qual requisito de software diz respeito à falha. Também será de responsabilidade do Gerenciador de Falhas a associação entre Falha e Caso de Teste, pois é no registro da falha oriunda dos testes que haverá a sua associação com o caso de teste que gerou a falha. A figura 4.7 representa estas conclusões.

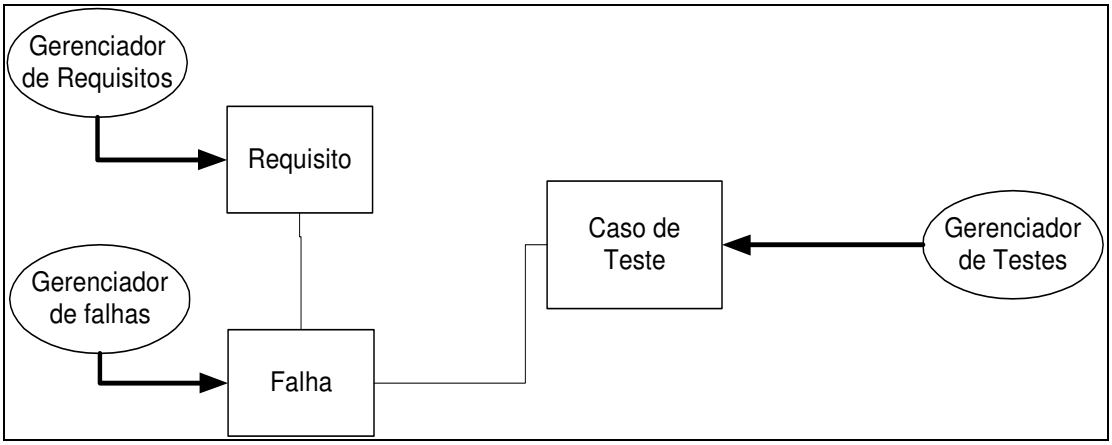


Figura 4.6: Subconjunto Requisito, Falha e Caso de Teste

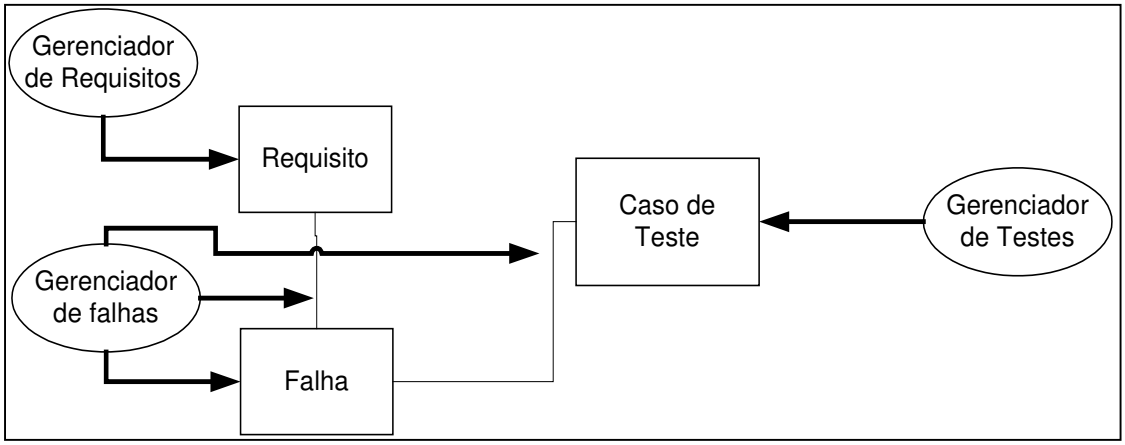


Figura 4.7: Responsabilidade pelos relacionamentos

4.3.4 Modelo das Ferramentas e os Relacionamentos entre Artefatos

Outras análises foram feitas análogas as que descrevemos nas seções anteriores e permitiram a construção de um modelo que demonstra as responsabilidades das ferramentas pelos relacionamentos entre os artefatos, ampliando nosso modelo anterior que já possuía a responsabilidade das ferramentas pelos artefatos, a figura 4.8 apresenta este modelo.

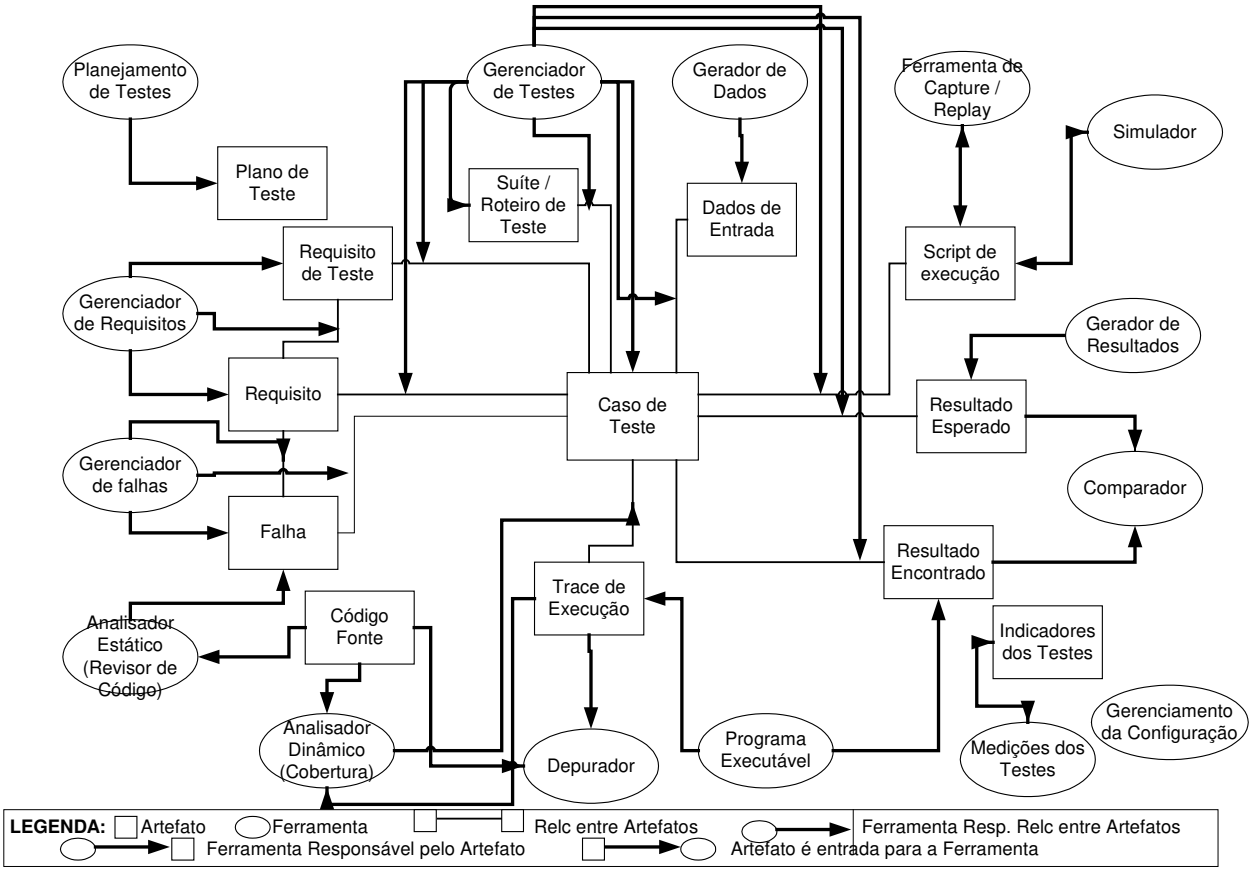


Figura 4.8: Modelo com os artefatos e as ferramentas responsáveis por gerá-los

4.3.5 Modelo de Integrações via Passagem de Dados

Analisando-se os relacionamentos entre os artefatos e as ferramentas responsáveis por eles no modelo apresentado na seção anterior pode-se concluir qual é a necessidade de passagem de dados entre as ferramentas envolvidas. Por exemplo, para que a ferramenta de Gerenciamento de Testes possa fazer a associação do caso de teste a um requisito de software, é necessário que tenha disponível uma lista de requisitos de software para permitir a seleção. Isso significa que há a necessidade de uma integração entre ferramentas com passagem de dados: a ferramenta de Gerenciamento de Requisitos precisa passar dados para a ferramenta de Gerenciamento de Testes.

O modelo apresentado na figura 4.9 representa a necessidade de integração entre as ferramentas via passagem de dados, ou seja, uma ferramenta necessita de dados de uma outra

ferramenta para poder fazer algo. Optamos por não evoluir o modelo apresentado nas seções anteriores para não deixá-lo mais "poluído" visualmente. As linhas em destaque são relativas às integrações qiue consideramos prioritárias.

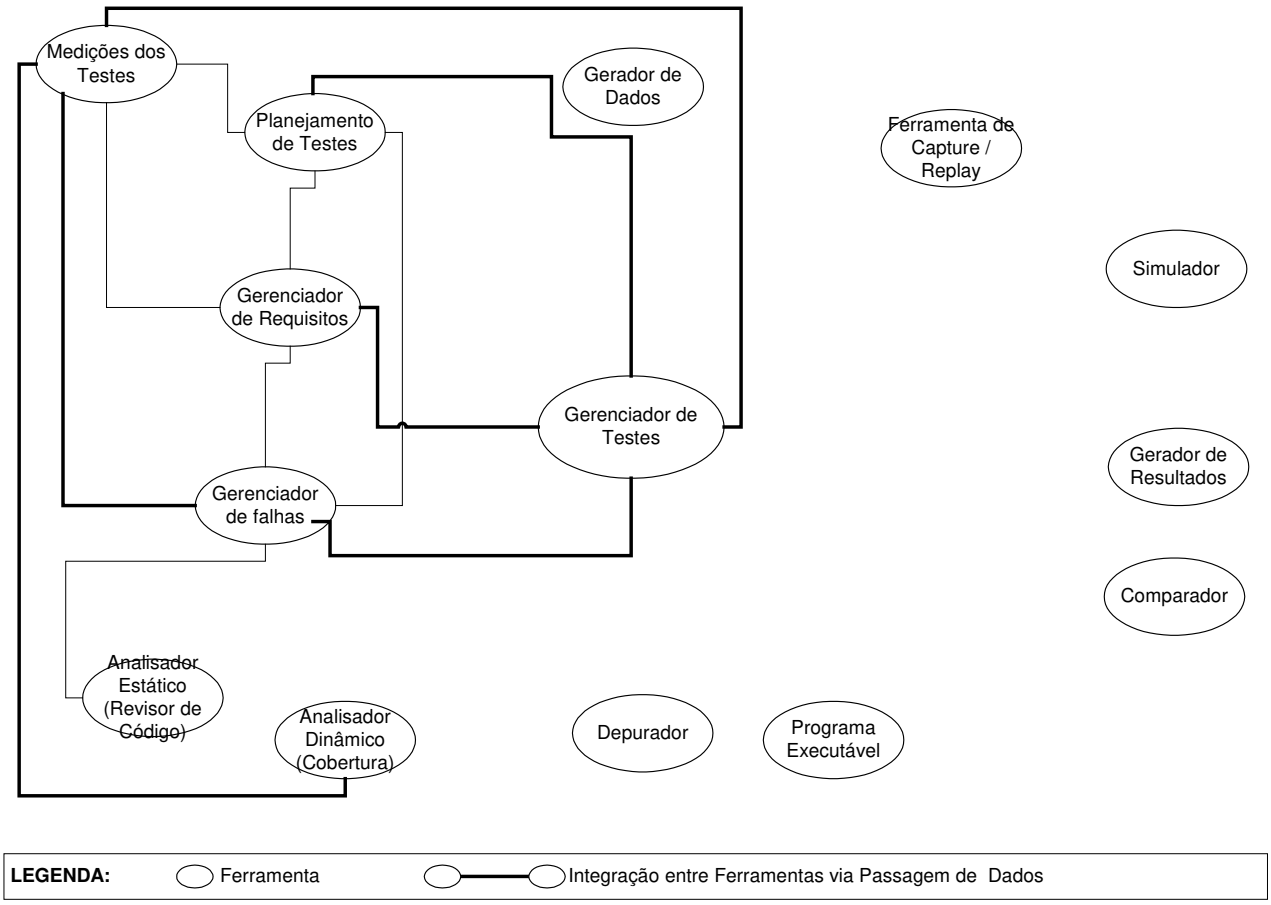


Figura 4.9: Necessidade de Integração via Passagem de Dados

Como vimos, para a arquitetura de automação adotada, a ferramenta de planejamento não pode estar isolada das demais, é esperado que troque dados com as ferramentas de gerenciamento de testes, de requisitos, de gerenciamento de falhas e de medições.

Também de acordo com a arquitetura de automação a ferramenta de medições é responsável por receber dados de outros módulos e concentrar as medições esperadas, para isso deve se integrar com as ferramentas de gerenciamento do teste, de planejamento, de cobertura, de requisitos e com a ferramenta de gerenciamento de falhas.

É necessário que a ferramenta de Gerenciamento de Falhas troque dados com a de Gerenciamento de Requisitos para que possa associar um requisito de software a uma falha encontrada nas revisões.

4.3.6 Modelo de Integrações via Transferência de Controle

A necessidade de integração entre as ferramentas não é somente de passagem de dados entre elas. Algumas vezes é necessário que uma ferramenta possa chamar (iniciar / ativar) a execução de uma outra. Por exemplo, na execução dos testes é necessário que a ferramenta de Gerenciamento de Teste chame a ferramenta de Geração de Dados para gerar uma massa de dados para o teste, chame a ferramenta de *Capture & Replay* para reproduzir os passos do teste, chame o Gerador de Resultados para produzir o resultado esperado, chame o Comparador para determinar se o resultado esperado e o encontrado são os mesmos e chame a ferramenta de Gerenciamento de Falhas para o registro da falha quando ela ocorre.

Como vimos, é necessária a integração entre as ferramentas de Gerenciamento de Testes e de Depuração. Para verificar a cobertura é necessário que a ferramenta de Gerenciamento dos Testes chame a de Cobertura. Por sua vez, o Programa Executável é chamado pelas ferramentas de *Capture & Replay* e de Simulação.

Estas análises permitiram a construção de um modelo que demonstra as necessidades de integrações entre as ferramentas quanto ao aspecto de uma ferramenta chamar a outra. Estas chamadas estão representadas pelas linhas pontilhadas na figura 4.10. As linhas em destaque são relativas às integrações qiue consideramos prioritárias.

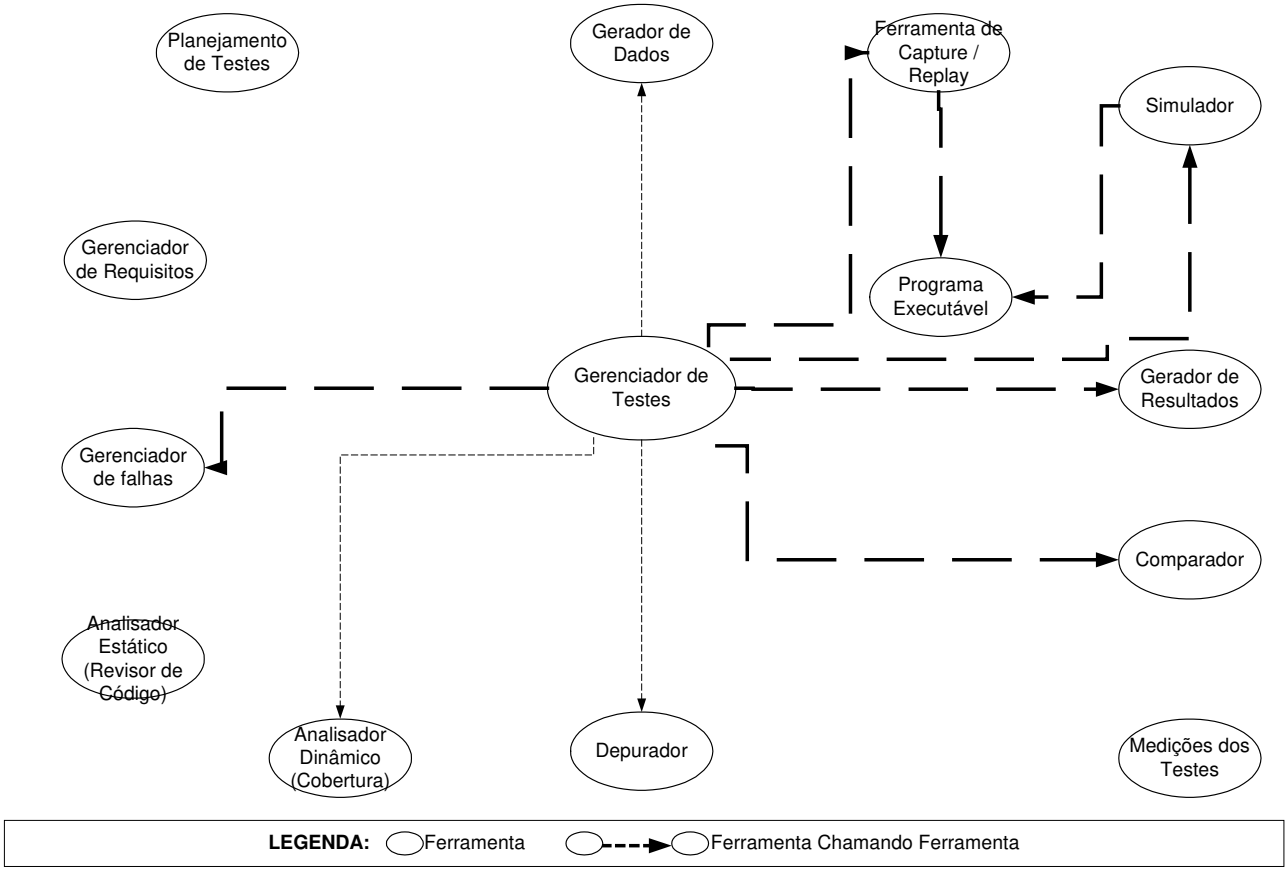


Figura 4.10: Necessidade de Integração via Chamada

4.4 Resumo do Capítulo

Neste capítulo apresentamos uma arquitetura de automação em testes e os requisitos de automação de testes baseados na literatura e na metodologia de testes existente (MTS). Agrupamos estes requisitos de automação pelas seis grandes funcionalidades do teste: planejamento, desenvolvimento, execução, análise de falhas, medições e gerência de configuração dos artefatos de testes. Também apresentamos modelos que representam características desejadas que a automação contemple e detalham ainda mais os requisitos de automação levantados: o modelo de dados dos artefatos, modelos que demonstram a responsabilidade que as ferramentas têm sobre os artefatos e sobre os relacionamentos entre os artefatos e modelos que indicam as necessidades de integração entre as ferramentas. No capítulo seguinte determinaremos se as ferramentas existentes na empresa do campo de estudo deste trabalho suprem ou não as necessidades apresentadas neste capítulo.

Capítulo 5

Ferramentas Disponíveis e o Atendimento aos Requisitos de Automação

Neste capítulo passaremos a trabalhar com as ferramentas reais existentes na empresa do campo de estudo deste trabalho, não mais com ferramentas hipotéticas como no capítulo 4. Por exemplo, enquanto no capítulo 4 tínhamos a ferramenta Gerenciador de Testes, aqui passaremos a denominar de TestDirector (ferramenta que será definida na próxima seção). Determinaremos que ferramentas, dentre as disponíveis, deveriam atender alguns dos requisitos de automação levantados no capítulo 4 e se efetivamente ocorre este atendimento. Para isso vamos primeiro descrever sucintamente cada ferramenta e depois partiremos para a análise dos requisitos de automação.

5.1 Ferramentas Disponíveis

Nesta seção descreveremos as ferramentas relacionadas a testes que já estão disponíveis no nosso campo de estudo. Apresentaremos para cada ferramenta as suas funcionalidades, plataforma e outros dados relevantes para este trabalho. Esta coleta de informações foi feita a partir de consultas aos fornecedores, aos sites e nos manuais das ferramentas. A tabela 5.1 apresenta as ferramentas para plataforma mainframe.

Tabela 5.1: Ferramentas Disponíveis no Mainframe

Ferramenta	**Principais Funcionalidades**
SmartTest ([Asg05])	▪ Depuração de problemas e erros na execução de programas *batch e on-line* ▪ Atende as linguagens COBOL e PL/I ▪ Acompanhamento interativo da execução de programas, com análise do fluxo da lógica, controle da sua execução e análise do

Ferramenta	Principais Funcionalidades
	conteúdo dos dados manuseados pelo programa ▪ Verificação de cobertura de código dos programas ▪ Opção de gravar a sessão de teste para registro histórico
DB/IQ-QA ([Rsi05])	▪ Medir e manter a qualidade de comandos SQL para DB2, evitando problemas causados por SQL mal codificado ▪ Evita problemas de desempenho e segurança ▪ Possui regras de validação já disponíveis e permite a criação de novas
File-AID/MVS ([Com05a])	▪ Agilidade na localização e alocação de arquivos ▪ Criação e armazenamento de critérios (filtros) ▪ Edição e atualização de registros de forma eficiente, respeitando as regras de segurança ▪ Comparação de arquivos ▪ Pesquisa e alteração de ocorrências de dados inválidos ▪ Pesquisa e atualização de grandes mudanças ▪ Reformatação de arquivos ▪ Junção de dados de diferentes arquivos em um único ▪ Detalhamento das mudanças em relatórios de comparação e em trilhas de auditoria ▪ Processamento interativo ou *batch*
File-AID/Data Solutions ([Com05b])	▪ Descaracterização de dados ▪ Geração de massas de teste ▪ Manipulação e validação de datas ▪ Simulação de alterações ▪ Verificação de dados de fontes externas ▪ Cálculos em registros com múltiplos *layouts*
File-Aid/DB2 ([Com05c])	▪ Criação e cópia de bases DB2 ▪ Extração e carga ▪ Geração de dados ▪ Edição e atualização

Ferramenta	Principais Funcionalidades
	▪ Possui trilha de auditoria ▪ Descrição detalhada de erros ▪ Prototipação sem recompilação
File-Aid/RDX ([Com05d])	▪ Facilita a criação de um ambiente integrado de testes ▪ Bases de dados e arquivos sincronizados ▪ Restauração dos dados com facilidade
QAHiperstation ([Com05e])	▪ Utilização de ferramenta nos ambientes *on-line* e *batch* para automatização da execução de testes ▪ Simulação de usuários e/ou aplicações remotas ▪ Utilização de ferramenta para simulação de carga (*stress*), visando verificar o desempenho da aplicação ▪ Apóia o uso das abordagens data-driven e *record & playback*, não a dirigida a palavras-chave ▪ Indica as diferenças encontradas ▪ Cria documentação automática dos testes ▪ Apresenta as teclas de funções acionadas e tempo de resposta ▪ Controle e agendamento efetivo de cada roteiro de teste ▪ Retorno à posição inicial das bases de dados envolvidas nas alterações realizadas ▪ Testes em ambientes heterogêneos, se comunicando e trocando informações

A tabela 5.2 apresenta as ferramentas relacionadas a testes para a baixa plataforma.

Tabela 5.2: Ferramentas Disponíveis para Baixa Plataforma

Ferramenta	Principais Funcionalidades
CaliberRM ([Bor05a])	▪ Gerenciador de requisitos ▪ Ambiente colaborativo facilitando a comunicação entre os membros da equipe de desenvolvimento ▪ Estimativas para o planejamento do projeto

Ferramenta	Principais Funcionalidades
	▪ Versionamento de requisitos de software e criação de baseline
TestDirector ([Mer05a])	▪ Gerenciador de testes manuais ou atomatizados ▪ Administração de requisitos de software ▪ Especificação, agendamento e execução de testes ▪ Administração de erros ▪ Facilita a comunicação e a colaboração entre os membros da equipe
QuickTest ([Mer05b])	▪ Automação da execução do teste e da análise do resultado ▪ Apóia as abordagens dirigida a palavras-chave, *data-driven* e *Capture & Replay* ▪ Usa a linguagem VBA ("*VisualBasic for Applications*") como linguagem dos scripts de teste ▪ Integração nativa com o TestDirector e com o WinRunner ▪ Pode ser utilizada em vários ambientes, incluindo: Windows, Web, .NET, Java/J2EE, SAP, Oracle, PeopleSoft, Visual Basic, ActiveX e emulação de terminais mainframe.
WinRunner ([Mer05c])	▪ Automação da execução do teste e da análise do resultado ▪ Apóia o uso das abordagens data-driven e *record & playback*, não a dirigida a palavras-chave ▪ Criação de script em linguagem proprietária similar ao C ▪ Integração transparente com o TestDirector e com o QuickTest ▪ Pode ser utilizada em vários ambientes, por exemplo: C/S (Java, Visual Basic, ActiveX), Web (HTML, DHTML, JavaScript), emulação de terminais mainframe 3270, Browsers (Internet Explorer, Netscape, AOL), entre outros.
LoadRunner ([Mer05d])	▪ Testes de carga e desempenho com criação de script ▪ Emula centenas ou milhares de usuários utilizando a aplicação ▪ Faz análise de desempenho e realiza diagnóstico através de monitores não intrusivos ▪ Possui mecanismo que faz correlação entre diversos fatores para

Ferramenta	Principais Funcionalidades
	apresentar possíveis causas de problemas ▪ Integra-se com o TestDirector, com o QuickTest e com o WinRunner ▪ Pode ser utilizada em diversos ambientes, como: Web, Client-Server, Citrix, Java, .NET, PeopleSoft, SAP, Oracle e Siebel
Devpartner ([Com05g])	▪ Análise de desempenho ▪ Verificação de cobertura de código ▪ Depuração remota ▪ Revisão de código ▪ Simulação de erros ▪ Acompanhamento interativo da execução de programas, com análise do fluxo da lógica, controle da sua execução e análise do conteúdo dos dados manuseados pelo programa ▪ Linguagens: .NET, Java ▪ Plataformas: Windows, Solaris, Linux, Aix
FileAid/CS ([Com05h])	▪ Geração de massa de testes ▪ Editor de dados para banco de dados ▪ Comparação automática de base de dados para validar testes ▪ Cópia, transformação e conversão de diversos tipos e formatos de dados ▪ Extração e carga de banco de dados ▪ Trabalha com uma grande variedade de banco de dados

5.2 Atendimento aos Requisitos de Automação pelas Ferramentas

Nesta seção definiremos se as ferramentas existentes na empresa deste campo de estudo atendem ou não alguns dos requisitos de automação definidos no capítulo 4. É importante ressaltar que partes dos atendimentos aos requisitos de automação foram definidas de acordo com as descrições fornecidas, portanto há a necessidade de serem validados na prática, enquanto que para outros esta validação já foi feita.

5.2.1 Relacionamento entre Requisito de Software e Teste

A tabela 5.3 apresenta, para os requisitos de automação do relacionamento entre requisito de software e teste, qual a ferramenta responsável pelo atendimento e uma avaliação se este atendimento é alcançado.

Tabela 5.3: Automação do Relacionamento entre Requisito de Software e Teste

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
2.01 - Tratamento do ciclo de vida do artefato Requisito de Teste	Caliber	Atende parcialmente, pois permite somente o cadastramento de um texto corrido (não estruturado)
2.02 - Que o gerenciamento de Requisitos de Software e o gerenciamento do teste sejam feitos por uma mesma ferramenta ou com uma forte integração	Caliber / TestDirector	Atendem
2.03 - Mudanças nos Requisitos de Software devem ser percebidas nas ferramentas que tratam os testes	TestDirector e Caliber	Não atendem
2.04 - A descrição do Requisito de Software deve estar disponível nos testes	TestDirector e Caliber	Atendem
2.05 - Para priorizar os testes os Requisitos de Software devem ter a informação do risco	Caliber	Não atende
2.06 - A informação do Risco do Requisito de Software deve estar disponível nos testes	TestDirector e Caliber	Não atendem
2.07 - Permitir que o Requisito de Teste seja validado sem a necessidade do Caso de Teste	TestDirector	Não atendem

Requisito de Automação Id. Requisito – Descrição Resumida	Ferramenta	Avaliação do Atendimento
2.08 - Manter a rastreabilidade entre Requisito de Software e Requisito de Teste	TestDirector e Caliber	Não Atendem
2.09 - Manter a rastreabilidade entre Requisito de Teste e Caso de Teste	TestDirector e Caliber	Não atendem
2.10 - Manter a rastreabilidade entre Requisito de Software e Caso de Teste	TestDirector e Caliber	Atendem
2.11 - Disponibilizar as funcionalidades que são supridas pelo documento GRT – Gestão e Rastreabilidade dos Testes	TestDirector e Caliber	Não atendem

5.2.2 Desenvolvimento dos Artefatos Básicos de Testes

Serão apresentadas na tabela 5.4 as ferramentas responsáveis pelo atendimento aos requisitos da automação da construção dos artefatos básicos de testes e se este atendimento é alcançado.

Tabela 5.4: Automação da Construção dos Artefatos Básicos

Requisito de Automação Id. Requisito – Descrição Resumida	Ferramenta	Avaliação do Atendimento
3.01 - Tratamento do ciclo de vida do artefato Caso de Teste	TestDirector	Atende
3.02 - Tratamento do ciclo de vida do artefato Roteiro de Teste	TestDirector	Atende parcialmente, pois permite a utilização de diretórios como alternativa
3.03 - Tratamento do ciclo de vida do artefato Suíte de Teste	TestDirector	Atende parcialmente, pois permite a utilização de diretórios

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
		como alternativa
3.04 - Tratamento do ciclo de vida do artefato Critério de Teste	TestDirector	Não atende
3.05 - Geração de dados de Teste através da seleção de dados de um banco de dados	File-AID	Atende
3.06 - Geração de dados de Teste aleatórios a partir de uma gramática	File-AID	Atende
3.07 - Geração de dados para popular adequadamente um banco de dados	Em análise	Em análise

5.2.3 Execução dos Testes

Os atendimentos aos requisitos da automação da execução do teste pelas ferramentas são apresentados na tabela 5.5.

Tabela 5.5: Automação da Execução do Teste

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
5.01 - Utilização de Driver de Teste	QuickTest, WinRunner, Hiperstation	Atendem desde que adequadamente programado
5.02 – O Driver deve utilizar um diretório de execução para guardar os arquivos utilizados no teste	QuickTest, WinRunner, Hiperstation	Atendem desde que adequadamente programado
5.03 - Utilização de arquitetura *Capture & Replay*	QuickTest, WinRunner, Hiperstation	Atendem
5.04 - Desenvolvimento de scripts sem o auxílio	QuickTest,	Atendem

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
de ferramentas	WinRunner, Hiperstation	
5.05 - Desenvolvimento de scripts a partir de scripts gerados pela ferramenta	QuickTest, WinRunner, Hiperstation	Atendem
5.06 - Biblioteca de scripts genéricos	QuickTest, WinRunner, Hiperstation	Não atendem
5.07 - Biblioteca de scripts modificáveis	QuickTest, WinRunner, Hiperstation	Não atendem
5.08 - Arquitetura *Data-Driven* (ver seção 4.2.3)	QuickTest, WinRunner, Hiperstation	Atendem
5.09 - Arquitetura dirigida a Palavras-Chaves (ver seção 4.2.3)	QuickTest, WinRunner, Hiperstation	QuickTest: Atende (a ser confirmado na prática); WinRunner e HiperStation: não atendem
5.10 - Utilização de um vocabulário independente para a Arquitetura dirigida a Palavras-Chaves	QuickTest, WinRunner, Hiperstation	Não atendem
5.11 - Simulação da máquina que o programa será executado	Não definida	Em análise
5.12 - Simulação de interação com múltiplos usuários	LoadRunner	Atende
5.13 - Permitir escolher a execução de alguns casos de teste ou de toda uma suíte de teste	TestDirector	Atende
5.14 - Facilidade de não executar os testes mais	TestDirector	Atende parcialmente, pois

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
propensos a falhas		permite a seleção de que teste executar, mas não possui a informação de propensão a falha
5.15 - Execução de casos de teste em paralelo	TestDirector	Atende parcialmente, pois tem de estar em diferentes máquinas
5.16 - Execução automática dos casos de testes que falharam	TestDirector	Atende
5.17 - Integração entre as ferramentas de Gerenciamento de Teste e Analisador Dinâmico / Debug	TestDirector e SmartTest / TestDirector e DevPartner	Não atendem
5.18 - Integração entre as ferramentas Gerenciador de Teste e Gerador de Dados	TestDirector e File-AID	Não atendem
5.19 - Integração entre as ferramentas de Gerenciamento de Teste e *Capture & Replay*	TestDirector e HiperStation/ QuickTest/ WinRunner	TestDirector e HiperStation: Não atendem. TestDirector e QuickTest/WinRunner: Atendem
5.20 - Suporte ao Debug com geração de registro (trace) de execução	SmartTest e DevPartner	Atendem
5.21 - Armazenagem do registro (*trace*) de execução em repositório para ser utilizado pelo módulo de medições	SmartTest e DevPartner	SmartTest: não armazena em repositório. DevPartner: Em análise
5.22 - Permitir que partes do teste sejam executadas manualmente e partes de maneira automatizada	TestDirector	Não atende
5.23 - Registrar informações para o Roteiro de	TestDirector	Atende

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
Teste sobre a sua execução		
5.24 - Registrar para o caso de teste o correto resultado da execução do teste	TestDirector	Atende

5.2.4 Análise de Falhas

A tabela 5.6 relaciona as ferramentas e o atendimento aos requisitos de automação da análise de falhas.

Tabela 5.6: Automação da Análise de Falhas

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
6.01 - Chamar Gerador de Resultados para determinar qual o resultado esperado do teste	TestDirector e QuickTest / WinRunner / HiperStation	Atendem
6.02 - Chamar Comparadores para determinar se o resultado esperado e encontrado são os mesmos	TestDirector e QuickTest / WinRunner / HiperStation	Atendem
6.03 - Tratamento do ciclo de vida da Falha	TestDirector	Atende parcialmente, pois não possui algumas informações
6.04 - Histórico da Falha	TestDirector	Atende
6.05 - Workflow e controle de permissão para o tratamento da Falha	TestDirector	Atende
6.06 - Consulta a partir dos campos do registro da Falha	TestDirector	Atende

Requisito de Automação **Id. Requisito – Descrição Resumida**	**Ferramenta**	**Avaliação do Atendimento**
6.07 - Ambiente de aprendizado colaborativo e interativo para o tratamento das falhas	TestDirector	Não atende, pois apenas consulta a falhas do próprio projeto
6.08 - Tratamento de Falhas oriundas das revisões	TestDirector	Atende parcialmente, pois não possui a informação de requisito de software associado a falha
6.09 – Análise automatizada de código para linguagens estruturadas do Mainframe	Não Há	Não atende
6.10 - Análise automatizada de código para Sql DB2 do Mainframe	DB-IQ	Atende
6.11 - Integração do Gerenciamento de Falhas com o módulo de Medições	TestDirector	Atende parcialmente, não se integra, mas ele mesmo tira as métricas desejadas
6.12 - Integração do Gerenciamento de Falhas com a ferramenta de Gerenciamento de Teste	TestDirector	Atende

5.3 Resumo do Capítulo

Neste capítulo apresentamos as ferramentas disponíveis na empresa do campo de estudo deste trabalho com as suas principais funcionalidades e para alguns dos requisitos de automação que foram levantados no capítulo 4 determinamos que ferramentas já existentes são responsáveis pelos seus atendimentos e se esses são efetivamente alcançados. Os requisitos de automação que não foram atendidos serão alvo de busca de soluções de atendimentos que serão relatadas no capítulo seguinte.

Capítulo 6

Soluções Práticas para os Requisitos de Automação Não Atendidos

Enquanto no capítulo 4 listamos os requisitos de automação de teste e no capítulo 5 verificamos se estes são supridos pelas ferramentas disponíveis, neste capítulo iremos propor e acompanhar os encaminhamentos de soluções práticas para possibilitar o atendimento a alguns dos requisitos de automação não atendidos pelas ferramentas existentes. As seções a seguir apresentam acada um desses requisitos e as soluções dadas.

6.1 Relacionamento entre Requisito de Software e Teste

Nesta seção apresentaremos as soluções dadas para o atendimento ao relacionamento entre requisito e teste e ao artefato requisito de teste.

Implementação da informação do risco do requisito de software (IdReq: 2.05 e 2.06) (Não crítico)

Foi implementada a parametrização da ferramenta Caliber para incluir esta informação como risco do requisito de software para diferenciar do risco do projeto. Porém, não será possível a exportação desta informação para o TestDirector sem customização. Será analisada a real necessidade desta integração, pois queremos evitar customizações que possam trazer dificuldades com as futuras versões das ferramentas. Temos a tendência de encerrar esta pendência sem o atendimento completo, a alternativa seria consultar a ferramenta de gerenciamento de requisitos para obter a informação de risco do requisito de software.

Implementação do artefato requisito de teste e sua disponibilização nos testes (IdReq: 2.01) (Crítico)

Será permitido o cadastramento do requisito de teste já no Caliber em forma de rascunho. No TestDirector será feito o registro do requisito de teste e associado a ele os casos de testes.

Não ficou boa esta solução. No Caliber não foi suprida a estruturação de vários requisitos de testes para um requisito de software. É necessária nova digitação dos requisitos de teste na ferramenta de gerenciamento dos testes (TestDirector), os primeiros pilotos mostraram-se resistentes a fazê-lo. Não estão garantidas as rastreabilidades entre requisitos de software, requisitos de teste e casos de teste. Devemos buscar uma solução mais adequada, talvez com customizações com certa complexidade, ou diminuir nossas expectativas neste assunto.

Permitir validar o requisito de teste sem a necessidade do caso de teste (IdReq: 2.07) (Não crítico)

Não iremos prosseguir com a busca do atendimento a este requisito de automação, não há solução razoável para implementá-lo no TestDirector.

Garantir a rastreabilidade e permitir a visualização da hierarquia entre requisito de teste e requisito de software e entre requisito de teste e caso de teste (IdReq: 2.08 e 2.09) (Crítico)

Não é possível a rastreabilidade entre requisito de teste e requisito de software ou entre requisito de teste e caso de teste. A partir do Caliber será possível visualizar a rastreabilidade entre requisitos de software e entre requisitos de software e casos de testes. No TestDirector teremos a visualização entre requisitos de software, requisitos de teste e casos de testes desde que se cadastre o requisito de teste conforme já proposto nesta seção. A dificuldade em trabalhar com requisito de teste também ocorre neste item.

6.2 Artefatos básicos do Teste

Nesta seção apresentaremos as soluções dadas para o atendimento aos requisitos dos artefatos básicos do teste.

Implementação dos artefatos Roteiro e Suíte de Teste (IdReq: 3.02 e 3.03) (Crítico)

Estamos utilizando os diretórios (folder) do TestDirector para que se estruture os Roteiros e as Suítes de Teste. Implementamos a facilidade que para um novo projeto é disponibilizada uma estrutura modelo sendo permitido modificá-la. Assim, uma estrutura com diretórios para as fases de testes (Teste de Unidade, Teste de Integração, etc) estará disponível para cada projeto

criado. Caso se deseja uma estruturação ainda maior é sugerida a divisão destes diretórios em subdiretórios, por exemplo, o diretório Teste de Unidade dividido em suítes por programa e o Teste de Sistema dividido em stress, carga, desempenho e funcional. É uma solução adequada que atende aos interesses das equipes de maneira simples.

Implementação do artefato Critério de Teste (IdReq: 3.04) (Crítico)

Uma possível solução seria criar no TestDirector um campo de descrição para documentar o critério de teste. Consideramos esta solução fraca porque não permite o cadastramento estruturado do critério e a comparação entre o critério e a cobertura alcançada seria feita manualmente. Porém parece-nos a solução mais viável neste momento.

6.3 Execução do Teste

Nesta seção apresentaremos as soluções dadas para o atendimento aos requisitos relacionados à execução do teste.

6.3.1 Implementação de Framework para Mainframe Batch (IdReq: 5.01) (Crítico)

No mainframe foi desenvolvido um *framework* para gerar drivers que testem programas batch. Criamos um aplicativo que auxilia o testador na construção de um driver de testes para programa batch. O modo de operação *batch* possui algumas características, tais como: recebimento de arquivos de entrada, processamento de múltiplos registros e geração de arquivo de saída. Estas características foram incorporadas na construção do *framework*.

O framework deve gerar um driver de testes com as funções:

- Executar os scripts de montagem do ambiente de dados necessário;
- Executar o programa;
- Comparar arquivos de saídas mais recentes com arquivos armazenados da execução anterior (com filtros de comparação de arquivos);

- Verificar a cobertura de comandos do programa (esta funcionalidade utilizará a ferramenta SmartTest e ainda não está implementada);
- Gerar um "recibo" de evidência dos testes;

A figura 6.1 apresenta a primeira tela do *framework*, em que são solicitados os dados básicos sobre o driver a ser gerado.

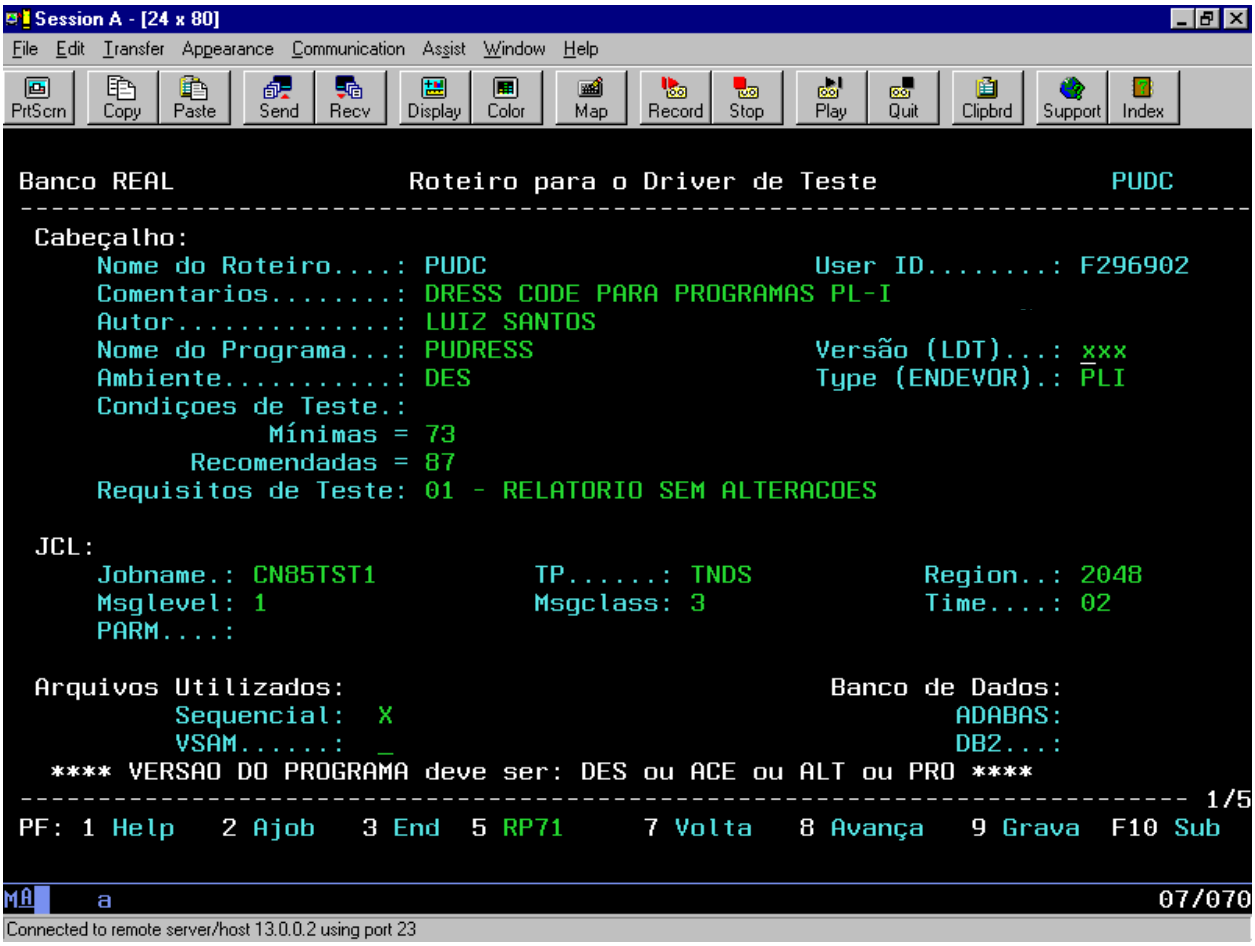


Figura 6.1: Tela de Solicitação dos dados básicos do Driver de Teste

O passo seguinte é fornecer as definições dos arquivos que servirão de entrada e dos arquivos de saída a serem gerados com a execução do programa a ser testado. A figura 6.2 apresenta a tela que solicita estas informações. Neste ponto tivemos que escolher qual seria a quantidade limite de arquivos de entrada e de saída. Resolvemos limitar porque é de mais fácil implementação, apesar de sabermos que a solução mais elegante seria permitir um número não limitado de arquivos. Fizemos alguns levantamentos e concluímos que cinco arquivos de entrada

e três de saída cobrem a grande maioria dos programas *batch*, deixa a aplicação amigável de ser utilizada e não é uma solução de alta complexidade.

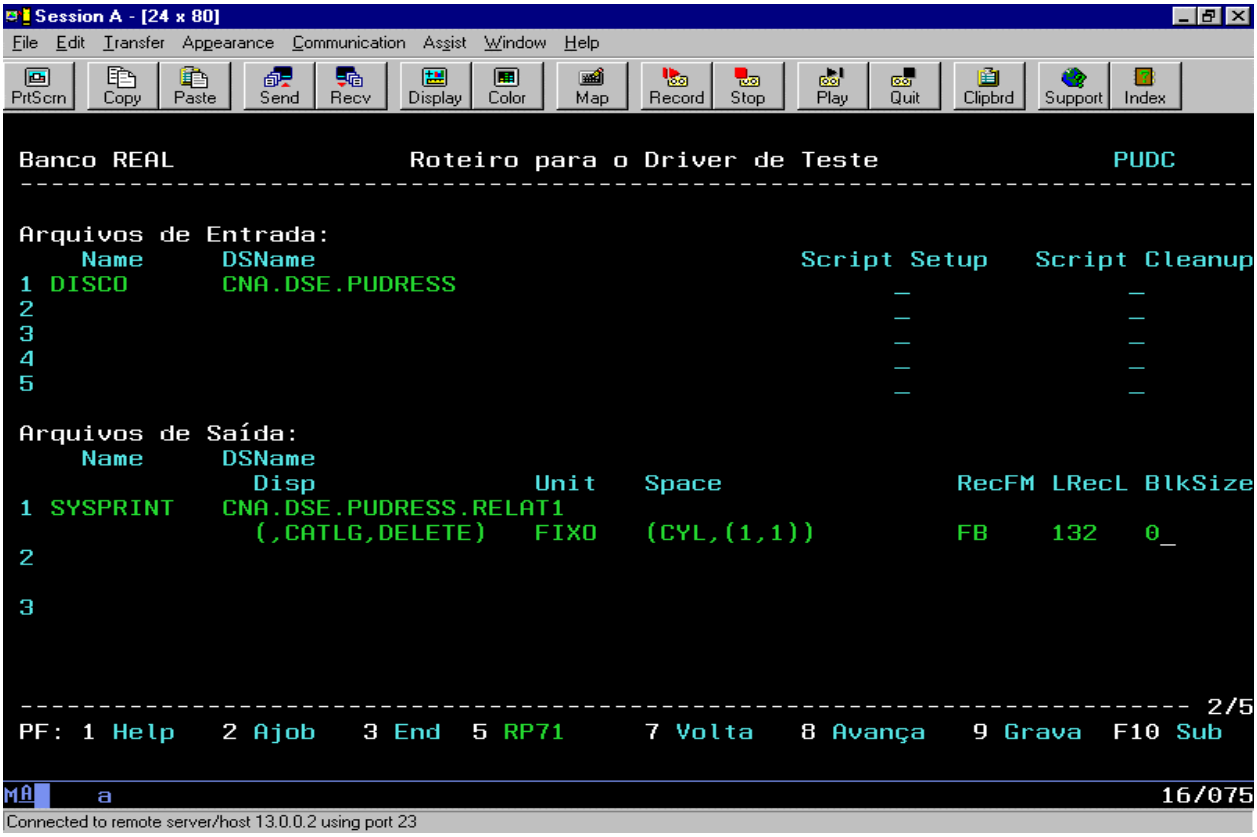


Figura 6.2: Solicitação dos Arquivos de Entrada e Saída

Uma outra tela é a que solicita informações sobre a análise de resultados, por enquanto implementamos esta análise através da comparação entre os arquivos de saída e arquivos contendo os dados esperados. A figura 6.3 apresenta esta tela, observe que são oferecidas as facilidades de: comparar partes dos arquivos e criar o arquivo esperado a partir do arquivo gerado pelo programa para que possa ser utilizado numa comparação futura.

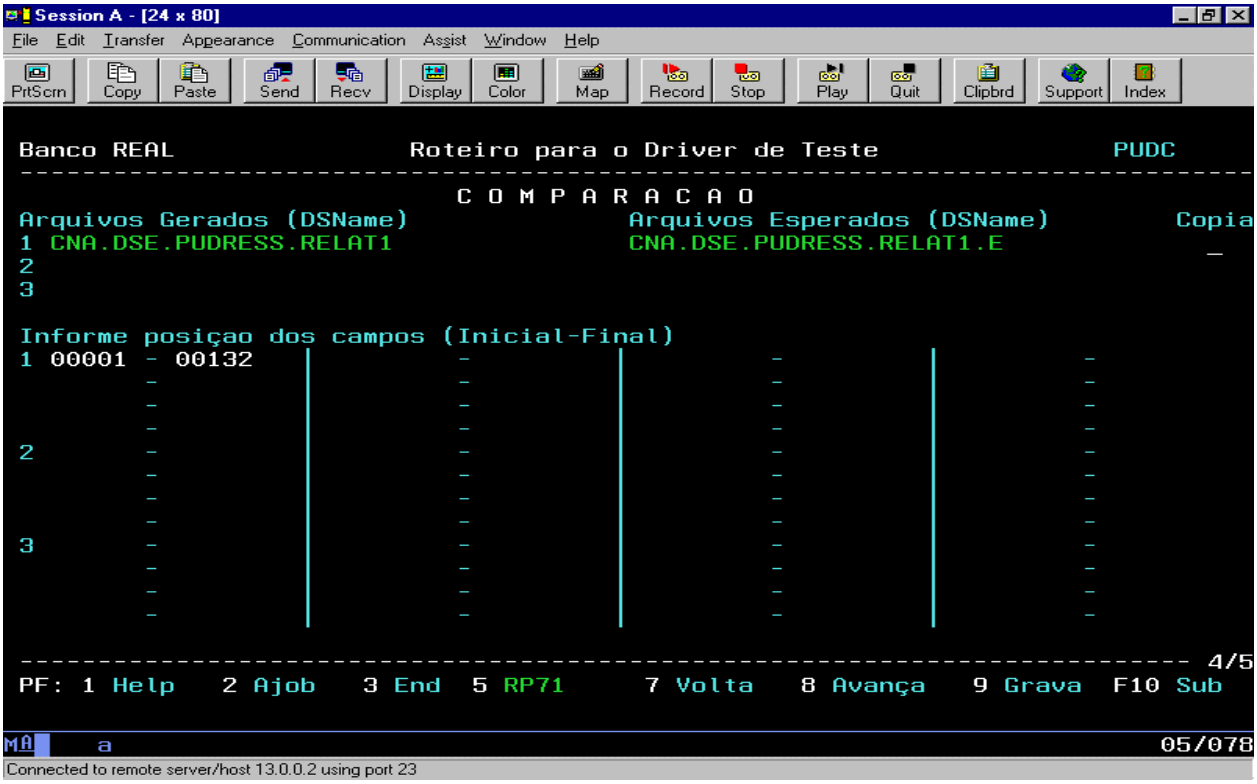


Figura 6.3: Comparação entre o Resultado Esperado e o Encontrado

As figuras 6.4 e 6.5 apresentam trechos do jcl (que atua como um Driver de Testes) gerados pelo framework como a sua utilização em um projeto piloto.

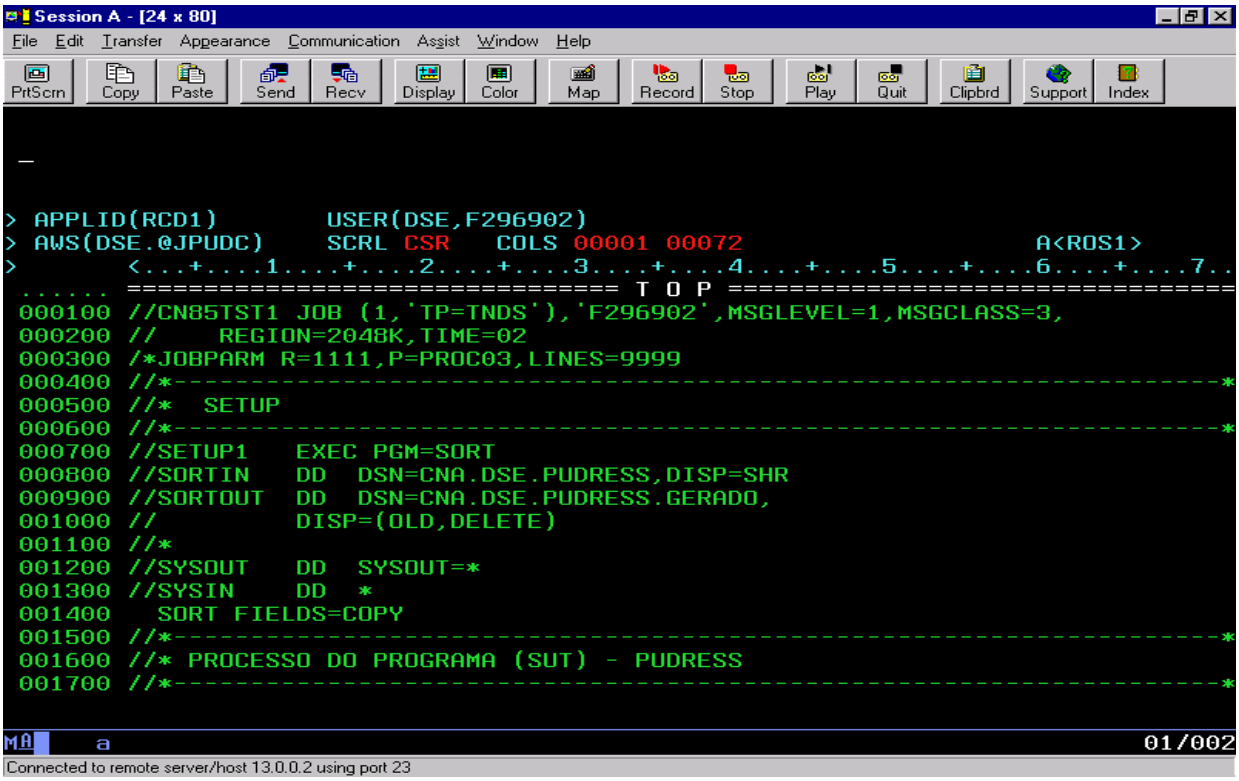


Figura 6.4: Trecho do JCL gerado pelo Framework

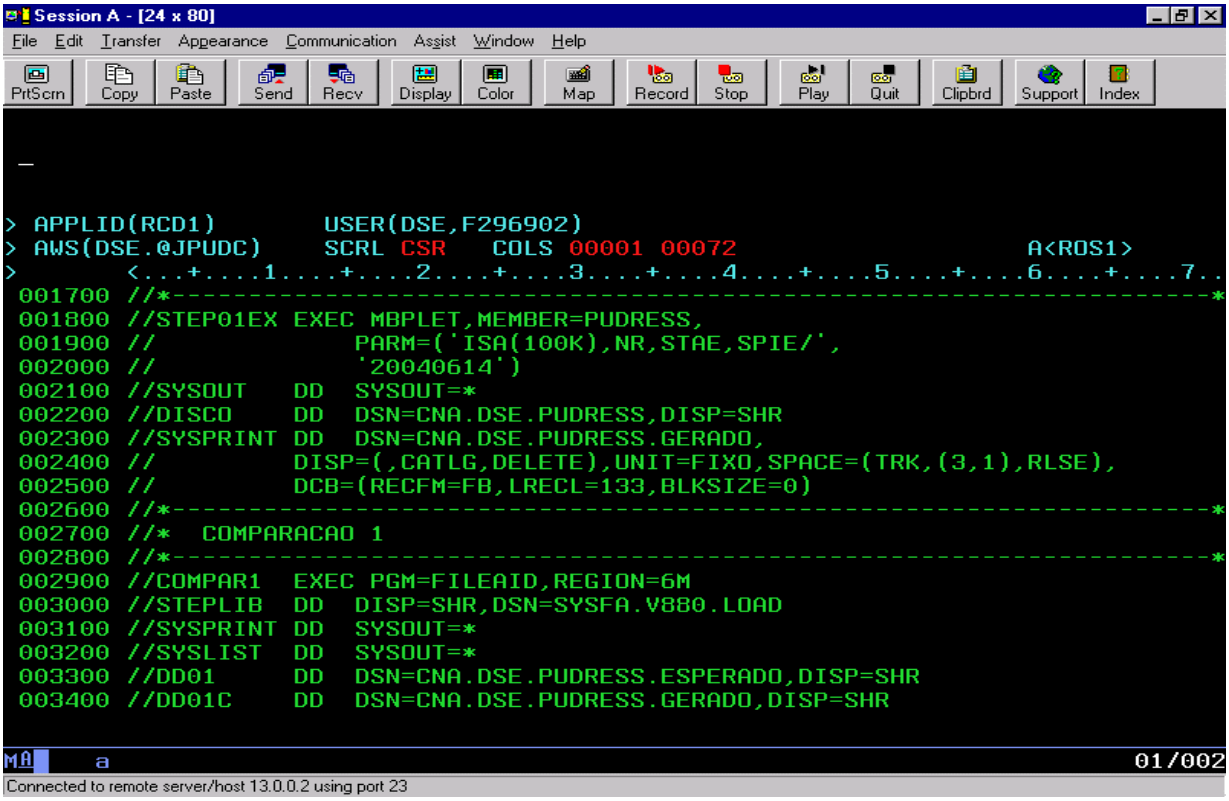
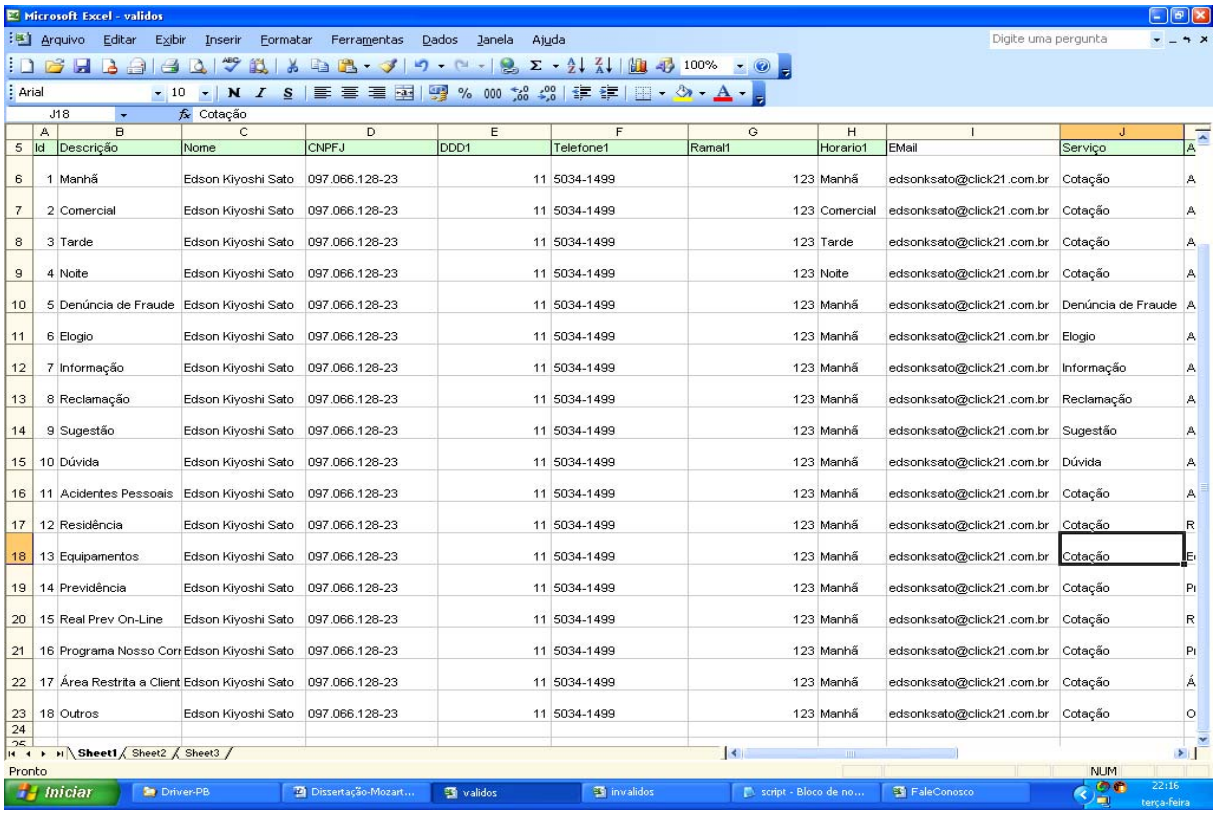


Figura 6.5: Trecho do JCL gerado pelo Framework

6.3.2 Implementação dirigida a Palavras-Chaves para Baixa Plataforma (IdReq: 5.01, 5.08 , 5.09, 5.10 e 5.19) (Crítico)

No caso da baixa plataforma está sendo desenvolvido um framework que gere drivers para testar programas on-line. Ainda se trata de um experimento utilizando a ferramenta WinRunner, no qual estamos utilizando os conceitos de data-driven e vocabulário independente. O script é gerado na linguagem nativa do WinRunner que possui a mesma sintaxe do C. No projeto piloto utilizamos um aplicativo Web escrito na linguagem C++.

Com a utilização do conceito Data-Driven os dados foram separados do script e colocados em arquivos Excel, divididos em dados válidos e inválidos, conforme as figuras 6.6 e 6.7.



	A	B	C	D	E	F	G	H	I	J	
5	Id	Descrição	Nome	CNPFJ	DDD1	Telefone1	Ramal1	Horario1	EMail	Serviço	A
6	1	Manhã	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	A
7	2	Comercial	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Comercial	edsonksato@click21.com.br	Cotação	A
8	3	Tarde	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Tarde	edsonksato@click21.com.br	Cotação	A
9	4	Noite	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Noite	edsonksato@click21.com.br	Cotação	A
10	5	Denúncia de Fraude	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Denúncia de Fraude	A
11	6	Elogio	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Elogio	A
12	7	Informação	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Informação	A
13	8	Reclamação	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Reclamação	A
14	9	Sugestão	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Sugestão	A
15	10	Dúvida	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Dúvida	A
16	11	Acidentes Pessoais	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	A
17	12	Residência	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	R
18	13	Equipamentos	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	E
19	14	Previdência	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	P
20	15	Real Prev On-Line	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	R
21	16	Programa Nosso Corr	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	P
22	17	Área Restrita a Client	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	Á
23	18	Outros	Edson Kiyoshi Sato	097.066.128-23	11	5034-1499	123	Manhã	edsonksato@click21.com.br	Cotação	O
24											

Figura 6.6: Dados válidos para a execução do Driver

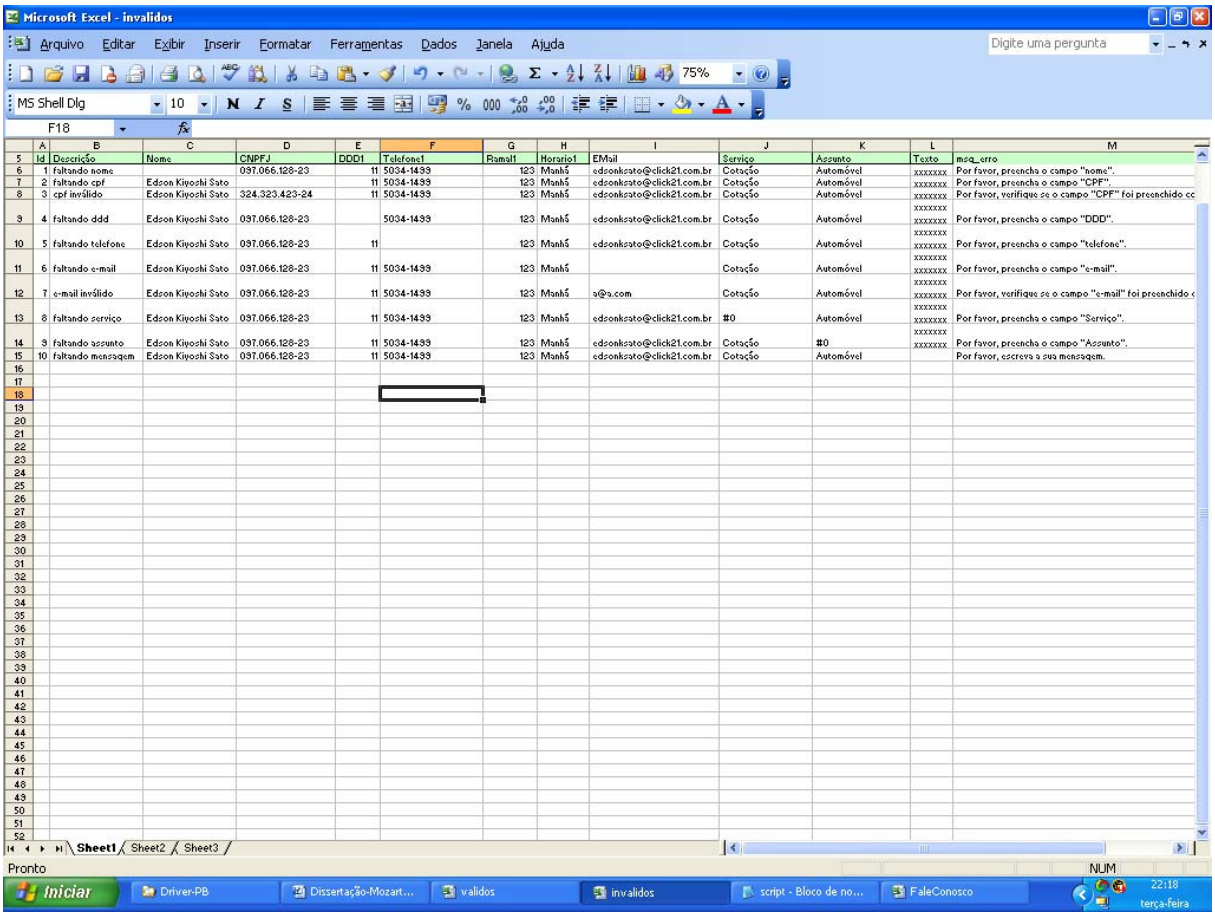


Figura 6.7: Dados inválidos para a execução do Driver

A figura 6.8 apresenta um trecho da planilha Excel em que é utilizada a arquitetura dirigida a Palavras-Chaves. Através de um vocabulário independente de ferramenta são definidos os passos que devem ser realizados para a execução do teste. A partir destes passos é gerado pelo *Framework* o script para a execução dos testes através de um Driver de Teste.

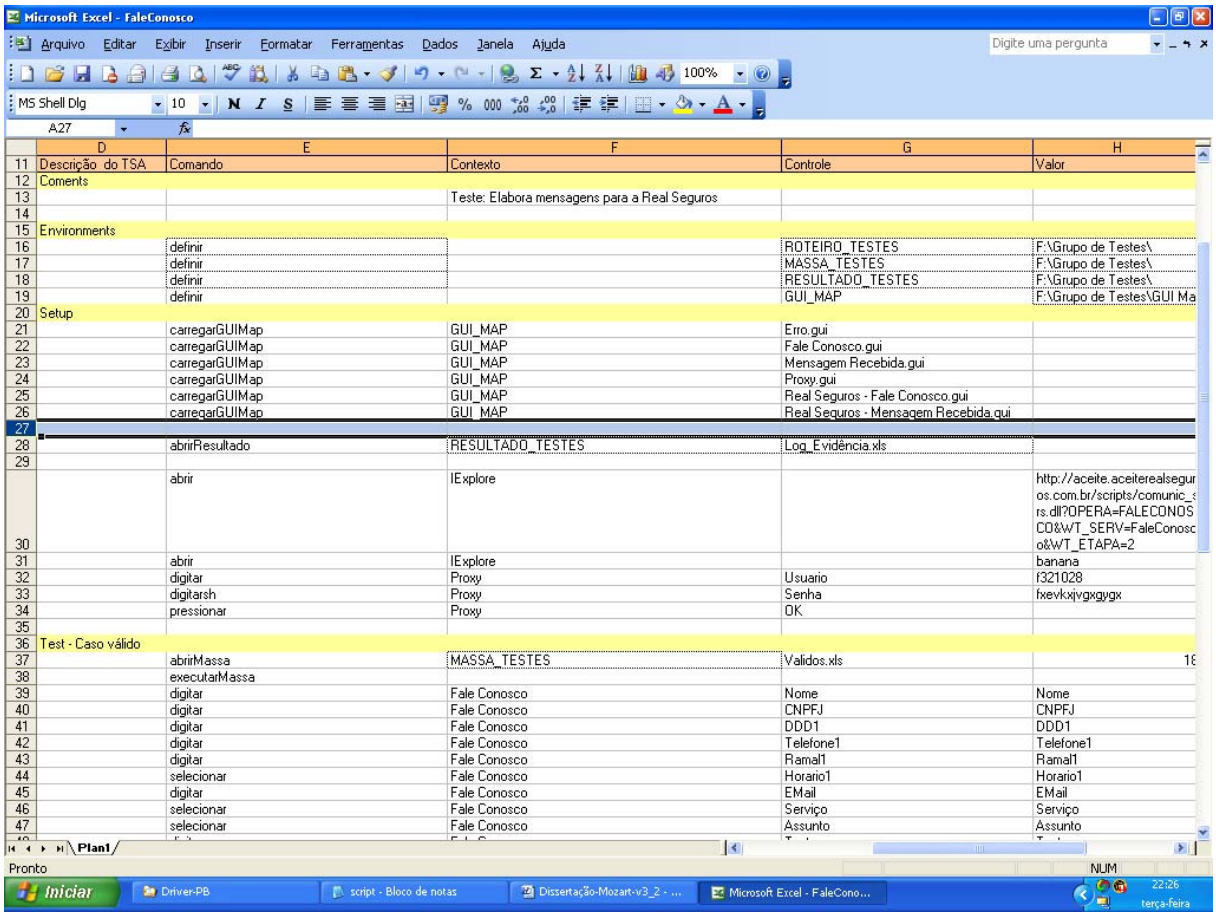


Figura 6.8: Especificação do Teste a ser executado pelo Driver

6.3.3 Integração entre Ferramentas de Testes (IdReq: 5.17, 5.18, 5.19 e 6.11) (Crítico)

De uma maneira geral, as ferramentas de fornecedores diferentes não se integram entre si, esta situação fica ainda mais notória quando falamos de integração entre ferramentas do mainframe com ferramentas de outras plataformas. Estamos estruturando um projeto que irá abordar o tema considerando as necessidades de integração, entre outras, da ferramenta de Gerenciamento do Teste (TestDirector) com a ferramenta de Geração de Dados (File-Aid), com as de Cobertura e Depuração (SmartTest e DevPartner) e com a de *Capture & Replay* do mainframe (HiperStation).

Os requisitos de automação de integração entre ferramentas podem ser agrupados em quatro tipos ([Was90]):

- Integração de interface de usuário: este tipo de integração possibilita que as ferramentas tenham similares look-and-feel e estilos de interação.
- Integração de dados: necessária porque diferentes ferramentas podem necessitar trocar dados, resultados de uma ferramenta podem ser passados como entrada para outra. Pode ser alcançada através de um repositório compartilhado ou diretamente através de transferência de dados.
- Integração de controle: ferramentas devem estar habilitadas a notificar umas as outras de eventos, ativar umas as outras e compartilhar funções. Padrões têm sido desenvolvidos para especificar um conjunto de protocolos de mensagem padrão. É algumas vezes implementado através de produtos de framework de integração de controle.
- Integração de Processo: Ferramentas são ativadas numa ordem consistente com o modelo de processos. Ou seja, as ferramentas são integradas às atividades de engenharia de software de acordo com um modleo de processos.

As fronteiras entre estes tipos de integração muitas vezes se confudem, por exemplo, a integração de processo pode utilizar-se da integração por controle para alcançar os seus objetivos.

6.3.4 Não execução de casos de testes propensos à falha (IdReq: 5.14) (Não crítico)

O TestDirector não resolve esta questão diretamente, porém podemos criar mecanismos que possibilitem esta funcionalidade. Uma forma seria criar um novo campo para o caso de teste com a informação se ele é ou não propenso à falhas. Durante a execução dos testes manuais caberia ao testador executar ou não o teste que houvesse popensão a falhas. No teste automatizado é necessária uma análise mais detalhada para a questão. Outra solução poderia ser sugerir criar suítes contendo os testes mais propensos a erros.

6.3.5 Execução Parcialmente Automatizada (IdReq: 5.22) (Crítico)

Deve ser permitido que parte do teste seja feita de forma automática e que outra não, por exemplo, a execução do teste poderia ser manual enquanto que a análise de resultados poderia ser automatizada. Uma solução que não nos agrada é dividir o caso de teste em dois: um manual e o outro automatizado. O TestDirector não permite chamar um mecanismo, por exemplo: a versão anterior do programa, que produza o resultado esperado para um teste que foi executado manualmente, pois para a ferramenta ou um caso de teste é manual ou é automatizado. O Driver de Teste parece ser a solução para esta questão, pois poderia para determinados procedimentos solicitar a intervenção humana enquanto que outros procedimentos seriam automatizados.

6.4 Análise de Falhas

Nesta seção apresentaremos as soluções dadas para o atendimento aos requisitos do gerenciamento de falhas.

Ambiente de aprendizado colaborativo e interativo (IdReq: 6.07) (Não crítico)

O TestDirector possui esta funcionalidade apenas para falhas dentro do mesmo projeto, seria importante que houvesse a possibilidade de consultar e aprender com as falhas resolvidas também por outros projetos. Uma solução poderia ser termos um projeto "fictício" que receberia periodicamente (diariamente) as falhas registradas nos diversos projetos. Temos de verificar a viabilidade desta solução.

Integração com o módulo de Medições (IdReq: 6.11) (Não crítico)

Espera-se que as medições não sejam efetuadas pela ferramenta que gerencia as falhas e sim pelo módulo de medições, porém no TestDirector ele mesmo gerencia as falhas e realiza as medições. Como a ferramenta já foi adquirida não tem sentido questionar a sua arquitetura. O que podemos é, além das medições que a ferramenta faz, exportar dados para serem trabalhados em outras ferramentas, o que é permitido pelo TestDirector. Acreditamos que isso não vale à pena, pois o TestDirector já satisfaz as nossas necessidades de métricas relacionadas a falhas.

Revisão de código para o mainframe (IdReq: 6.09) (Não crítico)

Está sendo construído na ferramenta de documentação de sistemas – Hexavision, um analisador estático de código mainframe para as linguagens Cobol e PLI. Trata-se de um utilitário para análise de programas novos e do legado que analisa programas batch e on-line, pesquisando a utilização de regras tais como: comandos que podem degradar o desempenho, rotinas antigas e proibidas, warnings passíveis de conduzir a erros de execução, utilização indevida das normas e boas práticas e apontar o uso indevido de constantes. Além disso, serão apresentados indicadores e métricas, como as linhas totais do código, a quantidade de condições de testes estimadas e a densidade de comentários. O utilitário também deve:

- Ser parametrizável, permitindo a inclusão de regras, comandos, rotinas e boas práticas que serão analisados para cada linguagem.
- Sugerir uma ação corretiva para cada problema detectado.
- Permitir a classificação dos problemas encontrados em níveis de criticidade: alto, médio e baixo.
- Fornecer subsídios para bloquear a compilação de um programa, ou tratá-lo como exceção.

6.5 Resumo do Capítulo

Neste capítulo apresentamos o relato de algumas soluções práticas para os requisitos de automação que não foram atendidos diretamente pelas ferramentas disponíveis conforme vimos no capítulo 5. Estas soluções tanto são simples como a parametrização da criação de um novo campo, como podem necessitar de um novo aplicativo ou até a abertura de um projeto que busque atender o desejado. Apresentamos também avaliações se as soluções implantadas corresponderam ou não ao que se esperava.

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

A necessidade de se criar sistemas com cada vez menos erros aumentou a importância das atividades relacionadas à qualidade, entre elas a atividade de teste. Além disso, temos de reduzir custos o que aumenta a pressão para que a atividade de teste seja feita de uma maneira mais produtiva. A automação do teste feita adequadamente é uma das maneiras de se conseguir um teste mais rápido e efetivo em encontrar erros.

Muito se tem escrito sobre automação de testes e um dos benefícios deste trabalho foi levantar e organizar diversos requisitos de automação. Este levantamento foi baseado na literatura e na metodologia de testes existente na empresa (MTS). A partir de uma arquitetura de automação proposta por Eickelmann ([EIC96]) criamos uma espécie de mapa geral de automação de testes. Esta organização além de trazer a separação de funcionalidades, permite a construção de componentes independentes e facilita o trabalho de gerenciamento do projeto de automação.

Acreditamos que os modelos propostos no workbench de testes são de grande valia, pois representam muitos dos conceitos relacionados ao assunto automação de testes e servem como base para as implantações necessárias. O modelo de dados dos artefatos de testes, por exemplo, será útil para o projeto de construção de um repositório para teste.

Um benefício foi que evitamos nos basear no que as ferramentas propõem sobre automação, passando a nos guiar pelo o que deveríamos efetivamente esperar que elas atendessem. Não foi o nosso caso, mas os requisitos de automação e os modelos apresentados no capítulo 4 podem ser úteis em um processo de avaliação para aquisição de ferramentas.

Para nós as ferramentas já estavam adquiridas e este trabalho serviu, conforme visto no capítulo 5, como um meio de avaliar o que as ferramentas atendem e buscar soluções para o que não são capazes de atender. Entre os pontos de não atendimento aos requisitos de automação podemos citar: a dificuldade de integração entre ferramentas de fornecedores diferentes; a falta de ferramenta para o planejamento de testes que façam estimativas e se integre com as demais

ferramentas; e a não consideração dos artefatos requisito de teste e critério de teste. Em relação ao planejamento de testes deveremos, por enquanto, permanecer com o template que já possuímos, enquanto que os demais templates (GRT, RT e RF) foram substituídos pelo uso das ferramentas. O requisito de teste ficará opcional devido a dificuldade que as ferramentas apresentam; o critério de teste será alvo de estudos futuros.

Consideramos que a avaliação das ferramentas foi satisfatória: cerca de 55% dos requisitos foram atendidos diretamente por elas, enquanto que 25% foram atendidos mediante alguma customização, restando 20% dos requisitos que não foram atendidos pelas ferramentas. Este trabalho auxiliou na implantação de várias ferramentas, tais como: Caliber, TestDirector, DB/IQ, SmartTest, File-Aid, WinRunner e LoadRunner. Outras ferramentas com o QuickTest, HiperStation e DevPartner estão em implantação.

Um ponto que merece destaque foi a implementação de um *framework* que gera *driver* para testar aplicações no *mainframe* que rodam em *batch.* Este *framework* está sendo bastante utilizado com resultados satisfatórios. Em relação a gerência de configuração dos artefatos de testes adotaremos uma solução em conjunto com a gerência de configuração dos demais artefatos do desenvolvimento de sistemas.

Estamos em andamento com o projeto e muito ainda falta para que a automação de testes tenha todas as suas funcionalidades implantadas e utilizadas. Porém, este trabalho contribuiu para o que já implantamos e dá uma maior visibilidade do que ainda temos a percorrer.

7.2 Trabalhos Futuros

No Modelo de Dados dos Artefatos é necessário colocar os atributos e identificar a chave primária. Também devemos verificar o relacionamento com outras entidades que não foram apresentadas no modelo, como é o caso da entidade componente. Posteriormente deve-se gerar o modelo físico considerando aspectos tais como volumes, desempenho e disponibilidade.

Deveremos detalhar o fluxo dos dados na Integração entre Ferramentas via Passagem de Dados, informando quais são os dados a serem passados entre as ferramentas e qual a direção do fluxo de dados. Teremos que analisar a necessidade de se utilizar outras ferramentas além das existentes. A empresa em que este trabalho foi realizado não pretende adquirir novas ferramentas, porém pode ser válido fazê-lo, principalmente em se tratando de ferramentas livres (“*freeware*”).

É necessário desenvolver um Framework dirigido a Palavras-Chaves para gerar driver para testar aplicações no mainframe que rodem on-line. A maior parte dos sistemas existentes está na plataforma mainframe, temos então que priorizar a construção de drivers para esta plataforma e uma das questões a ser considerada é se utilizaremos a ferramenta HiperStation ou a QuickTest. Também é necessário desenvolver um Framework dirigido a Palavras-Chaves para gerar driver para testar aplicações na Baixa Plataforma utilizando o QuickTest. Serão realizadas experiências utilizando o vocabulário existente na ferramenta e outras utilizando um vocabulário definido por nós. Quanto ao Driver Mainframe Batch temos a necessidade de implementar a integração com uma ferramenta para determinar a cobertura alcançada e possibilitar outras formas de análise de resultados como o acesso à base de dados, por exemplo.

Deve-se detalhar e implementar uma arquitetura de integração entre ferramentas, onde serão priorizadas as integrações que julgamos mais importantes, como a integração entre as ferramentas de gerenciamento de teste (TestDirector) e a de Capture/Replay para o mainframe (HiperStation). Hoje, a maior parte das integrações desejadas não é alcançada.

Deveremos aprimorar soluções para execuções parcialmente automatizadas, nos parece que o uso de drivers de teste é a solução mais adequada para esta necessidade. Para tornar possível um ambiente de aprendizado colaborativo e interativo para solucionar as falhas viabilizaremos o uso de um repositório contendo o histórico das falhas encontradas e solucionadas na corporação.

Temos também que evoluir no atendimento aos requisitos de automação e soluções para as áreas de Planejamento, Oráculos, Biblioteca de Scripts, Medições e Gerência de Configuração. Alguns dos requisitos destas áreas já foram avaliados e implementados enquanto que outros ainda possuem a análise e implementação pendentes.

Referências

[Asg05] ASG Software Solutions, “Smarttest”. http://www.asg.com/products/product_details.asp?code=STC. Acesso em 27/10/2005.

[Bac96] Bach, James, “Test Automation Snake Oil”. Windows Technical Journal, p. 40-44, out. 1996.

[Bei84] Beizer, B. Software system testing and quality assurance. New York: Van Nostrand Reinhold, 1984.

[Bei90] Beizer, Boris, “Software testing techniques”, segunda edição. New York, 1990.

[Bin00] Binder, Robert, “Testing Object–Oriented Systems: Models, Patterns, and Tools”. Addisson Wesley, 2000.

[Bor05a] Borland, “CaliberRM”. http://www.borland.com.br/caliber/pdf/crm51_datasheet.pdf. Acesso em 25/10/2005.

[Bor05b] Borland, “Optimizeit Enterprise Suite”. http://www.borland.com.br/optimizeit/index.html. Acesso em 28/10/2005.

[Cha00] Chays, David, Dan, Saikat e Frankl, Phyllis, “A Framework for Testing Database Applications”. Proceedings of the Intl. Symposium on Software Testing and Analysis, Portland, Oregon, USA, p.147-157, ago. 2000.

[Com05a] Compuware, “File-AID/MVS”. http://www.compuware.com/products/fileaid/mvs.htm. Acesso em 22/10/2005.

[Com05b] Compuware, “File-AID/Data Solutions”. http://www.compuware.com/products/fileaid/datasolutions.htm. Acesso em 22/10/2005.

[Com05c] Compuware, “File-AID for DB2”. http://www.compuware.com/products/fileaid/db2.htm. Acesso em 07/08/2005.

[Com05d] Compuware, “File-AID/RDX”. http://www.compuware.com/products/fileaid/rdx.htm. Acesso em 27/10/2005.

[Com05e] Compuware, “QAHiperstation”. http://www.compuware.com/products/qacenter/qahiperstation.htm. Acesso em 17/09/2005.

[Com05f]) Compuware, “Strobe”. http://www.compuware.com/products/strobe/default.htm. Acesso em 02/10/2005.

[Com05g] Compuware, “DevPartner”. http://www.compuware.com/products/devpartner/default.htm. Acesso em 04/09/2005.

[Com05h] Compuware, “File-AID/CS”. http://www.compuware.com/products/fileaid/cs.htm. Acesso em 19/10/2005.

[Cun04] Cunha, Adriano, “AutoTest: um sistema para testes funcionais e distribuídos de aplicações em ambiente hipermídia”, Campinas, Dezembro, 2004.

[Dav00] Davies, R.A., Beynon, R.J.A., Jones, B.F., “Automating the Testing of Databases”. Proc. of the First Intl. Workshop on Automated Program Analysis, Testing and Verification, Limerick, Ireland, June, 2000.

[Eic96] Eickelmann, Nancy, “An Evaluation of Software Test Environment Architectures”. Proceeding of the Eighteenth International Conference of Software Engineering, Berlin, Germany, p. 353-364, Mar, 1996.

[Few01] Fewster, Mark, “Commom Mistakes in Test Automation”. Fall Test Automation Conference, Boston, MA, Ago, 2001.

[Fro04] Fross, Margaret, “Selecting a Software Process Management Tool”. ProtoTest LLC, Abr., 2004.

[Het87] Hetzel, William, “Guia Completo ao Teste de Software”. Campus, 1987.

[Iee83] IEEE-AS Standards Board, “IEEE Standard for Software Test Documentation”. ANSI/IEEEE Std 829-1983. (IEEE Standard Collection Software Engineering).

[Iee90] IEEE-AS Standards Board, “IEEE Std 610.12-1990 - IEEE glossary of software engineering terminology”. Software Engineering Technical Committee of the IEEE Computer Society, 1990.

[Kan97] Kaner, Cem, “Improving the Maintainability of Automated Test Suites". Proceedings of the Tenth International Quality Week, San Francisco, CA, Mai, 1997.

[Kan98] Kaner, Cem, “Avoiding Shelfware: A Managers´ View of Automated GUI Testing”. STAR East, Orlando, FL, Mai, 1998.

[Kan00] Kaner, Cem, “Architectures of Test Automation”. *Software Testing, Analysis & Review Conference (Star) West*, San Jose, CA, Out, 2000.

[Mer05a] Mercury Interactive, “TestDirector”. http://www.mercury.com/us/products/quality-center/testdirector/. Acesso em 14/08/2005.

[Mer05b] Mercury Interactive, “QuickTest Professional”. http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/. Acesso em 24/09/2005.

[Mer05c] Mercury Interactive, “WinRunner”. http://www.mercury.com/us/products/quality-center/functional-testing/winrunner/. Acesso em 25/08/2005.

[Mer05d] Mercury Interactive, “LoadRunner”. http://www.mercury.com/us/products/performance-center/loadrunner/. Acesso em 22/10/2005.

[Met04] “Metodologia de Projeto, Sistemas e Serviços de TI”. Banco Real Abn Amro, Brasil, 2004.

[Mye79] Myers, Glenford, “The Art of Software Testing”. John Wiley & Sons, 1979.

[Nag00] Nagle, Carl, “Test Automation Frameworks”. Software Automation Framework Support, 2000.

[Pet01a] Pettichord, Bret, “Success with Test Automation Success”. STAR West, San Francisco, Mai, 1996, Version of 2001.

[Pet01b] Pettichord, Bret, “Seven Steps to Test Automation Success”. STAR West, San Jose, Nov, 1999, Version of 2001.

[Pre95] Pressman, Roger, “Engenharia de Software”. Makron Books do Brasil Editora Ltda, 1995.

[Ric94] Richardson, Debra, “TAOS: Testing with Analysis and Oracle Support”. In Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis, Washington, p. 138-153, Ago, 1994.

[Roc01] Rocha, Ana, Maldonado, José e Weber, Kival, “Qualidade de software – teoria e prática”. Prentice Hall, 2001.

[Rsi05] RSI, “DB/IQ QA”. http://www.rsinet.com.br/Modelos/Modelo1.asp?item=31. Acesso em 11/11/2005.

[Shu96] Shukla, Shilpa, Redmiles, David, “Collaborative Learning in a Software Bug-Tracking Scenario”. CSCW 96 Workshop on Approaches for Distributed Learning Through Computer Supported Collaborative Learning, Boston, Massachusetts, Nov, 1996.

[Som03] Sommerville, Ian, “Engenharia de Software”. Addison Wesley, 2003.

[Tre05] Treehouse Software, Inc., “Trim”. http://www.treehouse.com/trim.shtml. Acesso em 08/11/2005.

[Vog93] Vogel, Peter, “An Integrated General Purpose Automated Test Environment”. In Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis, Cambridge, Massachusetts, p.61-69, Jun, 1993.

[Was90] Wasserman, Anthony, “Tool Integration in Software Engineering Environments”. In Software Engineering Environments, Fred Long, ed. Springer-Verlag, 1990.