UNIVERSIDADE DO SUL DE SANTA CATARINA

MÁRIO CARLOS DE BEM OSÓRIO NETO

PROPOSTA DE IMPLANTAÇÃO DE METODOLOGIA MÍNIMA DE TESTES FUNCIONAIS DE SOFTWARE EM UM PROCESSO DE DESENVOLVIMENTO PRESCRITIVO

Florianópolis
2015

MÁRIO CARLOS DE BEM OSÓRIO NETO

PROPOSTA DE IMPLANTAÇÃO DE METODOLOGIA MÍNIMA DE TESTES FUNCIONAIS DE SOFTWARE EM UM PROCESSO DE DESENVOLVIMENTO PRESCRITIVO

Monografia apresentada ao curso de especialização em Engenharia de Projetos de Software, da Universidade do Sul de Santa Catarina, como requisito parcial para a obtenção do título de Especialista em Engenharia de Projetos de Software.

Orientadora: Profª. Vera Rejane Niedersberg Schuhmacher, Dra.

Florianópolis

2015

MÁRIO CARLOS DE BEM OSÓRIO NETO

PROPOSTA DE IMPLANTAÇÃO DE METODOLOGIA MÍNIMA DE TESTES FUNCIONAIS DE SOFTWARE EM UM PROCESSO DE DESENVOLVIMENTO PRESCRITIVO

Esta monografia foi julgada adequada à obtenção do título de Especialista em Engenharia de Projetos de Software e aprovada em sua forma final pelo curso de Engenharia de Projetos de Software, da Universidade do Sul de Santa Catarina.

Florianópolis, 16 de março de 2015.

_____

Profª. e orientadora Vera Rejane Niedersberg Schuhmacher, Dra.

Universidade do Sul de Santa Catarina

_____

Profª. Maria Inés Castiñeira, Dra.

Universidade do Sul de Santa Catarina

RESUMO

O desafio de implantar um processo formal de testes e os benefícios por ele trazidos foram os responsáveis por incentivar o desenvolvimento deste trabalho. Na empresa pública utilizada como estudo de caso, os testes eram pensados apenas quando a empresa terceirizada contratada para o desenvolvimento entregava o código fonte e, obviamente, devido ao prazo apertado, eram postos em segundo plano. Como consequência, a quantidade de erros encontrados em produção era alta. Este trabalho apresenta uma proposta de metodologia de testes funcionais aplicada ao processo de desenvolvimento da empresa pública em estudo. Contando com um processo de desenvolvimento bem definido, com pouca documentação, mas seguido por todos os funcionários, o objetivo do trabalho foi desenvolver atividades relacionadas ao teste de software que se encaixassem de forma suave no processo atual, de forma que o planejamento dos testes iniciasse já no começo do processo de desenvolvimento e houvesse rastreabilidade entre requisitos e casos de teste. Foi dado ênfase no trabalho de planejamento e modelagem dos testes de forma a tornar a execução mais simples, porém, com a preocupação de implementar métricas para avaliar a melhoria proposta e garantir o correto registro dos erros encontrados.

Palavras-chave: Teste de software. Teste funcional. Planejamento do teste. Processo de desenvolvimento de software.

ABSTRACT

The challenge of implementing a formal testing process and the benefits it brings about were responsible for encouraging the development of this work. The government-owned corporation used as a case study of this research, outsources software coding but is responsible for the testing. However, the software testing is considered only when the code is delivered back, usually already late, and the tests are usually neglected. As a result, the production defects are high. This work proposes a functional testing methodology applied to the existing software development process, which is well-known by the teams, but has little documentation. The goals of the research was to present activities related to software testing and apply them smoothly in the development process, in order to emphasize the test planning in the beginning of the software lifecycle and assure the traceability between requirements and test cases. The test process tries to highlight the importance of test case modeling over execution as well as the use of metrics to evaluate the results and also a correct error report activity.

Keywords: Software testing. Functional testing. Test planning. Software development process.

LISTA DE SIGLAS

CSJT – Conselho Superior da Justiça do Trabalho

EA – *Enterprise Architect*

ISTQB – *International Software Testing Qualifications Board*

NIST – *National Institute of Standards and Technology*

PTG – Plano de Testes Global

RUP – *Rational Unified Process*

SVN – Apache Subversion

TI – Tecnologia da Informação

TRT-SC - Tribunal Regional do Trabalho da 12ª Região - Santa Catarina

UML – *Unified Modeling Language*

V&V – Validação e Verificação

XP – Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

A disciplina de teste, no âmbito do desenvolvimento de software, há muito vem sendo discutida e tornou-se bastante importante no momento atual considerando um aumento da audiência e a diversidade de perfis que utilizam os softwares, incluindo pessoas com pouco ou nenhum conhecimento em informática. Sejam sistemas de uso corporativo, comerciais, ou aplicativos para celular e outros bens de consumo (eletrodomésticos, carros, etc.), o software já faz parte da vida de todos e passou de um produto de nicho para um bem de consumo geral. Em vista disso, o teste de software cresceu em importância à medida que a qualidade afeta diretamente a satisfação do usuário e também o retorno financeiro do investimento realizado para o desenvolvimento dos sistemas.

Essa entrega de software com qualidade apresenta-se como um dos grandes desafios da indústria de tecnologia da informação (TI), concomitantemente com os limites de prazo e orçamento. A empresa de pesquisa em TI, The Standish Group, publicou no CHAOS Manifest de 2013 (THE STANDISH GROUP, 2014) que ainda 18% dos projetos de TI falham e 43% dos que são concluídos ficam além do custo ou prazo estimados ou ainda são entregues incompletos. Uma peça chave para que um projeto de software tenha sucesso é o teste de software, garantindo tanto que o sistema faz aquilo que o cliente solicitou, como também que o sistema não faz o que não deveria fazer, contribuindo sempre para a garantia da qualidade do produto. A implantação de um processo de teste tornou-se, portanto, um problema a ser resolvido, não mais como uma atividade isolada das equipes de qualidade, mas envolvendo o processo de desenvolvimento de software como um todo e que tem reflexo na entrega de valor aos clientes e usuários, atingindo também diretamente a satisfação dos colaboradores da empresa, visto que um software entregue com qualidade trará menos transtorno à equipe na pós-implantação.

O amadurecimento dos processos de desenvolvimento de software trouxe como básicas as atividades de validação e verificação, mas, assim mesmo, a implantação de tais atividades nas empresas não ocorre de forma trivial, dependendo em parte de fatores econômicos, como a falta de visibilidade dos gestores da economia que o teste pode trazer à empresa, e em parte de fatores psicológicos da própria equipe técnica envolvida nas atividades de programação e teste de software. Uma pesquisa do portal de TI voltado a negócios, Information

Week, mostrou em 2002 que 97% dos gestores de TI entrevistados enfrentaram problemas devido a falhas nos softwares e 90% ainda informaram que suas empresas tiveram perdas financeiras decorrentes desses erros (INFORMATION WEEK, 2014). Muitas dessas perdas traduzir-se-ão em ações contra a empresa responsável pelo desenvolvimento do software, no que ela acaba por coparticipar no prejuízo dos clientes. Em outro trabalho, também de 2002, o instituto de padrões e tecnologia dos Estados Unidos (NIST, na sigla em inglês) estimou que o custo de correção dos erros em software, nos Estados Unidos, chega a U$ 60 bilhões por ano. O mesmo estudo ainda afirma que a correção antecipada dos erros poderia economizar cerca de U$ 22 bilhões/ano (NIST, 2002).

Mesmo com números tão assustadores, dez anos depois da publicação dos dois estudos, o panorama mostrado por pesquisas como (LEE, 2012), (GAROUSI, 2013) e (STRATE, 2013) mantém verdadeira a premissa da dificuldade na adoção de um processo de testes efetivo, o que torna esse trabalho ainda desafiador. O real montante a ser investido em teste de sistemas dependerá, entre outros motivos, se o mau funcionamento pode trazer prejuízos, como perdas financeiras aos usuários ou à própria empresa detentora do software, impactos ambientais e risco à vida dos envolvidos.

1.1 OBJETO DA PESQUISA

O presente trabalho apresentará um estudo sobre a implantação de um projeto para formalização das atividades de verificação e validação no departamento de desenvolvimento de sistemas do Tribunal Regional do Trabalho de Santa Catarina (TRT-SC). A equipe de desenvolvimento tem como responsabilidade produzir sistemas de qualidade para auxiliar estrategicamente as diversas áreas da empresa, sejam sistemas corporativos e/ou de uso interno e também sistemas utilizados pelo público externo.

A necessidade dessa pesquisa tornou-se real na medida em que surgiram problemas decorrentes de entregas de softwares com baixa qualidade e que não atendiam as necessidades dos usuários. Não havia um processo de teste de software e a atividade de teste era feita informalmente, quando realizada.

Existem hoje no órgão diferentes tecnologias empregadas no desenvolvimento (Oracle Forms/Report, Visual Basic, ASP e Java EE), bem como

diferentes processos (desenvolvimento interno, desenvolvimento terceirizado e sustentação de sistemas nacionais desenvolvidos por outros tribunais e nacionalizados pelo Conselho Superior da Justiça do Trabalho). Este trabalho tratará especificamente da implantação do processo de testes pela equipe responsável pelo desenvolvimento de sistemas Java para web, na qual a engenharia de requisitos é realizada pelos analistas da empresa, porém a construção do software é totalmente terceirizada. O processo de desenvolvimento em questão, não é institucionalizado, porém é repetitivo e intuitivo, seguido por todos os funcionários, mas com pouca documentação.

O trabalho não aborda a automação de testes, difícil hoje devido ao fato de a empresa contar com poucos desenvolvedores.

1.2 OBJETIVOS

Serão apresentados o objetivo geral e os objetivos específicos do trabalho.

1.2.1 Objetivo geral

O objetivo geral da pesquisa é apresentar uma proposta de implantação das atividades de teste no departamento de desenvolvimento de sistemas do TRT-SC, com base no processo de desenvolvimento atual.

1.2.2 Objetivos específicos

Os objetivos específicos do projeto são:

a) Estudar as técnicas de gerenciamento e planejamento de teste;

b) Estudar as técnicas de escrita e modelagem de teste;

c) Documentar as macroatividades do processo de desenvolvimento da empresa;

d) Demonstrar as alterações necessárias para que as atividades de teste pudessem ser adotadas no processo atual da empresa;

e) Documentar o processo de desenvolvimento após a proposta de melhoria para melhor divulgação entre os membros da equipe.

1.3 JUSTIFICATIVA

Considerando que o desenvolvimento de sistemas na Justiça do Trabalho é hoje custeado com dinheiro do governo, torna-se necessário a evolução das práticas de engenharia de software para alcançar o princípio da economicidade na administração pública, conforme exposto na Constituição Federal (BRASIL, 1988). Conforme já relatado na introdução, há estudos comprovando que investimentos em teste de software resultam em economia de recursos.

Há ainda, o fato de que a informatização no serviço público, em especial na área jurídica, aumentou a transparência para a população e para os advogados no andamento processual, e também incrementou a economia do ponto de vista ambiental, reduzindo o consumo de recursos naturais. Soma-se a isso a crescente participação do TRT-SC na construção de sistemas nacionais, reduzindo assim o custo do retrabalho, quando outros tribunais adotam as soluções já prontas e validadas pelo tribunal catarinense. Essas ações, porém, somente tornam-se eficientes se os sistemas desenvolvidos apresentarem qualidade alta, atendendo às necessidades dos usuários e agregando valor à atividade fim dos tribunais.

1.4 ESTRUTURA DA MONOGRAFIA

O trabalho está organizado da seguinte forma: o capítulo dois apresenta a fundamentação teórica do processo de desenvolvimento de software.

No capítulo três, é abordada a teoria do teste de software, incluindo técnicas de modelagem e o gerenciamento do processo de teste.

No capítulo seguinte, é apresentado o processo de desenvolvimento da empresa antes da inclusão das atividades de teste.

O capítulo cinco demonstra como foi realizada a implantação das atividades de teste e as alterações no processo de desenvolvimento.

Finalmente, no capítulo seis são apresentadas as conclusões do trabalho.

2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Softwares desenvolvidos antigamente careciam de qualidade, não eram confiáveis e quase sempre superavam as previsões de custo e prazo. Para o mundo corporativo, tais variáveis não poderiam ser ignoradas e o desenvolvimento de software informal não atendia mais. Nesse contexto, foram criadas novas técnicas para o controle da complexidade dos sistemas, o que culminou com a definição atual de engenharia de software, cujo objetivo era, e continua sendo, desenvolver software com alta qualidade dentro de custos e prazo adequados.

Pressman (2006) nos dá uma visão da engenharia de software em camadas (figura 1). Essas camadas formam um arcabouço abrangendo processo, ferramentas e métodos. Segundo o autor, a camada de processo é o alicerce da engenharia de software que mantém unidas as camadas de tecnologia e permite o desenvolvimento racional e oportuno. O processo é usado de forma a tornar efetiva a tecnologia presente na engenharia de software.

Figura 1 - Engenharia de software em camadas.

Fonte: (PRESSMAN, 2006, p. 17).

Segundo Sommerville (2007), processo de software é um conjunto de atividades que leva à produção de um produto de software. É um roteiro usado como base para aplicação dos métodos, geração dos artefatos de trabalho, gestão eficiente e também a garantia da qualidade.

É importante adotar um processo para que a atividade de desenvolvimento de software não se torne caótica. Para alcançar os objetivos da engenharia de software (alta qualidade dentro de custo e prazo adequado), é

recomendado um mínimo de organização e controle para o desenvolvimento fluir de forma estável (PRESSMAN, 2006, p. 16).

No entanto, não existe um processo ideal. A adoção plena dependerá da empresa, das pessoas da equipe e também do tipo de software que está sendo desenvolvido**.**

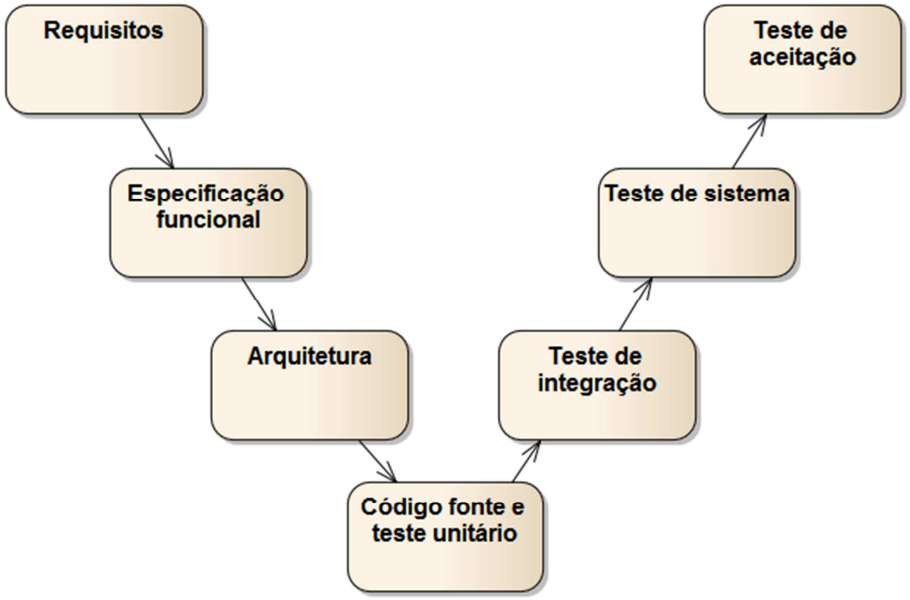
2.1 MODELOS DE DESENVOLVIMENTO DE SOFTWARE

Existem diversos modelos de desenvolvimento de software. Esta seção aborda dois modelos genéricos a fim de apresentar, no final, o modelo seguido pela empresa estudada.

2.1.1 Modelo V

Um modelo idealizado de desenvolvimento de software, que explicita as atividades de teste de software é o modelo V, comumente desenhado com quatro níveis, conforme figura 2 a seguir. As macroatividades de desenvolvimentos são sequenciadas e a cada uma é atribuída também uma macroatividade responsável pela garantia da qualidade do produto. Essas, por sua vez, representam os níveis de teste que serão tratados mais adiante neste trabalho (ROOK, 1986).

Figura 2 - Modelo V.



Fonte: Adaptado de (ROOK, 1986, p. 11).

O modelo V surgiu como uma visão diferente para a criação e execução dos testes. Em cada fase do ciclo de vida do desenvolvimento, os testes são realizados levando-se em consideração os níveis de testes. Assim, os testes podem começar desde o início do desenvolvimento, iniciando pelas revisões dos requisitos, inspeções/revisões de código, alcançando até os testes do produto.

Apesar disso, o ciclo de vida inerentemente sequencial desse modelo torna-o falho para a maioria dos projetos de software, que sofrem com requisitos que mudam frequentemente.

2.1.2 Modelos iterativos e incrementais

Modelos iterativos e incrementais buscam mitigar o problema de requisitos incompletos e aumentar o *feedback* ao usuário/cliente de forma a melhorar a comunicação entre as equipes. O desenvolvimento inicia com a especificação e construção de parte do sistema, até um marco definido pela equipe do projeto. O teste pode ser realizado nos vários níveis descritos no modelo V, porém distribuídos em cada iteração. O produto resultante é então revisado para identificar a necessidade de novos requisitos. Idealmente, cada iteração deve resultar em um software que implemente uma funcionalidade completa e pode ou não (de acordo com o acordado com o cliente) ser implantado em produção (SOMMERVILLE, 2007; PRESSMAN, 2006).

Exemplos de modelos iterativos e incrementais utilizados são: prototipagem, Rapid Application Development (RAD), Rational Unified Process (RUP) e modelos ágeis (PRESSMAN, 2006).

O modelo RUP será descrito em mais detalhes adiante por ser a base do modelo utilizado para o processo de desenvolvimento do TRT-SC.

2.1.2.1 Rational Unified Process

O RUP é definido como um processo para guiar as equipes de projeto de software e também como um conjunto de melhores práticas comprovadas da engenharia de software orientada a objetos usando a linguagem unificada de modelagem (UML, na sigla em inglês). O objetivo do RUP é, através da aplicação de

uma abordagem disciplinada para a atribuição de tarefas e responsabilidades, a produção de software de alta qualidade que atenda as necessidades dos clientes, dentro de cronograma e orçamento previsíveis. Além disso, o RUP provê um framework de processo a fim de ser adaptado e estendido para se encaixar às necessidades da empresa que decidir adotá-lo. O RUP utiliza a UML para documentar artefatos e, por ser flexível e configurável, pode ser utilizado em projetos de pequeno, médio e grande porte (KRUCHTEN, 2003).

O RUP, como produto, é mantido pela IBM Rational, a qual se encarrega de continuamente atualizá-lo para garantir que o processo reflita experiências recentes e a evolução das melhores práticas comprovadas.

Partindo de uma visão gerencial, a dinâmica do processo unificado é dividida em quatro fases: concepção, elaboração, construção e transição. A seguir uma descrição breve das tarefas mais comuns envolvidas em cada fase (KRUCHTEN, 2003):

a) Fase de Concepção: definição do escopo do projeto, identificação de requisitos de negócio, rascunho arquitetural do sistema, comunicação com as pessoas envolvidas (stakeholders); ao final, deve entregar uma definição inicial de custos e cronograma;

b) Fase de Elaboração: planejamento das atividades e recursos necessários para o projeto, avaliação de riscos, definição de cronograma, expansão dos casos de uso e da arquitetura do sistema; além disso, o projeto do sistema deve estar o mais completo possível para o início da fase de construção;

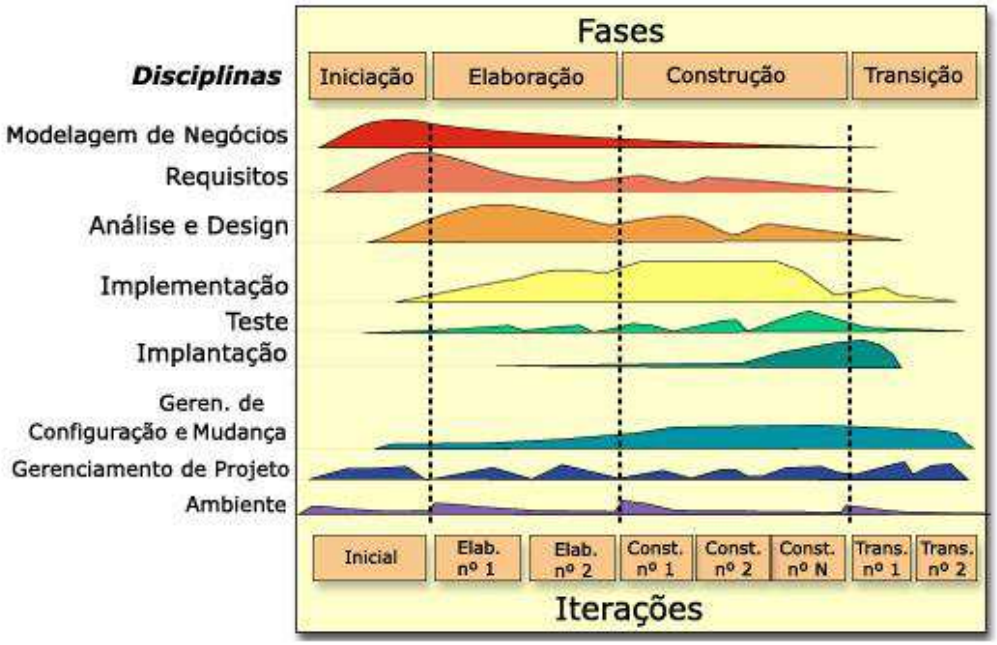
c) Fase de Construção: evolução da arquitetura, programação e testes do sistema; ao final, deve entregar um produto operacional;

d) Fase de Transição: representa a transição do produto para os usuários finais, incluindo implantação, treinamento, suporte e manutenção; o aceite principal desta fase deve sempre ser a satisfação do cliente (mesmo que através de documentos formais marcando o encerramento do projeto).

De forma ortogonal às quatro fases, são definidas nove disciplinas que representam as atividades que ocorrem durante o processo de desenvolvimento, conforme ilustra a figura 3. É perceptível, pela imagem, que o foco de cada fase está em determinadas atividades; porém, isso não impede que parte de uma atividade seja aplicada em uma fase qualquer (posterior ou anterior).

A figura 3 também inclui as nove disciplinas do RUP. São elas que descrevem as atividades realizadas em cada fase do processo. Essas nove disciplinas são divididas em seis técnicas e três de apoio.

Figura 3 - Disciplinas do RUP.



Fonte: KRUCHTEN, 2003.

As seis disciplinas técnicas e as atividades principais vinculadas a elas são:

a) Modelagem do negócio: levantamento dos processos de negócio para entender o funcionamento da organização e unificar o entendimento entre clientes, usuários e desenvolvedores;

b) Requisitos: identificação de atores, requisitos e casos de uso para definir com stakeholders o que o sistema deve fazer;

c) Análise e projeto: criação de um modelo de projeto e arquitetura do sistema;

d) Implementação: construção do software e testes unitários;

e) Teste: garantia da qualidade, validação dos objetivos e dos requisitos;

f) Implantação: beta test, distribuição, instalação, treinamento.

As três disciplinas restantes são chamadas de disciplinas de apoio:

a) Gerenciamento de projetos: planejar o projeto e as iterações, gerenciar riscos e monitorar o progresso do projeto;

b) Gerenciamento de configuração e mudanças: define a responsabilidade da organização pelos artefatos gerados no projeto bem como o tratamento das requisições de mudança para mitigar o impacto no processo de desenvolvimento;

c) Ambiente: configuração do processo e de ferramentas de suporte ao desenvolvimento.

3 TESTE DE SOFTWARE

Myers define o teste de software como “o processo de executar um programa com o objetivo de encontrar erros” (MYERS, 2011, p. 6, tradução nossa). À primeira vista, essa definição pode parecer incompleta, pois há a convicção de que sempre testamos algo para verificar se está funcionando. No entanto, a miríade de combinações de entradas de um software torna impraticável um teste de confirmação de que o sistema está 100% funcionando. A definição de Myers, portanto, afirma que o objetivo do teste não é demonstrar que o sistema está livre de erros, mas sim que eles existem.

A abordagem muitas vezes com enfoque destrutivo dessa atividade é um dos problemas de se conseguir um teste bem feito: o testador deve ter sempre como alvo encontrar erros (um teste bem sucedido) e não mostrar que o software funciona (um teste falho), já que nunca conseguiremos 100% de cobertura de todos os casos de teste possíveis.

A Comissão Internacional para Qualificação de Teste de Software (ISTQB, na sigla em inglês) em seu *Syllabus* (ISTQB, 2011), de forma a aumentar a abrangência da atividade de teste de software, define-o como um processo, o qual deve ter um planejamento, modelagem e execução dos testes e, principalmente, quando lidamos com negócios, a definição clara de critérios de aceite dos resultados. Tudo isso correlacionado com os riscos associados à escolha de quais testes serão executados.

Sommerville também adiciona um componente fundamental às metas do processo de teste: “Demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos” (SOMMERVILLE, 2007, p. 355). E esse objetivo pode ainda ser expandido para além do desenvolvedor e o cliente alcançando todos os interessados no processo de desenvolvimento do software.

Para que o processo de testes flua de forma suave e produza as saídas desejadas, o gerenciamento do teste deve garantir a execução correta das atividades. O planejamento deve envolver a seleção dos testes que serão executados e a estimativa do esforço. O esforço está diretamente ligado à eficiência dos casos de teste que serão produzidos na modelagem. A produção desses cenários deve selecionar dados de forma a maximizar a quantidade de erros encontrados para um número finito e controlado de casos de teste (MYERS, 2011).

O teste está fortemente ligado à qualidade do produto e, segundo Myers (2011), a medição da qualidade passa pela execução dos testes, mostrando tanto que o produto faz o que deveria fazer, como também evidenciando que o software não faz o que não deveria fazer.

Não obstante, o teste por si só é incapaz de aumentar a qualidade do produto. Os resultados do teste são indicadores de qualidade, mas sozinhos não a incrementarão. Para aumentar o índice de qualidade do software, um bom teste deve sempre estar integrado com a melhoria contínua no processo de desenvolvimento, na arquitetura e na própria escrita do código (MCCONNELL, 2004).

3.1 VERIFICAÇÃO E VALIDAÇÃO

Verificação se refere ao conjunto de atividades que garante que o software implementa suas funções corretamente. *Validação* se refere às atividades que garantem que o software realmente corresponde ao que o cliente deseja e necessita (GRAHAM, 2007; PRESSMAN, 2006).

Boehm, nos primórdios da Engenharia de Software, já incluía verificação e validação (V&V) como atividades contínuas em todo o desenvolvimento e afirmava: “um dos erros mais caros e com maior ocorrência nos projetos de software atuais é adiar o teste e validação para ocorrerem somente após a construção do software” (BOEHM, 1983, p. 15, tradução nossa). Segundo o autor, existem dois motivos para isso ser considerado um erro: (1) A maioria dos erros de projeto já ocorreu antes da construção iniciar; e (2) quanto mais tarde um erro é encontrado, mais caro é sua correção.

Verificação e validação, portanto, representam conjuntos de atividades que devem ser executados de forma contínua no ciclo de vida do desenvolvimento de software a fim de garantir que o produto saia conforme o especificado e, principalmente, que agregue valor ao cliente.

3.2 NÍVEIS DE TESTE

Os níveis de teste podem ser associados ao modelo V (ROOK, 1986), ou seja, indicam em que fase do ciclo de vida do desenvolvimento os testes devem ser

executados. Serão apresentados quatro níveis de teste: testes unitários, testes de integração, testes de sistema e testes de aceitação.

3.2.1 Testes unitários

Rios (2013) define o teste unitário como:

> [O] estágio mais baixo da escala de testes aplicados nos menores componentes de código criados, visando garantir que estes atendam as especificações, em termos de características e funcionalidade.

Geralmente são testes automatizados e construídos pelos próprios programadores e os defeitos são corrigidos no momento em que são encontrados, sem serem registrados formalmente (GRAHAM, 2007).

3.2.2 Testes de integração

O foco do teste de integração é verificar se os diversos componentes de um sistema conversam corretamente uns com os outros, se são chamados corretamente e se os dados são transferidos com integridade (SOMMERVILLE, 2007).

Devem ser seguidas estratégias para elaboração dos testes de integração. Numa integração bottom-up, o teste é realizado inicialmente pelos componentes de mais baixo nível (infraestrutura, banco de dados, etc.) que não representam funcionalidades do sistema, garantindo assim, que haverá poucos problemas de infraestrutura ao integrar a parte funcional do software. Na integração top-down, faz-se inicialmente um esqueleto geral do sistema e os componentes são adicionados a ele. Torna-se necessário a geração de códigos adicionais para simular o comportamento dos componentes de mais baixo nível. Outras formas de integração podem ser realizadas baseadas em prioridades do usuário ou no sequenciamento do processamento de transações (GRAHAM, 2007; RIOS, 2013).

3.2.3 Testes de sistema

São as validações geralmente realizadas por uma equipe de testes, buscando a visão do usuário e checando os requisitos do sistema. Devem preferencialmente ser executadas em um ambiente operacional controlado para não serem sensíveis a problemas e inconsistências de configuração e dados (RIOS, 2013).

Segundo Graham (2007), testes de sistema devem tratar tanto requisitos funcionais como não funcionais, para tanto, o ambiente de testes deve corresponder o máximo possível ao ambiente de produção.

3.2.4 Testes de aceitação

Pode ser compreendido como o processo de comparar o software com os requisitos e necessidades dos usuários finais. É o teste normalmente executado pelo cliente a fim de aferir se realmente o sistema é o produto estabelecido no contrato (MYERS, 2011).

Graham (2007) define o objetivo do teste de aceitação como estabelecer a confiança no sistema, cujo principal foco não é procurar defeitos e sim avaliar se o software (ou a funcionalidade em avaliação) está pronto para entrar em produção.

3.3 TÉCNICAS DE TESTE

Este trabalho classificará as possíveis técnicas utilizadas nos testes de software da seguinte forma: teste funcional, teste estrutural, teste de regressão e técnicas estáticas.

3.3.1 Teste funcional (caixa-preta)

O objetivo dessa técnica é exercitar o programa de modo a verificar se ele se comporta de acordo com seus requisitos, ficando o testador alheio à estrutura interna. O planejamento do teste caixa-preta requer que o testador defina combinações de entradas e conheça as saídas esperadas. O sistema assim é executado com as entradas especificadas e as saídas reais são comparadas com as esperadas.

Para que seja feito um bom teste funcional, portanto, deve-se garantir que os requisitos estejam bem escritos seguindo o princípio de que os requisitos devem ser testáveis.

O projeto de testes funcionais requer o conhecimento de estratégias específicas para que o teste tenha boa cobertura. Conforme Myers (2011), para ter 100% de cobertura dos requisitos é necessário um teste exaustivo com todas as entradas possíveis. Isso torna inviável a cobertura plena. Ao invés disso, o testador deve se basear em técnicas de modelagem que otimizem a busca de áreas nas quais o sistema pode falhar, como **partições de equivalência, análise de valor limite e grafos de causa e efeito.**

Testes funcionais podem ser projetados tanto para encontrar falhas no sistema (nos níveis de sistema ou integração), como para demonstrar que o software implementa os requisitos elicitados (teste de aceitação).

3.3.2 Teste estrutural (caixa-branca)

O teste caixa-branca é uma estratégia que leva o testador a olhar dentro do sistema, sua estrutura, sua lógica, seu comportamento. Os testes são derivados do conhecimento da implementação do software.

Para termos 100% de cobertura aqui, seria necessário exercitar todos os caminhos possíveis no código fonte. Novamente, torna-se virtualmente impossível garantir essa cobertura. Algumas estratégias utilizadas para maximizar a quantidade de erros encontrados para um número finito e controlado de casos de testes são **cobertura de declarações** e **cobertura de decisões.**

A complexidade do teste estrutural será proporcional à complexidade da rotina (função/método) desenvolvida, portanto o projeto dos testes será simplificado se o código fonte tiver rotinas curtas e expressões booleanas simples (MCCONNELL, 2004).

Uma desvantagem do teste estrutural é que esse teste não garante que o sistema escrito atenda suas especificações. Assim, torna-se necessário integrar essas técnicas com os testes funcionais (MYERS, 2011).

3.3.3 Técnicas estáticas (revisões)

Técnicas estáticas compreendem revisões de artefatos do sistema em encontros colaborativo, nos quais não se pressupõe a execução do software em teste.

O ISTQB lista como benefícios das revisões:

> Detecção e correção antecipada de defeitos, ganho no desenvolvimento em termos de produtividade, redução do tempo no desenvolvimento, redução do custo e tempo de teste, menos defeito e melhoria na comunicação (ISTQB, 2011, p. 30).

De fato, a comunicação é a característica mais marcante das revisões, pois durante as reuniões ocorre disseminação do conhecimento. Entretanto, como frisa McConnell (2004), o objetivo das revisões é descobrir defeitos no projeto ou no código e não explorar alternativas ou debater quem está certo e quem está errado. A experiência deve ser positiva para todos os participantes. Algumas estratégias de análise estática incluem **inspeção, acompanhamento (walkthrough) e a programação em pares.**

Esse tipo de teste pode ser estendido a diversos artefatos do sistema, além do código fonte, como documentos, especificações de requisitos e até os casos de teste utilizados nos testes funcionais e estruturais. Uma omissão encontrada antecipadamente nos requisitos pode reduzir custo e prazo se comparado quando encontrada nos testes de aceitação, por exemplo (MYERS, 2011).

Atualmente, é crescente o uso de ferramentas para automatizar a busca de possíveis erros na análise estática e diversas ferramentas também se integram com as IDEs para agilizar a busca (e consequente correção) de erros (ISTQB, 2011).

3.3.4 Teste de regressão

O teste de regressão está fortemente ligado à manutenção de software. Mudanças nas necessidades dos usuários, melhorias e adaptações envolvendo requisitos não funcionais e até mesmo a correção de erros geram alterações em sistemas na fase de pós-implantação. O teste de regressão deve ser usado para garantir que essas modificações não causem impacto nas funcionalidades já existentes (BELL, 2005).

Graham (2007) define o teste de regressão como o teste repetido após a modificação, de um programa que já foi testado, para descobrir a existência de algum defeito introduzido ou não coberto originalmente como resultado da mudança. Deve ser realizado quando o software, ou seu ambiente, é modificado.

Myers (2011) defende que não basta apenas executar novamente uma bateria de testes para garantir o teste de regressão. Se for necessário retestar um programa, os casos de teste devem ser reinventados para que mantenham sua qualidade. Para um plano de testes simplesmente reexecutado, a probabilidade de encontrar novos erros é pequena, o que pode dar a sensação de que a qualidade do sistema aumentou, quando, na verdade, os testes é que não exercitaram as áreas do programa com os novos defeitos.

Devido ao alto custo dessa reescrita, o teste de regressão costuma ser evitado. Além disso, a reexecução de testes é uma tarefa burocrática e o testador tende a negligenciar a tarefa. Uma forma de diminuir esse impacto é automatizar os testes de regressão mais utilizados (MCCONNELL, 2004).

A quantidade de teste de regressão a ser utilizada no projeto deve ser baseada no risco de não encontrar defeitos no software que estava funcionando previamente.

3.4 GERENCIAMENTO DO TESTE

Sendo o teste de software um processo, será ele mais eficiente quando tiver suas atividades planejadas antecipadamente e quando a equipe tiver definida uma estratégia de execução do processo.

As técnicas previamente listadas nesta seção devem ser consideradas e selecionadas de acordo com o tipo de projeto de software que se esteja trabalhando. Pressman (2006, p. 289) resume a estratégia que deve ser adotada no teste em quatro tópicos:

a) Revisões (técnicas estáticas);

b) Definição dos níveis de teste que serão executados;

c) Escolha oportuna das técnicas de modelagem de teste em diferentes momentos;

d) Quem conduzirá o processo de teste (a própria equipe de desenvolvimento ou um grupo de teste independente).

O gerenciamento dos casos de teste costuma ser desafiador e considerando o alto custo do teste, torna-se importante.

Louridas (2011) apresenta dados que indicam que o teste de software representa de 16 a 29% do total do esforço de construção do sistema (sendo maior o percentual quanto maior o tamanho e mais riscos estiverem envolvidos no projeto). Este número, portanto, deve conduzir a equipe a desenvolver os testes de forma mais eficiente possível.

3.4.1 Organização do teste

A organização do processo de teste refere-se à definição de quais níveis de teste serão executados e quais equipes os executarão. A dúvida mais comum nesse caso é: deve-se usar uma equipe de testes independente ou não?

A resposta nunca é fácil ou mesmo definitiva. Do ponto de vista da qualidade, Graham (2007) afirma que o teste, como parte da garantia da qualidade, nem sempre é um feedback positivo para a equipe que desenvolveu o software. Assim, justifica-se que um time independente possa produzir uma avaliação mais objetiva e impessoal do sistema. Entretanto, o isolamento pode se traduzir em problemas de comunicação e falta de identificação da equipe de testes com o objetivo do projeto.

No outro extremo, há os casos em que é delegado aos próprios desenvolvedores testar o código. Esse caso encaixa-se na atividade psicologicamente destrutiva em que o programador deve buscar quebrar seu próprio código. A responsabilidade de apontar falhas no próprio trabalho torna essa organização complexa, perdendo a imparcialidade necessária no teste do sistema.

Na prática, é possível adotar uma estratégia entre os dois extremos, cujo foco é dar responsabilidade ao desenvolvedor de modo a garantir a qualidade das unidades individuais do software (teste no nível de componente), certificando-se de que cada unidade realiza a função para a qual foi projetada (PRESSMAN, 2006).

O papel da equipe independente de testes costuma ser mais eficiente quando se chega ao nível de teste de sistema. Myers (2011) afirma categoricamente que os programadores não devem executar um teste de sistema e, ainda, que, de todas as fases no processo de teste, essa é a fase que a organização responsável pelo desenvolvimento definitivamente não deve executar.

Mesmo servindo-se de um grupo independente, a equipe de testes deve fazer parte da equipe do projeto, acompanhando as fases de análise e projeto, sempre que possível (PRESSMAN, 2006).

Os clientes ou usuários finais também devem participar do processo de teste, principalmente na execução do último nível de teste, o de aceitação, no qual é feito o batimento entre os requisitos e o sistema concluído (ou parte dele). Beck (1999) enfatiza em sua metodologia ágil (Extreme Programming) que o cliente também deve ser responsável por escrever os casos de teste funcionais, não diretamente em formato técnico, mas com a ajuda de um testador que possa traduzir os dados informados pelo cliente em testes.

3.4.2 Planejamento

O plano de testes é um documento completo que engloba não só a descrição dos testes de software, mas todos os padrões utilizados, alocação de recursos, cronograma e metas. Como já dito anteriormente, esse planejamento deve iniciar o quanto antes no ciclo de vida de desenvolvimento, porém é importante lembrar que o plano de testes não é um documento estático. Atrasos em outros estágios do desenvolvimento implicarão em mudanças no plano, principalmente no que diz respeito à alocação de recursos (SOMMERVILE, 2007).

Brooks (1995, p. 20, tradução nossa), no clássico Mythical Man-Month, descreve a importância do planejamento do teste:

> Não deixar tempo suficiente para o teste do sistema, em particular, é desastroso. Como o atraso surge no final do cronograma, ninguém está atento ao problema até o projeto chegar perto da data de entrega [...]. Atrasos neste ponto trazem sérias repercussões financeiras e psicológicas. O projeto geralmente está com a equipe completa alocada e o custo por dia de trabalho chega ao nível máximo. Além disso, o custo secundário do atraso (ligado ao suporte que o software dará às atividades da empresa que o terá implantado) pode ser maior que todos os outros. É, portanto, muito importante alocar tempo suficiente para o teste do sistema no cronograma original.

O maior erro no planejamento costuma ser a crença de que não serão encontrados erros. Essa suposição forçosamente implicará em subdimensionamento de recursos e prazo para a execução dos testes. Somam-se a esse fato outros dois problemas: (a) o processo de teste está localizado no final do ciclo de vida do

desenvolvimento, onde o prazo já está apertado e a mudança de recursos requer grande esforço; e (b) o teste não é planejado tendo em vista sua correta definição (“o processo de executar um programa com o objetivo de encontrar erros”) e consequentemente os erros não são encontrados (MYERS, 2011).

Um bom plano de testes é crucial para que o processo possa ser bem executado. Segue uma compilação de itens encontrados na literatura que devem ser considerados para um bom plano de testes. A utilização, ou não, de cada item deve ser considerada de acordo com o projeto em que se está trabalhando. Fatores como risco, complexidade, importância e criticidade devem ser levados em conta para definir o conteúdo do plano de testes.

a) *Objetivos*: o objetivo de cada fase do teste deve estar bem claro, junto com escopo determinado (MYERS, 2011);

b) *Critérios de saída*: definem quando parar o teste, sendo que deve haver critérios específicos para cada nível de teste (MYERS, 2011). Estes critérios podem ser constituídos de métricas (cobertura de código ou número de erros encontrados, por exemplo) ou então considerar dados de custos, riscos e cronograma (GRAHAM, 2007);

c) *Cronograma*: deve indicar as fases de planejamento, especificação, modelagem e execução dos testes. Metodologias ágeis, como XP, exigem que os testes automatizados estejam prontos antes da construção iniciar (BECK, 1999; MYERS, 2011);

d) *Responsabilidades*: devem ser listados os responsáveis por executar as tarefas de cada fase do cronograma, bem como as equipe responsáveis pelas correções dos erros encontrados. Deve ficar claro também quais integrantes da equipe de desenvolvimento ajudarão a equipe de testes (MYERS, 2011);

e) *Padrões de casos de teste*: os padrões utilizados para identificar e escrever os casos de teste, bem como o nível de detalhe desejado (MYERS, 2011);

f) *Ferramentas*: as ferramentas necessárias devem ser listadas, incluindo plano para desenvolvimento ou aquisição. Deve também conter informação de como a ferramenta será usada e em que fase do processo (MYERS, 2011);

g) *Configuração do ambiente*: hardware e software necessários para execução do teste (MYERS, 2011);

h) *Rastreabilidade*: envolve o procedimento para permitir a rastreabilidade a partir dos erros reportados, os passos para reproduzi-lo, requisitos e fontes afetados (MYERS, 2011);

i) *Procedimento para registro dos testes e defeitos*: o registro do teste deve ser bom o suficiente para que sua execução possa ser auditada (SOMMERVILLE, 2007);

j) *Teste de regressão*: caso alguma funcionalidade atual seja impactada, deve ser descrito no plano a estratégia a ser usada para retestá-la. Dependendo do risco, pode envolver outras funcionalidades críticas não diretamente afetadas (MYERS, 2011);

k) *Métricas*: devem ser definidas que métricas serão utilizadas no projeto de testes a fim de aferir a qualidade de produto e processo (ISTQB, 2011).

3.4.3 Estimativas

A estimativa do teste está ligada ao esforço necessário para atingir os objetivos descritos no plano de testes e para definir recursos e cronograma.

Graham (2007) lista duas abordagens possíveis para estimar o esforço a ser gasto no processo de testes:

a) Estimativa de esforço baseado em métricas de projetos similares ou em valores típicos;

b) Estimativa das tarefas pelo próprio executor ou por especialistas.

Além desses, os critérios de saída do teste tem impacto direto na estimativa de esforço. Como descreve Myers (2011), três estimativas devem ser feitas para definir aqueles critérios:

a) Estimativa do total de erros no sistema;

b) Estimativa de quantos desses erros podem efetivamente ser encontrados durante o teste;

c) Estimativa de quantos erros são originários de cada fase do processo de desenvolvimento e em quais fases do processo de testes esses erros serão encontrados.

A estimativa (a) pode ser encontrada utilizando a primeira abordagem definida por Graham (estimativa de esforço baseado em métricas de projetos similares ou em valores típicos), incluindo um pré-teste com esse objetivo. Por outro lado, o item (b), segundo Myers (2011), envolve estimativas arbitrárias, levando em consideração a natureza do programa e as consequências de erros não encontrados. Na opinião do autor, a terceira estimativa é a mais difícil de derivar. Pode-se partir de dados existentes: “considera-se que aproximadamente 40% dos erros são inseridos na codificação e o resto é gerado nas fases iniciais processo”, por exemplo. Entretanto, é necessário que a equipe desenvolva suas próprias estimativas para seus processos e características dos sistemas.

Graham (2007) lista fatores que podem impactar na estimativa de esforço dos testes:

a) Presença, ou não, de documentação atualizada do sistema;
b) Complexidade associada ao problema que o software resolve (regras de negócio) e também à solução utilizada para tal;
c) Complexidade associada à tecnologia utilizada;
d) Regras e normas de segurança a que o software deve atender;
e) Características da comunicação entre as equipes;
f) Ferramentas disponíveis para auxiliar na manutenção dos casos de teste, execução e automação dos testes e registro de defeitos;
g) Modelo de desenvolvimento: o modelo V tende a ser mais suscetível a grandes impactos que ocorrem no final do desenvolvimento, enquanto que modelos incrementais tendem a ter alto custo em testes de regressão.

3.4.4 Métricas de Teste

Para controlar se os casos de teste são eficientes, a única maneira é medindo a execução. Louridas (2011) lista algumas métricas que podem ser utilizadas para controlar a execução do teste e aferir a qualidade:

a) Eficiência de remoção de defeitos: também chamado de eficácia do teste, é a relação entre o número de erros encontrados no teste e o total de erros encontrados (seja em produção, por usuários, etc.);

b) Eficiência do teste: número de erros encontrados dividido pelo esforço despendido para encontrá-los. O esforço pode ser medido, por exemplo, pelo número de casos de teste. Esta métrica pode ser usada também para mostrar quais testes são mais eficazes na busca de erros;

c) Taxa de sucesso: relação entre os casos de testes que tiveram sucesso (no sentido de executar o programa corretamente, sem erros) e o total de casos de teste. Esta métrica deve ser entendida com o sentido de falha, ou seja, se tiver um valor muito alto, é mais provável que os casos de teste estejam mal elaborados do que o sistema esteja livre de erros;

d) Sucesso versus falha: razão entre testes que passaram (sem erros) e os que falharam (com erros). Útil para mostrar falhas em projetos com um grande número de casos de teste;

e) Cobertura de testes: mede o quanto o teste cobre o código a ser testado. Pode ser medido no nível funcional, no nível de usuário (requisitos, casos de uso, histórias) ou, com ajuda de ferramentas, diretamente no código-fonte.

4 VISÃO GERAL DO PROCESSO DE DESENVOLVIMENTO DA EMPRESA

O TRT-SC não tem a tecnologia da informação como atividade fim, mas possui uma área de informática especializada cuja missão é “prover soluções e serviços de TI que priorizem e viabilizem o cumprimento da função institucional do tribunal”. A área de TI presta suporte aos funcionários, atua como provedora de soluções de infraestrutura e de desenvolvimento de software para auxílio na atividade fim do órgão.

A área responsável pelo desenvolvimento de sistemas para a web segue um processo prescritivo e que abrange as quatro fases do RUP (KRUCHTEN, 2003), conforme pode ser visto na figura 4.

4.1 PAPÉIS

Figura 4 – Adaptação do RUP.

Fonte: Elaboração dos autores, 2014.

O grupo de desenvolvimento é bastante heterogêneo e as equipes têm deficiências em determinados perfis, que não podem ser supridos facilmente, já que depende de abertura de vagas em concurso.

Os papéis existentes hoje são:

a) Líder técnico: responsável técnico pelo projeto, coordena o time de desenvolvimento.

b) Analista de requisitos: responsável por conduzir os encontros e conversas com os clientes e usuários a fim de levantar os requisitos dos sistemas. Deve ter conhecimento pleno do negócio. Confunde-se também com um papel de analista de negócio, quando o usuário/cliente não tem presença constante na equipe do projeto.

c) Analista de sistemas: responsável por validar o projeto físico entregue pela fábrica de software. Em alguns projetos também identifica soluções de mais baixo nível e/ou de integração entre sistemas a fim de ajudar no detalhamento para a fábrica.

d) Arquiteto: tem conhecimento da arquitetura corporativa e valida se a solução (projeto físico e código fonte) entregue pela fábrica segue os padrões do TRT-SC.

e) Testador: responsável por validar (V&V) o software entregue pela fábrica.

f) Preposto da fábrica de software: responsável pelo contrato do lado da fábrica.

g) Gerente de projetos da fábrica: responsável pelo encaminhamento das atividades na fábrica de software. Costuma ser um ponto único de comunicação entre a equipe técnica do TRT-SC e a empresa terceirizada.

h) Analista da fábrica: representa, no modelo, a equipe técnica da fábrica de software. Ao contratante (TRT-SC) é vedado o gerenciamento interno da empresa contratada, portanto a equipe técnica da empresa não é detalhada.

Na prática, apenas o papel de gerente de projetos é especialista e ocupado pelos diretores das subáreas de desenvolvimento. Os outros papéis costumam ser aplicados a todos os membros da equipe. Existe certo padrão informal seguido pelas equipes, no qual se tem a situação em que o analista de requisitos e o testador são representados pelas mesmas pessoas, assim como o analista de sistemas e o arquiteto.

Já o papel de desenvolvedor é terceirizado para uma fábrica de software. O contrato que guia esta parceria é pré-definido antes do processo licitatório e a

técnica utilizada para a medição do trabalho executado é a análise por pontos de caso de uso (IBARRA, 2010).

4.2 FASES DO CICLO DE VIDA

O detalhamento das atividades executadas pela organização dentro de cada fase do processo de desenvolvimento será visto a seguir. As cinco fases (concepção, elaboração, construção, homologação e implantação) correspondem a uma iteração completa no ciclo de vida do desenvolvimento, cujo planejamento pode envolver mais de uma iteração para conclusão do projeto.

4.2.1 Concepção

As solicitações de novos sistemas ou melhorias vêm da área responsável pelo planejamento estratégico já com o caso de negócio pronto. Após o acordo entre as partes envolvidas (presidência, área de negócio afetada pelo software e a TI), inicia-se o processo. A gerência prepara o documento de visão e um esboço do plano de projeto, com estimativa de cronograma, possíveis riscos e os recursos envolvidos. Enquanto isso, inicia-se o levantamento de requisitos com entrevistas e discussões com usuários e entre a equipe técnica.

A documentação é totalmente feita na ferramenta Entreprise Architect (EA) mantendo rastreabilidade entre requisitos, regras de negócio e casos de uso. Entretanto, para projetos de melhorias, não é feita uma análise criteriosa de quais casos de uso serão impactados por essas melhorias. Não há preocupação também em definir cenários de uso do sistema, com dados reais, de modo a antecipar possíveis casos de teste a serem executados na entrega do software.

Após a conclusão dessa fase, é gerada uma estimativa de custos baseada na métrica de análise por pontos de caso de uso.

4.2.2 Elaboração

Nesta fase, os artefatos gerados na concepção são detalhados e o foco são os riscos técnicos e arquiteturais. Foi padronizada a linguagem Java para todos

os novos softwares desenvolvidos pela a equipe, bem como um framework específico que define uma arquitetura básica.

Na fase de elaboração aparece a questão contratual com a fábrica de software. Devem ser repassados à empresa contratada os requisitos (no EA), o plano de projeto e o documento de visão, bem como um documento de arquitetura contemplando os principais componentes e padrões de software usados pelo órgão.

O retorno da fábrica de software é um conjunto de artefatos com a elaboração (projeto físico) do sistema. Por sofrer auditorias, todos os artefatos definidos no contrato devem estar presentes, mesmo que sejam pouco informativos. Por exemplo:

a) Detalhamento de todos os casos de uso com fluxos básico e alternativos;
b) Diagramas de sequência para os casos de uso que envolvam regras de negócio adicionais ao padrão estabelecido no framework escolhido;
c) Descrição de métodos e seus comportamentos;
d) Protótipos refinados de todas as telas do sistema.

Ao final dessa fase, deve estar pronta a planilha de pontos por caso de uso da iteração que será realizada com a fábrica de software.

4.2.3 Construção

A fase de Construção inicia com o aceite da elaboração produzida pela fábrica de software. A gerência da fábrica, com base na planilha de pontos de caso de uso, entrega o cronograma previsto para a construção.

Como ferramental, é utilizado o software de controle de versão do TRT-SC como repositório de código fonte. Existe um ambiente de integração contínua que realiza uma construção completa no sistema após cada *commit* no repositório e acusa por email qualquer falha na construção. Diariamente é executada uma ferramenta de análise estática de código, a fim de coletar as métricas necessárias para a manutenção mínima do software.

O acompanhamento dos chamados à fábrica de software é feito cadastrando tíquetes na ferramenta Atlassian Jira, permitindo rastrear os commits. Um ponto negativo durante essa fase é que há pouco envolvimento dos clientes

(áreas demandantes ou possíveis usuários do software) com a equipe de desenvolvimento do TRT-SC.

O produto de trabalho da fábrica é o código fonte e a documentação de configuração do sistema. A essa lista, somam-se as evidências de teste anexadas ao Jira.

Não é cobrada da fábrica uma metodologia de teste formal, nem cenários obrigatórios que devam ser evidenciados como funcionais. As evidências de teste acabam por ser meramente ilustrativas.

Ao receber o software da empresa contratada, os analistas responsáveis fazem testes exploratórios de forma aleatória. Não é utilizado nenhum método para derivar valores que aumentem a cobertura dos testes realizados, tampouco é investido tempo em testes de regressão e testes de desempenho.

4.2.4 Homologação

Essa fase compreende uma homologação assistida, em que usuários são chamados a testar o sistema podendo seguir um roteiro proposto pela equipe técnica, mas também realizando testes livres. O teste se caracteriza pelo uso em um ambiente semelhante ao de produção e serve também para formalizar um aceite completo em relação ao contrato da fábrica de software.

Após a homologação, os clientes ainda podem solicitar melhorias que, se forem imprescindíveis para a implantação do sistema, são realizadas imediatamente iniciando uma nova iteração com a fábrica de software. Caso os clientes aceitem que as melhorias sejam feitas posteriormente, o sistema é liberado para a implantação.

4.2.5 Implantação

A fase de Implantação inclui treinamento dos usuários, uso paralelo com o sistema legado (se aplicável), implantação do produto, suporte, correção de erros e implementação de melhorias que foram adiadas.

Em relação aos erros encontrados em produção, não são coletadas métricas no sentido de aferir se o teste do sistema foi bem realizado.

4.3 CONSIDERAÇÕES FINAIS

Este processo de desenvolvimento vem sendo utilizado (e melhorado) ao longo de oito anos e vem trazendo bons resultados perante a administração do TRT e os outros tribunais do trabalho, como já comentado na introdução. Uma boa parte desse reconhecimento deve-se ao fato do investimento realizado na engenharia de requisitos. Os casos de uso e requisitos são a interface entre a equipe do TRT e a fábrica de software e influenciam diretamente na qualidade do sistema entregue.

Apesar de ser adotado um processo prescritivo, as equipes consideram que não há excesso de burocracia na elaboração de documentos e modelos. O processo é constantemente repensado, mas, durante a vigência de um contrato com fábrica, não é simples alterar esse processo de trabalho, já que envolve questões legais. Assim, a cada novo contrato, busca-se melhorar o processo, rever a necessidade de artefatos e alterar a forma de trabalho e comunicação com a empresa terceirizada.

Um ponto fraco no processo é o excesso de dependência de mão-de-obra terceirizada para o desenvolvimento. Essa realidade resulta em dificuldade de correções rápidas nos sistemas em produção, já que há poucos desenvolvedores na equipe. Outro ponto fraco identificado foi justamente o assunto tratado nesta pesquisa: o processo de teste de software.

5 PROCESSO DE DESENVOLVIMENTO COM METODOLOGIA DE TESTE DE SOFTWARE

Este capítulo apresenta a proposta de metodologia de teste de software utilizada como base para a implantação das atividades de teste na empresa. As alterações concentraram-se nas fases de concepção, elaboração, construção e homologação, cujos diagramas detalhados são apresentados a seguir. Nesses diagramas as atividades de teste (propostas neste trabalho) são identificadas em azul. As atividades que já eram executadas no ciclo de vida são mostradas em nível macro, pois não representam o foco deste trabalho.

Na descrição das atividades, será feito referência a ferramentas utilizadas pela empresa, que já foram identificadas no capítulo anterior.

5.1 OBJETO DA MELHORIA E ATIVIDADES FORA DO ESCOPO

Esta seção aborda o objeto da melhoria no processo bem como as atividades que foram deixadas fora do escopo inicial.

5.1.1 O que o trabalho se propõe a demonstrar

O foco do trabalho realizado é sobre testes funcionais, planejamento dos testes, verificação e validação dos artefatos dos sistemas. A metodologia de testes funcionais foi confeccionada baseada no processo de desenvolvimento de software já implantado na empresa (descrito na seção anterior).

Dois produtos de trabalho foram projetados:

a) Gerenciamento do ciclo de vida dos requisitos e casos de uso na ferramenta Enterprise Architect baseado nas atividades de validação e verificação;

b) Adoção de um plano de testes formal (contendo planejamento, especificação e resultados de execução dos testes) utilizando o software Testlink, integrado com as demais ferramentas da empresa;

Esta pesquisa não detalhará a implantação nem o uso das ferramentas utilizadas e/ou mencionadas no texto, apenas apresentará as atividades nas quais elas foram sugeridas, bem como as possíveis consequências de sua utilização.

5.1.2 Delimitações da Pesquisa

Não é objeto deste trabalho a proposta de atividades de testes estruturais (caixa-branca), técnicas de inspeção de código fonte, nem automação de testes, já que a construção é realizada por empresa terceirizada e existem poucos desenvolvedores no quadro do TRT-SC a fim de analisar o código fonte e programar testes automatizados, sejam unitários ou funcionais.

5.2 PAPÉIS

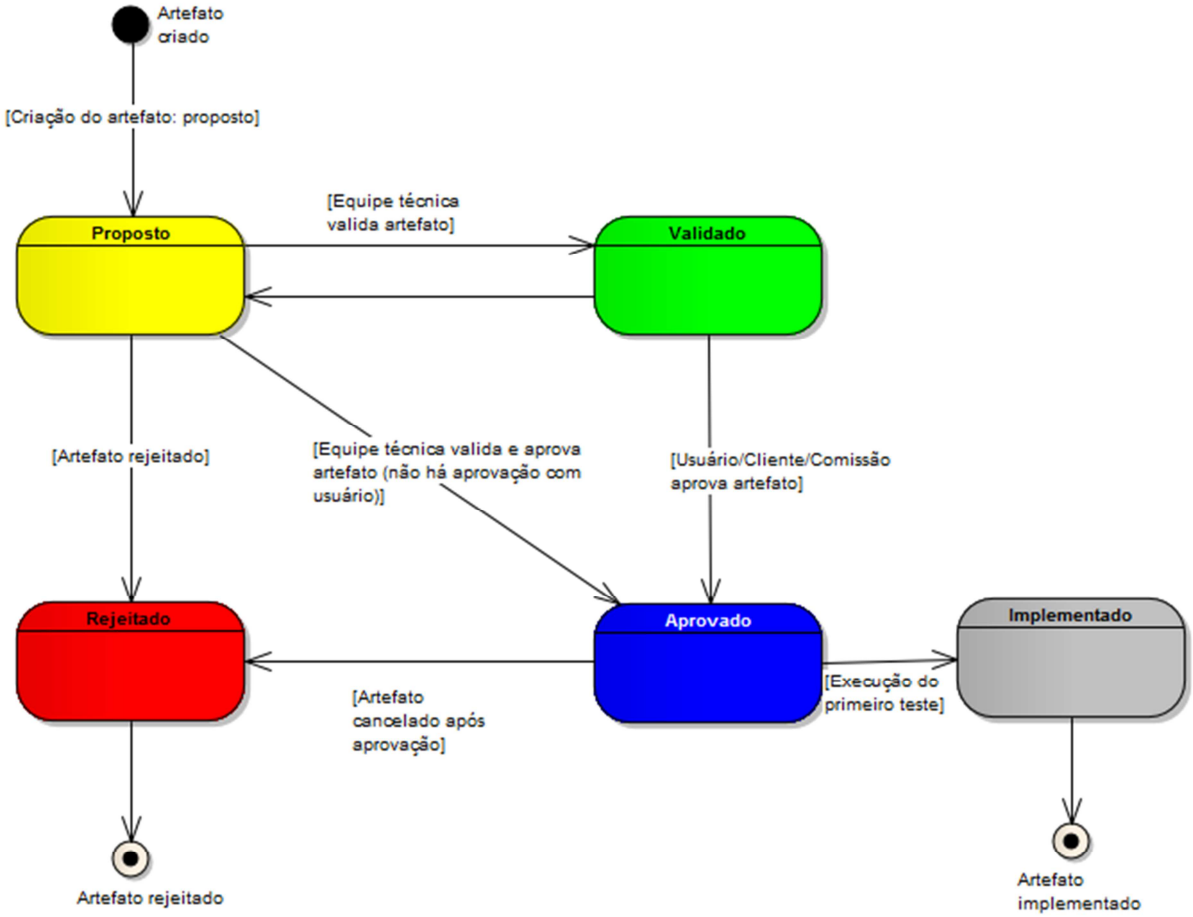
Todos os papéis já existentes no processo de desenvolvimento se mantêm. Ao testador, será acrescida responsabilidade de modelar e executar os casos de teste e comunicar à equipe do projeto as falhas encontradas. Sugere-se na melhoria do processo a criação do papel de **líder de testes**, inicialmente também compartilhado pela mesma pessoa que exerce a liderança técnica da equipe.

5.3 CICLO DE VIDA DOS REQUISITOS

Essa seção apresenta uma proposta de abordagem para garantia da qualidade com foco na validação e verificação de requisitos e casos de uso. Partindo das definições de técnicas estáticas para inspeção de código, foi derivada uma metodologia de forma a garantir que os requisitos levantados e entregues para a fábrica de software representem o mais fielmente possível o que o usuário deseja e estejam dentro do que é factível realizar. Essa metodologia abrange também à execução dos testes no sistema, a ser apresentada na seção 5.4.

A figura 5 ilustra os possíveis estados em que um requisito ou caso de uso (nomenclatura que na imagem foi generalizada para “artefato”) podem estar, quais sejam: proposto, validado, aprovado, implementado e rejeitado. As cores representam as mnemônicas utilizadas para facilitar a identificação visual dos artefatos no EA. A proposta define pré-requisitos para a troca de fases do projeto, baseado no estado dos artefatos da documentação, e uma padronização da manutenção de requisitos e casos de uso durante o projeto.

Figura 5 - Ciclo de vida dos requisitos e casos de uso.



Fonte: Elaboração dos autores, 2014.

Uma convenção a ser seguida é que um caso de uso só terá seu estado alterado quando todos os requisitos associados a ele também forem alterados. Por exemplo, um caso de uso só será considerado aprovado, quando todos os requisitos que implementa forem também aprovados.

Segue uma descrição de cada estado, bem como as regras envolvidas na transição entre os estados.

5.3.1 Estados possíveis de requisitos e casos de uso

a) *Proposto*: um artefato recém criado cai diretamente no estado Proposto e indica que não houve revisão pela equipe técnica ou pela equipe de negócio. No âmbito do contrato com fábrica de software, é importante que os analistas da fábrica rejeitem artefatos "propostos", pois não passaram por revisões.

b) *Validado*: demonstra que o artefato foi validado pela equipe técnica (representa um "teste da documentação"). A fábrica de software não deve receber artefatos “validados”, pois indica que não houve conferência pela equipe de negócio do TRT-SC, portanto não há certeza de que o artefato realmente seja uma necessidade real do usuário.

c) *Aprovado*: indica que o artefato foi aprovado para ser implementado (é viável de ser construído do ponto de vista técnico e de negócio). A aprovação pode ser feita pelo cliente, por um comitê ou mesmo pela equipe técnica (depende do projeto).

d) *Implementado*: ilustra um artefato já implementado no sistema. Esse estado é importante quando a documentação de uma nova versão precisar referenciar um requisito implementado em versões anteriores. É um estado final do requisito.

e) *Rejeitado*: estado para guardar histórico de requisitos e casos de uso rejeitados no projeto atual. Não representa exclusão de funcionalidades em versões futuras. Um artefato implementado nunca deve ser alterado para rejeitado. Devem ser registrados data, local, justificativa e os responsáveis pela rejeição.

5.3.2 Transições entre os estados

As possíveis transições entre os estados definidos no ciclo de vida dos artefatos são detalhadas a seguir.

a) *Proposto para Validado:* quando for feita a revisão técnica do requisito ou caso de uso, utilizando-se uma técnica de revisão informal entre os membros da equipe, o artefato é considerado validado.

b) *Proposto para Aprovado:* A passagem direta do estado Proposto para Aprovado ocorre quando não houver necessidade de aprovação do requisito pelo cliente (considera-se pré-aprovado). Nesse caso, após validar o requisito, ele deve ir diretamente para o estado Aprovado.

c) *Proposto para Rejeitado:* Quando a equipe técnica decidir que o requisito não pode ser implementado (com anuência do cliente).

d) *Validado para Aprovado:* Quando o artefato for aceito pelo usuário.

e) *Validado para Proposto:* Quando for detectada uma pendência ou lacuna no requisito que o torne "não válido".

f) *Aprovado para Implementado:* Quando o testador verificar que o requisito está implementado no sistema. Caso encontre um erro referente ao requisito, o testador deve registrar esse erro no Jira e assim mesmo marcar o artefato como implementado. O acompanhamento da correção do erro passará a ser feito pelo chamado no Jira. A transição só não deve ocorrer quando o requisito efetivamente não tiver sido implementado (por esquecimento ou falha no processo de desenvolvimento).

g) *Aprovado para Rejeitado:* Quando a equipe técnica decidir que o requisito não pode ser implementado ou quando o usuário decidir não querer mais o requisito durante o curso do projeto.

5.4 PLANO DE TESTES

Serão descritas a seguir as atividades incluídas para controle de qualidade dos sistemas, baseado em uma proposta de plano de testes funcionais, a fim de realizar a verificação do sistema contra seus requisitos. As atividades serão detalhadas dentro de cada fase do processo de desenvolvimento.

5.4.1 Fase de concepção

O objetivo na fase de concepção foi iniciar o planejamento dos testes necessários tanto para o projeto como um todo, como também para a iteração corrente. O produto de trabalho esperado para essa etapa é um plano de teste com, no mínimo, a lista de casos de teste (sem descrição detalhada ainda) ordenada por priorização, baseadas no risco e impacto das funcionalidades.

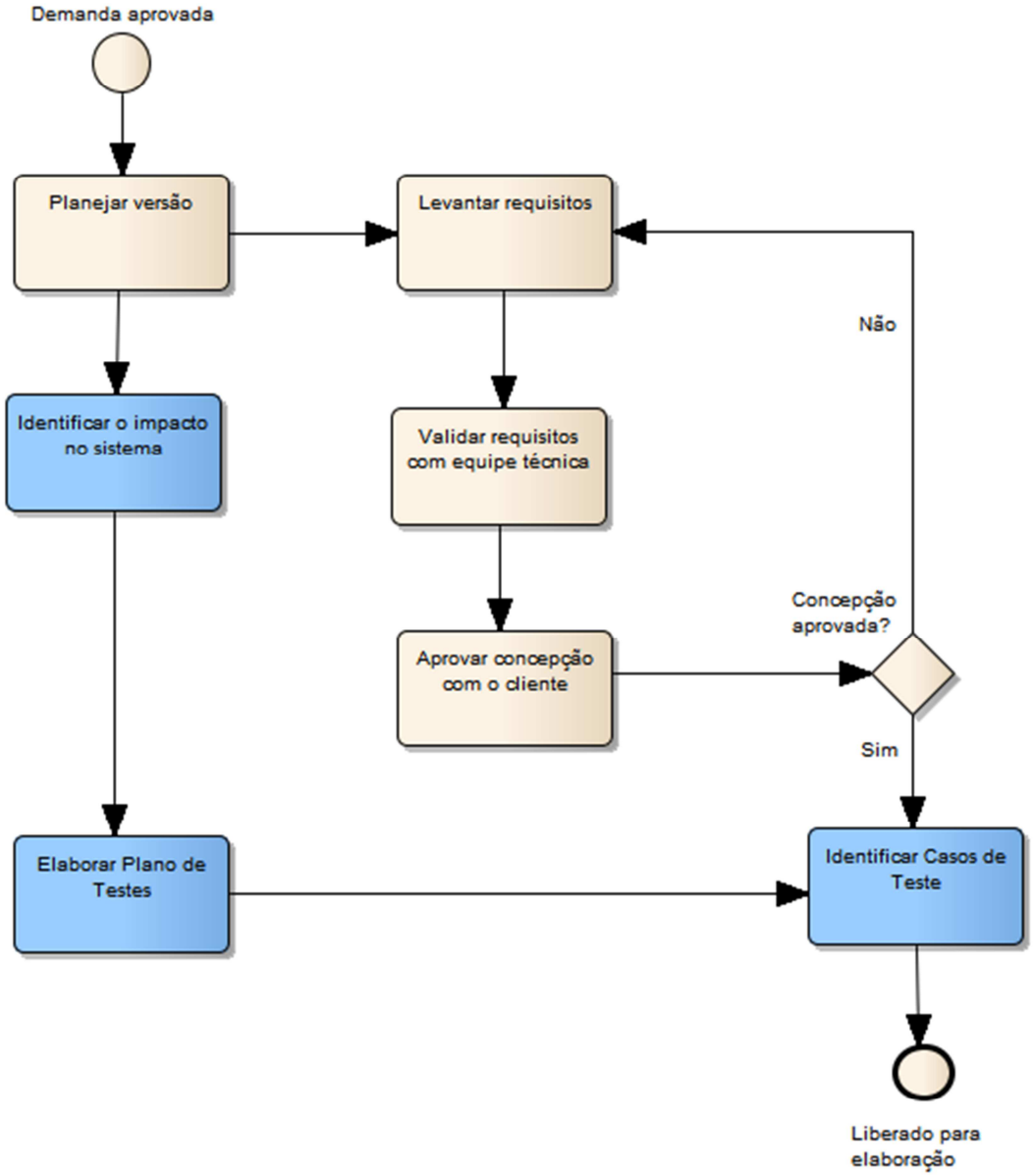
As alterações no processo são exibidas na figura 6 e as novas atividades são detalhadas a seguir.

5.4.1.1 Identificar impacto no sistema

A identificação do impacto no sistema é uma atividade realizada pelo analista de sistemas (que deve ter conhecimento tanto da regra de negócio quanto

do código) e de um arquiteto para aprovação de alterações estruturais dos dados do sistema, responsáveis pela demanda.

Figura 6 - Alterações na fase de Concepção



Fonte: Elaboração do autor, 2014.

Os casos de uso, diagrama de classes e modelo de entidades (caso existirem) devem ser analisados em conjunto para identificar os eventuais impactos da nova versão.

O uso de ferramentas (dinâmicas ou estáticas) para avaliar o acoplamento e/ou dependência entre os módulos afetados deve ser considerado.

Se for uma nova versão de um software, é necessário identificar quais novas funcionalidades serão entregues e quais casos de uso são impactados pelas falhas a serem corrigidas. Com essa informação pode-se identificar quais testes deverão ser criados e quais deverão ser executados.

Se for um projeto de software novo (primeira versão) é necessário identificar quais funcionalidades esse software possuirá. Esse insumo é importantíssimo para identificar quais testes serão necessários para validar o software.

O quadro 1 apresenta os papéis envolvidos, entradas e saídas da atividade de identificação do impacto.

Quadro 1 – Detalhamento da atividade Identificar Impacto no Sistema

Papéis
• Analista de sistemas; • Arquiteto; • Líder de testes.
Entradas
• Especificação de requisitos do software; • Documentação (inclui documentação do Enterprise Architect, emails, lista de tíquetes no Jira, etc.) listando quais funcionalidades serão implementadas e quais falhas serão corrigidas; • Casos de uso (opcional); • Modelo de dados ou diagrama de classes (opcional); • Documento de arquitetura de software (opcional).
Saídas
Criação no Testlink das atividades relacionadas a testes de acordo com o tipo do projeto: • Se atualização de software: o Gerar plano de testes específico para a versão; o Identificar os casos de teste. • Se projeto novo de software: o Gerar plano de testes geral para o software; o Gerar plano de testes específico para a versão;

o Identificar os casos de teste.

Fonte: Elaboração do autor, 2014

5.4.1.2 Elaborar plano de testes

Um sistema deve ter sempre um Plano de Testes Global (PTG) – válido para o sistema inteiro – e um plano de testes específico para cada *release*.

O PTG terá as seguintes seções:

a) Objetivo do plano de testes;
b) Escopo dos testes;
c) Aspectos que serão testados;
d) Aspectos que não serão testados;
e) Abordagem do teste;
f) Ambiente;
g) Cronograma;
h) Riscos;
i) Responsabilidades;
j) Rastreabilidade;
k) Resultado dos testes.

Se for um projeto de atualização de versão, o PTG já deve estar criado. O plano de testes específico será um apêndice do PTG e terá apenas as seguintes seções:

a) Objetivo do plano de testes;
b) Escopo dos testes;
c) Aspectos que serão testados;
d) Aspectos que não serão testados;
e) Critérios de aceitação da versão do software (elaborado pelo cliente).

Caso seja necessário, informar mais algum detalhe daquela versão através das demais seções do plano de testes geral.

O quadro 2 apresenta os papéis envolvidos, entradas e saídas da atividade de elaboração do plano de testes.

Quadro 2 – Detalhamento da atividade Elaborar Plano de Testes

Papéis
• Gerente de projetos; • Líder de testes.
Entradas
• Projeto do Jira; • Especificação de requisitos (opcional).
Saídas
• Plano de testes geral (opcional); • Plano de testes específico para a versão.

Fonte: Elaboração do autor, 2014.

5.4.1.3 Identificar casos de testes

Nessa atividade, o objetivo é identificar as funcionalidades principais do software e mapear casos de testes necessários para cobrir essas funcionalidades. Os casos de testes ainda não são detalhados nessa etapa.

Com essa informação é possível estimar o esforço necessário para detalhar os casos de teste e executar a bateria de testes.

Os passos para concluir a atividade, do ponto de vista do testador responsável por elaborar o plano de testes, são os seguintes:

a) Reunir-se com usuário final ou cliente do projeto para entendimento geral do negócio (sempre que possível, o testador deve acompanhar as reuniões de levantamento de requisitos);

b) Reunir-se com analista de sistema responsável pelo projeto para repasse dos requisitos e regras de negócio;

c) Definir funcionalidades do software que são essenciais para a operação do sistema, isto é, aquelas que se falharem vão impactar o andamento das atividades do usuário final;

d) Estudar documentação técnica disponível;

e) Identificar os casos de testes necessários para exercitar as funcionalidades consideradas essenciais para a operação do software. Como critério para identificar os casos de testes procure garantir cobertura para:

- Fluxos principais de cada caso de uso;

- Regras de negócio;
- Principais comandos disponíveis em cada interface;
- Requisitos funcionais considerados essenciais para o usuário.

f) Identificar a amarração dos casos de testes, a interdependência entre eles:

- Casos de testes interdependentes são aqueles que executados em sequência exercitam por completo um fluxo de negócio na aplicação;
- Em outras palavras, simulam o trabalho completo do usuário ou das diversas áreas que utilizarão o software;
- O uso desse tipo de técnica gera economia de tempo especialmente na preparação da massa de testes, já que um caso de teste pode gerar dados para o caso de teste seguinte.

O quadro 3 apresenta os papéis envolvidos, entradas e saídas da atividade de identificação dos casos de testes.

Quadro 3 – Detalhamento da atividade Identificar Casos de Testes.

Papéis
• Testador. • Líder de testes;
Entradas
• Especificação de requisitos; • Especificação de casos de uso; • Arquitetura do software (opcional); • Fluxos de negócio (e.g. descritos em BPMN).
Saídas
• Identificação de casos de teste (sem detalhamento) no Testlink; • Estimativa de tamanho da bateria de testes de regressão. Quantos e quais casos de testes deverão ser retestados naquela *release*. • Estimativa de tempo para detalhamento dos casos de teste. Na ausência de uma base histórica, sugere-se iniciar a estimativa com um caso de teste especificado por hora. • Estimativa de tempo para execução da bateria de testes. Na ausência de

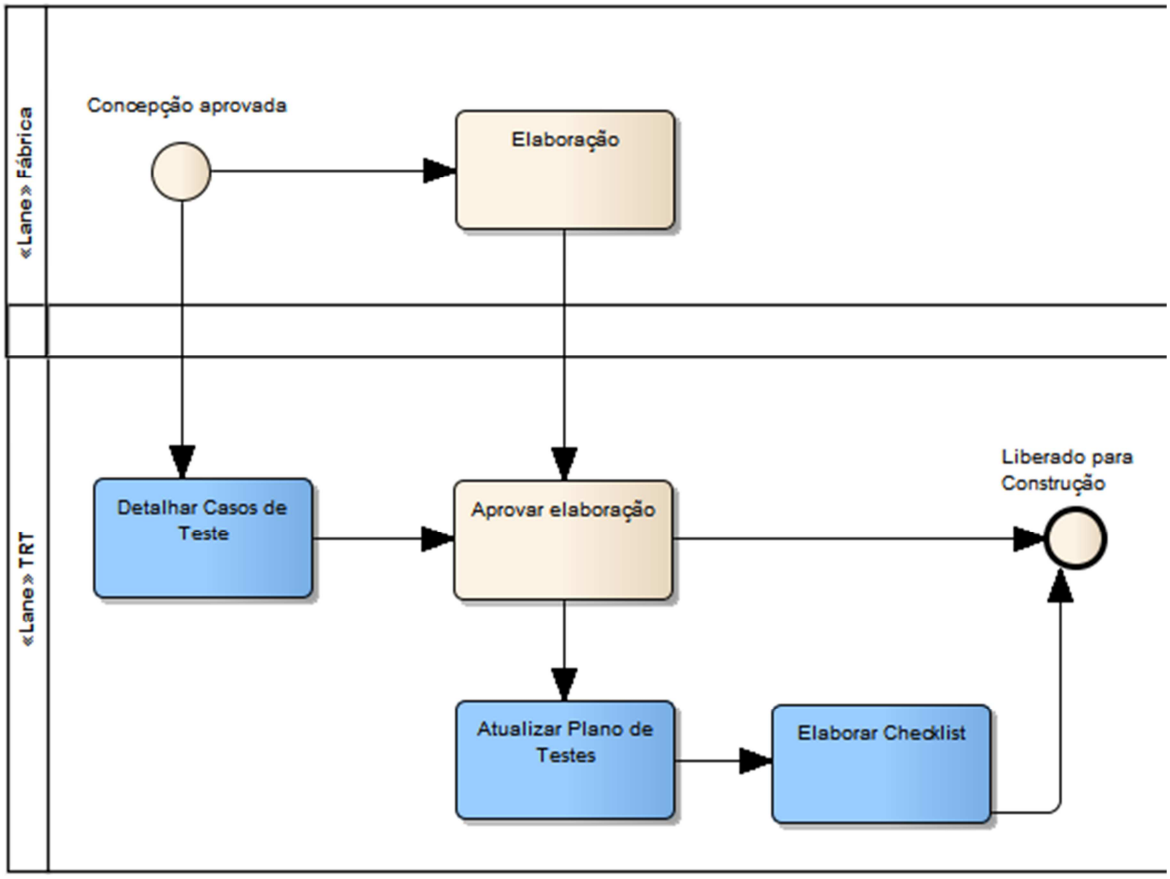
uma base histórica, sugere-se iniciar a estimativa com quatro casos de teste executados por hora.

Fonte: Elaboração do autor, 2014.

5.4.2 Fase de elaboração

O foco na fase de elaboração é a conclusão da especificação dos casos de teste. Nessa etapa, devem ser aplicadas as técnicas de modelagem de casos de teste para derivar as entradas necessárias para validação e verificação dos requisitos. A figura 7 ilustra as alterações.

Figura 7 - Alterações na fase de Elaboração.



Fonte: Elaboração do autor, 2014.

5.4.2.1 Detalhar casos de testes

O objetivo dessa atividade é detalhar os casos de testes identificados na concepção. Cada teste deve receber:

a) Pré-condições;

b) Valores de entrada;

c) Passos;

d) Resultados esperados.

Os casos de teste deverão ser criados a partir dos casos de uso, protótipos de tela, requisitos funcionais e regras de negócio. Sugerem-se os seguintes passos para a elaboração dos casos de teste:

a) Definir o estado que o sistema deve estar antes de iniciar os testes, ou seja, os pré-requisitos para o caso de teste;

b) Definir quais testes são necessários executar antes (ou estarem funcionando);

c) Definir quais dados são necessários ter previamente cadastrados no sistema (para serem criados através de um banco de dados controlado);

d) É recomendado, nesse caso, o uso das técnicas de modelagem de casos de teste apresentadas na seção 3.3.1 (partições de equivalência, análise de valor limite e grafos de causa e efeito) para maximizar a cobertura dos casos de testes;

e) Definir o passo a passo;

f) Definir os resultados esperados;

g) É boa prática criar tags para os dados utilizados no caso de teste, declaradas na suíte de teste, de forma a facilitar a alteração dos valores, caso necessário no futuro;

h) À medida que o teste é detalhado ele deve ser executado para validação (isso depende da disponibilidade do software).

O quadro 4 apresenta os papéis envolvidos, entradas e saídas da atividade de detalhamento dos casos de testes.

Quadro 4 – Detalhamento da atividade Detalhar Casos de Testes.

Papéis
• Testador. o Nessa atividade específica, o papel pode ser desempenhado por alguém da equipe do tribunal ou até mesmo por alguém da equipe do fornecedor, com supervisão de um servidor do TRT-SC (depende do contrato com a empresa terceirizada); o No caso de ser uma entrega do fornecedor, é importante cobrar que

todos os artefatos esperados sejam entregues.
Entradas
• Árvore de casos de teste (produzida na primeira etapa); • Especificação de requisitos; • Especificação de casos de uso; • Arquitetura do software (opcional); • Fluxos de negócio (e.g. descritos em BPMN).
Saídas
• Planilha de particionamento de equivalência (uma por caso de uso); • Especificação completa dos casos de teste.

Fonte: Elaboração do autor, 2014.

5.4.2.2 Atualizar plano de testes

Essa atividade tem como alvo a atualização do plano de testes conforme o detalhamento das informações coletadas durante a etapa de elaboração. Pode incluir atualização de qualquer item, bem como alguma reescrita na modelagem dos testes, considerando alterações em requisitos durante a elaboração.

Apesar de estar explicitamente inserida ao final da fase de elaboração, esta atividade deve ser constantemente executada, a fim de que o plano de testes mantenha-se sempre atualizado e represente a real situação do projeto dos testes.

O quadro 5 apresenta os papéis envolvidos, entradas e saídas da atividade de atualização do plano de testes.

Quadro 5 – Detalhamento da atividade Atualizar Plano de Testes.

Papéis
• Líder de testes; • Testador.
Entradas
• Especificação do software (requisitos, casos de uso, arquitetura etc.); • Documento de elaboração (projeto físico); • Plano de testes.
Saídas

• Plano de testes atualizado.

Fonte: Elaboração do autor, 2014.

5.4.2.3 Elaborar checklist

Baseado nas informações atualizadas do plano de testes, deve-se elaborar os checklists para cada caso de uso que será implementado ou alterado nessa iteração.

Os checklists aqui sugeridos servem para validar itens geralmente não funcionais e que dificilmente são testados nos casos de teste. Exemplos desses itens são:

a) Tamanho mínimo/máximo dos campos (caso o campo seja crítico para a execução da tarefa no sistema, deve-se incluir um caso de teste usando técnicas como análise de valor limite);
b) Máscaras;
c) Ortografia e gramática;
d) Alinhamento dos campos na tela;
e) Campos obrigatórios;
f) Tipos de dados (por exemplo, se um campo só aceita número, forçar o uso de letras);
g) Acessibilidade;
h) Funcionamento na resolução mínima requisitada;
i) Outros itens de usabilidade (de acordo com manual da empresa).

O uso do checklist é proposto ao responsável pelo caso de uso na fábrica de software, como um teste preliminar para que erros pequenos não sejam passados adiante no processo de desenvolvimento. Deve ser criado um checklist para cada caso de uso.

O quadro 6 apresenta os papéis envolvidos, entradas e saídas da atividade de elaboração do checklist.

Quadro 6 – Detalhamento da atividade Elaborar Checklist.

Papéis
• Analista de requisitos;

• Testador.
Entradas
• Especificação do software (requisitos, casos de uso, protótipos de tela, etc.); • Plano de testes.
Saídas
• Checklist por caso de teste.

Fonte: Elaboração do autor, 2014.

5.4.3 Fase de construção

A ênfase na fase de construção foi na execução dos casos de teste na versão do software entregue pela fábrica de software, bem como o relatório de defeitos a fim de medir a qualidade do sistema. As alterações são ilustradas na figura 8.

5.4.3.1 Executar testes do checklist

O checklist deverá ser executado pelo analista responsável pelos casos de uso na fábrica de software. As evidências de execução devem ser documentadas para que não seja aceita a entrega do software com violações.

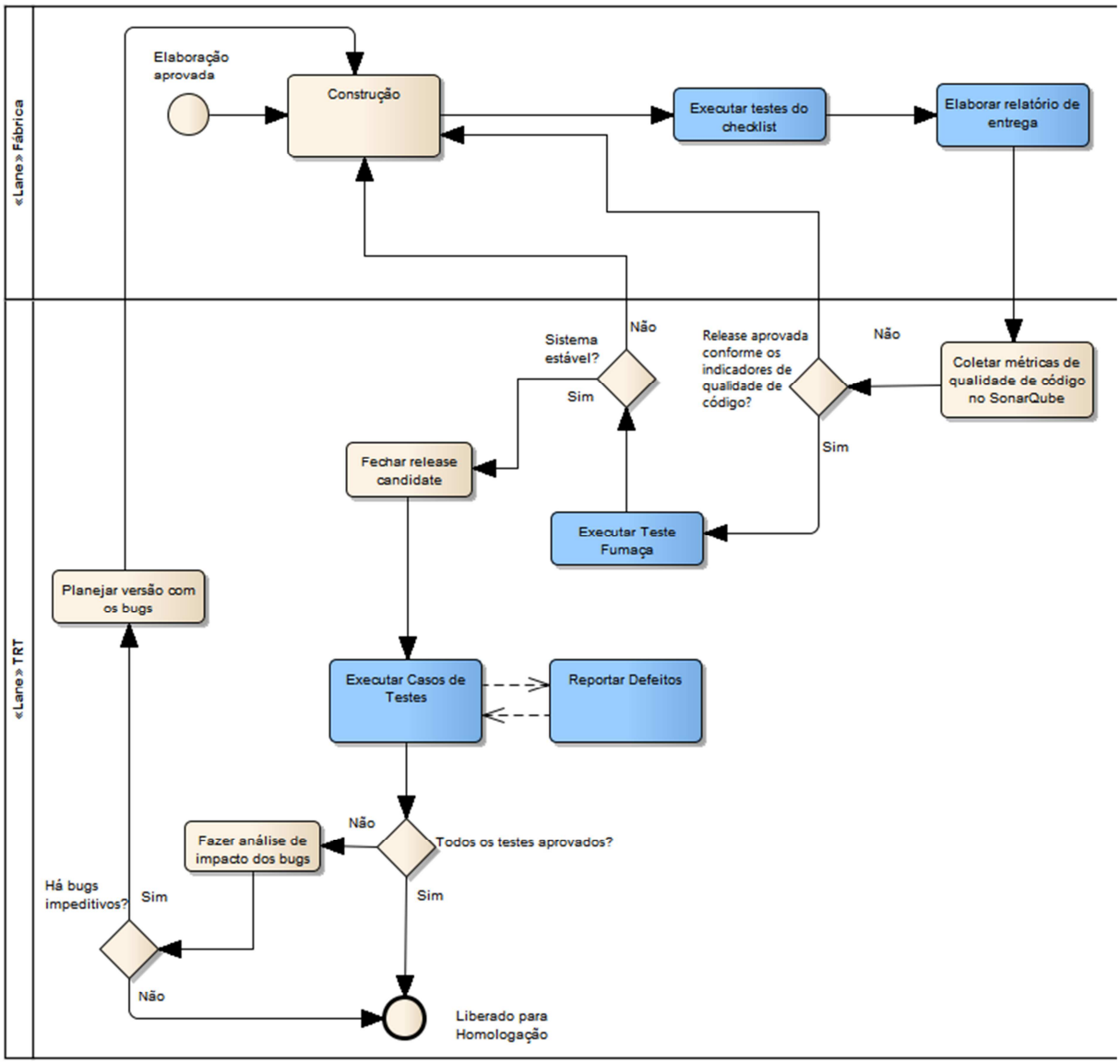
O quadro 7 apresenta os papéis envolvidos, entradas e saídas da atividade de execução do checklist.

Quadro 7– Detalhamento da atividade Executar testes do checklist.

Papéis
• Analista da fábrica.
Entradas
• Checklist por caso de uso.
Saídas
• Para cada caso de uso entregue um checklist devidamente preenchido mostrando sucesso na conferência de cada item.

Fonte: Elaboração do autor, 2014.

Figura 8 - Alterações na fase de Construção



Fonte: Elaboração do autor, 2014.

5.4.3.2 Elaborar relatório de entrega

No relatório de entrega, a fábrica de software deverá enviar os seguintes documentos:

a) Screenshots de cada teste realizado;

b) Dump do banco de dados utilizado para os testes;

c) Evidências da execução do checklist (para cada caso de uso);

d) Relatório de incidentes (Jira).

O quadro 8 apresenta os papéis envolvidos, entradas e saídas da atividade de elaboração do relatório de entrega da iteração.

Quadro 8 – Detalhamento da atividade Elaborar relatório de entrega.

Papéis
• Analista da fábrica; • Gerente da fábrica.
Entradas
• Checklist por caso de uso; • Aplicação rodando local (na fábrica).
Saídas
• Relatório de entrega.

Fonte: Elaboração do autor, 2014.

5.4.3.3 Executar testes de fumaça

Teste de fumaça é aquele teste feito com o objetivo de verificar se a aplicação se encontra estável para iniciar a execução dos casos de teste.

Sugere-se executar os testes de fumaça a partir dos requisitos e casos de uso do EA, utilizando a abordagem apresentada na seção 5.3.4 que descreve o ciclo de vida dos requisitos.

Qualquer inconsistência identificada deve ser encaminhada para a fábrica corrigir, pois se entende que, caso haja algum defeito nessa etapa do processo, os demais testes poderão estar completamente invalidados.

Estes testes são candidatos a serem automatizados no futuro.

O quadro 9 apresenta os papéis envolvidos, entradas e saídas da atividade de execução dos testes de fumaça.

Quadro 9 – Detalhamento da atividade Executar testes de fumaça.

Papéis
• Testador.
Entradas
• Plano de testes; • Casos de uso.
Saídas
• Aceite ou recusa da entrega do software.

Fonte: Elaboração do autor, 2014.

5.4.3.4 Executar casos de testes

Executar os casos de teste e registrar as ocorrências que encontrar. O enfoque desta atividade está nos artefatos de análise.

O quadro 10 apresenta os papéis envolvidos, entradas e saídas da atividade de execução dos casos de teste.

Quadro 10 – Detalhamento da atividade Executar casos de teste.

Papéis
• Testador.
Entradas
• Plano de testes; • Lista de casos de teste para execução.
Saídas
• Aceite ou recusa da entrega do software.

Fonte: Elaboração do autor, 2014.

5.4.3.5 Reportar Defeitos

Atividade importante de ser formalizada no processo, pois dá o embasamento para a análise da qualidade do software produzido. O registro correto dos erros encontrados permitirá também a produção de métricas de qualidade.

As seguintes informações devem ser reportadas:

a) Versão: deve ser indicada a versão do software em que o erro foi encontrado. Isto é feito para ter o controle de quando foram corrigidos ou, principalmente, se foram deferidos;

b) Fase: é importante saber em qual fase do projeto o defeito foi descoberto;

c) Caso de uso: em que parte do sistema aquele defeito foi encontrado, para que possa ser direcionado para os responsáveis;

d) *Login*/senha: esse é o primeiro passo para reproduzir um defeito. Usuários diferentes possuem, às vezes, diferentes permissões no sistema;

e) Severidade: o responsável por registrar o defeito define o quanto o sistema é afetado (por exemplo, catastrófico, alto impacto, baixo impacto e cosmético);

f) Prioridade: o quão rápido o defeito deve ser corrigido pelos responsáveis;

g) URL: em caso dos testes serem feitos em diferentes ambientes;

h) *Screenshot*: identificação visual da tela com erro;

i) Passos: é importante dizer quais passos foram executados e quais entradas foram digitadas em cada tela para que aquele defeito ocorra;

j) Navegador: diferentes navegadores possuem diferentes engines. Os browsers que o sistema deve rodar deverão estar definidos no plano de testes.

Procedimentos e orientações para o testador:

a) Encontre o defeito e reproduza-o;

b) Reporte a reprodução fiel do defeito encontrado (inclusive informando os dados utilizados);

c) Para um sistema, um erro só é um erro quando pode ser reproduzido.

O quadro 11 apresenta os papéis envolvidos, entradas e saídas da atividade de registro dos defeitos.

Quadro 11 – Detalhamento da atividade Reportar defeitos.

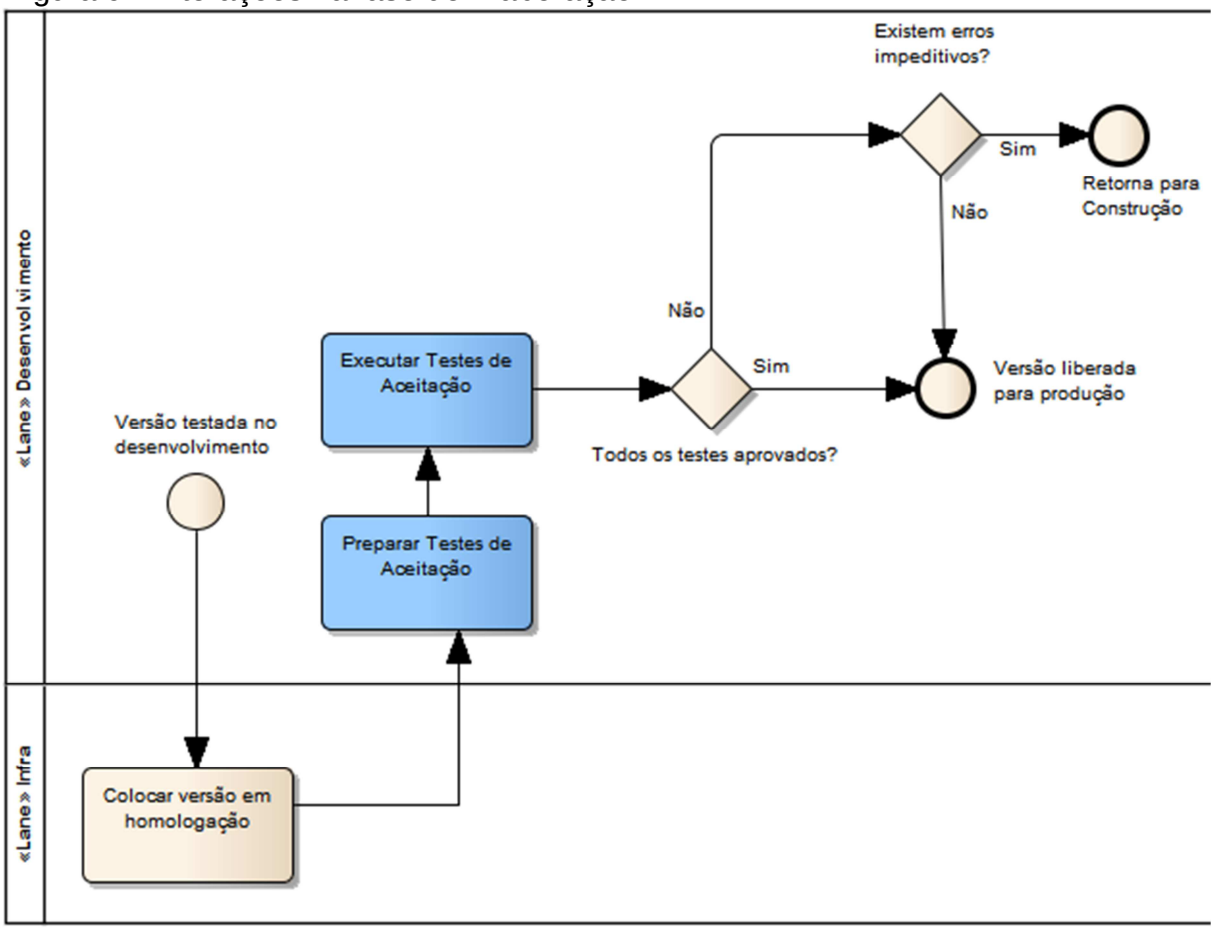
Papéis
• Testador.
Entradas
• Relatório de execução dos casos de teste.
Saídas
• Lista dos erros cadastrados (Jira).

Fonte: Elaboração do autor, 2014.

5.4.4 Fase de homologação

Na fase de homologação foi formalizada e padronizada a execução de testes de aceitação, conforme exibido na figura 9.

Figura 9 - Alterações na fase de Elaboração



Fonte: Elaboração do autor, 2014.

5.4.4.1 Preparar testes de aceitação

Esta tarefa tem como objetivo padronizar o teste de aceitação dentro da equipe de desenvolvimento. O teste de aceitação deve basear-se nos requisitos solicitados pelo cliente, mas deve demonstrar que o sistema realmente faz aquilo que o cliente deseja, ou seja, o roteiro de testes de aceitação deve estar alinhado com os anseios do cliente e não apenas com um documento.

O quadro 12 apresenta os papéis envolvidos, entradas e saídas da atividade de preparação dos testes de aceitação.

Quadro 12 – Detalhamento da atividade Preparar testes de aceitação.

Papéis
• Testador;

• Analista de requisitos.
Entradas
• Plano de testes, com orientação geral sobre os testes de aceitação: o Quais funcionalidades devem ser testadas; o Quais os critérios de aceitação dessa versão; o Quais os envolvidos na homologação.
Saídas
• Especificação dos testes de aceitação.

Fonte: Elaboração do autor, 2014.

5.4.4.2 Executar testes de aceitação

O objetivo dessa tarefa é garantir o aceite do sistema de maneira formal. O enfoque desta atividade está nos requisitos funcionais e não na busca de erros. Os testes de aceitação podem tanto ser executados pela equipe técnica do projeto como pela equipe de homologação (formada por cliente, usuários, patrocinadores, etc.). Caso sejam executados pela equipe técnica, é necessário que seja feito de forma que a equipe de homologação acompanhe e entenda o que está sendo testado.

Todos os erros encontrados nessa fase devem ser registrados conforme a atividade de “Reportar defeitos” da fase de construção.

O quadro 13 apresenta os papéis envolvidos, entradas e saídas da atividade de execução dos testes de aceitação.

Quadro 13 – Detalhamento da atividade Executar testes de aceitação.

Papéis
• Testador; • Analista de requisitos; • Equipe de homologação.
Entradas
• Casos de teste de aceitação.
Saídas
• Termo de aceite da versão.

Fonte: Elaboração do autor, 2014.

5.5 MÉTRICAS

A fim de avaliar se as alterações no processo de desenvolvimento foram efetivas, sugere-se a adoção das seguintes métricas:

Eficiência de remoção de defeitos (ver seção 3.4.4): relação de erros encontrados no teste de sistema (durante a fase de construção) e o total de erros encontrados nas fases posteriores. Essa métrica ilustra a eficácia do processo de testes.

Eficiência de remoção de defeitos por unidade de tamanho do software: medida relativa e que permite a comparação de diferentes sistemas. É possível extraí-la já que o contrato com a fábrica de software exige que o pacote seja mensurado em pontos por caso de uso.

Taxa de sucesso (ver seção 3.4.4): relação entre os casos de teste que passaram (sem erros) e o total de casos de teste. Esta métrica deve ser usada em projetos de melhorias de sistemas e comparada com os valores passados. Caso sejam utilizados testes de regressão e a medida tenha um incremento se comparado ao histórico, pode indicar que os testes de regressão não estão exercitando corretamente o sistema em busca de erros.

Além do objetivo de medição e aferição da qualidade, a adoção das métricas poderá embasar um trabalho futuro de inclusão de acordo de nível de serviço a ser incluído no contrato de prestação de serviço entre o TRT-SC e a empresa contratada para a construção dos sistemas.

6 CONCLUSÕES E TRABALHOS FUTUROS

Foi apresentada a proposta de melhoria no processo de desenvolvimento de software para web, no departamento de informática do TRT-SC, baseada na adoção de atividades formais, efetivamente implementando uma metodologia de testes funcionais. A pesquisa identificou melhores práticas e padrões que podem ser adotados de forma a gerenciar corretamente o processo de teste de software, modelar e executar os casos de teste, reportar os incidentes encontrados e, por fim, medir a qualidade do trabalho realizado.

O objetivo da proposta foi de que as alterações no processo pudessem ser realizadas de forma suave a fim de que houvesse pouco impacto no desenvolvimento atual. No entanto, sabe-se que o teste de software é custoso no que diz respeito a recursos humanos e cronograma e, para que a adoção do processo de teste dê-se de forma plena, é necessário que a administração entenda a necessidade de planejar o projeto de desenvolvimento incluindo nos custos, cronograma e recursos as atividades relativas ao teste. Como contrapartida, o teste bem realizado trará tranquilidade à equipe de desenvolvimento e, o mais importante, poderá garantir a qualidade do produto final e a satisfação dos usuários.

À medida que o processo de teste for amadurecendo, sugere-se, como trabalhos futuros, a inclusão de um acordo de nível de serviços baseado nas métricas sugeridas, para auxiliar na fiscalização do contrato com a fábrica de software.

Além disso, com a possibilidade de aumento do quadro efetivo da equipe de desenvolvimento, torna-se viável a automação gradativa dos testes funcionais resultantes desse trabalho, como parte da suíte de testes de regressão, de forma a executá-los na integração contínua dos sistemas. Conforme já demonstrado na revisão de literatura, os testes de regressão são sempre fortes candidatos à automação, pois a execução repetitiva de casos de testes costuma transformar-se em negligência (mesmo que involuntária) por parte do testador.

REFERÊNCIAS

BECK, Kent. **Extreme programming explained**: embrace change. Addison-Wesley Professional, 1999.

BELL, Douglas. **Software engineering for students**: a programming approach. 4. ed. Pearson Education Limited, 2005.

BOEHM, W. Barry. **Seven basic principles of software engineering**. Journal of Systems and Software, v. 3, n. 1, p. 3-24, mar. 1983.

BRASIL. Constituição (1988). **Constituição da República Federativa do Brasil**. Brasília, DF:Senado Federal, 1988.

BROOKS Jr., Frederick P. **The mythical man-month** (anniversary ed.). Addison-Wesley Longman, 1995.

GAROUSI, Vahid; ZHI, Junji. **A survey of software testing practices in Canada**. The Journal of Systems and Software, n. 86, p. 1354-1376, 2013.

GRAHAM, Dorothy. et al. **Foundations of software testing:** ISTQB certification. Thomson Press (India) Ltd, 2007.

IBARRA, Gustavo B.; VILAIN, Patrícia. **Software Estimation Based on Use Case Size**. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 2010, Salvador. Anais... Salvador: IEEE, 2010. p. 178-187.

INFORMATION WEEK. **Quality first**. Disponível em: <http://www.informationweek.com/quality-first/d/d-id/1015089>. Acesso em: 07 dez. 2014.

ISTQB. **Certified tester:** foundation level syllabus 2011br. 2011.

KRUCHTEN, Philippe. **The rational unified process**: an introduction. 3. ed. Addison Wesley, 2003.

LEE, J.; KANG, S.; LEE, D. **Survey on software testing practices**. IET Software, v. 6, n. 3, p. 275-282, 2012.

LOURIDAS, Panos. **Test management**. IEEE Software, v. 28, n. 5, p. 86-91, set./out. 2011.

MCCONNELL, Steve. **Code complete**. 2 ed. Microsoft Press, 2004.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. **The art of software testing**. 3. ed. Wiley, 2011.

NIST. **The economic impacts of inadequate infrastructure for software testing**. Gaithersburg, 2002. Planning report 02-3.

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

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software**. 3. ed. Rio de Janeiro: Alta Books, 2013.

ROOK, Paul. **Controlling software projects**. Software Engineering Journal, v. 1, n. 1, p. 7-16, jan. 1986.

SOMMERVILE, Ian. **Engenharia de software**. 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

STRATE, Jonathan; LAPLANTE, Phillip. **A literature review of research in software defect reporting**. IEEE Transactions on Reliability, v. 62, n. 2, jun. 2013.

THE STANDISH GROUP. **CHAOS manifesto 2013**. Disponível em: <http://www.standishgroup.com>. Acesso em: 07 dez. 2014.