ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Mário Almeida Rodrigues

Proposta de processo de estimativa e desenvolvimento de software utilizando Desenvolvimento Orientado a Testes (TDD) e Análise de Ponto de Teste (APT)

Palmas - TO
2013

ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Mário Almeida Rodrigues

Proposta de processo de estimativa e desenvolvimento de software utilizando Desenvolvimento Orientado a Testes (TDD) e Análise de Ponto de Teste (APT)

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientadora: Prof. M.Sc. Cristina D'Ornellas Filipakis Souza.

Palmas - TO
2013

ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Mário Almeida Rodrigues

Proposta de processo de estimativa e desenvolvimento de software utilizando Desenvolvimento Orientado a Testes (TDD) e Análise de Ponto de Teste (APT)

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientadora: Prof. M.Sc. Cristina D'Ornellas Filipakis Souza.

Aprovada em 05 de Dezembro de 2013.

BANCA EXAMINADORA

_____

Prof. M.Sc. Cristina D'Ornellas Filipakis Souza

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Parcilene Fernandes de Brito

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas

Palmas - TO
2013

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

De acordo com dados de estudos realizados pelo *The Standishi Group* (*online*, 2009), durante anos os projetos de software apontavam resultados insatisfatórios e os índices de sucesso eram inferiores aos de fracassos totais e fracassos parciais. Para minimizar este cenário, as empresas desenvolvedoras de software procuraram cada vez mais mudar este cenário. Para tanto, técnicas e metodologias foram surgindo a fim de aumentar a qualidade e consequentemente a confiabilidade nos sistemas. Um dos pontos indicados pelo estudo da *The Standishi Group* (*online*, 2009) é que as falhas nos produtos custavam grandes quantidades de dinheiro para as empresas norte americanas. E, para mudança desse cenário, novas técnicas e metodologias são utilizadas no desenvolvimento de software, que priorizam a atividade de teste, e focam na redução de falhas nos sistemas.

Uma das metodologias utilizadas para redução de falhas é o TDD, que consiste em uma metodologia ágil que visa a elaboração dos testes antes mesmo do desenvolvimento das funcionalidades. Ela foi criada tendo em vista a economia e principalmente o aumento da qualidade no desenvolvimento de software. Segundo Borges (2006, p.2), "no TDD, desenvolvedores usam testes para guiar o projeto do sistema durante o desenvolvimento".

Nos estudos realizados em livros, artigos, monografias e teses sobre o TDD, não foi possível identificar uma proposta de uso da metodologia de desenvolvimento orientado a testes em conjunto a uma técnica de estimativas de testes. Por ser uma metodologia orientada a testes, o esforço pode ser melhor calculado com o auxílio de um processo para realizar estimativas de testes.

O processo de estimativa de teste tende a ser complexo, principalmente por ser uma atividade que pode determinar o sucesso ou o fracasso do projeto. Estimar o esforço para desenvolver um sistema é considerado por muitos uma atividade muito importante, pois o esforço definirá os recursos humanos e materiais que serão alocados em um projeto. A subestimativa ou superestimativa destes recursos podem indicar o nível de competitividade da empresa e do projeto no mercado. Boas estimativas de software são importantes para torná-lo viável, tanto para a empresa quanto para o

cliente. Como subsídio para tal estimativa, existem algumas técnicas na literatura, entre elas, pode-se destacar a Análise de Pontos de Teste (APT).

A APT é utilizada para medir o esforço, para definir, desenvolver e executar testes, fundamentando-se na complexidade do desenvolvimento do software, levando em consideração o tamanho funcional determinado pelos pontos de função, a estratégia de teste e a produtividade (VEENENDAAL e DEKKERS, 1999).

Este trabalho propõe que a técnica APT seja utilizada em conjunto a metodologia TDD, com o intuito de aumentar a qualidade dos testes construídos em um processo de desenvolvimento de software. Para tanto, será necessário que sejam inseridas fases no desenvolvimento na metodologia proposta pelo TDD, como o levantamento de requisitos, a concepção do projeto e as definições de prioridades. O levantamento de requisitos e a concepção do projeto são fases nas quais a equipe deverá produzir artefatos, como diagramas, sendo que estes artefatos irão fornecer subsídios para que a APT seja realizada. Portanto, após realizar o levantamento de requisitos e a concepção do projeto as estimativas serão feitas utilizando a APT.

As três fases citadas acima podem influenciar diretamente na escolha do cliente em relação às prioridades das funcionalidades a serem desenvolvidas. Com as funcionalidades ordenadas de forma prioritária pelo cliente, os testes são planejados de acordo com a descrição da funcionalidade. Com o planejamento dos testes se inicia a implementação, com a equipe escrevendo os testes, testando a funcionalidade, implementando a funcionalidade, realizando novamente o teste e refatorando o código. A próxima funcionalidade só será implementada após a anterior passar nos testes e, se necessário, ser refatorada.

Este trabalho trouxe como objetivo geral a construção de uma proposta de um processo de desenvolvimento de software onde envolveu a utilização em conjunto do processo de desenvolvimento ágil *Test Driven Development* – TDD e da técnica de estimativa Análise de Pontos de Teste – APT, sendo que, para que o objetivo geral seja alcançado elaborou-se um referencial teórico sobre estimativas, métricas, teste de software, *Test Driven Development* e Análise de Ponto de Teste; criou-se o fluxo do processo em conjunto da metodologia TDD a técnica APT, especificando-se papéis, atividades e artefatos; e, por fim, propôs-se a aplicação da técnica de estimativa de

esforço Análise de Pontos de Testes em projetos desenvolvidos com *Test Driven Development* – TDD.

1.1 JUSTIFICATIVA

Grande parte do custo e do tempo gastos no processo de desenvolvimento de um software são gerados pelas falhas encontradas. Mesmo não sendo possível desenvolver um software sem falhas, é possível diminuí-las de forma considerável utilizando testes elaborados conforme técnicas com eficiência comprovada, que permitam identificar e remover falhas o quanto antes e de forma eficaz.

Uma pesquisa do *The Standish Group*, realizada em 1995, aponta que já eram gastos mais de 250 bilhões de dólares por ano em desenvolvimento de aplicações, sendo cerca de 175 mil projetos. Destes, 52,7% dos projetos custaram 189% a mais que as suas estimativas de custo originais, e 222% a mais na estimativa de tempo original. Com base nos resultados obtidos na pesquisa, o *The Standish Group* estimou que as empresas americanas gastariam 81 bilhões de dólares em projetos cancelados.

Uma pesquisa publicada em 2009 pelo *The Standish Group* aponta que 44% dos projetos ainda apresentam perdas em relação a suas estimativas inicias. Em contrapartida, o número de projetos bem sucedidos cresceu de 16% em 2006, para 32% em 2009.

Uma pesquisa realizada em 20 empresas sugeriu que 75% das falhas de software aparecem após o primeiro ano de lançamento (WESTLAND, et. al. 2003 apud INWOOD 1993, p.2). Os erros que são descobertos após o lançamento podem custar muito mais que o processo de teste no desenvolvimento do software. Em 2003, o nordeste dos Estados Unidos sofreu um “apagão” de informações devido a uma falha no GE’s XA/21, o sistema de alerta de gestão. Os custos estimados a esta falha estavam entre 7 e 10 bilhões de dólares (SMITH, 2009).

Muitos testes devem ser realizados em um software para que grandes falhas não tragam prejuízos. Sendo assim, a utilização do TDD que sugere o desenvolvimento de testes antes mesmo do desenvolvimento das funcionalidades, pode auxiliar na redução de falhas de software. Além das falhas, o não cumprimento dos prazos gera custos no desenvolvimento do software. A Análise de Pontos de Testes é uma técnica que pode auxiliar no processo de estimativa de esforço dos testes a serem realizados.

Este trabalho apresenta a utilização da Análise de Pontos de Testes como auxílio nas estimativas iniciais do esforço de testes do software, no intuito de se obter estimativas de tempo e custo mais precisas, com menor margem de erros, reduzindo custos e cumprindo prazos sem perder a qualidade do software.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentados os tópicos que foram utilizados de base bibliográfica para a elaboração da proposta da utilização da Análise de Pontos de Teste como técnica de estimativa para testes de software que utilizam como metodologia o desenvolvimento orientado a teste (TDD), sendo os temas:

a) Estimativas de Software: esta seção apresenta a importância das estimativas de software, além de apresentar duas técnicas para realizar as estimativas de esforço e custo de software. Ainda nesta seção, é apresentada a importância das estimativas em teste de software;
b) Testes de Software: nesta seção pode ser encontrado o processo da atividade de testes de software, a importância dos testes, além de apresentar os testes de verificação e validação;
c) Análise de Pontos de Testes (APT): nesta seção é apresentada a técnica de APT, com os fatores importantes que interferem diretamente na utilização da técnica, além de todos os processos para realizar o cálculo da estimativa utilizando a APT;
d) *Test Driven Development* (TDD): nesta seção são apresentados os processos e características do TDD, além do fluxo das atividades.

2.1 Estimativas de Software

Estimar prazos e custos faz parte da rotina no ramo da engenharia, sendo que na engenharia de software a estimativa é utilizada para compreender o cálculo de esforço, custo e prazo no desenvolvimento de um sistema (CUNHA 2012, p. 15). Contudo, estimativas de um projeto são difíceis de serem obtidas. Para Sommerville (2011, p. 442), “existem tantas incertezas que é impossível estimar com precisão os custos de desenvolvimento de sistema durante os estágios iniciais de um projeto”. Desta forma, para realizar estimativas podem ser empregadas dois tipos de técnicas que podem ser aplicadas para realizar as estimativas de esforço e custo de software, as baseadas em experiências anteriores e a modelagem algorítmica de custo.

Métodos baseados em experiências anteriores são métodos que utilizam dados de projetos antigos para realizar as estimativas para o projeto corrente (BARCELLOS *online*, p.4). Para realizar este tipo de estimativa, a empresa deve ter uma base histórica

de projetos, na qual consiga fornecer dados que possam ser utilizados como ponto de apoio para novas estimativas. O segundo tipo é a modelagem algorítmica de custo na qual, conforme Sommerville (2011, p. 443), é utilizada uma fórmula matemática para antecipar os custos do projeto com base no tamanho do sistema expressado em pontos de função ou de aplicação, tipo de software que está sendo desenvolvido e outros fatores de equipe, processo e produto. Ainda sobre modelagem algorítmica de custo ou também modelos empíricos, para Falbo (2005, p. 14), este modelo pode ser usado como ponto de partida para a organização que ainda não tem dados históricos.

Uma boa estimativa pode determinar o sucesso do projeto, pois não basta que o produto atenda aos requisitos desejados, também deve ser produzido dentro dos prazos e custos estabelecidos pelo cliente. A capacidade de realizar estimativas da equipe é de suma importância, pois estimativas não devem originar perdas ao final do projeto, por outro lado, não pode ser uma estimativa muito superior ao necessário para o desenvolvimento, pois, levará a um planejamento de recursos e tempo não otimizados.

2.1.1 Estimativas de Teste

Dentro do processo de estimativa de software, pode-se analisar a estimativa de teste, na qual a quantidade de testes para satisfazer uma funcionalidade é prevista. Além da quantidade de testes, outro fator determinante deve ser observado: o tempo que será gasto para realizar um determinado teste. Para Cicilia (2012, *online*) a estimativa específica para teste é importante para mensurar o esforço que a atividade de teste demanda. Ainda segundo Cicilia (2012, *online*), as equipes que realizam a atividade de teste utilizam uma base histórica ou a experiência do testador para estimar o esforço na execução de casos de testes. Porém, este tipo de estimativa pode ser imprecisa devido aos muitos fatores subjetivos que influenciam na estimativa.

Os fatores citados podem afetar o resultado da estimativa e causar problemas de cronograma. Contudo, existem técnicas que estão sendo avaliadas e melhoradas para proporcionar uma estimativa mais objetiva, sendo uma delas a Análise de Pontos de Testes (APT), proposta por Veenendaal e Dekkers. A APT é baseada em três elementos que determinam a medição, o tamanho do sistema, a estratégia de teste e a produtividade da esquipe. A APT será abordada na seção 2.5 deste trabalho.

Na atividade de teste é essencial que o conjunto de testes utilizados tenha uma abrangência de todo o domínio de dados de entrada para que os aspectos funcionais e operacionais do produto sejam avaliados (MALDONADO, *online*). Portanto, os testes devem satisfazer todos os pontos do sistema aumentando a sua confiabilidade.

Este trabalho envolveu a utilização da metodologia TDD, portanto, as estimativas de testes foram diretamente influenciadas pelo tamanho e complexidade das funcionalidades a serem desenvolvidas.

2.2 Métricas

Para Sommerville (2011, p. 466), "uma métrica de software é uma característica de um sistema, documento de sistema ou processo de desenvolvimento que pode ser objetivamente medido". Exemplos de métricas são o tamanho de um software em linhas de código, a quantidade de requisitos, ou ainda os pontos de função do software. Desta forma, pode-se estabelecer o tamanho do processo ou produto em um projeto de software. As métricas podem ser de controle ou de previsão. As métricas de controle, de acordo com Sommerville (2011, p. 466), são associadas aos processos de software. Exemplo de métricas de controle são o esforço médio e o tempo gasto para reparar os defeitos descritos. Já as métricas de previsão ou também conhecidas como métricas de produto são associadas ao produto em si. Como exemplo de métricas de produto pode-se citar a complexidade ciclomática, que é a quantidade de caminhos de execução independentes a partir de um código fonte.

Medições podem ser definidas como a atribuição de dados quantitativos a um processo de software, tais como esforço para realizar certa atividade (SOMMERVILLE 2011, p. 497). A medição gera evidências sobre o processo permitindo que gerentes de projeto acompanhem o andamento do mesmo. Com isso, os gerentes podem ter uma visão geral do tamanho e dos prazos necessários para que o produto seja desenvolvido de forma eficiente e rápida sem deixar de lado os testes e a qualidade de software.

2.3 Teste de Software

Os testes de software podem ser encontrados na literatura desde 1950 (FRANZEN 2005, p.4). Porém, segundo Rios (2002, p. 1), existe uma falta de interesse

pela atividade. Este desinteresse se dá devido a atividade de teste ser considerada cansativa e muitas vezes altamente onerosa quando realizadas com o software pronto.

As correções de erro em produtos que já foram entregues ao cliente podem custar cem vezes mais do que se o erro fosse evitado na fase de teste (MYERS 1979, p.7). Para Barbosa (*online*, p. 2), "a atividade de teste consiste de uma análise dinâmica do produto e é uma atividade relevante para a identificação e eliminação de erros que persistem". O teste é destinado a mostrar que um programa faz o que é proposto fazer e descobrir os defeitos do programa antes do seu uso (SOMMERVILLE 2011, p. 144). Mesmo não sendo possível produzir um software sem defeitos, a atividade de teste pode evitar muitos deles. Porém, os testes devem ser bem projetados e executados para que bons resultados sejam obtidos.

Os testes de software incluem o planejamento, projeto de casos de testes, implementação, execução e avaliação dos resultados (SANCHES, 1996, *online*). O planejamento dos testes torna o processo mais confiável, pois a equipe sabe os resultados que o sistema deve apresentar ao ser submetido aos testes. Os resultados obtidos com os testes podem ser comparados aos resultados esperados no planejamento, e assim realizar a avaliação dos testes e do sistema.

Casos de testes consistem em um conjunto de possíveis entradas e saídas esperadas para elas (JUNIOR, p.2). As entradas são definidas de forma sistemática a fim de atingir saídas também definidas no planejamento. As entradas devem satisfazer as saídas, se não houver essa simetria entre as saídas definidas e as saídas reais, o sistema ou o teste deve ser revisto. Bons casos de testes são aqueles que têm alta probabilidade de revelar defeitos ainda não descobertos. Os testes podem ser executados durante todo o planejamento e desenvolvimento do projeto. Testes de verificação podem ser executados durante o período de engenharia do *software*, na qual são verificados os documentos produzidos se estão de acordo com o planejado, e na fase de desenvolvimento podem ser executados os testes de validação, que são aplicados no produto ou parte do produto já desenvolvido.

2.3.1 Testes de Verificação

De acordo com Bartié (2002, p. 19) pode existir um equívoco em algumas empresas sobre a atividade de teste durante o ciclo de desenvolvimento. Devido

questões históricas, os profissionais de tecnologia relacionam os testes como sendo uma atividade aplicada apenas a software. A qualidade do produto pode ser medida a partir de testes aplicados em documentações geradas em toda fase de engenharia de software. Este teste é conhecido como teste de verificação, é realizado evitando que dúvidas e assuntos "mal resolvidos" durante cada fase do projeto passem para a próxima etapa (BARTIÉ 2002, p. 37). Contudo, diversas ações devem ser feitas para garantir a qualidade das atividades e documentos produzidos durante as etapas do projeto, para Bartié (2002, p. 37) duas delas são a realização de revisões de documentos e auditorias de qualidade.

2.3.1.1 Revisões

Revisão é um processo de análise de determinados documentos realizados por uma ou mais pessoas (BARTIÉ 2002, p. 75). O objetivo da revisão é garantir que a documentação produzida durante o processo de desenvolvimento esteja de acordo com o planejado. Esta atividade requer profissionais com experiência e que tenham afinidade com os documentos e materiais que estão sendo revisados, para que os revisores possam confrontar o documento com as definições estabelecidas.

Para conduzir as sessões de revisão, os revisores podem se organizar para executar formas diferentes de acordo com o estágio do desenvolvimento do projeto. Na fase inicial da elaboração dos documentos pode-se usar a revisão isolada, que para Bartié (2002, p. 75), trata-se de uma verificação individual do material produzido. Essa revisão é executada por alguém diferente do autor com objetivo de revisar e identificar a maior quantidade de possíveis problemas.

Em sequência, com os documentos já finalizados, pode-se usar a revisão formal para validar os documentos juntamente com todos os grupos interessados. De acordo com Bartié (2002, p. 76), essa forma de revisão se baseia em reuniões com um grupo responsável por identificar falhas nos documentos gerados nas diversas etapas do desenvolvimento. A revisão formal conta com a presença do autor para que eventuais dúvidas levantadas pelos revisores sejam respondidas. Ao final da reunião é fundamental documentar tudo que foi discutido, os problemas detectados, suas correções e sugestões de melhoria, com este documento os autores poderão realizar as mudanças necessárias e submeter a uma nova revisão, sendo que a nova revisão terá o foco apenas nos pontos modificados.

Como parte posterior à revisão formal, na fase de divulgação pode-se usar as reuniões de acompanhamento, que para Bartié (2002, p. 77), têm por objetivo "garantir a leitura do documento por todas as pessoas-chave envolvidas". As reuniões de acompanhamento permitem que mais pessoas sejam envolvidas e o seu objetivo é tornar o documento familiar para todos participantes, assim, envolvendo mais pessoas no processo de desenvolvimento. Esta reunião não é tão eficaz na detecção de erros, isto ocorre pelo despreparo dos participantes e pelo fato do autor conduzir a dinâmica da reunião (BARTIÉ 2002, p. 78).

2.3.1.2 Auditorias

As auditorias de qualidade buscam principalmente avaliar se as equipes estão respeitando o processo de desenvolvimento. Bartié (2002, p. 78) afirma que,

> *"o principal objetivo das auditorias é avaliar se em determinado projeto as diversas equipes estão respeitando o processo de desenvolvimento se estão registrando os defeitos encontrados, se estão produzindo as atas de reuniões, se estão realizando as reuniões de revisões, se estão realizando as documentações obrigatórias, se estão envolvendo clientes e usuários do projeto."*

Os auditores devem buscar principalmente as "quebras de processos", ou seja, membros da equipe que realizem processos fora ou diferente da metodologia estabelecida no planejamento do projeto, sendo que essa atitude pode levar a uma divergência entre os resultados de cada equipe. Sabendo que parte da equipe respeita os processos da metodologia e outra parte desrespeita os processos, ao final, haverá uma divergência entre os artefatos produzidos causando uma confusão, além da falta de padrão que é fator diretamente ligado a qualidade. Para Bartié (2002, p. 79), apenas a equipe que representa o modelo de desenvolvimento poderá estabelecer as regras do processo de engenharia de software.

A auditoria deve garantir que a documentação e os processos de engenharia de software sigam o que foi proposto no planejamento e na metodologia, usando como parâmetros um conjunto mínimo de controles e documentação que deverá ser obrigatório. O conjunto mínimo de controle e documentação deve ser determinado pela equipe de processo de engenharia de software.

2.3.2 Testes de Validação

Os testes buscam mostrar erros ainda não descobertos, para tanto, existem diversas técnicas para se testar um software. Para Neto (*online*, p. 56), as técnicas de testes podem ser classificas de acordo com a origem das informações, estabelecendo assim os requisitos de testes. Sabendo que os testes contemplam diferentes aspectos do software, é necessário formar uma estratégia de teste na qual as técnicas se complementem, de forma a ter maior abrangência nos testes. Ainda de acordo com Neto (*online*, p. 57), existem duas técnicas de testes: a funcional e a estrutural.

Testes estruturais ou de caixa-branca, são um ou mais processos projetados para validar se o código desenvolvido realiza o que foi realmente projetado para fazer, pois sistemas devem ser previsíveis e consistentes e não oferecer nenhuma surpresa ao usuário (MYERS 1979, p.7). Portanto, o sistema deve ser testado para que se possa identificar o maior número de erros possíveis e não para demostrar que está funcionando. Para o desenvolvedor e o cliente, os testes devem mostrar que o software atende aos requisitos, além de descobrir situações em que o sistema se comporta de forma incorreta, ou de forma diferente das especificações (SOMMERVILLE 2011, p. 144).

Os testes funcionais ou teste de caixa-preta são conhecidos por estes nomes pelo fato de tratar o sistema como uma caixa, cujo conteúdo é desconhecido, do qual só é possível visualizar o lado externo (BARBOSA *online*, p. 7). Este teste não se preocupa com a estrutura interna do sistema, mas com os resultados obtidos através de entradas fornecidas. Os resultados obtidos são comparados aos resultados esperados, então o teste terá sucesso se os resultados obtidos forem iguais aos esperados (NETO *online*, p.58). Contudo, nesta ocasião os testes estruturais terão apenas uma breve introdução, pois a metodologia utilizada no trabalho tem como foco os testes funcionais.

2.3.2.1 Teste de Caixa Branca

Peters e Pedrycz (2001) definem como sendo o objetivo do teste de caixa branca, testar a parte interna do software, casos de testes baseados na estrutura lógica interna do software. Inclui os caminhos lógicos, os loops de repetição e a abrangência dos dados.

O teste de caixa branca é realizado no código fonte, analisando de forma minuciosa a sua estrutura. É observado o caminho que é percorrido da entrada dos

dados até serem geradas as saídas, além da abrangência da entrada e saída de dados e dos ciclos de repetições, dentro dos seus limites.

Teste de caixa branca ou teste estrutural tem como referência o fluxo de controle do programa (SANCHES 1996, *online*), o qual é uma notação que representa o controle lógico do sistema e consiste basicamente em um grafo direcional. No grafo, se tem nós que representam uma instrução procedimental e as arestas representam o fluxo de controle, conforme pode ser observado na Figura 1.

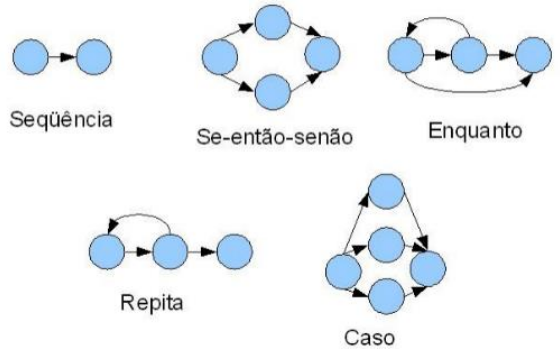


Figura 1 – Estruturas em forma de grafo.

A Figura 1 demostra a forma de representação do fluxo de controle, no qual temos a representação de uma sequência, de se-então-senão, enquanto, repita e caso. Estas formas são utilizadas para representar as estruturas do sistema, de acordo com o caminho percorrido pelo fluxo da funcionalidade, ou seja, uma função tem um laço de repetição e uma condição de parada, portanto, pode ser utilizada a estrutura “repita” apresentada na Figura 1, sendo que os testes de caixa branca devem ser projetados para percorrer todos os caminhos do fluxo pelo menos uma vez. Na Figura 2 temos um trecho de código que faz o cálculo de média dos valores de entrada.

```
Procedimento media
INTERFACE ACEITA valor, min, max
INTERFACE RETORNA media, entradas, validas

var
   valor[1..100] vetor de real
   media, entradas, validas, min, max, soma: real
   i : inteiro
inicio
   i = 1
   totalEntradas = 0
   totalValidas = 0
   soma = 0
   enquanto valor[i]<>-999 e entradas<100 faça
      entradas=entradas+1
      se valor[i]>=min e valor[i]<=max então
         validas=validas+1
         soma=soma+valor[i]
      senão pule
      fimse
      i=i+1
   fimenquanto
   se validas>0 então
      media=soma/validas
   senão
      media=-999
   fimse
fim
```

Figura 2 – Código para Calcular Média

A Figura 2 apresenta o código para calcular a média dos valores de entrada válidos. Ele será utilizado para entender melhor o fluxo com método do caminho básico. Cada círculo azul com um número na Figura 2 representa um processo a ser verificado, no qual são representados cada caminho que deve ser percorrido de forma individual, ou seja, cada nó pode ser uma variável, uma condição ou um laço de repetição. A representação do fluxo do código em grafo, Figura 2, é apresentada na Figura 3.

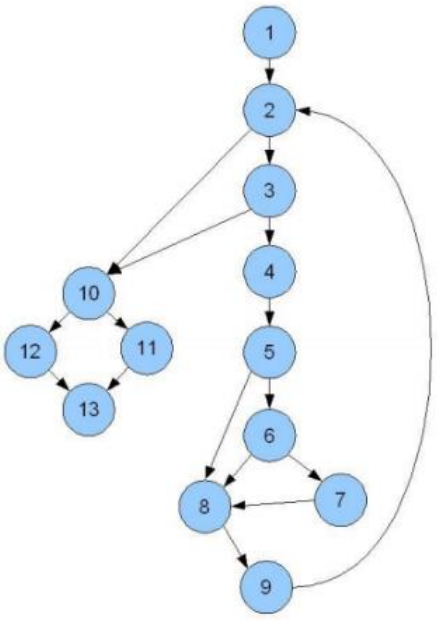


Figura 3 – Grafo correspondente ao código.

A Figura 3 apesenta o grafo correspondente ao código da Figura 2. Com o fluxo definido em grafo, os testes podem ser elaborados de forma a percorrer todos os caminhos básicos. Além dos caminhos básicos, os loops e a abrangência dos dados devem ser contemplados nos testes de caixa-branca.

2.3.2.2 Testes de Caixa Preta

Os testes de caixa preta ou testes funcionais têm como objetivo realizar os testes sem se preocupar com a estrutura interna do software, ou seja, esses testes não têm acesso ao código fonte da funcionalidade. Para realizar estes testes, segundo Neto (*online,* p. 57), dados são fornecidos como entrada para a funcionalidade, o teste é executado e o resultado obtido é confrontado a um resultado estabelecido na definição do teste. Sendo assim, o sucesso do teste está diretamente ligado à igualdade dos resultados obtidos e os resultados esperados.

Para Myers (1979, p. 13), em uma situação ideal de teste é necessário testar todas as entradas possíveis, e não só as válidas, mas na maioria dos casos isso é impossível devido à grande quantidade de entradas possíveis a uma única funcionalidade. Sendo assim, deve-se usar uma abordagem realista para os testes de caixa preta, escolhendo um subconjunto de entradas, no qual, os dados irão maximizar a abrangência dos testes. Um exemplo é uma funcionalidade que tem como dado do tipo inteiro como entrada, seriam necessários alguns milhares de casos de testes para que se possa testar todas as entradas possíveis. Os subconjuntos que melhor atenderem os testes funcionais devem ser escolhidos a partir da especificação da funcionalidade. Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste (MALDONADO 2004, p. 7). Para que os testes sejam ainda mais eficazes, além de casos de testes com entradas válidas, casos de testes com entradas inválidas devem ser usados na execução dos testes. Para definir as classes que melhor atendem uma entrada de dado podem ser utilizadas as seguintes técnicas:

- Particionamento em Classe de Equivalência: é um método que divide o domínio de entrada de um programa em classes de dados a partir das quais os casos de testes podem ser derivados (PESSOA *online*, p. 13). A partir da entrada de dados identificada nas especificações, é divido o domínio de entrada em classes de equivalências válidas e inválidas. Em seguida é escolhido o menor número de

casos de testes, baseado na hipótese de que eles representem toda a classe de dados. Segundo Maldonado (2004, p. 8), o uso de particionamento permite examinar mais sinteticamente e restringir o número de casos de teste exigidos.

- Análise do Valor Limite: é uma técnica que leva a escolha de casos de testes que põem a prova os valores fronteiriços (PESSOA *online*, 14). Os casos de testes são escolhidos nas extremidades das classes, pois segundo Maldonado (2004, p. 8), nesses pontos se concentram um número grande de erros. A Análise do Valor Limite é um complemento ao critério de escolha dos casos de testes do Particionamento em Classes de Equivalência.

2.4 Testes Automatizados

Testes automatizados são programas ou *scripts* simples que exercitam funcionalidades do sistema e fazem verificações automáticas nos efeitos colaterais obtidos (BERNADO 2008, p. 2). Para Maldonado (2004, p. 32), "a qualidade e produtividade da atividade de teste são dependentes do critério de teste utilizado e da existência de uma ferramenta de teste que o suporte". A ferramenta de teste é essencial para a produção de software de alta qualidade. Além disso, as ferramentas de testes disponibilizam recursos para testes de regressão, facilitando assim a execução deste teste que quando executado manualmente são demorados e complexos. Com as ferramentas automatizadas, os casos de testes utilizados durante a atividade de teste podem ser facilmente obtidos para revalidação do software após uma modificação, facilitando e agilizando o teste de uma funcionalidade que foi alterada ou corrigida, e consequentemente reduzindo os custos.

Os testes automatizados são bem abrangentes, pois envolvem todas as camadas do sistema, para tanto, segundo Bartié (2002, p. 112), existem categorias distintas para os testes, sendo elas divididas em teste de:

- Funcionalidade: tem como objetivo simular os cenários de negócio e garantir que todos os requisitos funcionais sejam implementados. A ideia é garantir que os requisitos funcionais não sejam diferentes do comportamento do software desenvolvido;

- Usabilidade: esta categoria visa medir a facilidade disponibilizada na aplicação, ou seja, se o software é fácil e intuitivo. O software deve ser fácil de usar e seus botões e telas devem ser intuitivos a fim de melhor atender o cliente;
- Carga: tem por objetivo simular condições atípicas de utilização do software, utilizando-se de aumentos e reduções de transações que ultrapassam o volume máximo previsto para o software. A ideia é provocar momentos de picos a fim de avaliar o comportamento do software e da infraestrutura;
- Volume: essa categoria tem por objetivo determinar os limites de processamento e carga do software e de toda a infraestrutura. Neste teste a ideia é incrementar sucessivas operações realizadas no sistema, até que atinja o limite o qual a infraestrutura está preparada para processar;
- Configuração: tem por objetivo utilizar diversas configurações de software e hardware e executar o sistema, a fim de verificar se o sistema se comporta adequadamente em diversos ambientes que foram previstos na fase de levantamento de requisitos;
- Compatibilidade: o objetivo desta categoria é garantir que as novas versões estejam suportando antigas interfaces, ou seja, o software é executado interagindo com as versões anteriores de outras aplicações e é verificado se o comportamento é normal. É comum durante atualizações ocorrer conflito de compatibilidade entre as versões, isso significa que alguma alteração na interface provoca incompatibilidade;
- Segurança: tem como objetivo detectar as falhas de segurança que podem comprometer o sigilo e a fidelidade das informações, bem como provocar perda de dados ou interrupção do processamento. A segurança do software pode ser comprometida por ataques internos e externos. A ideia é avaliar o nível de segurança da infraestrutura, simulando ações de criminosos que provocam quebra na segurança;
- Performance: esta categoria é baseada nos picos simulados para verificar o máximo de acesso e concorrência. Para ter sucesso neste teste é estabelecido um critério no qual é atribuído um número de transações e o tempo de respostas esperado, estes valores são comparados com os valores obtidos nos testes, uma análise aponta se os valores obtidos são satisfatórios. O sistema deve funcionar com performance considerável em momentos críticos com muitas transações;

- Instalação: tem por objetivo validar o procedimento de instalação, para tanto, deve ser executado as variações de instalação e avaliar o comportamento durante a execução destes processos;
- Confiabilidade e Disponibilidade: esta categoria de testes é interessante para ser executada nos testes de aceite no qual o sistema está totalmente disponível para o usuário final. Durante os testes é determinado um período no qual o sistema será observado e assim avaliar o nível de confiabilidade da arquitetura da solução. Em caso de falhas é necessário contabilizar o tempo necessário para resolução do problema;
- Recuperação: em sistemas que necessitam de disponibilidade integral essa categoria de teste é importante, pois ela visa avaliar a capacidade do sistema em manter a execução mesmo após a ocorrência de um erro, ou seja, no caso de erros de qualquer natureza, o sistema deve ter a capacidade de manter em execução até que a condição de impedimento desapareça.

As categorias de testes devem ser escolhidas de acordo com as características do sistema e os riscos envolvidos no projeto, ou seja, existe uma relação de custo benefício que envolve as categorias de testes que serão executadas, sendo que as categorias mais importantes são aquelas que irão garantir o funcionamento essencial do sistema (BARTIÉ 2002, p. 119).

As ferramentas de testes usam casos de testes descritos por meio de códigos interpretados, sendo assim, é possível criar cenários de testes bem mais elaborados e complexos (BERNADO 2008, p. 2). Portanto, é possível solucionar um maior número de erros aumentando a qualidade do software. As fases de testes de validação são divididas em teste de unidade, de integração, de sistema e de aceitação.

O teste de unidade é o processo de testar as funções individuais de forma que o teste forneça cobertura a todas as características do objeto (SOMMERVILLE 2011, p. 148). Ou seja, deve-se testar todas as operações associadas, o objeto deve ser colocado em todos os estados possíveis para que todos os eventos que causem mudança de estado sejam testados. Por exemplo, é muito importante simular que centenas de usuários estão acessando o sistema, ou ainda, inserir milhares de registros na base de dados, o que seria quase impossível realizar de forma manual. A Figura 4 ilustra um exemplo de teste unidade.

```
public class ExemploJUnitTest {
    @Test // Este método é um caso de teste
    public void testaAdicaoDeDiasEmUmaData() throws Exception {
        SimpleDateFormat formatador = new SimpleDateFormat("dd/MM/yyyy");
        Date dataDeReferencia = formatador.parse("05/06/2008");
        Date dataDaqui5Dias = formatador.parse("10/06/2008");
        Date dataObtida = DateUtil.adicionaDiasEmUmaData(dataDeReferencia, 5);
        assertEquals(dataDaqui5Dias, dataObtida);
    }
}
```

Figura 4 – Exemplo de código de teste unidade

A Figura 4 apresenta um exemplo de código de teste unidade, neste exemplo a linguagem utilizada é o Java com o auxílio do JUnit. No código, o teste a ser realizado é a adição de cinco dias em uma data de referência, para tanto, foi definido um formato de data, uma data de referência, uma data como resultado. É chamada a funcionalidade que adiciona dias a uma data, para ela é passado o parâmetro de data de referência e a quantidade de dias que será adicionado a ela, logo após é comparado o resultado estabelecido no teste e o resultado obtido com a funcionalidade, e se os resultados forem iguais a funcionalidade passou no teste. Nos testes de unidade se encaixam as categorias de testes de funcionalidade e usabilidade.

Os testes de integração ou testes de interface de componentes tem como objetivo testar os componentes que foram integrados para construir uma funcionalidade maior. Para tanto, as categorias de testes que podem ser usadas são os testes de funcionalidade, de usabilidade e de segurança. Os componentes que compõem essa integração devem passar pelos testes de unidades antes de passar pelos testes de integração, pois nos testes de integração os componentes não podem ser modificados de forma individual. Os testes de integração testam as interfaces criadas nos componentes para que eles se comuniquem entre si. A Figura 5 ilustra a ideia de testes de integração, nos quais os casos de testes não se aplicam em apenas um componente, mas sim nas interfaces criadas entre eles.

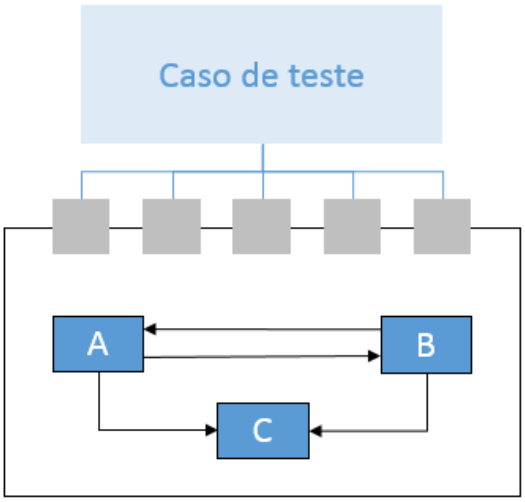


Figura 5 – Ideia do teste de integração

Segundo Sommerville (2011, p. 152), existem diferentes tipos de interface, sendo eles:

- Parâmetro: são interfaces nas quais os dados são passados de uma interface para outra, ou seja, um componente recebe os dados de um outro componente para executar uma determinada função;
- Memória compartilhada: são interfaces nas quais um bloco de memória é compartilhado entre componentes. Os dados são colocados de forma que subsistemas possam ser acessados;
- Procedimento: são interfaces nas quais um componente encapsula um conjunto de procedimentos que podem ser chamados por outros componentes;
- Passagem de mensagem: são interfaces nas quais um componente solicita a execução de um serviço e o outro responde com uma mensagem com o resultado a execução do serviço.

Com os testes de integração já executados, em seguida, vem os testes de sistemas nos quais verifica-se o comportamento dos componentes quando executados todos juntos. Este teste é executado em um ambiente igual ao de produção para garantir que o sistema funcione como um todo. O teste de sistema sob a ótica de Couto (*online*, p.4), tem como objetivo executar o sistema no ponto de vista do usuário final, executando as funcionalidades em busca de falhas. Nos testes de sistema as categorias de performance, instalação, recuperação e carga são importantes para que, quando o

sistema estiver em produção o cliente não tenha surpresas com lentidões ou falhas impeditivas.

Para Sommerville (2011, p. 155), os testes automatizados de sistema geralmente são mais difíceis do que testes automatizados de unidades ou de integração, sendo que os testes unitários se baseiam em estabelecer uma entrada e uma saída esperada e compilar as previsões em um programa. Os resultados são comparados e é obtida uma conclusão dos testes. Nos de integração são escritos funcionalidades que se integram e são também compilados e os resultados são analisados para que uma conclusão do teste seja estabelecida. Já os testes de sistema são rotinas do dia a dia do usuário final em que infinitas possibilidades e situações podem acontecer. No entanto, o importante na aplicação do teste de sistema é gerar situações atípicas que levem o sistema aos limites de suas funcionalidades garantindo assim que o sistema funcione como um todo.

Os testes de aceitação são realizados com o software pronto para entrar em produção, ou seja, esse teste é geralmente realizado com a participação intensa do usuário final. Testes de aceitação, para Bernador (2008, p. 5), visam verificar se o que foi implementado atende corretamente ao que o cliente esperava. Este teste valida o sistema no ponto de vista do cliente. Nos testes de aceitação se encaixam as categorias de testes de funcionalidade, de usabilidade e de segurança.

2.5 Análise de Pontos de Testes (APT)

Segundo Cruz (*online*, p. 1), Análise de Pontos de Teste (APT) “é uma técnica de estimativa de tamanho de teste que gera previsões de esforço e tempo”. Ou ainda, segundo Campos (2010, *online*), “é uma técnica de medição para a área de Teste de Software, baseada na técnica de medição de Análise de Pontos de Função”. Proposta por Martin Pol, Ruud Tennissen e Erik van Veenendaal em 1999, tem como base principal, para determinar as estimativas, os Pontos de Função (PF) de um sistema, além do ambiente de teste e a produtividade da equipe. O tamanho em PF do sistema e a estratégia de teste, juntos, determinam o volume de trabalho de teste a ser realizado, expressados em Pontos de Teste (PT) (VEENENDAAL 1999, p. 1). Para se obter as horas de testes primárias, os PT são diretamente relacionado com a produtividade da equipe e o ambiente de teste.

Para Campos (2010, *online*), é importante utilizar uma técnica de medição específica para a atividade de testes, na qual detalhes importantes precisam ser considerados para estimar os testes de software de maneira eficaz. A APT considera pontos importantes que influenciam diretamente nas estimativas, sendo eles, o número de pontos de testes dinâmicos e estáticos, o ambiente de teste, além da produtividade da equipe. A Figura 6 apresenta uma visão geral da APT.

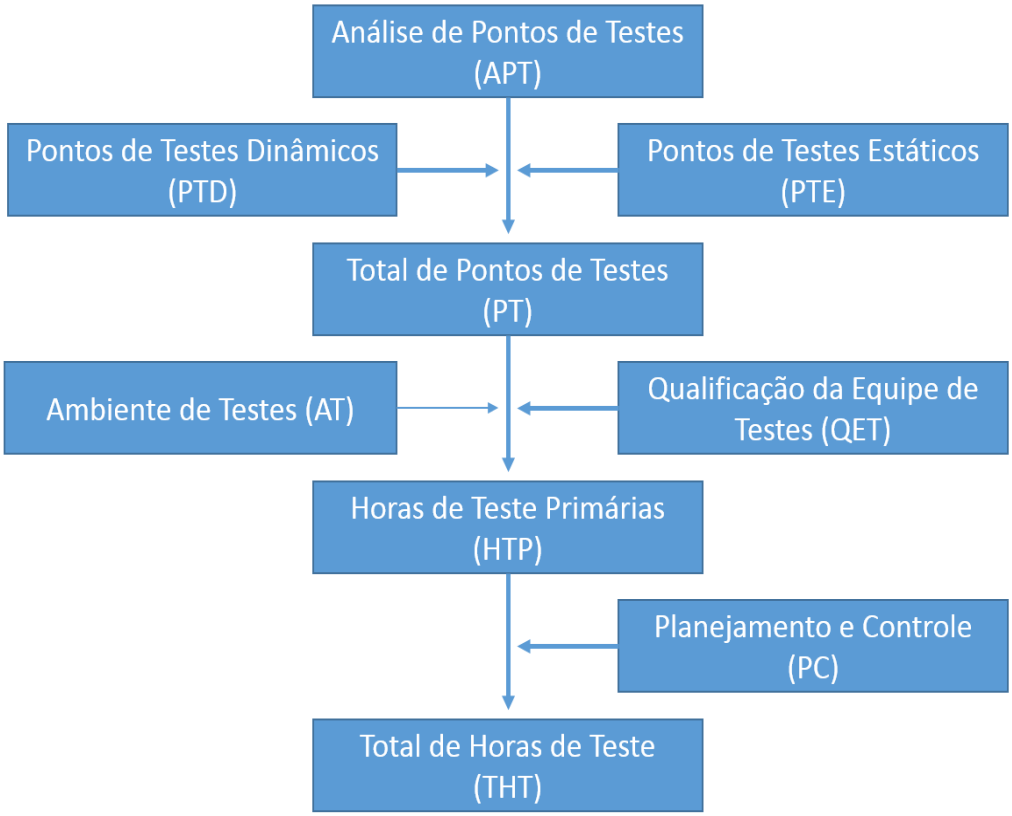


Figura 6 – Visão geral da técnica Análise de Pontos de Testes

A Figura 6 apresenta a visão geral da técnica Análise de Pontos de Testes, no qual o fluxo deve ser seguido de acordo com a Figura 6. Cada passo do fluxo apresenta um cálculo que compõe o cálculo do total de horas de testes.

Pontos de Testes Dinâmicos (PTD) são calculados com base no número de pontos de função (PT) atribuído a funcionalidade, as funções dependentes (FDf) que são diretamente relacionados com a complexidade, interfaces, uniformidade, importância do usuário e a intensidade de uso (ARANHA 2009, p. 13). Além das características da qualidade dinâmica (QRd) dos requisitos descrita na ISO 9126-1 (funcionalidade, desempenho, segurança e aderência). Para tanto, para se obter o valor de PTD, os

fatores PT, FDf e QRd devem ser multiplicados e o resultado é número de pontos de testes dinâmicos.

Pontos de Testes Estáticos (PTE) são calculados com base no número total de pontos de função do software e os requisitos de qualidade, contudo, os PTE são calculados apenas em projetos que executam a atividade de revisão de documentação, sendo assim, o valor dos PTE é nulo para projetos que não realizam esta atividade.

O total de Pontos de Testes (PT) expresso pelo somatório de todos os pontos PTD, somado pelo resultado da expressão PTE multiplicado pelos PF e dividindo o resultado por quinhentos. Com esta expressão, o valor de PT é obtido. O número de Horas de Testes Primárias (HTP) podem ser calculados multiplicando o número total de PT pelo fator ambiente de teste e o fator qualidade da equipe de teste. As HTP representam o tempo necessário para conclusão das fases de preparação, especificação, execução e conclusão dos testes (VEENENDAAL, et. al. 1999 apud TEUNISSEN 1997, p.3).

O Planejamento e Controle (PC) é representado por um índice que tem como base o tamanho da equipe e as ferramentas disponíveis. O PC é um índice que irá ser incluso nas horas primárias para obter o THP. Por fim, o número Total de Horas de Teste é o tempo necessário para todas as atividades de testes (VEENENDAAL 1999, p. 3), e é a inclusão das atividades de PC ao THP.

Para realizar uma APT é necessário acesso as descrições detalhadas dos processos e um modelo lógico de dados. Além da contagem de pontos de função usando técnicas como: IFPUG ou IFPA (VEENENDAAL 1999, p. 4). A contagem de Pontos de Função (PF) utilizando umas das técnicas citadas pode ser utilizada como base para a APT, ressaltando que a escolha da técnica não afeta o cálculo dos PT, mas pode influenciar o fator produtividades.

Para servir de base para o APT, é esperado que haja a contagem de PF, mas não havendo a contagem de PF, o tempo necessário para realizar a contagem pode ser determinado pela multiplicação dos conjuntos de dados lógicos por trinta (VEENENDAAL 1999, p. 5). O resultado é uma aproximação dos PF, e é dividido por quatrocentos para obter o número de dias, sabendo que, é possível contar 400-500 pontos de função por dia.

2.5.1 Pontos de Testes Dinâmicos(PTD)

Os pontos de testes dinâmicos são calculados com base na análise de pontos de função do sistema (PESSOA *online*, p. 18). Além dos pontos de função, outros fatores são tomados como base para obter os pontos de testes dinâmicos, sendo:

- Funções dependentes (FDf);
- Qualidade dos requisitos relacionados com as características de qualidade a serem testadas dinamicamente (QRd);

Os Pontos de Testes Dinâmicos (PTD) são calculados com base nos fatores citados anteriormente, conforme Fórmula 1.

$$PTD = PFf \bullet FDf \bullet QRd$$

Fórmula 1 – Cálculo dos Pontos de Testes Dinâmicos

Na Fórmula 1, PTD é o valor obtido para pontos de testes dinâmicos, PFf é os pontos de funções do sistema, FDf é o fator funções dependentes conforme Fórmula 2 e QRd é o valor obtido para qualidade dinâmica conforme Fórmula 6.

2.5.1.1 Funções Dependentes (FDf)

Segundo Pessoa (*online*, p. 19), as funções dependentes são chamadas assim em decorrência do grau de dependência que têm das funções correspondentes medidas pela teoria de pontos de função. Nesta relação, Veenendaal (1999, p. 5), considera: os fatores de importância que o usuário dá ao processo de teste, a intensidade do uso da função, as interfaces com os arquivos, a complexidade do código e a uniformidade do material de teste; os quais serão apresentados a seguir:

- Importância do Usuário (Ue): este fator pode ser obtido através de entrevistas com o usuário. Para determinar o grau de importância do usuário os graus de importâncias devem apresentar valores conforme os mostrados na Figura 7.

Complexidade	Peso
Baixa	3
Normal	6
Alta	12

Figura 7 – Classificação de Importância para o usuário

Intensidade de Uso (Uy): a intensidade deve ser determinada pela utilização em um espaço de tempo. Funções podem ser utilizadas muitas vezes durante um espaço de tempo e outras usadas eventualmente. A intensidade deve ser representada conforme a Figura 8.

Complexidade	Peso
Baixa	2
Normal	4
Alta	8

Figura 8 – Classificação de Intensidade de Uso

Interface (I): este fator mede o nível de relações entre os arquivos e as funções que estão sendo medidas, levando em consideração o número de arquivos afetados pela função que está sendo medida e o número de funções que afetam um arquivo específico. A interface deve ser considerada “Baixa” se a função não modificar nenhum arquivo. As interfaces devem ser representadas conforme Figura 9 e Figura 10.

Arquivo	Funções		
	1	2 - 5	> 5
1	Baixa	Baixa	Normal
2 - 5	Baixa	Normal	Alta
> 5	Normal	Alta	Alta

Figura 9 – Fator de Complexidade da interface (Tabela de Pontos de Função)

A Figura 9 apresenta os níveis de importância que devem ser estabelecidos de acordo com a quantidade de arquivos que são alterados a partir da execução da funcionalidade.

Complexidade	Peso
Baixa	2
Normal	4
Alta	8

Figura 10 – Classificação da complexidade da interface

A Figura 10 representa o valor atribuído a interface de acordo com a complexidade apresentada na Figura 9.

Complexidade (C): este fator é medido pela quantidade de comandos condicionais em seus algoritmos, tais como IF ou CASE. Por exemplo, uma CASE com *n* cases conta *n-1* vezes. Com tudo, a metodologia utilizada nesta proposta, TDD, as funcionalidades serão implementadas após a elaboração dos testes, portanto, os códigos das funcionalidades não estão disponíveis. Para Pessoa (*online*, p. 20), nestas ocasiões é razoável a utilização do nível normal. A complexidade deve apresentar valores conforme Figura 11.

Complexidade	Peso
Baixa	3
Normal	6
Alta	12

Figura 11 – Classificação do grau de complexidade

Uniformidade (U): mede o nível de reutilização do material de teste, ou seja, considera especificações já definidas para ser utilizada em estruturas semelhantes. Para tanto, os valores para a uniformidade tem variação entre 1,0 e 0,6. Em matérias não reutilizáveis em outras funções é atribuído do valor de 1,0, em casos de funções semelhantes em que o material pode ser reutilizado é atribuído valores de 0,8 ou 0,6. Para Veenendaal (1999, p. 10), normalmente é atribuído o valor 1,0 para a uniformidade. A uniformidade deve apresentar os valores conforme a Figura 12.

Complexidade	Peso
Completa utilização do material de teste	0,6
Parte do material será reutilizado	0,8
Não há material de reutilização	1,0

Figura 12 – Classificação de uniformidade

Com os fatores determinados é possível realizar o cálculo das funções dependente conforme Fórmula2:

$$FDf = \left(\frac{Ue + Uy + I + C}{20}\right) \bullet U$$

Fórmula 2 – Cálculo das Funções Dependentes

Nesta Fórmula 2, Ue é a importância do usuário, Uy é a intensidade de uso, I é a interface, C é a complexidade e U é a uniformidade.

2.5.1.2 Características de Qualidade Dinâmica (QRd)

As características de qualidade dinâmica medem como a qualidade dos requisitos pode afetar a qualidade dos testes (PESSOA *online*, p. 21). Para a medição da qualidade dinâmica são consideradas medidas explícitas e medidas implícitas.

As características explícitas (CE), são calculadas levando em consideração a qualidade das seguintes:

a) Funcionalidade (F): funcionalidade está conforme o processo estabelecido;
b) Desempenho (D): desempenho do sistema em realizações de atividades diárias;
c) Segurança (S): segurança do sistema em relação a ataques internos e externos;
d) Aderência e Efetividade (A): real aderência do sistema, ou seja, a facilidade que o usuário tem para aprender ou utilizar o sistema.

Antes de definir as CE, devem ser considerados os valores da Figura 13, que apresenta os valores definidos para a importância da qualidade dos requisitos.

Descrição	Valor
A qualidade dos requisitos não é importante para o resultado dos testes	0
A qualidade dos requisitos não é importante, mas precisa ser considerada para o resultado dos testes	3
A qualidade dos requisitos tem importância média	4
A qualidade dos requisitos é muito importante	5
A qualidade dos requisitos é extremamente importante	6

Figura 13 – Classificação da importância da qualidade dos requisitos

Com o valor atribuído para a funcionalidade deve-se multiplicar o valor pelo peso de cada característica, conforme Figura 14.

Característica	Peso
Funcionalidade (F)	0,75
Desempenho (D)	0,10
Segurança (S)	0,05
Aderência e Efetividade (de acordo)	0,10

Figura 14 – Características explícitas

Cada característica pode ser calculada com a Fórmula 3:

$$F = \frac{(VT) \bullet P}{4}$$

Fórmula 3 – Cálculo de cada característica

Nesta Fórmula 3, F é a característica explícita, podendo ser Funcionalidade (F), Desempenho (D), Segurança (S), Aderência e Efetividade (A). VT é o valor atribuído a característica conforme Figura 13, e P é o peso da característica conforme Figura 14.

Com o cálculo dos valores relacionados a cada característica conforme Fórmula 2, as Características Explícitas são calculadas de acordo com a Fórmula 4.

$$CE = F + D + S + A$$

Fórmula 4 – Cálculo da característica explícita

Na Fórmula 4, CE é o valor obtido da característica explícita, F o valor calculado para Funcionalidade conforme Fórmula 3, D o valor calculado para Desempenho conforme Fórmula 3, S o valor calculado para Segurança conforme Fórmula 3 e A o valor calculado para Aderência conforme Fórmula 3.

Sempre que houver indicadores que possam ser utilizados para avaliar umas das características explícitas (funcionalidade, desempenho, segurança e aderência), considera-se que pode existir uma característica implícita (PESSOA *online,* p. 22). As características implícitas são utilizadas para fornecer uma medida padrão de comparação com outros projetos. Para calcular a Característica Implícita (CI), cada avaliação da CE deve ser multiplicada por 0,02 deve-se utilizar a Fórmula 5.

$$CI = n \bullet 0{,}02$$

Fórmula 5 – Cálculo da característica implícita

Na Fórmula 5, CI são as características implícitas, *n* é quantidade de características que foram avaliadas por indicadores, ou seja, se as características funcionalidade e aderência tiverem indicadores que permitam a sua avaliação, então *n* será 2, portanto *n* pode variar de 1 a 4.

A QRd é a soma das características explícitas e implícitas conforme Fórmula 6.

$$QRd = CE + CI$$

Fórmula 6 – Cálculo da qualidade de requisitos

Na Fórmula 6, QRd é o valor obtido para a qualidade de requisitos, CE é o valor obtido para características explícitas e CI o valor obtido para características implícitas.

2.5.2 Pontos de Testes Estáticos (PTE)

Os pontos de testes estáticos devem ser incluídos nas estimativas em projetos que a equipe de teste adotar o processo de revisão de documento. Para Pessoa (*online,* p. 23), os pontos de testes estáticos levam em consideração o sistema como um todo. Em projetos que não realizam a atividade de revisão de documento, os PTE terão valor nulo.

O cálculo dos pontos de testes estáticos tem como base os critérios de qualidade para a avalição das características de qualidade de software da ISO 9126-1 (funcionalidade, desempenho, segurança e aderência). De acordo com Pessoa (*online,* p. 23), esta avaliação pode ser feita a partir de um *checklist*, sendo um para cada característica. Ainda segundo Pessoa (*online,* p. 23), para cada *checklist* são adicionados 16 pontos de testes. Para tanto, os PTE devem ser calculados conforme Fórmula 7.

$$PTE = 16 \bullet n$$

Fórmula 7 – Cálculo dos Pontos de testes estáticos

Na Fórmula 7, PTE é o valor obtido para os pontos de testes estáticos e *n* é a quantidade de características avaliadas, portanto, se será utilizado *checklist* para avaliar as características desempenho e aderência, então *n* será 2. O *n* sempre será menor que 4.

2.5.3 Total de Pontos de Testes (PT)

O total de pontos de testes é calculado através dos dados PTD, PTE e PF, conforme Fórmula 8.

$$PT = \sum PTD + \frac{(PF \bullet PTE)}{500}$$

Fórmula 8 – Cálculo dos Pontos de Testes

Na Fórmula 8, PT é o valor obtido de pontos de testes, ∑ PTD é a somatória dos pontos de testes dinâmicos, PF é o valor de pontos de função e PTE é o valor de pontos de testes estáticos.

A segunda parte da Fórmula 8, (PF*PTE)/500, será nula em projetos que não realizarem a atividade de revisão de documentos. Com o valor para a quantidade de pontos de testes obtidos o próximo passo é obter as horas de testes primárias (HTP), e para esse passo é necessário que seja definido valores para qualidade da equipe de teste (QET) e um valor para o ambiente de teste (AT).

2.5.4 Qualidade da Equipe de Testes (QET)

A qualidade da equipe de teste é diretamente ligada a produtividade da equipe de teste. Para Veenendaal (1999, p. 11), o fator produtividade é uma medida da experiência, conhecimento e habilidades da equipe de teste, sendo que o fator produtividade indica o número de horas de teste necessárias por ponto de teste, ou ainda, segundo Campos (2010, *online*), produtividade é o tempo total para realizar determinado volume de teste. Para Pessoa (*online*, p.24), o valor da produtividade pode ser baseado em uma base histórica, na qual dados de projetos anteriores são analisados afim de determinar a produtividade da equipe. Entretanto, em equipes sem base histórica para servir de base, deve ser estabelecida a produtividade com o peso médio conforme apresentado na Figura 15.

Produtividade	Valor
Alta	0,7
Média	1,3
Baixa	2,0

Figura 15 – Classificação da produtividade da equipe de teste

Na Figura 15, são apresentados os valores que podem ser atribuídos a QET, sendo que, quanto melhor ou mais produtiva for a equipe menor é o valor atribuído para QET.

2.5.5 Ambiente de Testes (AT)

O número de horas de testes primárias é influenciado não só pelo fator produtividade, mas também pelo fator ambiente de teste (VEENENDAAL 1999, p. 11). O cálculo do ambiente deve levar em consideração: ferramenta de teste, teste de precedência, documentação de testes, ambiente de desenvolvimento e *testware* (PESSOA *online*, p. 26).

As ferramentas de testes devem ser mencionadas no planejamento dos testes, sendo que, a equipe de teste deve deixar explícito se irá utilizar uma ferramenta para automatizar os testes. Segundo Veenendaal (1999, 12), a disponibilidade de ferramentas de testes significa que algumas das atividades de testes podem ser executadas automaticamente. O valor atribuído para as ferramentas de testes devem ser atribuídos conforme Figura 16.

Descrição	Peso
Existe uma ferramenta de automação para as fases de especificação **E** execução dos testes.	1
Existe uma ferramenta de automação para as fases de especificação **OU** para a fase de execução.	2
NÃO existe ferramenta de automação de teste.	4

Figura 16 – Ferramentas de testes

Nos testes de precedência Pessoa (*online*, p.25) discorre que, "se um teste for bem executado, os resultados do teste seguinte terão resultados melhores". Para isso, os testes de precedência devem ser descritos de forma explícita no plano de teste, deixando claro toda a documentação utilizada para a elaboração dos testes. Segundo Pessoa (*online*, p. 25), nos testes de procedência, para cada etapa do processo de teste, a atividade imediatamente anterior deve produzir bons resultados para que a atividade seguinte possa ser bem executada, para tanto, o plano de teste deve conter resultados palpáveis. O valor atribuído para os testes de precedência deve ser atribuído conforme Figura 17.

Descrição	Peso
Existe um plano para o teste precedente e a equipe está familiarizada com ele, assim como os casos de teste e resultados de teste.	2
Existe um plano para o teste precedente.	4
Não existe um plano para o teste precedente.	8

Figura 17– Tabela de Precedência

A documentação de testes recebe valor conforme Figura 18, sendo que, se o processo de testes tem documentos bem definidos implicará no valor do peso.

Descrição	Peso
Durante o desenvolvimento do sistema são usados padrões de documentação e templates. Acontecem revisões periódicas da documentação.	3
Durante o desenvolvimento do sistema são usados padrões de documentação e templates.	6
A documentação não segue nenhum padrão nem templates são usados.	12

Figura 18 – Qualidade da Documentação

A variável de ambiente de desenvolvimento para Veenendaal (1999, p. 12) reflete a natureza do ambiente dentro do qual o sistema de informação foi desenvolvido, para tanto, é considerado como base a linguagem na qual o sistema foi desenvolvido, sendo que ela pode ser de quarta geração, terceira geração ou a combinação das duas gerações. O valor para o ambiente de desenvolvimento deve ser atribuído conforme Figura 19.

Descrição	Peso
O sistema foi desenvolvido usando uma linguagem de 4ª geração (integrada ao sistema de gerência de banco de dados).	2
O sistema foi desenvolvido usando uma combinação de linguagem de 4ª e 3ª geração.	4
O sistema foi desenvolvido em linguagem de 3ª geração.	8

Figura 19 – Linguagem de Programação

O ambiente de teste é um valor atribuído a familiarização da equipe de teste com o ambiente de teste. Segundo Veenendaal (1999, p. 12), a variável ambiente de teste é

reflexo do quanto o ambiente de teste já foi utilizado pela equipe. O valor para ambiente de teste deve ser atribuído de acordo com a Figura 20.

Descrição	Peso
O ambiente de teste já foi usado inúmeras vezes.	1
O ambiente de teste é similar ao que já havia sido usado anteriormente.	2
O ambiente de teste é completamente novo e experimental.	4

Figura 20 – Ambiente de Teste

Como o último fator considerado para calcular o valor para ambiente de teste (AT), tem-se o fator *testware*, que é o reflexo das atividades de testes em que pode-se utilizar *testware* já existentes (VEENENDAAL, 1999, p. 13). O valor para *testware* deve ser atribuído de acordo com a Figura 21.

Descrição	Peso
Existem materiais de testes, como bases de dados, tabelas, casos de teste e outros, que poderão ser reutilizados.	1
Existem apenas tabelas e bases de dados disponíveis para reutilização.	2
Não existe material disponível.	4

Figura 21 – Base de dados reutilizável

Com os fatores para o cálculo do ambiente de teste (AT) especificado, o valor do AT pode ser calculado conforme Fórmula 6.

$$AT = \frac{STF}{21}$$

Fórmula 6 – Cálculo do Ambiente de Teste

Na Fórmula 9, AT é o valor obtido para ambiente de teste, STF e a soma de todos os fatores citados acima (ferramenta de teste, teste de precedência, documentação de testes, ambiente de desenvolvimento e *testware*).

2.5.6 Estimativa de Horas de Teste Primárias (HTP)

O número de horas de testes primárias (HTP) é obtido pela multiplicação dos fatores pontos de testes (PT), qualidade da equipe de testes (QET) e ambiente de testes

(AT). Para Veenendaal (1999, p. 11), a contagem de horas de teste primárias é o número de horas necessário para a realização das atividades envolvidas no ciclo de vida do teste sendo: preparação, especificação, execução e conclusão. Para obter-se o valor de HTP o cálculo deve ser feito conforme Fórmula 10.

$$HTP = PT \bullet QET \bullet AT$$

Fórmula 10 – Cálculo das Horas Primárias de Teste

Na Fórmula 10, HTP é o valor obtido para o número de horas de testes primárias, PT é o total de pontos de testes, QET é o valor para a qualidade da equipe de teste e AT é o valor para ambiente de teste.

As HTP devem ser corrigidas com a inclusão das atividades de planejamento e controle (PC), cujo percentual é dado pelos fatores tamanho da equipe e ferramentas gerenciais.

2.5.7 Número total de horas de Teste (THT)

Para corrigir as HTP deve ser levado em consideração o PC. Para Veenendaal (1999, p. 14), o número HTP e o PC, juntos, dão o número total de horas de teste (THT). No entanto, o índice do PC pode aumentar ou diminuir, de acordo com os fatores: tamanho da equipe e ferramentas de gestão.

O fator tamanho da equipe (TE) corresponde ao número de profissionais que forma a equipe de teste, para tanto, o valor atribuído para TE deve ser conforme Figura 22.

Descrição	Peso
Entre 1 e 4 técnicos	0,03
Entre 5 e 10 técnicos	0,06
Mais de 10 técnicos	0,12

Figura 22 – Classificação do tamanho da equipe de teste

A variável de ferramentas de gestão (FG) recebe um valor de acordo com os recursos que são utilizados para automatizar o PC. Para tanto, o valor deve ser atribuído de acordo com a Figura 23.

Descrição	Peso
Existem ferramentas de registro de tempo e de gerência de defeitos (devtrack), além de ferramentas de gerência de configuração.	0,02
Apenas uma das ferramentas citadas acima está disponível.	0,04
Não existem ferramentas disponíveis.	0,08

Figura 23 – Classificação das ferramentas gerenciais

O índice de PC é calculado pela soma dos percentuais de TE e FG, para que o valor de PC seja acrescentado as HTP para que o THT seja obtido. Para conseguir o valor de PC, o cálculo deve ser feito de acordo com a Fórmula 11.

$$PC = 1 + (TE + FG)$$

Fórmula 11 – Cálculo do índice de Planejamento e Controle

Na Fórmula 11, PC é o valor obtido para o índice de planejamento de controle, TE é o percentual para o tamanho da equipe e FG é o percentual para a ferramenta de gestão.

Com o índice de PC obtido as HTP podem ser corrigidas conforme Fórmula 12, para se conseguir o número total de horas de teste.

$$THT = HTP \bullet PC$$

Fórmula 12 – Cálculo do Total de Horas de Testes

Na Fórmula 12, THT é o valor obtido para o total de horas de testes, HTP é o valor de horas de teste primária e PC é o índice de planejamento e controle.

Com a Fórmula 12, o número total de horas de testes é obtida e, assim, a equipe terá um número expressado em horas para a atividade de teste do projeto.

2.6 Test Driven Development (TDD)

O *Test Driven Development* (TDD) ou desenvolvimento orientado a teste é uma metodologia ágil derivada do método *Extreme Programming* (XP) e do *Agile Manifesto* (BORGES *online*, p. 2). Trata-se do desenvolvimento de um sistema começando pelos casos de teste de um objeto. Para Sommerville (2011, p. 155), "TDD é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código". Basicamente é desenvolvido um código em conjunto com um teste, sendo que

o próximo incremento só pode ser desenvolvido quando o código passar no teste, logo após o sucesso do código se necessário deve acontecer a refatoração, abordada na seção 2.6.1.

Camelo (2010, p. 2) afirma que decorrente do manifesto ágil, como o XP, o TDD propõe a criação de teste antes da codificação. Ou seja, uma abordagem diferente do modelo tradicional em relação aos testes, na qual os testes são desenvolvidos e executados dias ou mesmo meses depois da funcionalidade pronta. Segundo Sommerville (2011, p. 155), as etapas fundamentais do processo do TDD estão representadas na Figura 24, sendo elas:

1. É identificado o incremento de funcionalidade necessário. Este deve ser implementado de forma simples e em poucas linhas de código;
2. O teste é escrito para esta funcionalidade, e quando ele for executado será relatado se passou ou falhou;
3. O teste é executado juntamente com os outros testes implementados. Como a funcionalidade ainda não foi desenvolvida, os testes logo falharam;
4. Então, a funcionalidade é desenvolvida e submetida novamente aos testes, sendo que isso pode envolver a refatoração do código para melhorá-lo;
5. Depois que os testes forem executados com sucesso, as próximas funcionalidades serão implementadas.

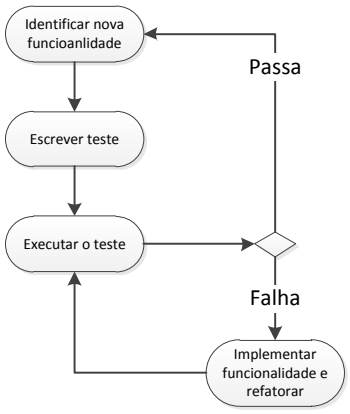


Figura 24 – Etapas do processo do TDD

A Figura 24 apresenta o fluxo do processo do TDD, que se inicia com a identificação da funcionalidade que deve ser desenvolvida. Com a funcionalidade identificada, um teste deve ser escrito seguindo as especificações da funcionalidade no

seu planejamento, sendo que no primeiro momento a funcionalidade não deve passar no teste, com isto, chega o momento de escrever a funcionalidade de maneira mais simples possível para que ela passe no teste. Antes de desenvolver a próxima funcionalidade, o código escrito deve ser refatorado para que o design do código seja simplificado e códigos repetidos sejam retirados, só então, a próxima funcionalidade poderá ser desenvolvida.

Na prática o TDD inicia com a criação de uma classe de teste geral com o teste muito simples. Em seguida será ilustrado um exemplo no qual o algoritmo deve gerar uma lista de números primos a partir no número N, usado como parâmetro. O resultado esperado é uma *string* com a sequência de números primos até um determinado valor, ou seja, primos até dez o resultado é uma *string* "2,3,5,7". A Figura 25 apresenta a classe de teste.

```
1  import junit.framework.TestCase;
2  public class GeradorPrimosTeste extends TestCase {
3      public void testePrimosGeradosAteNumeroDois() throws Exception {
4          GeradorPrimos geradorPrimos = new GeradorPrimos();
5          assertEquals("2", geradorPrimos.gerarPrimosAte(2));
6      }
7  }
```

Figura 25 – Teste testePrimosGeradosAteNumeroDois()

Na classe de teste da Figura 25, o método da linha 5 irá verificar se é possível gerar números primos até o número 2. Entretanto, o código ainda não é compilado, pois a classe GeradorPrimos() ainda não existe, portanto, antes de compilar a classe de teste é necessário criar a classe conforme Figura 26.

```
1  public class GeradorPrimos {
2      public String gerarPrimosAte(int i) {
3          return null;
4      }
5  }
```

Figura 26 – Classe GerarPrimos()

A Figura 26 apresenta o código suficiente para que a classe de teste possa ser compilada. O teste é executado e falha, isto é o esperado, pois o métodos gerarPrimosAte() ainda não executa de acordo com o esperado. No TDD este teste deve sempre falhar, pois a funcionalidade ainda não foi escrita. Em seguida, é feita a implementação simples do método de gerar primos, conforme Figura 27.

```
1  public String gerarPrimosAte(int i) {
2      return "2";
3  }
```

Figura 27 – Método gerarPrimosAte()

Com o método gerarPrimosAte() ilustrado na Figura 27, o teste é novamente executado e assim ele irá passar, logo, o programador tem segurança de que o teste vai falhar quando for necessário falhar. Isto garante que o teste está correto. Em sequência, é feito um teste para gerar até o número 3, conforme Figura 28.

```
1  public void testePrimosGeradosAteNumeroTres() throws Exception {
2      GeradorPrimos geradorPrimos = new GeradorPrimos();
3      assertEquals("2, 3", geradorPrimos.gerarPrimosAte(3));
4  }
```

Figura 28 – Teste testePrimosGeradosAteNumeroTres()

A Figura 28 apesenta um segundo teste, agora o algoritmo deve gerar o primos até o numeral 3, como a funcionalidade, apresentada na Figura 26, ainda não pode resolver o teste, consequentemente ele irá falhar. Então para atender o teste o método de gerarPrimosAte() é modificada para atender o teste, conforme Figura29.

```
1  public String gerarPrimosAte(int i) {
2      if (i == 2) return "2";
3      else return "2, 3";
4  }
```

Figura 29 - Método modificado gerarPrimosAte()

A Figura 29 apresenta o método para solucionar quando o argumento passado for até 3, mas podemos notar que com apenas estes dois testes já temos duplicação no código, conforme Figura 25 e Figura 28, que possuem a implementação quase idêntica. Para tanto, no TDD deve ser executado o princípio de refatorar para eliminar códigos duplicados e simplificar o código.

2.6.1 Refatoração

Com o desenvolvimento orientado a testes o foco do desenvolvedor é voltado para escrever códigos que passem nos testes criados. Contudo, os programadores criam os códigos sem se preocupar com a repetição ou design, pois o objetivo é escrever um código que passe no teste. Assim, os códigos podem se tornar complexos ou ainda duplicados. Portanto, os desenvolvedores devem realizar o processo de refatoração, que para Camelo (2010, p. 3), “não adiciona código novo ao código existente, apenas

reestrutura-o". Então, a cada adição de um novo método deve haver, se necessário, uma refatoração, melhorando assim o *design* do código desenvolvido e retirando duplicidade de código, isto garante que o código seja reescrito utilizando as boas práticas de programação. Continuando a ilustração prática tem-se o código da classe teste na qual dois métodos distintos apresentam quase a mesma função (Figura 30).

```
1  import junit.framework.TestCase;
2
3  public class GeradorPrimosTeste extends TestCase {
4      public void testePrimosGeradosAteNumeroDois() throws Exception {
5          verificaPrimosGerados("2", 2);
6      }
7      public void testePrimosGeradosAteNumeroTres() throws Exception {
8          verificaPrimosGerados("2, 3", 3);
9      }
10 }
```

Figura 30 – Classe teste

A Figura 30 apresenta a classe teste, com os métodos para testar o algoritmo de gerar uma lista de números primos até 2 ou até 3, entretanto, considerando que N números podem compor essa lista sequencial de números primos, apenas esses dois testes não irão abranger a funcionalidade por completo. Portanto, a partir dos métodos muito semelhantes do código da Figura 30, é extraído um método que pode testar uma lista que vai até o número N primo. A Figura 31 ilustra este método.

```
1  import junit.framework.TestCase;
2
3  public class GeradorPrimosTeste extends TestCase {
4      private void verificaPrimosGerados(String listaEsperada,
5      int numeroMaximo) throws Exception {
6          GeradorPrimos geradorPrimos = new GeradorPrimos();
7          assertEquals(listaEsperada,
8          geradorPrimos.gerarPrimosAte(numeroMaximo));
9      }
10 }
```

Figura 31 – Classe teste refatorada

A Figura 31 apresenta a classe de teste refatorada, na qual o método realiza a mesma função dos métodos apresentados na Figura 31, entretanto o método pode testar o gerador de números primos até N. Os parâmetros que devem ser passados para o método são a lista de números primos que é esperada e o número máximo até onde a lista vai gerar o número primo, ou seja, se o número máximo for 15 a lista esperada que o gerador retorne é "2, 3, 5, 7, 11, 13".

O processo de testar e codificar a funcionalidade é repetido até que a funcionalidade atenda aos requisitos estabelecidos no planejamento. Portanto, para resolver o problema de desenvolver um algoritmo que retorne uma lista de número primos até um valor N, o processo de criação dos testes deve ser executado até que a funcionalidade fique pronta, sendo que, sempre que necessário, o código deve ser refatorado. A Figura 32 apresenta o algoritmo implementado e refatorado.

```
1  public String gerarPrimosAte(int valorMaximo)
2  throws ValorMaximoInvalidoException {
3      if (valorMaximo >= MENOR_PRIMO) {
4          return numerosPrimos(valorMaximo);
5      } else {
6          throw new ValorMaximoInvalidoException();
7      }
8  }
9  private String numerosPrimos(int valorMaximo) {
10     boolean [] ehPrimo = inicializaListaCandidatos(valorMaximo);
11     for (int valor = MENOR_PRIMO; valor <= valorMaximo; valor++) {
12         if (ehPrimo[valor]) {
13             for (int naoPrimos = MENOR_PRIMO * valor;
14             naoPrimos <= valorMaximo; naoPrimos += valor) {
15                 ehPrimo[naoPrimos] = false;
16             }
17         }
18     }
19     return apresentaResultado(valorMaximo, ehPrimo);
20 }
21 private String apresentaResultado(int valorMaximo, boolean[] ehPrimo)
22     String resultado = String.valueOf(MENOR_PRIMO);
23     for (int i = MENOR_PRIMO + 1; i <= valorMaximo; i++) {
24         if (ehPrimo[i]) {resultado += ", " + i;}
25     }
26     return resultado;
27 }
28 boolean[] inicializaListaDePrimosPotenciais(int valorMaximo) {
29     boolean [] resultado = new boolean[valorMaximo + 1];
30     resultado[0] = resultado [1] = false;
31     for (int i = MENOR_PRIMO; i < resultado.length; i++) {
32         resultado[i] = true;
33     }
34     return resultado;
35 }
```

Figura 32 – Algoritmo GerarPrimos()

A Figura 32 ilustra o código do algoritmo que gera uma lista de números primos. Este código foi desenvolvido utilizando a linha de pensamento do TDD, no qual testes foram escritos e compilados depois de falharem e a funcionalidade foi desenvolvida para atender ao teste. Sempre que necessário foi feito o refatoramento do código, tornando-o simples e sem duplicações.

Com o exemplo descrito nesta seção, o fluxo de atividades do TDD podem ser melhor entendidas, sendo que o código da Figura 32 está testado e o programador pode

agora passar para a próxima funcionalidade do sistema que também seguirá o mesmo fluxo de atividades até que esteja pronta e consequentemente testada.

3 MATERIAIS E MÉTODOS

Nesta seção serão apresentados os materiais e métodos utilizados na realização desse trabalho.

3.1 Local e Período

Esse trabalho foi desenvolvido no Complexo de Informática do CEULP/ULBRA e na residência do autor, ambos localizados na cidade de Palmas - TO. O desenvolvimento do trabalho compreendeu o período de fevereiro a junho de 2013, período em que foi definida a proposta e desenvolvida a revisão de literatura, e agosto a dezembro de 2013, período em que foram realizadas análises e desenvolvida a proposta.

3.2 Materiais

Os materiais utilizados para realização desse trabalho foram trabalhos científicos, tais como livros, artigos, monografias, teses e dissertações. Grande parte do material foi coletado no meio eletrônico. Também utilizou-se os softwares Visual Studio 2012 para o desenvolvimento, o Microsoft Visio 2010 para a elaboração de parte dos artefatos e o Microsoft Word 2013 para o restante.

3.3 Metodologia

O projeto teve início com a busca e o estudo de referencial teórico com foco nos temas que envolvem o trabalho - Análise de Pontos de Teste (APT) e Desenvolvimento Orientado a Teste (TDD). Os primeiros estudos realizados foram feitos a fim de ajudar a reunir informações sobre o mercado de desenvolvimento de software e sua relação com a qualidade dos produtos desenvolvidos. Para obter melhor embasamento foram feitos estudos sobre teste de software e técnicas de medição para melhor compreender a abordagem do TDD.

Em seguida foi realizado um estudo sobre Análise de Pontos de Teste e a metodologia de desenvolvimento TDD. O conhecimento obtido com os estudos realizados permitiu que fosse possível vislumbrar algumas das possíveis relações entre os temas estudados. Também serviu como base para a realização das próximas etapas do projeto.

O projeto foi executado no segundo semestre letivo do ano de 2013 e uma das primeiras atividades que foram realizadas nessa etapa foi o refinamento dos estudos realizados e eventuais aprofundamentos, caso houvesse necessidade. A proposta desenvolvida tem a intensão de proporcionar um desenvolvimento de software a partir da metodologia TDD e um melhor planejamento dos testes desenvolvidos a partir da técnica Análise de Pontos de Testes. Para tanto, foram definidas formas gráficas para representar os artefatos, atividades e papéis, sendo utilizadas para a construção da representação gráfica da proposta deste trabalho. O trabalho foi dividido em três momentos: a concepção do projeto, a APT e o desenvolvimento com TDD, como ilustrado na Figura 33.

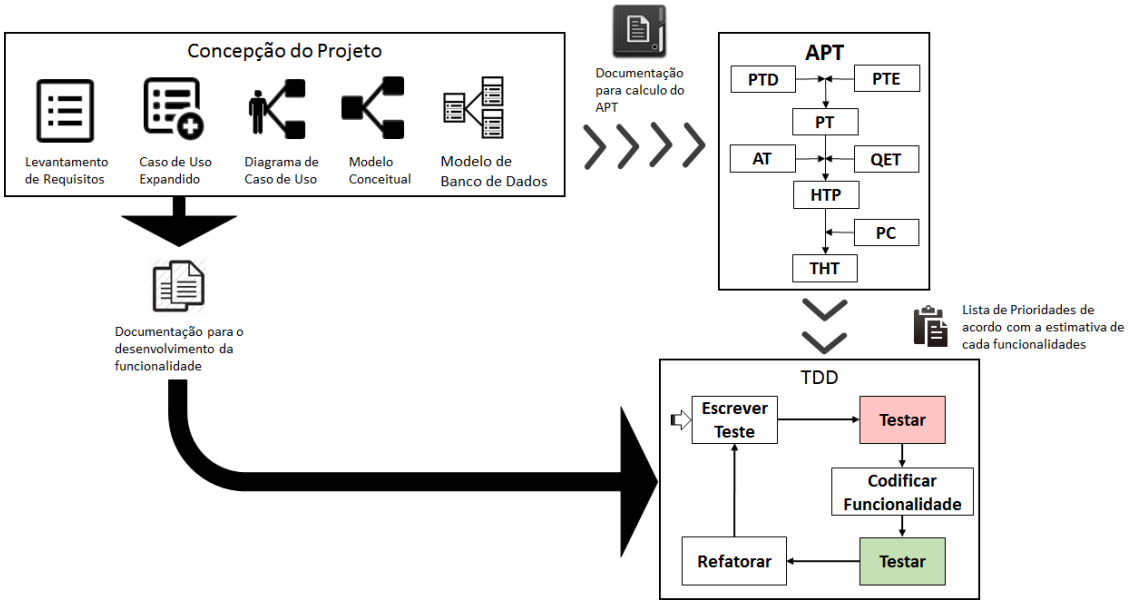


Figura 33 – Fluxo de processo da proposta da utilização em conjunto do APT e do TDD

A Figura 33 ilustra a utilização da técnica APT em projetos que utilizam a metodologia TDD no seu desenvolvimento, sendo que a proposta foi dividida em três momentos que são a concepção do projeto, a estimativa utilizando a técnica APT e o desenvolvimento utilizando a metodologia TDD.

A concepção do projeto envolve o levantamento de requisitos, a concepção do projeto por meio dos artefatos. A segunda parte consiste nas estimativas utilizando a técnica APT, bem como na definição das prioridades de desenvolvimento. O terceiro momento é o de desenvolvimento, baseado no TDD, iniciando com a elaboração dos testes, sendo que o primeiro teste deve falhar, o desenvolvimento da funcionalidade, o refatoramento e só então o desenvolvedor passa para as próximas funcionalidades.

3.4 Domínio

Para exemplificar a utilização da proposta utilizou-se um projeto elaborado na disciplina de Tópicos Especiais em Sistemas de Informação 4, que teve, em 2013/2, como primeiro conteúdo programático, a Análise de Pontos de Função (APF). O sistema modelado na disciplina tem o nome de KET-KAT e foi modelado pelos alunos Mário Almeida Rodrigues, Willian Almeida Rodrigues e Gabriela Fachini Brito. Para realizar a APF do sistema, o grupo elaborou os seguintes artefatos: Levantamento de Requisitos, Caso de Uso Expandido, Modelo Conceitual e Modelo de Banco de Dados. O KET-KAT é um sistema que recebe informações de atletas, times e partidas de basquete, realiza o processamento e gera indicadores de acordo com a necessidade do usuário.

4 RESULTADO

Nesta seção é apresentado o desenvolvimento da metodologia proposta, utilizando como base o referencial teórico deste trabalho. Para iniciar a apresentação do processo foi elaborado um fluxo, Figura 34, que ilustrou as atividades propostas neste trabalho, considerando a utilização do APT e o TDD em conjunto. Em seguida é apresentada uma exemplificação da utilização da técnica em conjunto, através do domínio apresentado na seção 3.4.

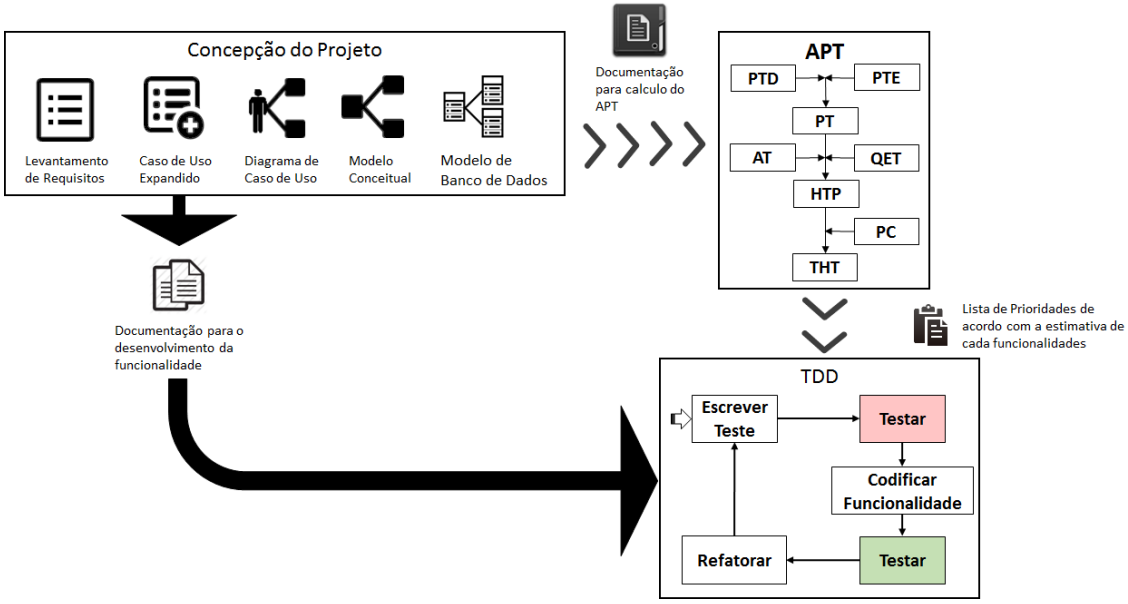


Figura 34 – Fluxo da Metodologia

A Figura 34 apresenta o fluxo da metodologia que foi proposta, utilizando a APT para realizar as estimativas e TDD como metodologia de desenvolvimento. O fluxo foi dividido em três partes, sendo que a primeira parte é a concepção do projeto, na qual as entrevistas são realizadas e é feito o levantamento de requisitos, a partir disto a equipe de modelagem elaborará os artefatos definidos na proposta. A segunda parte é a APT que utilizará como principal base para o cálculo os pontos de função. A terceira e última parte é o desenvolvimento utilizando o TDD, no qual a equipe de desenvolvimento/teste irá utilizar os artefatos para desenvolver os testes e as funcionalidades do projeto.

As próximas seções apresentarão papéis dentro da proposta deste trabalho, além das responsabilidades e importância dentro do processo proposto.

4.1 Papéis

Nas técnicas que foram utilizadas para a elaboração desta proposta não são definidos os papéis necessários para a execução do projeto, entretanto, nesta proposta definiram-se papéis, atribuindo atividades e a importância de cada um dentro da proposta.

4.1.1 Cliente

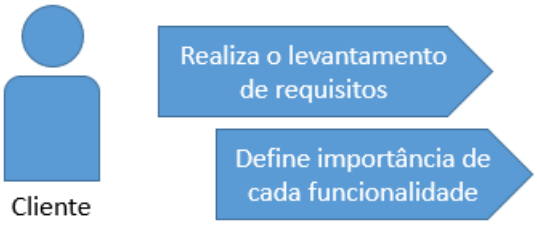


Figura 35 – Atividades do Cliente

A Figura 35 apresenta as atividades do cliente, ele é, na maioria das vezes, quem procura a empresa desenvolvedora de software para demonstrar seu interesse em um sistema que atenda suas necessidades. Nestes casos, é ele que vai apontar suas exigências para que o sistema possa ser modelado e desenvolvido. As necessidades do cliente são identificadas em reuniões que devem ser devidamente formalizadas de forma que possam ser usadas como apoio na modelagem do sistema.

4.1.2 Gerente de Projeto

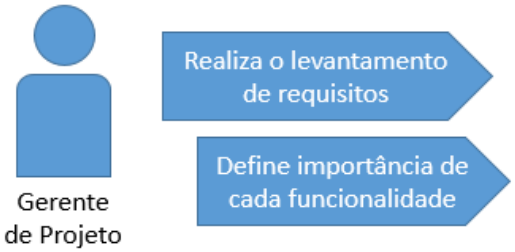


Figura 36 – Atividades do Gerente

A Figura 36 apresenta as atividades do gerente, que além destas, também monitora a execução todas as atividades da proposta. Ele é um papel importante dentro da utilização da proposta, sendo que é quem fica responsável por gerenciar prazos e custos do projeto, e para isto, ele deve utilizar como apoio a sua gerencia a APT, que aponta uma estimativa de horas de testes. Participa ativamente dos contatos com o cliente e tem conhecimento de todas as necessidades do cliente para que possa ser

repassada para a equipe de modelagem, além de conhecer toda a equipe e o ambiente de desenvolvimento.

4.1.3 Analista de Negócios

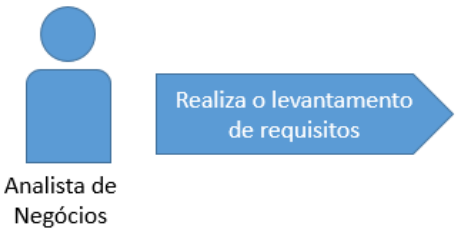


Figura 37 – Atividades do Analista de Negócios

A Figura 37 apresenta as atividades do analista de negócios, nas quais o seu papel é de realizar as entrevistas com o cliente, com o intuído de levantar de requisitos de acordo com a necessidade do cliente. Durante a entrevista, o analista deve esclarecer que a proposta de desenvolvimento de software necessita de informações essenciais que devem ser formalizadas para fornecer dados para a equipe de modelagem. O analista de negócio é o contato direto com o cliente, ele deve obter informações das necessidades do cliente e ainda obter informações que serão usadas na APT, que são as informações que foram, de acordo com a proposta, adicionadas ao documento de levantamento de requisitos, sendo elas a importância que o usuário atribui a uma funcionalidade e a intensidade de seu uso. Ele também é responsável por orientar que a intensidade de uso da funcionalidade deve ser medida por período de tempo, por exemplo, a funcionalidade editar usuário pode ser usada quatro vezes ao dia, e então, definir se essa quantidade é alta ou não de acordo com a funcionalidade.

4.1.4 Equipe de Modelagem

Figura 38 – Atividades da Equipe de Modelagem

A Figura 38 apresenta as atividades da equipe de modelagem. A equipe de modelagem é responsável por elaborar os artefatos do projeto de acordo com o

levantamento de requisitos e a descrição feita pelo analista de negócios. Os documentos elaborados devem seguir o padrão de acordo com a proposta elaborada neste trabalho, os documentos foram estabelecidos de forma a atender necessidades de estimativa e desenvolvimento, ou seja, o padrão de artefatos que ficou definido neste projeto atende os requisitos necessários para realizar a APT e o desenvolvimento do software seguindo a metodologia TDD.

4.1.5 Equipe de Desenvolvimento/Teste

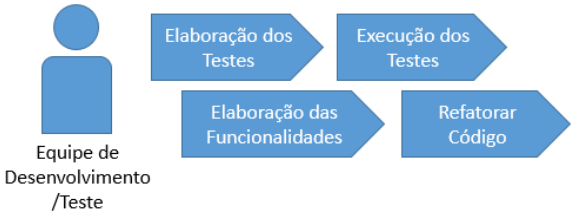


Figura 39 – Atividade da Equipe de Desenvolvimento/Teste

A Figura 39 apresenta as atividades da equipe de desenvolvimento/teste, que é responsável por desenvolver os testes e as funcionalidades, bem como testá-las e refatorá-las. Para elaborar os testes e as funcionalidades a equipe deve utilizar os artefatos e se preocupar em seguir as documentações elaboradas pela equipe de modelagem.

4.2 Atividades

As atividades foram divididas em três partes, conforme apresentado na Figura 34. A primeira parte do fluxo é a concepção do projeto. A segunda é a estimativa utilizando o APT e a terceira é o desenvolvimento e teste do sistema utilizando a metodologia TDD.

4.2.1 Concepção do Projeto

Figura 40 – Concepção do Projeto

A Figura 40 apresenta a primeira parte do fluxo de atividades, que se inicia com as entrevistas do Analista de Negócios ao cliente, na qual foi elaborado e documentado o levantamento de requisitos. Este é o primeiro artefato, sendo de suma importância para o projeto, pois é ele que traz todas as necessidades do cliente e ainda informações para a APT. A partir do levantamento de requisitos a equipe de modelagem produziu os artefatos de Caso de Uso Expandido, Diagrama de Caso de Uso, Modelo Conceitual e Modelos de Banco de Dados seguindo o padrão definido na proposta de utilização das técnicas de APT e TDD em conjunto. Na seção 4.2.1.1 serão apresentados os artefatos produzidos.

4.2.1.1 Artefatos

Os artefatos utilizados seguiram o modelo UML e os definidos como essenciais a serem produzidos e utilizados como base para a estimativa e para o desenvolvimento de um sistema. São eles: levantamento de requisitos, caso de uso expandido, diagrama de caso de uso, modelo conceitual e o modelo de banco de dados.

O levantamento de requisitos apresenta as funcionalidades que serão desenvolvidas no projeto. A Figura 41 apresenta o levantamento de requisitos do projeto. Neste artefato de exemplo foram definidos dezoito requisitos para atender as necessidades do cliente.

Levantamento de Requisitos

Cliente: Empreendedor Empreende **Levantamento Nº:** 001/2013

Data: 17/09/2013

Responsável pelo Levantamento: Mário Almeida

Nº	Cod. Requisito	Requisito	Importância	Intensidade de Uso
1	U1	Criar Usuário	( )Baixa ( )Normal (X)Alta	( )Baixa (X)Normal ( )Alta
2	U2	Editar Usuário	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
3	U3	Deletar Usuário	(X)Baixa ( )Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
4	A1	Criar Atleta	( )Baixa ( )Normal (X)Alta	( )Baixa (X)Normal ( )Alta
5	A2	Editar Atleta	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
6	A3	Deletar Atleta	(X)Baixa ( )Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
7	A4	Pesquisar Atleta	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
8	T1	Criar Time	( )Baixa ( )Normal (X)Alta	( )Baixa (X)Normal ( )Alta
9	T2	Editar Time	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
10	T3	Pesquisar Time	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
11	T4	Deletar Time	(X)Baixa ( )Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
12	P1	Iniciar Partida	( )Baixa ( )Normal (X)Alta	( )Baixa ( )Normal (X)Alta
13	P2	Finalizar Partida	( )Baixa ( )Normal (X)Alta	( )Baixa ( )Normal (X)Alta
14	P3	Pesquisar Partida	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
15	P4	Editar Partida	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
16	E1	Criar Estatística	( )Baixa ( )Normal (X)Alta	( )Baixa ( )Normal (X)Alta
17	E2	Editar Estatística	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
18	E3	Pesquisar Estatística	( )Baixa (X)Normal ( )Alta	( )Baixa (X)Normal ( )Alta

_____

Cliente

_____

Responsável pelo Levantamento

Palmas, 17 de Setembro de 2013

Figura 41 – Levantamento de Requisitos

A Figura 41 apresenta o levantamento de requisitos que foi elaborado no projeto. Para que este documento atendesse as necessidades desta proposta foram adicionadas duas colunas, sendo elas a “Importância do Requisito”, na qual o usuário deve informar qual a importância de ele atribui para cada funcionalidade, e ainda a “Intensidade de Uso” que é utilizada para que o cliente indique a quantidade de vezes que a

funcionalidade será utilizada em um período de tempo. O próximo artefato é o caso de uso expandido, conforme Figura 42.

Caso de Uso Expandido

Cliente: Empreendedor Empreende **CUE Nº:** 001/2013
Data: 17/09/2013

Responsável: Mário Almeida

U1 - Criar Usuário	
Atores	Administrador
Responsabilidade	Criar um novo usuário no sistema.
Descrição	O administrador acessa a parte de cadastro de usuário, preenche o formulário e envia.

Fluxo 1 – Administrador	
Ações do Ator	**Respostas do Sistema**
1. O Administrador acessar a parte de casdastro de usuario.	2. O sistema exibe a área de cadastro de usuario.
3. O Administrador preenche o formulário com os dados solicitados e envia.	4. O sistema informa que o cadastro foi realizado com sucesso.

Tratamento de Exceções
Passo 4 - dados não preenchidos. O sistema emite alerta e retorna para o passo 3.

Cliente

Responsável

Palmas, 17 de Setembro de 2013

Figura 42 – Caso de Uso Expandido

A Figura 42 apresenta um dos casos de uso expandido que foi elaborado na modelagem do sistema. Este artefato contém informações que irão auxiliar a equipe de desenvolvimento/teste. O campo ator deve conter apenas os atores que poderão executar a funcionalidade, a parte “Fluxo 1” contêm a sequência de ações realizadas pelo atores e as respostas do sistema, sendo que estas informações são utilizadas no desenvolvimento para definir, por exemplo, o que o sistema irá apresentar quando o usuário faz solicitações através da interface do sistema. O próximo artefato é o diagrama de caso de uso, Figura 43.

Diagrama de Caso de Uso

Cliente: Empreendedor Empreende **DCU Nº:** 001/2013
Data: 17/09/2013

Responsável: Mário Almeida

Cliente

Responsável

Palmas, 17 de Setembro de 2013

Figura 43 – Diagrama de Caso de Uso

A Figura 43 apresenta o Diagrama de Caso de Uso que foi elaborado na modelagem do sistema. Ele descreve um cenário que mostra as funcionalidades do sistema em uma visão geral e a relação destas funcionalidades com os atores do sistema.

Este artefato auxilia o cliente a entender melhor o sistema em uma visão global. A Figura 44 apresenta o artefato modelo conceitual.

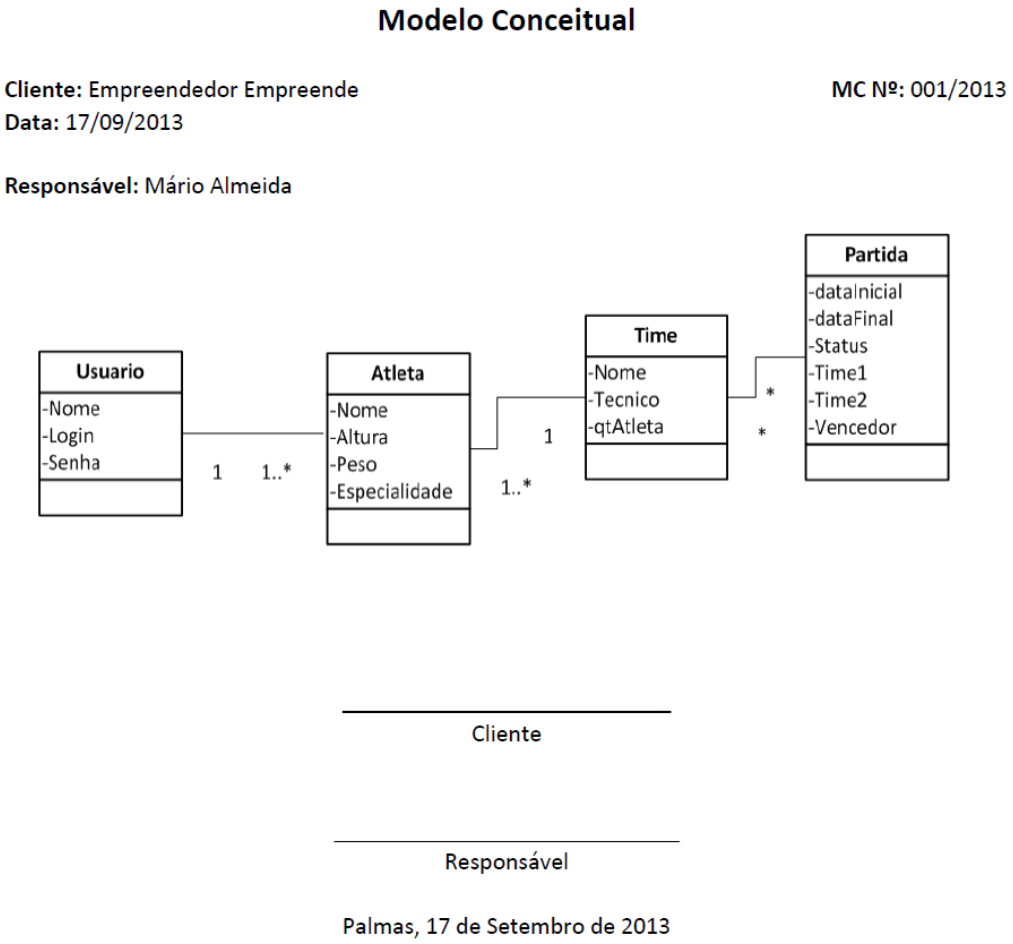


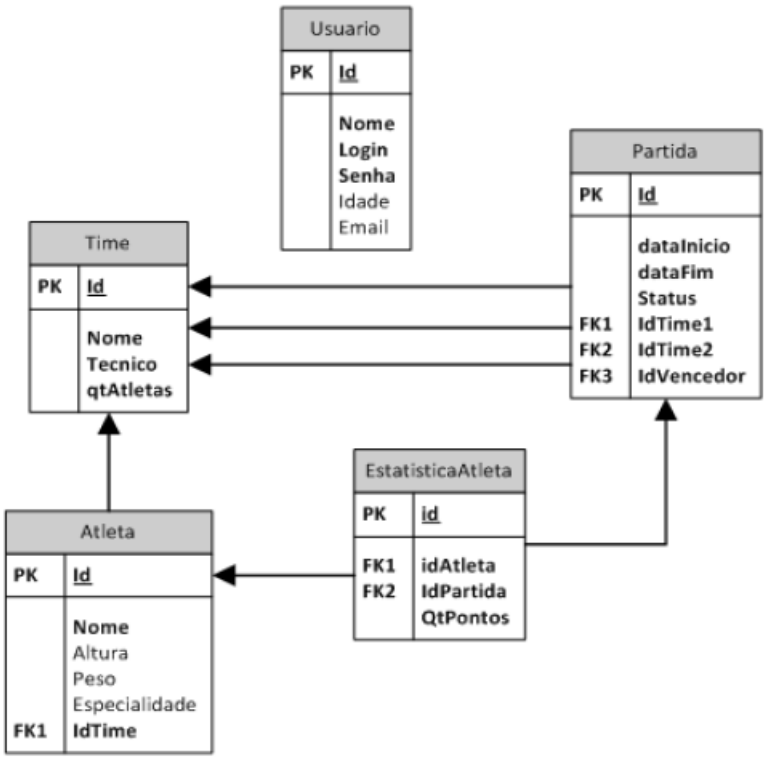
Figura 44 – Modelo Conceitual

A Figura 44 apresenta o modelo conceitual que demostra como os dados serão estruturados na visão do cliente. Neste artefato deve conter a estrutura de dados que são relevantes para o cliente, ele fornece dados para que na fase de desenvolvimento a equipe saiba quais os dados que devem ser exigidos ou apresentados para o cliente. A Figura 45 apresenta o artefato modelo de banco de dados.

Modelo de Banco de Dados

Cliente: Empreendedor Empreende **MR Nº:** 001/2013
Data: 17/09/2013

Responsável: Mário Almeida



Cliente

Responsável

Palmas, 17 de Setembro de 2013

Figura 45 – Modelo de Banco de Dados

A Figura 45 apresenta o modelo de banco de dados. Ele apresenta como os dados serão estruturados no banco de dados e as chaves primárias e estrangeiras do banco de dados.

Com a elaboração de todos os artefatos, a fase de concepção do projeto foi finalizada. A próxima atividade é a estimativa utilizando a APT e esta atividade é de responsabilidade do gerente, sendo que se necessário, ele poderá solicitar o auxílio ao analista de negócios, ou de algum membro da equipe de modelagem ou de desenvolvimento/teste. O gerente deve conhecer as necessidades do cliente, os artefatos produzidos, o ambiente de desenvolvimento e os membros da equipe para poder realizar a APT.

4.2.2 Estimativa utilizando APT

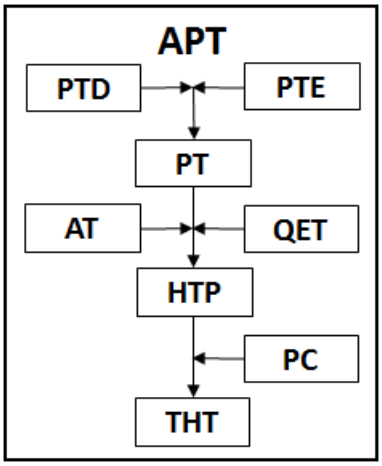


Figura 46 – Análise de Pontos de Testes

A Figura 46 apresenta a segunda parte do fluxo de atividades, definido na Figura 34. A Figura 46 detalha o passo a passo do APT. Com os artefatos produzidos se inicia a parte de estimativas do projeto. A APT utiliza como base a Análise de Pontos de Função. A Tabela 1 apresenta a contagem de pontos de função do domínio utilizado na exemplificação deste projeto.

Tabela 1 – Contagem de Pontos de Função

Descrição	Tipo	TD	AR	Complexidade	PF
Criar Usuário	EE	3	1	Baixa	3 PF
Editar Usuário	CE	3	1	Baixa	3 PF
Deletar Usuário	EE	2	1	Baixa	3 PF
Criar Atleta	EE	4	1	Baixa	3 PF
Editar Atleta	CE	4	1	Baixa	3 PF
Deletar Atleta	EE	2	1	Baixa	3 PF
Pesquisar Atleta	CE	3	1	Baixa	3 PF
Criar Time	EE	5	1	Baixa	3 PF
Editar Time	CE	5	1	Baixa	3 PF
Pesquisar Time	CE	3	1	Baixa	3 PF
Deletar Time	EE	2	1	Baixa	3 PF
Iniciar Partida	EE	8	1	Baixa	3 PF
Finalizar Partida	EE	8	1	Baixa	3 PF
Pesquisar Partida	SE	3	1	Baixa	4 PF
Editar Partida	EE	8	1	Baixa	3 PF
Criar Estatística	EE	5	1	Baixa	3 PF
Editar Estatística	EE	5	1	Baixa	3 PF
Pesquisar Estatística	SE	3	1	Baixa	4 PF
TOTAL					56 PF

A Tabela 1 apresenta a contagem de pontos de função. Para a contagem de Pontos de Função (PF) foram utilizados os artefatos apresentados na seção 4.2.1.1. Na Tabela 1 a contagem de PF está dividida por cada funcionalidade, totalizando cinquenta e seis PF, na somatória de todos os pontos de função.

A primeira parte técnica da APT que o gerente deve realizar é o cálculo dos pontos de testes. Para realizar o cálculo dos pontos de testes foi utilizada a fórmula apresentada na técnica, seção 2.5.3. A técnica define que os pontos de testes são divididos em dois fatores principais, os pontos de testes dinâmicos e pontos de testes estáticos.

Para definir os pontos de testes dinâmicos devem ser considerados o fator dinâmico, a qualidade dos requisitos e os pontos de funções. A Tabela 2 apresenta os valores definidos e obtidos a partir da utilização do cálculo dos pontos de testes.

Tabela 2 – Pontos de Teste Dinâmicos

Fator Dinâmico						
Descrição	**Ue**	**Uy**	**I**	**C**	**U**	**FDf**
Criar Usuário	12	4	2	3	1	1,05
Editar Usuário	6	2	2	3	1	0,65
Deletar Usuário	3	2	2	3	1	0,5
Criar Atleta	12	4	2	3	1	1,05
Editar Atleta	6	2	2	3	1	0,65
Deletar Atleta	3	2	2	3	1	0,5
Pesquisar Atleta	6	2	2	3	1	0,65
Criar Time	12	4	2	3	1	1,05
Editar Time	6	2	2	3	1	0,65
Pesquisar Time	6	2	2	3	1	0,65
Deletar Time	3	2	2	3	1	0,5
Iniciar Partida	12	8	2	3	1	1,25
Finalizar Partida	12	8	2	3	1	1,25
Pesquisar Partida	6	2	2	4	1	0,7
Editar Partida	6	2	2	3	1	0,65
Criar Estatística	12	8	2	3	1	1,25
Editar Estatística	6	2	2	3	1	0,65
Pesquisar Estatística	6	4	2	4	1	0,8

A Tabela 2 apresenta o cálculo do fator dinâmico de acordo com a técnica da APT. A segunda coluna (Ue) representa a importância que o usuário atribui para a funcionalidade, a terceira coluna (Uy) é a intensidade de uso da funcionalidade. Estas duas primeiras colunas são obtidas pelo Analista de Negócios, nas entrevistas com o cliente, que são devidamente documentadas no levantamento de requisitos, como mostrado na Figura 35. A quarta coluna (I) é o fator interface, a quinta coluna (C) o fator complexidade, a sexta coluna (U) é o fator uniformidade e a sétima coluna (FDf) é o fator dinâmico. A fórmula abaixo exemplifica como foram obtidos os valores da coluna FDf:

$$FDf = \left(\frac{Ue+Uy+I+C}{20}\right) \cdot U = \left(\frac{12+4+2+3}{20}\right) \cdot 1 = 1{,}05$$

Fórmula 13 – Cálculo do fator dinâmico

A Fórmula 13 apresenta o cálculo do fator dinâmico da funcionalidade ‘Criar Usuário’ que, de acordo com a Tabela 2, o resultado da Fórmula 13 é o valor de 1,05

para fator dinâmico. Este valor está na sétima coluna da Tabela 2 na linha da respectiva funcionalidade. Depois de obter o valor do fator dinâmico para cada funcionalidade foi obtido o valor da qualidade dinâmica utilizando a fórmula apresentada na seção 2.5.1.2, de acordo com a técnica APT.

$$F = \frac{(VT) \bullet P}{4} = \frac{(3) \bullet 0{,}75}{4} = 0{,}5625$$

Fórmula 14 – Cálculo dos Fatores Explícitos

A Fórmula 14 apresenta o cálculo das características explicitas (CE), funcionalidade, na qual, foi atribuído o valor 3 para VT e multiplicado pelo seu peso de acordo com a técnica, seção 2.5.1.2, e dividido por quatro. Os valores obtidos são apresentados na Tabela 3, e foram utilizados no cálculo da qualidade dinâmica.

Tabela 3 – Valores para Características Explicitas

Característica Explicita	**Valor**	**Peso**	**Total**
Funcionalidade	3	0,75	0,5625
Desempenho	0	0,10	0
Segurança	0	0,05	0
Aderência	3	0,10	0,075
Total			**0,6375**

A Tabela 3 apresenta os valores considerados para todos os fatores relevantes para o cálculo da qualidade dinâmica. Sendo que, foi considerado para a funcionalidade o valor 3 de acordo com tabela definida pela técnica, seção 2.5.1.2, o valor foi multiplicado pelo seu peso e obteve-se 0,5625, e para a característica aderência também foi atribuído o valor 3 que multiplicado pelo seu peso obteve-se o valor de 0,075, portanto, o valor calculado para características explicitas foi de 0,6375. A Tabela 4 apresenta os valores atribuídos para as características implícitas (CI).

Tabela 4 - Valores para Características Implícitas

Característica Implícita	**Valor**
Funcionalidade	1
Desempenho	0
Segurança	0
Aderência	1
Total	**0,04**

QRd	**0,6775**

A Tabela 4 apresenta os valores atribuídos para as características implícitas, na qual foi considerada apenas funcionalidade e aderência, pois foram os únicos também considerados nas características explicitas. Para as característica implícitas foi obtido o valor de 0,04, utilizando a fórmula apresentada na seção 2.5.1.2, para calcular, de acordo com a técnica APT. O valor para qualidade dinâmica e a soma das duas características, explicita e implícita, com isto o valor é de 0,6775, este valor foi utilizado no cálculo dos pontos de teste. A Fórmula 15 apresenta o cálculo que foi feito para obter o valor de PTD para a funcionalidade Criar Usuário.

$$PTD = PF \bullet FDf \bullet QRd = 3 \bullet 1{,}05 \bullet 0{,}6775 = 2{,}13413$$

Fórmula 15 – Cálculo de Pontos de Teste Dinâmico

A Fórmula 15 apresenta o cálculo que foi realizado para cada funcionalidade, na qual, o valor 3 é o de pontos de função, o valor 1,05 é o fator dependente e o valor 0,6775 é a qualidade dinâmica. A Tabela 6 apresenta os valores obtidos de pontos de testes dinâmicos utilizando a Fórmula 15, apenas substituindo os valores de pontos de função e de fator dependente para os valores respectivos da funcionalidade medida.

Tabela 5 – Valores para pontos de testes dinâmicos

Pontos de Teste Dinâmicos			
Descrição	**PF**	**FDf**	**PTD**
Criar Usuário	3	1,05	2,13413
Editar Usuário	3	0,65	1,32113
Deletar Usuário	3	0,5	1,01625
Criar Atleta	3	1,05	2,13413
Editar Atleta	3	0,65	1,32113
Deletar Atleta	3	0,5	1,01625
Pesquisar Atleta	3	0,65	1,32113
Criar Time	3	1,05	2,13413
Editar Time	3	0,65	1,32113
Pesquisar Time	3	0,65	1,32113
Deletar Time	3	0,5	1,01625
Iniciar Partida	3	1,25	2,54063
Finalizar Partida	3	1,25	2,54063
Pesquisar Partida	4	0,7	1,897
Editar Partida	3	0,65	1,32113
Criar Estatística	3	1,25	2,54063
Editar Estatística	3	0,65	1,32113
Pesquisar Estatística	4	0,8	2,168
TOTAL			30,3859

A Tabela 5 apresenta os valores que foram obtidos no cálculo de pontos de testes dinâmicos, no caso da funcionalidade Pesquisar Partida foi utilizado o valor de pontos de função 4, multiplicado pelo valor do fator dinâmico 0,7, multiplicado pelo valor da qualidade dinâmica, apresentado na Tabela 3, com o valor de 0,6775, portanto o valor de pontos de testes dinâmicos para a funcionalidade Pesquisar Partida é de 1,897. O próximo passo para calcular os pontos de testes é o de obter o valor de pontos de testes estáticos.

Como neste projeto não foram definidos testes de verificação então o valor dos testes estáticos é zero. A Fórmula 16 apresenta o cálculo dos pontos de testes estáticos.

$$PTE = 16 \bullet n = \ 16 \bullet 0 = 0$$

Fórmula 16 – Cálculo dos pontos de testes estáticos

A Fórmula 16 apresenta o cálculo dos pontos de testes estáticos, na qual, 16 é uma constante definida na técnica, seção 2.5.2, e zero é o para os testes estáticos. A Fórmula 17 apresenta o cálculo do total de pontos de testes.

$$PT = \sum PTD + \frac{(PF \bullet PTE)}{500} = 30{,}3859 + \frac{(56 \bullet 0)}{500} = 30{,}3859$$

Fórmula 17 – Cálculo do total de pontos de testes

A Fórmula 17 apresenta o cálculo para o total de pontos de testes que foi obtido utilizando os fatores já calculados nesta seção, sendo que, 30,3859 é a somatória dos pontos de testes dinâmicos, 56 é o valor de pontos de função de todo o projeto, 0 é o valor encontrado de pontos de testes estáticos e 500 é uma constante definida na técnica APT, seção 2.5.3. E como resultado da Fórmula 17 foi obtido o valor de aproximadamente trinta pontos de testes neste projeto.

Os Pontos de Testes (PT) foram usados para a próxima etapa da APT, que foi o cálculo das Horas Teste Primaria (HPT). Além dos PT outros fatores foram considerados para calcular a HTP, sendo eles, Qualidade da Equipe de Teste (QET) e Ambiente de Teste (AT).

A QET foi considerada baixa, pois, além de não existir nenhum histórico de trabalho da equipe para ser utilizado de base da produtividade, a equipe nunca trabalhou em conjunto em projetos. Para o fator AT os valores da Tabela 6, foram considerados.

Tabela 6 – Cálculo do Ambiente de Teste

Ferramenta de Teste	2
Teste de Procedência	8
Documentação de Teste	12
Ambiente de Desenvolvimento	2
Ambiente de Teste	4
Testware	4
Total	**32**

A Tabela 6 apresenta os valores que foram atribuídos para cada fator do AT, conforme tabela apresentada na técnica APT, seção 2.5.5, sendo que os valores foram atribuídos da seguinte forma:

a) Ferramenta de Teste: a ferramenta disponível para os testes deste projeto é apenas para a execução dos testes, portanto, o valor atribuído foi dois;
b) Teste de Procedência: não existe nenhum plano de precedência no projeto, portanto, o valor atribuído foi oito;
c) Documentação de Teste: não ficou definido nenhum padrão para a documentação dos testes, portanto, o valor atribuído foi doze;
d) Ambiente de Desenvolvimento: a linguagem que foi definida é uma linguagem de 4ª geração, portanto, o valor atribuído foi dois;
e) Ambiente de Teste: a equipe não trabalhou no ambiente e em nenhum outro similar, portanto, o valor atribuído foram quatro.
f) *Testware*: não existe material adicional de teste, como base de dados e tabelas, portanto, o valor atribuído foram quatro.

Com os valores atribuídos em cada fator foi utilizado a Fórmula 18 para realizar o cálculo da AT.

$$AT = \frac{STF}{21} = \frac{32}{21} = 1{,}523$$

Fórmula 18 – Cálculo do Ambiente de Teste

A Fórmula 18 apresenta o cálculo do AT de acordo com a técnica APT, seção 2.5.5, sendo que trinta e dois é o valor da somatória dos valores atribuídos aos fatores de acordo com a Tabela 7, e vinte e um é a constante definida na técnica. Portanto, o valor que foi obtido para AT é 1,523. Posteriormente ao realizar o cálculo do AT foi utilizado a Fórmula 19 para se conseguir as Horas de Teste Primárias.

$$HTP = 30{,}3859 \bullet 0{,}7 \bullet 1{,}523 = 32{,}39$$

Fórmula 19 – Cálculo das Horas de Teste Primárias

A Fórmula 19 apresenta o cálculo das horas de teste primárias, sendo que, o valor de 30,3859 é dos pontos de teste, 0,7 é o valor obtido para a qualidade da equipe e 1,523 é o valor obtido para o ambiente de teste e o resultado é de aproximadamente 32 horas primárias de teste. Entretanto segundo a técnica APT, o valor de horas primárias deve ser ajustado pelo índice de planejamento e controle. De Acordo com a Fórmula 20.

$$PC = 1 + (TE + FG) = 1 + (0{,}03 + 0{,}08) = 1{,}11$$

Fórmula 20 – Cálculo do índice de planejamento e controle

A Fórmula 20 apresenta o cálculo do índice de planejamento e controle, que considera os fatores tamanho da equipe, que foi atribuído 0,03, e se existe ferramenta de gerência, sendo que, não ficou definido nenhuma ferramenta de gerência, portanto o valor atribuído foi 0,08 de acordo com a técnica APT, seção 2.5.7. Como último passo do APT foi utilizado a Fórmula 20 para obter o total de horas de testes.

$$THT = HTP \bullet IPC = \ 32{,}39 \bullet 1{,}11 = 35{,}9577$$

Fórmula 20 – Cálculo do total de horas de testes

A Fórmula 20 apresenta o cálculo do total de horas de testes, na qual, o valor de horas primárias é multiplicado pelo índice de planejamento e controle e foi obtido o valor do total de horas de aproximadamente 36 horas para realizar a atividade de testes.

4.2.3 TDD

Durante o processo de desenvolvimento utilizando as técnicas APT e TDD em conjunto, após o gerente ter realizado as estimativas, inicia-se a fase de desenvolvimento do sistema. Neste trabalho, com o objetivo de exemplificar, o projeto do KET-KAT continuará a ser utilizado nesta seção.

Na fase de desenvolvimento, toda a documentação gerada na fase de modelagem é utilizada para desenvolver as funcionalidades. Nesta fase a equipe responsável deve utilizar os artefatos para codificar testes e métodos que atendam às exigências das funcionalidades.

A equipe utiliza o levantamento de requisitos para saber quais as funcionalidades devem ser desenvolvidas e os outros artefatos para elaborar testes, regras de acesso, organização de dados, estruturação dos dados e fluxo de navegação.

Dentro do TDD, após a escolha das prioridades das funcionalidades a serem desenvolvidas, o passo seguinte é a criação de um teste que irá testar as funcionalidades, para isto foi utilizada a ferramenta Microsoft Visual Studio 2012 e a linguagem C#.

Para ilustração da fase de desenvolvimento utilizando o TDD, a funcionalidade que foi testada é a de Criar Usuário, na qual os campos de preenchimento obrigatórios

são: Nome, Login e Senha. Os testes que foram realizados na funcionalidade são apenas para garantir que os três campos estejam preenchidos ao cadastrar um novo usuário. A Figura 47 apresenta o código do primeiro teste desenvolvido.

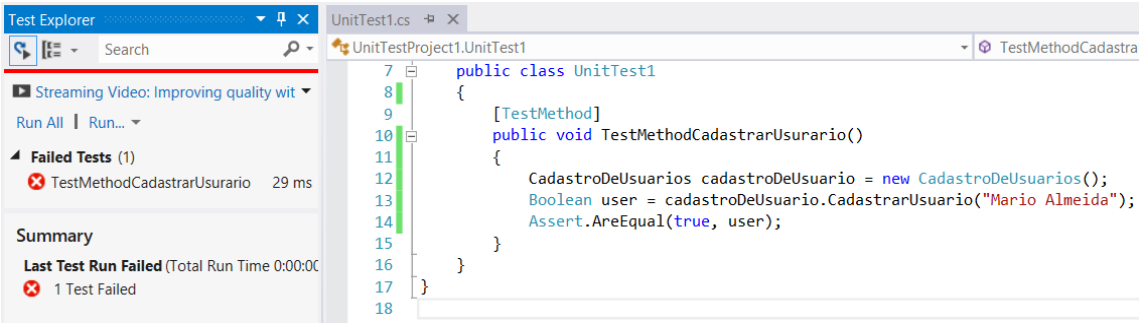


Figura 47 – Testar Cadastrar Usuário

A Figura 47 apresenta o código para testar o método de CadastrarUsuario(), sendo que este primeiro teste garante apenas que o nome do usuário não esteja vazio. Na linha 13 da Figura 40, o método CadastrarUsuario() é chamado e é passada como parâmetro uma *string*. Na linha 14 o teste verifica se o resultado esperado é igual ao valor retornado pelo método. No entanto, o método CadastrarUsuario() ainda não foi desenvolvido, então, o teste apresentou erro, como pode ser observado no lado esquerdo da Figura 40. O próximo passo foi desenvolver o método para que ele atenda ao teste TestMethodCadastrarUsurario(). A Figura 48 apresenta o método CadastrarUsuario() desenvolvido.

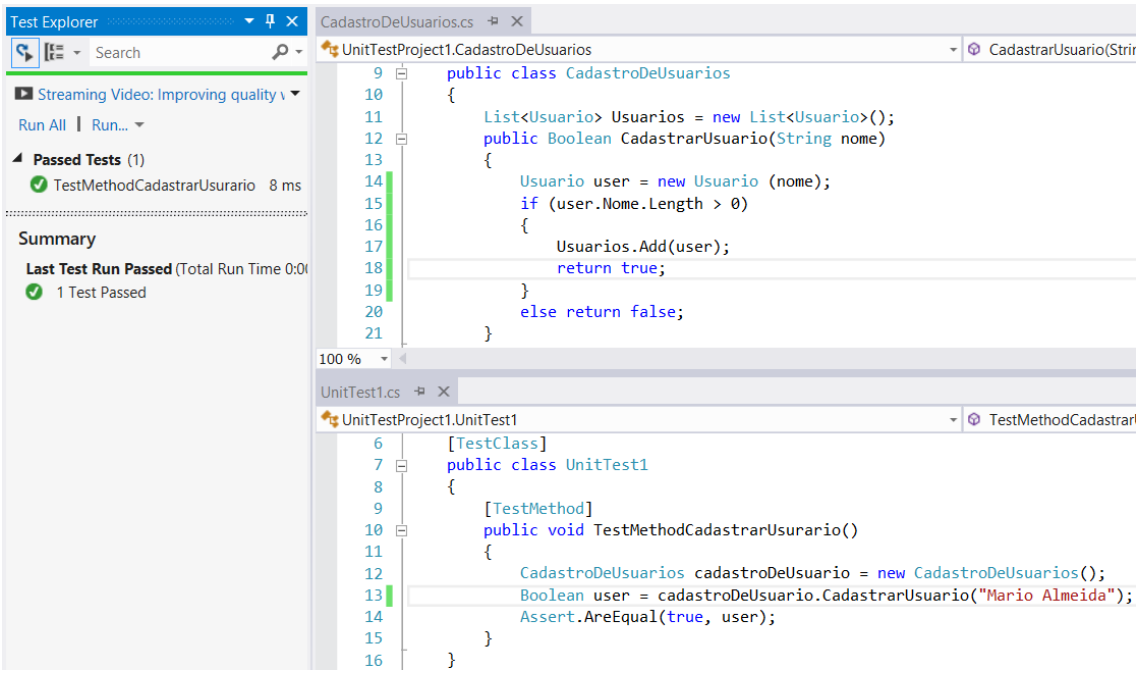


Figura 48 – Método CriarUsuario()

A Figura 48 apresenta o método CadastrarUsuario() desenvolvido e testado, no qual o método adiciona o nome do usuário. O teste recebe como parâmetro uma *string* e verifica se o parâmetro passado contém algum caractere, se houver, o método adiciona na List<> e retorna *true*, se não, retorna *false*. Neste caso, o método passou no teste, pois o parâmetro não é vazio. Os passos seguintes envolvem o desenvolvimento de testes que garantam que todos os campos estejam preenchidos no momento que o usuário for cadastrar outro usuário. A Figura 49 apresenta o código dos próximos testes desenvolvidos.

```
9         [TestMethod]
10        public void TestMethodCadastrarUsurario1()
11        {
12            CadastroDeUsuarios cadastroDeUsuario = new CadastroDeUsuarios();
13            Boolean user = cadastroDeUsuario.CadastrarUsuario("Mario Almeida");
14            Assert.AreEqual(true, user);
15        }
16        public void TestMethodCadastrarUsurario2()
17        {
18            CadastroDeUsuarios cadastroDeUsuario = new CadastroDeUsuarios();
19            Boolean login = cadastroDeUsuario.CadastrarUsuario("mario");
20            Assert.AreEqual(true, login);
21        }
```

Figura 49 – Testes sem Refatorar

A Figura 49 apresenta o código de dois testes que realizam praticamente a mesma ação na mesma funcionalidade, ou seja, o primeiro teste verifica se o nome não

pode ser vazio e o segundo se o login não pode ser vazio. Neste caso, o teste foi alterado para que em apenas um método de teste a funcionalidade possa ser testada sem que o código apresentasse duplicações desnecessárias. A Figura 50 apresenta o código da funcionalidade e do teste já refatoradas.

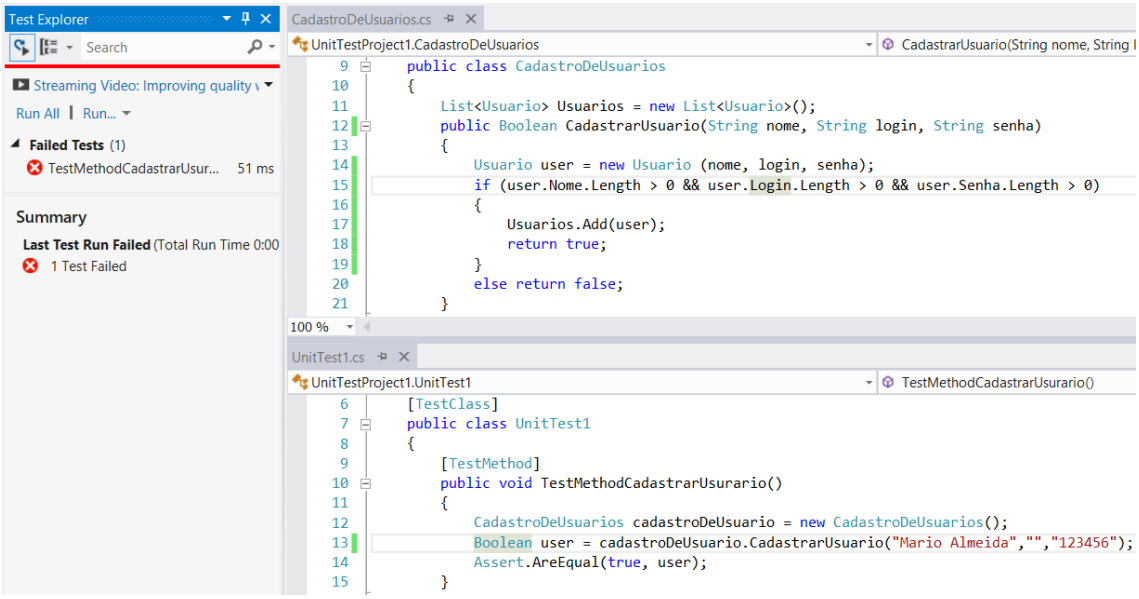


Figura 50 - Teste e método de CriarUsuario()

A Figura 50 apresenta o método CadastrarUsuario(), que foi desenvolvido de forma que os campos nome, login e senha tenham que ser sempre preenchidos para que o usuário seja cadastrado. Na Figura 50 o teste falhou, mas como pode ser observado na linha 14 do UnitTest1.cs, os parâmetros para o cadastro do usuário o campo correspondente ao login está vazio, ou seja, o teste confirmou que não é possível cadastrar um usuário com nenhum dos campos vazios.

O teste da Figura 50 foi executado novamente, mas com os campos corretamente preenchidos e assim o teste passou, ou seja, o método de cadastrar usuário está funcionando corretamente e não deixa que o usuário seja cadastrado com algum dos campos sem preencher.

Neste caso, foi necessário refatorar o código, pois contém duplicidades desnecessárias. Com a funcionalidade de cadastrar usuário pronto a equipe de desenvolvimento/teste poderia passar para a próxima funcionalidade seguindo os passos descritos nesta seção, ou seja, criar o teste, o teste falhar, desenvolver a funcionalidade e testar novamente assim, a funcionalidade passa no teste e uma outra funcionalidade pode ser desenvolvida.

Para a utilização da APT e do TDD, alguns processos foram definidos para que as técnicas pudessem ser utilizadas em conjunto, e ainda garantir um conjunto mínimo de atividades e artefatos para gerar um produto de software. A principal definição foi a elaboração de um fluxo de atividades, Figura 34, na qual os processos foram representados graficamente, configurando a proposta do trabalho.

O fluxo ficou dividido em três partes principais no qual a primeira é a concepção do projeto. Nesta fase a equipe de modelagem elabora os artefatos que devem seguir o padrão definido na proposta. Os artefatos devem ser baseados nos documentos gerados pelo analista de negócios durante as entrevistas. A equipe de modelagem deve ter acesso às documentações da entrevista e ainda solicitar reuniões com o gerente e com o analista de negócios para sanar possíveis dúvidas relacionadas ao projeto.

O gerente utiliza informações da equipe, do ambiente de desenvolvimento e os artefatos produzidos para realizar a APT, os artefatos foram definidos para que no momento que o gerente for realizar a APT, as informações necessárias fornecidas pelo cliente estejam documentas e de fácil acesso ao gerente. Utilizar o padrão de artefatos definidos pode garantir a agilidade ao realizar a APT.

A APT realizada pelo gerente pode ser utilizada para que uma lista de prioridades de funcionalidades a serem desenvolvidas seja estabelecida. Entretanto, neste trabalho, devido à ausência de um cliente, ficou definido que seriam seguidas as prioridades estabelecidas na documentação Levantamento de Requisitos.

Seguindo o fluxo de atividades definido nesta proposta, o passo seguinte foi o desenvolvimento das funcionalidades do sistema. Neste caso, apenas umas funcionalidade foi desenvolvida e testada, entretanto, foi utilizada a metodologia proposta, TDD, para primeiramente desenvolver um teste, no qual a funcionalidade não passou. Isso garante que o teste esteja funcionado, a partir disto a equipe de desenvolvimento/teste desenvolveu uma funcionalidade de criar usuário, sendo que, na documentação elaborada define que são utilizados três informações para cadastrar o usuário, sendo elas: nome, login e senha; e que nenhuma das três informações poderiam estar vazias. Com isto, a equipe criou um teste para garantir que o usuário não fosse cadastrado sem estas informações. Após a funcionalidade passar por este teste a equipe garante que a funcionalidade não poderá ser executada sem que todos os campos exigidos estejam preenchidos.

A utilização em conjunto da APT e do TDD proporcionou que uma estimativa com foco em teste, fosse realizada e que uma metodologia com foco também em teste fosse utilizada na parte de desenvolvimento. Está união de duas técnicas pode aumentar a qualidade dos softwares e tornar a atividade de teste uma parte obrigatória e que faça parte dos processos do desenvolvimento de um software.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou conceitos e técnicas de estimativas de software, bem como suas principais características e seus objetivos. Dentre as técnicas apresentadas, destaca-se a Análise de Pontos de Teste, por ter como foco principal a atividade de teste.

Dentre os conceitos da APT foram apresentados um passo a passo da utilização da técnica, processos e atividades, bem como o detalhamento das atividades, fatores utilizados para o cálculo, detalhamento de cada fator, e ainda tabelas e fórmulas definidas na APT.

Também foram apresentados conceitos da metodologia de desenvolvimento TDD. Sobre o TDD foram apresentados conceitos, seu fluxo de atividade, descrição das atividades, processos e boas práticas adotadas na metodologia.

Com os estudos realizados foi possível observar que a metodologia TDD não possui nenhuma atividade para realizar a estimativas de custos e prazos, então, com a intenção de agregar um processo de estimativa, foi utilizado o APT, que possui o foco em testes. As técnicas foram unidas, gerando um processo que, após se realizar a APT, inicia-se a utilização do TDD.

A união das duas técnicas tem a intensão de otimizar o processo de software, com a utilização da APT para definir recursos e prazos necessários para a conclusão do projeto. A ideia é utilizar uma metodologia que foca nos testes para reduzir a quantidades de falhas do software e ainda cumprir prazos com menor custo.

A união das técnicas foi representada graficamente, demonstrando os processos, papéis, fluxo de atividades e ainda os artefatos utilizados na proposta deste trabalho. Os artefatos seguem o padrão UML, entretanto, o levantamento de requisitos foi enriquecido com informações de importância e intensidade de uso da funcionalidade.

A ideia da proposta é complementar o TDD com uma fase de estimativas, na qual os resultados podem auxiliar o gerente a definir melhor os prazos e custos, diminuindo os riscos de prejuízos, e ainda, utilizando uma metodologia que prioriza os testes, consequentemente, reduzindo as falhas no software. Para comprovar a ideia da

proposta é necessário aplicá-la a um projeto de desenvolvimento de software, e durante o projeto coletar informações para validar ou refutar a proposta deste trabalho.

Com a validação, acredita-se que o processo de desenvolvimento utilizando a proposta trará um aumento de qualidade do software, o estabelecimento de estimativas mais precisas, melhor alocação de recursos, cumprimento dos prazos, sem perder a característica ágil da metodologia TDD.

5.1.1 Trabalhos Futuros

O presente trabalho tem como trabalhos futuros:

- Validar a aplicação da proposta em um ambiente de desenvolvimento de software;
- Após a validação, ensinar primeiramente a parte teórica proposta em cursos ou minicursos, e demostrar em seguida a parte pratica através de pequenos estudos de caso;
- Obter dados de projetos que utilizarem a propostas, com o intuído de melhorar a qualidade de processo de desenvolvimento e do software;
- Aplicar em pequenas, médias e grandes equipes para definir melhor o perfil de equipes que podem utilizar a proposta.

6 BIBLIOGRAFIA

Aranha, E. H. (Janeiro de 2009). Estimating Test Execution Effort Based on Test Specifications. Recife. Acesso em 13 de Maio de 2013, disponível em http://twiki.cin.ufpe.br/twiki/pub/SPG/GenteAreaThesis/EduardoAranha-thesis.pdf

Barbosa, E. F., Maldonado, J. C., Vincenzi, A. M., Delamaro, M. E., Souza, S. d., & Jino, M. (s.d.). Introdução ao Teste de Software. Acesso em 04 de Abril de 2013, disponível em http://informaticaemgeral.yolasite.com/resources/apostila%20-%20introdu%C3%A7%C3%A3o%20ao%20teste%20de%20software%20-%20jose%20carlos%20maldonado.pdf

Barcellos, M. P., Rocha, A. R., & Travassos, G. H. (s.d.). Planejamento de Custos em Ambientes de Desenvolvimento de. Rio de Janeiro, RJ, Brasil. Acesso em 16 de Abril de 2013, disponível em http://www.lbd.dcc.ufmg.br/colecoes/sbqs/2004/030.pdf

Beck, K. (2003). *Test-Driven Development.*Boston, MA. Acesso em 16 de Março de 2013, disponível em http://books.google.co.uk/books?id=gFgnde_vwMAC&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r#v=onepage&q&f=false

Bernardo, P. C., & Kon, F. (s.d.). A Importância dos Testes Automatizados. Acesso em 06 de Junho de 2013, disponível em http://novowebmail.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf

Borges, E. N. (2006). Conceitos e Benefícios do Test Driven Development. Porto Alegre, Rio Grande do Sul, Brasil: Universidade Federal do Rio Grande do Sul (UFRGS). Acesso em 16 de Março de 2013, disponível em http://www.inf.ufrgs.br/~cesantin/TDD-Eduardo.pdf

Camelo, I. R., & Serra, A. d. (2010). Desenvolvimento orientado a testes: Benefícios, técnicas e limitações. Acesso em 19 de Abril de 2013, disponível em http://www.infobrasil.inf.br/userfiles/27-05-S5-2-67986-Desenvolvimento%20orientado%20a%20testes.pdf

Campos, F. F., & Birnfeld, K. (2010). *Estimativas de Testes (APT).* Acesso em 21 de Abril de 2013, disponível em http://dftestes.gershon.info/Capitulo9.html

Cicilia, S. (22 de Agosto de 2012). ESTIMANDO ESFORÇO PARA TESTE DE SOFTWARE. Acesso em 16 de Abril de 2013, disponível em http://www.tiespecialistas.com.br/2012/08/estimando-esforco-para-teste-de-software/#.UXIbybXFXzx

Couto, B., Fernando, P., & Cezar, R. (s.d.). COnstrução de Sistema: Validação e Testes. Rio de Janeiro. Acesso em 08 de Junho de 2013, disponível em https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&ved=0CEgQFjADOAo&url=http%3A%2F%2Fwww.uniriotec.br%2F~seibel%2Fconstrunirio%2Farquivos%2FCS_Apresentacao_GRUPO_1_Validacao_e_Testes_de_Sistemas.doc&ei=xVKzUcOHA7HI0gGEm4GICw&usg=AFQjCNEyE

Cruz, W., Nelson, M. A., Barbosa, W., Souza, P., & Rioga, R. (s.d.). Objeto de Aprendizagem de Análise de Pontos de Testes. Belo Horizonte, MG, Brasil. Acesso em 20 de Abril de 2013, disponível em http://fees.inf.puc-rio.br/FEESArtigos/artigos/artigos_FEES12/104685.pdf

Falbo, R. A. (2005). Engenharia de Software: Notas de Aula. Acesso em 31 de Março de 2013, disponível em http://www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/NotasDeAula.pdf

Fenton, N. E., & Pfleeger, S. L. (1997). *Software Metrics.* Boston: PWS Publishing Company. Acesso em 29 de Março de 2013, disponível em http://ia700404.us.archive.org/24/items/softwaremetricsr00fent/softwaremetricsr00fent.pdf

Franzen, M. B., & Bellini, C. G. (Maio-Junho de 2005). ARTE OU PRÁTICA EM TESTE DE SOFTWARE? *11*(45). Acesso em 04 de Abril de 2013, disponível em http://www.read.ea.ufrgs.br/edicoes/pdf/artigo_340.pdf

International Organization of Standardization. (1998). ISO/IEC FDC 9126-1: Information technology - Software product quality. Acesso em 21 de Maio de 2013, disponível em http://twiki.cin.ufpe.br/twiki/pub/SPG/GenteAreaThesis/EduardoAranha-thesis.pdf

Junior, P. O. (s.d.). Testes de Caixa Branca e Caixa Preta. Minas Gerais, Brasil. Acesso em 31 de Março de 2013, disponível em http://www.tesestec.com.br/pasteurjr/TCPB.pdf

Maldonado, J. C., Vincenzi, A. M., Barbosa, E. F., Souza, S. d., & Delamaro, M. E. (s.d.). Aspectos Teóricos e Empíricos de Teste de Corbetura de Software. São Paulo, SP, Brasil. Acesso em 29 de Março de 2013, disponível em http://www.labes.icmc.usp.br/site/sites/default/files/NotaDidatica31.pdf

Michelez, J. (2011). The 4 Pillars os CAPEX Accurate Estimates. Acesso em 21 de Março de 2013, disponível em http://www.projectsmart.co.uk/pdf/the-4-pillars-of-capex-accurate-estimates.pdf

Myers, G. J. (1979). The Art of SoftwareTesting. Word Association. Acesso em 29 de Março de 2013, disponível em http://www.carlosfau.com.ar/nqi/nqifiles/The%20Art%20of%20Software%20Testing%20-%20Second%20Edition.pdf

Neto, A. C. (s.d.). Introdução a Teste de Software. *Engenharia de Software Magazine*(1). Acesso em 04 de Abril de 2013, disponível em http://vqv.com.br/es/esm01_IntroducaoATesteDeSoftware.pdf

Sanches, I. J. (s.d.). Integração da Ferramenta POKE-TOOL em um PSEE. Maringá, Paraná, Brasil. Acesso em 31 de Março de 2013, disponível em http://www.dainf.ct.utfpr.edu.br/~ionildo/tg/cap2.htm

Sommerville, I. (2011). Engenharia de Software. (9). São Palo, SP, Brasil.

Teste de Software. (2010). Em *Engenharia de Software: Metodologias de Desenvolvimento de Sistemas* (pp. 89-96). Acesso em 29 de Março de 2013, disponível em http://www.professorgersonborges.com.br/site/pdf/apostila/Engenharia/Intru_Engenharia_Software.pdf

The Standish Group. (1995). Chaos Report. Acesso em 21 de Março de 2013, disponível em http://www.projectsmart.co.uk/docs/chaos-report.pdf

The Standish Group. (2009). Chaos Report. Acesso em 21 de Março de 2013, disponível em http://www.projectsmart.co.uk/pdf/the-curious-case-of-the-chaos-report-2009.pdf

Veenendaal, E. P., & Dekkers, T. (1999). Testpointanalysis: a method for test estimation. Acesso em 16 de Março de 2013, disponível em http://www.erikvanveenendaal.nl/UK/files/Testpointanalysis%20a%20method%20for%20test%20estimation.pdf

Zhivich, M., & Cunningham, R. K. (Abril de 2009). The Real Cost of Software Errors. *IEEE SECURITY & PRIVACY*, *7*, 87-90. Acesso em 24 de Março de 2013, disponível em http://dl.acm.org/citation.cfm?id=1048712.1525815&coll=DL&dl=GUIDE

APÊNDICES

Levantamento de Requisitos

Cliente: Empreendedor Empreende **Levantamento Nº:** 001/2013

Data: 17/09/2013

Responsável pelo Levantamento: Mário Almeida

Nº	Cod. Requisito	Requisito	Importância	Intensidade de Uso
1	U1	Criar Usuário	( )Baixa ( )Normal (X)Alta	( )Baixa (X)Normal ( )Alta
2	U2	Editar Usuário	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
3	U3	Deletar Usuário	(X)Baixa ( )Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
4	A1	Criar Atleta	( )Baixa ( )Normal (X)Alta	( )Baixa (X)Normal ( )Alta
5	A2	Editar Atleta	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
6	A3	Deletar Atleta	(X)Baixa ( )Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
7	A4	Pesquisar Atleta	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
8	T1	Criar Time	( )Baixa ( )Normal (X)Alta	( )Baixa (X)Normal ( )Alta
9	T2	Editar Time	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
10	T3	Pesquisar Time	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
11	T4	Deletar Time	(X)Baixa ( )Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
12	P1	Iniciar Partida	( )Baixa ( )Normal (X)Alta	( )Baixa ( )Normal (X)Alta
13	P2	Finalizar Partida	( )Baixa ( )Normal (X)Alta	( )Baixa ( )Normal (X)Alta
14	P3	Pesquisar Partida	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
15	P4	Editar Partida	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
16	E1	Criar Estatística	( )Baixa ( )Normal (X)Alta	( )Baixa ( )Normal (X)Alta
17	E2	Editar Estatística	( )Baixa (X)Normal ( )Alta	(X)Baixa ( )Normal ( )Alta
18	E3	Pesquisar Estatística	( )Baixa (X)Normal ( )Alta	( )Baixa (X)Normal ( )Alta

_____

Cliente

_____

Responsável pelo Levantamento

Palmas, 17 de Setembro de 2013

Caso de Uso Expandido

Cliente: Empreendedor Empreende **CUE Nº:** 001/2013
Data: 17/09/2013

Responsável: Mário Almeida

U1 - Criar Usuário	
Atores	Administrador
Responsabilidade	Criar um novo usuário no sistema.
Descrição	O administrador acessa a parte de cadastro de usuário, preenche o formulário e envia.

Fluxo 1 – Administrador	
Ações do Ator	**Respostas do Sistema**
1. O Administrador acessar a parte de casdastro de usuario.	2. O sistema exibe a área de cadastro de usuario.
3. O Administrador preenche o formulário com os dados solicitados e envia.	4. O sistema informa que o cadastro foi realizado com sucesso.
Tratamento de Exceções	
Passo 4 - dados não preenchidos. O sistema emite alerta e retorna para o passo 3.	

_____
Cliente

_____
Responsável

Palmas, 17 de Setembro de 2013

Diagrama de Caso de Uso

DCU Nº:
001/2013

Cliente: Empreendedor Empreende
Data: 17/09/2013

Responsável: Mário Almeida

_____
Cliente

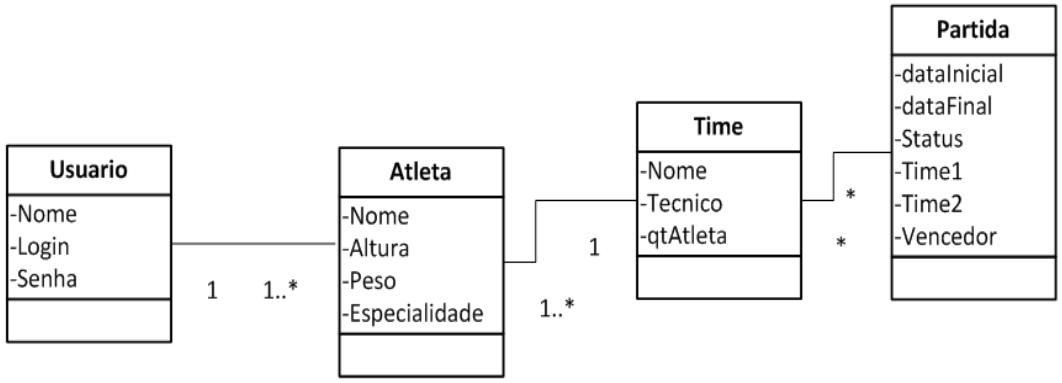
_____
Responsável

Palmas, 17 de Setembro de 2013
Modelo Conceitual

MC Nº:
001/2013

Cliente: Empreendedor Empreende
Data: 17/09/2013

Responsável: Mário Almeida



_____
Cliente

_____
Responsável

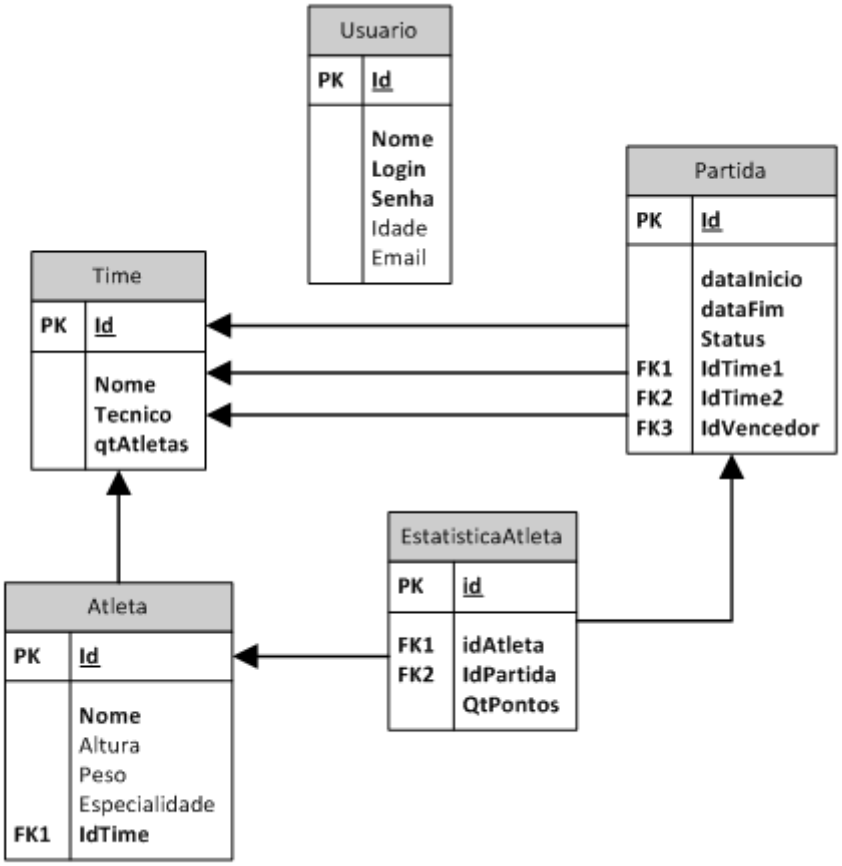
Palmas, 17 de Setembro de 2013

Modelo de Banco de Dados

MR Nº:
001/2013

Cliente: Empreendedor Empreende
Data: 17/09/2013

Responsável: Mário Almeida



_____
Cliente

_____
Responsável

Palmas, 17 de Setembro de 2013