UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

PLUGIN DA FERRAMENTA TESTCOMPLETE PARA INTEGRAÇÃO COM A FERRAMENTA TESTLINK

DOUGLAS DE OLIVEIRA WALTRICK

BLUMENAU
2011

2011/2-10

DOUGLAS DE OLIVEIRA WALTRICK

PLUGIN DA FERRAMENTA TESTCOMPLETE PARA INTEGRAÇÃO COM A FERRAMENTA TESTLINK

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação— Bacharelado.

Prof. Everaldo Artur Grahl, Mestre - Orientador

BLUMENAU
2011

2011/2-10

PLUGIN DA FERRAMENTA TESTCOMPLETE PARA INTEGRAÇÃO COM A FERRAMENTA TESTLINK

Por

DOUGLAS DE OLIVEIRA WALTRICK

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Jacques Robert Heckmann, Mestre – FURB

Blumenau, 07 de dezembro de 2011.

Dedico este trabalho aos meus pais que no esforço e determinação puderam me apoiar, incentivar e proporcionar a realização desta graduação.

AGRADECIMENTOS

Aos meus pais, André e Jocenira, que sempre me apoiaram e estiveram presentes nos meus estudos e que estão muito felizes e orgulhosos com essa conquista.

Ao meu orientador, Everaldo, por ter acreditado na proposta e na minha capacidade de concluir este trabalho.

Aos meus amigos, em especial a Ana Karina Lunardi que sempre esteve ao meu lado nessa jornada e muito me ajudou.

A Edusoft Tecnologia Ltda. e seus funcionários que me apoiaram e me proporcionaram subsídios para conclusão desta graduação com êxito.

Ao colegiado do curso de Sistemas de Informação da FURB, pelo aprendizado adquirido durante a minha graduação.

O único lugar onde o sucesso vem antes do trabalho, é no dicionário!

Albert Einstein

RESUMO

Este trabalho apresenta um *plugin* desenvolvido no *framework* TestComplete que integra-se com a ferramenta CASE TestLink. Basicamente o *plugin* reduz o esforço despreendido em manter as duas ferramentas síncronas. Para o desenvolvimento do sistema foram utilizadas a linguagem DelphiScript, com o *framework* TestComplete, e banco de dados MySQL. Sabe-se que no processo de desenvolvimento de um software as atividades relacionadas a controle de qualidade, em específico os testes, normalmente trabalham com recursos e cronogramas muito reduzidos. O objetivo deste trabalho foi eliminar as atividades operacionais de ambas as ferramentas, no tocante o registro de informações redundantes. A utilização do *plugin* tornou a atividade de codificação de *scripts* do analista de teste mais produtiva e os resultados referentes a execução dos testes automatizados são obtidos de forma rápida e simples.

Palavras-chave: Teste de software. Ferramenta CASE. TestComplete. TestLink.

ABSTRACT

This works presents a framework developed in TestComplete plugin that integrates with the TestLink CASE tool. Basically the plugin reduces the effort to keep the two detached synchronous tools. For the development of the system were used language DelphiScript, TestComplete with the framework, and MySQL database. It's known that in the software development process activities related to quality control, particular tests, usually work with resources and schedules very low. The objective of this study was to eliminate the operational activities of both tools regarding the registration of redundant information. Using the plugin has the activity of coding scripts test analyst and the most productive results for the execution of automated tests are obtained quickly and easily.

Key-words: Software test. CASE tool. TestComplete. TestLink.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

CT – Caso de Teste

DDT - *Data-Driven Testing*

DER – Diagrama Entidade-Relacionamento

EA - *Enterprise Architect*

RF – Requisito Funcional

RNF – Requisito Não Funcional

SUMÁRIO

1 INTRODUÇÃO

Desenvolver um software de qualidade não é mais um requinte para poucos, transformou-se num fator de competitividade num mercado cada vez mais exigente. As empresas mais competitivas são as que trabalham sob a ótica de melhoria contínua dos seus processos para aumentar a qualidade do processo de desenvolvimento e, conseqüentemente, aumentar a qualidade do produto final. Neste contexto, deve-se destacar a adoção crescente de ferramentas para dar suporte ao processo de melhoria contínua. Estas ferramentas servem para dar suporte a todas as atividades relacionadas ao ciclo de vida de desenvolvimento de software: da concepção à implantação.

O relatório The Economic Impact of Inadequade Infraestruture for Software Testing estima que o custo total dos softwares com defeitos para organizações nos EUA corresponde, aproximadamente, a um valor um pouco abaixo de 1% do Produto Interno Bruto (PIB) (NIST, 2002). Este mesmo relatório atribui como conseqüências de um processo de teste falho ou inexistente:

a) aumento dos custos de desenvolvimento;
b) aumento do “time to market” para novas funcionalidades;
c) aumento do custo de suporte aos produtos de mercado.

Conforme Rios e Moreira Filho (2006, p. 11), “quanto mais tarde um defeito for identificado mais caro fica para corrigi-lo e mais ainda, os custos de descobrir e corrigir defeitos no software aumentam exponencialmente na proporção que o trabalho evolui através das fases do projeto de desenvolvimento.”

A ineficiência dos testes é indicada por Jones (1995) em Patterns of Software System Failure and Sucess como um dos principais motivos de falha nos projetos de desenvolvimento de software. Segundo o Gartner Group (2001), apenas uma parcela pequena das aplicações entregues (7%) atendem aos requisitos de tempo de resposta e performance.

Com a evolução das aplicações, onde estas a cada dia tornam-se cada vez maiores e complexas, ter um processo de testes implementado em uma empresa desenvolvedora de software não é mais uma oportunidade de negócio, e sim um item vital para garantir a sobrevida de um software no mercado.

Existem dois grandes grupos, nos quais pode-se subdividir todas as atividades de um processo de teste. O primeiro é a Garantia de Qualidade ou Qualidade do Processo, onde tem-se como foco trabalhar na prevenção de falhas. Está ligada diretamente ao processo de

desenvolvimento. Outro grande grupo, o Controle de Qualidade ou Qualidade do Produto, tem como objetivo assegurar que os produtos gerados pelo desenvolvimento estão de acordo com os padrões estabelecidos previamente (ENGHOLM JÚNIOR, 2010). Uma dessas sub-áreas do controle de qualidade é a Automação de Testes, que será um dos temas abordados por este trabalho. Muitas organizações investem avidamente seus recursos na atividade de automação, mesmo não possuindo um grau de maturidade suficiente para comportá-la. Como conseqüência não se consegue obter os resultados almejados e a automatização cai em descrédito (MOLINARI, 2003).

Outro fator que acaba prejudicando a automatização, é a falta de uma ferramenta centralizada que incorpore todas as atividades de testes. Ou seja, tem-se os casos de testes mantidos em uma ferramenta e os *scripts* de automatização em outra, e ambas trabalham isoladamente. Isto ocorre com as duas ferramentas de testes muito usadas no mercado: TestLink e TestComplete.

Tem-se como finalidade neste trabalho, acabar com a lacuna existente entre as ferramentas de Gerência de Testes (TestLink) e Automatização de Testes (TestComplete), integrando-as e reduzindo o esforço operacional para mantê-las consistentes e atualizadas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um *plugin* para a ferramenta TestComplete que permita a integração com os casos de testes gerenciados na ferramenta TestLink.

Os objetivos específicos do trabalho são:

a) permitir uma melhor manipulação dos dados (DDT[1]) utilizados para *input* nos testes;
b) facilitar a execução dos casos de testes;
c) manter histórico de execuções dos casos de testes nas duas ferramentas.

[1] DDT (*Data-Driven Testing*) é uma técnica utilizada na automatização de testes, onde os dados de entradas do teste (*input)* não estão explicitados no código fonte do script de teste. Normalmente é utilizado uma fonte de dados alternativa, como uma planilha Excel (AUTOMATEDQA CORPORATION, 2010).

1.2 ESTRUTURA DO TRABALHO

Este trabalho está disposto em quatro capítulos. No primeiro capítulo apresenta-se a introdução, os objetivos e a estrutura do trabalho.

No segundo capítulo tem-se a fundamentação teórica, destacando-se os conceitos relacionados ao controle de qualidade, tais como dimensões do teste, o processo de testes de um *software* e ferramentas de teste, e na última seção apresenta-se os trabalhos correlatos.

No terceiro capítulo é apresentado o desenvolvimento do *plugin*, incluindo detalhes sobre a especificação, implementação e tecnologia utilizada.

No quarto capítulo apresenta-se a conclusão e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os aspectos teóricos sobre o trabalho, tais como dimensões do teste, o processo de testes de um *software* e ferramentas de teste. A última seção deste capítulo mapeia alguns trabalhos correlatos que possuem características semelhantes.

2.1 TESTE DE SOFTWARE

Segundo Rios e Moreira Filho (2006) estima-se que nas décadas de 1960 e 1970, os desenvolvedores dedicavam 80% dos seus esforços nas atividades de codificação e nos testes unitários. Uma parcela menor de esforço era dedicada à integração dos programas e aos testes dos sistemas. Porém as atividades de testes eram consideradas apenas um mal necessário para provar aos usuários que os produtos funcionavam e não eram tratadas como um processo formal e alinhado com as atividades do processo de desenvolvimento de sistemas.

A partir dos anos oitenta, durante o processo de desenvolvimento, passou a ser dada maior importância à análise dos requisitos, ao desenho funcional e técnico dos novos sistemas. Um esforço maior passou a ser dedicado à integração das diversas peças que compunham os softwares e ao teste destes para funcionarem como um sistema. As atividades passaram a ser tratadas como um processo formal, aparecendo as metodologias de testes que evoluíram até os dias de hoje.

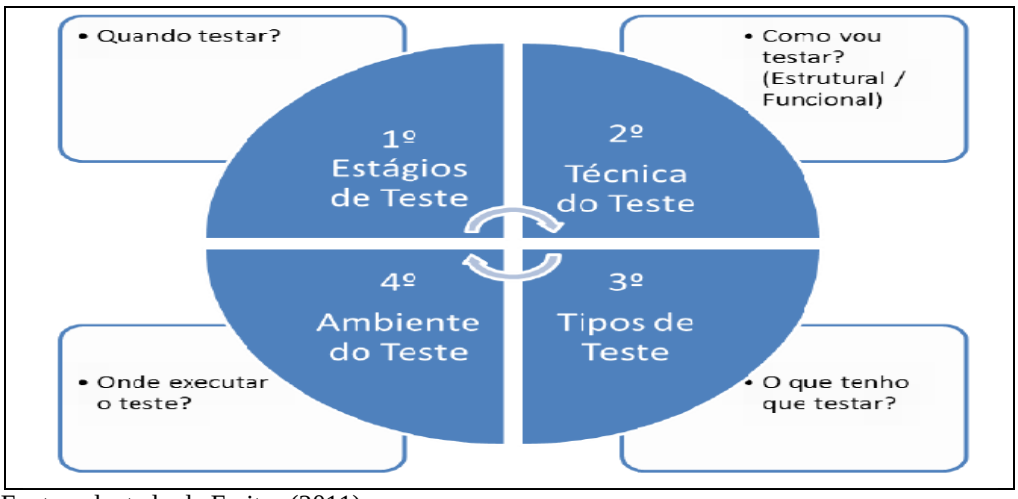
Existem várias definições de teste, entre as quais encontram-se relacionadas algumas:

a) testar é verificar se o software está fazendo o que deveria fazer, de acordo com os seus requisitos, e não está fazendo o que não deveria fazer (RIOS; MOREIRA FILHO, 2006);
b) testar é o processo de executar um programa ou sistema com a intenção de encontrar defeitos (teste negativo) (MYERS, 1979);
c) testar é qualquer atividade que a partir da avaliação de um atributo ou capacidade de um programa ou sistema, seja possível determinar se ele alcança os resultados desejados (HETZEL, 1987).

Muitas outras definições poderiam ser citadas, mas essencialmente, teste de software é o processo que visa executar o software de forma controlada, com o objetivo de avaliar o seu comportamento, baseado no que foi especificado. A execução dos testes é considerada um tipo de validação.

2.2 DIMENSÕES DO TESTE

Os testes devem ser planejados de forma organizada, a fim de proporcionar um teste mais preciso. É importante que os testes sejam bem planejados e desenhados, para conseguir o melhor proveito possível dos recursos alocados para eles. Neste planejamento é fundamental preocupar-se com as dimensões do teste que são apresentadas na Figura 1.



Fonte: adaptado de Freitas (2011).

Figura 1 - Dimensões do teste

Nas subseções seguintes serão explanadas todas as dimensões apresentadas na Figura 1 de acordo com Freitas (2011).

2.2.1 Primeira dimensão: Estágios de Teste

A primeira dimensão trata do momento em que haverá o teste e o responsável por sua execução (FREITAS, 2011). Seguem alguns exemplos:

a) teste de unidade**:** é a escala mais baixa de teste. O responsável é o programador;
b) teste de integração**:** é a integração das unidades testadas para verificar se executam suas funções corretamente quando colocadas juntas, isto é, quando integradas. Normalmente executado pelo analista de sistemas;
c) teste de sistema**:** é a execução do sistema como um todo para validar a exatidão e perfeição na execução de suas funções. Normalmente executado pelo analista de teste;
d) teste de aceitação**:** é a execução dos testes finais do sistema. Normalmente executado pelos usuários.

Nesta dimensão, destaca-se o teste de sistema, onde a execução dos testes dependerá do nível de maturidade e infra-estrutura disponível para a realização dos mesmos, podendo ser executada de forma manual ou automatizada.

2.2.2 Segunda dimensão: Técnica do Teste

A segunda dimensão trata de fazer uma análise da aplicação e dizer qual técnica é mais apropriado para ser seguida:

a) técnica de teste estrutural (caixa branca): é o tipo de teste onde a estrutura interna do software é analisada (FREITAS, 2011). O objetivo do teste estrutural é identificar defeitos nas estruturas internas dos programas através da simulação de situações que exercitem adequadamente todas as estruturas utilizadas na codificação, como, desvios condicionais, laços de processamento e caminhos alternativos de execução;
b) técnica de teste funcional (caixa preta): é o tipo de teste onde o programa é considerado uma entidade fechada e a sua estrutura interna não é considerada. Apenas as especificações do programa e os requisitos do software são considerados no momento da execução dos testes. Esse tipo de teste reflete, de certa forma, a ótica do usuário, que está interessado em se servir do programa sem considerar os detalhes de sua construção (FREITAS, 2011). Comparado a outros tipos de teste, este é relativamente mais simples. Um exemplo de aplicação é verificar a consistência de dados de interface.

De acordo com a técnica selecionada, deve-se escolher os tipos de teste a serem realizados que serão vistos na terceira dimensão.

2.2.3 Terceira dimensão: Tipos de Teste

A terceira dimensão trata de fazer uma análise do projeto e informar quais tipos de testes serão executados (FREITAS, 2011). É uma das mais fortes das três dimensões, mais conhecida como “tipo de teste”. Como principais tipos de teste tem-se:

a) testes de regressão: implica em executar novamente um conjunto de testes já conduzidos anteriormente para garantir que as mudanças realizadas não produziram efeitos colaterais indesejados (PRESSMAN, 2006). Um conjunto de dados e *scripts* deve ser mantido como *baseline*[2] e executado para verificar que mudanças introduzidas posteriormente não danificarão códigos já considerados bons e aceitos. Os resultados esperados a partir da *baseline* devem ser comparados aos resultados após as mudanças. As discrepâncias devem ser resolvidas antes de atingir o próximo nível de testes;
b) testes de carga: visam avaliar a resposta de um software sob uma pesada carga de dados ou uma grande quantidade de usuários simultâneos para verificar o nível de escalabilidade, ou seja, o momento onde o tempo de resposta começa a degradar ou a aplicação começa a falhar;
c) testes de usabilidade: verificam o nível de facilidade de uso do software pelos usuários. Deve ser efetuado principalmente em aplicações web, onde existe muita navegação entre páginas. As telas de ajuda devem ser avaliadas quanto ao seu conteúdo e clareza de linguagem, assim como as mensagens de erro. Testa-se também os *links* para outras páginas;
d) testes funcionais: são executados através de técnicas de validação, e são realizados para assegurar que as especificações e os requisitos do *software* foram atendidos. Esses testes são caracterizados pelo foco no negócio, garantindo que não existam diferenças entre especificações/requisitos e o comportamento do sistema.

[2] Uma *baseline* é um conjunto de especificações ou produtos de trabalho que foram formalmente revisados e sobre os quais foi feito um acordo, que serve como base para desenvolvimento posterior e que pode ser modificado somente através dos procedimentos de controle de mudanças.

O grande problema dos testes de regressão é o gasto excessivo de tempo e a repetitiva execução dos procedimentos de teste, se realizado de formal manual. Caso a organização tenha maturidade e domínio na automatização dos testes, os mesmos devem ser automatizados, para que posteriormente com a integração de novos componentes ou novas funcionalidades os testes possam ser re-executados, validando assim a integridade do sistema.

2.2.4 Quarta dimensão: Ambiente do Teste

Nessa dimensão deve-se definir o ambiente onde as outras três dimensões irão rodar, observando as evoluções tecnológicas e fazendo um bom planejamento da fase de teste (FREITAS, 2011). Os testes utilizados são:

a) teste de aplicações *mainframe*;
b) teste de aplicações *client*;
c) teste de aplicações *server*;
d) teste de aplicações *network*;
e) teste de aplicações *web*.

O ambiente é tudo que está “em volta do teste” no mundo real. Um ambiente de teste é composto por dois grandes grupos:

a) Elementos de apoio:
- espaço físico para o trabalho de teste;
- equipamentos de apoio ao teste;
- ferramentas de apoio, como ferramentas de monitoração;
- tecnologias de apoio;
- material de apoio, treinamento.

b) Elementos centrais de teste em si:
- equipamento disponível para teste;
- software e hardware disponíveis para teste;
- recursos humanos disponíveis para interagir ou realizar o teste.

A primeira ação que deve ser tomada é verificar se o ambiente de teste está criado e estável para rodar as demais dimensões. O ambiente deve estar de acordo com o que foi definido no plano de teste e este deve estar compatível com o ambiente de produção.

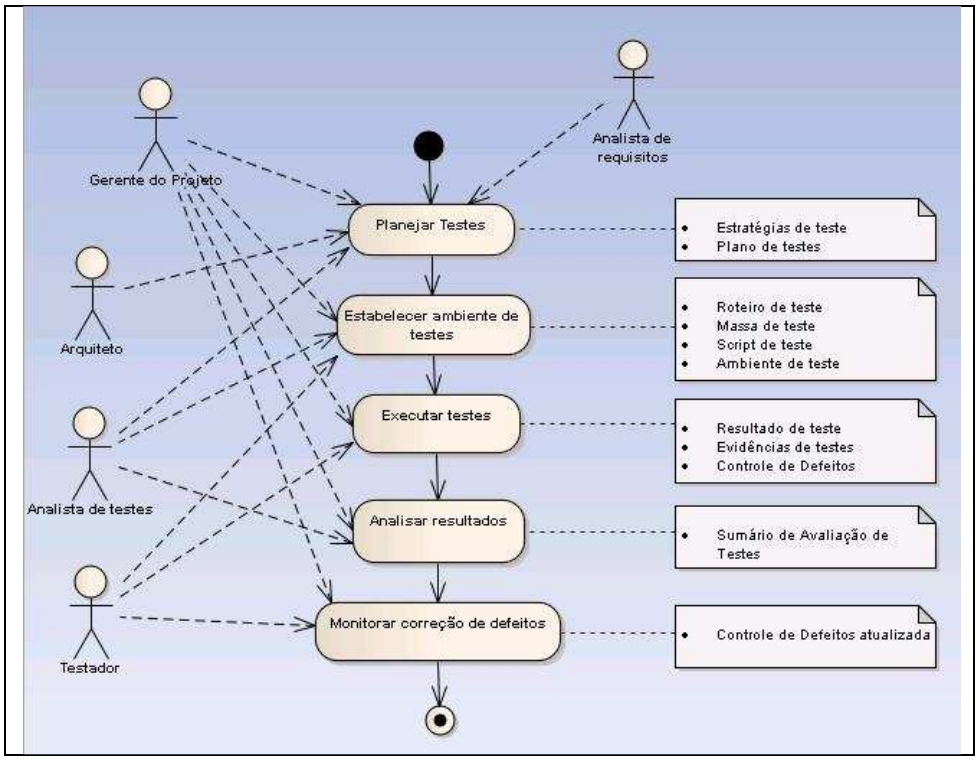
2.3 PROCESSO DE TESTES

O processo de testes de software representa uma estruturação de etapas, atividades, artefatos, papéis e responsabilidades que buscam a padronização dos trabalhos e ampliar a organização e controle dos projetos de testes. Ele deve basear-se em uma metodologia aderente ao processo de desenvolvimento, pessoal técnico qualificado, ambiente e ferramentas adequadas (RIOS; MOREIRA FILHO, 2006).

Segundo Moreira Filho e Rios (2003), a metodologia de testes deve ser o documento básico para organizar a atividade de testar aplicações no contexto da empresa. Como é indesejável o desenvolvimento de sistemas sem uma metodologia adequada, também acontece o mesmo com a atividade de testes.

O objetivo central de toda metodologia de teste é melhorar o processo de teste, incluindo técnicas a serem utilizados, procedimentos adotados e ferramentas. Toda metodologia procura implantar um processo de teste envolvendo um conjunto de atividades que vai desde, o levantamento da necessidade da empresa até um completo ciclo de implantação das atividades de teste dentro da empresa (FREITAS, 2011).

Uma metodologia de teste procura definir um processo genérico de teste que prevê a realização das atividades de planejamento, projeto, execução e acompanhamento dos testes de unidade, integração, sistemas e aceitação. A partir deste processo genérico cada empresa deve instanciar um processo específico que melhor atenda as suas necessidades. A Figura 2 sugere um processo de teste descrevendo os sub-processos, o seqüenciamento dos mesmos, as saídas de cada sub-processo e a matriz de responsabilidades (FREITAS, 2011).



Fonte: adaptado de Engholm Júnior (2010).

Figura 2 - Processo de Teste

Pode-se observar na Figura 2, o seqüenciamento dos sub-processos através das atividades de planejamento, definição de ambiente, execução dos testes, análise dos resultados e monitoramento. Para cada atividade é gerado um ou mais artefatos, e estes estão sob responsabilidade dos envolvidos no processo.

A melhoria de processo de teste deve ser uma atividade contínua e permanente. Como testar um sistema ou software é uma atividade complexa, sempre existirão itens ou atividades a serem melhorados (RIOS; MOREIRA FILHO, 2006).

Diante deste contexto, deve-se destacar a adoção crescente de ferramentas para dar suporte ao processo de melhoria contínua do processo e no auxílio à gestão do processo de teste.

2.4 FERRAMENTAS

Como apresentado no tópico anterior, a adoção de ferramentas para auxiliar a melhoria contínua do processo de testes se tornou indispensável. Sendo assim, neste tópico serão apresentadas duas ferramentas que se encaixam neste perfil, o Testlink que é uma ferramenta

para Gestão do Projeto de Testes, desde o planejamento à análise dos resultados dos testes, e o TestComplete que é uma ferramenta para automatização de testes.

2.4.1 Testlink

O TestLink é uma ferramenta *open source* para o gerenciamento de testes. Ela permite os cadastros de planos e casos de testes e o controle e execução dos testes. Com o TestLink é possível que equipes de testes trabalhem de forma sincronizada mesmo em locais diferentes. Por ter uma interface *Web* e permitir níveis de acesso diferenciados, analistas de testes podem gerar as especificações de testes que outras equipes poderão executar. Outra característica interessante é o controle de execuções, gerando uma base histórica dos testes aos quais a aplicação foi submetida (CAETANO, 2007).

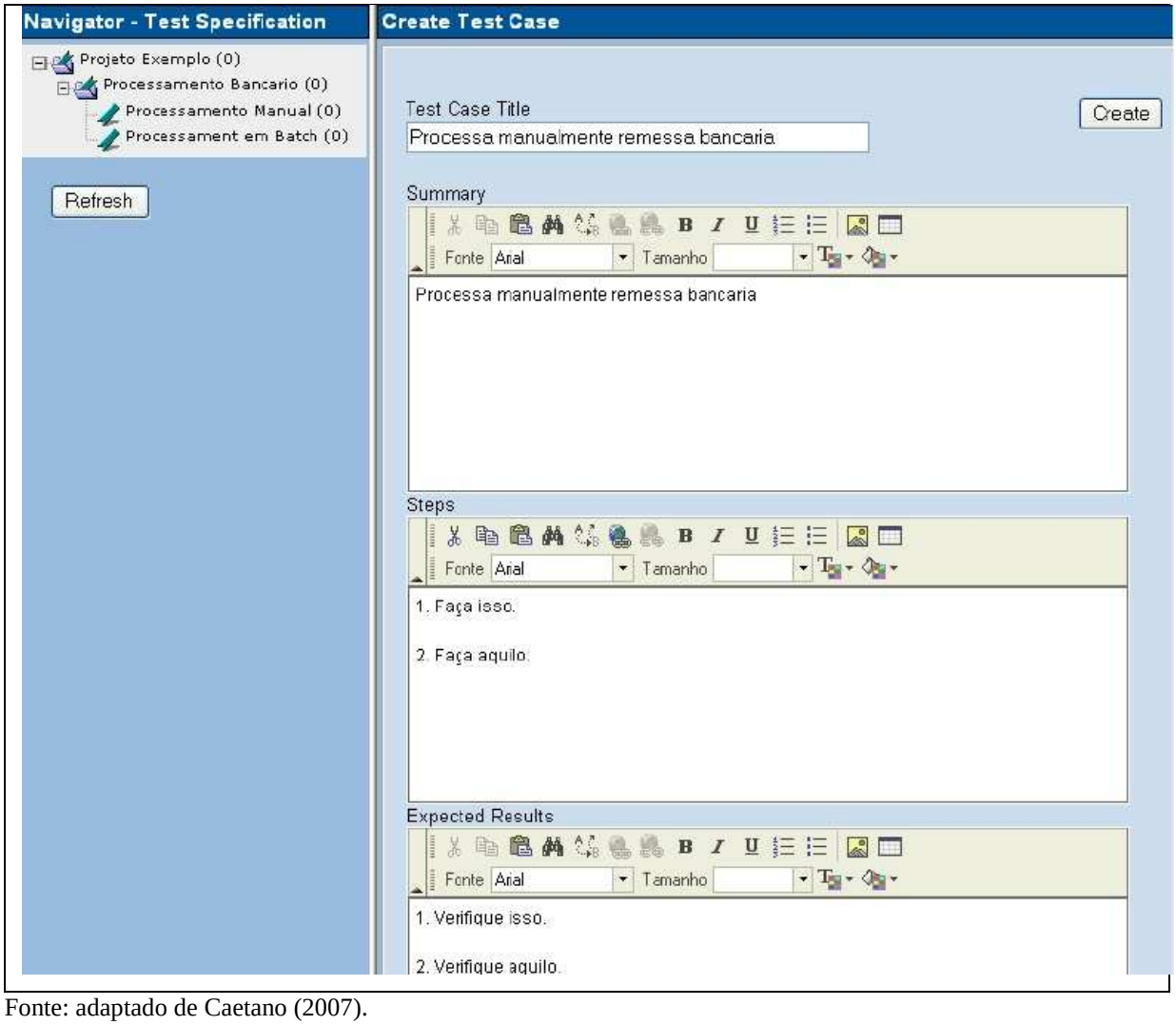
Para entender de que forma o Testlink pode auxiliar na gestão de projeto de teste, abaixo são apresentadas algumas das características básicas desta ferramenta, que tem como pilares centrais o Projeto de Teste, o Plano de Teste e o Usuário (CAETANO, 2007):

a) projeto de teste: é a unidade de organização básica do TestLink, nele há a documentação;
b) especificação do teste: é uma estrutura de organização do projeto que contém as Suítes de Teste e os Casos de Testes;
c) suíte de teste: é uma estrutura para a organização de Casos de testes ou outras Suítes de teste. Geralmente usada para separar os Casos de testes em grupos;
d) caso de teste: é um conjunto de entradas, condições estabelecidas e resultados esperados que atendem a um determinado objetivo. A Figura 3 exemplifica a criação de caso de teste na ferramenta;
e) plano de teste: é um plano que descreve detalhadamente o ambiente de teste, as técnicas e ferramentas que serão usadas e um objetivo a ser alcançado. A descrição do plano de teste deve conter as funcionalidades a serem testadas, as funcionalidades que não serão testadas, o critério para que uma funcionalidade receba o status “Passou”, informações sobre o ambiente de testes como ferramentas utilizadas e plataforma onde os testes serão executados;

f) *build*: são versões (*releases*) específicas do software a ser testado. Para execução dos casos de testes é necessário uma *build*, pois um caso de teste deve ser executado em uma versão do software.

De acordo com as características apresentadas deve-se perceber que esta ferramenta é uma grande aliada na gestão dos projetos de teste e melhoria contínua do processo de teste. A Figura 3 apresenta os atributos, dados do caso de teste que é mantido pelo TestLink.

A Figura 3 demonstra a criação de um caso de teste na ferramenta TestLink. Inicialmente é definido um título (*Test Case Title*) para o caso de teste (título pertinente ao teste a ser realizado), é informada uma breve descrição do propósito (*Summary*) do caso de teste, os passos (*Steps*) para simulação do teste em si e por final o resultado esperado (*Expected Results*) do caso de teste.



Fonte: adaptado de Caetano (2007).

Figura 3 - Criação de um Caso de Teste na ferramenta TestLink.

2.4.2 TestComplete

Apesar do teste manual de software permitir encontrar vários erros em uma aplicação, é um trabalho maçante e que consome bastante tempo. Também pode não ser efetivo na procura de classes específicas de defeitos (ex: funcionais, interface). A automação é o processo de escrita de um programa de computador para realizar o teste. Uma vez automatizado, um grande número de casos de teste pode ser validado rapidamente. As vantagens da automação tornam-se mais evidentes para os casos dos produtos que possuem longa vida no mercado, porque mesmo pequenas correções no código da aplicação podem causar a quebra de funcionalidades que antes funcionavam (MOREIRA FILHO; RIOS, 2003).

A automação envolve testes de caixa-preta, em que o desenvolvedor não possui conhecimento sobre a estrutura interna do sistema, ou caixa-branca, em que há pleno conhecimento da estrutura interna. Para os dois casos, a cobertura de teste é determinada respectivamente pela experiência do desenvolvedor ou pela métrica de cobertura de código.

A automação de teste pode ser cara, e geralmente é usada em conjunto com técnicas manuais. Entretanto, pode cortar custos a longo prazo, especialmente na fase de teste de regressão (MOLINARI, 2003). Neste tópico será apresentada a automatização de testes de caixa preta na ferramenta TestComplete.

Segundo a AutomatedQA Corporation (2010), a ferramenta TestComplete é um ambiente utilizado para automatização de testes de aplicações Windows e .NET. Fornece apoio para testar projetos desenvolvidos em Microsoft Visual C++, Microsoft Visual Basic, Delphi, C++Builder, C# e Java, bem como aplicações web.

TestComplete fornece várias funcionalidades, entre elas o Test Log, onde podem ser verificados os resultados da execução dos testes, conforme mostrado na Figura 4.

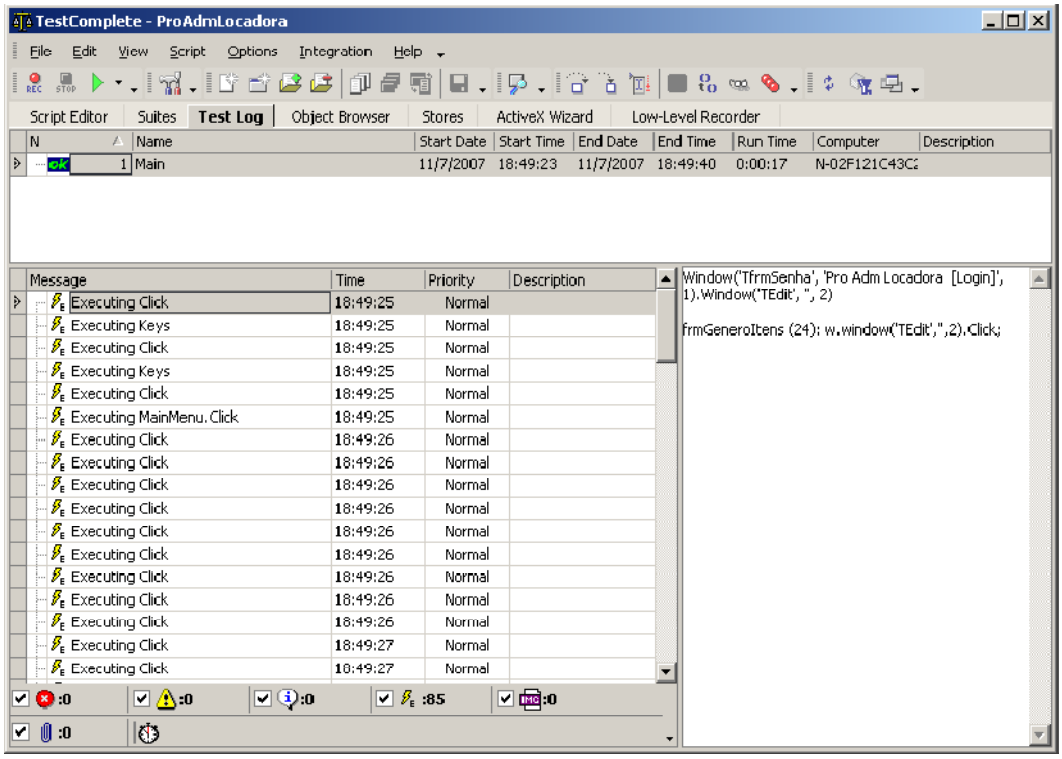


Figura 4 - Test Log funcionalidade TestComplete

Na Figura 4 podemos observar o resultado da execução do TestComplete, para cada ação realizada uma linha é mantida no *log* e este registro é realizado automaticamente pelo TestComplete. Dentre as ações mantidas pelo *log* estão os erros, avisos, caixa de mensagem, teclas pressionadas, cliques, etc.

Além disso, para elaborar os *scripts* de teste deve-se usar uma das cinco linguagens (C#Script, C++Script, DelphiScript, JScript e VBScript) de *scripts* suportadas pela ferramenta (AUTOMATEDQA CORPORATION, 2010).

Assim, aqueles que dominam a linguagem Object Pascal podem optar por desenvolver os *scripts* em DelphiScript, um subconjunto de Object Pascal. Esta funcionalidade visa auxiliar no aprendizado da linguagem de *script* e, conseqüentemente, no uso da ferramenta.

Dessa forma, para testar, por exemplo, a inclusão de dados em uma agenda telefônica desenvolvida em Delphi (Figura 5), através da ferramenta TestComplete, pode-se escrever o código de teste apresentado no Quadro 1.

Figura 5 - Interface de uma agenda telefônica

```
Procedure Inclui_Dados (TelaRef : OleVariant);
varTelaPar : OleVariant;
begin
TelaPar := TelaRef.VCLObject('AgendaForm'); // identificação do formulário
TelaPar.VCLObject('ComboNome').Keys('Adriana Fronza Marcos');
// preenche nome a ser incluído
TelaPar.VCLObject('EditTelRes').Keys('33260507'); // preenche telefone residencial
TelaPar.VCLObject('EditTelCom').Keys('32218888'); // preenche telefone comercial
TelaPar.VCLObject('EditTelCel').Keys('91673213'); // preenche telefone celular
TelaPar.VCLObject('BitBtnSalvar').Keys('[Enter]'); // clica no botão salvar
End;
```

Quadro 1 - Código para a ferramenta TestComplete para testar a inclusão de dados

Observa-se que os componentes do formulário da Agenda são referenciados no TestComplete utilizando o método “VCLObject” passando como parâmetro seu nome ou *caption* utilizado na aplicação Delphi. A entrada de dados (*input*) é simulada através do método “Keys” onde seu parâmetro é o dado a ser inserido para o teste.

Baseando-se nas características da ferramenta TesteComplete apresentadas pode-se afirmar que a mesma é uma forte aliada nos testes funcionais de software, e principalmente nos testes de regressão, haja visto que a realização dos testes é quase sempre limitada por restrições de cronograma e orçamento. Os testes de regressão muitas vezes deixam de ser realizados por algum ou ambos os fatores.

2.5 *PLUGIN* NO TESTCOMPLETE

Segundo Prada (2008) na informática, toda ferramenta, programa ou extensão que se encaixa a outro programa principal para adicionar mais funções e recursos a ele, pode ser definido como um *plugin*, e eles geralmente são leves, não comprometem o funcionamento do software e são de fácil instalação e manuseio.

Nem todo programa aceita *plugins*, é preciso que ele tenha compatibilidade para estas ferramentas. Os principais softwares que utilizam extensões são o Mozilla Firefox, o Internet Explorer, Outlook, Thunderbird e mensageiros como o MSN. Com eles você pode aprender a digitar melhor, obter uma navegação mais segura, melhorar sua forma de receber mensagens ou mesmo gravar suas conversas entre várias outras funções.

O TestComplete é construído sobre uma arquitetura aberta, que permite escrever *plugins* externos para ele ou instalar *plugins* a partir de qualquer fonte. Você seleciona os *plugins* para ser incluído no TestComplete durante a instalação do produto ou pode incluí-lo ao seu projeto em um segundo momento (AUTOMATEDQA CORPORATION, 2010).

A criação de *plugins* pode ser realizada manualmente, através da codificação dentro do próprio TestComplete (que é o caso deste *plugin*), ou pode ser realizada por um programa externo (TestComplete SDK[3]) disponibilizado pela empresa que distribui o TestComplete, a AutomatedQa. Este programa pode ser baixado gratuitamente na página[4] do distribuidor.

2.6 TRABALHOS CORRELATOS

Foram encontrados alguns trabalhos de conclusão de curso da FURB pertinentes ao tema, relacionados ao uso da automatização de testes e utilização da ferramenta TestComplete. Nas subseções a seguir são apresentados.

---

[3] TestComplete SDK permite aos desenvolvedores ou mesmo testadores com conhecimento avançado no TestComplete a criação de *plugin* que podem estender ou criar novas funcionalidades do TestComplete (AUTOMATEDQA CORPORATION, 2010).

[4] http://smartbear.com/support/downloads/testcomplete/sdk/

2.6.1 Ferramenta de apoio à automatização de testes através do TestComplete para programas desenvolvidos em Delphi

Marcos (2007) apresentou um trabalho de conclusão de curso na Universidade Regional de Blumenau, intitulado de Ferramenta de Apoio à automatização de testes através do TestComplete para programas desenvolvidos em Delphi. O objetivo era gerar automaticamente código DelphiScript na ferramenta de automatização TestComplete baseando-se no código fonte gerado em Delphi no desenvolvimento das aplicações.

Basicamente é uma aplicação (Figura 6) que importa formulários DFM e através de analisadores de linguagens (léxico, sintático e semântico) gera código em uma das linguagens que o TestComplete utiliza como base (DelphiScript).

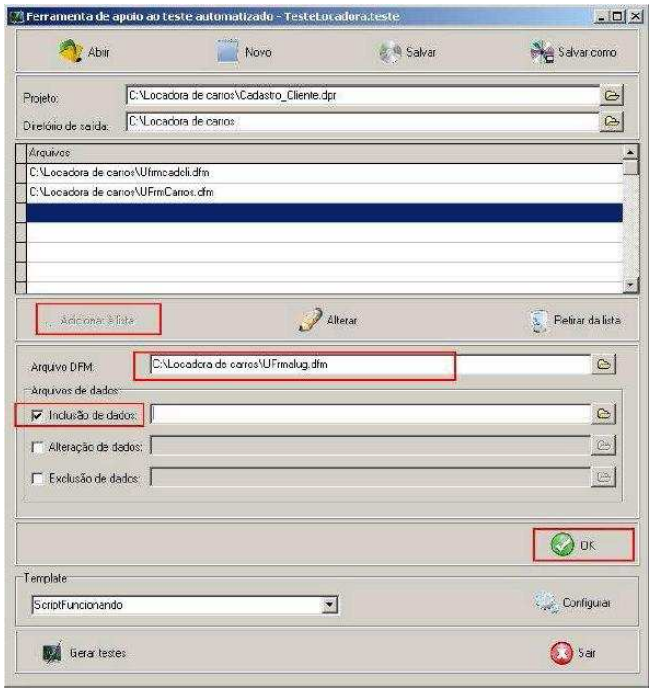


Figura 6 - Tela principal da Aplicação

Pode-se observar na ferramenta desenvolvida no trabalho correlato o processo de criação de *scripts* de testes. É possível adicionar vários formulários DFM através do botão 'Adicionar à Lista'. Para cada formulário selecionado é possível informar uma fonte para Inclusão\Alteração\Exclusão dos dados (no exemplo é marcada a opção 'Inclusão de dados') e para finalizar o processo, e consequentemente gerar o *script*, é utilizado o botão 'OK'.

2.6.2 Testes de Software a partir da Ferramenta Visual Test

Na monografia de conclusão de curso, Tomelin (2001) apresenta seu trabalho que se intitula de Testes de Software a partir da Ferramenta Visual Test. Através de pesquisa de campo, este trabalho procurou avaliar uma determinada empresa no tocante às atividades de testes praticadas. Utilizou-se como base para avaliação algumas normas de qualidade. Depois de constatado um processo de testes já bem amadurecido, implementou soluções automatizadas para auxiliar a atividade de testes, em específico testes de regressão. Como solução tecnológica para a automatização, utilizou uma ferramenta proprietária chamada Visual Test (Figura 7). Esta ferramenta simula de forma automatizada a entrada de dados informadas pelo usuário final de forma a alimentar o sistema com cadastros, movimentações, relatórios e outros.

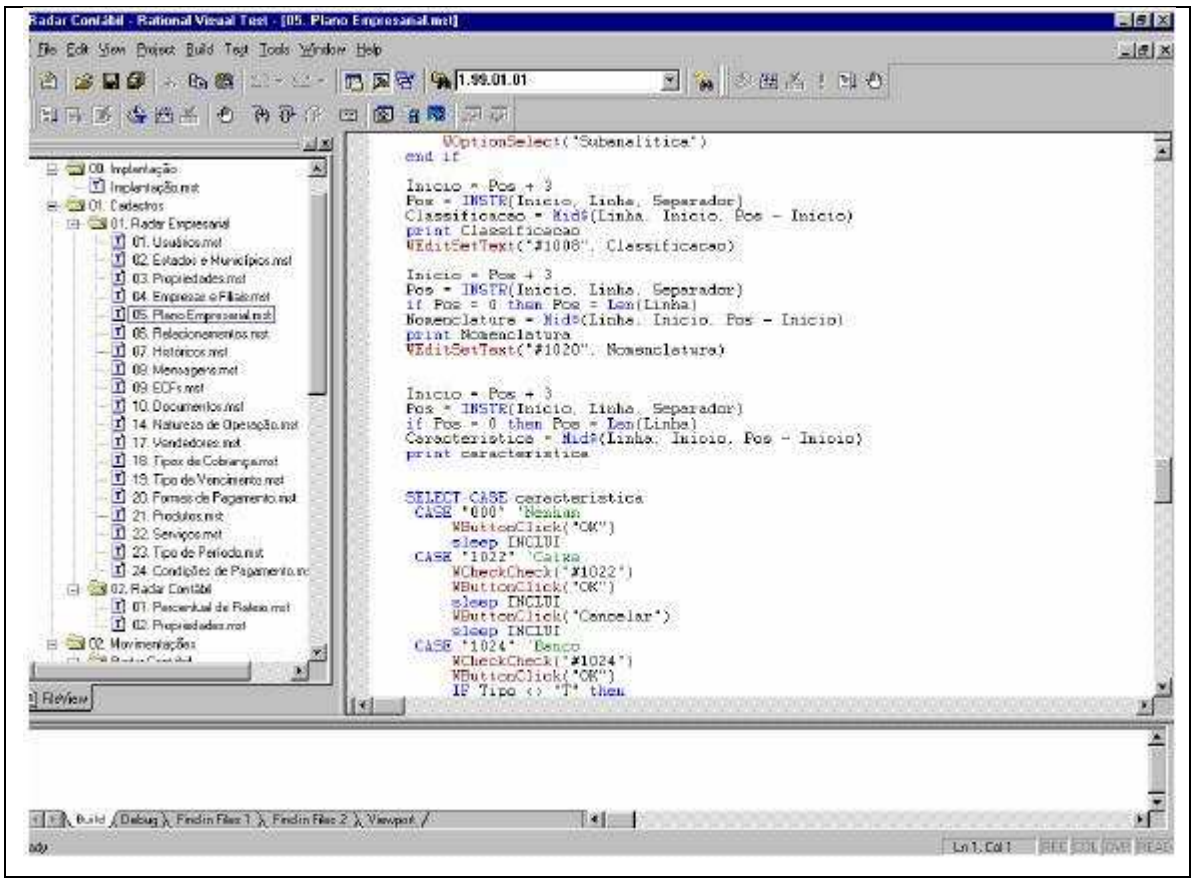


Figura 7 - Interface do Visual Test

3 DESENVOLVIMENTO DO SISTEMA

Neste capítulo são descritas as informações levantadas, detalhes da especificação, os diagramas de casos de uso e de atividade, a operacionalidade do sistema e ao final, os resultados e discussão.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Em um processo de teste a adoção de ferramentas é imprescindível para garantir a maturidade deste. Estas ferramentas devem prover mecanismos que garantam um bom planejamento e execução dos testes e que viabilizem grande parte das práticas utilizadas em todas as dimensões de um teste e paralelamente estejam aderentes ao processo.

Teve-se a oportunidade de participar ativamente de uma implantação e institucionalização de um processo de teste na empresa onde este autor trabalha. Este processo tinha como objetivo principal amadurecer a validação dos produtos gerados que consequentemente culminou na redução de falhas e elevou o grau de confiança do produto por parte dos clientes.

Optou-se em iniciar este processo através da utilização dos testes funcionais (caixa preta), haja visto que são mais simples de implementar e os resultados são facilmente constatados. Para manter os casos de testes optou-se pela ferramenta TestLink, pois percebeu-se que a forma de gerenciamento desta era aderente ao processo de teste estabelecido.

Outra prática estabelecida pelo processo, era a utilização de técnicas de regressão de testes, tendo em vista o número alto de falhas reincidentes entre as liberações de novas funcionalidades. Porém com o crescimento exponencial dos casos de testes, um cronograma de desenvolvimento reduzido e recursos humanos sendo empregados em sua totalidade, ficou claro e evidente a necessidade de utilização de uma ferramenta de automatização de testes para comportar a atividade de regressão. Após pesquisas de mercado, optou-se pela ferramenta TestComplete.

Seguida da aquisição desta ferramenta e capacitação das pessoas para criação dos *scripts* de automatização, percebeu-se que havia uma grande lacuna entre esta e a ferramenta que gerenciava os casos de testes (TestLink). Não havia um sincronismo entre estas e o

retrabalho operacional acabava por tomar grande parte dos recursos empregados para os testes. Novamente foram realizadas pesquisas de mercado, porém nada no tocante a integração destas foi encontrada. Como projeto pessoal, teve-se início o desenvolvimento desta integração através de um *plugin* na ferramenta TestComplete. Os requisitos básicos e necessários para desenvolver o *plugin*, foram extraídos do *know-how*[5] deste autor obtido na área de testes de software e na constatação dos problemas encontrados diariamente na utilização das duas ferramentas.

3.2 ESPECIFICAÇÃO

Esta seção descreve os requisitos funcionais (RF) e não funcionais (RNF), bem como os diagramas de casos de uso e diagrama de atividades. A ferramenta Enterprise Architect (EA), foi utilizada na elaboração dos diagramas de casos de uso e diagrama de atividades.

3.2.1 Requisitos funcionais e não funcionais

O Quadro 2 apresenta os requisitos funcionais do *plugin*, bem como a vinculação com os casos de uso. Por questões de padrões de descrição de requisitos foi utilizado a palavra "sistema" para indicar "*plugin* do TestComplete".

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deverá permitir configurar ambiente de testes antes de iniciar o processo de automatização.	UC.01
RF02: O sistema deverá possibilitar que um caso de teste criado no TestLink possa ser executado automaticamente no TestComplete.	UC.02
RF03: O sistema deverá possibilitar a execução de lotes de casos de testes automaticamente no TestComplete.	UC.02
RF04: O sistema deverá manter um histórico de cada caso de teste executado pelo TestComplete e ao final da automatização gerar automaticamente no TestLink informações referente a cada caso de teste executado.	UC.04, UC.05

[5] Termo em inglês, utilizado para designar uma técnica, um conhecimento ou uma capacidade desenvolvida por uma organização ou por uma pessoa.

RF05: O sistema deverá possibilitar a codificação de script de testes a partir de funções pré-definidas.	UC.03
RF06: O sistema deverá possibilitar a utilização da massa de dados vinculada no TestLink sem a necessidade de carregá-la explicitamente no TestComplete.	UC.06

Quadro 2 - Requisitos funcionais

O Quadro 3 lista os requisitos não funcionais do *plugin*.

Requisitos Não Funcionais
RNF01: O sistema deverá ser desenvolvido utilizando as linguagens Pascal, DelphiScript e nativa do TestComplete.
RNF02: O sistema deverá utilizar o banco de dados MySQL para manter a base de dados da ferramenta TestLink.
RNF03: O sistema deverá ser desenvolvido para automatizar testes em ambiente Desktop.

Quadro 3 - Requisitos não funcionais

3.2.2 Diagramas de casos de uso

Os casos de uso foram criados a partir da visão do ator usuário que pode ser um Analista de Teste ou Testador. Estes tem como responsabilidade realizar as codificações dos scripts de testes na ferramenta. O detalhamento dos principais casos de uso está descrito no Apêndice A. Na Figura 8, pode-se observar as funcionalidades disponíveis ao usuário.

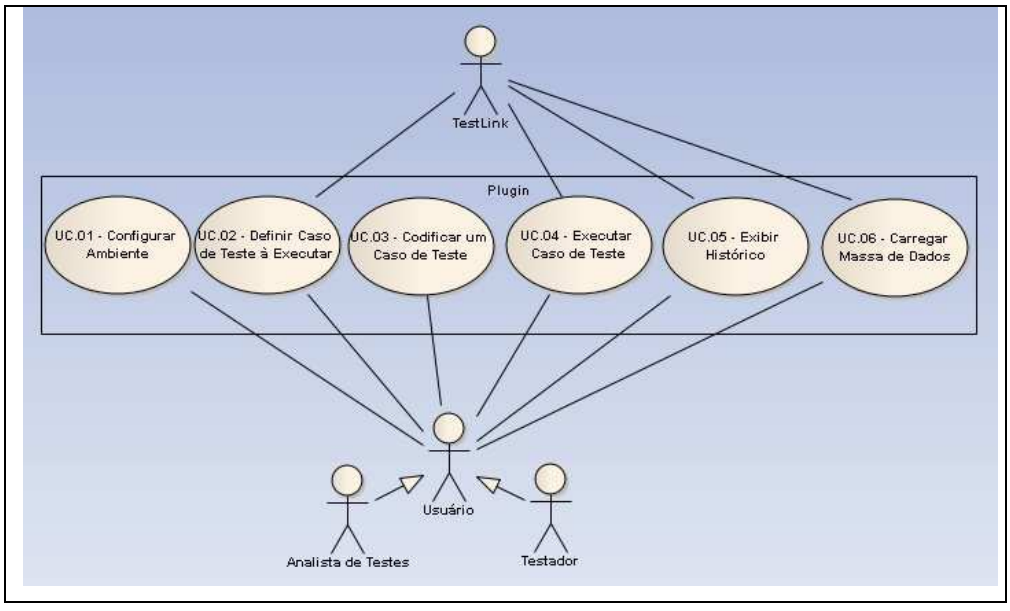


Figura 8 - Diagrama de caso de uso do usuário do sistema

Os atores “Analista de Testes” e “Testador” herdam os mesmos casos de usos, pois no TestComplete e consequentemente no *plugin* não existe a possibilidade de realizar um *login* para cada usuário e acesso a seu respectivo perfil.

Somente os casos de uso (UC.02 , UC.04, UC.05 e UC.06) possuem associação com o ator “TestLink” pois em determinados momentos o *plugin* acessa a base de dados da ferramenta TestLink.

3.2.3 Diagrama de atividades

Esta sub-seção apresenta um diagrama de atividades (Figura 9) demonstrando a execução de um *script* de teste no TestComplete utilizando o *plugin* de integração.

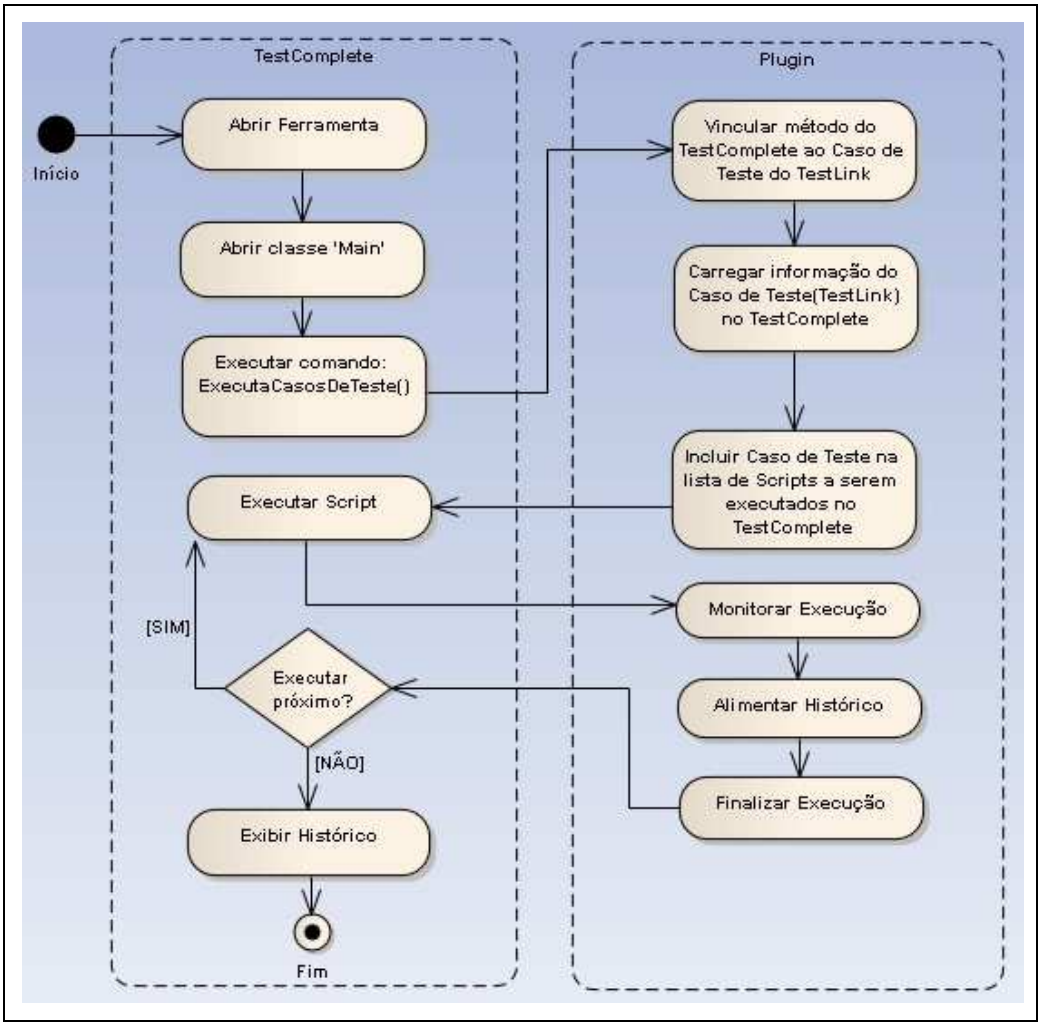


Figura 9 - Diagrama de Atividades: Executando um *Script*

O diagrama demonstrado na Figura 9 refere-se a um processo completo no tocante a execução dos *scripts* vinculados aos casos de testes. Esta atividade é de responsabilidade do ator “Testador”. Salvo os casos de uso UC.01 e UC.03 que são de responsabilidade do ator “Analista de Testes” e que previamente ao fluxo desta atividade já foram executados, todos os demais casos de usos são utilizados no diagrama da Figura 9.

3.2.4 Diagrama entidade-relacionamento

O Diagrama Entidade-Relacionamento (DER) é um diagrama que descreve o modelo de dados de um sistema com alto nível de abstração. A Figura 10 apresenta o DER com as principais entidades da ferramenta TestLink que serão mantidas através da integração do *plugin*.

Figura 10 – Diagrama entidade-relacionamento

A seguir é apresentada uma breve descrição das entidades utilizadas pelo *plugin* na integração:

a) *attachments*: entidade responsável por manter informações referentes ao anexos que são vinculados ao casos de testes e na execução destes;
b) *builds*: entidade responsável por manter informações das *baselines* definidas na execução de plano de testes;
c) *cfield_design_values*: entidade responsável por manter informações referentes aos

valores dos campos personalizados vinculados aos casos de testes no momento da criação destes;

d) *cfield_execution_values*: entidade responsável por manter informações referentes aos valores dos campos personalizados vinculados aos casos de testes no momento da execução destes;

e) *tcversions*: entidade responsável por manter informações referentes aos casos de testes;

f) *tcsteps*: entidade responsável por manter informações referentes aos passos definidos em caso de teste;

g) *keywords*: entidade responsável por manter informações referentes as palavras chaves vinculadas a um caso de teste;

h) *executions*: entidade responsável por manter informações referentes a execução dos casos de testes.

3.3 IMPLEMENTAÇÃO

Nesta seção estão apresentadas informações sobre as ferramentas e técnicas utilizadas no desenvolvimento do *plugin*, juntamente com a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

No desenvolvimento do *plugin*, foram utilizados os *softwares*:

a) mySQL 5.0.45 como banco de dados;

b) mySQL administrator 1.2.12 como gerenciador de banco de dados;

c) linguagens Pascal, DelphiScript e nativa do TestComplete como linguagem de programação;

d) testComplete 7.10.475.7 como IDE de desenvolvimento;

e) testLink 1.9.2 como gerenciador dos casos de testes.

3.3.2 Operacionalidade da implementação

Esta sub-seção apresenta as principais funcionalidades do *plugin* desenvolvido, tendo em vista que o resultado da implementação não pôde ser constatada através de telas do *plugin* propriamente dito, mas sim no uso deste interagindo com as duas ferramentas (Testlink e TestComplete). O que é demonstrado através de imagens é a utilização destas ferramentas e o *plugin* realizando ações no tocante a integração destas.

3.3.2.1 Analista de Testes utilizando o TestLink

Na tela apresentada na Figura 11, o usuário deve informar seu nome de usuário e senha para acessar a ferramenta TestLink.

Figura 11 - Tela de *login* do TestLink

No TestLink, cada usuário pode ter um perfil de acesso diferenciado, no caso do ator “Analista de Teste”, possui permissão para cadastrar o casos de testes que serão executados através da ferramenta de automatização, o TestComplete. Na Figura 12 é demonstrado a criação de um caso de teste, para tal deverá ser acessado o item de *menu* “Especificação” e em seguida escolher na árvore de suíte de testes onde será criado o novo caso de teste.

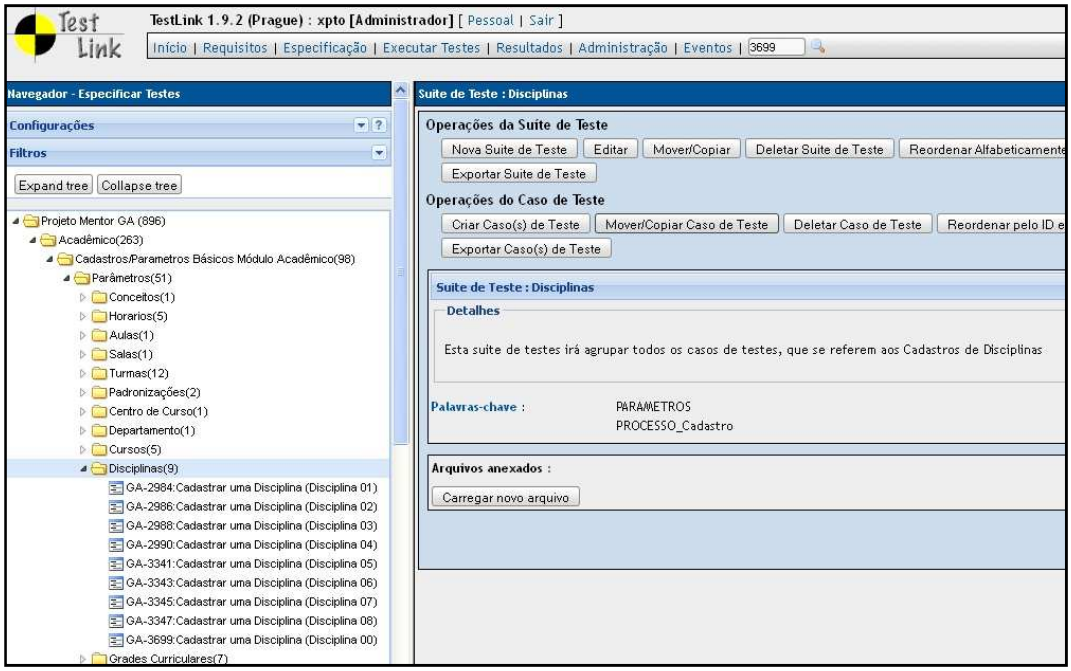


Figura 12 – Tela de criação do CT – escolha da suíte de testes

A organização de casos de testes entre as suítes deve respeitar o critério lógico da aplicação a ser testada, ou seja, nos nodos mais acima da árvore encontram-se os casos de testes referentes aos cadastros (CRUD[6]) e se há dependência entre casos de testes. Por convenção do processo de teste estabelecido, deverão estar dispostos em níveis superiores dos quais os dependem.

Depois de definido o local do caso de teste, é selecionada a opção “Criar Caso(s) de Teste” e dá-se início, conforme a Figura 13, a etapa da criação propriamente dita do caso de teste.

[6] CRUD (acrônimo de *Create*, *Read*, *Update* e *Delete* em língua Inglesa) é utilizado para representar as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface dos usuários para criação, consulta, atualização e destruição de dados.

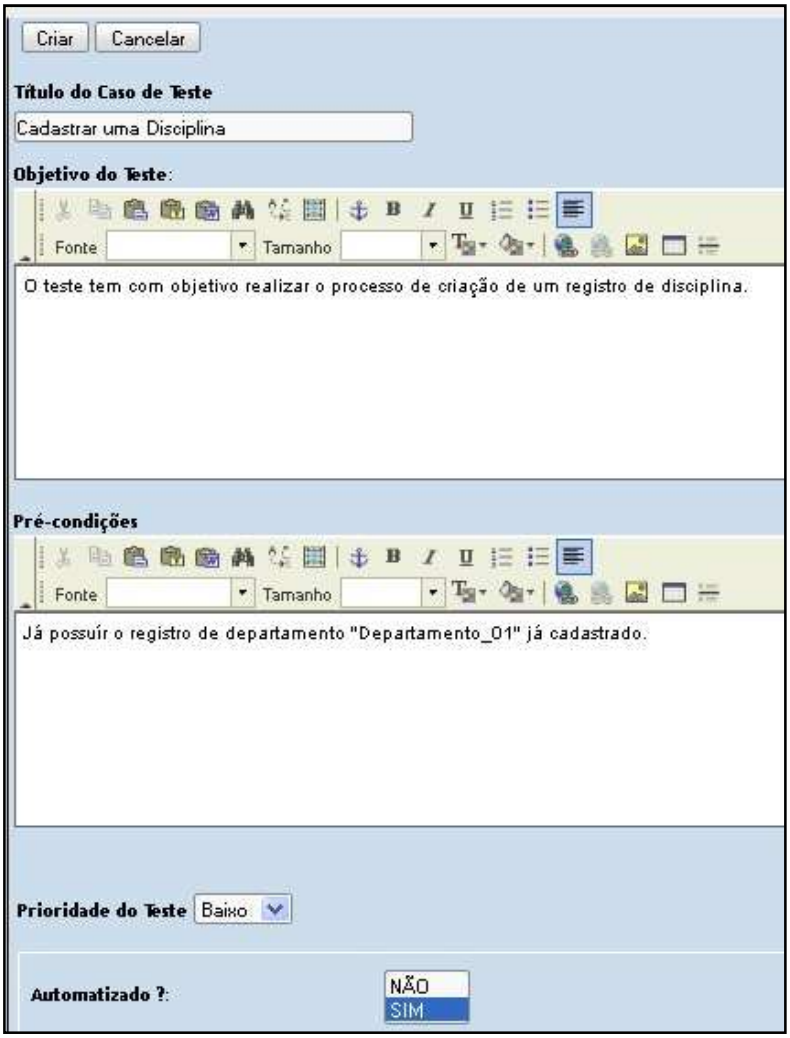


Figura 13 - Tela de criação do CT – novo

É nesta etapa que são definidos quais casos de testes serão carregados pelo *plugin* no TestComplete (UC.02 – Definir Caso de Teste à Executar). Para tal, faz-se necessário selecionar a opção “SIM” do campo “Automatizado?”.

Depois de informados os dados básicos do caso de teste, para efetivar a criação faz-se necessário clicar no botão “Criar” conforme Figura 13. A próxima etapa na criação é a definição de passos para o caso de teste conforme a Figura 14.

Figura 14 - Tela de criação do CT : Passos

Nesta etapa são definidos os valores de entrada do caso de teste e seu(s) respectivo(s) resultado(s) esperado(s). Aqui também é vinculada uma planilha com os dados de entrada (UC.06 – Carregar Massa de Dados), para tal procedimento faz-se necessário clicar no botão “Carregar novo arquivo” conforme Figura 14 e selecionar uma planilha do Excel para ser vinculada ao caso de teste. Definiu-se para utilização no *plugin* que o arquivo deve ser nomeado conforme o seguinte padrão:

a) possuir o prefixo “CT_”;
b) os quatros caracteres após o prefixo, devem ser substituídos pelo código do caso de teste que é exibido no canto superior esquerdo da Figura 14, neste caso, o valor “4998”.

Já os valores informados dentro da planilha também devem seguir os padrões:

a) os dados devem ser preenchidos sempre na primeira aba, que deve ser nomeada como “DADOS”;
b) as colunas da primeira linha da planilha dizem respeito a identificação dos campos

dos valores de entrada informados na criação dos passos do caso de teste;

c) a partir da segunda linha, cada coluna representará o dado de entrada propriamente dito, e o conjunto de valores de cada coluna de uma linha representará todos os valores de entradas definidos nos passos do caso de teste.

Desta forma, é possível ter um único caso de teste simulando vários processos de inclusão, pois os valores de entradas não estão explicitados nos passos do caso de teste e sim na planilha vinculada a este. Já quando o Analista de Teste for realizar a codificação de um *script* de teste e houver a necessidade de buscar os dados desta planilha através da técnica de DDT disponibilizada pelo TestComplete, basta ele simplesmente fazer chamada da variável pública "FPlanilhaDados", que o *plugin* durante a execução dos *scripts* realizará a vinculação lógica da planilha com a variável acima.

Basicamente a atividade do ator "Analista de Teste" no TestLink restringe-se em manter os casos de testes, e a cada nova funcionalidade disponibilizada, este deve criar tantos casos de testes quantos forem necessários para proporcionar uma boa cobertura de teste.

3.3.2.2 Analista de Teste configurando o *plugin*

A primeira atividade do analista de teste depois que é criado um novo projeto no TestComplete é a configuração do *plugin* (UC.01 – Configurar Ambiente), para tal atividade faz-se necessário realizar o seguinte procedimento na *unit* principal do TestComplete, na declaração de *uses*[7] desta, deve-se incluir a *unit* "unIntegracao";

Ao realizar a primeira execução das rotinas automatizadas utilizando-se do *plugin*, será solicitado ao usuário algumas informações (conforme Figura 15) que serão utilizadas para realizar o acesso a base de dados do TestLink.

[7] Por padrão, o código de script pode chamar somente aquelas rotinas (procedimentos ou funções), variáveis e constantes que residem na mesma unidade. Para chamar rotinas, variáveis ou constantes declaradas em outra unidade, a unidade atual deve ser "ligados" a ele (AUTOMATEDQA CORPORATION, 2010).

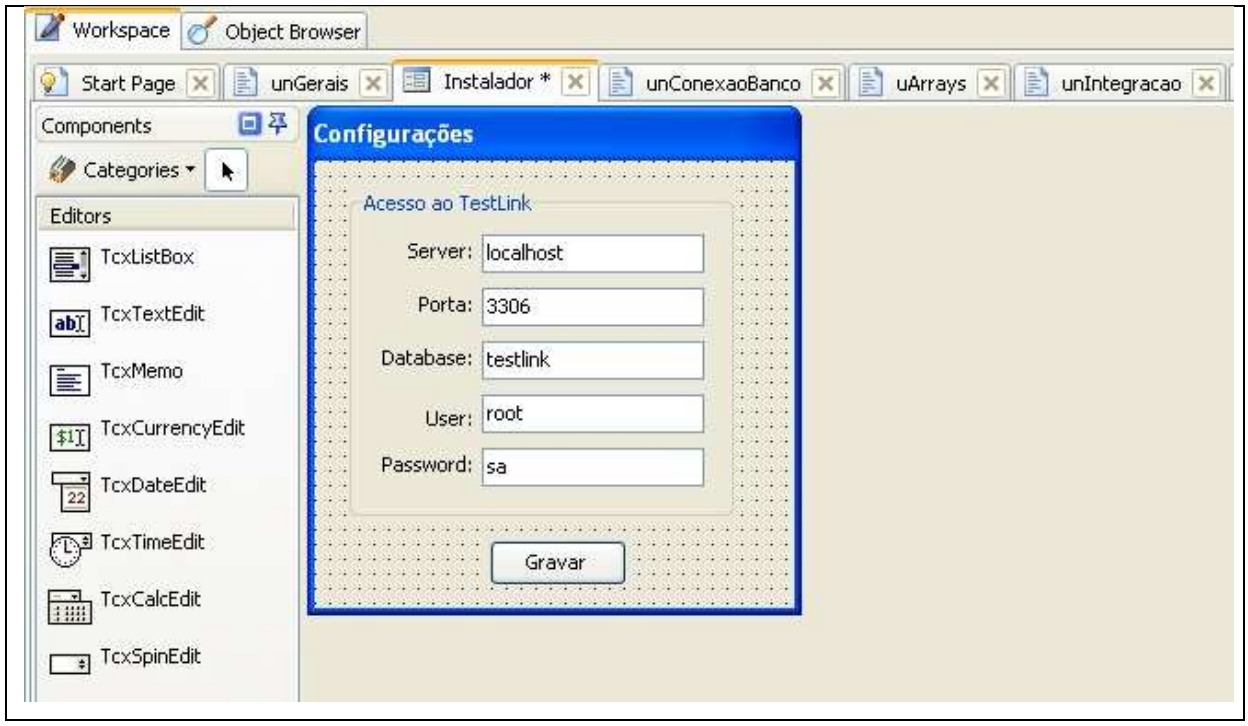


Figura 15 – Configurando o *plugin*

A Figura 15 demonstra a utilização da funcionalidade *UserForms*[8] disponível no TestComplete, esta funcionalidade permite a criação de formulário estáticos que serão exibidos durante a execução dos *scripts* de automatização. Por tanto, ao realizar a primeira execução, o *plugin* identificará se as informações referentes ao acesso à base de dados do TestLink estão preenchidas, se não o *plugin* realizará a chamada da *UserForms* "Instalador" que solicitará as informações conforme a Figura 15 e passará a mantê-las.

3.3.2.3 Analista de Teste utilizando o TestComplete

Uma das atividades do analista de testes é a criação dos *scripts* de automatização de testes na ferramenta TestComplete. Todos os *scripts* são baseados nos casos de testes mantidos na ferramenta TestLink, o analista transcreve os passos do casos de teste (Figura 16) através de um *script* codificado (Figura 17) na linguagem do TestComplete.

[8] Utilizando o recurso de *UserForms* você pode cria formulários personalizados da mesma forma com o que se faz isso em uma ferramenta de desenvolvimento qualquer. É fornecido com um conjunto comum de componentes, campos de entrada, caixas de listagem, botões de rádio, caixas de seleção, e assim por diante. Você também pode criar manipuladores de eventos para componentes e utilizá-los da forma que necessitar (AUTOMATEDQA CORPORATION, 2010).

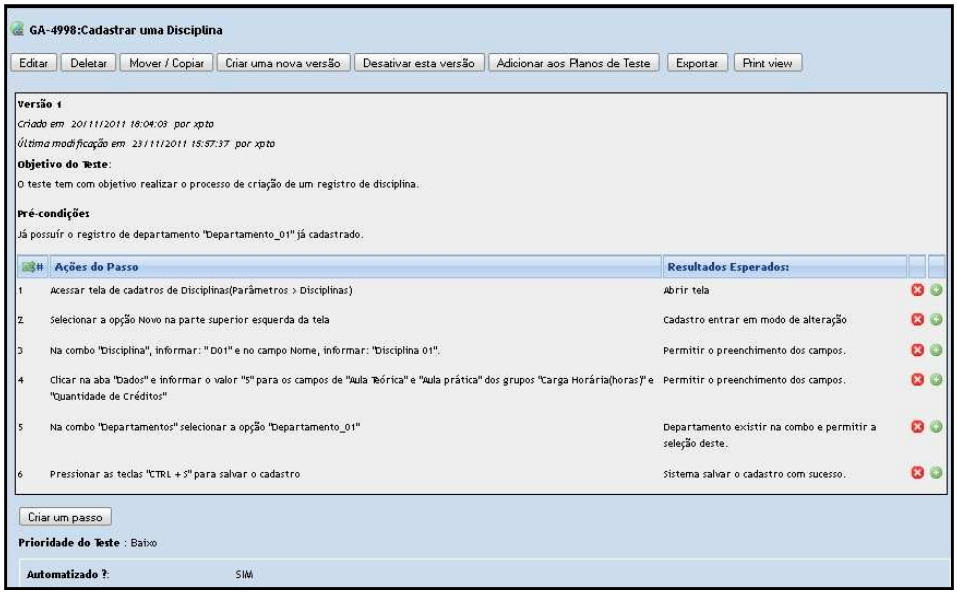


Figura 16 - Passos de um caso de teste a ser codificado

Por definição para ser utilizado pelo *plugin*, todo *script* codificado para automatizar um caso de teste, deve manter o seguinte padrão na sua assinatura:

a) ser uma *procedure*;

b) possuir o prefixo “CT_”;

c) os cinco caracteres após o prefixo, devem ser substituídos pelo código do caso de teste, neste caso o valor “04998”.

A Figura 17 apresenta o código gerado para o *script* de automatização dos passos do caso de teste da Figura 16.

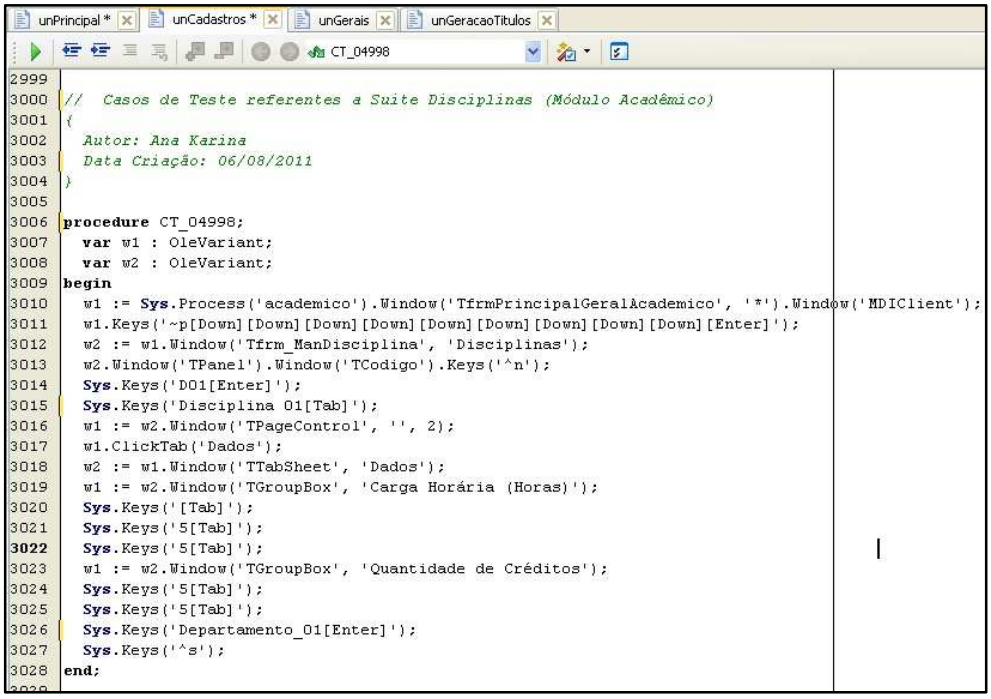


Figura 17 - Código fonte do *script* de automatização do caso de teste 4998

No código fonte da Figura 17, as linhas iniciais (3010 até 3012) após a declaração de variáveis representam o passo 1 do caso de teste, já a linha 3013 é responsável por tornar o cadastro editável (passo 2), o preenchimento do código e nome da disciplina estão respectivamente nas linhas 3014 e 3015 (passo 3), o quarto passo encontra-se da linha 3016 até a 3025, a seleção do departamento (passo 5) encontra-se na linha 3026 e por fim o último passo que salva o cadastro esta codificado na linha 3027.

Nesta etapa, a de codificação, o *plugin* possibilita algumas facilidades para o codificador (UC.03 – Codificar um Caso de Teste). As mais significativas são o fato de não precisar explicitar via código o tratamento exceção e retorno para o estado inicial antes da execução do teste. Com isso o Analista de Testes preocupa-se com a qualidade do código em relação ao teste e não precisa realizar vários controles em cada *script* para evitar que a automatização falhe por conta de erros operacionais do TestComplete e sim, se venha a falhar que seja por conta de uma mudança na aplicação em teste. Sem inserir nenhuma linha de código a mais, cada script executado pelo *plugin* gerencia possíveis exceções (ex: campo obrigatório não informado) e quando isso ocorre encerra a execução do *script*, fecha a tela em execução, grava a imagem do erro ocorrido e aponta para o próximo caso de teste na lista de automatizados a serem executados. Com isso resolve-se um problema típico na execução de *scripts* que é a “falha em cascata”, ou seja, em situações normais no TestComplete (sem o uso do *plugin*) quando uma falha ocorria e não era codificado uma finalização adequada, consequentemente a tela em execução não era encerrada, o TestComplete executava o próximo *script* e este falhava também, pois como pode ser observado na Figura 17 os *scripts* são baseados nos nomes dos objetos e como estes não eram encontrados na tela ativa, considerava-se um erro e o teste falhava.

A atividade do ator “Analista de testes” no TestComplete resume a criação e manutenção dos *scripts* de automatização dos casos de testes, a execução e monitoramento da automatização são atividades do ator “Testador” que é apresentado na sub-seção abaixo.

3.3.2.4 Testador acessando o TestComplete

O ator “Testador” tem como responsabilidade no TestComplete montar a lista de *scripts* a serem executados, baseado na lista de casos de testes cobertos naquela versão*release* que foi previamente elencada no plano de testes. Sem a utilização do *plugin*, esta atividade demanda um esforço operacional muito elevado e exige um grau de atenção do

testador na definição da ordem dos *scripts* a serem executados, isso sem contar que novos *scripts* são criados diariamente, e consequentemente a revisão periódica desta lista deve ser realizada.

Por definição, existe uma *unit* principal e dentro desta uma *procedure* nomeada como “Main”, no TestComplete foi parametrizado (conforme Figura 18) que a execução dos *scripts* sempre partirá desta *procedure.*

Portanto, ainda não se utilizando do *plugin*, o testador obrigatoriamente dentro da *procedure* “main” realiza a chamada via código de cada *script* vinculado a um caso de teste, sempre procurando manter a ordem correta para não correr o risco de executar algum *script* dependente de outro que ainda não foi executado, assim gerando uma falha no teste por conta de um erro operacional.

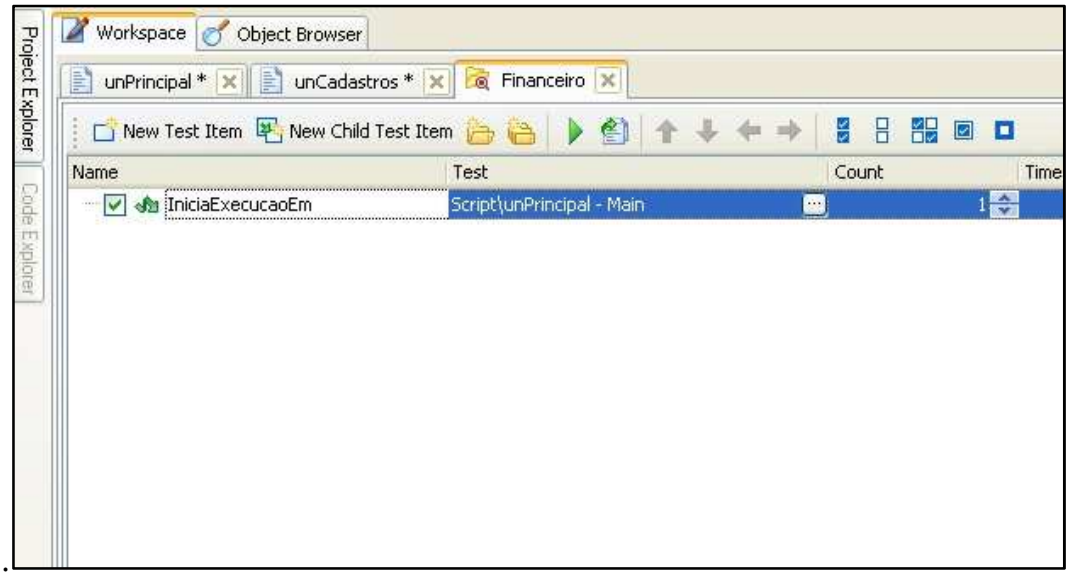


Figura 18 - Configurando no TestComplete onde iniciará a execução do testes

Com a utilização do *plugin* (UC.02 – Definir Caso de Teste à Executar), o testador não precisa mais empregar seu tempo nesta tão onerosa atividade, pois todas as chamadas dos *scripts* passam a ser através do método “executaCasosDeTestes()” disponibilizado pelo *plugin*. Neste método é possível realizar a chamada de um único *script* vinculado a um caso de teste, ou por exemplo todos os *scripts* vinculados aos casos de testes marcados como automatizados na ferramenta TestLink. Para utilizar a execução de *scripts* em lote basta indicar o nome de uma suíte de teste em qualquer ponto da árvore de suítes do TestLink, e o *plugin* montará a lista de *scripts* respeitando a ordem lógica dos nodos da árvore.

Na Figura 19 é demonstrada a chamada de *script* através das duas formas possíveis, individualmente ou em lote, ambas utilizam-se do seguinte procedimento:

a) acessar a base de dados do TestLink;

b) realizar consulta na base identificando quais casos de testes estão marcados como ‘automatizado’ (conforme Figura 20);

c) dentre os registros carregados, identificar se estes entram na condição imposta pelo parâmetro do método “executaCasosDeTestes()”;

d) os registros que satisfazem a condição acima são incluídos na lista de casos de testes a serem executados no TestComplete.

unPrincipal * | unCadastros | Financeiro

```
5   var FModulo: OleVariant;
6   begin
7     {Inicialização do módulo a ser testado e das variáveis utilizada no Tcp}
8       FModulo := Sys.Find('ProcessName','academico');
9       if not FModulo.Exists then
10      begin
11        FModulo := TestedApps.Academico.Run();
12        setModulo(FModulo);
13        fazerLogin(FModulo);
14      end
15      else
16      begin
17        setModulo(FModulo,true);
18        FModulo.Window('TApplication', 'academico', 1).Restore;
19      end;
20    {Final Incilização ** Não chamar nada acima desse comentário}
21
22    {Inicio do Bloco de chamada para execução dos CTs}
23      //Realizado chamada de um único caso de teste
24      executaCasosDeTestes('',4998);
25
26      //Realizado chamada de todos os casos de testes automatizados
27      //a partir da suite 'Disciplinas'
28      executaCasosDeTestes('Disciplinas');
29
30    {Final do Bloco de chamada dos CTs}
31
32    {Finalização do Tcp e chamada do relatório de execução dos CTs}
33      encerraModulo;
34      Historico;
35  end;
```

Figura 19 - Realizando a chamada de um *script* através do *plugin*

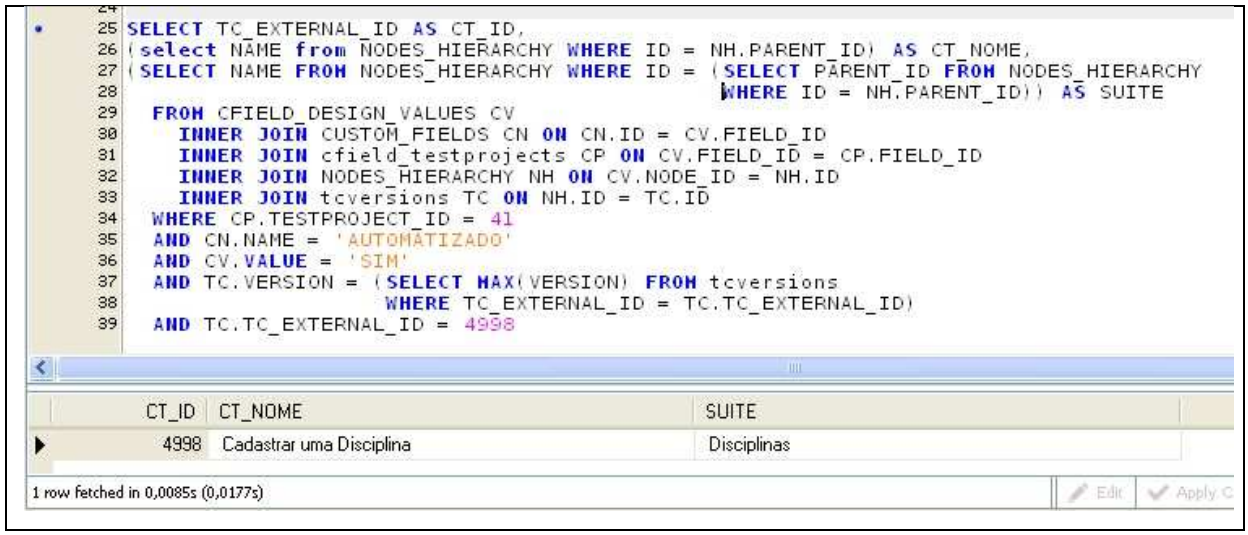


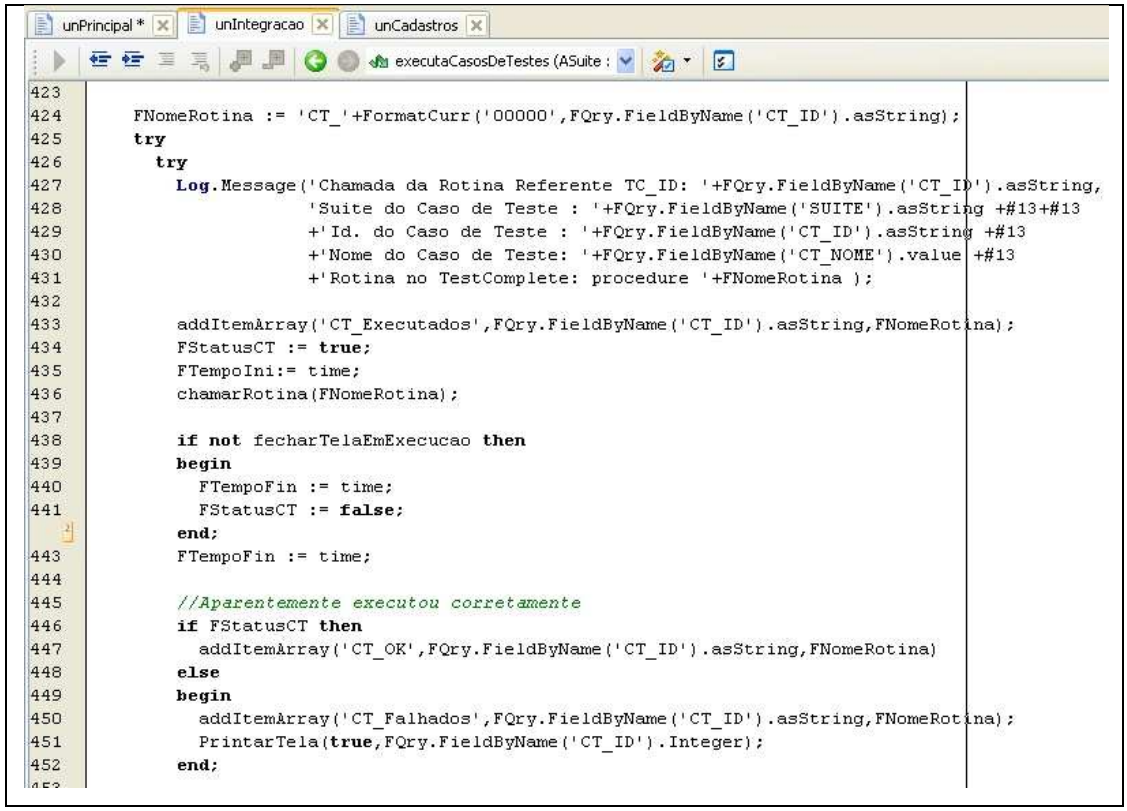
Figura 20 – Instrução SQL para retornar caso de teste automatizado

Como definição, sempre utiliza-se o nodo mais alto da árvore de suítes como parâmetro de entrada do método, pois assim evita-se o risco da execução de *scripts* fora de ordem, haja visto que o *plugin* montará a lista exatamente da mesma forma de como esta definida no TestLink.

Passada a etapa de definição dos *scripts* a serem executados, o ator “Testador” pode iniciar a atividade de execução (UC.04 – Executar Caso de Teste) dos *scripts* propriamente ditos. Onde a partir da utilização do método acima citado, o *plugin* seqüencialmente passa a executar os *scripts* e monitorar seus resultados conforme descrito na sub-seção acima (UC.03 – Codificar um Caso de Teste), e para os resultados obtidos mantém uma lista de casos de testes que passaram e outra dos que falharam.

Para cada lista, algumas informações adicionais também são gravadas, estas são:

a) *script* (nome da *procedure*) do TestComplete que foi executado para o caso de teste;
b) suíte de teste do TestLink vinculada ao caso de teste;
c) identificação do caso de teste (número) no TestLink;
d) nome do caso de teste no TestLink;
e) tempo gasto para execução do *script*;
f) em caso de falha, uma imagem do erro exibido em tela.




```
423
424     FNomeRotina := 'CT_'+FormatCurr('00000',FQry.FieldByName('CT_ID').asString);
425     try
426       try
427         Log.Message('Chamada da Rotina Referente TC_ID: '+FQry.FieldByName('CT_ID').asString,
428                     'Suite do Caso de Teste : '+FQry.FieldByName('SUITE').asString +#13+#13
429                     +'Id. do Caso de Teste : '+FQry.FieldByName('CT_ID').asString +#13
430                     +'Nome do Caso de Teste: '+FQry.FieldByName('CT_NOME').value +#13
431                     +'Rotina no TestComplete: procedure '+FNomeRotina );
432
433         addItemArray('CT_Executados',FQry.FieldByName('CT_ID').asString,FNomeRotina);
434         FStatusCT := true;
435         FTempoIni:= time;
436         chamarRotina(FNomeRotina);
437
438         if not fecharTelaEmExecucao then
439         begin
440           FTempoFin := time;
441           FStatusCT := false;
          end;
443         FTempoFin := time;
444
445         //Aparentemente executou corretamente
446         if FStatusCT then
447           addItemArray('CT_OK',FQry.FieldByName('CT_ID').asString,FNomeRotina)
448         else
449         begin
450           addItemArray('CT_Falhados',FQry.FieldByName('CT_ID').asString,FNomeRotina);
451           PrintarTela(true,FQry.FieldByName('CT_ID').Integer);
452         end;
```

Figura 21 - Trecho do código fonte responsável por executar e monitorar os *scripts*

A Figura 21 demonstra parte do código que é responsável por executar os *scripts* através do método privado do *plugin* "chamarRotina()" e de acordo com o *status* da execução, incluir caso de teste na lista de executados com sucesso ou falha.

Depois de executado todos os *scripts* previstos para o teste, a etapa final do *plugin* é descarregar todas essas informações mantidas durante as execuções no TestLink. Em situações sem o uso do *plugin* , depois de realizado a automatização o testador acessava o *log* padrão do TestComplete e tentava identificar em que ponto das execuções ocorreram falhas. Quando encontrava uma relação da falha com algum *script* vinculado ao caso de teste, acessava a ferramenta TestLink e manualmente realizava o registro da execução deste (Figura 22).

Figura 22 - Registro manual da execução de um caso de teste no TestLink

Essa também era uma atividade que demandava muito tempo do testador. Além do tempo de análise do *log* padrão gerado pelo TestComplete, havia o tempo para o registro manual desta execução, haja visto que é o TestLink responsável por gerir toda a atividade de teste e o respectivo registro de automatização. Sem contar que por definição, os testes automatizados eram executados sempre ao final de cada fluxo de desenvolvimento (aproximadamente 2 semanas) para garantir através do conceito de "regressão" que nenhuma das rotinas do sistema em teste fora impactada por novas alterações.

Para armazenar (UC.05 – Exibir Histórico) os resultados dos testes no TestLink, o *plugin* disponibiliza o método "Historico" (conforme Figura 19) , e este deve ser utilizado sempre depois de executado todos os *scripts*. A Figura 23 apresenta um resultado de um caso de teste automatizado que resultou em falha e seu respectivo registro pelo *plugin* no TestLink.

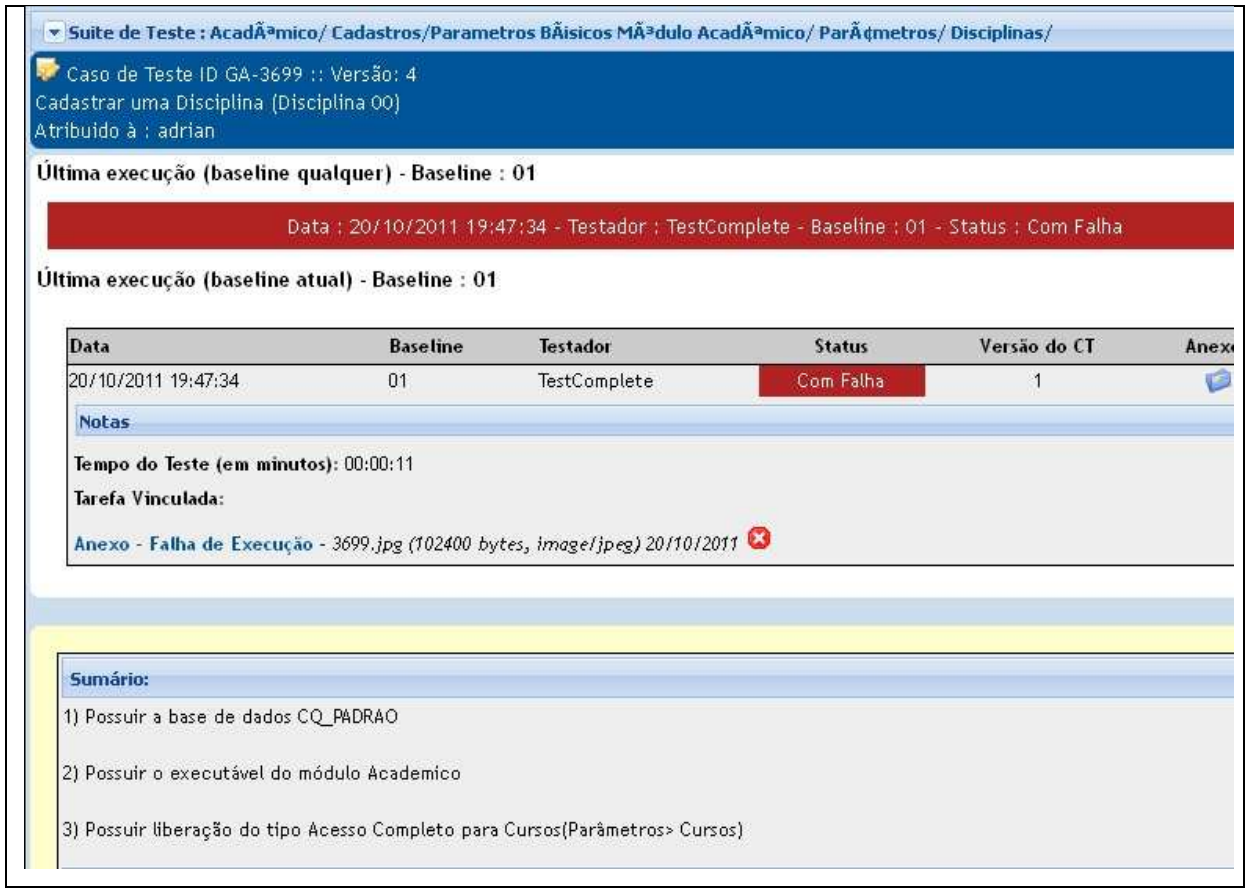


Figura 23 - Registro automático da execução de um caso de teste no TestLink

Na Figura 23 pode-se observar o registro de falha pelo Testador “TestComplete”, o tempo gasto na execução deste e um anexo referente a imagem do erro ocorrido durante a execução.

3.4 RESULTADOS E DISCUSSÃO

Observando os trabalhos correlatos apresentados na seção 2.6 e realizando uma busca nas principais páginas especializadas em teste de software, pode-se constatar que não havia até o desenvolvimento do presente trabalho nada relacionado a uma integração entre as ferramentas TestComplete e TestLink. O trabalho correlato de Tomelin (2001), contribuiu para evidenciar, assim como neste trabalho, que além de um processo de teste maduro, ferramentas de apoio no tocante a automatização de testes são fundamentais. Já o trabalho correlato de Marcos (2007) que se utilizou da ferramenta TestComplete, teve sua parcela de contribuição com relação ao uso da ferramenta propriamente dita.

Com relação aos objetivos específicos deste trabalho, todos foram considerados atingidos, haja visto que a partir da utilização do *plugin* pela equipe de testes da empresa (Analistas e Testadores) não gastou-se mais tempo na atividade de registro de execução de casos de testes automatizados e diminui-se o tempo de retrabalho na atividade de ajustes de *scripts* já automatizados (situações de tratamento de exceção não esperada no código). Para se ter uma idéia, antes do *plugin* o tempo médio para a codificação de um *script* de teste referente a um teste de CRUD[9] em uma tela de cadastro levava em torno de duas horas, onde para cada *script* obrigatoriamente os seguintes itens deveriam constar no código:

a) codificação referente a chamada da tela a ser testada;
b) inicializações de variáveis de controle do teste;
c) codificação referente aos passos do caso de testes;
d) validações de resultados dos passos;
e) tratamentos de exceções;
f) registros no log padrão do TestComplete;
g) em caso de falha, retorno para posição inicial antes da execução deste *script*.

Foi realizado pelo autor um estudo do tempo utilizado para codificação de *scripts* com a utilização do *plugin*, onde utilizou-se como base a codificação de um *script* de mesma complexidade. Este procedimento foi executado 10 vezes e constatou-se que o tempo médio baixou para 16 minutos. Já com relação a codificação do *script*, manteve-se apenas os seguintes itens:

a) codificação referente a chamada da tela a ser testada;
b) codificação referente aos passos do caso de testes;
c) validações de resultados dos passos.

Atualmente utiliza-se o tempo de codificação apenas para agregar valor ao produto e não mais com controle operacionais.

Já para a atividade de registro dos resultados obtidos pela execução dos casos de testes automatizados, retirou-se da ferramenta (0800Net), que é responsável pelo registro de tarefas e seus respectivos tempos, em um período de 2 meses (aproximadamente é o tempo de liberação entre uma versão e outra) o quanto foi gasto na atividade relacionada ao registro de execuções no TestLink. O resultado foi que gastou-se 64 horas nesta atividade, levando em conta que além do tempo do registro manual havia o tempo de análise do log padrão gerado

[9] CRUD (acrônimo de *Create*, *Read*, *Update* e *Delete* em língua Inglesa) é utilizado para representar as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface dos usuários para criação, consulta, atualização e destruição de dados.

pelo TestComplete. A partir da utilização do *plugin*, o número de horas empregadas para esta atividade veio à zero, pois todo o registro de histórico de execução passou a ser realizado automaticamente pelo *plugin*.

A Figura 24 demonstra o resultado do relatório de métricas (relatório disponibilizado pelo TestLink).

Relatório de Métricas de Queries

Projeto de Teste : **Projeto Mentor GA**
Plano de Teste : **Plano de teste Mentor GA - 4.18.x**

Caso de teste	Baseline	Testador	Data/Hora	Status	Descrição
GA-2984: Cadastrar uma Disciplina (Disciplina 01)	01	TestComplete	17/10/2011 19:18:15	Passou	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 19:46:41	Passou	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 19:51:10	Passou	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 19:56:02	Passou	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 20:04:35	Passou	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 20:15:59	Com Falha	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 20:32:07	Com Falha	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 20:37:00	Com Falha	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	17/10/2011 20:41:07	Com Falha	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	20/10/2011 19:31:56	Com Falha	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina 00)	01	TestComplete	20/10/2011 19:38:22	Com Falha	Retorno da Integração TestComplete/TestLink
GA-3699: Cadastrar uma Disciplina (Disciplina	01	TestComplete	20/10/2011	Com Falha	Retorno da Integração

Figura 24 – Relatório de métricas de casos de testes executados

Na Figura 24 pode ser observado o registro da execução de casos de testes, onde o testador foi o usuário “TestComplete” (usuário padrão definido pelo *plugin*). Estes casos de testes foram executados no plano de teste referente a liberação de versão 4.18, ou seja, a etapa de automatização (utilizando-se do *plugin*) foi respeitada no processo de testes padrão para a liberação de novas funcionalidade em uma versão.

Durante o desenvolvimento deste trabalho foram enfrentados alguns problemas, principalmente relacionados com as limitações da linguagem disponibilizada pelo TestComplete. Foi necessário desenvolver do zero, mecanismos que simulavam a mesma estrutura de Arrays Dinâmicos (haja visto que o TestComplete não tem suporte), pois havia a necessidade de manter as listas de casos de testes e seu monitoramente em alguma estrutura de dados que não houvesse limitação de tamanho(estática).

4 CONCLUSÕES

O desenvolvimento deste trabalho proporcionou para os envolvidos na atividade de codificação, execução e registro de *scripts* de automatização de casos de testes, benefícios reais pois passou a proporcionar mais abstração e facilidade nestas atividades e reduziu de forma significativa o retrabalho operacional dos testadores.

Diante das funcionalidades implementadas já apresentadas, o *plugin* desenvolvido conseguiu atender seu objetivo principal e específico. Através do desenvolvimento de uma nova camada no TestComplete pode-se resolver problemas (principalmente a falta de sincronismo entre o TestLink e o TestComplete) que assolavam a equipe de testes periodicamente. O problema relacionado a execução dos *scripts* também foi resolvido, pois a partir da utilização do *plugin* a chamada dos *scripts* passaram a ser através de um método disponibilizado por este e não mais explicitado no código fonte. Para a codificação dos *scripts* reduziu-se o tempo de desenvolvimento através de controles realizados pelo *plugin* e para se utilizar da técnica de DDT, o *plugin* proporcionou mais facilidade e transparência. Estas constatações foram percebidas no uso do *plugin* junto a empresa onde o autor trabalha.

Com relação às tecnologias empregadas, o conhecimento técnico avançado na ferramenta TestComplete foi um grande aliado na identificação das limitações do próprio TestComplete. A vivência com os processos relacionados as atividades de testes e o uso da ferramenta TestLink também ajudaram na escolha de quais processos deveriam ser integrados.

Pode-se dizer que a conclusão deste trabalho atingiu objetivos pessoais, pois quanto mais se utiliza o conhecimento para reduzir as atividades operacionais que não agregam valor ao processo, mais tempo tem-se para pensar em como se pode evoluir e melhorar nossas atividades relacionadas à atividade de testes.

O *plugin* limitou-se a utilização de *scripts* para automatização de aplicações desktop, pois as estruturas de controle para aplicações desktop e web são bem distintas no TestComplete e mantê-las dentro do *plugin* tornaria-se muito oneroso.

A versão do TestLink também é um limitador. O *plugin* está validado apenas para a versão 1.9 (última versão). Historicamente a cada nova versão do TestLink a sua estrutura de metadados sofre grandes alterações e como o *plugin* utiliza-se desta, possivelmente necessitará de ajustes.

4.1 EXTENSÕES

Baseado em algumas das limitações do trabalho desenvolvido, pode-se destacar algumas extensões deste:

a) possibilitar a execução de testes em aplicações web utilizando o *plugin*, pois atualmente suas estruturas de controles atende apenas aplicações desktop;
b) criação de mecanismos para automatizar também a atividade de automatização de *scripts*. Os *scripts* são criados baseados em casos de testes e estes normalmente são criados baseados no registro de falhas da aplicação por conta do cliente ou mesmo da equipe de teste. O simples fato de gerar a falha (que é o resultado de uma seqüência de passos executados) por si só já deveria montar o *script* de automatização. Desta forma reduziria o retrabalho de simulação da falha e a codificação do *script.*

REFERÊNCIAS BIBLIOGRÁFICAS

AUTOMATEDQA CORPORATION. **Getting Started With TestComplete 8.** [S.l.], 2010. Disponível em: < http://downloads.automatedqa.com/Docs/Getting_Started_With_TestComplete.pdf>. Acesso em: 24 nov. 2011.

CAETANO, Cristiano. **Automação e Gerenciamento de Testes:** Aumentando a Produtividade com as Principais Soluções Open Source e Gratuitas. [S.l.]: [s.n.], 2007.

ENGHOLM JÚNIOR, Hélio. **Engenharia de Software na Prática**, São Paulo : Novatec Editora, 2010.

GARTNER GROUP. **Gartner Groups Reports**. [Stamford] : [s.n.], 2001.

HETZEL, William; FLOW INFORMATICA. **Guia completo ao teste de software**. Rio de Janeiro : Campus, 1987.

FREITAS, Costa Cleziana. **Fundamentos em Testes de Software.** [Brasília] : [s.n.], 2011.

JONES, Capers. **Patterns of Software System Failure and Success**. Boston, MA: International Thompson Computer Press, 1995.

MARCOS, Fronza Adriana. **Ferramenta de Apoio à automatização de testes através do TestComplete para programas desenvolvidos em Delphi**. 2007. 75f. Trabalho de conclusão de Curso (Graduação em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2007.

MYERS, Glenford J. **The Art of software testing**. New York : J. Wiley, 1979.

MOLINARI, Leonardo. **Testes de Software: Produzindo Sistemas Melhores e Mais Confiáveis**, São Paulo: Érica, 2003.

MOREIRA FILHO, Trayahú Rodrigues; RIOS, Emerson. **Projeto & engenharia de software**: teste de software. Rio de Janeiro : Alta Books, 2003.

NIST. **The Economic Impact of Inadequade Infraestruture for Software Testing**. [S.l] : National Institute of Standards and Technology, 2002.

PRADA, Rodrigo. **O que é Plugin?** [S.l], 2008. Disponível em: < http://www.tecmundo.com.br/210-o-que-e-plugin-.htm>. Acesso em: 24 nov. 2011.

PRESSMAN, Roger S. **Engenharia de software**.6. ed. São Paulo : McGraw-Hill, 2006.

RIOS, Emerson; MOREIRA FILHO, Trayahú Rodrigues. **Teste de software**.2. ed. rev. e ampl. Rio de Janeiro : Alta Books, 2006.

TOMELIN, Marcio. **Teste de Software a partir da Ferramenta Visual Test**. 2001. 57f. Trabalho de conclusão de Curso (Graduação em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2001.

APÊNDICE A – Detalhamento dos casos de uso

Este apêndice contém a descrição dos casos de uso representadas na sub-seção 3.2.2.

No Quadro 4 apresenta-se o caso de uso "`Configurar Ambiente`".

Nome do Caso de Uso	UC.01 – Configurar Ambiente
Descrição	Este caso de refere-se a atividade de configuração de informações utilizadas pelo *plugin*.
Ator	*Usuário*
Pré-condição	O usuário deve possuir acesso a ferramenta TestComplete
Cenário principal	1. Usuário acessa o TestComplete 2. Usuário abre um projeto de teste 3. Usuário inicia a execução do TestComplete 4. Sistema identifica que as configurações ainda não estão gravadas 5. Sistema exibe tela de configurações 6. Usuário informa valores referentes ao acesso a base de dados do TestLink 7. Sistema grava informações e fecha tela de configuração 8. Sistema continua com a execução do TestComplete
Cenário alternativo	Nos passos 4, caso o sistema identifique que as configurações já estão gravadas é pulado os passos 5,6,7 até o passo 8.
Pós-condição	*Plugin* configurado

Quadro 4 - Descrição do caso de uso "`Configurar Ambiente`"

No Quadro 5 apresenta-se o caso de uso "`Definir Caso de Teste à Executar`".

Nome do Caso de Uso	UC.02 - Definir Caso de Teste à Executar
Descrição	Todo Caso de Teste que será executado via automatizador obrigatoriamente deverá utilizar o comando 'executaCasosDeTestes()'.
Ator	*Usuário*
Pré-condição	O usuário deve ter realizado a codificação do Caso de Teste que irá executar
Cenário principal	1. Usuário acessa o TestComplete 2. Usuário abre um projeto de teste 3. Usuário acessa *unit* principal do TestComplete 4. Usuário digita comando "executaCasosDeTestes(0001)" 5. Usuário roda a automatização pressionando a tecla 'F5' 6. Sistema identifica no TestLink qual caso de teste deverá ser executado 7. Sistema inclui *script* na lista de casos de teste à executar
Cenário alternativo	No passo 4 do cenário principal, o usuário poderia informar como parâmetro uma suíte e não o código de um caso de teste ,o sistema buscará no TestLink todos os

	casos de testes marcados como ‘automatizado’ na suíte de teste informada ou seja executaria todos as rotinas automatizadas referentes aos casos de testes vinculadas a suíte.
Pós-condição	Caso(s) de Teste(s) pronto para ser(em) executado(s)

Quadro 5 - Descrição do caso de uso “Definir Caso de Teste à Executar”

No Quadro 6 apresenta-se o caso de uso "Codificar um Caso de Teste".

Nome do Caso de Uso	UC.03 - Codificar um Caso de Teste
Descrição	Este caso de refere-se a atividade de codificação de um *script* vinculado a um caso de teste do TestLink.
Ator	*Usuário*
Pré-condição	O usuário deve possuir acesso a ferramenta TestComplete
Cenário principal	9. Usuário acessa o TestComplete 10. Usuário abre um projeto de teste 11. Usuário abre a *unit* referente aos *scripts* de cadastros 12. Usuário cria uma nova *procedure*, chamada “CT_00001” 13. Usuário realiza a chamada da tela que será testada 14. Usuário codifica a entrada de valores definido nos passos do caso de teste 15. Usuário codifica a verificação dos resultados espereados 16. Usuário finaliza *procedure* 17. Sistema na execução da *procedure* codificada irá realizar o monitoramento e registro de falha ou sucesso.
Cenário alternativo	Nos passos 6 e 7 ao invés do usuário codificar manualmente, pode ser utilizado o recurso de gravação das ações do usuário sob a aplicação em teste
Pós-condição	*Script* codificado

Quadro 6 - Descrição do caso de uso “Codificar um Caso de Teste”

No Quadro 7 apresenta-se o caso de uso "Exibir Histórico".

Nome do Caso de Uso	UC.05 - Exibir Histórico
Descrição	Ao término da execução de todos os Casos de Testes o sistema exibirá informações específicas de cada caso de teste e algumas informações de totalizadores.
Ator	*Usuário*
Pré-condição	O usuário deve ter realizado a codificação do Caso de Teste que irá executar
Cenário principal	1. Usuário dispara a execução dos *scripts* 2. Sistema finaliza a execução dos *scripts* 3. Sistema exibe valores utilizados de entrada para cada caso de teste executado 4. Sistema exibe informação de “sucesso” para os casos de testes que não falharam 5. Sistema lista o resumo de todos os casos de testes executados, executados com

	falha e executados com sucesso 6. Sistema exibe tempo total gasto na execução 7. Sistema registra no TestLink os casos de testes executados com todos os valores registrados nos passos anteriores.
Cenário alternativo	No passo 4 do cenário principal, caso a execução do caso de testes falhe, é apresentado ao usuário uma imagem com a exceção retornada da aplicação testada e esta imagem também é anexada no TestLink no histórico da execução do caso de teste.
Pós-condição	Informações referentes à execução do caso de teste

Quadro 7 - Descrição do caso de uso "`Exibir Histórico`"

No Quadro 8 apresenta-se o caso de uso "`Executar Caso de Teste`".

Nome do Caso de Uso	`UC.04 - Executar Caso de Teste`
Descrição	Para cada *script* configurado para ser executado, este caso de uso realiza a execução, monitoramento e finalização do *script*
Ator	*Usuário*
Pré-condição	O usuário deve ter realizado a codificação do Caso de Teste que irá executar
Cenário principal	1. Usuário acessa o TestComplete 2. Usuário abre um projeto de teste 3. Usuário acessa *unit* principal do TestComplete 4. Usuário digita com comando "executaCasosDeTestes(0001)" 5. Usuário roda a automatização pressionando a tecla 'F5' 6. Sistema identifica no TestLink qual caso de teste deverá ser executado 7. Sistema inclui *script* na lista de casos de teste à executar 8. Sistema executa o *script* 9. Sistema monitora a execução do *script* 10. Sistema finaliza *script* e fecha tela ativa 11. Sistema executa próximo *script* da lista
Cenário alternativo	No passo 9 do cenário principal, caso a execução do caso de testes falhe, é mantido a imagem que gerou a falha e este *script* é incluído na lista de falhados
Pós-condição	Todos os *scripts* executados

Quadro 8 - Descrição do caso de uso "`Executar Caso de Teste`"

No Quadro 9 apresenta-se o caso de uso "`Carregar Massa de Dados`".

Nome do Caso de Uso	UC.06 - Carregar Massa de Dados
Descrição	Na execução de um *script* vinculado a um caso de teste que possui uma planilha anexada, este caso de uso faz a importação esta e deixa em modo de consulta para a utilização no *script* que esta sendo executado
Ator	*Usuário*
Pré-condição	O usuário deve ter criado um caso de teste no TestLink, ter marcado-o como

	automatizado e vinculado uma planilha de Excel.
Cenário principal	1. Usuário acessa o TestComplete 2. Usuário abre um projeto de teste 3. Usuário acessa *unit* principal do TestComplete 4. Usuário digita com comando “executaCasosDeTestes(0001)” 5. Usuário roda a automatização pressionando a tecla ‘F5’ 6. Sistema identifica no TestLink qual caso de teste deverá ser executado 7. Sistema inclui *script* na lista de casos de teste à executar 8. Sistema identifica que há uma planilha anexada ao caso de teste no TestLink 9. Sistema abre a planilha e vincula esta a variável global “FPlanilhaDados” 10. Sistema executa o *script* 11. Sistema preenche valores de entrada buscando valores através da variável FPlanilhaDados” 12. Sistema finaliza *script*
Cenário alternativo	No passo 11, caso não haver referencia a variável “FPlanilhaDados” dentro do *script* em execução, o sistema ignora o passo.
Pós-condição	Valores de entrada do *Script* carregados pela planilha

Quadro 9 - Descrição do caso de uso “`Carregar Massa de Dados`”