Universidade Federal de Goiás
Instituto de Informática

Adriana Rocha Vidal

Teste Funcional Sistemático Estendido: Uma Contribuição na Aplicação de Critérios de Teste Caixa-Preta

Goiânia
2011

Universidade Federal de Goiás

Instituto de Informática

Autorização para Publicação de Dissertação em Formato Eletrônico

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG , entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Teste Funcional Sistemático Estendido: Uma Contribuição na Aplicação de Critérios de Teste Caixa-Preta

Autor(a): Adriana Rocha Vidal

Goiânia, 19 de Abril de 2011 .

Adriana Rocha Vidal – Autor

Dr. Auri Marcelo Rizzo Vincenzi – Orientador

Dr. Plínio de Sá Leitão Júnior – Co-Orientador

Adriana Rocha Vidal

Teste Funcional Sistemático Estendido: Uma Contribuição na Aplicação de Critérios de Teste Caixa-Preta

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Mestrado em Ciências da Computação.

Área de concentração: Sistema de Informação.

Orientador: Prof. Dr. Auri Marcelo Rizzo Vincenzi

Co-Orientador: Prof. Dr. Plínio de Sá Leitão Júnior

Goiânia
2011

Adriana Rocha Vidal

Teste Funcional Sistemático Estendido: Uma Contribuição na Aplicação de Critérios de Teste Caixa-Preta

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Mestrado em Ciências da Computação, aprovada em 19 de Abril de 2011 , pela Banca Examinadora constituída pelos professores:

Prof. Dr. Auri Marcelo Rizzo Vincenzi
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Plínio de Sá Leitão Júnior
Instituto de Informática – UFG

Prof. Dr. Marcos Lordello Chaim
Universidade de São Paulo – USP

Prof. Dr. Edmundo Sérgio Spoto
Universidade Federal de Goiás – UFG

Adriana Rocha Vidal

Graduada em Ciência da Computação na UFG - Universidade Federal de Goiás. Durante sua graduação, foi pesquisadora do CNPq em projeto desenvolvido em conjunto ao IEL e que consistia em definir e implantar um processo de testes no contexto de uma determinada empresa. Pós-graduada em Gestão de Tecnologia da Informação, seu projeto final de curso, envolvia inteligência artificial e Teste de software, intitulado: Redes Neurais: Um Estudo Prático em Teste de Software. Atualmente, mestranda em Ciências da Computação, participou do grupo de certificação de PAF-ECF. E, como gestora da SEFAZ- GO, é responsável por definir, implantar e manter o processo de testes da equipe de desenvolvimento.

Aos meus pais por terem proporcionado ininterrupto estímulo, apoio e incentivo.

Ao meu querido Wesley, por acreditar em mim, em momentos que eu mesma duvidei.

Agradecimentos

Agradeço a Deus por Sua graça sobre minha vida. A Ele toda honra e glória!

Ao meu querido Wesley, pela dedicação, cuidado, paciência e por sempre se preocupar com a "nossa" dissertação. Ao meu precioso marido, meus agradecimentos por lutar pela concretização desse projeto.

Aos meus pais por me ensinarem o valor do conhecimento. Por serem constantes fontes de inspiração e incentivo.

À Cristina e Ana Júlia, pelos sorrisos, pelos abraços, pelo carinho, pela vida compartilhada.

Aos meus sogros e cunhados pelo apoio tão sincero e necessário.

Ao professor Auri, por, apesar de seu escasso tempo, me orientar não somente em assuntos acadêmicos, mas em conselhos que levarei para vida. A esse exemplo de professor e pessoa, minha gratidão.

Ao professor Edmundo e Plínio pelos ensinamentos diários.

À Simeon e à InforPlace por disponibilizarem seus softwares agregando maior credibilidade a esse trabalho.

Aos amigos do mestrado e de toda a vida: Elisabete, Elisângela, Renata, Marcelo, André, Luiz, Fabiana, Patrícia, Sofia, Carine, Renan, Rogério, Danillo e Gilmar pelos momentos cômicos, pelos dias de estudo, pela sonhos compartilhados.

Aos alunos do PAF-ECF, especialmente à aluna Jackeline por embasar esse trabalho com sua dedicação.

Aos amigos de sempre por festejarem as vitórias, mas principalmente, por apoiarem em momentos difíceis.

Aos professores do Instituto de Informática por se dedicarem com maestria ao crescimento de seus alunos.

Aos funcionários do Instituto de Informática, em especial ao Edir, Berenice, Ricardo pela amizade e disposição sempre tão presentes.

Aos colegas da LG Sistemas, por me proporcionar viver em meio profissional as teorias da engenharia de software.

Aos colegas da SEFAZ-GO por possibilitarem a conclusão desse trabalho.

Aos torcedores, muito obrigada!!

...mas como está escrito: Nem olhos viram, nem ouvidos ouviram, nem jamais penetrou em coração humano o que Deus tem preparado para aqueles que O amam.

1 Coríntios 2:9,
Bíblia.

Resumo

ROCHA, Adriana Vidal. **Teste Funcional Sistemático Estendido: Uma Contribuição na Aplicação de Critérios de Teste Caixa-Preta**. Goiânia, 2011 . 136p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

A construção de um software envolve um processo composto de atividades e métodos. Mesmo seguindo tais atividades e utilizando os métodos propostos, um produto infiel aos requisitos funcionais e não funcionais pode ser gerado, não correspondendo às funcionalidades esperadas. Para amenizar tais problemas, a atividade de teste visa a assegurar tanto a construção do produto correto quanto a sua correta construção. Por ser uma atividade considerada onerosa, pesquisas para reduzir os custos da aplicação dos testes são realizadas. Este trabalho se enquadra nesse contexto, objetivando melhorar a seleção de casos de testes, aumentando assim, a qualidade de produtos de software e o desempenho de roteiros de teste. É interessante ressaltar que, roteiro de teste é um artefato fundamental do processo de testes e é constituído por casos de testes que, por definição, executam uma funcionalidade particular do programa ou verificam a adequação do produto em relação aos requisitos especificados. Uma vez que a qualidade dos casos de testes selecionados impacta fortemente na qualidade do produto final, este trabalho apresenta o Teste Funcional Sistemático Estendido (TFSE) como forma de sistematizar a elaboração e seleção de casos de testes, adotando critérios da técnica de teste funcional para essa finalidade. Um sistema web e um roteiro de teste utilizado em certificações foram avaliados utilizando o TFSE visando a demonstrar a aplicabilidade do mesmo e as possíveis contribuições de sua utilização em termos de detecção de defeitos. Os resultados obtidos são promissores uma vez que a sistematização, aumenta o número de dados de teste selecionados, melhora a capacidade de detecção dos defeitos, e permitir justificar o por quê da seleção de determinado dado de teste com base em critérios funcionais.

Palavras–chave

Engenharia de Software, Processo de Testes, Documentação de Testes, Teste Funcional.

Abstract

ROCHA, Adriana Vidal. **Systematic Functional Test Extended: A Contribution to the Application of Criteria Black Box Testing**. Goiânia, 2011 . 136p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Building software involves a process composed of activities and methods. Even following these activities and using the proposed methods the resultant product may have some deviation with respect to its functional and nonfunctional requirements, not corresponding to the expected features. To minimize such problems, the test activity aims to ensure both the construction of the correct product and its correct construction. Since testing is considered a costly activity, research are conducted aiming at to make it feasible. This work fits in this context, in order improve the selection of test cases, thus increasing the quality of software products and the performance of testing guideline. It is interesting to note that, testing guideline is a fundamental artifact of the testing process and consists of test cases that, by definition, execute a particular functionality of the program or check the suitability of the product over its specified requirements. Since the quality of the selected test cases have a great impact on the quality of the final product, this work introduces the Extended Systematic Functional Test (ESFT) as a way to systematize the development and selection of test cases based on functional testing. A web system and a testing guideline used in certification were assessed using the ESFT in order to demonstrate the applicability and possible contributions of its use in terms of defect detection. The results are promising since the systematization, increases the number of selected test data, improves the detection of defects, and allow to justify why a particular test data is selected based on functional criteria.

Keywords

Software Engineering, Testing Process, Testing Documentation, Functional Testing

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

Produtos de software são elementos essenciais na atualidade, independente da atividade e do local utilizados, são de suma importância para o pleno exercício das mais variadas tarefas. Assim, sistemas computacionais devem possuir como características essenciais: qualidade e alta produtividade, tanto do ponto de vista do processo de produção, quanto dos produtos gerados.

Nesse contexto, a Engenharia de Software visa a definir padrões que auxiliam na elaboração de métodos e técnicas de modo que haja garantia da qualidade de software. Porém, sendo o engano uma característica inerente ao ser humano, apesar dessas definições de atividades, técnicas e métodos auxiliarem no processo de desenvolvimento do software, defeitos podem estar presentes no produto final. Algumas atividades foram sugeridas e agrupadas sob o nome Garantia de Qualidade de Software, denominadas atividades de Verificação e Validação (V&V). Dentro delas, a atividade de teste é uma das mais empregadas (MALDONADO et al., 2004).

Esta atividade é frequentemente entendida como uma investigação empírica objetivando adquirir informações sobre a qualidade do produto de software (REPASI, 2009). A Norma NBR ISO/IEC 12119 (IEEE, 1998b) define teste como uma operação técnica que consiste na constatação de uma ou mais características de um dado produto, processo, serviço, de acordo com um procedimento especificado. Segundo Maldonado et al. (2004), a atividade de teste consiste de uma análise dinâmica do produto e é uma atividade relevante para a identificação de defeitos que persistem. Para uma melhor compreensão, no contexto desse trabalho, será utilizada a terminologia abaixo (IEEE, 2002):

- Defeito (*fault*): Passo, processo ou definição de dados incorretos.
- Erro (*error*): A manifestação do defeito, ou seja, um estado inconsistente ou inesperado do programa.
- Falha (*failure*): O erro pode levar a uma falha – o resultado obtido é diferente do esperado.

A partir dessas definições, têm-se que testes provocam a ocorrência de falhas, que por sua vez, são consequências de defeitos, inseridos no código em decorrência de enganos cometidos por programadores. Assim, o objetivo geral da atividades de testes é criar testes capazes de descobrir no mínimo defeitos essenciais no software (REPASI, 2009), melhorando assim, a sua qualidade. Sabe-se que a aplicação de um processo de teste, em conjunto a outras atividades de V&V (inspeção, revisão, dentre outras) (MALDONADO et al., 2004), além de denotar um dos elementos para fornecer evidências da confiabilidade e qualidade do software, vem cobrir deficiências geradas por metodologias ineficientes de construção de software (*code and fix*). Desse modo, a grande demanda de clientes não satisfeitos com software de baixa qualidade e fortes pressões para desenvolver software de boa qualidade em curto espaço de tempo são algumas das forças motrizes para a utilização de técnicas e critérios de teste de software.

Diversas são as técnicas e critérios de testes existentes, porém, no contexto deste trabalho, serão adotados critérios de testes da técnica funcional ou caixa-preta, devido à não dependência do código – em muitas situações o testador não tem acesso à implementação – e à facilidade de entendimento destes critérios.

1.1 Motivação e Objetivo

Segundo Itkonen et al. (2009), estudos práticos na indústria e relatórios profissionais mostram que testes rigorosos e baseados em casos de testes completamente documentados não são comuns na indústria e que, devido a esse fato, muitos pesquisadores e profissionais têm enfatizado o papel da experiência e habilidade na atividade de teste. Porém, segundo o mesmo artigo, testadores demonstram ter necessidade de empregar critérios de teste mesmo durante a aplicação de testes exploratórios - abordagem de teste de software que é descrita de forma concisa como a aprendizagem simultânea, projeto e execução de teste (WHITTAKER, 2009).

É evidente que a experiência, habilidade e até mesmo a motivação do testador são fundamentais no processo de testes, porém, ao sistematizar esse processo, adotando técnicas e a devida documentação, a probabilidade de encontrar defeitos se torna maior (MILLS, 1996).

Buscando aprimorar a forma de se empregar critérios de teste funcionais e torná-los mais efetivos em detectar defeitos, Linkman et al. (2003) propôs o chamado Teste Funcional Sistemático (*Systematic Functional Testing*) que consiste de uma forma organizada e sistemática de se empregar critérios de testes caixa-preta de modo a maximizar o número de defeitos detectados e contribuir de forma significativa

para a execução completa ou quase completa das instruções que implementam determinado programa, ou seja, atingir uma boa cobertura de código.

O presente trabalho insere-se nesse contexto e visa tanto a contribuir com a sistematização do teste funcional, revisitando o trabalho de Linkman et al. (2003), quanto com a documentação de testes e identificação de ferramentas que apoiem tal atividade, buscando facilitar a geração de dados de teste por parte do testador e a posterior documentação desses testes.

Desse modo, o objetivo do presente trabalho é a proposição do Teste Funcional Sistemático Estendido (TFSE), ampliando os conceitos do Teste Funcional Sistemático (LINKMAN et al., 2003), e a avaliação do mesmo em dois cenários diferentes demonstrando a possibilidade de emprego do TFSE tanto na geração de conjuntos de teste quanto na avaliação de conjuntos existentes.

Um estudo inicial de algumas ferramentas para apoiar a aplicação do TFSE é apresentado visando a contribuir com a futura construção de um ambiente integrado para a documentação e execução de testes tendo como base os critérios definidos pelo TFSE.

1.2 Metodologia

1. Embasamento Teórico:
   Durante a revisão bibliográfica foram pesquisadas diversas fontes de forma a contemplar dentre os diversos contextos, abordagens utilizadas na validação de roteiro de testes, porém, foi verificada a falta de trabalhos que empregam teste funcional na avaliação desse tipo de documentação. Nesse sentido, o embasamento teórico consistirá na investigação, de maneira genérica, de conceitos e metodologias relativas ao processo de teste e aos critérios funcionais;
2. Proposição do TFSE:
   A proposta do TFSE consistirá em revisitar e aprimorar o trabalho de Linkman et al. (2003), além de elaborar formas alternativas de visualização dessa técnica;
3. Adoção do TFSE em softwares implementados por empresas de desenvolvimento:
   Nesse passo, o TFSE será aplicado em contexto de empresas desenvolvedoras de software para avaliar suas contribuições. Serão apresentados dois estudos de casos distintos, um visando à melhoria da qualidade de determinado software e o outro propondo melhorias para o roteiro utilizado em certificação de software. Além disso, será utilizado um padrão de documentação com o objetivo de automatizar a documentação e execução dos casos de testes e de agilizar o processo de testes.

1.3 Organização do Trabalho

Considerando a motivação, objetivos e a metodologia apresentada acima, o restante desta dissertação de mestrado está organizada da seguinte forma:

- o Capítulo 2 apresenta os fundamentos de Engenharia de Software e Testes de Software. Assim, são evidenciadas as técnicas e os critérios de testes, as fases de testes, processos de desenvolvimento de software e processo de testes;
- no Capítulo 3 são apresentados os conceitos e critérios de Teste Funcional. Também serão apresentados os fundamentos do Teste Funcional Sistemático (TFS);
- o Capítulo 4 discute as extensões ao TFS e a sistematização dos passos para a definição do Teste Funcional Sistemático Estendido (TFSE), os algoritmos propostos e a concepção gráfica desses algoritmos;
- no Capítulo 5 são apresentados os estudos de casos que foram adotados para a validação do TFSE. São explorados os contextos de cada empresa, de cada software, dos órgãos certificadores e os resultados obtidos com a aplicação do TFSE;
- no Capítulo 6 são apresentadas as conclusões do trabalho realizado, suas contribuições e perspectivas de desdobramentos para trabalhos futuros.

CAPÍTULO 2

Teste de Software

Segundo Tian (2005), a expectativa das pessoas em relação à qualidade do software pode ser definida em duas vertentes:

- O software deve fazer o que se propõe a fazer.
- O software deve realizar tarefas específicas corretamente.

Assim, temos que, para os usuários de software, qualidade se refere ao software executar o que foi definido e de forma correta e satisfatória.

Porém, garantir qualidade no contexto atual – sistemas complexos, que possuem milhares de linhas de código e muitas vezes envolvendo grandes equipes e longos prazos – se tornou demasiadamente trabalhoso. Brooks (1987), em seu artigo "*No Silver Bullet*", afirma que não há bala de prata para os problemas encontrados no desenvolvimento de software, ou seja, não há uma solução poderosa para a complexidade, tamanho, qualidade e outros problemas de engenharia de software, devido a requisitos e restrições que um software deve satisfazer. Assim, de acordo com Tian (2005), lidar com problemas que impactam negativamente clientes e usuários e tentar gerenciar e melhorar a qualidade de software é um fato da vida de pessoas envolvidas no desenvolvimento, gestão, marketing e suporte operacional de muitos sistemas modernos. A Engenharia de Software atua nesse contexto, fornecendo técnicas, métodos, metodologias e ferramentas de apoio, e visa a contribuir para o desenvolvimento de produtos de software de alta qualidade maximizando o uso dos recursos e minimizando os custos de desenvolvimento.

2.1 A Engenharia de Software

Segundo Repasi (2009), desde 1960, a programação de computador tem evoluído para uma disciplina de engenharia. Foi definido, numa conferência em 1968, que o objetivo da Engenharia de Software era o estabelecimento e o uso de princípios de engenharia, a fim de obter softwares confiáveis, eficientes e economicamente viá-

veis. Portanto, uma metodologia antiga aplicada à construção de novas tecnologias, objetivando aprimorar a qualidade dos produtos desenvolvidos.

Formalmente definida no SWEBOK (2004), a Engenharia de Software é conceituada como a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção do software, ou seja, é a aplicação da engenharia ao software. E assim, todo estudo relacionado a essa definição, também é definido como Engenharia de Software. Ainda, segundo Sommerville (2007), Engenharia de Software é uma disciplina da engenharia que está relacionada a todos os aspectos da produção de software, desde os primeiros estágios da especificação do sistema até a sua manutenção. Dentre os diversos princípios da engenharia pode-se citar: aplicar garantia da qualidade em produtos, processos, componentes, materiais e o estabelecimento de ciclo de vida e modelos de desenvolvimento (REPASI, 2009). Analisando alguns desses conceitos, no contexto de desenvolvimento de software, têm-se as seguintes definições:

- **Produto** "são programas de computadores que quando executados fornecem a função e o desempenho desejados, estruturas de dados que permitem aos programas manipular adequadamente a informação e documentos que descrevem a operação e o uso dos programas" (PRESSMAN, 2006).
- **Processo** é um conceito chave na engenharia de software, definido como o adesivo que mantém unidas as camadas de tecnologia e permite o desenvolvimento racional e oportuno de software para computador, comparando-o a um roteiro que auxilia na criação, em tempo hábil, de produtos de alta qualidade (PRESSMAN, 2006). Segundo Sommerville (2007), processo é um conjunto de atividades que fundamentam o desenvolvimento e a evolução do software. A ISO/IEC (2004) define processo como o "conjunto de atividades interrelacionadas, que transforma entradas em saídas".
- **Ciclo de Vida de Software** consiste no período de tempo que inicia quando o software é concebido e finaliza quando o software não está mais disponível para uso. Tipicamente inclui as fases de conceitos, requisitos, projeto, implementação, teste, instalação, operação, manutenção e, algumas vezes, a fase de aposentadoria do software (IEEE, 2002).
- **Modelo de software** é uma descrição simplificada do processo e deve incluir atividades que são partes do processo de software, produtos de software e papéis de pessoas envolvidas na engenharia de software (SOMMERVILLE, 2007). Pressman (2006) afirma que modelos definem um conjunto distinto de ações, atividades, marcos e produtos de trabalho que são necessários para a engenharia de alta qualidade, providenciando, assim, estabilidade, controle e organização para atividades que, se não forem controladas, tornam-

se completamente caóticas. São exemplos de modelos: Modelo Sequencial Linear (Cascata), Modelo de Prototipagem, Modelo RAD (*Rapid Application Development*), Modelo Incremental, Modelo Espiral, Modelo de Métodos Formais (PRESSMAN, 2006; SOMMERVILLE, 2007).

- **Garantia da Qualidade** é definido como o conjunto de atividades planejadas e sistemáticas, implementadas no sistema da qualidade e demonstradas como necessárias, para prover confiança adequada de que uma entidade atenderá aos requisitos para a qualidade especificados (ISO/IEC, 2004).

Nesse contexto, ressaltando que a Garantia da Qualidade é uma peça fundamental do processo de desenvolvimento de software e que, como definido anteriormente, é composta por um conjunto de atividades, Maldonado et al. (2004) definem que essas atividades, denominadadas verificação e validação, são inseridas em algumas etapas do processo de desenvolvimento. Conceitualmente, a atividade de verificação realiza inspeções/revisões sobre os produtos gerados pelas diversas etapas do processo, enquanto que, a validação avalia se o sistema atende aos requisitos do projeto (usuário). Dentre as atividades de verificação e validação, o teste é uma das mais utilizadas por permitir avaliar o aspecto comportamental do produto. São várias as definições de teste na literatura:

- um processo de engenharia concorrente ao processo de ciclo de vida do software, que faz uso e mantém artefatos de teste usados para medir e melhorar a qualidade do produto de software sendo testado (CRAIG; JASKIEL, 2002).
- processo de analisar um item do software a fim de detectar as diferenças entre condições existentes e necessárias e avaliar as características do item de software (IEEE, 1998a).
- processo de executar um programa ou sistema com a intenção de encontrar defeitos (MOREIRA; RIOS, 2003);
- 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 (MOREIRA; RIOS, 2003);
- processo de execução de um programa com a finalidade de encontrar um defeito (PRESSMAN, 2006);
- processo, ou uma série de processos, designados a assegurar que o código faça o que é proposto e não execute algo fora do escopo especificado (MYERS et al., 2004).
- atividade desempenhada para avaliar qualidade do produto de software e melhorá-la, identificando defeitos e problemas (SWEBOK, 2004).

Em todas essas definições, fica evidente que teste de software trata-se de um processo fundamentado na análise dinâmica – exige execução – do software em teste, objetivando encontrar defeitos, melhorando assim, a qualidade do produto. Porém, empresas de desenvolvimento muitas vezes por limitação de tempo e recursos não reconhecem o valor da fase de teste de software, ou seja, não há a equiparação entre os benefícios do teste e seu custo. De acordo com Pressman (2006), a fase de teste ocupa, normalmente, 40% do tempo planejado para um projeto e um defeito descoberto tardiamente em um sistema provoca um acréscimo de 60% nos custos do projeto. Esses valores explicitam a necessidade dos defeitos serem encontrados o mais breve possível. Atualmente, empresas gastam muito tempo e dinheiro na manutenção de software, investimento que seria economizado, caso um processo de teste de software fosse bem implementado e conduzido em paralelo com o processo de desenvolvimento.

Deve-se observar que a atividade de teste não prova a ausência de defeitos, apenas a existência dos mesmos (MALDONADO et al., 2004). Logo, bons casos de teste são aqueles que provocam falhas no sistema até então não manifestadas. Uma forma de possibilitar o aumento da garantia da eficácia dos testes é executá-los de forma metódica, por meio de processos definidos e com pontos de checagem obrigatórios. Na próxima seção serão apresentadas características essenciais e peculiaridades do processo de teste.

2.2 O Processo de Teste de Software

Como exposto anteriormente, um processo de desenvolvimento de software será otimizado se acompanhado por um processo de teste de software. Como ilustrado na Figura 2.1, todas as atividades do processo de desenvolvimento do software podem ser acompanhadas por etapas do processo de teste. Assim, todas as atividades do desenvolvimento do software serão monitoradas e a probabilidade de existirem defeitos no produto final será minimizada, se comparada a um produto que passou por uma série de testes *ad-hoc* exercidos pelo próprio programador.

No entanto, para que um processo de teste seja bem sucedido, alguns pontos devem ser analisados, conforme destacam Moreira e Rios (2003):

- O processo de teste deve ser tratado pelas organizações como um processo: Verifica-se que, a maioria das empresas desenvolvedoras de software não dá a devida importância à atividade de teste, sendo esta uma fase informal, conduzida sem metodologia e com funções não definidas, se confundindo com a própria gestão do processo de desenvolvimento.

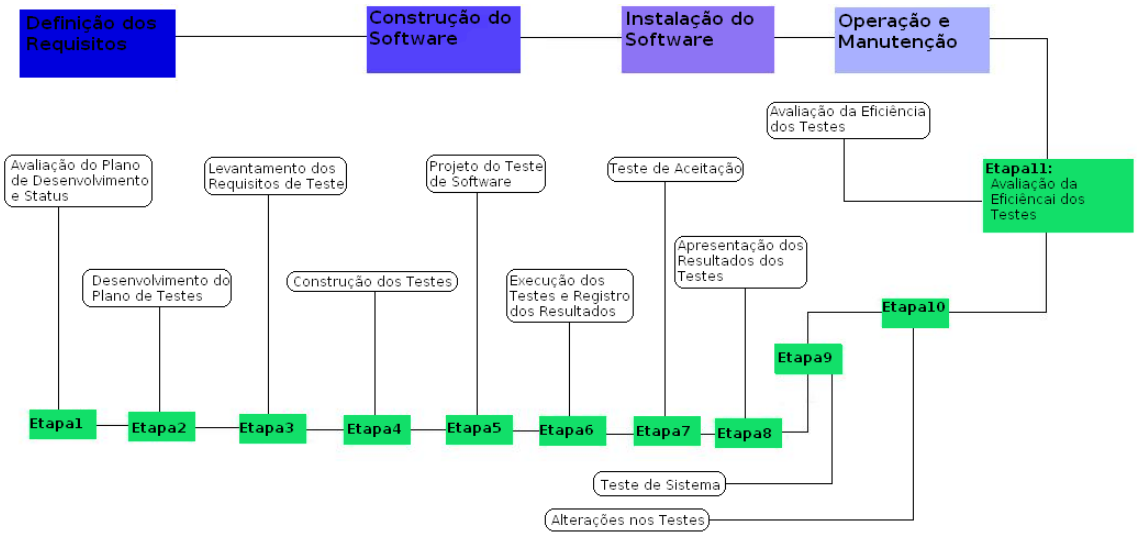


Figura 2.1: *Etapas do Processo de Teste de Software (PERRY, 2000)*

- Os testes devem abranger o sistema completamente:
  É evidente que, se os testes forem incompletos durante o desenvolvimento, a probabilidade de ocorrerem problemas após sua implantação é notória.
- A abordagem de testes deve ser adequada a novas tecnologias:
  Investir na reciclagem e/ou treinamento do pessoal técnico de testes, de forma a adequar os procedimentos de testes às novas tecnologias.
- A estrutura organizacional para testes deve ser modificada:
  Em geral, quase todas as etapas do processo de teste são realizadas pelos desenvolvedores. É interessante ressaltar que, essa característica não deve ser necessariamente eliminada, porém, em alguns estágios do processo de testes, há a necessidade de que pessoas qualificadas e com o perfil adequado à execução dos testes passem a avaliar o produto.
- Ferramentas de automação de testes devem ser usadas:
  A automação agiliza o processo de testes e diminui os custos na etapa de manutenção. Além disso, há alguns tipos de testes, de desempenho por exemplo, que são inviáveis de serem executados manualmente.
- Os artefatos produzidos durante o processo de testes devem ser documentados:
  Cada fase do processo de teste deve ser devidamente documentada, pois, além de facilitar a futura automação das atividades de teste, estreita a relação entre os processos de teste e de desenvolvimento e ainda, fornece estrutura para organizar, conduzir e gerenciar o processo de teste. Além disso, as informações

dos testes são de grande ajuda para a realização de outras atividades, tais como a atividade de depuração[1].

Além dos pontos levantados no item anterior, em experiência passada, na implantação de um processo de teste em uma empresa de desenvolvimento de software de Goiânia, observou-se a carência de um padrão de documentação de requisitos e outros artefatos que facilitasse a geração de casos de teste por parte do testador. Na empresa em questão, embora todas as unidades a serem desenvolvidas fossem documentadas, tal documentação não contemplava aspectos relevantes aos testadores, sendo difícil a extração de casos de testes de tais documentos (ROCHA, 2005).

A documentação também é responsável pela identificação de itens de testes específicos, de critérios de testes que serão utilizados, de atividades de testes, de definir quem será encarregado de cada atividade e quais os riscos de ocorrerem falhas no produto (KANER et al., 1999). Nesse contexto, há diversos modelos que definem metodologias, práticas e artefatos relacionados ao processo de teste, tais como: MPS.BR (SOFTEX, 2010) e CMMI (CMMI, 2010). Além desses modelos, há padrões específicos para o processo de testes. Por exemplo, o Padrão (IEEE, 1998a) define um conjunto de documentos básicos de testes de software. São referenciados oito documentos que cobrem desde a preparação dos testes até o registro do resultado da execução, providenciando assim, artefatos que abrangem todo processo de testes.

Um ponto chave tanto do processo de testes quanto da documentação é o Caso de Teste. Conforme o SWEBOK (2004), teste de software consiste na verificação dinâmica de um conjunto finito de casos de testes, adequadamente selecionados. Um caso de teste é um artefato fundamental no processo de teste de software e definido como um conjunto de entradas de testes, condições de execução e resultados esperados desenvolvido para um objetivo particular, tal como exercitar um caminho específico do programa ou verificar a adequação com uma especificação de requisitos (IEEE, 2002; IEEE, 1998a). De uma forma simples, Naik e Tripathy (2008) definem caso de teste como um par <entrada, saída esperada>. É possível ainda que o caso de teste inclua outros itens como pré-condições e ordem de execução (teste em cascata ou independente) (COPELAND, 2004), além dos passos para a sua execução, os quais determinam a sequência de ações a serem tomadas pelo testador para executar o teste.

[1]Testar implica em executar o produto em teste com a intenção de mostrar que tal produto falha durante sua execução para algum valor do domínio de entrada. Depurar significa identificar em qual ponto do código do produto em teste encontra-se o comando defeituoso (que está fazendo o produto falhar).

Na implantação de um processo, objetivando sistematizar a execução dos testes, diversas técnicas podem ser utilizadas. Assim, dentre os vários critérios propostos, encontram-se os que pertencem às técnicas de testes estruturais, funcionais e baseada em defeitos.

A principal diferença entre esses critérios está na origem da informação que será avaliada e que servirá de base para se gerar os requisitos de testes. Pode-se descrever cada uma dessas técnicas da seguinte forma:

- Funcional: Também conhecida como técnica caixa preta, pois se preocupam com a função que o programa desempenha, considerando irrelevantes a maneira como a função foi implementada. Essa técnica propõe examinar as características do domínio de entrada na tentativa de descobrir formas de derivar partições ou classes de equivalência (subdomínios) e que podem ser classificados em três classes, de fronteira, não de fronteira e especiais (PRESSMAN, 2006). Dentre os diversos critérios dessa técnicas podem ser citados: Particionamento de Equivalência, Análise do Valor Limite, Tabela de Decisão, dentre outros (PRESSMAN, 2006).
- Estrutural: também conhecida como caixa branca, difere da técnica funcional pois enfoca a implementação e a estrutura interna do programa. O objetivo do teste estrutural é descobrir dados de teste que garantam cobertura suficiente de todas as estruturas presentes no código do produto em teste (PRESSMAN, 2006). Por cobertura do código entende-se a porcentagem de comandos ou instruções do programa que foram efetivamente executadas durante os testes. Idealmente, um requisito mínimo de teste seria exigir que o conjunto de teste construído executasse ao menos uma vez cada comando do produto em teste. Dentre os critérios dessa técnica podem ser citados aqueles Baseados em Fluxo de Controle, Baseados em Fluxo de Dados (SWEBOK, 2004) e Baseados na Complexidade (DELAMARO et al., 2007). Em cada uma dessas categorias existem diversos critérios de teste, formando uma hierarquia, denominada relação de inclusão, que é caracterizada pela capacidade de determinado critério exigir um conjunto de teste que, além de satisfazer os requisitos exigidos por ele, também satisfaz os requisitos de teste dos critérios abaixo dele na hierarquia (RAPPS; WEYUKER, 1982).
- Baseada em defeitos: para auxiliar no teste de determinado produto, essa técnica utiliza informações sobre defeitos recorrentes no desenvolvimento de software e sobre os tipos de defeitos que se deseja descobrir (PRESSMAN, 2006). Um dos critérios desta técnica é o teste de mutação, na qual, são inseridas pequenas modificações no programa em teste – os mutantes. Esses mutantes são utilizados para determinar a eficácia dos testes, se o conjunto

de testes "matarem" todos os mutantes, os testes executados são eficientes em demonstrar que o programa não contém o conjunto de defeitos representado pelos mutantes (DEMILLO et al., 1978; DELAMARO et al., 2007).

- Baseada na experiência e intuição do engenheiro de software: Nessa técnica, são conhecidos o teste *ad-hoc* e o teste exploratório. O teste *ad-hoc* é, provavelmente, um dos mais utilizados (SWEBOK, 2004) e adota a derivação de testes a partir da habilidade, intuição e experiência do testador. Enquanto que, no teste exploratório, os testes são definidos, projetados, executados e modificados simultaneamente. Nesse caso, os testes são derivados a partir de várias fontes: observação do comportamento do produto durante o teste, familiaridade com a aplicação, a plataforma, as falhas do processo, o tipo de possíveis defeitos e falhas, os riscos associados com uma particularidade do produto, metáforas que procuram sistematizar a forma como a exploração do software é realizada (WHITTAKER, 2009), dentre outros.

Para que a adequação entre o processo de desenvolvimento e processo de testes seja bem sucedida, a melhoria dos processos de engenharia de software é fundamental para produzir software com qualidade, dentro do prazo e custos aceitáveis. Assim, podem ser citados como modelos mais utilizados para gerar essa melhoria dos processos: CMM (PAULK, 1993), CMMI (CMMI, 2010), ISO/IEC-15504 (SPICE) (ISO/IEC, 2005) e ISO/IEC-12207 (ISO/IEC, 2004). Todos esses modelos, direta ou indiretamente, fazem referência ao processo de teste de software em algum de seus níveis de maturidade.

2.3 Fases de Teste

Nessa seção, serão descritos os estágios ou fases de teste. No entanto, a melhor estratégia de testes irá falhar caso uma série de aspectos prevalentes não seja considerada. Pressman (2006) alega que esses aspectos são:

- ter especificação dos requisitos do produto de um modo quantificável antes do início dos testes;
- enunciar os objetivos específicos do teste em termos mensuráveis;
- desenvolver um perfil para cada categoria de usuário;
- construir um software capaz de diagnosticar certas classes de defeitos;
- utilizar revisões técnicas formais efetivas como filtro, antes da execução dos testes;
- avaliar estratégias de teste e casos de teste por meio de revisões técnicas;
- aperfeiçoar continuamente o processo de teste.

Após validados esses aspectos, podem ser estabelecidas fases ou estratégias de testes, tais como Teste de Unidade, Teste de Integração, Teste de Sistema, Teste de Aceitação. Lemos (2004) afirma que: “O teste frequentemente responde por mais esforço de projeto que qualquer outra atividade de engenharia de software e começa no varejo e progride para o atacado”, ou seja, as atividades de teste são executadas a partir de um módulo do programa e finalizadas no sistema como um todo.

Segundo Pressman (2006), uma estratégia de testes de software pode ser vista no contexto espiral. O teste de unidade começa no centro da espiral e se concentra em cada unidade (componente) do software como implementada no código-fonte. O teste progride movendo-se para fora ao longo da espiral para o teste de integração, em que o foco fica no projeto e construção da arquitetura do software. No último ciclo da espiral encontram-se os testes de sistema e de aceitação. O primeiro verifica a integração do produto com o seu ambiente e o segundo se o produto em questão atende as exigências do usuário.

Na Figura 2.2, estão ilustrados de forma sucinta os processos de teste e desenvolvimento de software, explicitando que essas estratégias de testes, promovem o intercâmbio entre os dois processos. Nas seções seguintes, essas estratégias são melhor abordadas.

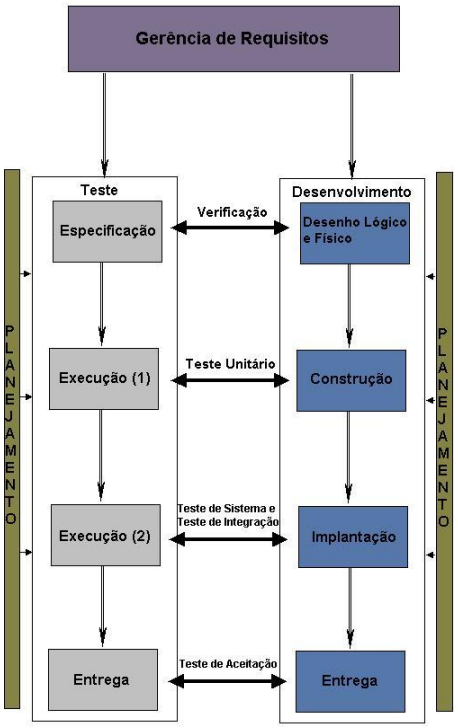


Figura 2.2: *Integração entre os Processos de Desenvolvimento e Testes (MOREIRA; RIOS, 2003)*

2.3.1 Teste de Unidade

É um estágio do processo, em que testes são aplicados nos menores componentes de código criados, visando a garantir que estes atendem as especificações, em termos de características e funcionalidade. Os testes unitários verificam o funcionamento de um pedaço do sistema ou software isoladamente ou partes que possam ser testados separadamente, podendo, inclusive, ser um programa ou um componente. Na grande maioria dos casos, estes testes são realizados pelos desenvolvedores.

Em sua execução são usadas descrições de projeto em nível de componente como guia para derivar casos de teste e caminhos dentro da unidade são testados para descobrir defeitos dentro dos limites do módulo.

De acordo com Pressman (2006), essa fase é composta por testes em vários níveis do módulo, assim, são considerados: a interface, as estruturas lógicas, as condições-limite, os caminhos independentes e os caminhos de tratamento de erros presentes na unidade.

Nesse contexto, a interface da unidade é testada para comprovar que a informação flui adequadamente. A estrutura de dados local é examinada para garantir

que os dados armazenados temporariamente mantêm sua integridade durante todos os passos de execução. As condições limite são testadas para garantir que o módulo opera adequadamente nos limiares estabelecidos para limitar ou restringir o processamento. Todos os caminhos independentes (caminhos básicos) ao longo da estrutura de controle são exercitados para garantir que todos os comandos de um módulo sejam executados pelo menos uma vez. Finalmente, todos os caminhos de tratamento de erros são testados visando a assegurar que as condições anormais de execução sejam manipuladas adequadamente.

Teste de limite é a última tarefa do passo de teste de unidade. O software normalmente falha nos seus limites. Ou seja, frequentemente ocorre erro quando o *n-ésimo* elemento de uma matriz n-dimensional é processada, quando a *i-ésima* repetição de um ciclo com i passos é invocada, quando o valor máximo ou mínimo permissível é encontrado. Pressman (2006) afirma que, casos de testes que exercitam as estruturas de dados e de fluxo de controle com valores de dados imediatamente abaixo, sobre e imediatamente acima do máximo e do mínimo permitidos têm grande probabilidade de levar o software a falhar.

2.3.2 Teste de Integração

O teste de integração é uma atividade sistemática aplicada durante a integração da estrutura do programa visando a descobrir erros associados às interfaces entre os módulos; o objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto (LEMOS, 2004).

Estes testes podem ser feitos de forma incremental, onde cada módulo ou componente é incluído sequencialmente até que todos os casos de testes sejam testados.

Podem ser citadas como estratégias deste nível: Bottom-up, Top-Down, Teste fumaça, Funcional, Teste de Regressão (PRESSMAN, 2006), Big-bang e Fluxo de Dados (MOREIRA; RIOS, 2003).

2.3.3 Teste de Sistema

O teste de sistema visa a identificar erros de funções e características de desempenho que não estejam de acordo com a especificação. Assim, o objetivo desta fase de teste é verificar se todos os elementos do sistema foram adequadamente integrados e executam as funções a eles alocadas.

Na realidade, essa fase de teste é uma série de diferentes testes com a finalidade de exercitar por completo o sistema. Dentre os testes executados nessa fase, podem ser citados: Teste de Recuperação (PRESSMAN, 2006), Teste

de Segurança (PRESSMAN, 2006), Teste de Estresse (PRESSMAN, 2006), Teste de Desempenho (PRESSMAN, 2006) e Teste end-to-end (MOREIRA; RIOS, 2003).

2.3.4 Teste de Aceitação

Nessa fase, os testes são realizados pelos usuários. Representam os testes finais de execução do sistema, e visam a verificar se a solução atende aos objetivos do negócio e seus requisitos, no que diz respeito à funcionalidade e usabilidade, antes da utilização no ambiente de produção.

Apesar de ser responsabilidade dos usuários, os testes são conduzidos com suporte da equipe de testes e projeto.

2.4 Considerações Finais

O presente trabalho está inserido na fase de teste de sistema e teste de aceitação e utiliza os critérios da técnica de teste funcional. Essa técnica será adotada pois é mais simples de ser compreendida pela equipe de teste e pode trazer bons resultados se conduzida de forma sistemática. E conforme explicitada na próxima seção, é independente de código, tornando a execução de testes possível em situações em que este não é acessível.

CAPÍTULO 3

Testes Funcionais de Software

A técnica de Teste Funcional é um tipo de teste dinâmico de software, conhecida também como Teste Caixa Preta. Classificado como dinâmico, pois o software é executado enquanto testado e Caixa Preta devido à peculiaridade de que o analista de teste não possui o conhecimento da estrutura interna da implementação do produto em teste.

Assim, o teste caixa preta é uma estratégia de testes baseada somente nos requisitos e especificações, e diferentemente do teste caixa branca não requer conhecimento de caminhos internos, estruturas ou implementações do software em execução (COPELAND, 2004).

Segundo Everett e Jr. (2007), o objetivo do teste funcional é validar o comportamento do software em relação às funcionalidades de negócios documentadas e às especificações.

É interessante ressaltar que o teste funcional pode ser aplicado a todos os níveis do processo de teste - desde o teste de unidade ao teste de aceitação.

A desvantagem desse critério é que, para cobrir todos os possíveis caminhos especificados, um exaustivo número de casos de testes deve ser criado. Myers et al. (2004) mostra que é impossível testar todas as possibilidades de entrada de um programa. Assim, na realidade, apenas um subconjunto desses casos de testes são executados, limitando a abrangência e o poder de detecção de defeitos dos testes.

Considerando que, em geral, teste exaustivo é inviável para qualquer empresa, o maior objetivo então é otimizar o rendimento do investimento em testes, maximizando assim, o número de defeitos encontrados por um número finito de casos de testes (MYERS et al., 2004).

Assim, a formalização do teste caixa preta direciona o testador a escolher subconjuntos de testes que, teoricamente, serão eficientes e efetivos na descoberta de defeitos. Segundo (COPELAND, 2004) estes subconjuntos encontrarão mais defeitos do que o mesmo número de casos de testes criados aleatoriamente, aumentando, portanto, o retorno sobre o investimento em teste.

Neste contexto, teste funcional é uma técnica de teste em que o código é considerado como uma caixa preta, ou seja, o analista de teste desconhece a implementação em teste. E que, por essa peculiaridade, possui limitações, tais como:

1. São dependentes de uma boa especificação de requisitos. Assim, se a especificação de requisitos é incompleta ou falha, não será possível derivar bons conjuntos de casos de testes, limitando portanto, a ação da atividade de testes.
2. Não permitem comprovar que partes importantes do código tenham sido executadas. Considerando que, o analista de teste não tem visibilidade da estrutura interna do produto em teste, não é possível garantir que todas as partes da implementação foram cobertas e nem quais partes o teste está cobrindo. Assim, nesse aspecto, esses critérios se tornam ineficientes.

Entretanto, mesmo com tais limitações, esses critérios são bastante utilizados, pois são compreendidos naturalmente e de fácil aplicação. Além disso, podem ser empregados em todas as fases de teste e são independentes de paradigma de programação.

Nas próximas seções, os principais critérios desta técnica serão analisados.

3.1 Testes por Classe de Equivalência

Esse critério de teste objetiva reduzir o número de casos de teste para um nível gerenciável mantendo, ainda, uma cobertura razoável de testes (COPELAND, 2004).

Segundo Patton (2005) Classe de equivalência ou Particionamento de equivalência é o processo de, sistematicamente, reduzir o enorme (ou infinito) conjunto de possíveis casos de testes para um conjunto menor, porém tão efetivo quanto o conjunto inicial.

Assim, segundo Pressman (2006) este critério divide o domínio de entrada em duas classes distintas, válida e inválida, sendo que, o número de casos de testes deve ser limitado, selecionando-se aqueles que, em hipótese, representam toda a classe. (MYERS et al., 2004) define duas propriedades que esse critério deve cumprir:

1. Reduzir o número de casos de testes que devem ser desenvolvidos para alcançar um objetivo pré-definido de teste.
2. Abranger um grande conjunto de outros possíveis casos de testes.

De forma geral, além de limitar de forma eficiente a quantidade de testes, essa técnica considera que todos os elementos de determinada partição são equivalentes, ou seja, se um caso de teste de uma classe de equivalência descobre (ou não)

um defeito, todos os outros casos de testes dessa classe também se comportam de forma equivalente.

As classes de testes podem ser definidas analisando o domínio de entrada segundo os seguintes critérios (PRESSMAN, 2006):

- Especificação de um valor numérico específico;
- Exigência de um determinado intervalo de valores;
- Especificação de um membro de um conjunto de valores relacionados;
- Definição de uma condição booleana.

Ainda sobre o projeto de casos de testes adotando o critério de particionamento em classes de equivalência, alguns autores sugerem os seguintes passos (COPELAND, 2004) (MYERS et al., 2004):

1. Identificar as classes de equivalência. São obtidas analisando as possíveis condições de entrada e particionando-as em dois ou mais grupos. Normalmente, dois tipos de classes de equivalência são identificadas: classe de equivalência válida - na qual os dados de entrada válidos são testados - e classe de equivalência inválida - dados inválidos são considerados. É importante ressaltar que, dessa forma, essa técnica garante que além das condições válidas, o testador deverá cobrir os casos inválidos e as situações inesperadas.
2. Criar um caso de teste para cada classe de equivalência.

   Outra abordagem para o uso dessa técnica é considerar os dados de saída ao invés dos dados de entrada. Que consiste em dividir as saídas em classes de equivalência e determinar quais os possíveis valores de entrada para os respectivos valores das classes de saída (COPELAND, 2004).

Finalmente, sobre o particionamento de equivalência, segundo (PATTON, 2005), este critério pode ser muito subjetivo. Testadores diferentes criam classes de equivalência diferentes. E assim, nesse contexto, é complexo garantir que as classes de equivalência obtidas são realmente eficientes. Além disso, como dependem de uma boa especificação, não garantem a execução de partes críticas da aplicação.

Assim, para complementar essa técnica, há o critério Análise do Valor Limite, descrito na próxima seção.

3.2 Análise do Valor Limite

Segundo (MYERS et al., 2004), os casos de testes que exploram condições de fronteira das classes de equivalência tem um retorno maior na detecção de defeitos

do que os casos de testes que não exploram. (BURNSTEIN, 2003) diz que, com a experiência, testadores percebem que muitos defeitos são encontrados diretamente sobre, acima e abaixo dos limite das classes de equivalência. E, (NAIK; TRIPATHY, 2008) afirmam que, na prática, projetistas e programadores tendem a ignorar as condições do limite e consequentemente, os defeitos estão concentrados próximos às fronteiras das classes de equivalência. Assim, o critério análise do valor limite complementa o critério Particionamento de Equivalência, auxiliando o testador a selecionar valores limítrofes das classes.

(HUTCHESON, 2003) afirma que a análise do valor limite é a técnica de testes mais importante e fundamental no desenvolvimento de software. Segundo ele, os valores-limite incluem valores máximos, mínimos, de entrada e saída do software, valores característicos e valores de erro. Assim, a idéia geral dessa técnica se baseia na premissa de que, se defeitos não forem encontrados utilizando esses valores especiais, então, também não o serão utilizando quaisquer outros valores das respectivas classes.

Conforme (MYERS et al., 2004), os critérios Particionamento em Classe de Equivalência e Análise do Valor Limite se diferenciam em:

- Ao invés de selecionar qualquer elemento de uma classe de equivalência, de forma representativa, a análise do valor-limite exige que um ou mais elementos sejam selecionados de forma que cada extremidade da classe de equivalência seja objeto de teste.

Segundo (COPELAND, 2004), os passos necessários para utilizar esse critério são simples, detalhados no exemplo a seguir:

1. Identificar as classes de equivalência;
2. Identificar os valores limites de cada classe;
3. Criar casos de teste para cada um dos valores limites, valores superiores e inferiores ao limites.

3.3 Teste Funcional Sistemático

Definida como uma abordagem sistêmica de testes funcionais, essa técnica propõe combinações de alguns critérios de testes funcionais objetivando obter, do código em testes, um índice de cobertura da implementação tão alto quanto do teste estrutural. Em (LINKMAN et al., 2003), os autores, utilizando teste de mutação, mostram que essa técnica matou 100%dos mutantes criados, enquanto que o índice utilizando outros critérios de testes funcionais foi consideravelmente

inferior. O objetivo dessa técnica é associar o benefício da técnica de teste funcional - independência da implementação - ao bom desempenho da técnica de teste estrutural - maior cobertura do código em teste. Nesse contexto, além dos critérios abordados na técnica de teste funcional clássica, outros pontos, como forma de sistematizar a execução do teste, devem ser considerados, conforme itens apontados abaixo:

1. Necessariamente, devem ser criados pelo menos dois casos de testes com cada partição de equivalência, minimizando assim, o problema de erros co-incidentes que mascaram falhas.
2. Para valores numéricos discretos, devem ser considerados os dados de entrada e saída e casos de testes são gerados para cada classe.
3. Para intervalos de valores numéricos, também devem ser considerados os dados de entrada e saída, e os casos de testes são derivados contemplando os limites e um valor interno da cada intervalo.
4. Devem ser gerados casos de testes com valores diferentes do esperado e com casos especiais, tanto para os dados de entrada quanto para os dados de saída.
5. Casos de Testes que exploram valores ilegais são necessários para mostrar que o software em teste trata desvios do caminho de sucesso. Assim, considerar valores fora dos limites máximo e mínimo de um intervalo, por exemplo, são casos de testes relevantes.
6. Ao criar casos de testes que contemplem números reais, deve ser observada uma especificidade, esse tipo de dado não possui um limite exato. Normalmente, números reais são inscritos em base de dez, armazenados em base de dois e finalmente recuperados em base de dez, gerando assim, dados inconsistentes. Assim, devem ser criados casos de testes que contemplem essa situação, adotando um intervalo de exatidão de erro, com diferentes valores-limite. Além disso, devem ser criados casos de testes que considerem valores reais muito pequenos e o valor zero.
7. Para valores de intervalo variáveis - que dependem de uma ou mais variáveis, devem ser criados casos de testes que contemplem todas as possibilidades de combinação dos possíveis valores.
8. Quando a informação a ser testada envolve vetor ou matriz de dados, devem ser criados casos de testes que avaliem os dados: tamanho da matriz e os dados da matriz. O tamanho da matriz deve ser testado, em todas as dimensões e em todas as possíveis combinações, no seu tamanho mínimo, máximo e valores intermediários. Para os valores dos dados do *array*, devem ser consideradas as questões levantadas anteriormente.

9. Para textos ou *string* de dados, é necessário validar a variação do tamanho e a validade de cada caracter, considerando o alfabeto apenas de caracteres ou o alfanumérico ou ainda, apenas o de pontuação.

Como apresentado em seu trabalho, (LINKMAN et al., 2003) mostra que considerando esses pontos durante a criação ou execução dos casos de testes, a possibilidade de atingir um grau de cobertura de código é bem maior do que a utilização de testes funcionais ou testes randômicos. Resumidamente, a Tabela 3.1 apresenta o conjunto de casos de testes gerados, a quantidade de casos de testes e os casos de teste efetivos, i.e., casos de testes que mataram pelo menos um mutante considerando a ordem de execução.

Tabela 3.1: *Conjuntos de Teste Randômicos e Funcionais.*

Conjunto de Teste	**Quantidade de Casos de Teste**	**Casos de Teste Efetivos**
TS_{SFT}	76	21
TS_{PB_1}	21	17
TS_{PB_2}	15	13
TS_{PB_3}	21	17
TS_{PB_4}	14	13
TS_{RA_1}	10	5
TS_{RA_2}	20	9
TS_{RA_3}	30	16
TS_{RA_4}	40	22
TS_{RA_5}	50	23
TS_{RA_6}	60	27
TS_{RA_7}	70	29

Os conjuntos de Teste foram gerados da seguinte forma:

- TS_{SFT}: Utilizando teste funcional sistemático;
- TS_{PB_1}, TS_{PB_2}, TS_{PB_3}, TS_{PB_4}: Por estudantes que usaram os critérios particionamento de equivalência e análise do valor limite;
- TS_{RA_1}, TS_{RA_2}, TS_{RA_3}, TS_{RA_4}, TS_{RA_5}, TS_{RA_6}, TS_{RA_7}: Gerados randomicamente, onde cada conjunto possui, respectivamente, 10, 20, 30, 40, 50, 60, 70 casos de teste.

Por exemplo, segundo a Tabela 3.1, analisando o TS_{SFT} tem-se que, foram gerados 76 casos de teste para cobrir as partições válidas e inválidas e desses 76 casos de teste, 21 mataram pelo menos um mutante quando executados.

Para ilustrar o custo do teste de mutação foi utilizado o programa *CAL*, cujo pode ser chamado com nenhum, um ou dois parâmetros. No primeiro caso, é apresentado o calendário correspondente ao mês e ano atuais, no segundo caso apresenta o calendário completo do ano correspondente e no último caso, apresenta o calendário do mês e do ano solicitado. Para este aplicativo foram gerados 4624 mutantes, sendo que 335 são mutantes equivalente, i.e., a modificação feita no programa original para criar um mutante não altera a função implementada (VINCENZI, 1998). Considerando que o *score* de mutação é obtido através da relação entre a quantidade de mutantes mortos e quantidade de mutantes não equivalentes, ele pode variar entre zero e um, e quanto maior for seu valor, mais adequado é o conjunto de casos de teste para o programa em testes. Nesse sentido, a Tabela 3.2 mostra, para cada conjunto de testes, o número de mutantes vivos, a porcentagem de mutantes vivos em relação ao total de mutantes e o *scores* de mutação. Por exemplo, pode ser observado que TS_{SFT} é o único conjunto de testes que revela todas as falhas geradas por todos os mutantes. E então, observa-se que todos os mutantes não equivalentes são mortos e o *score* de mutação é igual a 1. Assim, é interessante ressaltar que, TS_{SFT} apresentou o melhor score possível de ser obtido.

Tabela 3.2: *Cobertura dos conjuntos de Teste.*

Conjunto de Teste	**Quantidade de vivos**	**Porcentagem de vivos**	**Score de mutação**
TS_{SFT}	0	0	1.000000
TS_{PB_1}	371	8.02	0.913500
TS_{PB_2}	74	1.60	0.982747
TS_{PB_3}	124	2.68	0.971089
TS_{PB_4}	293	6.34	0.931686
TS_{RA_1}	1,875	40.55	0.563242
TS_{RA_2}	558	12.07	0.870021
TS_{RA_3}	419	9.06	0.902399
TS_{RA_4}	348	7.53	0.918938
TS_{RA_5}	311	6.73	0.927557
TS_{RA_6}	296	6.40	0.931051
TS_{RA_7}	69	1.49	0.983927

Nesse sentido, como forma de sistematizar a abordagem proposta no presente trabalho e assim aumentar a probabilidade de produzir um software com maior qualidade, serão adotados os conceitos de teste funcional sistemático.

3.4 Considerações Finais

Foram abordados os elementos essenciais da literatura que baseiam os conceitos explorados neste trabalho. De forma específica o TFS mostrou-se mais eficiente do que testes funcionais e aleatórios na revelação de programas mutantes. Com base nessa proposição, o presente trabalho visa estender os conceitos de TFS de forma a aplicá-lo no contexto prático de testes de sistemas desenvolvidos por empresas de software e na validação de roteiros utilizados em certificação.

CAPÍTULO 4

Extensão do Teste Funcional Sistemático

Conforme apresentado no Capítulo 3, existem vários critérios de testes funcionais que podem ser empregados pelo testador para a verificação e validação de um produto de software. Mesmo com suas limitações, os critérios funcionais podem ser aplicados indistintamente em todas as fases de teste e são os critérios de teste mais empregados durante a fase do teste de sistema e de aceitação. Devido à capacidade de abstração dos critérios funcionais, representando o produto em teste como uma caixa-preta da qual só se conhece a entrada e a saída, uma crítica que se faz a esses critérios é a impossibilidade de, quando aplicados, garantir que partes essenciais ou críticas do produto em teste sejam executadas.

Entretanto, mesmo com tais limitações, alguns experimentos iniciais (LINKMAN et al., 2003) mostram que, se a implementação for consistente com sua especificação, o emprego do teste funcional de forma sistemática pode resultar em uma excelente cobertura do código do produto em teste, mesmo que isso seja obtido de forma indireta.

Desse modo, para a proposição da estratégia de testes, serão adotados como critérios primários, os definidos por (LINKMAN et al., 2003) e, a partir desses, derivados outros com base nos tipos de dados existentes. Assim, será considerada a informação de que os critérios adotados são abstraídos de condições de entrada e saída, que, tipicamente, representam um valor numérico específico (inteiro ou real), um intervalo de valores, um conjunto de valores relacionados, dentre outras.

Objetivando tornar a abordagem mais visual e intuitiva ao testador, adotou-se como linguagem de descrição a linguagem algorítmica e a representação por fluxograma. Segundo (FORBELLONE; EBERSPäCHER, 2005), algoritmo pode ser definido como uma sequência de passos que visam a atingir um objetivo bem definido. Sendo, portanto, uma linha de raciocínio que pode ser descrita de diversas maneiras, de forma gráfica ou textual. Assim, fluxograma é uma das possíveis formas de representação gráfica de um algoritmo. (FORBELLONE; EBERSPäCHER, 2005) afirma que as formas gráficas são mais puras por serem mais fiéis ao raciocínio original, substituindo um grande número de palavras por convenções de desenhos.

Nas próximas seções, será apresentado o Teste Funcional Sistemático Estendido (TFSE) em linguagem algorítmica e em fluxograma.

4.1 Sistematização dos Passos

Como foi exposto no Capítulo 2, a Engenharia de Software define e propõe práticas e métodos que auxiliam no desenvolvimento de software de qualidade. Dentre essas boas práticas, documentar os artefatos gerados em cada atividade do processo é importante pois, todas as demais atividades desenvolvidas no decorrer do processo de desenvolvimento, incluindo as atividades de teste e manutenção, são dependentes da existência de uma boa documentação (DELAMARO et al., 2007). Dentre os documentos que podem ser gerados têm-se, por exemplo: o modelo do negócio, a especificação de requisitos, a especificação do projeto, a documentação dos casos de teste, dentre outros. Porém, uma série de fatores faz com que essa documentação seja negligenciada por grande parte das empresas e nem sempre pode-se contar que a mesma esteja disponível e atualizada. Nesse contexto, este trabalho utiliza como fonte para a criação dos casos de testes a especificação do sistema – quando existe –, um especialista do sistema e do negócio e as telas do software, quando a documentação não é consistente o suficiente para embasar a criação dos casos de testes.

Quando os testes são baseados numa especificação bem definida e há a garantia de que o código retrata o que está proposto e não há partes do código não documentadas, o teste funcional sistemático garante uma boa cobertura de código e um grande número de defeitos podem ser encontrados. E nos casos em que não há especificação de requisitos ou de negócio ou há uma documentação escassa ou desatualizada, não é possível garantir que essa técnica de teste cubra uma parte considerável do código. Mesmo em um contexto desfavorável, a aplicação do TFSE permite que testes relacionados aos dados das telas sejam executados de forma sistematizada e coerente com os recursos disponíveis para tal. É possível, ainda, completar a construção dos casos de testes verificando a funcionalidade e coletando informações de um especialista (desenvolvedor, analista de negócio, analista de requisitos, etc), avaliando, assim, a funcionalidade como um todo.

Independente da fonte adotada, o primeiro passo que faz parte do TFSE é a necessidade de identificar as classes de equivalência, conforme o critério Particionamento em Classe de Equivalência descrito na Seção 3.1. Estas classes são, normalmente, abstraídas a partir da especificação do software e, por ser um processo subjetivo, depende da interpretação e experiência do testador. Outro fator

importante durante a definição das classes de equivalência é considerar os domínios tanto de entrada de dados quanto os de saída, e observar os seguintes pontos:

- Tipo do dado
  - Com base na informação sobre o tipo do dado sendo considerado é possível avaliar qual a capacidade de armazenamento interno de valores para o tipo considerado. Por exemplo, se for um dado do tipo inteiro, é possível explorar valores inteiros de diferentes tamanhos, considerando limites máximos e mínimos assumindo que, internamente, esse inteiro seja armazenado em uma variável de dois, quatro ou mais bytes, dependente do ambiente/linguagem de desenvolvimento.
- Valores válidos para o dado
  - Mesmo considerando o tipo do dado, dentro do contexto da aplicação, pode haver uma limitação de quais valores ou intervalo de valores são aceitos para o dado em questão. Nesse sentido, é importante identificar esse subconjunto do domínio de entrada do dado para criar as partições válidas para o dado em questão, ou seja, aquelas que representam valores válidos e que seriam aceitos pelo programa em teste. Por exemplo, considerando dados do tipo inteiro, se esse tipo corresponder a idade de uma pessoa, ele poderia estar limitado aos inteiros positivos menores que 120.
- Valores inválidos dentro do domínio do tipo de dado
  - Uma vez definidos os domínios dos valores válidos, todos os demais valores do tipo considerado que ficarem fora desse domínio são considerados inválidos. Por exemplo, considerando o campo da idade, um valor inteiro negativo ou maior que 120 é um valor inválido para esse campo.
- Cardinalidade dos espaços de entrada e de saída (quantidade de dados);
  - Principalmente quando se trata de programas que recebem parâmetros de entrada, a quantidade de parâmetros em si também define uma classe de equivalência e, desse modo, deve ser usada para auxiliar na criação de casos de teste. A mesma observação vale para a saída e, portanto, também deve ser considerada.
- Valores pertinentes a casos especiais do tipo de dado em questão;
  - Em geral, dentro do conjunto de valores válidos ou inválidos, podem existir valores com tratamentos especiais e, nesses casos, subclasses de equivalência são criadas agrupando esses valores especiais de modo que o testador seja obrigado a criar testes considerando esses subconjuntos.

- Valores diferentes do aceito para o tipo de dado em questão.
  - Outros tipos de dados também devem ser explorados como possíveis valores de entrada. É comum ao usuário teclar valores inválidos por engano. Por exemplo, considerando um campo texto, o usuário que deseja entrar com o "C" o faria teclando, por exemplo, Shift+C, mas cometeu um engano e teclou Ctrl+C. Observe que nesse caso, um valor fora do domínio desejado foi fornecido. Outro exemplo seria fornecer caracteres a campos numéricos. Valores nulos e em branco também devem ser utilizados.

Assim, considerando que o critério Particionamento de Equivalência tenta identificar subconjuntos do domínio de entrada/saída que estejam relacionados com funcionalidades específicas do produto em testes, a definição das classes de equivalência é feita da seguinte forma:

- Classes são definidas para valores válidos, conforme as partições identificadas na especificação do software;
- Classes são também estabelecidas para os valores inválidos, pois dados dessas classes serão explorados durante o teste;
- Cada cardinalidade dos espaços de entrada e saída representa uma classe de equivalência.

Após a definição das classes de equivalência, o TFSE exige que sejam selecionados dados de testes representativos destas classes, conforme as seguintes diretrizes:

1. Derivar pelo menos dois casos de teste para as partições de dados pertinentes a valores válidos e inválidos, conforme Algoritmos 4.1 a 4.10;
2. Derivar casos de testes que cubram os valores limite de cada partição de dados válidos e inválidos, aplicando-se o critério Análise do Valor Limite conforme definido na Seção 3.2;
3. Gerar pelo menos um dado de teste para cada cardinalidade dos espaços de entrada e saída;
4. Gerar pelo menos um dado de teste para cada caso especial e para valores de tipos de dados diferentes do especificado.

4.2 Algoritmos Auxiliares

A seguir, os Algoritmos 4.1 a 4.10, descritos nas Seções 4.2.1 a 4.2.10, orientam a seleção de dados de teste das classes de equivalência dos valores válidos e inválidos para os domínios de entrada e de saída.

4.2.1 Considerando dados numéricos

Para dados numéricos, considere o Algoritmo 4.1. Valores numéricos podem ser classificados em valores discretos e intervalo de valores. Sabe-se que, valores discretos são características mensuráveis e podem assumir um número finito ou infinito contável de valores. No contexto de testes, será adotado que os valores discretos representam um conjunto finito contável, ou seja, representam valores específicos e é possível criar casos de testes para todos os valores definidos.

- Valores Específicos
  No contexto da aplicação do processo de testes, é interessante cobrir todos os aspectos determinados para um software. Assim, quando há valores específicos definidos, é necessário criar casos de testes que cubram todos os valores, sempre que possível. A linha 2 representa a criação de casos de testes que contemplem valores válidos e a linha 3 os casos de testes que contemplem valores inválidos. Na abordagem original do TFS, não há essa definição, porém, no contexto deste trabalho, sugerimos que os casos de testes que abordem as situações de insucesso, também sejam criados com base nos limites de cada valor específico.

  Em um exemplo prático, tem-se a seguinte situação: Uma escola deseja que determinadas atividades sejam realizadas pelos estudantes segundo suas notas. Assim, estudantes que obtiveram as notas 5, 7 e 9 devem fazer um trabalho da disciplina, aqueles que obtiveram as notas 6 e 8 devem resolver uma lista de exercícios e os estudantes nota 10 vão ao passeio organizado pela escola.

  Aplicando o Algoritmo 4.1, referente aos dados específicos, foram geradas as Tabelas abaixo. Na Tabela 4.1, estão representados os casos de testes referentes à classe válida, observe que todos os valores específicos foram considerados conforme definido no Algoritmo. Tabela 4.2 estão representados os dados referentes às classes inválidas, conforme as linhas 2 e 3 do Algoritmo 4.1.

  É interessante ressaltar que, ou há um erro de especificação ou a escola não trabalha com valores não inteiros, porém, os casos de testes contemplaram todas as possíveis situações de sucesso e insucesso.
- Intervalo de Valores
  No contexto desse trabalho, intervalo de valores é um conjunto sequencial de dados contidos entre dois valores determinados. Assim, serão considerados, os intervalos finitos com início e fim definidos. A diferença entre os valores específicos e o intervalo de valores é que, os valores específicos são contáveis

Tabela 4.1: *Valores numéricos específicos: classes válidas*

Casos de Testes	**Parâmetro de Entrada**	Saída Esperada
CT1	5	Trabalho
CT2	7	Trabalho
CT3	9	Trabalho
CT4	6	Lista
CT5	8	Lista
CT6	10	Passeio

Tabela 4.2: *Valores numéricos específicos: classes inválidas*

Casos de Testes	**Parâmetro de Entrada**	Saída Esperada
CT8	4	Nada definido
CT9	4,5	Nada definido
CT10	5,5	Nada definido
CT11	6,5	Nada definido
CT12	7,5	Nada definido
CT13	8,5	Nada definido
CT14	9,5	Nada definido
CT15	11	Nada definido

e pré-definidos, portanto, é viável, em geral, testar todas os valores definidos. Enquanto que, para valores definidos em forma de intervalo, pode ser difícil testar cada elemento do intervalo, sendo mais comum a escolha de valores específicos, conforme definido nas linhas de 5 a 23 do Algoritmo 4.1.

- Números Reais

  Segundo Linkman et al. (2003), há problemas especiais quando o teste envolve números reais pois, a precisão que são armazenados, normalmente, é diferente da precisão que são informados, gerando, potencialmente, dados inconsistentes. Por exemplo, o número decimal 0.1 não é representado com precisão pelo sistema binário dos computadores, de maneira que 12 * 0.1 é diferente de 1,2. Em Java, os seguintes valores foram obtidos. Assim, realizando um pequeno experimento, na linguagem C, são comparados os números 1.1999999999999999556 e 1.2000000000000001776. Assim, os testes de limites para os números reais podem não ser exatos, mas ainda devem ser considerados, adotando uma faixa aceitável de precisão de erro. Além disso, valores muito pequenos, próximos de zero também devem ser selecionados. A geração de dados de teste para números reais é apresentada nas linhas de 24 a 27 do Algoritmo 4.1.

Algoritmo 4.1: Tipo de dado numérico.

```
1  se (valores específicos) então
2    Criar CT's que contemple cada valor específico /*Válida*/
3    Criar dois ou mais CT's para valores diferentes e próximos dos valores
         específicos /*Inválida*/
4  fimse
5  se (intervalo de valores) então
6    se (intervalo for variável) então
7      /* Intervalo variável significa que o valor de uma determinada variável(x)
           depende de outra variável(y). */
8      Criar CT que contemple x=y=0 /*(Válida)*/
9      Criar dois ou mais CT's que contemple x=0<y /*Válida*/
10     Criar dois ou mais CT's que contemple 0<x=y /*Válida*/
11     Criar dois ou mais CT's que contemple 0<x<y /*Válida*/
12     Criar dois ou mais CT's que contemple y=0<x /*Inválida*/
13     Criar dois ou mais CT's que contemple 0<y<x /*Inválida*/
14     Criar dois ou mais CT's que contemple x<0 /*Inválida*/
15     Criar dois ou mais CT's que contemple y<0 /*Inválida*/
16   fimse
17   Criar CT's que contemplem cada valor limite do intervalo /*Válida*/
18   Criar dois ou mais CT's com valores superiores ao limite mínimo /*Válida*/
19   Criar dois ou mais CT's com valores inferiores ao limite máximo /*Válida*/
20   Criar dois ou mais CT's com valores intermediários ao intervalo /*Válida*/
21   Criar dois ou mais CT's com valores inferiores ao limite mínimo /*Inválida*/
22   Criar dois ou mais CT's com valores superiores ao limite máximo /*Inválida*/
23 fimse
24 se (números reais) então
25   Criar CT's para valores muito próximos de zero
26   Criar CT's para valores muito próximos dos limites estabelecidos - adotando o
         critério de precisão definido
27 fimse
```

4.2.2 Considerando dados Booleanos

Dados booleanos é um tipo de dado primitivo que possui dois valores - Verdadeiro ou Falso. O Algoritmo 4.2 contempla esse tipo de dado, criando casos de testes que abordam as duas possíveis situações.

Algoritmo 4.2: Tipo de dado booleano.

```
1  Criar CT para o caso de verdadeiro /*Válida*/
2  Criar CT para o caso de falso /*Válida*/
```

Em relação às condições booleanas é interessante ressaltar que, quando a condição booleana for formada por operadores "E" ou "OU", deverão ser criados CT's que contemplem as seguintes situações:

- Se há condições lógicas relacionadas por operadores "E", deverão ser criados CT's que verifiquem as possibilidades de combinação entre essas condições. Por exemplo: Se há três condições lógicas interligadas por operadores "E", deverão ser criados CT's para os casos de: todas as condições falsas, para uma condição verdadeira, para duas condições verdadeiras, para todas as condições verdadeiras.
- Se há condições lógicas relacionadas por operadores "OU", também deverão ser criados CT's que verifiquem as possíveis combinações entre essas condições.

Lembrando que para o operador “OU”, a única possibilidade de ocorrer falso é quando todas as condições forem falsas. Então, por exemplo, se há 3 condições lógicas ligadas por operadores “OU”, deverão ser criados CT's que abordem as seguintes situações: as três condições são falsas, todas as condições verdadeiras, um condição é verdadeira e as demais falsas.

4.2.3 Considerando a cardinalidade da entrada e saída

Há alguns programas em que a quantidade de dados de entrada e saída é variável e que, todas as possíveis situações devem ser consideradas. Um exemplo a ser considerado é o programa Cal (STATMATH, 2010) definido anteriormente.

No Algoritmo 4.3, são definidos os passos que auxiliam na criação de casos de testes para esse contexto.

Algoritmo 4.3: Quantidade de elementos de entrada e saída do software.

```
1  se (há um limite de quantidade de dados de entrada e saída) então
2    Criar CT's que contemple as quantidades aceitáveis /*Válida)*/
3    Criar dois ou mais CT's que contemplem quantidades acima da mínima aceitável
         /*Válida*/
4    Criar dois ou mais CT's que contemplem quantidades abaixo da máxima aceitável
         /*Válida*/
5    Criar dois ou mais CT's que contemplem quantidades abaixo da mínima aceitável
         /*Inválida*/
6    Criar dois ou mais CT's que contemplem quantidades acima da máxima aceitável
         /*Inválida*/
7  fimse
```

4.2.4 Considerando dados estruturados homogêneos (Matriz)

Segundo Linkman et al. (2003), quando os testes vão tratar de estrutura do tipo matriz, há o problema do tamanho da estrutura e dos dados armazenados. Devem ser descritos casos de testes que contemplem os possíveis valores para o tamanho da estrutura bem como para os tipos de dados armazenados. Portanto, devem ser realizados testes para os tamanhos mínimos, máximos e intermediários, em todas as combinações de dimensões. Para simplificar o teste, pode-se usar subestruturas da matriz (como uma linha, por exemplo), como unidade de teste. Porém, a matriz deve ser testada, primeiro, como uma estrutura simples, depois como uma coleção de subestruturas, e cada subestrutura deve ser testada independentemente.

O Algoritmo 4.4 resume os passos necessários para a criação de casos de teste referentes a esse tipo de dado.

Algoritmo 4.4: Tipo de dado Matriz.

```
1 Criar CT's que verifiquem o tamanho da matriz.
2 Criar CT's que avaliem os possíveis valores dos elementos do matriz.
3 Aplicar demais algoritmos considerando o tipo de dado armazenado.
```

4.2.5 Considerando dados strings

Ressaltando que string ou cadeia de caracteres é uma sequência ordenada de caracteres selecionados a partir de um conjunto pré-definido, então é possível haver strings compostas apenas de caracteres numéricos, alfanuméricos, sinais de pontuação ou ainda uma combinação desses. Desse modo, para esse tipo de dado, devem ser realizados testes que contemplem as seguintes situações:

- Ausência de caracteres;
- Número de caracteres abaixo do mínimo permitido.
- Número mínimo de caracteres;
- Número máximo de caracteres aceito para o campo;
- Quantidade intermediária de caracteres;
- Número de caracteres acima do máximo permitido.

O Algoritmo 4.5 define os passos necessários para criação desse tipo de dado.

Algoritmo 4.5: Tipo de dado texto ou string.

```
1 Criar CT's variando o tamanho do texto e/ou string /*incluindo ausência de
      caractere*/
2 Criar CT's que variem o conjunto de caracteres
```

4.2.6 Considerando datas

Para os campos do tipo data, foi estabelecido o Algoritmo 4.6. Considerou-se esse tipo de dado como um caso particular de dados numéricos, pois já possui alguns limites pré-definidos. Por exemplo, o campo mês deve estar no intervalo de 1 a 12, o de dia de 1 a 28, 29, 30 ou 31 dependendo do mês, e o campo ano deve estar no intervalo de 1 até o limite permitido para o campo.

O Algoritmo 4.6 foi elaborado objetivando auxiliar na proposição desse tipo de dado, evidenciando mais algumas possibilidades além das propostas por Linkman et al. (2003). Foram propostos passos contemplando as possibilidades para as classes válidas e referenciando o Algoritmo 4.1 para a criação de casos de testes que validem todos os possíveis valores do intervalo ao qual o dado pertence.

Algoritmo 4.6: Tipo Data.

```
1 Criar CT's que contemplem datas válidas-atuais /* Válida */
2 Criar CT's que contemplem datas válidas-anteriores /* Válida */
3 Criar CT's que contemplem datas válidas-futuras /* Válida */
4 Criar CT's que contemplem datas inválidas /* Inválida */
5
6 Aplicar Algoritmos 4.1
```

4.2.7 Considerando horas

A criação de casos de testes para o tipo de dado Hora é semelhante ao tipo de dado Data. Ambos possuem a peculiaridade de limites pré-definidos e são um caso particular do tipo de dado numérico. O Algoritmo 4.7 foi proposto para criação de casos de testes com valores válidos. Além disso, é referenciado o Algoritmo 4.1 para contemplar os possíveis valores para o intervalo desse tipo de dado.

Algoritmo 4.7: Tipo Hora.

```
1 Criar CT's que contemplem horas válidas
2 Aplicar Algoritmos 4.1
```

4.2.8 Considerando dados estruturados heterogêneos

Entende-se por valores estruturados a composição de valores de dados, cujos tipos estão previstos nos Algoritmos 4.1, 4.2, 4.4, 4.5, 4.6, e 4.7; neste caso, cada dado é tratado individualmente, com respeito aos seus valores válidos e inválidos.

O Algoritmo 4.8 apresenta os passos para a criação de casos de teste referentes a esse tipo de dado. Nesse caso, são apenas referenciados os respectivos

algoritmos de cada tipo de dado.

Algoritmo 4.8: Tipo de dado estruturado heterogêneo.

```
1  para (cada dado que compõe o dado estruturado) faça
2    se (dados numéricos) então
3      Aplicar Algoritmo 4.1
4    fimse
5    se (dados booleanos) então
6      Aplicar Algoritmo 4.2
7    fimse
8    se (dados do tipo array) então
9      Aplicar Algoritmo 4.4
10   fimse
11   se (dados dos tipo texto ou string) então
12     Aplicar Algoritmo 4.5
13   fimse
14  se (dados dos tipo Data) então
15     Aplicar Algoritmo 4.6
16   fimse
17  se (dados dos tipo Hora) então
18     Aplicar Algoritmo 4.7
19   fimse
20  fimpara
```

4.2.9 Considerações gerais a todos tipos de dados

Em (LINKMAN et al., 2003) há considerações sobre valores ilegais e valores de tipos diferentes e casos especiais. Assim, nesse trabalho, os casos ilegais foram tratados em cada algoritmo isoladamente e adotou-se que, valores de tipos diferentes e casos especiais podem se referir a todos os tipos de dados. E os passos referentes foram sintetizados nos Algoritmos 4.9 e 4.10.

No Algoritmo 4.9, está definido o passo para a criação de casos de teste que avaliem o comportamento do software em relação ao recebimento de tipos de dados diferentes do esperado para o campo. Por exemplo, se o campo for numérico, fornecer espaço ou caracter.

Algoritmo 4.9: Todos os tipos de dados.

```
1  Criar CT's que contemplem domínios de entrada e saída com tipos diferentes do
       especificado. /* Inválida */
2  /* Por exemplo, se o campo for númerico, insira espaço ou algum caracter especial
       ou de controle */
```

4.2.10 Considerando casos especiais

Os casos especiais se referem a casos peculiares de cada tipo. Por exemplo adotar o dia 10 do mês de setembro no ano de 1752: mês que possui apenas 20 dias, devido à reforma do calendário - substituição do calendário Juliano pelo

Gregoriano (TONDERING, 2011). A Tabela 4.3 mostra o resultado de execução da linha de comando `Cal 9 1752`, no sistema operacional Linux, observa-se que, o dia 2 é seguido do dia 14 devido a referida reforma, sendo portanto esses dias que faltam no calendários dados de testes considerados casos especiais.

Tabela 4.3: *Calendário referente ao mês de setembro de 1752 - execução do comando* `Cal 9 1752`.

D	**S**	**T**	**Q**	**Q**	**S**	**S**
		1	2	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Outro exemplo seria utilizar espaço para o tipo *string*; o valor zero para o tipo numérico, dentre outros. Segundo (WHITTAKER, 2009), casos especiais podem ocorrer devido a alguma circunstância extraordinária ou à completa coincidência ou a um acidente. Por exemplo, num sistema, a combinação Shift-c tem um determinado significado, porém, acidentalmente, o usuário clica em Ctrl-c, alterando completamente o valor do conjunto. Assim, todos as combinações de caracteres com Ctrl, Alt e Esc são exemplos de caracteres especiais e ao menos uma amostragem desses caracteres devem ser considerados nos testes. Testadores também podem instalar fontes especiais que usuários estão propensos a usar e assim, testar diferentes linguagens. Algumas fontes, como Unicode e outros conjuntos de caracteres multibyte podem causar falhas no software se este foi indevidamente concentrado em uma certa linguagem. Assim, um bom teste seria verificar na documentação do produto quais são as linguagens suportadas e então instalar pacotes de linguagem e bibliotecas de fontes que permitirão testar os casos especiais. Outra fonte de caracteres especiais é a plataforma de execução do sistema. Todo sistema operacional, linguagem de programa, *browser*, ambiente de execução, possui um conjunto de palavras reservadas que devem ser tratadas como casos especiais. Por exemplo, no sistema operacional *Windows*, um conjunto de palavras reservadas como LPT1, COM1, AUX que quando utilizadas em nomes de arquivos, o sistema trava ou falha completamente. A única maneira de encontrar esses caracteres especiais é criar casos de testes que associem palavras reservadas ao sistema.

O Algoritmo 4.10 denota a criação de testes para esses casos, ressaltando que, o algoritmo apenas pontua a criação desses casos de testes, cabendo ao testador definir quais são os casos especiais para cada tipo e para o negócio em questão.

Algoritmo 4.10: Todos os tipos de dados – casos especiais.

```
1 Criar CT's que contemplem os casos especiais -- domínio de entrada. /*Válida ou
    Inválida*/
2 /* Por exemplo, o valor zero sempre deve ser testado, mesmo que esteja dentro do
    intervalo.
3 E valores situados no limite do tipo de dado para garantir que são armazenados e
    recuperados corretamente. */
```

4.3 Síntese da Estratégia do TFSE

Como mencionado anteriormente, a representação gráfica é mais fiel ao raciocínio original, compactando inúmeras palavras em imagens. A forma gráfica é mais intuitiva e visual, permitindo ao testador que, o processo de criação de casos de testes seja mais acessível. Desse modo, os algoritmos descritos acima foram representados abaixo em forma de fluxograma.

A Figura 4.1 representa o fluxograma proposto para os Algoritmos 4.1, 4.2, 4.4, 4.5, 4.6, 4.7, 4.9, e 4.10. O testador, ao seguir esse fluxograma, pode contemplar uma grande quantidade de tipos de dados. Anotações nos pontos de decisão do fluxograma remetem o testador ao algoritmo empregado naquele ponto. Por exemplo, a partir do início do fluxograma, a primeira decisão avalia se são tipos de dados numéricos e, em caso afirmativo, remete o testador para o Algoritmo 4.1. Os passos desse algoritmo são representados nos demais pontos de decisão a direita da figura, tratando dos casos de dados numéricos referentes a valores específicos, intervalo de valores e números reais.

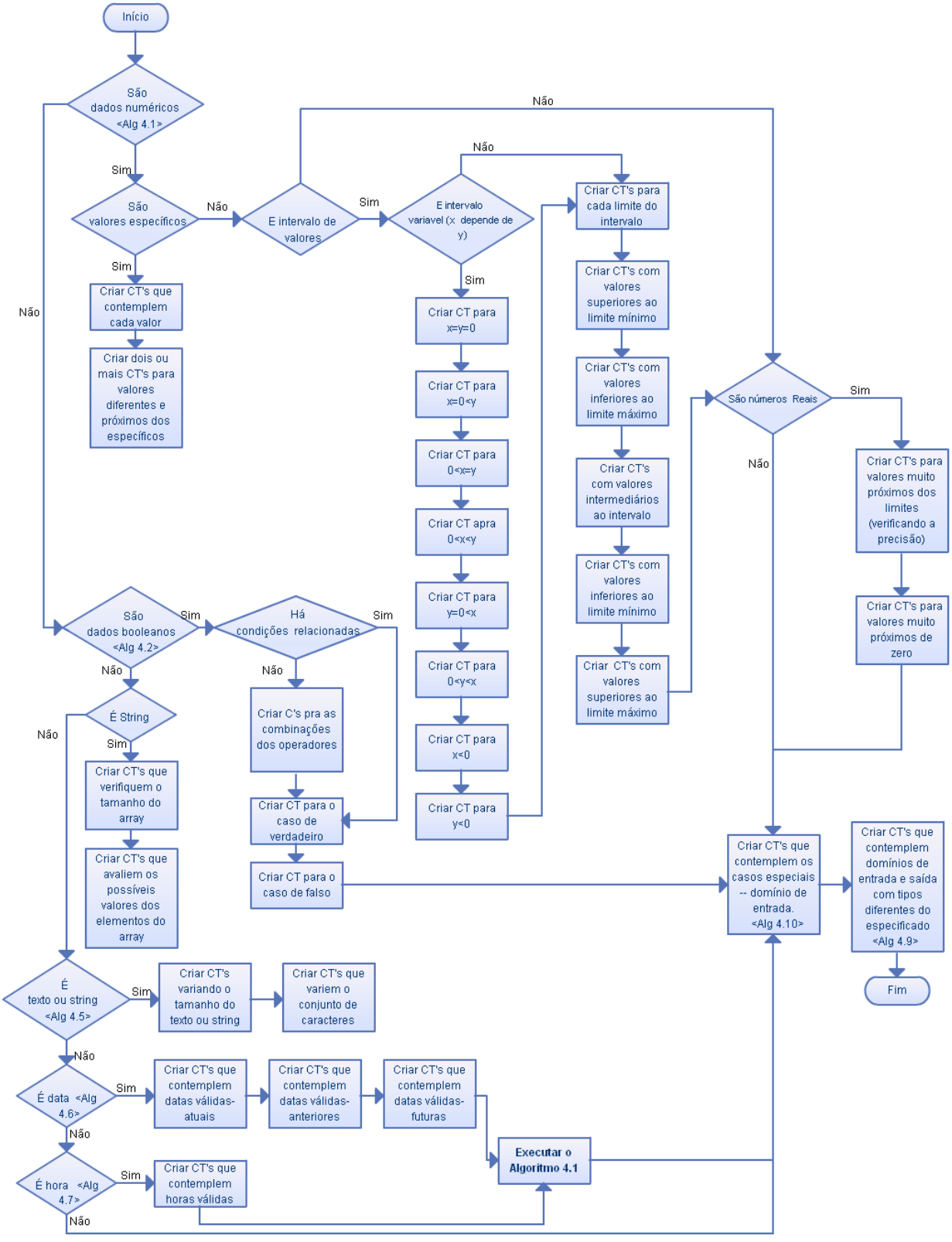


Figura 4.1: *Fluxograma de criação/verificação de casos de testes.*

Para não sobrecarregar o fluxograma anterior, os Algoritmos 4.3 e 4.8 foram representados em fluxogramas independentes.

Na Figura 4.2, está representado o fluxograma para o Algoritmo 4.3, definindo os possíveis caminhos para a criação de dados de testes referentes ao tipo de dado heterôgeneo.

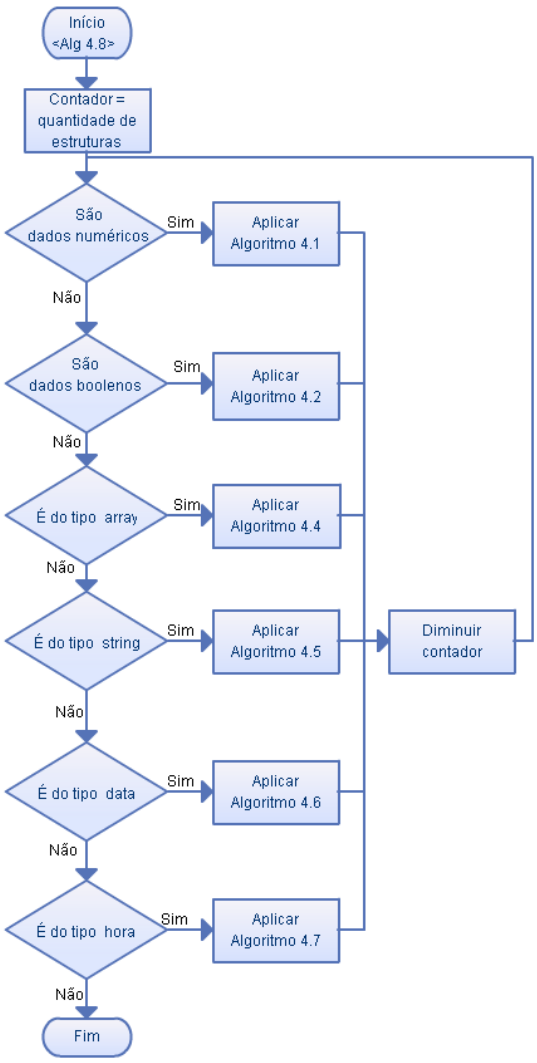


Figura 4.2: *Fluxograma de criação/verificação de casos de testes.*

Na Figura 4.3 está representado o fluxograma para o Algoritmo 4.8, definindo os possíveis passos para a criação de dados de testes que contemplem a variação da quantidade de dados de entrada.

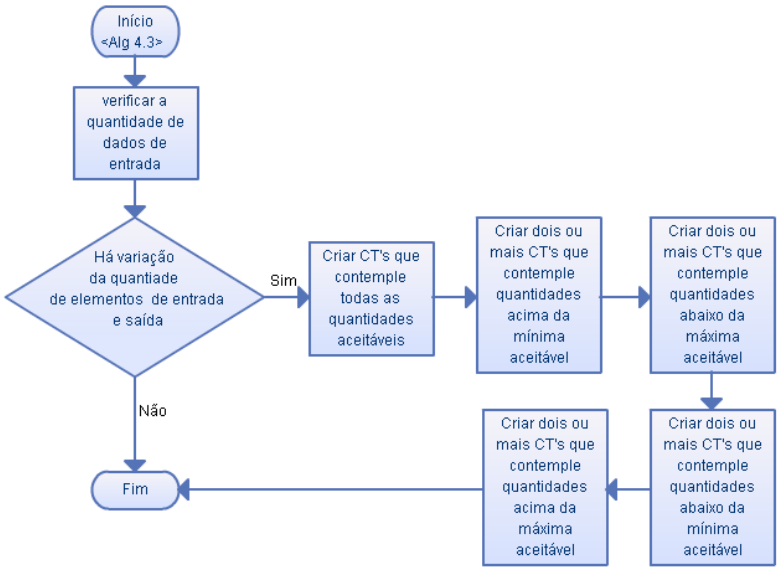


Figura 4.3: *Fluxograma de criação/verificação de casos de testes.*

O objetivo é que os fluxogramas sejam vistos como uma forma de rápido acesso aos conceitos por trás da geração de casos de teste para determinados tipos de dados e, em caso de dúvidas, o testador possa consultar o algoritmo específico referente ao tipo analisado.

4.4 Considerações Finais

Nesse capítulo, foram apresentados pontos de melhoria para o teste funcional sistemático, propondo o **T**este **F**uncional **S**istemático **E**stendido. Além disso, objetivou torná-lo mais prático e visual detalhando seus passos tanto em formato algoritmo quanto na forma gráfica.

As contribuições deste capítulo podem ser resumidas nos seguintes pontos:

1. Extensão do Teste Funcional Sistemático a dados do tipo data;
2. Extensão do Teste Funcional Sistemático a dados do tipo estruturado heterogêneo;
3. Extensão do Teste Funcional Sistemático a dados do tipo Hora;
4. Sugestão de divisão dos domínios de entrada e saída em classes de equivalência;
5. Representação dos passos do teste funcional sistemático em forma de algoritmo;
6. Representação dos passos do teste funcional sistemático em forma de fluxograma;

No próximo capítulo, o TFSE será adotado no contexto de software implementado por empresas de desenvolvimento. Será aplicado em duas situações distintas: agregando valor a um software - descobrindo defeitos e, melhorando um roteiro de testes utilizado em certificações.

CAPÍTULO 5

Estudos de Caso: Adoção do Teste Funcional Sistemático Estendido no Contexto de Empresas

Uma parte de grande importância na proposição de qualquer método, técnica ou estratégia é a sua validação. Nesta dissertação, dois estudos de caso foram empregados para avaliar o Teste Funcional Sistemático Estendido (TFSE) tanto na avaliação da qualidade de conjunto de teste existente quanto na geração de conjuntos de teste.

No primeiro caso, uma aplicação Web foi testada funcionalmente utilizando-se o TFSE e tanto o conjunto de teste resultante, quanto os defeitos encontrados são apresentados.

No segundo caso, um conjunto de teste foi inicialmente construído a partir de um roteiro de teste funcional. Uma vez que, em princípio, o roteiro não emprega critérios de teste explicitamente em sua estrutura, o TFSE foi utilizado para verificar qual a adequação do conjunto de teste gerado a partir do roteiro em relação as exigências do TFSE.

5.1 Estudo de Caso 1: *Software* Estratégia Para Ação (EPA)

5.1.1 Visão Geral da Organização 1 e do Software

Nesse estudo de caso, o TFSE foi aplicado em um *software* Web voltado para apoiar a Gestão Estratégica. A organização está no mercado desde 2003 e, e no ato da pesquisa deste trabalho, possui cerca de trinta clientes com uma média de vinte usuários por cliente.

O produto também está no mercado desde 2003 e, além do módulo de Gestão de Projetos, também agrega os seguintes módulos: Planejamento Estratégico,

Gestão da Qualidade, Ocorrências e Documentos e o módulo Produtividade. Dentre as diversas funcionalidades disponíveis no módulo Gestão de Projetos, objeto de estudo desse trabalho, podem ser citadas: montagem de projetos modelo para padronizar a criação de novos projetos; acompanhamento da capacidade de execução de um usuário e suas tarefas pendentes; vínculo de projetos com uma agenda corporativa; matriz de responsabilidades; gestão de risco do projeto; acompanhamento do orçamento do projeto; gestão da comunicação do projeto; emissão de relatórios gerenciais, dentre outras. O TFSE foi aplicado ao módulo de Criação de Projetos e Tarefas, por ser esse um módulo crítico em relação à funcionalidade do sistema, segundo sua equipe de desenvolvimento (SIMEON, 2010).

Em relação à codificação do *software*, as linguagens utilizadas são PHP (Hypertext Preprocessor (PHP, 2010)) e Javascript (Java Script, 2010) e o gerenciador de banco de dados MYSQL (MYSQL, 2010). Em termos de complexidade, o produto possui em torno de 580 KLOC (milhares de linhas de código).

A equipe de desenvolvimento é composta por quatro desenvolvedores e um testador. Sobre o processo de testes adotado, segundo a organização, há a execução de teste funcional sem a adoção de qualquer critério e/ou metodologia de teste. Além disso, atualmente, a atividade de manutenção corresponde a aproximadamente 20% do esforço do processo de desenvolvimento.

5.1.2 Aplicação das Diretrizes no contexto do Estudo de Caso 1

As diretrizes do TFSE foram aplicadas nos campos de dados apresentados nas telas do programa. Como a documentação disponível é escassa, inicialmente foram gerados os dados de testes seguindo as diretrizes do TFSE, num segundo momento, os casos de testes são gerados com o auxílio de um oráculo ou o desenvolvedor responsável pelo sistema, auxiliando na derivação das possíveis saídas esperadas para cada dado de teste definido.

No contexto desse estudo de caso, as funcionalidades selecionadas, Criação de Projetos e Criação de Tarefas, são descritas a seguir.

Funcionalidade *Criação de Projetos*.

Inicialmente, para uma melhor organização da descrição da aplicação do TFSE, serão adotados os campos destacados na Figura 5.1, para a geração dos dados de testes.

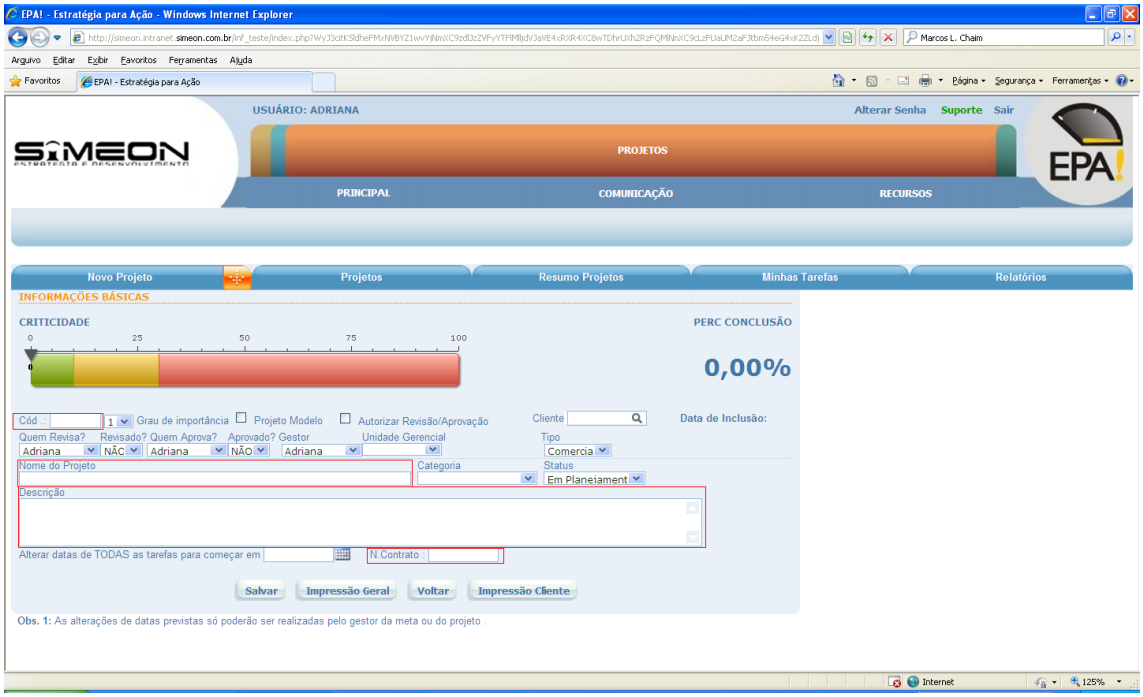


Figura 5.1: *Formulário de inclusão de projetos.*

Observa-se na tela em questão a presença de campos de dados do tipo *String*, permitindo que o Algoritmo 4.5 seja aplicado na derivação dos testes.

1. Aplicação do Algoritmo 4.5:

   (a) **Criação do Dado de Teste com ausência de caracteres:**
   Nesse caso de teste, não serão inseridos dados em nenhum dos campos da tela de Criação de Projetos. Na Figura 5.2, está representado o caso de teste correspondente à ausência de caracteres, observa-se que, a ausência de caracteres depende das regras de negócio do sistema. Nesse caso, somente os campos "Nome do Projeto" e "Unidade Gerencial" são dados obrigatórios, porém, o último campo não será considerado no escopo do teste. Assim, quando não inseridos caracteres no campo "Nome do Projeto", é emitida uma mensagem solicitando informação para o campo. Nota-se que, além do erro de acentuação da mensagem, esta não é intuitiva, pois se refere ao campo como "Titulo" o que pode confundir o usuário.

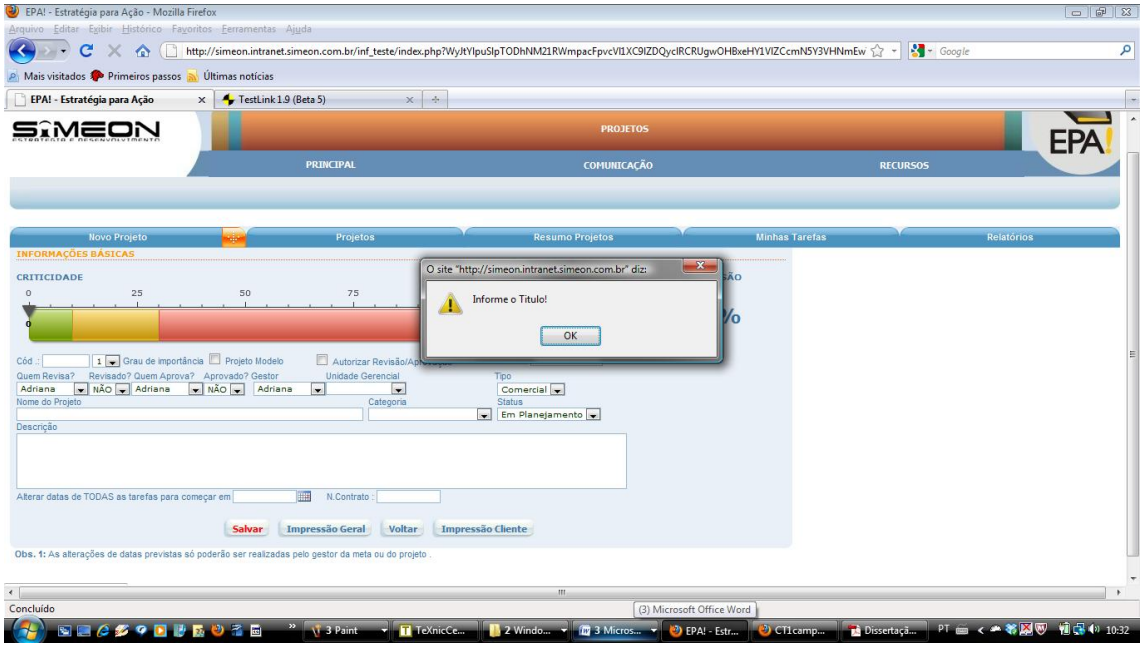


Figura 5.2: *Ausência de Caracteres - CT1.*

(b) **Criação de Dado de Teste com limite mínimo de caracteres.** Considerando fundamentos do Particionamento de Equivalência, nos próximos testes serão explorados os limites máximos e mínimos dos campos considerados no escopo dessa tela. Ao analisar os valores mínimos, foram criados casos de testes que avaliam o comportamento do sistema quando os campos recebem *strings* com um ou dois caracteres.

Conforme observado na Figura 5.3, é fornecido *string* com apenas um caractere para os campos "Código do Projeto", "Nº do Contrato", "Nome do Projeto", "Descrição", representando o Caso de Teste 2 (CT2).

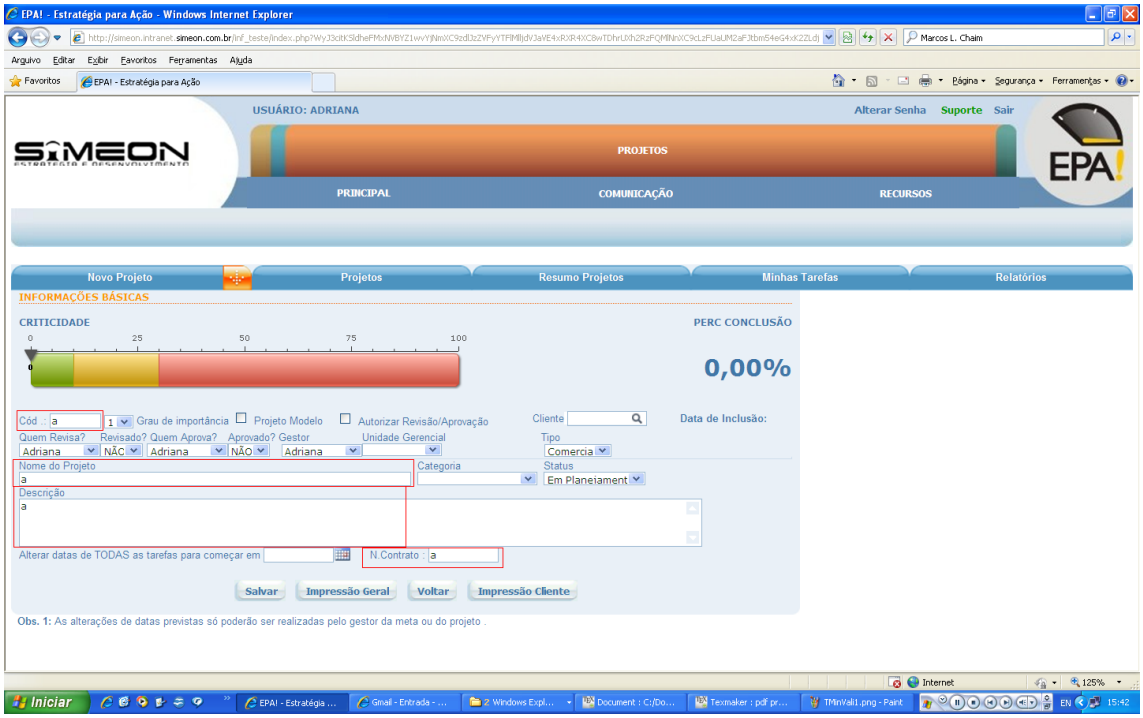


Figura 5.3: *Tamanho Mínimo Válido - CT2.*

Na Figura 5.4, é apresentado o caso de teste em que foi fornecido para os campos *string* com dois caracteres.

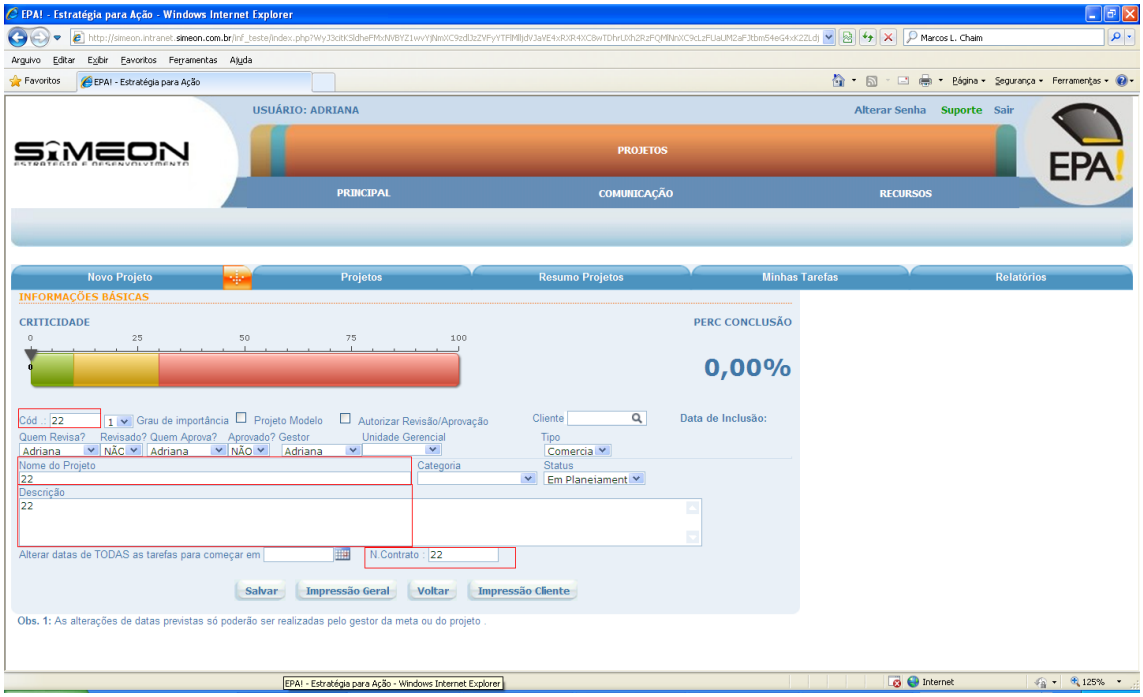


Figura 5.4: *Tamanho Mínimo Válido - CT3.*

Nos dois testes executados o sistema se comporta de forma esperada, cadastrando o projeto.

(c) **Criação de Dados de Testes com limite máximo de caracteres.** Como não há documentação que define a quantidade de caracteres aceitável para cada campo, foram inseridas uma extensa quantidade de caracteres, tentando extrapolar o máximo permitido.

Assim, para o primeiro dado de teste, o total de caracteres utilizados nos campos textuais foi de 617.897, representado na Figura 5.5. Observa-se que tais dados de teste foram gerados aleatoriamente, considerando caracteres válidos.

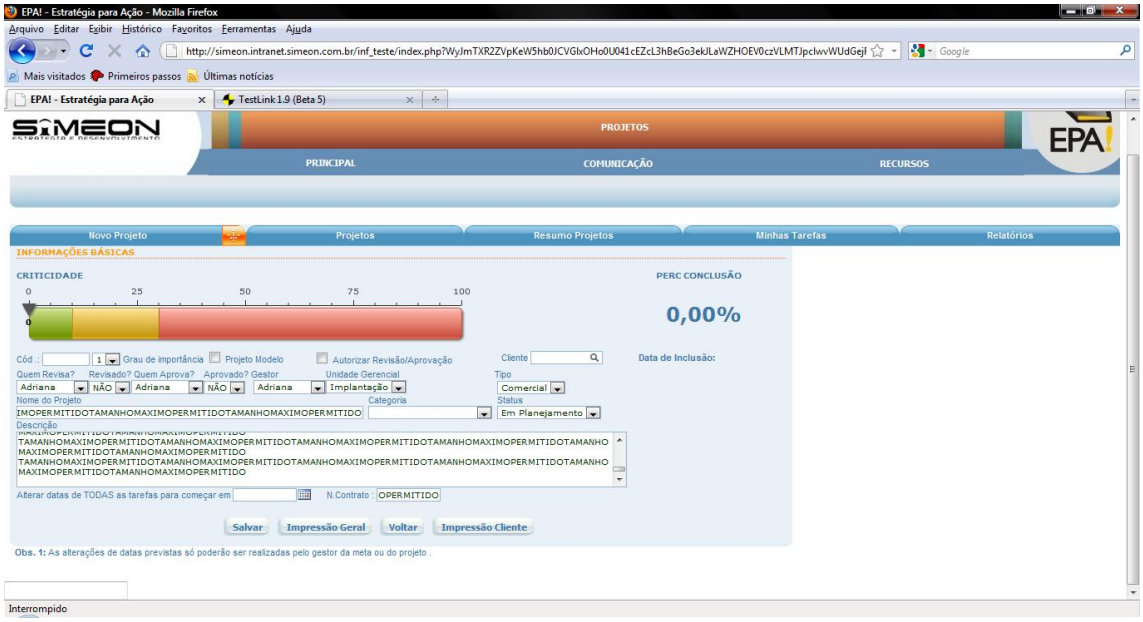


Figura 5.5: *Tamanho Máximo Válido - CT4.*

Na criação do segundo dado de teste foram inseridos 3.708.447 caracteres nos mesmos campos, representados na Figura 5.6

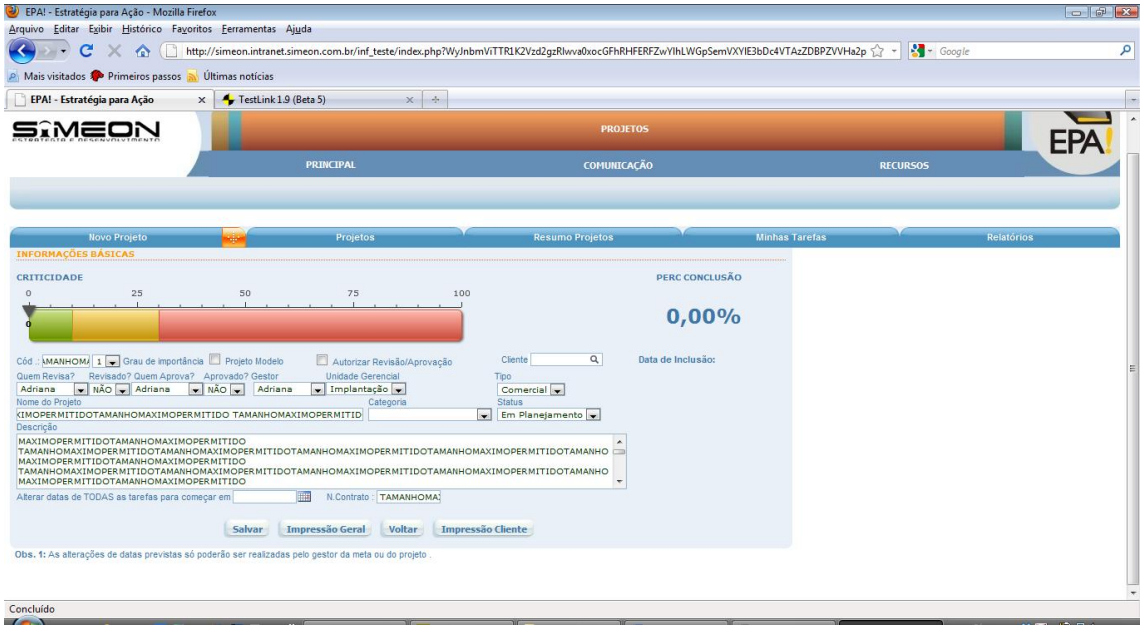


Figura 5.6: *Tamanho Máximo Válido - CT5.*

Em relação a esses testes, percebe-se que, apesar do campo permitir continuar inserindo os dados, ao salvar o projeto, apenas parte da *string*

original foi efetivamente armazenada. O que pode ser descrito como um problema, pois, ocorre perda de dados. Para cada dado notou-se que, para o campo: "Código do Projeto", apenas vinte caracteres são preservados, "Nº do Contrato", dez caracteres e o campo "Nome do Projeto", cento e trinta e dois caracteres. No campo "Descrição", notou-se que, quando a *string* inserida é muito grande, o campo não armazena nenhum caractere, indicando falhas no *software*.

(d) **Criação de Dados de Testes com *strings* com quantidade intermediária de caracteres.**
Na criação desses dados de teste, representados pelas Figuras 5.7 e 5.8, contemplou-se a variação do tamanho do campo de texto, considerando a quantidade de caracteres especificada abaixo (Figura 5.7):

- Cód.: 20 caracteres.
- Nome do Projeto: 132 caracteres.
- Descrição: 25.000 caracteres.

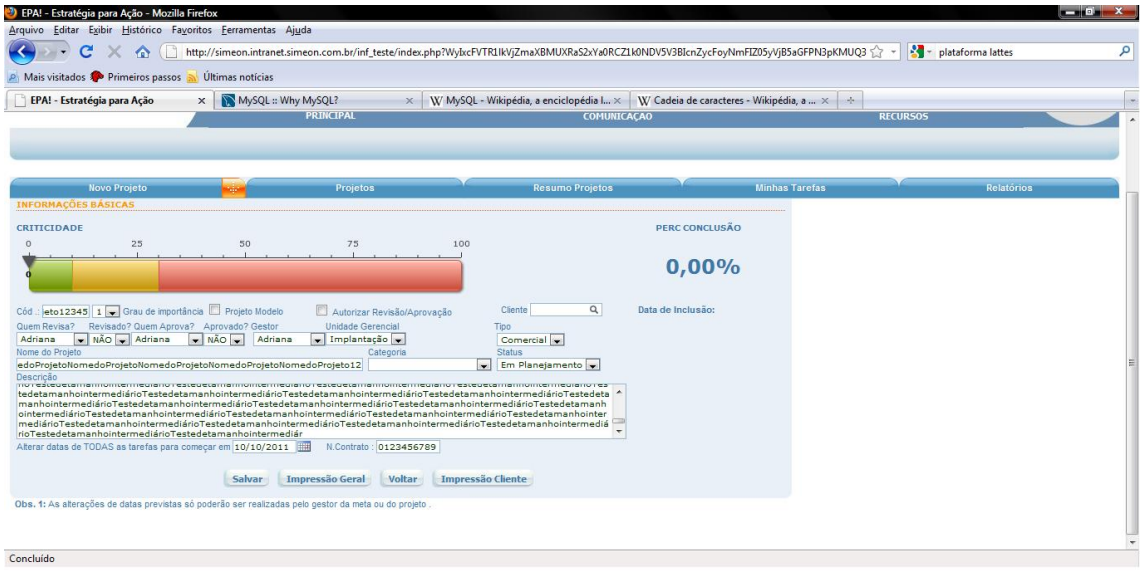


Figura 5.7: *String com tamanho intermediário - CT6.*

E para o dado de teste representado pela Figura 5.8 foi fornecida a seguinte quantidade de caracteres:

- Cód.: 19 caracteres.
- Nome do Projeto: 131 caracteres.
- Descrição: 24.000 caracteres.

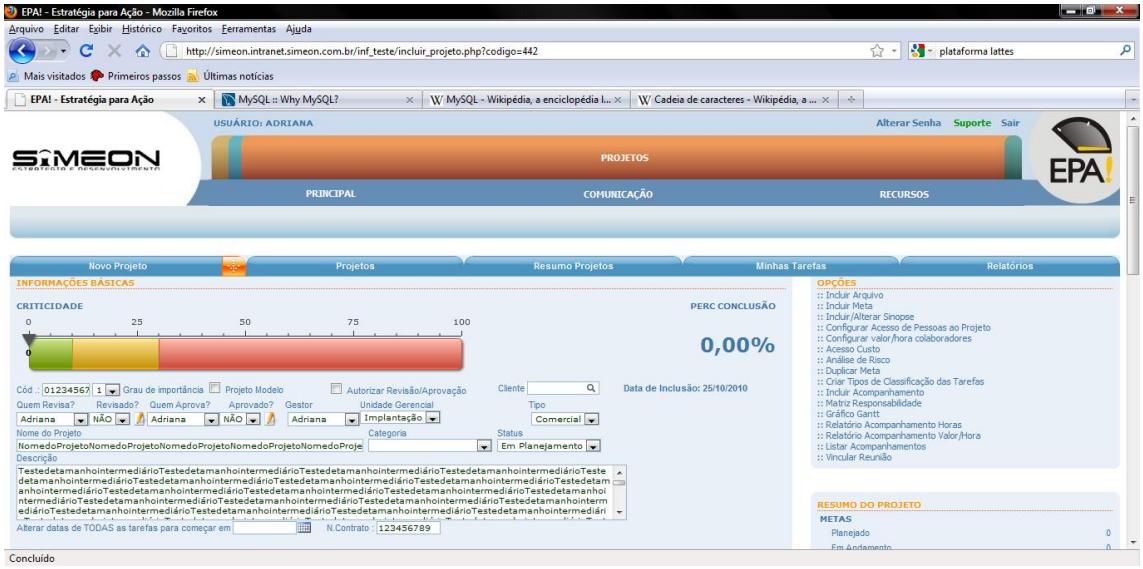


Figura 5.8: *String com tamanho intermediário - CT7.*

Observa-se que, considerando os limites estabelecidos, em ambos os testes apresentados acima, o sistema se comportou adequadamente, permitindo a criação dos projetos e o correto armazenamento das informações.

2. **Aplicando o Algoritmo 4.10.**

   A Figura 5.9 ilustra um dado de teste contendo caracteres especiais nos campos textuais. O conjunto de dados de caracteres especiais deve ser empregado, principalmente nesses campos, em função do significado que alguns deles têm para a linguagem de programação, sistema operacional, e/ou banco de dados empregados.

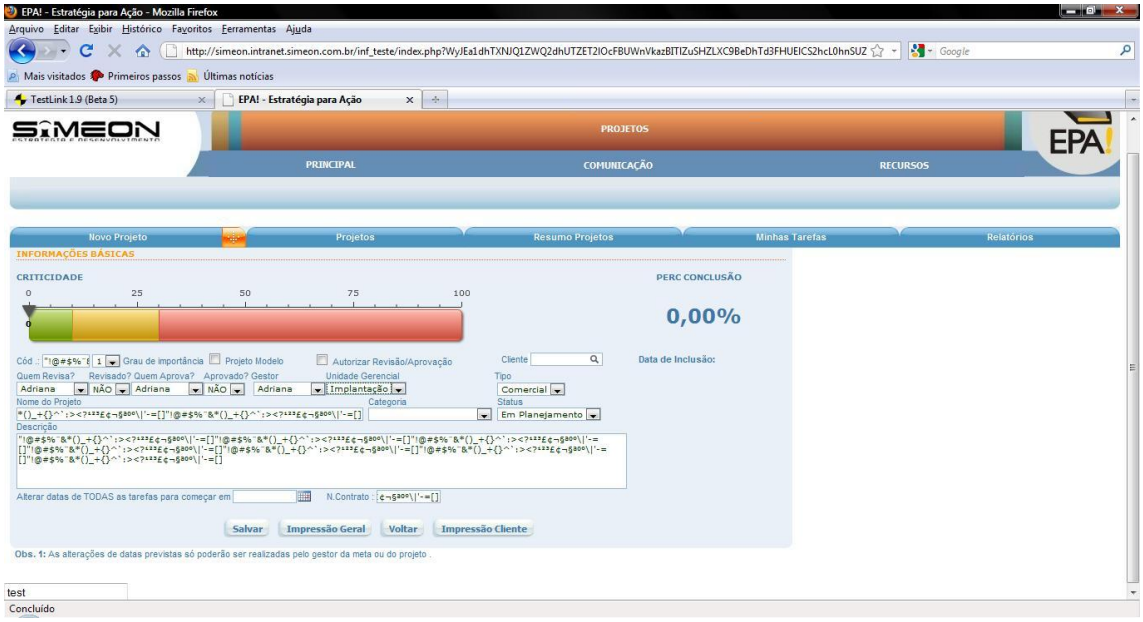


Figura 5.9: *String com caracteres especiais - CT8.*

Após a inserção da *string* de caracteres especiais foi apresentada a tela da Figura 5.10, na qual vários dados foram perdidos, o projeto não foi armazenado e a tela ficou desconfigurada, com os rótulos substituídos por parte do conteúdo dos campos.

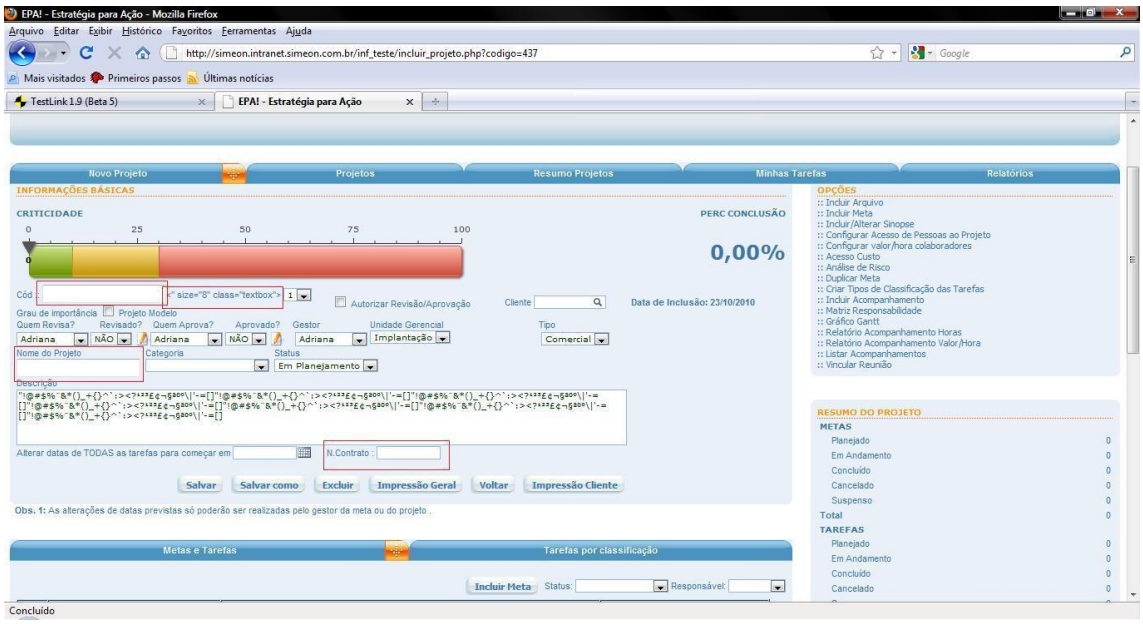


Figura 5.10: *Tela inconsistente ao inserir string com caracteres especiais.*

Observa-se que tal situação representa uma falha no sistema pois, os dados inseridos propositadamente podem, perfeitamente, serem inseridos pelo usuário por descuido ou desatenção. O ideal seria que os campos de dados incluíssem filtros de modo a impedir que caracteres especiais não permitidos não fossem aceitos, que então, uma mensagem de erro fosse exibida e os dados desconsiderados. Nesse sentido, o resumo dos defeitos encontrados e o número de casos de testes gerados são exibidos na Tabela 5.1.

Tabela 5.1: *Número de casos de testes e defeitos encontrados - Criação de Projetos.*

Casos de Testes Gerados	**Defeitos Encontrados**
8	8

Ao utilizar TFSE para derivar dados de testes para o contexto desse estudo de caso, notou-se que, apesar da empresa adotar teste funcional para a homologação de seu *software*, algumas inconsistências foram apontadas. Assim, ao empregar critérios e métodos de testes funcionais de forma sistematizada evidenciou-se que, apesar do sistema ter sido testado pela equipe responsável, um número considerável de defeitos foi encontrado. Embora, os testes realizados nesse estudo de caso não tenham sido apoiados por especificação, demonstrou-se que é possível, com o auxílio de um oráculo, a criação de dados de testes utilizando as telas do sistema.

Para uma maior contribuição, o TFSE também foi aplicado à funcionalidade "Incluir Tarefas", cujos passos são descritos no Apêndice A.

Os resultados aqui apresentados foram repassados para a empresa visando auxiliar na melhoria de seu produto de software.

5.2 Estudo de Caso 2: Programa Aplicativo Fiscal – Emissor de Cupom Fiscal (PAF-ECF)

5.2.1 Visão Geral do Contexto de Aplicação do Roteiro

O Conselho Nacional de Política Fazendária (Confaz) (CONFAZ, 2010) credencia órgãos técnicos para realizar a análise funcional de programas aplicativos fiscais. Esses órgãos técnicos são responsáveis por emitir certificação comprovando que o *software* homologado está de acordo com o Roteiro de Análise Funcional (ROTEIRO, 2010) aprovado pela Cotepe/ICMS (**Co**missão **Té**cnica **Pe**rmanente do Im-

posto sobre Operações Relativas à Circulação de Mercadorias e sobre Prestações de Serviços de Transporte Interestadual e Intermunicipal e de Comunicação).

O roteiro descreve os testes correspondentes aos requisitos para o Programa Aplicativo Fiscal – Emissor de Cupom Fiscal (PAF-ECF) estabelecidos na legislação (COTEPE, 2009b) e tem o intuito de padronizar o comportamento desse tipo de *software*, evitar que haja desvios dessa legislação e portanto, combater a sonegação fiscal.

O Programa Aplicativo Fiscal controla a máquina de Emissão de Cupom Fiscal (impressora fiscal), que substitui a emissão manual da nota fiscal. Até recentemente, cada estado definia como o aplicativo fiscal deveria atuar com o ECF. Finalmente, após diversas reuniões das entidades representantes dos estados, em 2008, a Confaz publicou três documentos contendo as informações para análise do PAF-ECF, que são: o Ato Cotepe 06/08 (COTEPE, 2009a), o Convênio ICMS 15/08 (CONVêNIO, 2008) e o Roteiro de Análise Funcional (ROTEIRO, 2010). Estes documentos são de abrangência nacional, ou seja, todas as empresas desenvolvedoras de sistema ECF deverão atendê-los.

O Ato Cotepe define regras para diversos ramos de atividades, conforme suas peculiaridades. Resumidamente, são estabelecidos 43 requisitos de *software*.Os requisitos são divididos por área de atuação, conforme abaixo:

- 31 requisitos gerais, obrigatórios para todo e qualquer PAF-ECF;
- 5 específicos para estabelecimento revendedor de combustíveis;
- 3 específicos para restaurantes, bares e estabelecimentos similares;
- 1 específico para farmácia de manipulação;
- 1 específico para oficina de conserto;
- 1 específico para transporte de passageiros;
- 1 específico para identificar a empresa desenvolvedora do PAF-EC.

Referenciando o Ato Cotepe, o Roteiro de Análise Funcional descreve 102 testes que devem ser executados para verificar se os requisitos são atendidos. Cada teste é composto por passos que são as ações individuais que devem ser executadas. Segundo as Orientações Gerais do Roteiro (ROTEIRO, 2010), os passos que constituem os testes devem ser executados sequencialmente, na ordem em que são apresentados e os resultados da execução devem ser confrontados com o requisito respectivo para se verificar o atendimento à legislação. Assim, além dos passos, cada teste é constituído da descrição de duas condições para o requisito (atendido ou não atendido). Nesse contexto, seguindo o padrão dos requisitos, o roteiro é dividido em seis blocos, nos quais são apresentados os requisitos e os testes. Esses blocos são assim definidos:

- Bloco I: oitenta e seis testes aplicáveis a todo tipo, qualquer PAF-ECF.
- Bloco II: cinco testes aplicáveis somente em PAF-ECF para posto revendedor de combústivel
- Bloco III: três testes aplicáveis somente em PAF-ECF para bares, restaurantes e similares.
- Bloco IV: dois testes aplicáveis somente em PAF-ECF para farmácia de manipulação.
- Bloco V: quatro testes aplicáveis somente em PAF-ECF para oficina de conserto.
- Bloco VI: dois testes aplicáveis somente em PAF-ECF para prestador de serviço de transporte de passageiros.

A aplicação do roteiro e homologação de PAF-ECF só pode ser realizada por um órgão técnico credenciado segundo as normas do Convêncio ICMS 15/08 (CONVÊNIO, 2008). técnico.

Como resultado, ao final da execução do roteiro de teste, um conjunto de teste é produzido por um testador da equipe de homologação do PAF-ECF. Deduz-se que, como os testes são gerados a partir de um roteiro, não há uma grande variação na quantidade de testes executados por qualquer um dos testadores desse grupo. Porém, como o roteiro é muito amplo em relação aos dados de testes, quando não especificado, cada testador fornece os dados como bem entender, sem a experiência de empregar critérios de testes específicos. Assim, não há garantia de que os mesmos valores, ou valores com características similares sejam fornecidos (homogeneização dos testes) e nem que valores essenciais sejam executados.

Desse modo, mesmo como o roteiro em mãos, a qualidade dos testes realizados é extremamente dependente da experiência e conhecimento do avaliador sobre técnicas e critérios de teste, além do conhecimento do domínio da aplicação. O TFSE auxilia a sistematizar a forma como a equipe de homologação passa a trabalhar.

5.2.2 Visão Geral das Equipes Homologadoras dos Órgãos credenciados

Em relação aos órgãos técnicos, normalmente, são compostos de profissionais e estudantes responsáveis pela certificação de programas aplicativos fiscais. Conforme explicitado anteriormente, esses programas devem possuir um conjunto de requisitos funcionais comuns estabelecidos pelo Ato Cotepe, mas são desenvolvidos em linguagens, por equipes e processos de desenvolvimento distintos. Desse modo, a equipe de homologação não tem conhecimento interno da implementação,

sendo possível executar apenas testes funcionais direcionados pelo Roteiro de análise funcional, sendo registradas quaisquer discrepâncias encontradas.

Com o objetivo de identificar o perfil das equipes que atuam na homologação de produtos PAF-ECF, um questionário foi desenvolvido e submetido a todos aos 24 órgãos técnicos credenciados na Confaz, no ano de 2010. A descrição das perguntas e as respostas obtidas estão descritas respectivamente nos Apêndices B e C.

5.2.3 Visão Geral da Organização 2 e do *Software*

Neste estudo de caso, a organização, há vinte anos, desenvolve e mantêm *softwares* para os segmentos: administrativo/financeiro, contábil, fiscal, comercial e gestão em engenharia. A finalidade do *software*, adotado no contexto deste trabalho, é atender pequenas e médias empresas do comércio em geral, tanto no atacado e no varejo e que devem emitir cupom fiscal, nota fiscal eletrônica e trabalham com vendas com cartão de crédito. Esse *software* possui trinta e dois clientes e está no mercado há seis anos.

A linguagem de programação utilizada na implementação do *software* é Delphi 7 (BORLAND, 2010) e a equipe responsável pelo desenvolvimento do *software* é composta por dois analistas de sistemas. Sobre a atividade de testes, segundo um dos analistas: "é realizado teste de sistema sob o ponto de vista do usuário final, em que as funcionalidade são varridas em busca de falhas em relação aos objetivos originais. Além disso, a atividade de manutenção corresponde a aproximadamente 30% do processo de desenvolvimento".

5.2.4 Aplicação do TFSE no contexto do Estudo de Caso 2

Nesse estudo de caso, foi adotada a seguinte metodologia para a utilização do TFSE:

1. **Um membro da equipe de um órgão credenciado executou partes do roteiro.**
   De 43 requisitos, 10 foram considerados, possíveis de serem aplicados ao PAF-ECF em questão. No estudo de caso, a limitação do escopo foi decorrente ao uso de um emulador de impressora fiscal que inviabilizava a aplicação dos demais requisitos.
2. **Os caso de testes executados conforme o roteiro foram documentados.**
   Foram executados 27 casos de testes referentes aos requisitos selecionados. Todos foram documentados, utilizando a ferramenta TestLink (2010), de forma a registrar os dados de testes utilizados pelo homologador.

3. **A partir dos testes gerados, da documentação disponível e do próprio roteiro foram gerados outros testes adotando as diretrizes do TFSE.** Ao empregar o TFSE no contexto do PAF-ECF utilizando os próprios passos descritos no roteiro e os respectivos requisitos, foram gerados casos de testes que ampliam a abrangência do roteiro, possibilitando assim que, aumente a probabilidade de encontrar defeitos.

No contexto desse trabalho, dos 27 casos de teste documentados, 3 deles foram utilizados para descrever a utilização do TFSE. Os três testes foram selecionados com base na criticidade da funcionalidade executada para a aplicação e são descritos abaixo.

Descrição do Teste 041 referente ao Requisito XII

A seguir são apresentadas as descrições dos itens dos requisitos e os respectivos testes[1].

- **REQUISITO XXI - ITEM 1**:
  O PAF-ECF deve disponibilizar tela para registro e emissão de Comprovante Não Fiscal relativo às operações de retirada e de suprimento de caixa.

- **TESTE 041:** Registro de Suprimento de Caixa.
  Passo 1: Localize nos menus do programa a opção que permite registrar suprimento de caixa.
  Passo 2: Registre um suprimento de caixa no valor de R$ 1,00. Observe se o ECF emitiu o Comprovante Não Fiscal relativo ao suprimento de caixa corretamente.
  Condição para requisito atendido: Emissão do Comprovante Não Fiscal de Suprimento de Caixa no valor de R$ 1,00.
  Condição para requisito não atendido: Inexistência de função para registro de Suprimento de Caixa ou falta de emissão do Comprovante Não Fiscal de Suprimento de Caixa.

- **Caso de teste derivado a partir do Roteiro**

  A Figura 5.11 representa os Passos 1 e 2 executados pelo testador do órgão certificador. Observa-se que, no lado esquerdo da tela é apresentado o emulador

[1] O teste referente à descrição, item e teste foram extraídos na íntegra do Roteiro de Testes (ROTEIRO, 2010).

da emissora fiscal, e como solicitado nesse caso de testes, é emitido um relatório gerencial relativo ao suprimento de caixa.

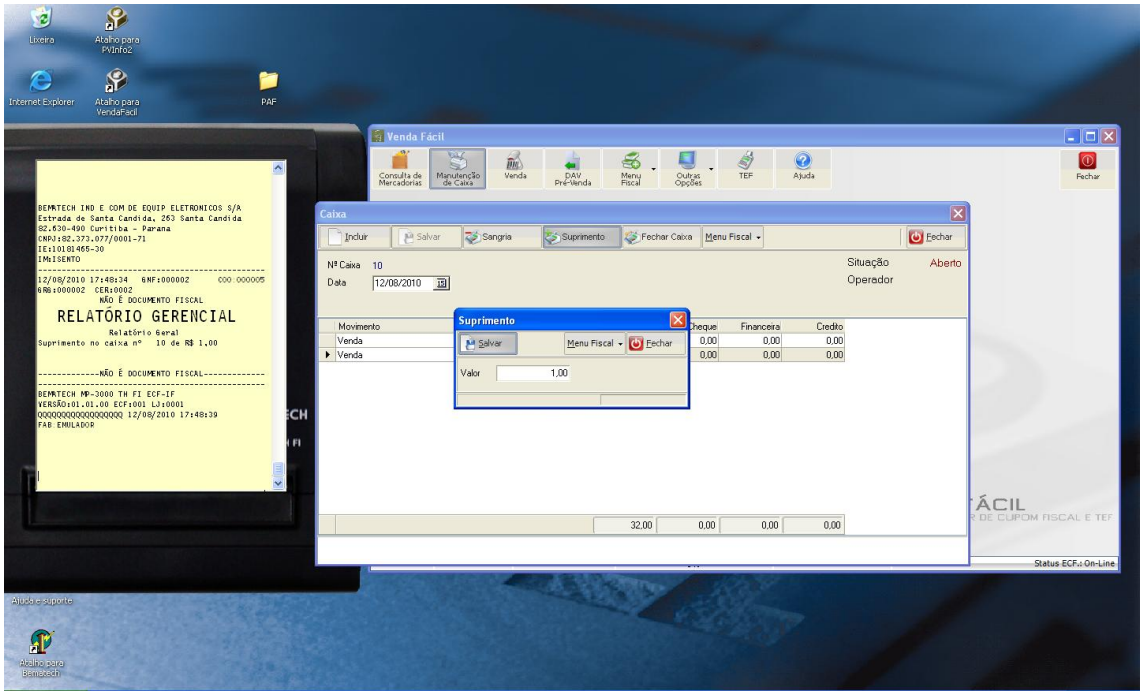


Figura 5.11: *Execução do teste 041 do Roteiro (ROTEIRO, 2010).*

Nesse caso, como o teste citava o dado de teste, o testador seguiu exatamente os passos descritos do roteiro sem propor nenhum valor diferente para o teste e, segundo o roteiro, a execução apresentada acima equivale à condição para requisito atendido. Porém, ao se avaliar a dimensão do conjunto de dados de teste aceitáveis para esse campo, percebe-se sua limitação. Portanto, não é possível garantir que para outros casos, além do especificado no roteiro, o requisito será atendido. Nesse contexto, será utilizado o TFSE para criar testes correspondentes o mesmo requisito, tentando ampliar assim, as possibilidades de encontrar algum desvio da especificação de requisitos.

Casos de Testes Gerados utilizando TFSE.

Considerando que, o campo Suprimento é um campo que aceita dados numéricos, para derivar casos de testes para essa tela, serão utilizados os passos definidos no Algoritmos 4.1 e 4.9 do TFSE.

Aplicando o Algoritmo 4.1:

Limite do intervalo: 0,01 $\leq$ Suprimento $\leq$ 9.999.999,99.

- Criar CT's com valores-limite do intervalo - Classe válida.

Casos de Testes	**Parâmetro de Entrada**
CT1	0,01
CT2	9.999.999,99

- Criar dois ou mais CT's com valores superiores ao limite mínimo - Classe Válida.

Casos de Testes	**Parâmetro de Entrada**
CT3	0,02
CT4	0,05

- Criar dois ou mais CT's com valores inferiores ao limite máximo - Clásse Válida

Casos de Testes	**Parâmetro de Entrada**
CT5	9.999.999,98
CT6	9.999.999,00

- Criar dois ou mais CT's com valores intermediários ao intervalo - Clásse Válida

Casos de Testes	**Parâmetro de Entrada**
CT7	1.000,02
CT8	1.987.876,09

- Criar dois ou mais CT's com valores inferiores ao limite mínimo - Clásse Inválida

Casos de Testes	**Parâmetro de Entrada**
CT9	0,00
CT10	-0,01

- Criar dois ou mais CT's com valores superiores ao limite máximo - Clásse Inválida

Casos de Testes	**Parâmetro de Entrada**
CT11	10.000.000,00
CT12	10.000.000,01

- Criar CT's para valores muito próximos de zero.
  Como o limite mínimo é próximo de zero, já foram criados casos de testes que contemplem esse caso.
- Criar CT's para valores muito próximos dos limites estabelecidos - adotando o critério de precisão definido.
  Como só são permitidas duas casas decimais, o critério de precisão também foi atendido e esse item também já foi contemplado.

Aplicando o Algoritmo 4.9:

- Criar CT's que contemplem domínios de entrada e saída com tipos diferentes do especificado - Classe Inválida

Casos de Testes	**Parâmetro de Entrada**
CT13	AE
CT14	l%$"""'@!#*≤≥≺≻ ◁▷ ≠≡≈≅∝

Descrição do Teste 042 referente ao Requisito XII

- **REQUISITO XII - ITEM 1**:
  O PAF-ECF deve disponibilizar tela para registro e emissão de Comprovante Não Fiscal relativo às operações de retirada e de suprimento de caixa.

- **TESTE 042:** Registro de Sangria ou Retirada de Caixa.
  Passo 1: Localize nos menus do programa a opção que permite registrar sangria ou retirada de caixa.
  Passo 2: Registre uma sangria ou retirada de caixa no valor de R$ 0,50. Observe se o ECF emitiu o Comprovante Não Fiscal relativo à sangria de caixa corretamente.
  Condição para requisito atendido: Emissão do Comprovante Não Fiscal de Sangria ou Retirada de Caixa no valor de R$ 0,50.
  Condição para requisito não atendido: Inexistência de função para registro de Sangria ou Retirada de Caixa ou falta de emissão do Comprovante Não Fiscal de Sangria ou Retirada de Caixa.

- **Caso de teste derivado a partir do Roteiro**
  A Figura 5.12 representa a execução dos testes descritos no roteiro, realizado por testador do órgão certificador. Na tela são apresentados do lado direito o programa aplicativo fiscal com destaque para a funcionalidade de sangria e do lado esquerdo o emulador da impressora fiscal. Observa-se que, a impressora emite um relatório gerencial da sangria efetuada.

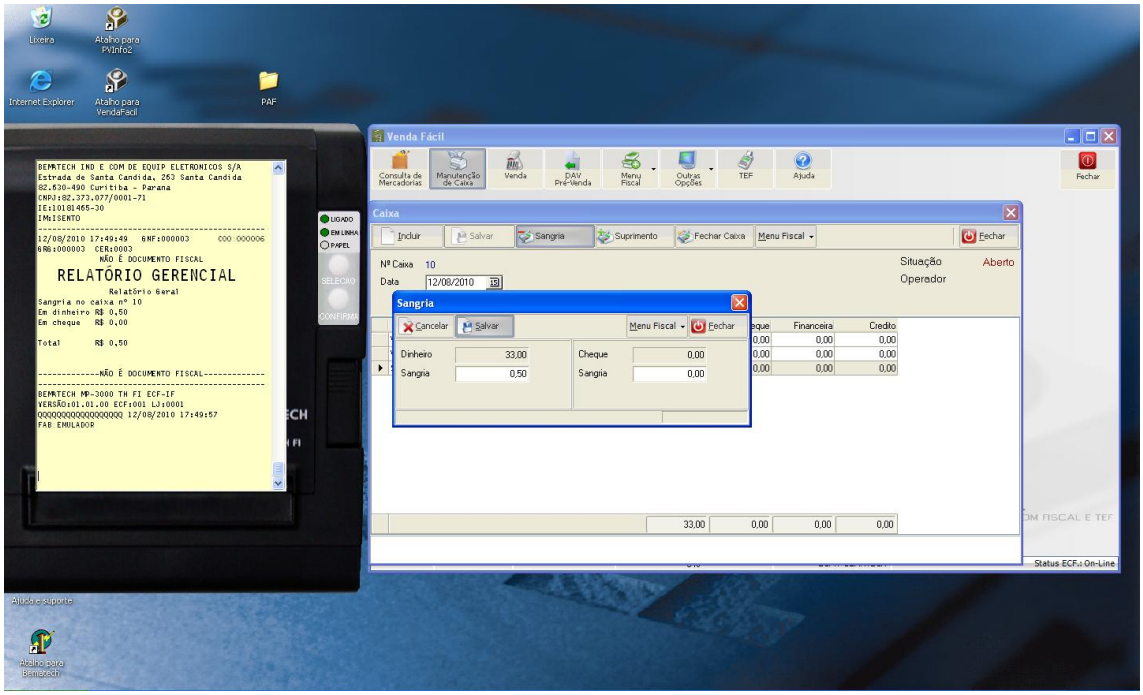


Figura 5.12: *Execução do teste 042 do roteiro (ROTEIRO, 2010).*

Semelhante ao Teste 041, descrito anteriormente, na execução do Teste 042 observa-se que o testador segue exatamente o que é definido pelo roteiro, fornecendo ao campo "Sangria" apenas o dado sugerido. E, conforme explicitado no roteiro essa execução modela a definição a condição para requisito atendido.

Nesse requisito há uma peculiaridade não descrita, relacionada ao negócio, em que a sangria não pode ser superior a quantia existente no caixa. Então, além de realizar testes considerando os valores aceitáveis para o campo, também serão descritos dados de testes que contemplem essa regra.

Casos de Testes Gerados utilizando TFSE.

Considerando que, o campo "Sangria" aceita dados do tipo numérico, ao aplicar as diretrizes do TFSE, seguiram-se os passos dos Algoritmos 4.1 e 4.9, descritos abaixo.

Aplicando o Algoritmo 4.1:

Limite do intervalo: *(Quantia no Caixa)*$\geq$ *(Sangria)*
O Algoritmo 4.1 referencia intervalos variáveis, onde o valor de uma variável x depende da variável y. Então, no caso dessa peculiaridade de negócio, como

a sangria depende da quantia em caixa, tem-se que a variável x equivale à sangria e a variável y equivale à quantia no caixa.

- Criar CT que contemple $x = y = 0$ - Classe Válida

Casos de Testes	**Sangria**	**Quantia no caixa**
CT15	0,00	0,00

- Criar dois ou mais CT's que contemple $x = 0 < y$ - Classe Válida

Casos de Testes	**Sangria**	**Quantia no caixa**
CT16	0,00	1,00
CT17	0,00	500.000,00

- Criar dois ou mais CT's que contemple $0 < x = y$ - Classe Válida

Casos de Testes	**Sangria**	**Quantia no caixa**
CT18	5,00	5,00
CT19	300.999,99	300.999,99

- Criar dois ou mais CT's que contemple $0 < x < y$ - Classe Válida

Casos de Testes	**Sangria**	**Quantia no caixa**
CT20	9,99	10,00
CT21	300.999,99	500.999,99

- Criar dois ou mais CT's que contemple $y = 0 < x$ - Classe Inválida

Casos de Testes	**Sangria**	**Quantia no caixa**
CT22	50,00	0,00
CT23	0,50	0,00

- Criar dois ou mais CT's que contemple $0 < y < x$ - Classe Inválida

Casos de Testes	**Sangria**	**Quantia no caixa**
CT24	0,02	0,01
CT25	10,00	5,00

- Criar dois ou mais CT's que contemple $x < 0$ - Classe Inválida

Casos de Testes	**Sangria**
CT26	- 0,01
CT27	-10,00

- Criar dois ou mais CT's que contemple $y < 0$ - Classe Inválida

Casos de Testes	**Quantia no caixa**
CT28	- 0,02
CT29	-0,10

Continuando a executar o Algoritmo 4.1, é necessário criar casos de testes que avaliem mais alguns dos possíveis valores para o campo. Assim, a fim de avaliar todas as possibilidades de limites aceitas para o campo Sangria, será adotada a fixação da quantia no caixa em seu valor máximo: 9.999.999,99. Então, tem-se que:
Limite do intervalo: 0,00 $\leq$ Sangria $\leq$ 9.999.999,99.

- Criar CT's com valores limite do intervalo - Classe válida.

Casos de Testes	**Sangria**
(Caso já contemplado anteriormente)	0,00
CT30	9.999.999,99

- Criar dois ou mais CT's com valores superiores ao limite mínimo - Classe Válida.

Casos de Testes	**Sangria**
CT31	0,01
CT32	0,02

- Criar dois ou mais CT's com valores inferiores ao limite máximo - Clásse Válida

Casos de Testes	**Sangria**
CT33	9.999.999,98
CT34	9.999.999,00

- Criar dois ou mais CT's com valores intermediários ao intervalo - Clásse Válida

Casos de Testes	**Sangria**
CT35	1.000,02
CT36	1.987.876,09

- Criar dois ou mais CT's com valores inferiores ao limite mínimo - Clásse Inválida

Casos de Testes	**Sangria**
Caso já contemplado anteriormente.	-0,01
Caso já contemplado anteriormente.	-0,02

- Criar dois ou mais CT's com valores superiores ao limite máximo - Classe Inválida

Casos de Testes	**Sangria**
CT36	10.000.000,00
CT37	10.000.000,01

- Criar CT's para valores muito próximos de zero.
  Como o limite mínimo é próximo de zero, já foram criados casos de testes que contemplem esse caso.
- Criar CT's para valores muito próximos dos limites estabelecidos - adotando o critério de precisão definido.
  Como só são permitidas duas casas decimais, o critério de precisão também foi atendido e esse item também já foi contemplado.

Aplicando o Algoritmo 4.9:

- Criar CT's que contemplem domínios de entrada e saída com tipos diferentes do especificado - Classe Inválida

Casos de Testes	**Sangria**
CT38	AE
CT39	l%$"""'@!#*≤≥≺≻ ◃▹ ≠≡≈≅∝

Assim, pode-se considerar que, a partir do Requisito XII foram gerados 39 casos de testes utilizando TFSE, enquanto que no roteiro, apenas 2 casos de testes. É interessante ressaltar que, os dados de testes propostos pelo Roteiro também foram contemplados pelos casos de teste gerados, sendo portanto, um complemento dos testes existentes.

Abaixo, será apresentado um outro exemplo utilizando TFSE, que irá abordar o item 3 requisito XXI.

Descrição do Teste 058 referente ao Requisito XXI

- **REQUISITO XXI - ITEM 3**:
  Recusar valor negativo ou nulo nos campos:
  a) valor unitário da mercadoria ou do serviço;
  b) quantidade da mercadoria ou do serviço;
  c) meios de pagamento;

- **TESTE 058:** Emissão de Cupom Fiscal com valor negativo ou nulo (zero) na quantidade do item.
  Passo 1: Abra um Cupom Fiscal.
  Passo 2: Registre um item comercializado.
  Passo 3: No campo relativo à quantidade comercializada, tente digitar um valor nulo (zero) e depois tente digitar um valor negativo.
  Condição para requisito atendido: Rejeição de valor nulo (zero) e de valor negativo.
  Condição para requisito não atendido: Permissão do registro com valor nulo (zero) ou negativo.

- **Caso de teste derivado a partir do Roteiro**

  As Figuras 5.13 e 5.14 representam a execução dos testes pelo testador da equipe de testes do órgão credenciado.

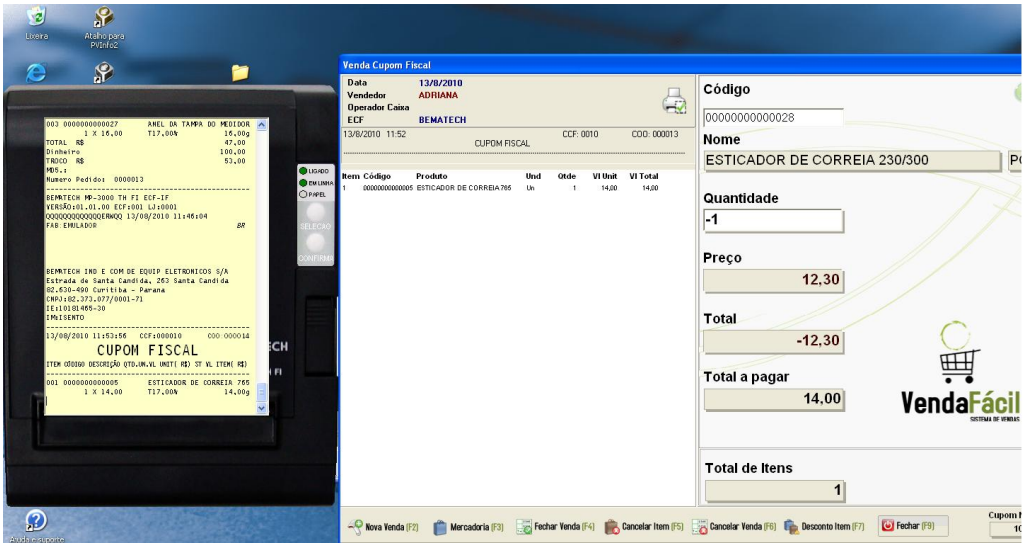


Figura 5.13: *Execução do teste 058 do roteiro (ROTEIRO, 2010).*

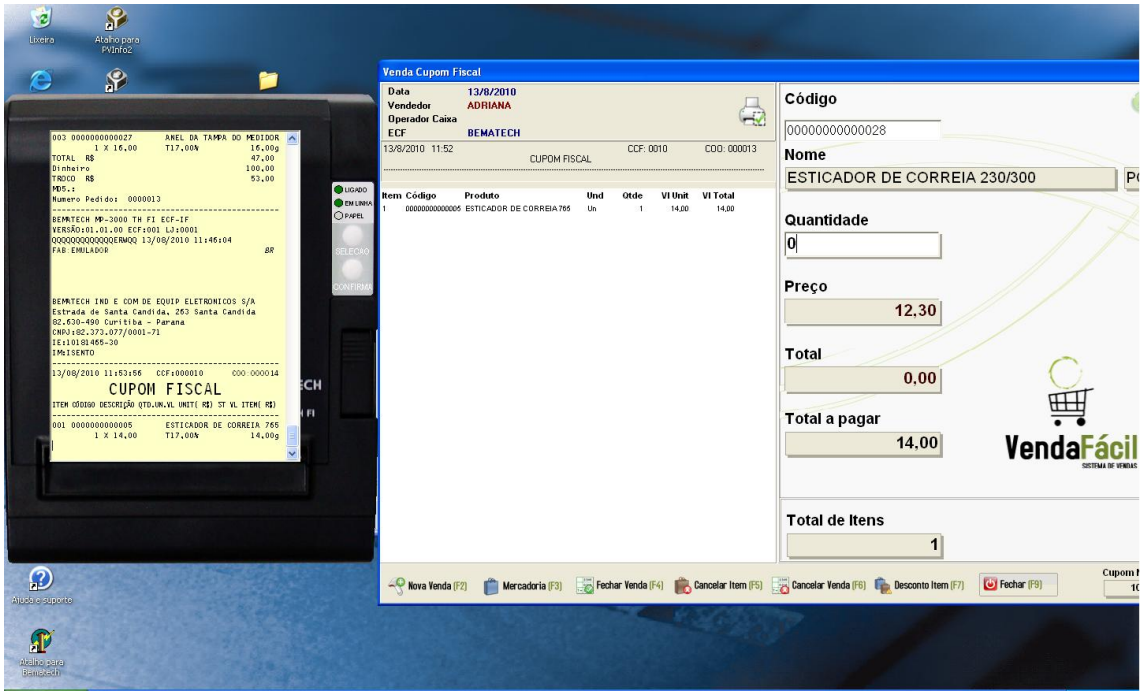


Figura 5.14: *Execução do teste 058 do roteiro (ROTEIRO, 2010).*

Novamente, é importante observar que, o testador seguiu literalmente os passos descritos no roteiro e essa execução caracteriza a condição para requisito atendido. E diferente do testes anteriores, não sugere valores específicos, e apesar disso, o testador forneceu apenas um valor aleatório, não contemplando mais do que um possível valor, minimizando a possibilidade de defeitos serem encontrados.

Casos de Testes Gerados utilizando TFSE.

Nesse caso, o TFSE será utilizado tanto para contemplar as possibilidades dos valores negativos, como também para as possibilidades dos valores positivos, tornando assim, os testes mais completos.

É interessante ressaltar que, nesse requisito, há uma peculiaridade, relacionada ao negócio, em que a quantidade vendida não pode ser superior a quantidade em estoque. Então, além de realizar testes avaliando as possibilidades para o campo quantidade, também serão descritos dados de testes que contemplem essa regra. Considerando as diretrizes do TFSE, verifica-se que o campo "Quantidade Vendida" é do tipo numérico, sendo possível portanto, aplicar os Algortimos 4.1 e 4.9.

Aplicando o Algoritmo 4.1:

Limite do intervalo: *(Quantidade Vendida)≤ (Quantidade em Estoque)*
O Algoritmo 4.1 referencia intervalos variáveis, onde o valor de uma variável x depende da variável y. Então, no caso dessa peculiaridade de negócio, como a quantidade vendida depende da quantidade em estoque, tem-se que a variável x equivale à quantidade vendida e a variável y equivale à quantidade em estoque.

- Criar CT que contemple $x = y = \mathbf{1}$ - Classe Válida

  Observe que, para esse caso, o Algoritmo 4.1 foi alterado para atender o limite mínimo do campo, substituindo o valor 0 por 1.

Casos de Testes	**Qtde. Vendida**	**Qtde. Estoque**
CT1	1	1

- Criar dois ou mais CT's que contemple $x = 1 < y$ - Classe Válida

Casos de Testes	**Qtde. Vendida**	**Qtde. Estoque**
CT2	1	2
CT3	1	500.001

- Criar dois ou mais CT's que contemple $0 < x = y$ - Classe Válida

Casos de Testes	**Qtde. Vendida**	**Qtde. Estoque**
CT4	10	10
CT5	459.999	459.999

- Criar dois ou mais CT's que contemple $0 < x < y$ - Classe Válida

Casos de Testes	**Qtde. Vendida**	**Qtde. Estoque**
CT6	9	10
CT7	305	506

- Criar dois ou mais CT's que contemple $y = 1 < x$ - Classe Inválida

Casos de Testes	**Qtde. Vendida**	**Qtde. Estoque**
CT8	50	1
CT9	1	1

- Criar dois ou mais CT's que contemple $1 < y < x$ - Classe Inválida

Casos de Testes	Qtde. Vendida	Qtde. Estoque
CT10	20	5
CT11	15	4

- Criar dois ou mais CT's que contemple $x < 0$ - Classe Inválida

Casos de Testes	Qtde. Estoque
CT12	- 1
CT13	-50

- Criar dois ou mais CT's que contemple $y < 0$ - Classe Inválida

Casos de Testes	Qtde. Vendida
CT14	- 2
CT15	- 10

Continuando a executar o Algoritmo 4.1, é necessário criar casos de testes que avaliem mais alguns dos possíveis valores para o campo. Assim, a fim de avaliar todas as possibilidades de limites aceitas para o campo: Quantidade Vendida, será adotada a fixação da quantia no caixa em seu valor máximo: 9.999.999. Então, tem-se que:
Limite do intervalo: $1 \leq$ Quantidade Vendida $\leq 9.999.999$.

- Criar CT's com valores limite do intervalo - Classe válida.

Casos de Testes	Qdte. Vendida
(Caso já contemplado anteriormente)	1
CT16	9.999.999

- Criar dois ou mais CT's com valores superiores ao limite mínimo - Classe Válida.

Casos de Testes	Qtde. Vendida
CT17	2
CT18	3

- Criar dois ou mais CT's com valores inferiores ao limite máximo - Clásse Válida

Casos de Testes	Qtde. Vendida
CT19	9.999.999
CT20	9.999.998

- Criar dois ou mais CT's com valores intermediários ao intervalo - Clásse Válida

Casos de Testes	**Qtde. Vendida**
CT21	1.005
CT22	8.987.659

- Criar dois ou mais CT's com valores inferiores ao limite mínimo - Clásse Inválida

Casos de Testes	**Qtde. Vendida**
Caso já contemplado anteriormente.	- 2
Caso já contemplado anteriormente.	- 10

- Criar dois ou mais CT's com valores superiores ao limite máximo - Classe Inválida

Casos de Testes	**Qtde. Vendida**
CT23	10.000.000
CT24	10.000.001

- Criar CT's para valores muito próximos de zero.
  Como o limite mínimo é próximo de zero, já foram criados casos de testes que contemplem esse caso.
- Criar CT's para valores muito próximos dos limites estabelecidos - adotando o critério de precisão definido.
  Como só são permitidas duas casas decimais, o critério de precisão também foi atendido e esse item também já foi contemplado.

Aplicando o Algoritmo 4.9:

- Criar CT's que contemplem domínios de entrada e saída com tipos diferentes do especificado - Classe Inválida

Casos de Testes	**Qtde. Vendida**
CT25	$$$$
CT26	venda%$”””’@!#*≤≥▷≠≡≈≅

Observa-se que, ao utilizar TFSE para derivar testes para o Item 3 do Requisito XXI, foram gerados 26 casos de testes, em contrapartida de 2 casos de testes exigidos pelo roteiro.

Contabilizando os testes gerados:

Para os dois requisitos descritos, o roteiro sugere 3 casos de testes para atendê-los. Porém, ao utilizar TFSE foram gerados 65 casos de teste, um aumento de 62 casos de testes, ou seja, aproximadamente, 20 vezes mais casos de testes do que demandado pelo roteiro. Assim, ao estender essa proporção para todo o roteiro, deveriam ser executados aproximadamente 2000 casos de testes a mais do que exigidos. Essa projeção resulta em um aumento no tempo e custo da atividade de testes. Em contrapartida, considerando que 21% da atividade de manutenção é consequência direta da execução de testes insuficientes (EVERETT; JR., 2007), aumentar a qualidade do conjunto de testes gerados agregaria mais qualidade ao *software*, reduzindo assim, o custo com a atividade de manutenção e aumentando portanto, a possibilidade de encontrar tentativas de burlar a legislação em vigor, objetivo principal do desenvolvimento e uso do roteiro de teste.

Revisão do Roteiro a partir do TFSE

Observa-se que, os casos de testes podem ser construídos no mesmo padrão adotado pelo roteiro: ou seja, passos e condições para o sucesso do teste. Assim, com os dados gerados acima, é possível criar casos de teste, considerando que, o preenchimento do campo corresponde a um passo. Por exemplo, casos de teste referentes aos dados de teste enunciados como CT5 e CT39 seriam descritos da seguinte forma:

- **TESTE 005:** Registro de Sangria ou Retirada de Caixa.
  Passo 1: Localize nos menus do programa a opção que permite registrar sangria ou retirada de caixa.
  Passo 2: Registre uma sangria ou retirada de caixa no valor de R$ 9.999.999,98. Observe se o ECF emitiu o Comprovante Não Fiscal relativo à sangria de caixa corretamente.
  Condição para requisito atendido: Emissão do Comprovante Não Fiscal de Suprimento de Caixa no valor de R$ 9.999.999,98.
  Condição para requisito não atendido: Inexistência de função para registro de Suprimento de Caixa ou falta de emissão do Comprovante Não Fiscal de Suprimento de Caixa.

- **TESTE 039:** Registro de Sangria ou Retirada de Caixa.
  Passo 1: Localize nos menus do programa a opção que permite registrar sangria ou retirada de caixa.
  Passo 2: Registre uma sangria ou retirada de caixa com o valor

l%$”””’@!#*≤≥≺≻ ◃▹ ≠≡≈≅∝. Observe se o ECF emitiu o Comprovante Não Fiscal relativo à sangria de caixa corretamente.
Condição para requisito atendido: Não emissão do cupom fiscal ou impossibilidade de inserção do valor..
Condição para requisito não atendido: Emissão do cupom fiscal.

5.3 Aspectos de Automação

É evidente que, a geração de um bom conjunto de testes depende da criatividade, conhecimento do negócio, habilidade e experiência do testador. Porém, apesar da execução manual dos testes explorar diversas possibilidades do negócio envolvido na implementação, essa atividade é bastante tediosa a quem a executa. Além disso, executar uma grande número de testes, demanda muito tempo, tornando a atividade dispendiosa. Assim, uma boa solução seria a automação dos testes, possibilitando a execução de um número maior de testes e reduzindo o tempo gasto na atividade. E, a execução manual se reduziria a explorar alguns casos específicos, permitindo ao testador que, a maior parte do tempo seja gasto com a especificação dos casos de teste, onde técnicas novas poderiam ser utilizadas.

Segundo (EVERETT; JR., 2007), ferramentas de automação de testes são uma coleção de produtos de software designados especialmente para auxiliar testadores e gerentes de testes em diferentes aspectos no processo de desenvolvimento de software. Assim, atualmente, no mercado, há inúmeras ferramentas que auxiliam ou automatizam alguma parte do processo de testes.

Nesse contexto, como forma de agilizar o processo de documentação dos testes, nesse seção, será proposto um documento padrão para a geração automática de dados de testes. A linguagem utilizada será XML (e**X**tensible **M**arkup **L**anguage), considerada uma meta linguagem de editoração, recomendada pelo consórcio W3C (W3C-XML, 2011), e reconhecida como padrão de publicação e intercâmbio de documentos, sendo adotada como o padrão utilizado neste trabalho para a produção de documentos contendo informações sobre o sistema em testes.

5.3.1 Linguagem XML e TestLink

Assim como outras linguagens de marcação, XML utiliza tags, instruções embutidas no corpo de documentos, possibilitando que dados sejam descritos. XML tem como base linguagens mais antigas como SGML (**S**tandard **G**eneralized **M**arkup **L**anguage) e HTML (**H**iper**T**ext **M**arkup **L**anguage), sendo atualmente empregada

na representação de estruturas de dados estruturados e semi-estruturados e seu intercâmbio na Web.

O objetivo da linguagem XML é fornecer diversos dos benefícios presentes em SGML (ISO/IEC, 1986) não disponíveis em HTML, e fornecê-los de forma mais fácil de aprender e utilizar do que a SGML completa. Esses benefícios incluem um conjunto extensível de tags, elementos encadeados, e uma validação opcional da estrutura do documento em relação a um esquema. O princípio desse objetivo é resultado das metas de projeto para XML, colocadas na 2ª edição das Recomendações do W3C (W3C, 11/2005) para XML, que compreende (FILHO, 2004):

1. Deve ser diretamente usável na Internet;
2. Deve prover suporte a ampla variedade de aplicações;
3. Deve ser compatível com a SGML;
4. Deve ser fácil escrever programas que processem documentos XML;
5. A quantidade de recursos adicionais na XML deve ser mantida ao mínimo absoluto, idealmente zero;
6. Documentos em XML devem ser legíveis e claros para pessoas;
7. O projeto de XML deve ser preparado rapidamente;
8. O projeto de XML deve ser conciso;
9. Documentos em XML devem ser fáceis de criar;
10. Concisão na marcação XML tem importância mínima.

Um documento XML pode ser facilmente manipulado pelas aplicações de software o que torna possível atingir níveis de automação bastante elevados. Nesse sentido, será proposto um documento de forma que seja manipulado pela ferramenta TestLink (TESTLINK, 2010).

TestLink é uma ferramenta *Open Source* de gestão de teste, que permite criar, gerenciar e executar casos de testes e organizá-los em plano de teste. Nestes planos, é possível associar testadores a casos de testes específicos, realizar a rastreabilidade dos casos de testes e requisitos, além de emitir relatórios da execução dos testes, possibilitando o gerenciamento dos defeitos encontrados. Também pode ser utilizada para gerenciar as métricas de defeitos encontrados, pois permite emitir relatórios com as mais variadas possibilidades de filtro, permitindo assim, que o responsável tenha real visão do software em homologação.

5.3.2 Padrão Proposto

O documento proposto conforme o Código 5.1, define casos de testes com id interno (*internalid*) e id externo (*externalid*) - respectivamente, contador de

casos de testes de um projeto intrínseco ao funcionamento do sistema e o contador visível ao usuário. Há também o número do nó (*node_order*) que realiza o papel de contador global dos casos de testes, o resumo do caso de teste, as pré-condições, o tipo de execução (se é manual ou automática), a prioridade, passos e referente a cada passo há o número do passo, sua ação e respectivo resultado.

Código XML 5.1 Documentação de Testes - Exemplo 1.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <testcases>
3    <testcase internalid="44" name="Campos tamanho minimo - 1 caractere">
4      <node_order><![CDATA[100]]></node_order>
5      <externalid><![CDATA[159]]></externalid>
6      <summary><![CDATA[]]></summary>
7      <preconditions><![CDATA[]]></preconditions>
8      <execution_type><![CDATA[1]]></execution_type>
9      <importance><![CDATA[2]]></importance>
10     <steps>
11       <step>
12         <step_number><![CDATA[1]]></step_number>
13         <actions><![CDATA[<p>Inserir um dado com apenas um caractere nos
               campos: Codigo do Projeto, Nome do Projeto, Descricao, Numero do
               Contrato</p>]]></actions>
14         <expectedresults><![CDATA[]]></expectedresults>
15       </step>
16     </steps>
17   </testcase>
18 </testcases>
```

No Código 5.2, é apresentado um outro exemplo de documentação utilizando o padrão proposto. Nos dois exemplos são utilizados passos que avaliem o tamanho mínimo permitido para uma *string*.

Código XML 5.2 Documentação de Testes - Exemplo 2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <testcases>
3    <testcase internalid="45" name="Campos tamanho minimo - 2 caracteres">
4      <node_order><![CDATA[101]]></node_order>
5      <externalid><![CDATA[160]]></externalid>
6      <summary><![CDATA[]]></summary>
7      <preconditions><![CDATA[]]></preconditions>
8      <execution_type><![CDATA[1]]></execution_type>
9      <importance><![CDATA[2]]></importance>
10     <steps>
11       <step>
12         <step_number><![CDATA[1]]></step_number>
13         <actions><![CDATA[<p>Inserir um dado com dois caractere nos campos:
               Codigo do Projeto, Nome do Projeto, Descricao, Numero do Contrato
               </p>]]></actions>
14         <expectedresults><![CDATA[]]></expectedresults>
15       </step>
16     </steps>
17   </testcase>
18 </testcases>
```

Conforme mencionado anteriormente, esse padrão visa a atender o formato aceitável pela ferramenta TestLink. Assim, pode ser importado e editado pela

interface da ferramenta. As figuras 5.15 e 5.16, apresentam, respectivamente, a importação e a edição do caso de teste documentado no Código 5.1.

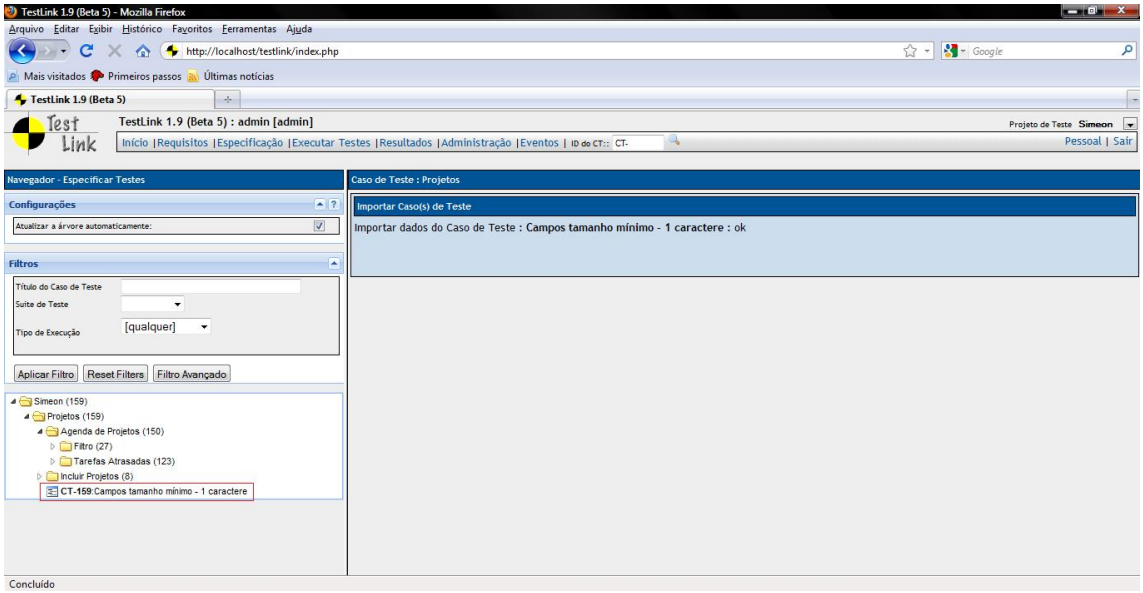


Figura 5.15: *Importação do documento em xml para a ferramenta Testlink.*

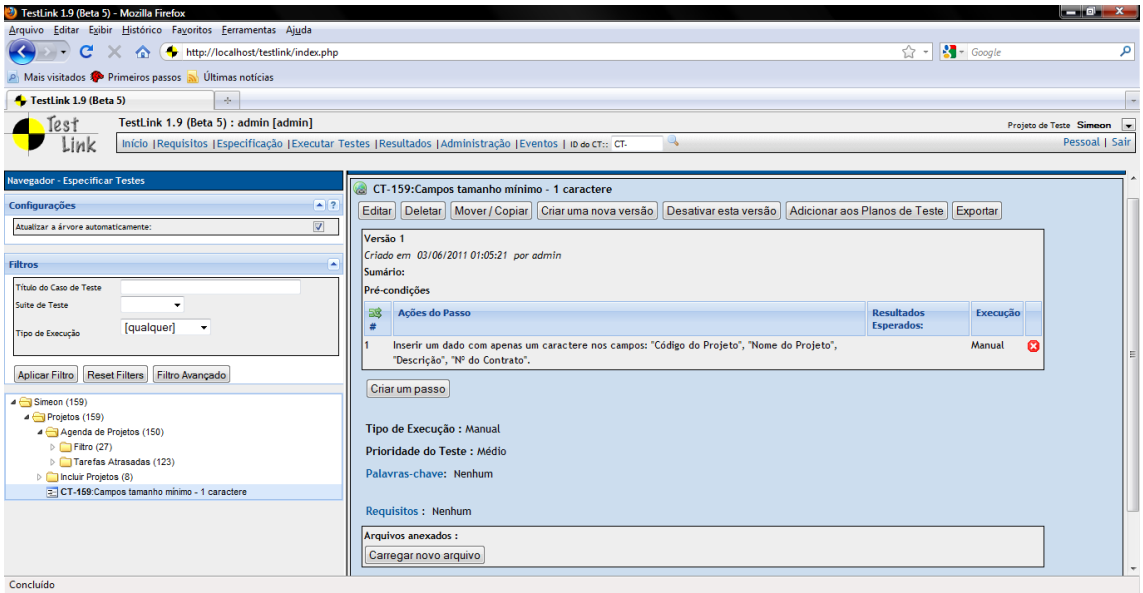


Figura 5.16: *Edição do documento em xml através da ferramenta Testlink.*

Utilizando o documento proposto, em xml, é possível documentar casos de testes alterando os identificadores interno e externo e informando as ações e passos

de cada caso de teste e que serão importados para a ferramenta TestLink. Com o auxílio da ferramenta, os resultados obtidos de cada execução são armazenados, sendo possível acessar o histórico de execução dos casos de testes. Assim, é possível agilizar o processo da atividade de documentação e execução dos casos de testes.

5.4 Considerações Finais

Esse capítulo apresentou a aplicação de TFSE em contextos distintos. No primeiro Estudo de Caso 5.1, foi evidenciada uma situação bastante comum nas empresas de desenvolvimento - documentação escassa - e a possibilidade da geração de dados de testes adotando as telas como fonte inicial e um oráculo descrevendo os resultados esperados para as funcionalidades. Portanto, apesar da empresa desenvolvedora alegar que há a realização de testes, adotou-se TFSE para realizar alguns testes e assim, foram descobertos alguns defeitos.

O segundo Estudo de Caso 5.2 apresentou o contexto de certificação de PAF-ECFs. Nessa situação, o roteiro de teste está pronto, aprovado pela entidade responsável e os órgão certificadores executam os testes segundo o roteiro para avaliar os *software* que desejam ser certificados. Nesse caso, o TFSE foi utilizado como referência de melhoria para o roteiro, pois, como foi evidenciado, os testes são limitados e os testadores não variam o conjunto de dados de testes proposto pelo roteiro. Assim, o TFSE pode ser aplicado para melhorar documentação já existente ou para avaliar a qualidade de um roteiro existente.

CAPÍTULO 6

Contribuições e Trabahos Futuros

Conforme descrito ao longo deste texto, a atividade de teste é de fundamental importância para assegurar a qualidade dos produtos de software desenvolvidos. Devido a vários motivos mencionados nos capítulos introdutórios, as empresas desenvolvedoras de software possuem mais familiaridade e facilidade em empregar critérios de teste funcionais para a validação e verificação de produtos de software. Mesmo que a quantidade de empresas que adotem tais critérios ainda seja pequena, o objetivo do presente trabalho foi apresentar uma forma sistemática de combinar critérios de teste funcionais e diretrizes de como aplicá-los de forma consistente, considerando os diferentes tipos de dados existentes.

Nesse sentido, este trabalho revisitou o trabalho do (LINKMAN et al., 2003), visando a sistematização do teste funcional no Capítulo 3, ampliou as diretrizes propostas no trabalho inicial, propôs abordagens algorítmica e gráfica objetivando tornar o TFSE intuitivo, conforme descrito no Capítulo 4, e adotou o TFSE em dois contextos diversos buscando avaliar a adoção da técnica em dois contextos distintos e ainda, auxilou na adoção de uma documentação voltada à automação e à utilização de ferramentas, descritos no Capítulo 5.

Desta forma, as principais contribuições do estudo aqui apresentado são:

- Representação dos passos do teste funcional sistemático em forma de algoritmo;
- Representação dos passos do teste funcional sistemático em forma de fluxograma;
- Extensão do Teste Funcional Sistemático a dados do tipo data;
- Extensão do Teste Funcional Sistemático a dados do tipo estruturado heterogêneo;
- Extensão do Teste Funcional Sistemático a dados do tipo Hora;
- Sugestão de divisão dos domínios de entrada e saída em classes de equivalência;
- Aplicação do Teste Funcional Estendido no contexto de softwares desenvolvidos para o mercado;

- Identificação de defeitos e sugestão de melhorias no Software de Planejamento Estratégico;
- Críticas e melhorias no Roteiro aprovado pela COTEPE;
- Levantamento do perfil dos envolvidos no processo de certificação do PAF-ECF;
- Levantamento de opinião sobre o Roteiro dos envolvidos no processo de certificação do PAF-ECF.
- Sugestão de documentação visando à automação de testes.

Para trabalhos futuros, algumas propostas foram levantas:

- A continuidade deste trabalho visa aplicar o TFSE em diversos outros tipos de software validando cada um dos possíveis caminhos de execução dos algoritmos propostos, aumentando as informações estatísticas sobre a relação custo/benefício de sua utilização.
- Aplicação deste trabalho no contexto de todo o roteiro do PAF-ECF validando-o por completo e assim, propondo melhorias consistentes e fundamentada em critérios de teste. O estudo aqui realizado para o contexto do PAF-ECF permitiu apenas dimensionar a necessidade de ampliação do roteiro tendo como base o TFSE ou outros critérios de teste que facilitem, não apenas a assimilação do roteiro, mas também a compreensão do por quê faz-se necessária a execução do software com determinado caso de teste.
- Investigação da composição de ferramentas de produtos de software de código aberto na criação de um ambiente de teste integrado de apoio ao TFSE. Tal ambiente, além de apoiar a transferência tecnológica também visará a condução de estudos comparativos e apoio a atividades didática para o ensino dos conceitos de teste de software.

Bibliografia

BORLAND. *Delphi Developer*. 2010. Página WEB. Disponível em: http://info.borland.com/devsupport/delphi/.

BROOKS, J. F. P. No silver bullet essence and accidents of software engineering. *Computer*, v. 20, n. 4, p. 10 –19, april 1987.

BURNSTEIN, I. *Practional Software Testing*. Verlag New York: Springer, 2003.

CMMI. *Capability Maturity Model Integration – Version 1.3*. [S.l.], 2010.

CONFAZ. *Conselho Nacional de Política Fazendária*. 2010. Página WEB. Disponível em http://www.fazenda.gov.br/confaz/default.htm.

CONVêNIO. *CONVÊNIO ICMS 15, DE 4 DE ABRIL*. 2008. Página WEB. Disponível em http://www.fazenda.gov.br/confaz/confaz/Convenios/ICMS/2008/cv015_08.htm.

COPELAND, L. *A Practitioner's Guide to Software Test Design*. [S.l.]: Artech House Publishers, 2004.

COTEPE. *ATO COTEPE/ICMS Nº 36, DE 10 DE SETEMBRO*. [S.l.], 2009.

COTEPE. *ATO COTEPE/ICMS Nº 46, DE 27 DE NOVEMBRO*. [S.l.], 2009.

CRAIG, R. D.; JASKIEL, S. P. *Systematic Software Testing*. [S.l.]: Artech House Publishers, 2002.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao Teste de Software*. Rio de Janeiro, RJ: Elsevier, 2007.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on test data selection: Help for the practicing programmer. *ieeec*, v. 11, n. 4, p. 34–43, abr. 1978.

EVERETT, G. D.; JR., R. M. *Software Testing Across the Entire Software Development Life Cycle*. 1. ed. [S.l.]: IEEE, 2007.

FILHO, A. M. da S. *Programando com XML-Leitura Recomendada para desenvolvedores de aplicações Web*. Rio de Janeiro: Campus, 2004.

FORBELLONE, A. L. V.; EBERSPäCHER, H. F. *Lógica da Programação - A Construção de Algoritmos e Estruturas de Dados*. [S.l.]: 3ª, 2005.

HUTCHESON, M. L. *Software Testing Fundamentals: Methods and Metrics*. [S.l.]: John Wiley & Sons, Inc., 2003.

IEEE. *IEEE Standard for Software Test Documentation*. [S.l.], set. 1998.

IEEE. *IEEE Standard Information Technology — Software Packages — Quality Requirements and Testing*. New York, 1998.

IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. [S.l.], 2002.

ISO/IEC. *ISO/IEC 8879 - Information Processing - Text and Office Systems - Standard Generalized Markup Language*. [S.l.], 1986.

ISO/IEC. *ISO/IEC 12207 - Software Life Cycle Processes*. [S.l.], 2004.

ISO/IEC. *ISO/IEC-15504 (SPICE)*. [S.l.], 2005.

ITKONEN, J.; MANTYLA, M. V.; LASSENIUS, C. How do testers do it? an exploratory study on manual testing practices. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2009. (ESEM '09), p. 494–497. ISBN 978-1-4244-4842-5. Disponível em: <http://dx.doi.org/10.1109/ESEM.2009.5314240>.

Java Script. *What is JavaScript?* 2010. Página WEB. Disponível em https://developer.mozilla.org/en/About_JavaScript.

KANER, C.; FALK, J.; NGUYEN, H. Q. *Testing Computer Software*. [S.l.]: Wiley, 1999.

LEMOS, A. *Estratégia de Testes de Software para Aplicação em Empresas de Desenvolvimento de Software*. Dissertação (Mestrado) — Universidade de Caxias do Sul, Caxias do Sul - RS, 2004.

LINKMAN, S.; VINCENZI, A. M. R.; MALDONADO, J. An evaluation of systematic functional testing using mutation testing. In: *7th International Conference on Empirical Assessment in Software Engineering – EASE*. [S.l.: s.n.], 2003.

MALDONADO, J. C.; BARBOSA, E. F.; VINCENZI, A. M. R. V.; DELAMARO, M. E.; SOUZA, S. d. R. S.; JINO, M. *Nota Didática-Introdução ao Teste de Software*. São Carlos, SP, Jan 2004.

MILLS, K. L. An experimental evaluation of specification techniques for improving functional testing. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 32, n. 1, p. 83–95, 1996. ISSN 0164-1212.

MOREIRA, T. R.; RIOS, E. *Projeto e & Engenharia de Software - Teste de Software*. [S.l.]: Alta Books, 2003.

MYERS, G. J.; SANDLER, C.; BADGETT, T.; THOMAS, T. M. *The Art of Software Testing*. 2. ed. [S.l.]: Wiley, New York, 2004.

MYSQL. *Why MySQL?* 2010. Página WEB. Disponível em http://www.mysql.com/.

NAIK, K.; TRIPATHY, P. *Software Testing and Quality Assurance: Theory and Practice*. [S.l.]: John Wiley & Sons, Inc., 2008.

PATTON, R. *Software Testing*. 2. ed. [S.l.]: Sams Publishing, 2005.

PAULK, M. C. *Capability Maturity Model for Software – Version 1.1*. [S.l.], fev. 1993.

PERRY, W. E. *Effective Methods for Software Testing*. 2. ed. [S.l.]: Wiley, 2000.

PHP. *What is PHP?* 2010. Página WEB. Disponível em http://www.php.net/.

PRESSMAN, R. S. *Engenharia de Software*. 6. ed. Rio de Janeiro: McGraw-Hill, 2006.

RAPPS, S.; WEYUKER, E. J. Data flow analysis techniques for test data selection. In: *Proceedings of the 6th international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1982. (ICSE '82), p. 272–278. Disponível em: <http://portal.acm.org/citation.cfm?id=800254.807769>.

REPASI, T. Software testing - state of the art and current research challanges. In: *5th International Symposium on Applied Computational Intelligence and Informatics – SACI'09*. [S.l.: s.n.], 2009. p. 47 –50.

ROCHA, A. *Desenvolver Metodologia para Testes de Homologação De Software*. 2005. Bolsa BITEC – Universidade Federal de Goiás.

ROTEIRO. *Roteiro de Análise Funcional de PAF-ECF*. 2010. Página WEB. Disponível em http://www.fazenda.gov.br/confaz/confaz/diversos/ROTEIRO_DE_ANALISE_DE_PAF-ECF_VERSAO_1_4.pdf.

SIMEON. 2010. Página WEB. Disponível em http://www.simeon.com.br/.

SOFTEX. *MPS.BR - Melhoria de Processo do Software Brasileiro*. 2010. Página WEB. Http://www.softex.br/mpsbr/_guias/default.asp.

SOMMERVILLE, I. *Software Enginnering*. 7. ed. Rio de Janeiro: Addison-Wesley, 2007.

STATMATH. *Cal*. 2010. Página WEB. Disponível em http://www.indiana.edu/ statmath/support/byos/unix/gettingstarted/5.html.

SWEBOK, A. I. *SWEBOK - Guide to Software Engineering Body of Knowledge*. California: IEEE - Computer Society, 2004.

TESTLINK. *TestLinkCommunity*. 2010. Página WEB. Disponível em http://www.teamst.org/.

TIAN, J. *Software Quality Engineering - Testing, Quality Assurance, and Quantifiable Improvement*. [S.l.]: IEEE Computer Society Publications, 2005.

TONDERING, C. *Frequently Asked Questions about Calendars*. 04 2011. Http://www.tondering.dk/claus/calendar.html.

VINCENZI, A. M. R. *Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação*. Dissertação (Mestrado) — ICMC/USP, 1998.

W3C. *World Wide Web Consortium*. `http://www.w3.org/`: [s.n.], 11/2005.

W3C-XML. *World Wide Web Consortium - XML*. `http://www.w3.org/xml/`: [s.n.], 04 2011. Http://www.w3.org/XML/.

WHITTAKER, J. A. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. [S.l.]: Addison-Wesley Professional, 2009. ISBN 0321636414, 9780321636416.

APÊNDICE A

Aplicação do TFSE na Funcionalidade Incluir Tarefas

Funcionalidade *Incluir Tarefas*.

Na Figura A.1 está representada a tela de Incluir Tarefas. Nessa tela há vários campos a serem testados, sendo que, inicialmente, serão considerados os campos Datas Previstas Inicial e Final, destacados na figura.

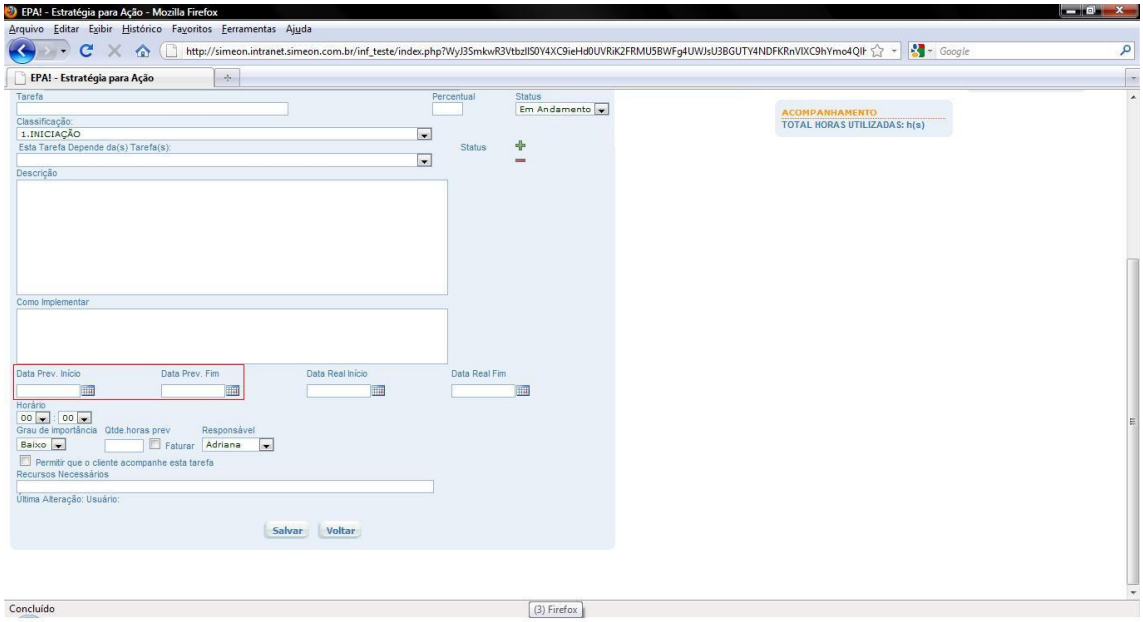


Figura A.1: *Formulário de inclusão de tarefas.*

1.Aplicação do Algoritmo 4.6:

(a)Criar CT's que contemplem datas válidas-atuais - Classe Válida:

A Figura A.2 representa a criação de dado de teste referente à data válida atual.

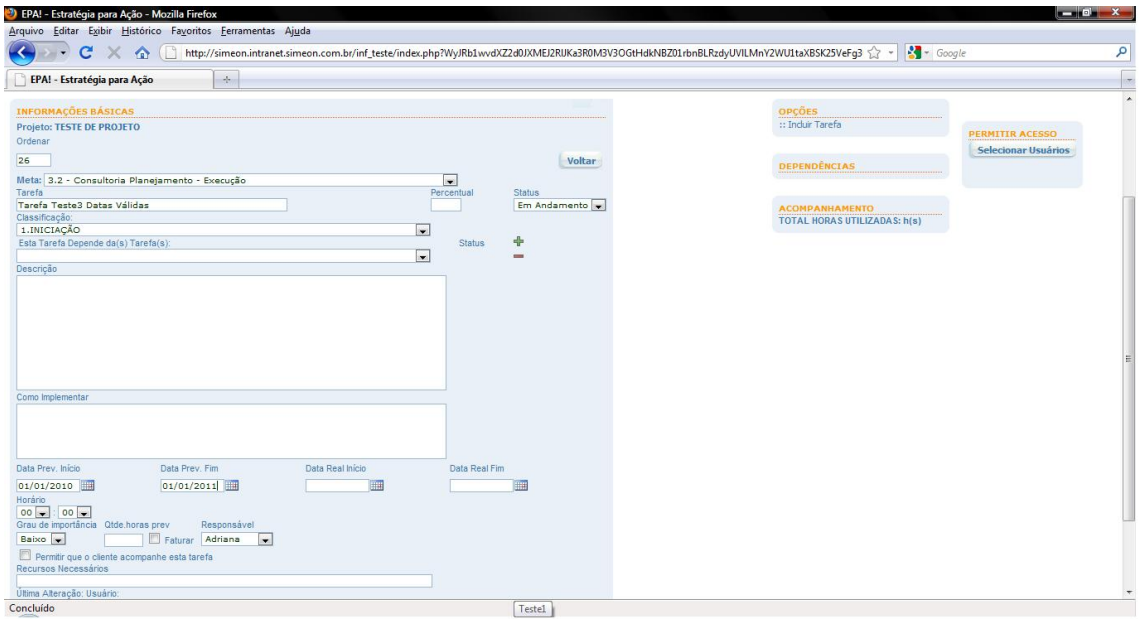


Figura A.2: *Data Válida Atual - CT1.*

(b)Criar CT's que contemplem datas válidas-anteriores - Classe Válida: A Figura A.3 representa a criação de dado de teste referente à data válida anterior em relação à data atual.

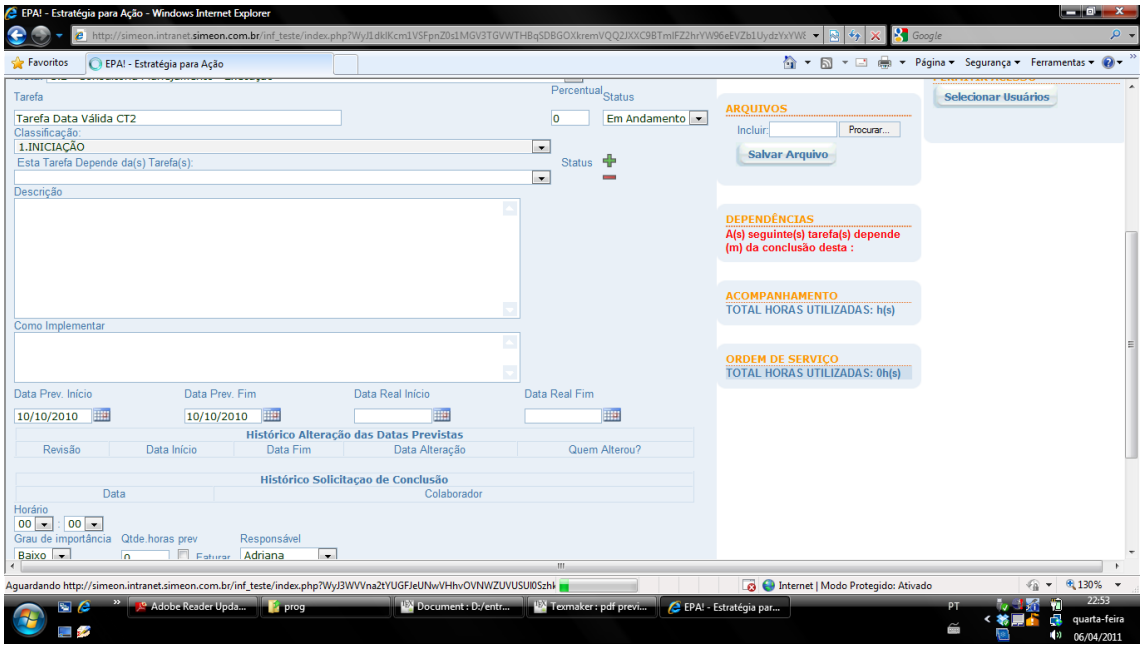


Figura A.3: *Data Válida Anterior à Data Atual - CT2.*

(c)Criar CT's que contemplem datas válidas-futuras - Classe Válida:
A Figura A.4 representa a criação de dado de teste referentes à data válida futura em relação à data atual.

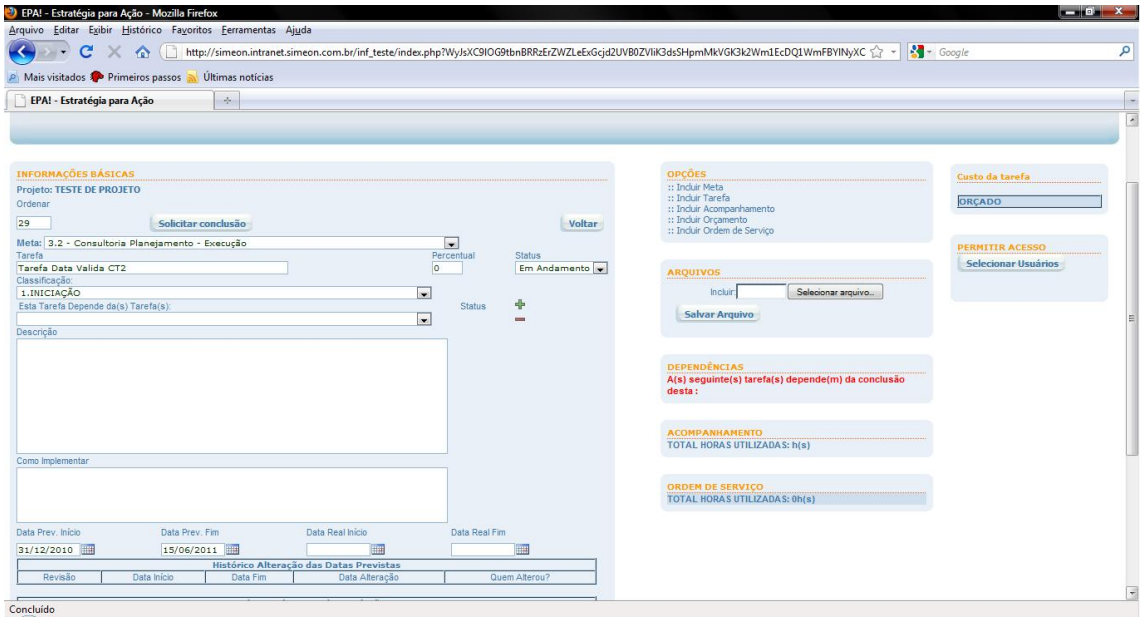


Figura A.4: *Data Válida Posterior à Data Atual - CT3.*

2.Aplicação do Algoritmo 4.1:
Para esse caso, o algoritmo será adaptado para atender peculiaridades do negócio. O algoritmo, foi descrito para atender situações em que, uma variável é dependente de outra, de forma que há um valor mínimo especificado (no caso o valor 0) e o limite máximo seria a variável que se depende. Conforme o algoritmo, o valor da variável x pode ir de 0 a y. Porém, no contexto dessa tela, o limite mínimo é definido por uma variável, assim, o valor da variável x pode ir de y até o limite máximo. Tem-se que, a Data Final depende da Data Inicial, de maneira que, o limite mínimo da Data Final é o valor da Data Inicial e o limite máximo é o aceito pelo campo.

A partir do Algoritmo 4.1 foi derivado o Algoritmo A.1 para melhor representar essa situação. Assim, cada passo define a criação de casos de teste referentes ao intervalo dos campos de Data Inicial e Data Final da tela A.1.

Algoritmo A.1: *Tipo de dado numérico*

```
1  se (intervalo for variável) então
2    /* Intervalo variável significa que o valor de uma determinada variável(
         Data Final) depende de outra variável(Data Inicial). */
3    Criar CT que contemple DataFinal = LimiteDataFinal = DataInicial // (Válida)
4    Criar dois ou mais CT's que contemple DataFinal = DataInicial < LimiteMaximo // (
         Válida)
5    Criar dois ou mais CT's que contemple DataInicial < DataFinal = LimiteMaximo // (
         Válida)
6    Criar dois ou mais CT's que contemple DataInicial < DataFinal < LimiteMaximo // (
         Válida)
7    Criar dois ou mais CT's que contemple LimiteMaximo = DataInicial < DataFinal // (
         Inválida)
8    Criar dois ou mais CT's que contemple DataInicial < LimiteMaximo < DataFinal // (
         Inválida)
9    Criar dois ou mais CT's que contemple DataFinal < DataInicial // (Inválida)
10   Criar dois ou mais CT's que contemple LimiteMaximo < DataInicial // (Inválida)
11 fimse
```

(a)Criar CT que contemple *DataFinal = LimiteDataFinal = DataInicial* - Classe Válida.

A Figura A.5 representa a criação de dado de teste referente à linha 3 do Algorimto A.1.

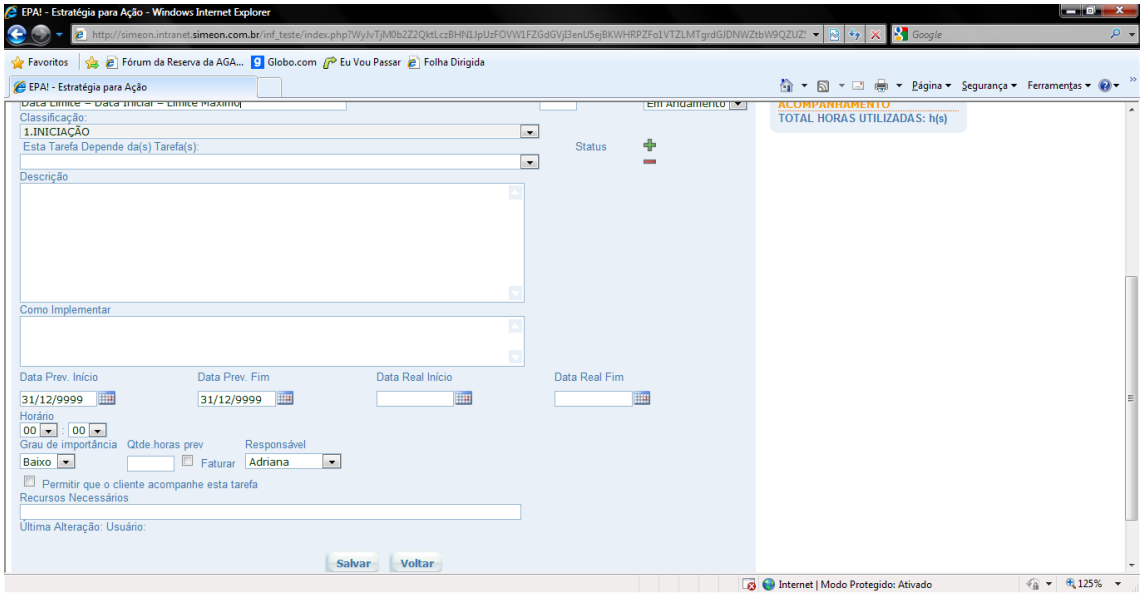


Figura A.5: *Data Final = Data Inicial = Limite - CT4.*

(b)Criar dois ou mais CT's que contemple *DataFinal = DataInicial < LimiteMaximo* - Classe Válida.

As Figuras A.6 e A.7 representam a criação de dados de teste referente à linha 4 do Algorimto A.1.

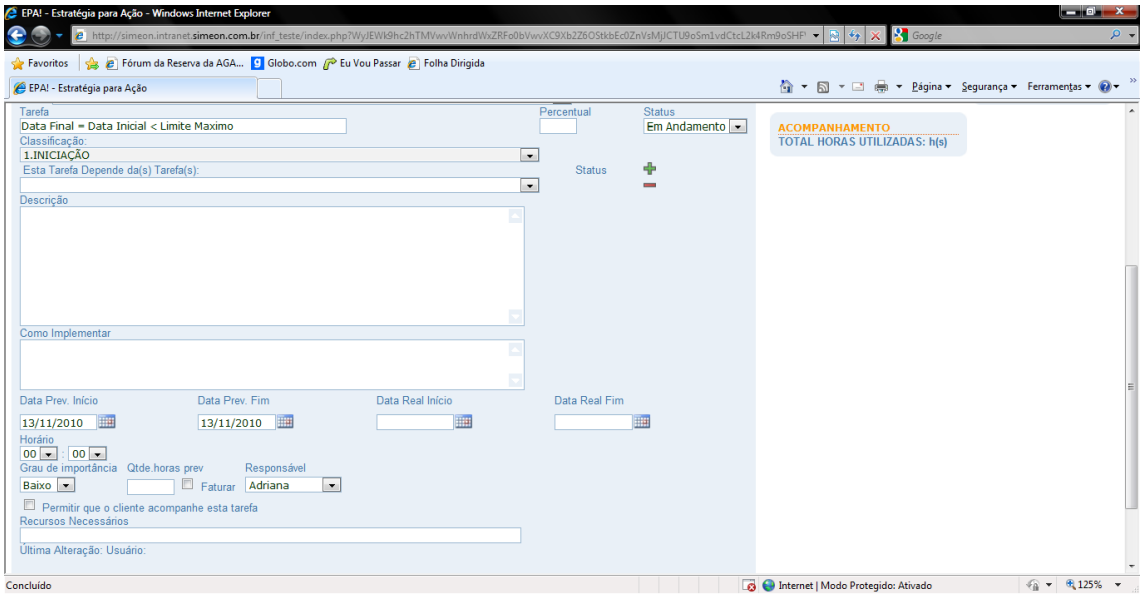


Figura A.6: *DataFinal = DataInicial < LimiteMaximo - CT5.*

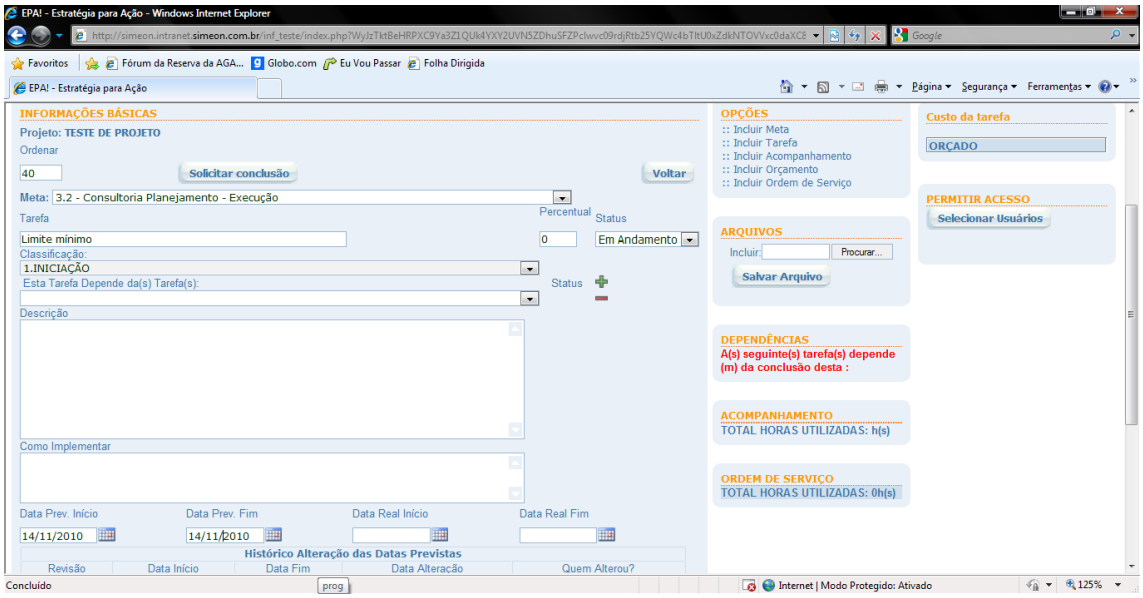


Figura A.7: *DataFinal = DataInicial < LimiteMaximo - CT6.*

(c)Criar dois ou mais CT's que contemple *DataInicial < DataFinal = LimiteMaximo* - Classe Válida.

As Figuras A.8 e A.9 representam a criação de dados de teste referente à linha 5 do Algorimto A.1.

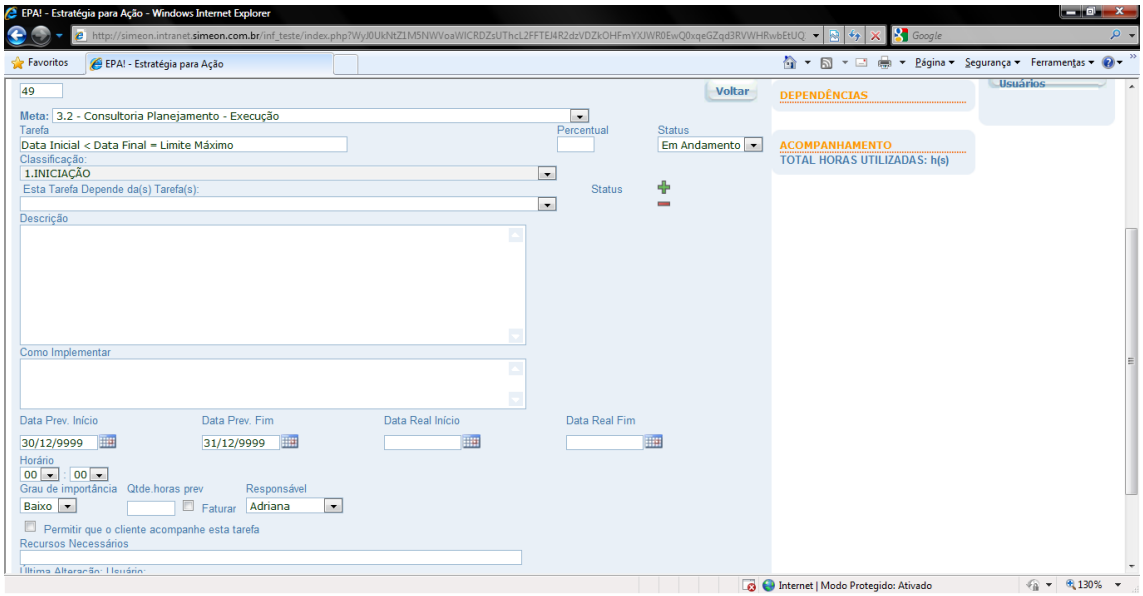


Figura A.8: *DataInicial < DataFinal = LimiteMaximo - CT7.*

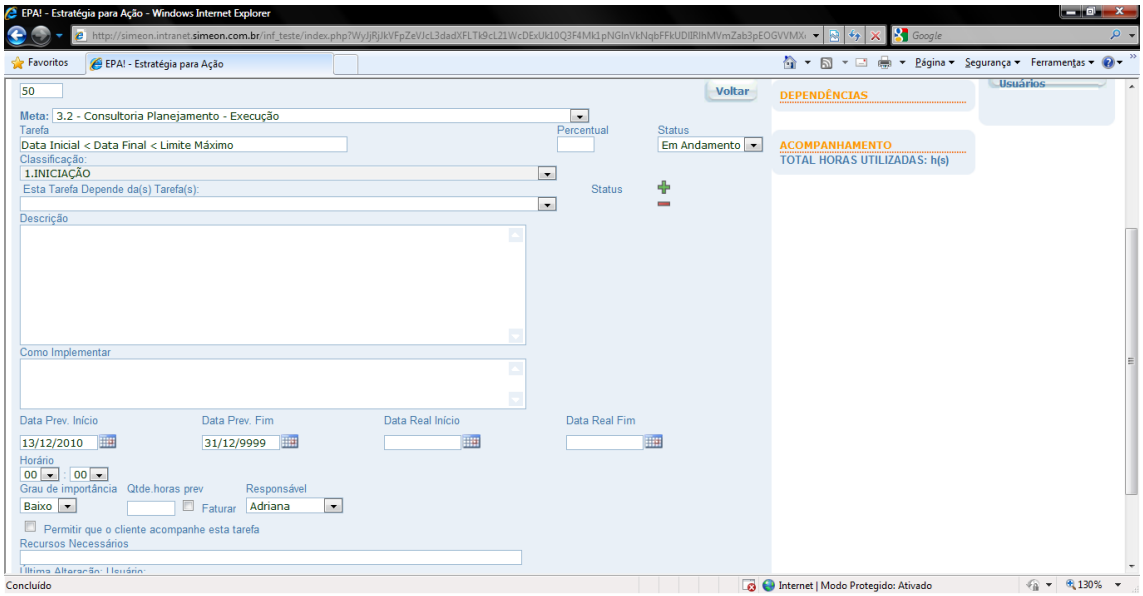


Figura A.9: *DataInicial < DataFinal = LimiteMaximo - CT8.*

(d)Criar dois ou mais CT's que contemple *DataInicial* < *DataFinal* < *LimiteMaximo* - Classe Válida.

As Figuras A.10 e A.11 representam a criação de dados de teste referente à linha 6 do Algorimto A.1.

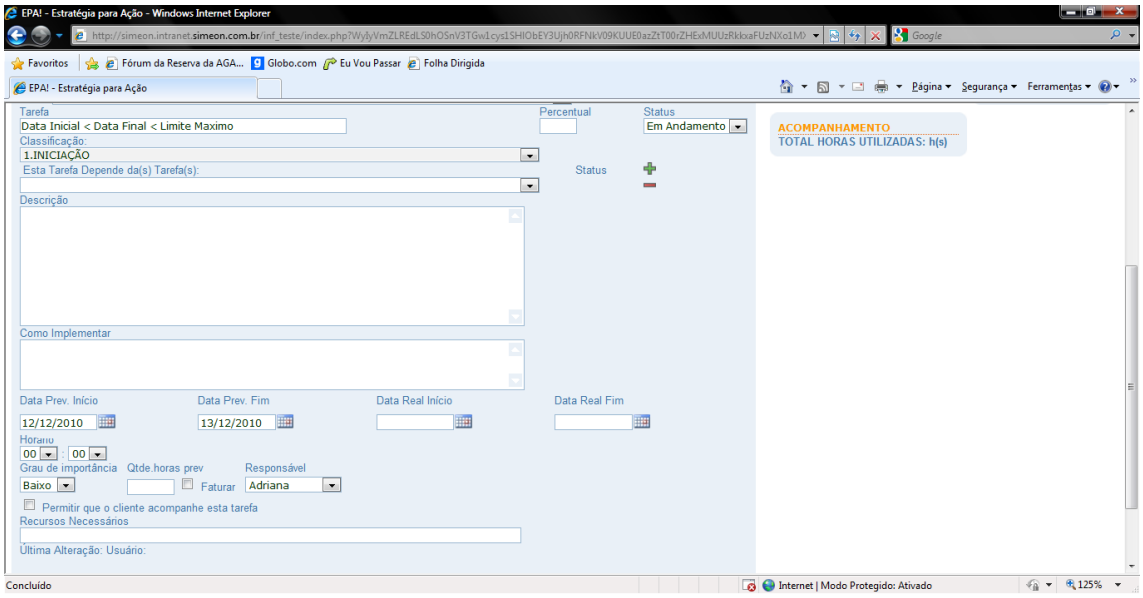


Figura A.10: *DataInicial* < *DataFinal* < *LimiteMaximo* - *CT9.*

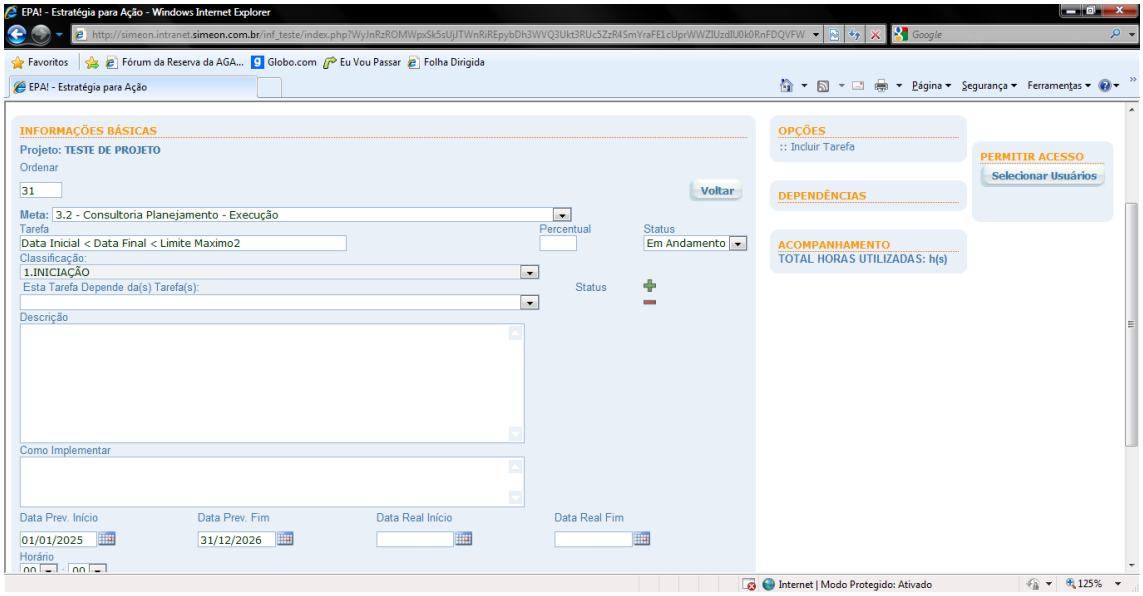


Figura A.11: *DataInicial* $<$ *DataFinal* $<$ *LimiteMaximo* - *CT10.*

(e)Criar dois ou mais CT's que contemple *LimiteMaximo* $=$ *DataInicial* $<$ *DataFinal* - Classe Inválida.

A Figura A.12 representa a criação do primeiro dado de teste referente à linha 7 do Algoritmo A.1. Observa-se que, nesse caso, que como não é possível inserir data válida maior que o limite máximo, foi utilizada uma data inválida como data final.

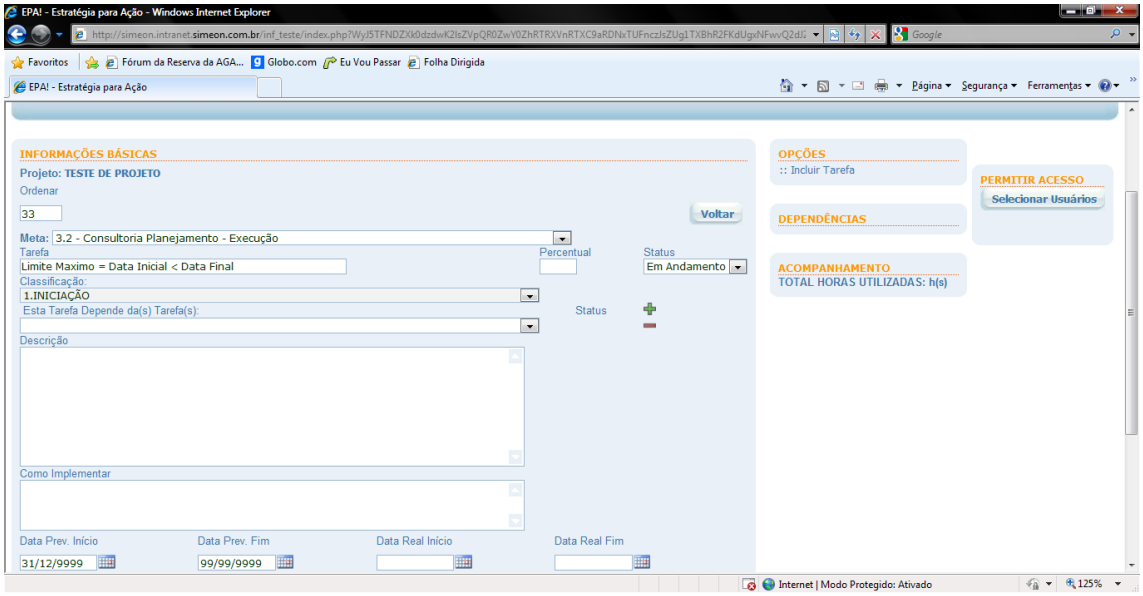


Figura A.12: *LimiteMaximo = DataInicial < DataFinal - CT11.*

A Figura A.13 representa a inconsistência gerada ao inserir uma data final superior ao limite máximo, ou seja, uma data inválida. Observou-se que, o sistema não apresenta mensagem alertando sobre a inconsistência do dado e apesar de ser um campo obrigatório, o dado não é armazenado.

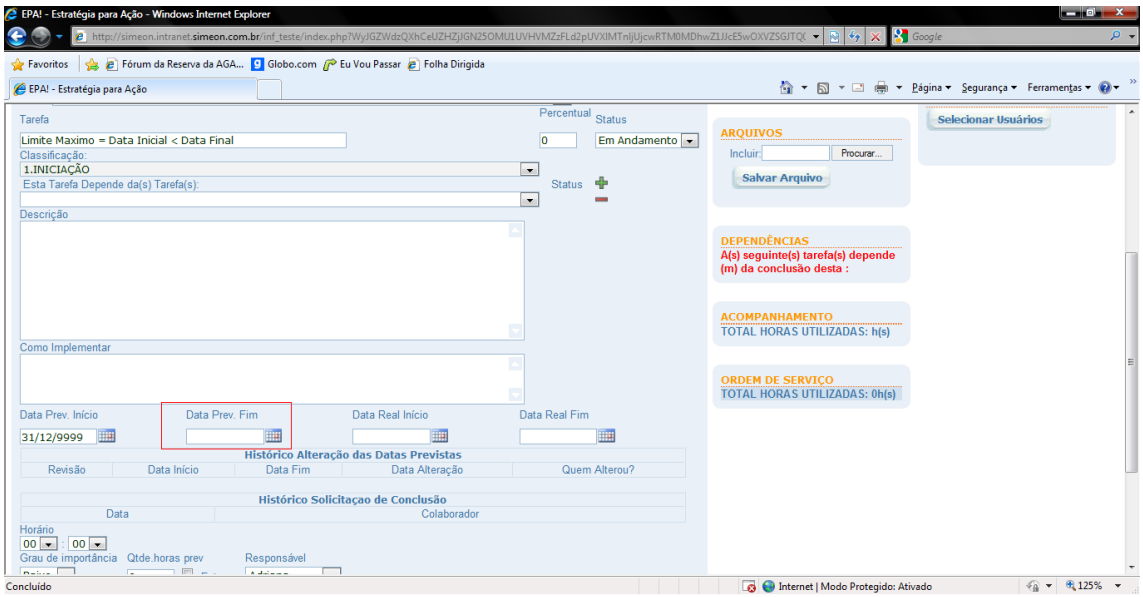


Figura A.13: *Tela de inconsistência gerada ao inserir Data Final superior ao limite máximo.*

A Figura A.14 representa a criação do segundo dado de teste referente à linha 7 do Algorimto A.1.

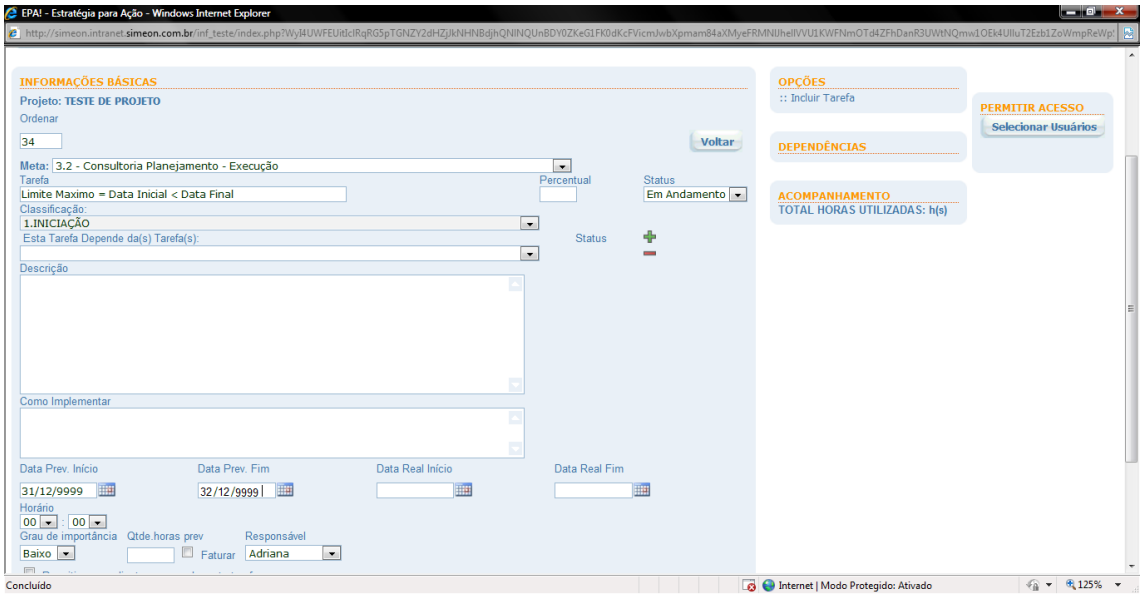


Figura A.14: *LimiteMaximo = DataInicial < DataFinal - CT12.*

Após inserção desses dados, observa-se a mesma inconsistência, representada pela Figura A.13. Assim, ao inserir uma data final superior ao limite máximo, ou seja, uma data inválida, o sistema não apresenta mensagem alertando sobre a inconsistência do dado e apesar de ser um campo obrigatório, o dado não é armazenado.

(f) Criar dois ou mais CT's que contemple *DataInicial < LimiteMaximo < DataFinal* - Classe Inválida.

As Figuras A.15 e A.16 representam a criação de dados de teste referente à linha 8 do Algorimto A.1.

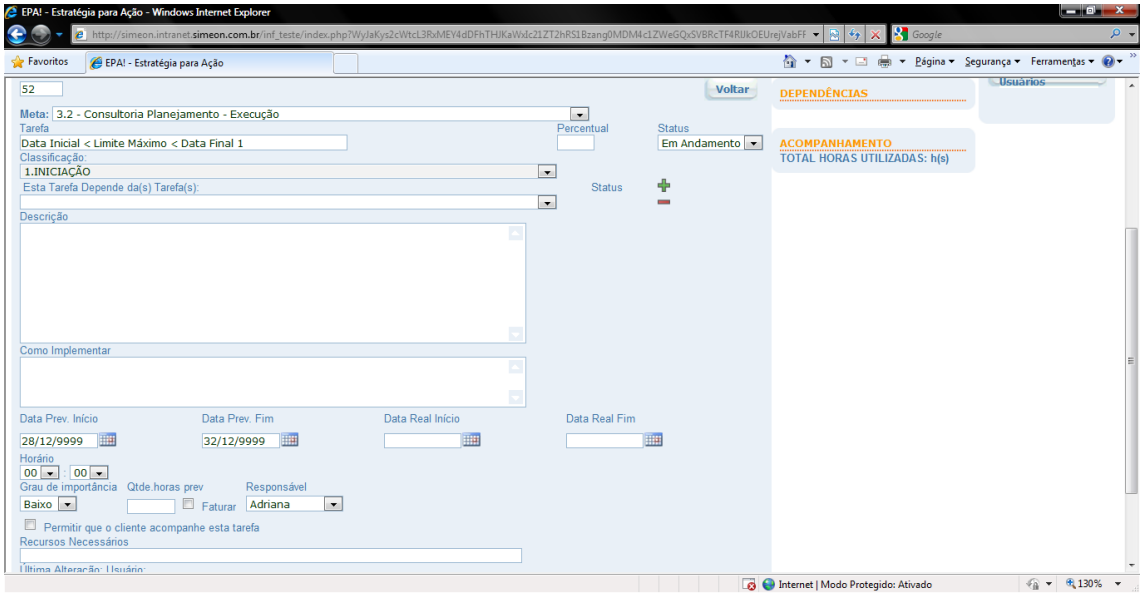


Figura A.15: *DataInicial < LimiteMaximo < DataFinal - CT13.*

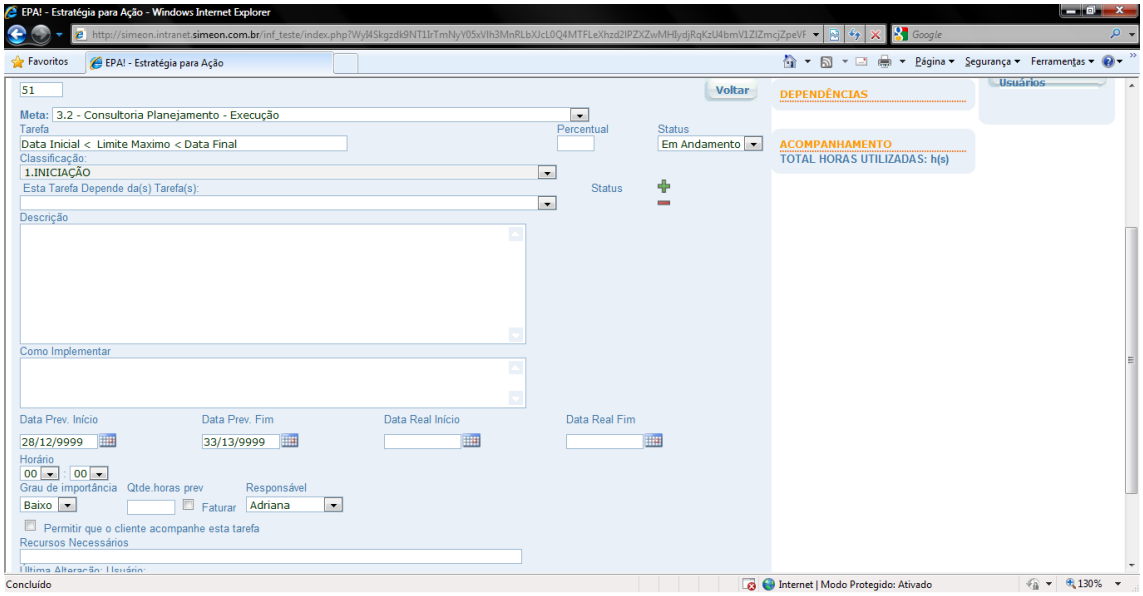


Figura A.16: *DataInicial < LimiteMaximo < DataFinal - CT14.*

Observa-se que, o sistema não efetua o cadastro, porém, não emite mensagem avisando ao usuário da inconsistência e apresenta o campo "Data Fim" em branco.

(g) Criar dois ou mais CT's que contemple *DataFinal < DataInicial* - Classe Inválida.

As Figuras A.17 e A.18 representam a criação de dados de teste referente à linha 9 do Algorimto A.1.

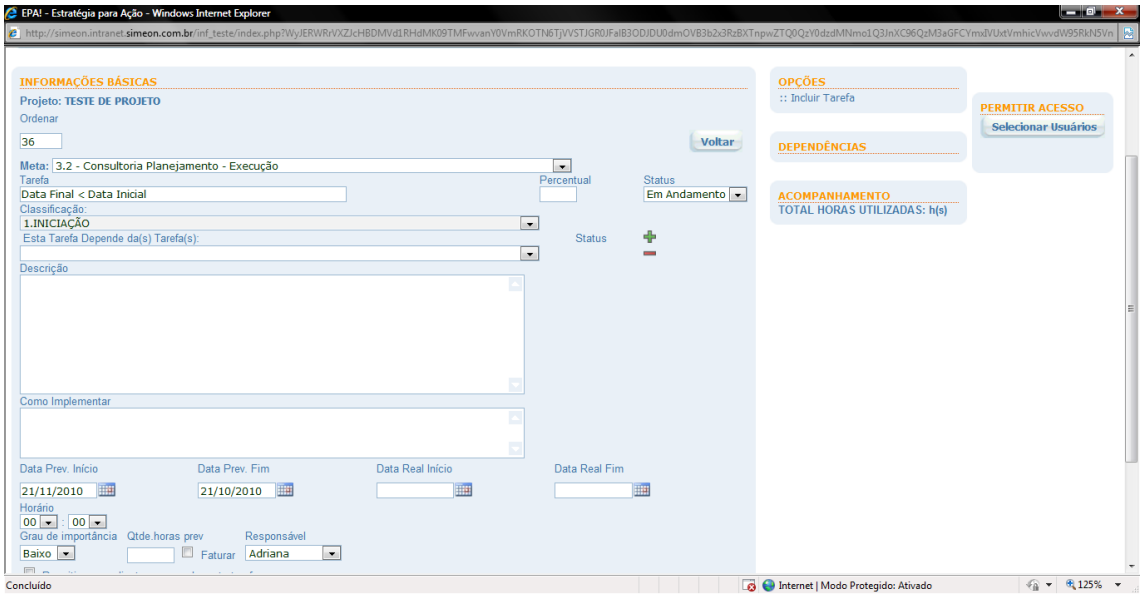


Figura A.17: *DataFinal < DataInicial - CT15.*

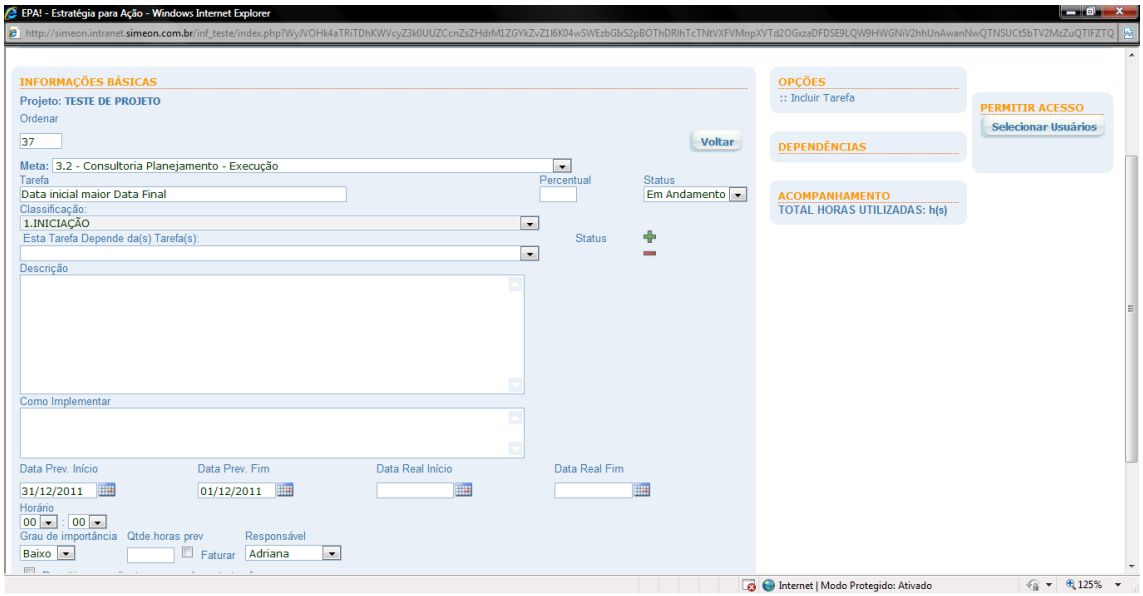


Figura A.18: *DataFinal < DataInicial - CT16.*

Observou-se que, não há a consistência dos dados em relação à comparação entre as Datas Iniciais e Finais e o cadastro das tarefas é concluído com sucesso.

(h) Criar dois ou mais CT's que contemple *LimiteMaximo* < *DataInicial* - Classe Inválida.

A Figura A.19 representa a criação do primeiro dado de teste referente à linha 10 do Algorimto A.1.

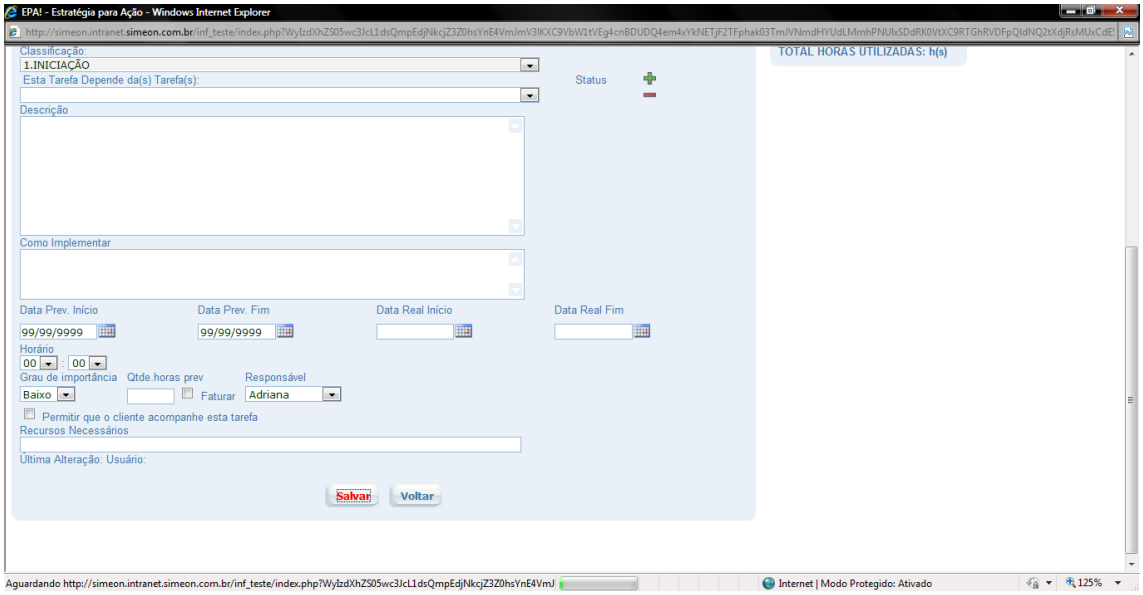


Figura A.19: *Limite Maximo < Data Inicial - CT17.*

Novamente, a Figura A.13 demonstra a inconsistência gerada. Observou-se que, não há emissão de mensagem notificando ao usuário da inconsistência dos dados e apesar dos campos serem obrigatórios, estes não são preservados.

A Figura A.20 representa a criação do segundo dado de teste referente à linha 10 do Algorimto A.1. Foram inseridos dados para a data inicial maiores do que os limites máximos aceitáveis para os campos referentes à dia, mês e ano.

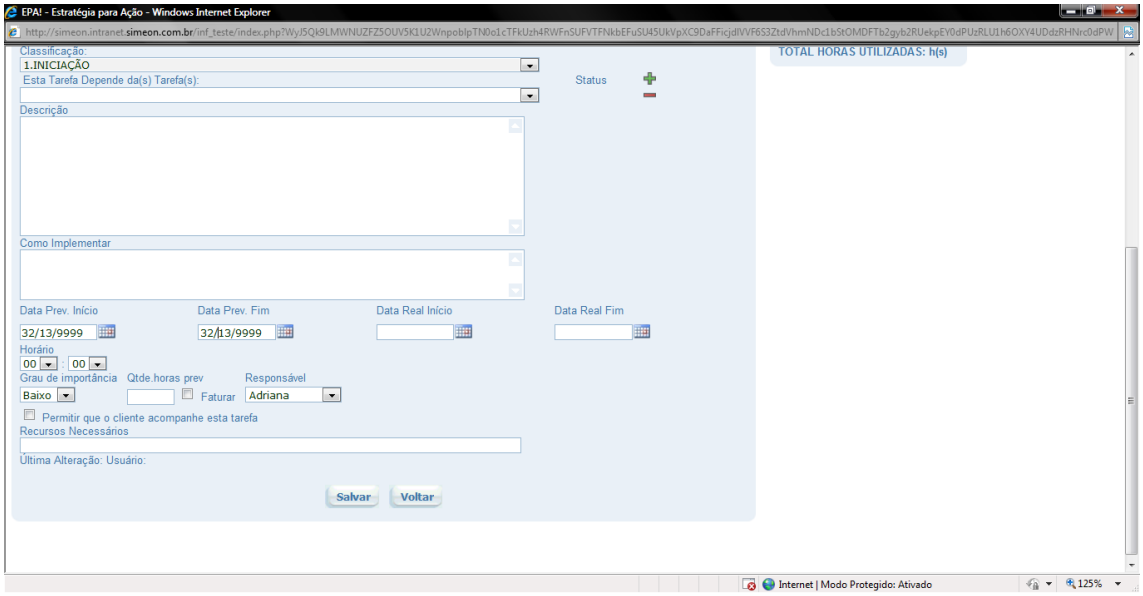


Figura A.20: *Limite Maximo < Data Inicial - CT18.*

Observou-se que, o sistema não emite mensagem avisando ao usuário da inconsistência dos dados e, após clicar em salvar a tarefa, o sistema apresenta os campos de data sem informação, conforme Figura A.13.

(i)Criar CT's que contemplem cada valor limite do intervalo - Classe Válida

Nas Figuras A.21 e A.22, são apresentados os dados de teste, da classe válida, referentes aos limites mínimo e máximo, respectivamente. Observa-se que, nesse caso, o limite mínimo refere-se à data de início desse teste (14/11/2010).

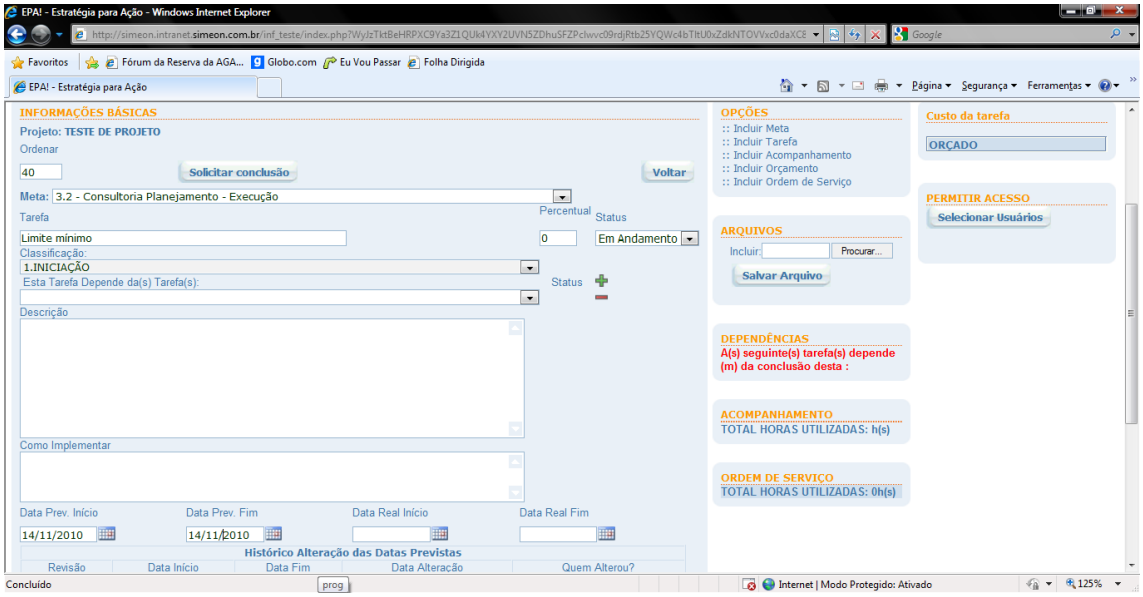


Figura A.21: *Limite Mínimo - CT19.*

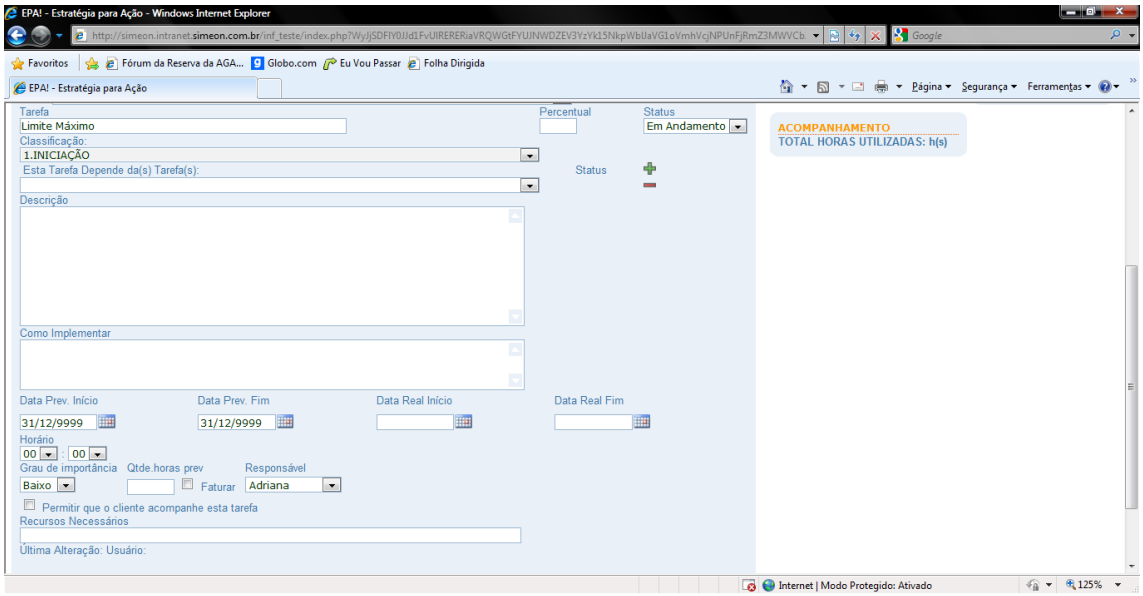


Figura A.22: *Limite Máximo - CT20.*

(j)Criar dois ou mais CT's com valores superiores ao limite mínimo - Classe Válida

Nas Figuras A.23 e A.24 são apresentados os dados de teste gerados a partir da execução da linha 18 do Algoritmo 4.1.

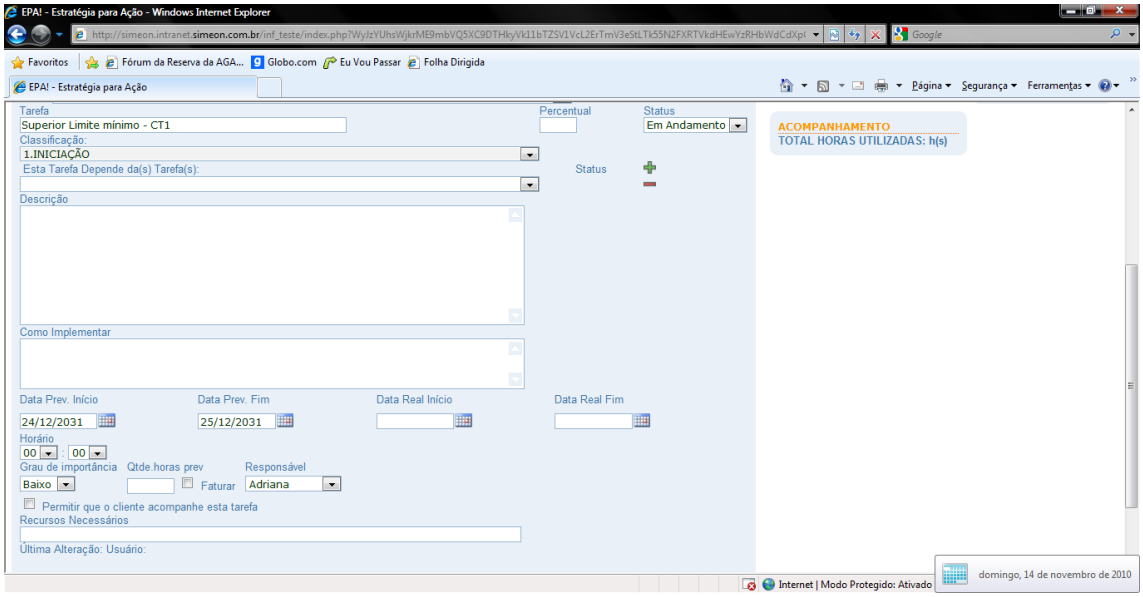


Figura A.23: *Datas superiores ao Limite Mínimo - CT21.*

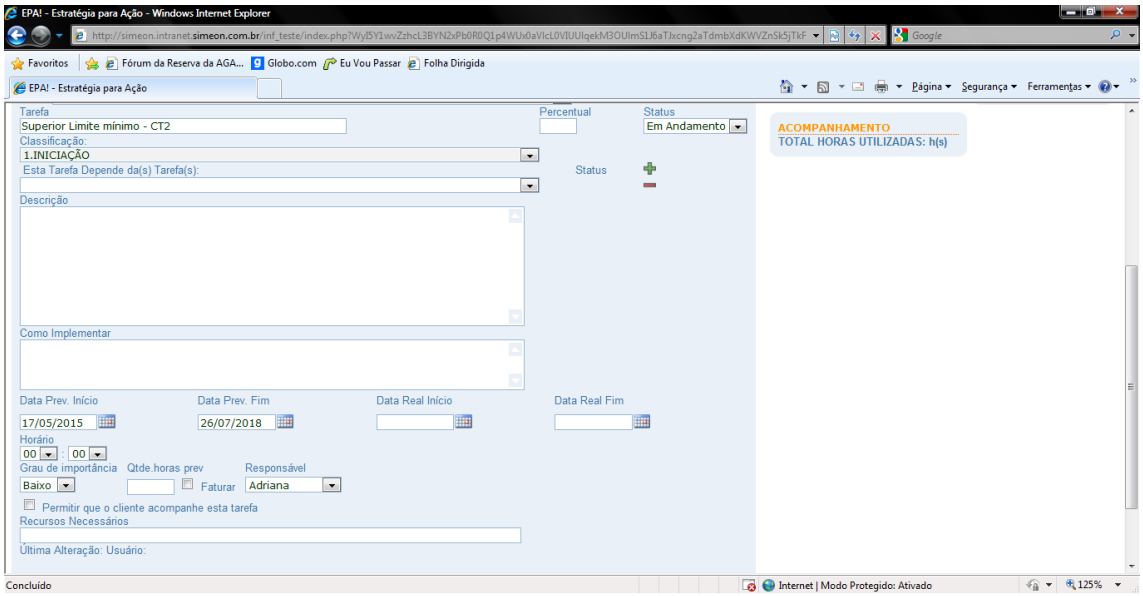


Figura A.24: *Datas superiores ao Limite Mínimo - CT22.*

(k)Criar dois ou mais CT's com valores inferiores ao limite máximo - Classe Válida

Nas Figuras A.25 e A.9 são apresentados os dados de teste gerados a partir da execução da linha 19 do Algoritmo 4.1.

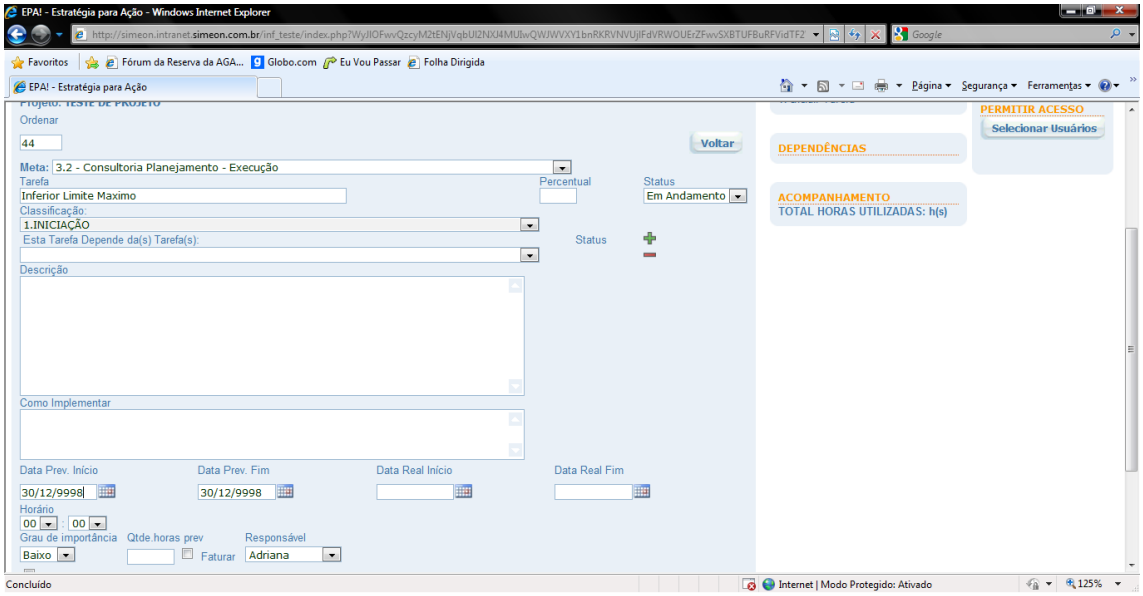


Figura A.25: *Datas inferiores ao Limite Máximo - CT23.*

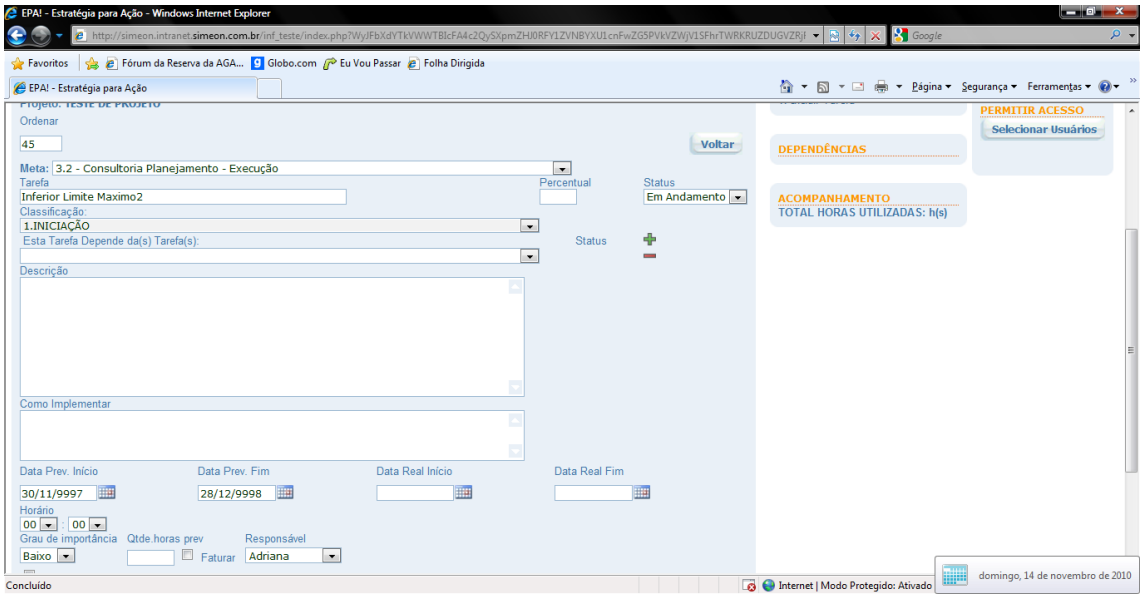


Figura A.26: *Datas inferiores ao Limite Máximo - CT24.*

(l)Criar dois ou mais CT's com valores intermediários ao intervalo - Classe Válida

Nas Figuras A.27 e A.28 são apresentados os dados de teste gerados a partir da execução da linha 20 do Algoritmo 4.1.

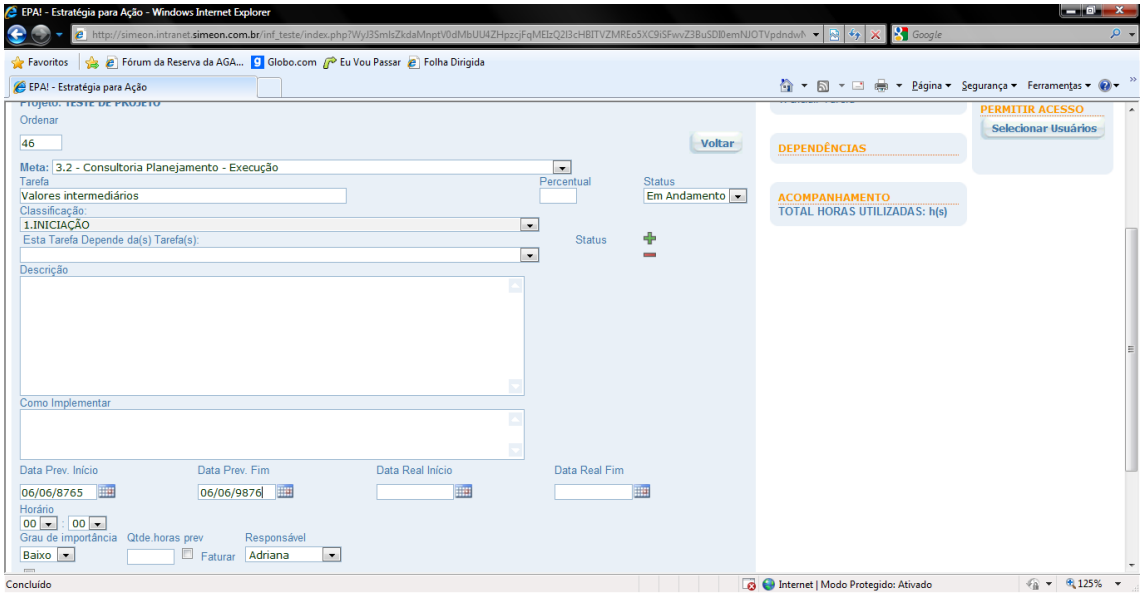


Figura A.27: *Datas Intermediárias - CT25.*

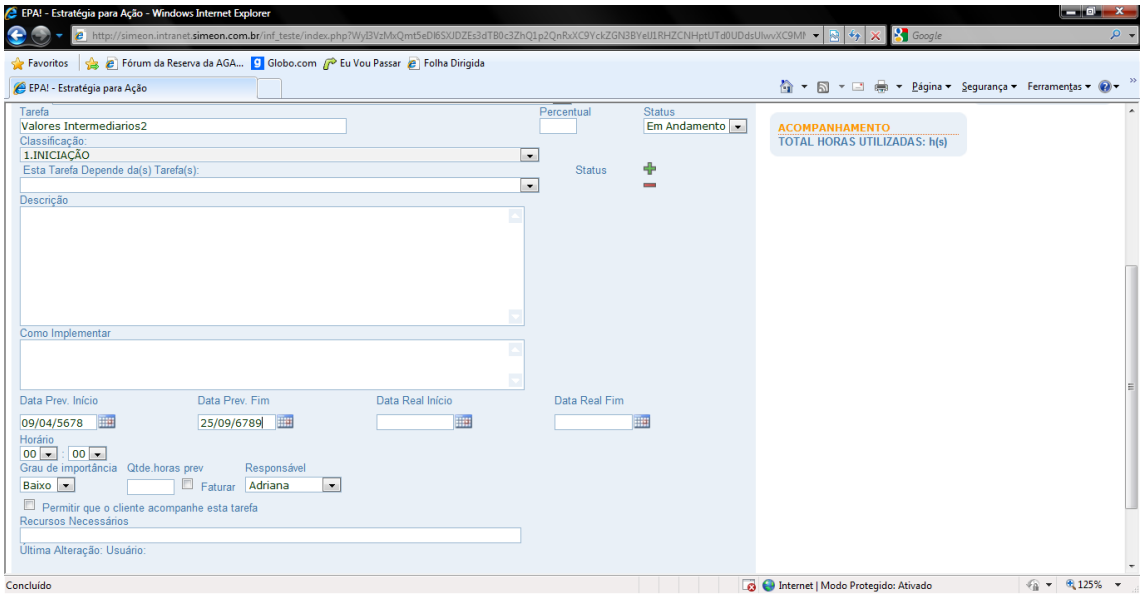


Figura A.28: *Datas Intermediárias - CT26.*

(m)Criar dois ou mais CT's com valores inferiores ao limite mínimo - Classe Inválida Na Figura A.29 é apresentado o primeiro dado de teste gerado a partir da execução da linha 21 do Algoritmo 4.1. Foram inseridos nos campos referentes à dia, mês e ano, o valor zero.

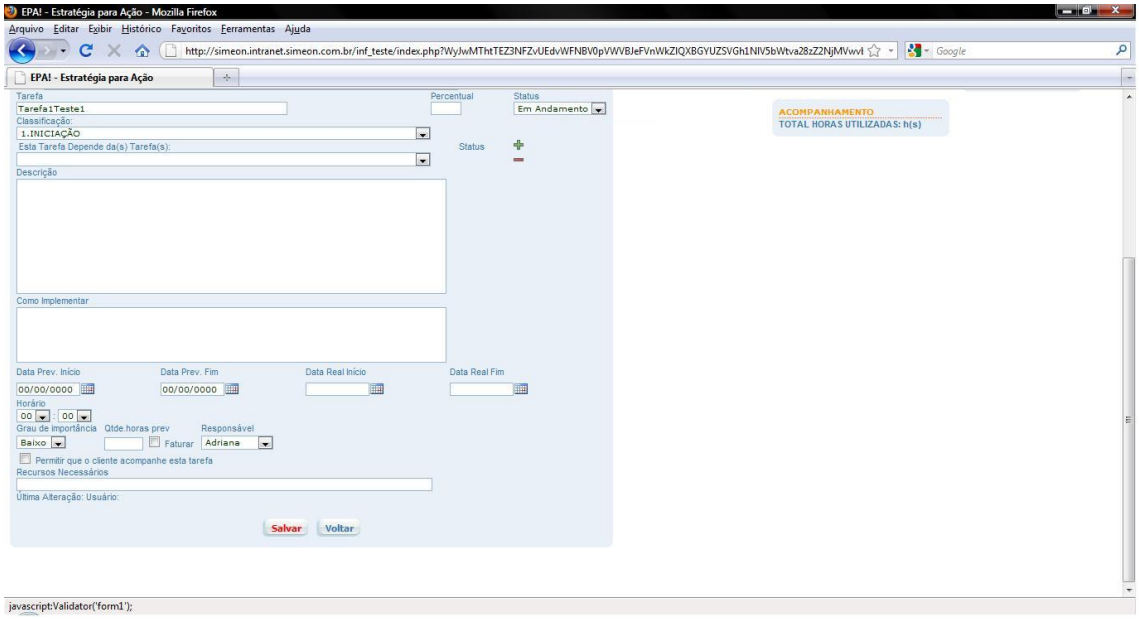


Figura A.29: *Data Inferior ao Limite Mínimo - CT27.*

Porém, observou-se que, conforme apresentado na Figura A.30, que após salvar o cadastro de tarefas, os campos são apresentados em branco e não há mensagem ao usuário de que os dados são inconsistentes.

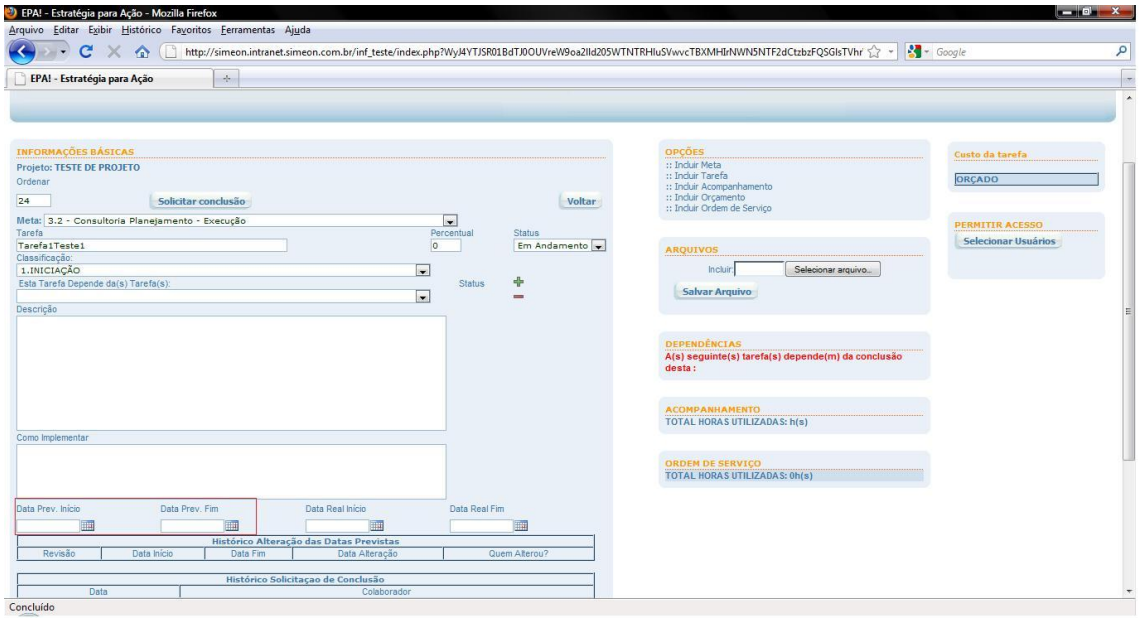


Figura A.30: *Erro gerado ao Inserir data inferior ao Limite Mínimo.*

Para a criação referente ao próximo dado de teste desse passo, tentou-se inserir valores negativos para os campos, porém, o sistema não permitiu. Assim, o CT28 corresponde à possibilidade de inserção de valores negativos nos campos.

(n)Criar dois ou mais CT's com valores superiores ao limite máximo - Classe Inválida

Na Figura A.29 é apresentado o primeiro dado de teste gerado a partir da execução da linha 22 do Algoritmo 4.1. Foram inseridos nos campos referentes a dia, mês e ano, os valores máximos aceitos para os campos.

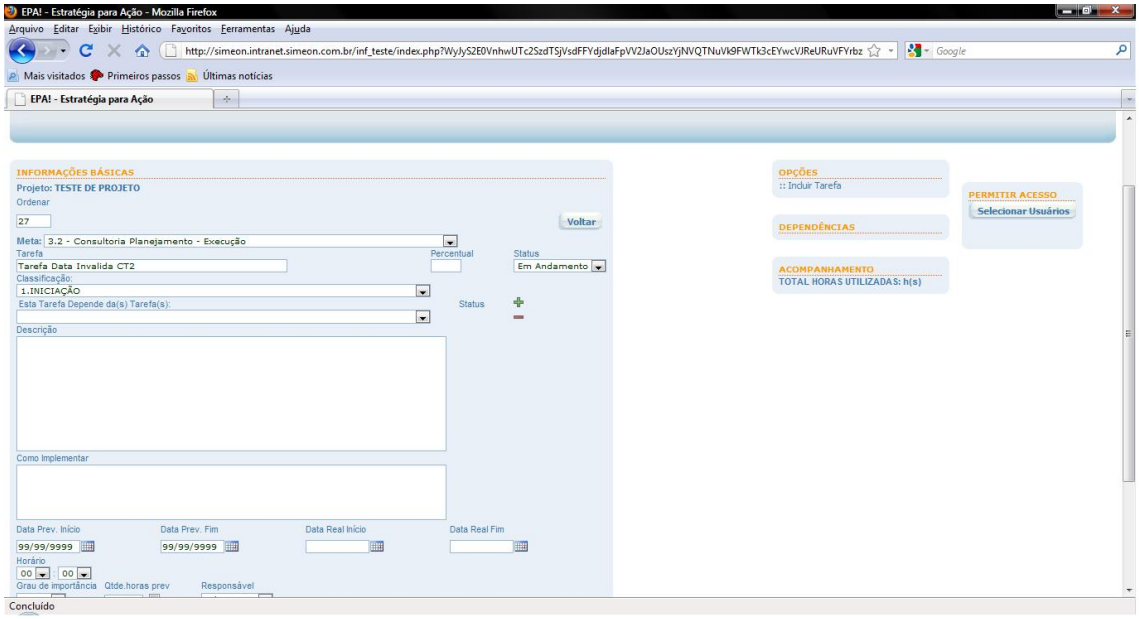


Figura A.31: *Data Superior ao Limite Máximo - CT29.*

A Figura A.32 representa a tela gerada após salvar o cadastro de tarefas. Observou-se que, não é emitida mensagem de inconsistência e os dados são apresentados em branco.

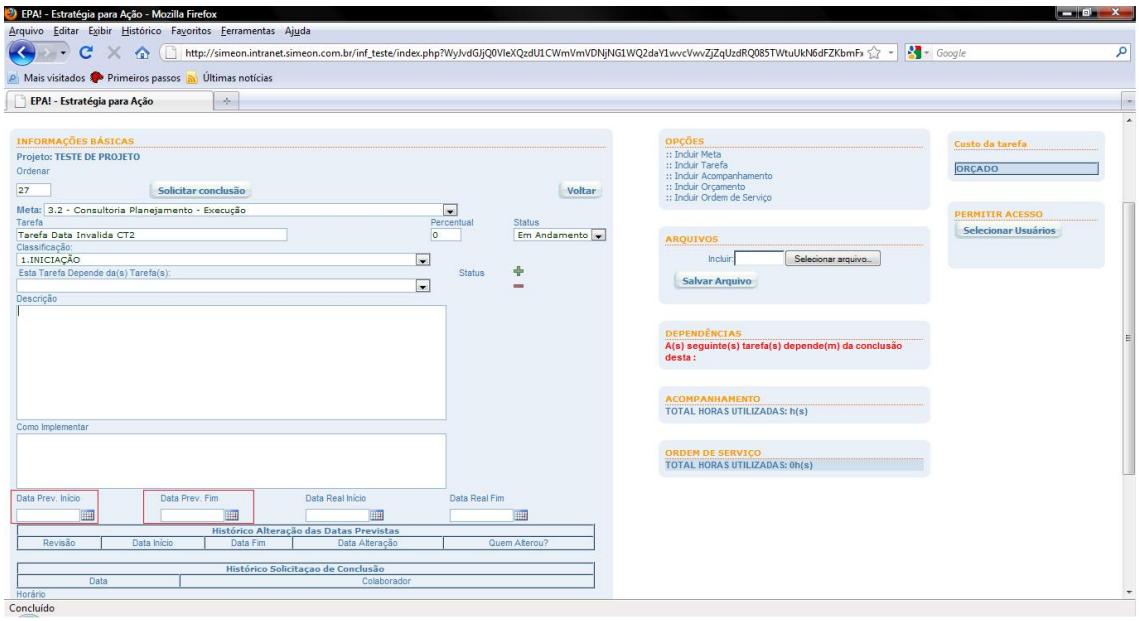


Figura A.32: *Erro gerado ao inserir Data Superior ao Limite Máximo.*

Em relação ao próximo dado de testes, tem-se que, outros dados de testes gerados anteriormente (Figuras A.19 e A.20, por exemplo) já contemplam esse caso.

3. Aplicando o Algoritmo 4.9 - Criar CT's que contemplem domínios de entrada e saída com tipos diferentes do especificado - Classe Inválida.
   Os CT's 30, 31, representados respectivamente pelas Figuras A.33 e A.34 e o CT 32 foram derivados de forma a contemplar o Algoritmo 4.9.

   (a) Campos com valores em branco.

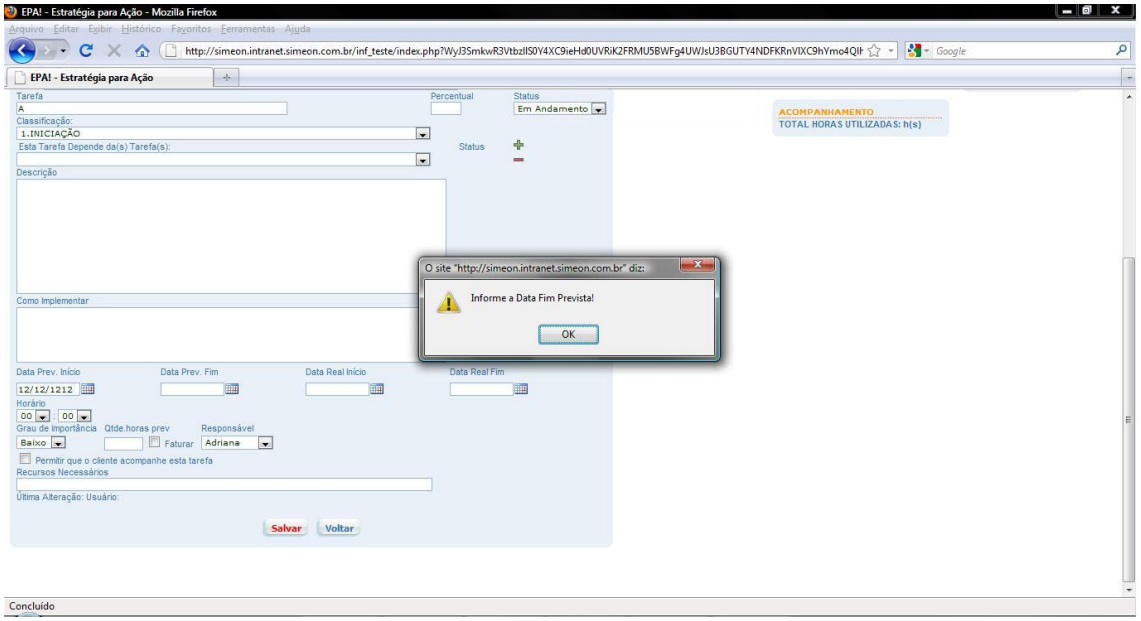


Figura A.33: *Campos vazios/Data Final - CT30.*

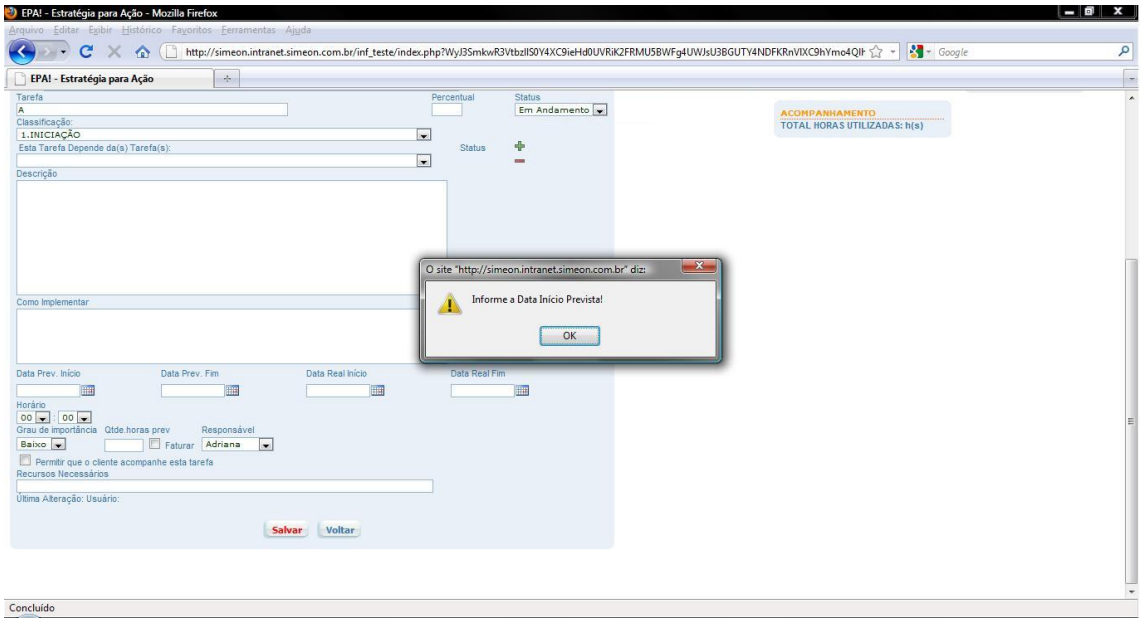


Figura A.34: *Campos vazios/Data Final - CT31.*

(b)Campos com caracteres.

O Sistema não permite fornecer caracteres para os campos - CT32.

4.Aplicando o Algoritmo 4.10 - Criar CT's que contemplem os casos especiais - domínio de entrada.

(a)Setembro de 1752 Um caso especial para o campo data seria contemplar datas do intervalo de 03/09/1752 a 13/09/1752, pois, segundo o calendário gregoriano, utilizado atualmente, estes dias não existem. A reforma no calendário Gregoriano (*The Gregorian Reformation*) ocorreu no dia 3 de setembro de 1752. Com essa reforma, dez dias foram eliminados do calendário, assim, no calendário, o dia 02/09/1752 é seguido do dia 14/09/1752 (STATMATH, 2010).

A Figura A.35 representa o dado de teste que contempla esse caso especial.

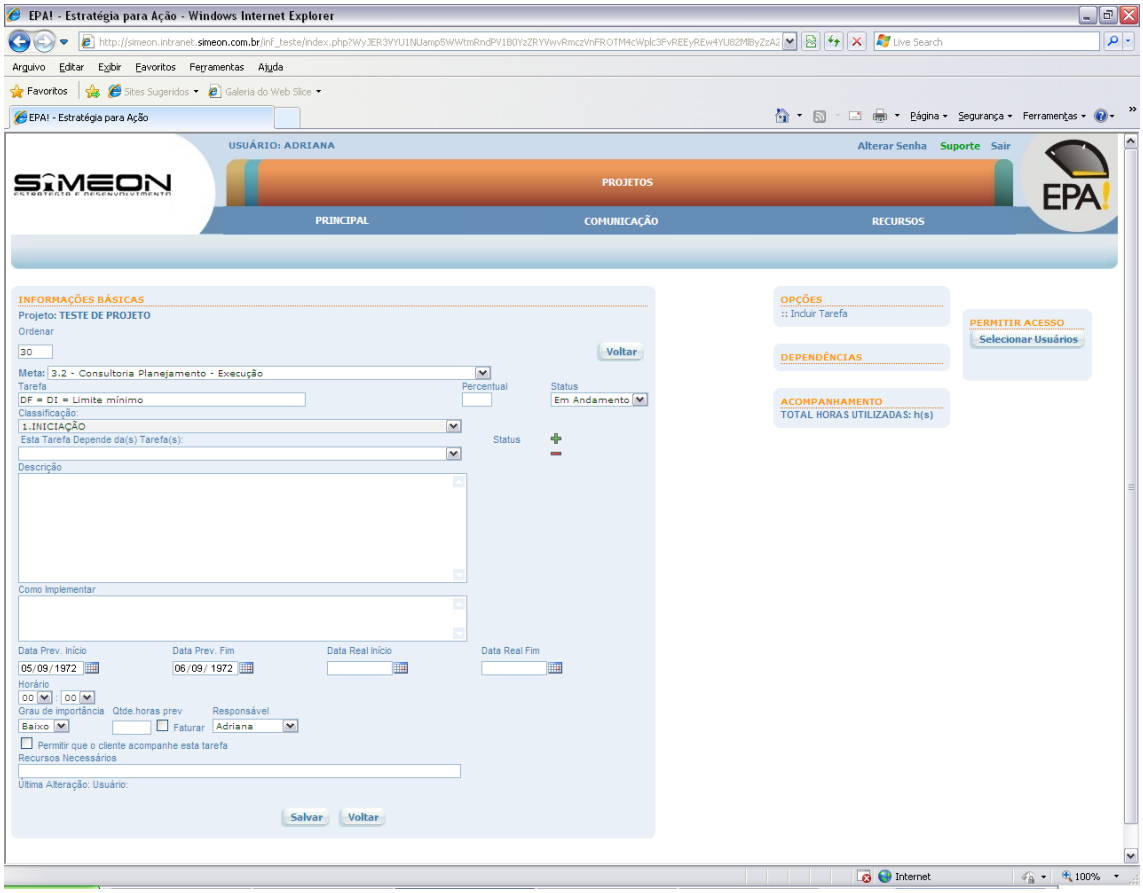


Figura A.35: *Data Especial - CT33.*

A Figura A.1 apresenta o resumo de casos de testes criados e o número de defeitos encontrados utilizando TFSE na criação e execução de casos de testes para a funcionalidade *Incluir Tarefas*.

Tabela A.1: *Número de casos de testes e defeitos encontrados - Incluir Tarefas.*

Casos de Testes Gerados	**Defeitos Encontrados**
33	10

APÊNDICE B

Instrumento de coleta de dados

B.1 Questões sobre o perfil pessoal

1.Sobre a identificação da equipe:

•Nome (campo opcional)

•A qual órgão técnico você pertence?

()Comunidade Evangélica Luterana São Paulo

()Escola Politécnica de Minas Gerais

()Fundação de Assistência e Educação

()Fundação Instituto Nacional de Telecomunicações

()Fundação Percival Farquhar

()Fundação Regional de Blumenau

()Fundação São Paulo – São Paulo

()Fundação Universidade Regional de Blumenau

()Fundação Universitária do Desenvolvimento do Oeste

()Fundação Visconde de Cairu

()Instituto de Tecnologia do Paraná

()Instituto de Pesquisas Tecnologicas do Estado de São Paulo

()Instituto Filadélfia de Londrina

()Pontifícia Universidade Católica do Rio Grande do Sul

()Sociedade Potiguar de Educação e Cultura

()Universidade Católica de Goiás

()Universidade Federal de Goiás

()Outro

•Qual o tamanho da equipe de avaliadores do órgão técnico que você pertence?

•Idade

() Menor do que 18

() Entre 18 e 24 anos

() Maior do que 24 anos

•Sexo

() Femino

() Masculino

2.Questões sobre o perfil profissional

•Formação

() Ciências da Computação

() Engenharia de Software

() Sistema de Informação

() Engenharia da Computação

() Engenharia Elétrica

() Outro:

•Cursando/Formado

()Cursando 1^o período

()Cursando 2^o período

()Cursando 3^o período

()Cursando 4^o período

()Cursando 5^o período

()Cursando 6^o período

()Cursando 7^o período

()Cursando 8^o período

()Formado

()Outro

•Tem experiência profissional em computação?

() Sim

() Não

•Se sim, quanto tempo?

() menos que 1 ano

() entre 1 e 5 anos

() entre 5 e 10 anos

() mais que 10 anos

•Tem conhecimento em Análise de Requisitos?

() Desconheço totalmente

() Desconheço parcialmente

() Indeciso

() Conheço parcialmente

() Conheço totalmente

•Tem conhecimento em Projeto de Software?

() Desconheço totalmente
() Desconheço parcialmente
() Indeciso
() Conheço parcialmente
() Conheço totalmente

•Já implementou algum programa de computador?
() Sim
() Não

•Se sim, em qual linguagem?
() Java
() C
() C#
() Rubi
() Outro:

3.Questões sobre o perfil relacionado a testes de Software

•Tem experiência profissional em testes?
() Sim
() Não

•Se sim, quanto tempo?
() menos que 1 ano
() entre 1 e 5 anos
() entre 5 e 10 anos
() mais que 10 anos

•Conhece alguma técnica de teste?
() Sim
() Não

•Se sim, qual?
() Técnica de Teste Funcional
() Técnica de Teste Estrutural
() Técnica de Teste baseada em Defeitos
() Testes Exploratórios
() Outro:

•Já utilizou alguma critério de teste?
() Sim
() Não

•Se sim, qual?
() Análise do Valor Limite
() Particionamento de Equivalência

() Tabela de Decisão
() Todos-Nós
() Todas-Arestas
() Todos-Caminhos
() Todas-Definições
() Todos-P-Uso
() Todos-C-Uso
() Todos-Uso
() Outro:

•Possui certificação em teste de software?
()Sim
()Não

•Se sim, qual?
() CBTS
() QAI(CSTE)
() IIST(CSTP)
() IIST(CTM)
() Outro:

•Atualmente, qual é seu perfil profissional?
() Analista de Teste
() Arquiteto de Teste
() Auditor de Qualidade de Software (SQA)
() Automatizador de Teste (Funcionais, Performance, etc)
() Gerente de Teste
() Líder de Teste
() Testador
() Outro:

•Gosta de atuar na área de testes?
() Sim
() Não

•Há quanto tempo participa do projeto PAF-ECF?
() Menos de um ano
() Entre 1 e 2 anos
() Entre 2 e 5 anos
() Mais de cinco anos

•Você acha importante conhecer os requisitos relacionados aos testes do roteiro?
() Sim

() Não

- Você considera o roteiro - em termos de entendimento e simplicidade - eficiente?
  () Sim
  () Não
- Você considera que o roteiro valida a funcionalidade PAF-ECF completamente?
  () Sim
  () Não
- Durante as certificações, os desenvolvedores demonstram conhecer o roteiro?
  () Sim
  () Não

APÊNDICE C

Respostas das Equipes Certificadoras

C.1 Dados Relacionados a Questões Gerais.

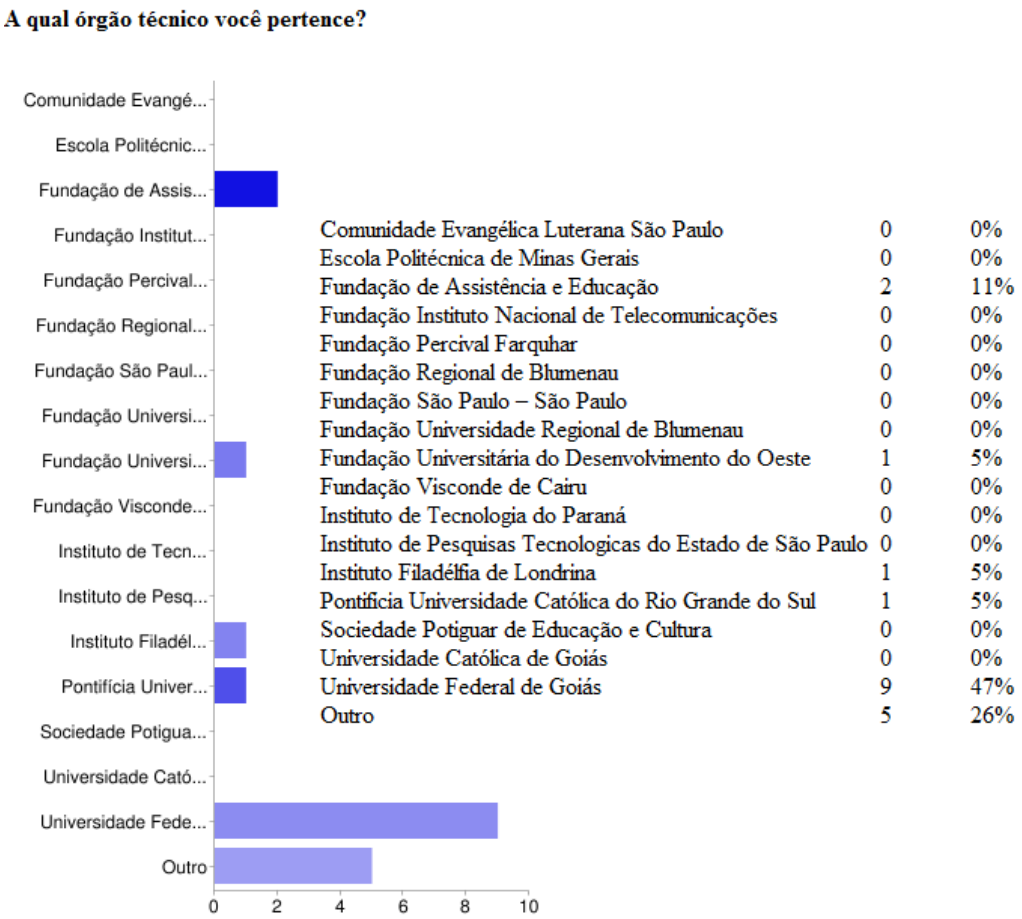


Figura C.1: *Dados obtidos referentes à pergunta: A qual órgão técnico você pertence?*

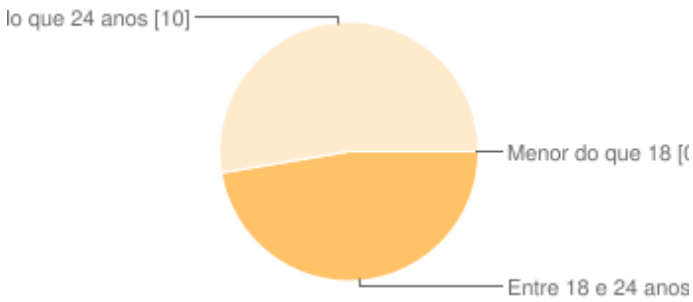


Figura C.2: *Dados obtidos referentes à pergunta: Qual a sua idade?*

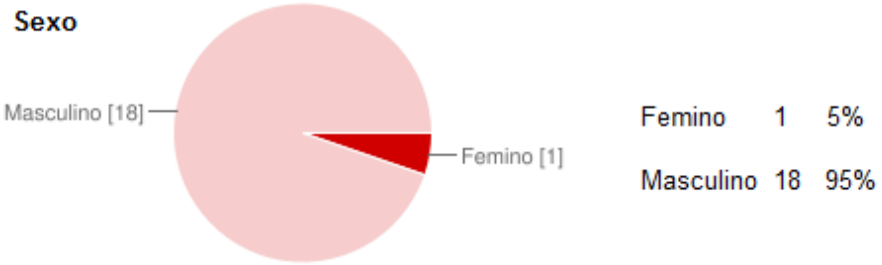


Figura C.3: *Dados obtidos referentes à pergunta: Qual o seu sexo?*

C.2 Dados Relacionados a Questões Profissionais/Formação.

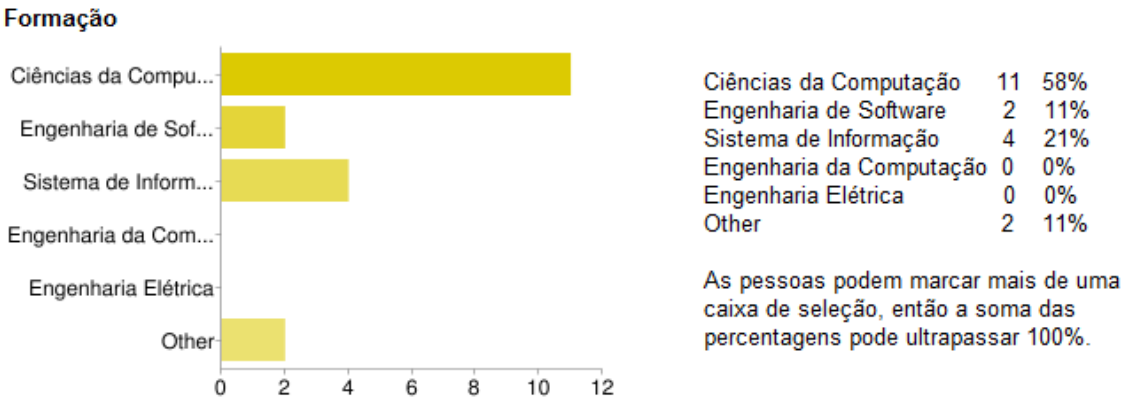


Figura C.4: *Dados obtidos referentes à pergunta: Qual a sua formação?*

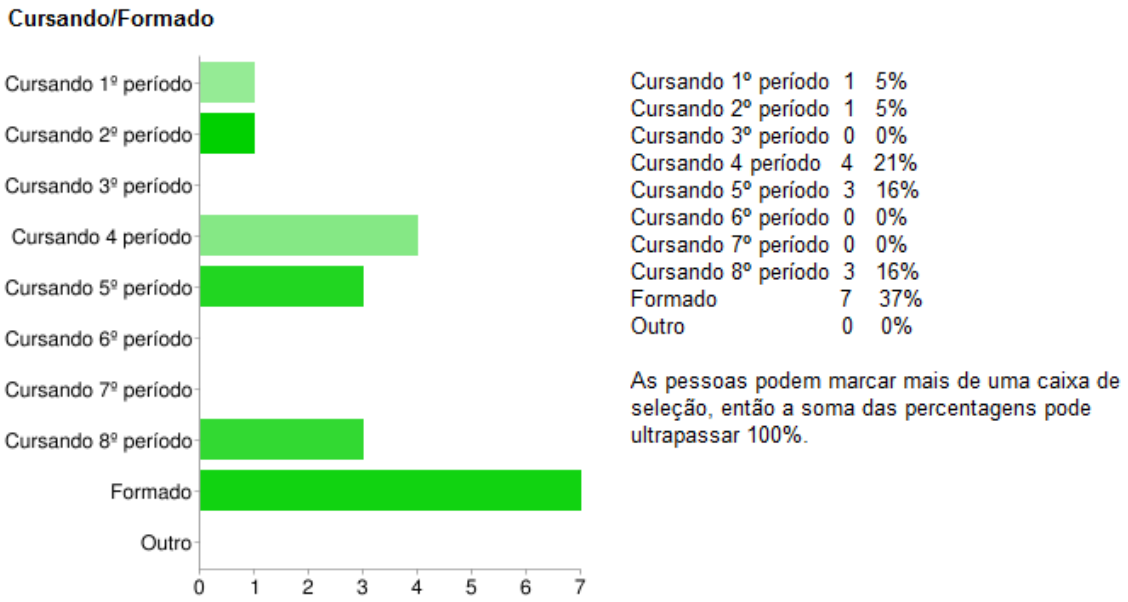


Figura C.5: *Dados obtidos referentes ao tempo de formação.*

Figura C.6: *Dados obtidos referentes à experiência profissional em computação.*

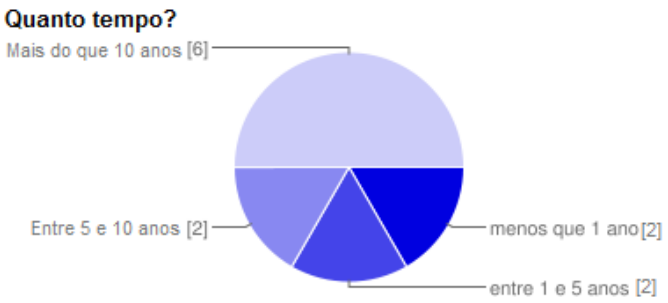


Figura C.7: *Dados obtidos relacionados ao tempo de experiência profissional.*

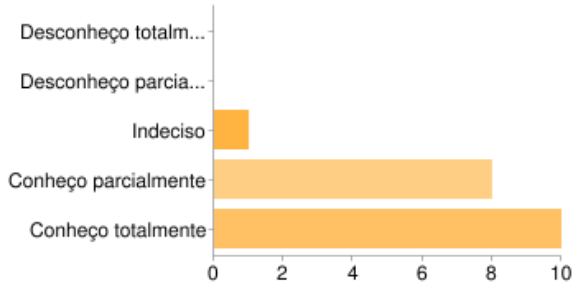


Figura C.8: *Dados obtidos concernentes ao conhecimento em requisitos de software.*

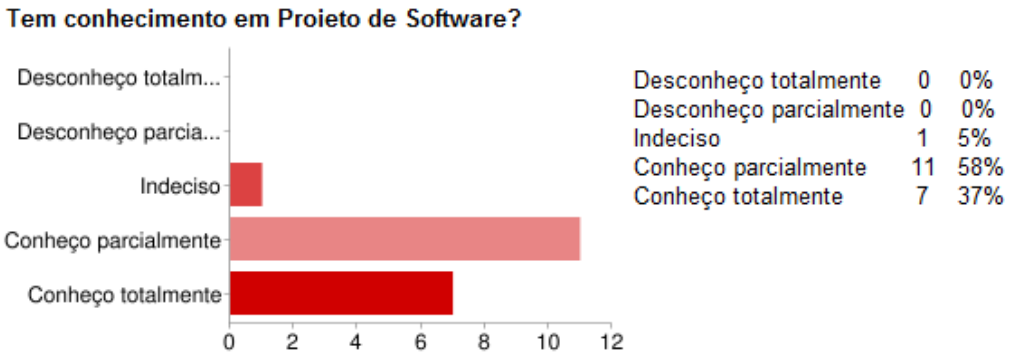


Figura C.9: *Dados obtidos concernentes ao conhecimento em projeto de software.*

Figura C.10: *Dados obtidos concernentes ao conhecimento em implementação de software.*

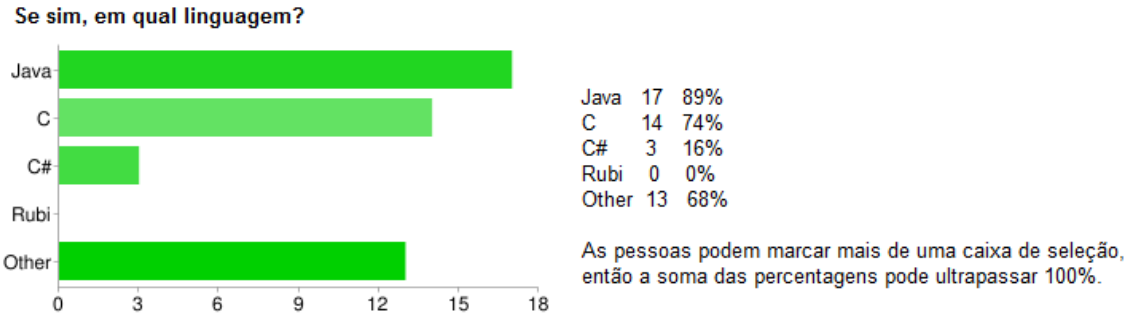


Figura C.11: *Dados obtidos concernentes ao conhecimento em linguagem de programação.*

C.3 Dados Relacionados a Questões referentes a Teste de Software.

Figura C.12: *Dados obtidos relacionados à experiência profissional em teste de software.*

Figura C.13: *Dados obtidos relacionados ao tempo de experiência profissional em teste de software.*

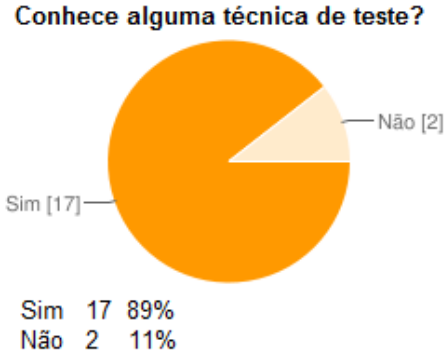
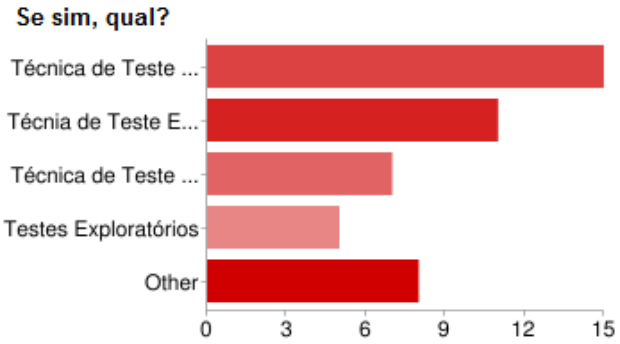


Figura C.14: *Dados obtidos relacionados ao conhecimento em técnicas de teste.*



Técnica de Teste Funcional	15	88%
Técnia de Teste Estrutural	11	65%
Técnica de Teste baseada em Defeitos	7	41%
Testes Exploratórios	5	29%
Other	8	47%

As pessoas podem marcar mais de uma caixa de seleção, então a soma das percentagens pode ultrapassar 100%.

Figura C.15: *Dados obtidos relacionados a pergunta sobre as técnicas de teste conhecidas.*

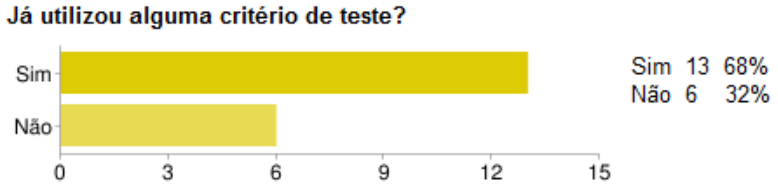


Figura C.16: *Dados obtidos relacionados ao conhecimento em Critério de teste de software.*

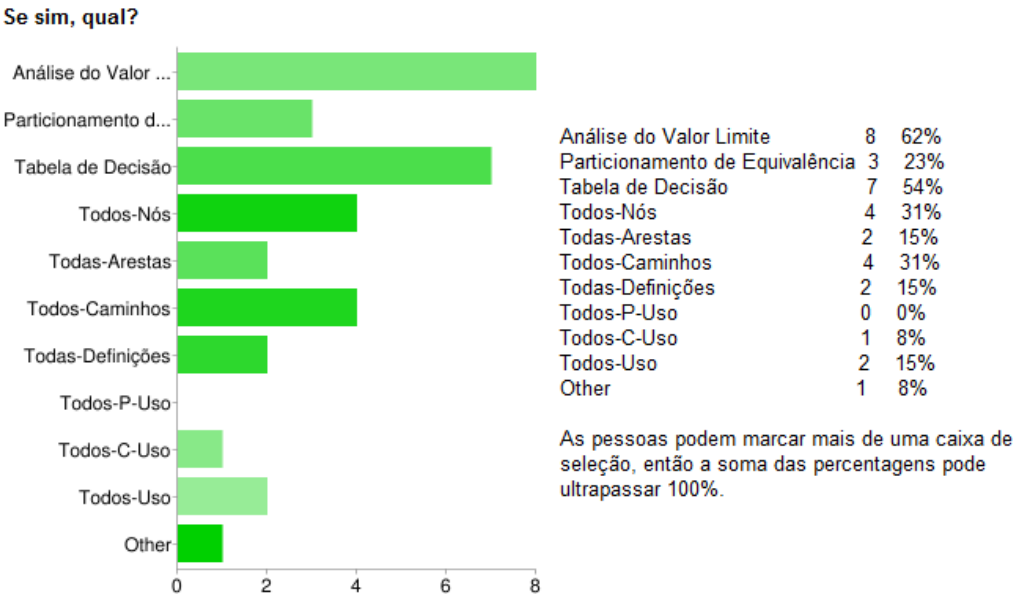


Figura C.17: *Dados obtidos relacionados a pergunta sobre os critérios de teste conhecidos.*

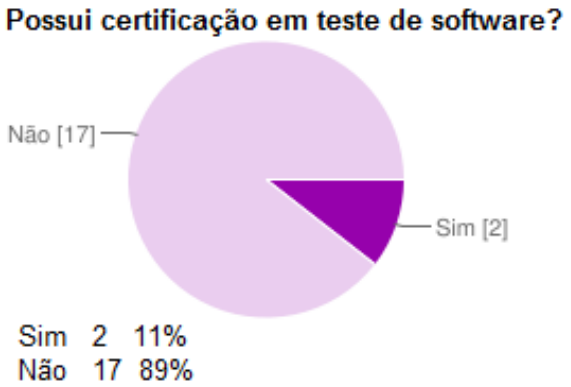


Figura C.18: *Dados obtidos relacionados à certificação em teste conhecidas.*

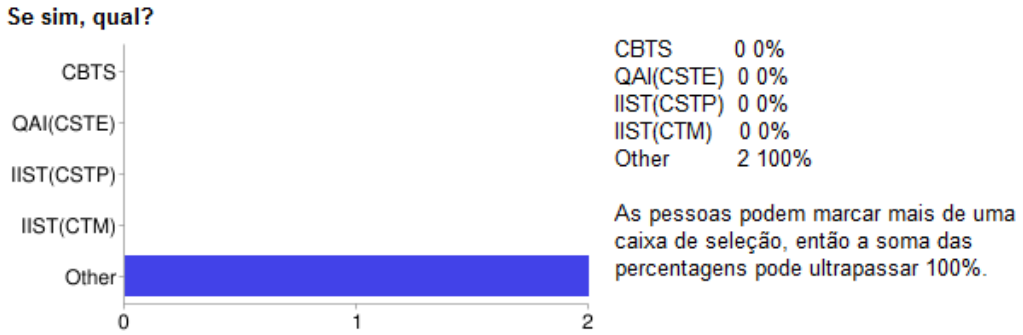


Figura C.19: *Dados obtidos relacionados a quais certificações em teste foram obtidas.*

C.4 Dados Relacionados a Questões Referentes à atuação no Órgão Credenciado.

Analista de Teste	1	5%
Arquiteto de Teste	0	0%
Auditor de Qualidade de Software (SQA)	0	0%
Automatizador de Teste (Funcionais, Performance, etc)	1	5%
Gerente de Teste	1	5%
Líder de Teste	1	5%
Testador	10	53%
Other	5	26%

Figura C.20: *Dados obtidos relacionados ao perfil profissional atual.*

Figura C.21: *Dados obtidos relacionados à aptidão na atividade de testes.*

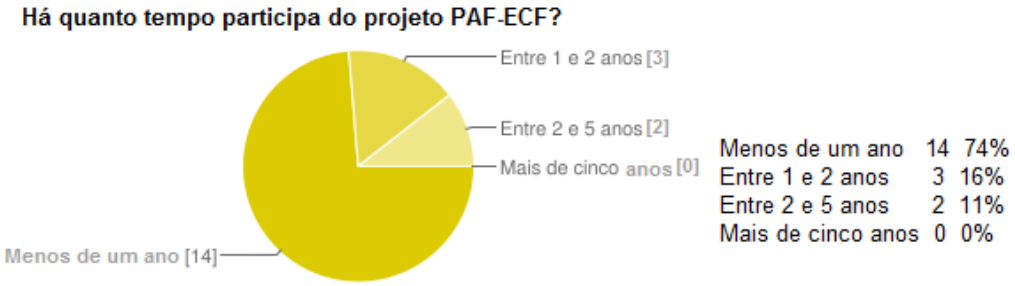


Figura C.22: *Dados obtidos relacionados ao tempo de atuação no órgão credenciado.*

Figura C.23: *Dados obtidos relacionados à importância de se conhecer os requisitos do PAF-ECF.*

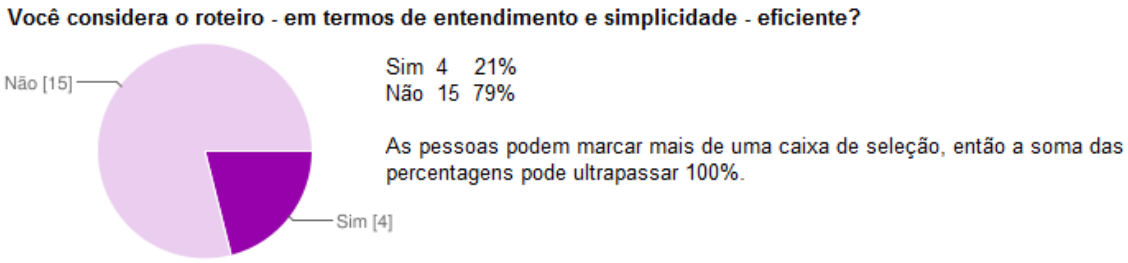


Figura C.24: *Dados obtidos relacionados à eficiência - em termos de simplicidade - do Roteiro.*

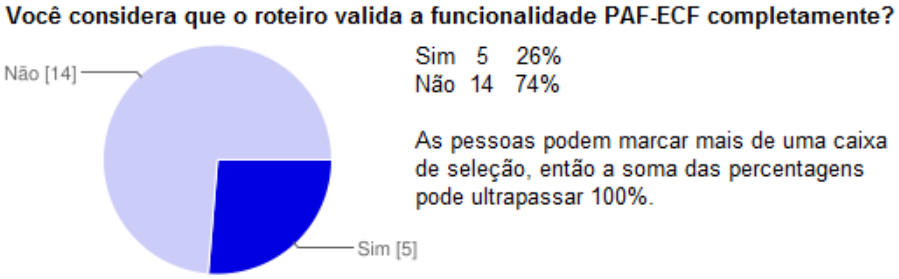


Figura C.25: *Dados obtidos relacionados à validação do Roteiro.*

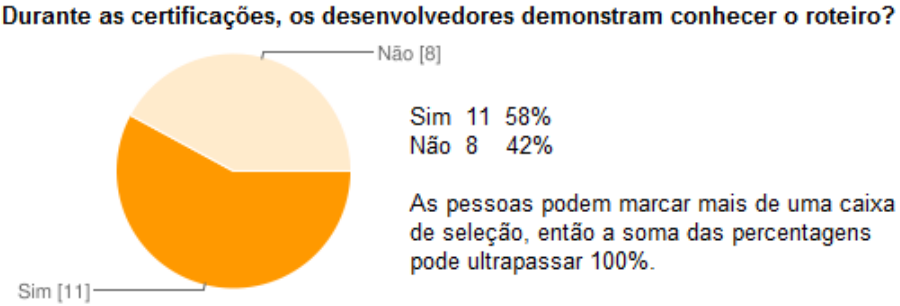


Figura C.26: *Dados obtidos relacionados ao conhecimento do Roteiro dos desenvolvedores que implementam PAF-ECF.*