UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E NFORMÁTICA
COORDENAÇÃO DE ESTÁGIOS E PROJETOS DE ENGENHARIA ELÉTRICA

RELATÓRIO DE ESTÁGIO INTEGRADO

Relatório de Estágio apresentado à coordenação de Engenharia elétrica da UFCG, como parte dos requisitos de obtenção do título de Engenheiro Eletricista.

Aluno: Nilo Sérgio de Aquino Arruda
Matícula: 29821131

Campina Grande
Março de 2006

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E NFORMÁTICA
COORDENAÇÃO DE ESTÁGIOS E PROJETOS DE ENGENHARIA ELÉTRICA

Relatório de Estágio Integrado

METODOLOGIAS DE TESTE DE SOFTWARE

Trabalho apresentado por: Nilo Sérgio de Aquino Arruda
Empresa: CIn/STP-Motorola□
Período de estágio: 04/01/2004 a 30/08/2004
Orientador: Prof. Glauco Fontgalland, Dr. Ing.

Campina Grande
Março de 2006

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Agradecimentos

Agradeço ao professores e funcionários do curso de engenharia elétrica da Universidade Federal de Campina Grande por todo conhecimento repassado no curso. Em especial, agradeço ao professor Glauco Fontgalland que me orientou no desenvolvimento deste trabalho com muita atenção.

Aos meus familiares, que estiveram do meu lado o tempo todo e sempre me apoiaram e incentivaram durante toda a minha formação acadêmica.

Aos meus amigos e colegas, por todo tempo e ajuda oferecida, em especial a Eisenhawer Fernandes, Vagner Vale e Kleber Melo, pois estivemos juntos durante toda a graduação compartilhando conhecimentos e dificuldades.

Índice

Lista de figuras

Resumo

Este trabalho apresenta uma abordagem sobre teste de software, levando-se em consideração noções elementares dos principais conceitos de engenharia de teste, tais como, estágios de teste, abordagens de teste, modelagem de sistema através da especificação SPEM, entre outros. E um relato das atividades práticas exercidas para a conclusão de teste nos aparelhos GSM da Motorola, do ponto de vista de execução de teste e de todo processo complementar a essa atividade. Dentro desse contexto foram realizadas diversas atividades relacionadas com teste de software, com atividades de execução de testes manuais e automáticos, planejamento e projetos de testes, requisições de mudança, inspeção de documentos.

Com o crescimento da utilização de sistemas de software, não apenas para PC's, como também para dispositivos eletrônico em geral, os dispositivos tornaram-se muito dependentes dos sistemas e esses sistemas tornaram-se cada vez mais complexo, necessitando de garantias criteriosas de suas funcionalidades e qualidade. Nesse contexto, surgiu um novo ramo de estudo que define metodologias que visam à detecção de falhas.

O projeto de teste atualmente é tão importante que é definido paralelamente ao projeto de desenvolvimento do software, e acompanha ativamente o desenvolvimento em todas suas etapas, deste os testes de Unidade, quando o software ainda está apenas em nível de código; passando por testes de integração, quando o software já está integrado no dispositivo e os requisitos básicos já estão funcionando; até os testes de validação que é realizado em conjunto com o cliente. Cada etapa de teste é necessária e atacam os defeitos sob diferentes contextos.

1. Introdução

O Projeto CIn-STP (*Software Test Program*) é o resultado de uma parceria entre a Motorola/BDC (*Brasil Design Center*) e o CIn/UFPE (Centro de Informática da Universidade Federal de Pernambuco), com o objetivo de capacitar engenheiros de testes para o desenvolvimento e a execução de testes de software para aparelhos celulares produzidos pela Motorola

Um dos critérios indispensáveis na medição da qualidade de software é a verificação de confiabilidade e funcionalidade de sistemas [1]. Apesar do conceito de qualidade não ser definido apenas com base nestes dois fatores, dificilmente um sistema será considerado portador de boa qualidade enquanto apresentar falhas na realização de suas funções. Tal idéia é verificada tanto quando se fala na definição intuitiva de qualidade de software, como na apresentação de modelos (modelo CMM - *Capability Maturity Model*) [10] ou normas de qualidade (série ISO 9001:2000) [11]. Portanto, a verificação de confiabilidade e de funcionalidade do programa como critério essencial de qualidade é tratada como um consenso comum e amplamente aceito.

A atividade principal realizada no CIn é o teste de software, dela são derivadas diversas outras, tais como, análise de CR's (Requisição de Mudanças), projetos e avaliação de casos de teste e inspeção de documentos. Portanto, no CIn são realizadas um conjunto de atividades necessário para validar produtos de software e garantir sua qualidade. Sua realização tem ênfase destrutiva visando encontrar todos os defeitos possíveis.

Esse trabalho é dividido em duas partes. Na primeira parte será apresentada uma abordagem teórica sobre teste de software, apresentando a fundamentação teórica das estratégias de teste e uma visão geral sobre o SPEM, especificação utilizada para modelagem do processo de desenvolvimento do CIn.

Em seguida, segunda parte, será apresentada uma descrição das atividades práticas desenvolvidas durante o estágio e descritas nesse trabalho, ou seja, a aplicação dos conhecimentos obtidos na área de teste de software, tanto do ponto de vista operacional quanto de planejamento. No entanto, por razões de segurança a descrição de várias etapas não contém detalhes ou

citações de ferramentas utilizadas ou do processo de trabalho aplicado. Isso se deve ao fato que o CIn utiliza todo o processo desenvolvido pela Motorola, com a qual deve ser mantido o sigilo das informações.

2. Objetivos

- Realizar teste de software nos aparelhos da motorola nas fases de integração e pré-integração.
- Elaborar projetos de casos de teste.
- Analisar e resolver CR's (*Change request*)
- Realizar inspeção de documentos

3. Teste de software

A atividade teste de software consiste, basicamente, na elaboração de três etapas : o planejamento de teste, o projeto de teste e a implementação dos testes. O planejamento oferece uma estrutura para a organização de alguns critérios, tais como: quais e quando os testes devem ser executados ou quem será responsável por cada teste; o projeto de teste define os passos necessários para interagir com a aplicação do sistema, pré e pós-condições para os testes e o critério de aceitação; na implementação são criados os casos de teste, que por sua vez, compreende a seqüência de passos e resultados esperados para que o testador verifique se a funcionalidade está sendo executada corretamente.

Um grande número de estratégias de teste de software tem sido proposto na literatura [5]. Elas oferecem ao desenvolvedor de software um esqueleto para testar características genéricas, ou seja, como o processo de teste deve ser planejado e projetado. Visando a análise comparativa dos diferentes conceitos e processos de testes, fez-se necessário o estudo na literatura de alguns procedimentos na área de planejamento de teste. Este capítulo apresenta as duas abordagens possíveis de teste de software, os

principais conceitos relacionados ao processo de teste e os estágios que cobrem todo o ciclo de teste.

3.1. Critérios para a Conclusão de teste

Todas as vezes que o cliente/usuário executa um programa de computador, o programa é testado em relação a um novo conjunto de dados. Esse simples fato fundamenta a importância de outras atividades relacionadas a garantia de qualidade de software. Do ponto de vista prático, a atividade de teste não tem fim, o que ocorre é que a carga simplesmente transfere-se do projetista para o cliente, através do atendimento ao consumidor e das agências de atendimento, ou até mesmo de sistemas de envio de relatórios de erros autmáticos.

3.2. Abordagens para testes de software

Os testes de software podem ser planejados sob duas abordagens de acordo com o nível de abstração que o testador julgue necessário, as quais são descritas a seguir. A abordagem Caixa Branca é utilizada no inicio do desenvolvimento do software durante os testes de unidade, mas de certa forma continua durante todo o processo se for levado em consideração que o simples fato do desenvolvedor compilar o software constitui uma etapa dos teste; A abordagem caixa preta ocorre apartir dos teste de integração e procegue durante toda "vida útil" do software.

3.2.1. Abordagem Caixa Branca

A abordagem caixa branca baseia-se num minucioso exame dos detalhes procedimentais. Os caminhos lógicos através do software são testados, fornecendo-se casos de teste que põem à prova conjuntos específicos de condições e/ou ciclos de teste. Ao utilizar métodos de caixa branca são derivados casos de testes que: garantam que todos os caminhos independentes dentro de um módulo tenham sido executados pelo menos uma vez, executem todas as decisões lógicas para valores falsos ou verdadeiros, executem todos os ciclos em suas fronteiras e dentro de seus limites

operacionais, bem como as estruturas de dados internas para garantir a sua validade.

3.2.2. Abordagem Caixa Preta

Os métodos de teste de caixa preta concentram-se nos requisitos funcionais do software. Ou seja, esse teste possibilita a derivação de conjuntos de condições de entrada que exercitem completamente todos os requisitos funcionais para um programa. Essa abordagem procura descobrir erros nas seguintes categorias: funções incorretas ou ausentes, erros de interface, erros nas estruturas de dados ou no acesso a bancos de dados externos, erros de desempenho e erros de inicialização e finalização do sistema. Os testes dessa abordagem tendem a ser aplicados durante as últimas etapas da atividade de teste.

3.3. Estágios de Teste de Software

O teste pode ser definido como um conjunto de atividades que pode ser planejada antecipadamente e realizada sistematicamente. Por essa razão, um esqueleto (*template*) de teste de software (o conjunto de passos no qual podemos alocar técnicas de projeto de casos de teste e métodos de teste específico) deve ser definido para o processo desenvolvimento de software.

Uma estratégia de teste de software deve ser flexível o bastante para promover a criatividade e a customização necessárias para testar adequadamente todos os grandes sistemas baseados em software. A estratégia deve acomodar teste de baixo nível, que é o conjunto de testes necessário para verificar se um segmento de código-fonte foi corretamente implementado e teste de alto nível, que são responsáveis pela validação de funções importantes do sistema contra requisitos do cliente.

A estratégia deve oferecer orientação aos testadores (ou engenheiros de teste) e um conjunto de marcos de referência ao administrador. Uma vez que os passos da estratégia de teste só ocorrem no momento em que as pressões de prazo final começam a surgir, o progresso deve ser mensurável e os problemas devem ir à tona o mais breve possível, dessa forma é

indispensavel a definicção de alguns elementos antes mesmo do inicio do desenvolvimento do software.

Plano de Teste: Apresenta o planejamento para execução do teste, incluindo a abrangência, abordagem, recursos e cronograma das atividades de teste. Identifica os itens e as funcionalidades a serem testados, as tarefas a serem realizadas e os riscos associados com a atividade de teste.

A tarefa de especificação de testes é coberta por 3 documentos:

Especificação de Projeto de Teste: Refina a abordagem apresentada no Plano de Teste e identifica as funcionalidades e características a serem testadas pelo projeto e por seus testes associados. Este documento também identifica os casos e os procedimentos de teste, se existirem, e apresenta os critérios de aprovação.

Especificação de Caso de Teste: Define os casos de teste, incluindo dados de entrada, resultados esperados, ações e condições gerais para a execução do teste.

Especificação de Procedimento de Teste: Especifica os passos para executar um conjunto de casos de teste.

Os relatórios de teste são cobertos por 4 documentos:

Diário de Teste: Apresenta registros cronológicos dos detalhes relevantes relacionados com a execução dos testes.

Relatório de Incidente de Teste: Documenta qualquer evento que ocorra durante a atividade de teste e que requeira análise posterior.

Relatório-Resumo de Teste: Apresenta de forma resumida os resultados das atividades de teste associadas com uma ou mais especificações de projeto de teste e provê avaliações baseadas nesses resultados

Relatório de Encaminhamento de Item de Teste: Identifica os itens encaminhados para teste no caso de equipes distintas serem responsáveis pelas tarefas de desenvolvimento e de teste.

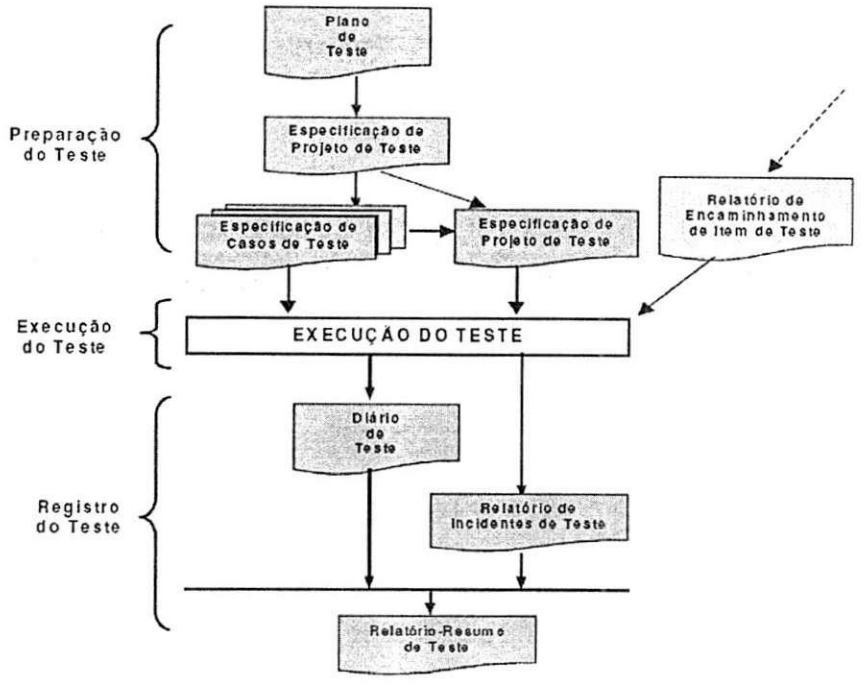

Figura 1: Relacionamento entre Documentos de Teste

E por fim, encontram-se os testes de validação, nos quais os requisitos estabelecidos como parte da análise de requisitos de software são validados em relação ao software que foi construído, a figura 1 ilustra todos os estágios de teste citados.

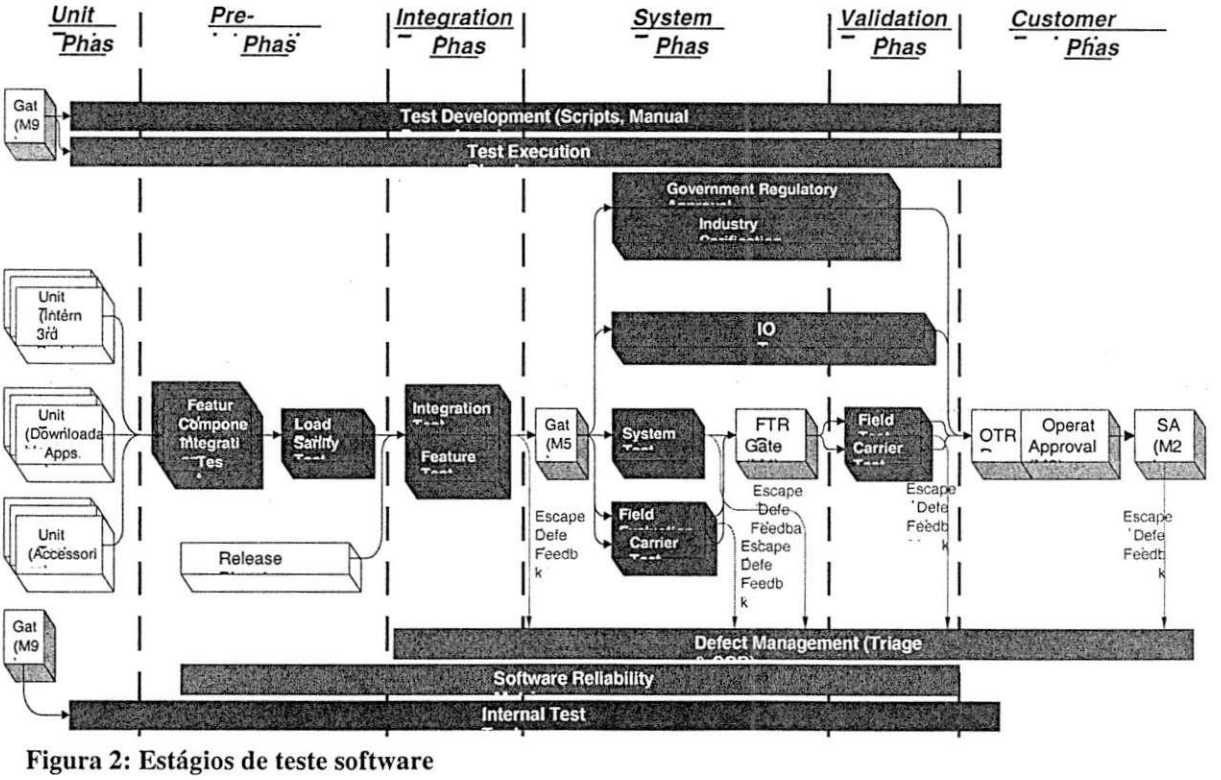

Figura 2: Estágios de teste software

Levando-se em consideração o processo para um ponto de vista procedimental, a atividade de teste dentro do contexto da engenharia de software é, de fato uma série de passos implementados seqüencialmente. Primeiramente, os testes focalizam cada módulo individualmente, garantindo que ele funcione adequadamente como uma unidade. Por isso, o nome teste de unidade. Esse teste faz muito uso das técnicas de teste de caixa branca, praticando-se caminhos específicos da estrutura de controle de um módulo, a fim de garantir uma completa cobertura e máxima detecção de erros.

Em seguida, os módulos devem ser montados ou integrados para formarem um pacote de software. O teste de integração cuida das questões associadas aos duplos problemas da verificação e construção de programas. As técnicas de projeto de casos de teste sob a abordagem caixa preta são mais encontradas durante a integração, não obstante uma quantidade limitada de teste de caixa branca possa ser usada para garantir a cobertura de fluxos importantes de controle.

Após o sistema ter sido integrado (construído), um conjunto de testes de alto nível deve ser executado para testar o sistema como um todo. Deve ser combinado os diversos elementos do sistema (por exemplo, hardware, usuários, bancos de dados, etc.). O teste de sistema verifica se todos os elementos combinam-se adequadamente e se a função/desempenho global do sistema é realizada com sucesso

Depois do software ser integrado com sucesso, os critérios de validação (estabelecidos durante análise de requisitos) devem ser testados. O teste de validação oferece a garantia final de que o software atende a todas as exigências funcionais, comportamentais e de desempenho. Os casos de teste sob abordagem caixa preta são usados integralmente durante a validação.

3.3.1. Teste de Unidade

Os testes de unidade têm como foco o esforço de verificação da menor unidade de projeto de software, o módulo. Esses testes baseiam-se sempre na caixa branca, e eles podem ser realizados em paralelo para múltiplos módulos.

O teste de unidade é composto por vários casos de testes, os quais testam a interface, a estrutura de dados locais, as condições de limite, os caminhos independentes e o caminho de tratamento de erros.

A interface é testada com o módulo para ter a garantia de que as informações fluem para dentro e para fora da unidade de programa que se encontra sob teste. A estrutura de dados local é examinada para ter a garantia de que os dados armazenados temporariamente mantenham sua integridade durante todos os passos de execução de um algoritmo. As condições de limite são testadas para ter a garantia de que o módulo opere adequadamente nos limites estabelecidos para demarcarem ou restringirem o processamento.

Todos os caminhos independentes (caminhos básicos) através da estrutura de controle são verificados para ter a garantia de que todas as instruções de um módulo foram executadas pelo menos uma vez. E, finalmente, todos os caminhos de tratamento de erros sejam testados.

Além das estruturas de dados locais, o impacto dos dados globais sobre um módulo deve ser verificado (se possível durante o teste de unidade).

3.3.2. Teste de Integração

O teste de integração é uma técnica sistemática para a construção da estrutura de programa, realizando-se, ao mesmo tempo testes para descobrir erros associados a interfaces. O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.

Na integração incremental o programa é construído e testado em pequenos segmentos, onde os erros são mais fáceis de ser isolados e corrigidos; as interfaces têm mais flexibilidade para serem testadas completamente; e uma abordagem sistemática ao teste pode ser aplicada. A seguir, serão abordadas as integrações *top-down* e *bottom-up*, conforme ilustrado na figura 2.

A integração *top-down* é uma abordagem incremental à construção da estrutura de programa. Os módulos são integrados movimentando-se de cima para baixo através da hierarquia de controle, o teste usa unidades auxiliares denominadas *drivers* e *stubs*. Os *drivers* substituem o programa principal que recebe dados do caso de teste, passa esses dados para o módulo a ser testado e imprime o resultado; Os *stubs* são módulos que substituem outros módulos subordinados, utiliza a interface do módulo subordinado, manipula os

dados e imprime uma verificaçãp. Ou seja, o *driver* é utilizado como módulo de controle principal, e os módulos reais são substituídos por *stubs*, a medida que os testes vão sendo realizados os *stubs* são substituídos por módulos reais.

A estratégia de integração *top-down* verifica antecipadamente os pontos de decisão ou de controle no processo de teste. Numa estrutura de programa bem trabalhada, a tomada de decisão ocorre nos níveis superiores da hierarquia e, por conseguinte, é encontrada primeiramente.

A integração *bottom-up*, como seu nome indica, inicia a construção e os testes com módulos atômicos (isto é, módulos localizados nos níveis mais baixos da estrutura de programa). Uma vez que os módulos são integrados de baixo para cima (*bottom-up*), o processamento exigido para os módulos subordinados em determinado nível está sempre disponível, não sendo, portanto, necessário a utilização de *stubs*.

Uma estratégia de integração *bottom-up* pode ser implementada com os seguintes passos:

- Módulos de baixo nível são combinados em *clusters* (ou construções) que executam uma subfunção de software específica.
- Um *driver* (um programa de controle para teste) é escrito para coordenar a entrada e a saída do caso de teste.
- O *cluster* é testado.
- Os *drivers* são removidos e os clusters são combinados ao se dirigir para cima na estrutura de programa.

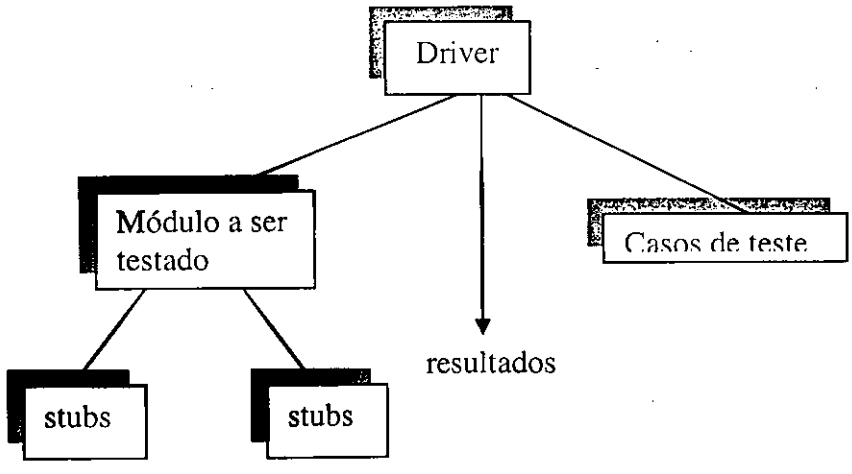

Figura 3: Diagrama ilustrativo dos fluxos de integração bottm up e top down

À medida que a integração desloca-se para cima, a necessidade de ter *drivers* de teste diminui. De fato, se os dois níveis superiores da estrutura de programa forem integrados de cima para baixo, o número de *drivers* pode ser reduzido substancialmente e a integração dos clusters torna-se grandemente simplificada.

3.3.3. Teste de Sistema

O teste de sistema é, na verdade, uma série de diferentes testes, cujo propósito primordial é pôr completamente à prova o sistema de software. Apesar de cada teste ter uma finalidade diferente, todo o trabalho deve verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas.

- **Teste de Recuperação**

O teste de recuperação é um teste de sistema que força o software a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada. Se a recuperação for automática (realizada pelo próprio sistema), a reinicialização, mecanismos de *checkpointing*, recuperação de dados e reinício são avaliados (cada um) quanto à corretude. Se a recuperação exigir intervenção humana, o tempo médio até o reparo é avaliado para determinar se ele se encontra dentro de limites aceitáveis.

- **Teste de Segurança**

O teste de segurança tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de fato, de acessos indevidos. Desde que tenha tempo e recursos suficientes, um bom teste de segurança penetrará por fim num sistema. O papel do projetista do sistema é fazer com que o acesso custe mais do que o valor da informação que será obtida.

- **Teste de Estresse**

O teste de estresse é realizado para confrontar os programas com situações limites. Esses testes executam o sistema de uma forma que exigem recursos em quantidade, freqüência ou volume excessivos. Exemplos desses testes são: (1) testes especiais que geram 10 interrupções por segundo podem

ser projetados, quando uma ou duas interrupções por segundo é a taxa média; (2) os casos de teste que exigem máxima utilização de memória ou outros recursos podem ser executados; (3) os casos de teste podem provocar procura excessiva por dados residentes em disco não projetados. Essencialmente, o analista tenta destruir o programa.

- **Teste de Desempenho**

O teste de desempenho é idealizado para testar o desempenho de *run-time* (traduzido como tempo de execução) do software dentro do contexto de um sistema integrado. O teste de desempenho ocorre ao longo de todos os passos do processo de teste. Até mesmo em nível de unidade, o desempenho de um módulo individual pode ser avaliado quando são realizados testes de caixa branca. Porém, só quando todos os elementos de sistema estão plenamente integrados é que o desempenho real de um sistema pode ser verificado.

3.3.4. Teste de Validação

Ao finalizar os testes de integração, o software está completamente montado como um pacote, erros de interface foram descobertos e corrigidos e uma série final de testes de software (os testes de validação) pode iniciar-se. Uma das definições da validação diz que ela é bem sucedida quando o software funciona de maneira razoavelmente esperada pelo cliente.

A validação do software é realizada por meio de uma série de testes de caixa preta que demonstram a conformidade com os requisitos. Um plano de teste esboça as classes de testes a serem realizadas e um procedimento de teste define os casos de teste específicos que serão usados para demonstrar a conformidade com os requisitos. Tanto o plano como o procedimento é projetado para ter a garantia de que todos os requisitos funcionais sejam satisfeitos, todos os requisitos de desempenho sejam conseguidos, a documentação esteja correta e outros requisitos sejam cumpridos, por exemplo, portabilidade, compatibilidade, remoção de erros e manutenibilidade.

Depois que cada caso de teste de validação for realizado, existirá uma dentre duas condições:

- As características de função ou desempenho conformam-se às especificações e são aceitas.
- Um desvio das especificações é descoberto e uma lista de deficiências é criada.

Os desvios ou erros descobertos nessa fase de um projeto raramente podem ser corrigidos antes da conclusão programada. Muitas vezes é necessário negociar com o cliente para estabelecer um método para a superação das deficiências.

- **Teste Alfa e Beta**

Construtores de software comerciais usam dois artifícios denominados teste alfa e teste beta para utilizar o usuário final como testador. O teste alfa é levado a efeito por um cliente nas instalações do desenvolvedor observando o usuário e registrando erros e problemas de uso. Os testes alfa são conduzidos em um ambiente controlado.

O teste beta é realizado em uma ou mais instalações do cliente pelo usuário final do software. Ao contrário do teste alfa, o desenvolvedor geralmente não está presente. Portanto, o teste beta é uma aplicação "viva" do software, num ambiente que não pode ser controlado pelo desenvolvedor. Como resultado dos problemas relatados (pelo usuário) durante o teste beta, o desenvolvedor do software faz modificações e então se prepara para lançar o produto software a toda base de clientes.

4. Modelagem de processo de software

O SPEM (*Software Process Engineering Metamodel*) é usado para descrever um processo de desenvolvimento de software ou uma família de processos. Para a modelagem do processo, é considerada uma abordagem orientada a objeto que utiliza a notação de UML (*Unified Modeling Language*).

A UML é uma linguagem gráfica para modelagem discreta de sistemas, que não é pretendida a uma área de aplicação particular ou modelagem de processo. Sua maior aplicabilidade é na área de projetos de software orientados a objetos [6], [7]. Para nosso propósito, existe uma variação do

modelo de UML que deve ser seguido, o UML profile, que usa o mecanismo de extensão de UML de uma maneira padronizada para um propósito particular.

4.1. Diagramas representativos

Seguindo a especificação SPEM [7] e utilizando a linguagem UML [6] fizemos a modelagem do processo através de três tipos de diagrama gráficos que podem descrever eficientemente os processos do CIn.

Diagrama de caso de uso: Esse diagrama deve representar apenas a relação entre *ProcessRole* e as principais *Activity*. Não deve ser inserido qualquer outro tipo de informação, a figura 3 ilustra um diagrama de caso de uso.

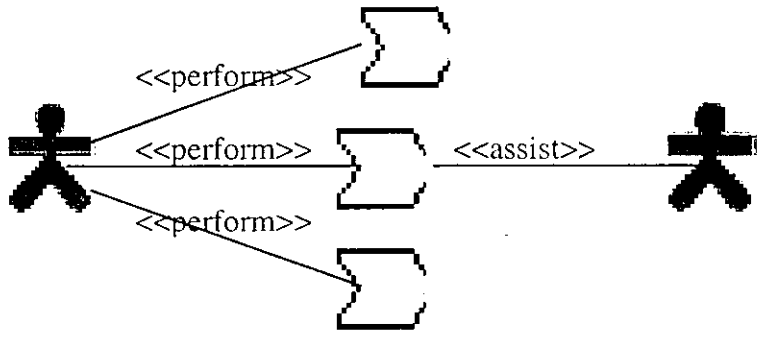

Figura 4: Diagrama de caso de uso para modelagem de processos

Diagrama de classe: Esse diagrama permite a representação de aspectos relacionados a processo de software, tais como: associações entre *ProcessRole* e *activity*; relações entre *processRole* e w*orkProcuct*; estrutura, decomposição e dependências de w*orkProduct*. Esse diagrama deve conter o *processRole* com as *Activity* que estão associadas a ele. As *Activity*, por sua vez, devem estar relacionadas a todos os *workProduct* que elas consomem, produz ou modifica. Neste trabalho não serão abordados os conceitos de associações, relações, estrutura, decomposição e dependência, os quais estão detalhados em [6]. A figura 4 ilustra um diagrama de classe.

Diagrama de atividades: Permite a representação da seqüência de atividades com fluxo de execução, seus *workProduts* de saída com os respectivos estados em que se encontra para cada faze do ciclo de atividades, além disso

pode ser usado linhas verticais para separar as responsabilidades dos diferentes *processRole*, a figura 5 ilustra um diagrama de atividades.

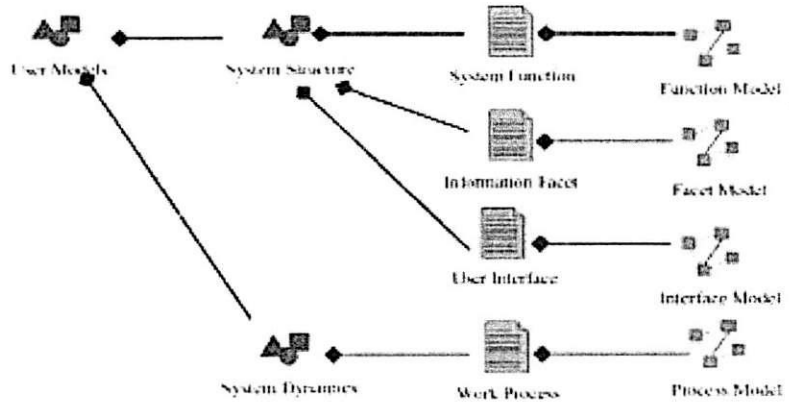

Figura 5: Diagrama de classes para modelagem de processos

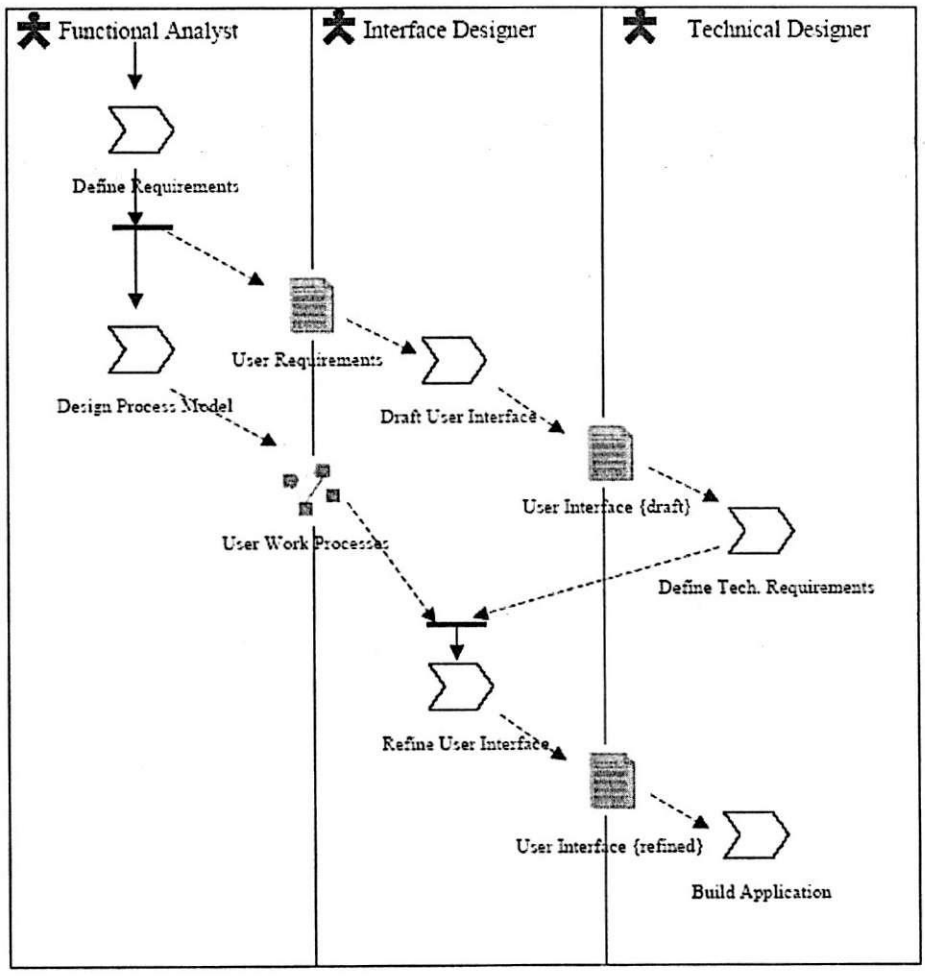

Figura 6: Diagrama de atividades para modelagem de processos

Após os diagramas gráficos utilizados nesse trabalho, apresentaremos os principais elementos dessa especificação.

4.2. Principais elementos do SPEM

O SPEM é um metamodelo para definição de processo e dos componentes do processo [7]. Nesta seção será apresentada uma descrição de um conjunto mínimo de elementos necessários para modelagem de processos de software.

Process: É um componente de processo que cobre um conjunto de *discipline*'s e que tem "vida própria", isso significa que ele constitui todo um sistema.

Discipline: É uma utilização particular de *package* que agrupa um conjunto de atividades que trabalham de acordo com um tema. Agrupando as atividades, dessa forma elas produzem *workproduct*'s similarmente categorizado sobe o tema em foco.

Activity: Descreve um trabalho executado por um *processrole*, ou seja, as tarefas, operações e ações que são executadas por um papel, ou as que um papel podem assistir. Uma atividade consiste de elementos chamados *steps*.

ProcessRole: Uma subclasse de processo que executa e define responsabilidades sobre *workproduct* específico. Isso define os papeis que executam e assistem uma tarefa específica.

WorkProduct: Workproduct ou artefato é tudo que é produzido, consumido ou modificado por um *process*. Pode ser um pedaço de informação, um documento, um modelo, um código fonte, entre outros.

Guidance: Representa um guia detalhado que deve ser usado como referência para produção ou modificação de um *workprodut*. Um *guidance* não pode ser modificado, deve ser usado apenas como guia.

5. Trabalho Desenvolvido

O nosso trabalho consiste, basicamente, em realizar teste de software (em sistemas embarcados) nos aparelhos de telefonia móvel desenvolvidos

pela Motorola em sua fase de integração conforme foi detalhada no capítulo anterior, que apresenta um panorama mais fácil para a detecção e correção de defeitos encontrados nos softwares, bem como testes para descobrir erros associados a interfaces.

Para o relato dos resultados dos testes, existe um processo adotado pelo CIn, que segue o modelo da Motorola. Sendo a Motorola, uma multinacional com um sistema de produção distribuído, existe a necessidade de um processo pesado para garantir a comunicação entre as diversas unidades espalhadas por todo o mundo, além disso, deve existir um controle do fluxo de informações, para que elas não sejam perdidas, ou até mesmo violadas, motivo que justifica o fato de não ser descrito neste trabalho.

Como atividade complementar à atividade de testes, devemos abrir e analisar CR's, realizar inspeção de documentos, projetar casos de teste, as quais serão descritas na próxima sessão.

Paralelamente a essas atividades, foi desenvolvida uma atividade a qual tentamos implantar alguma melhoria ao CIn, seja de natureza funcional (desenvolvimento de uma ferramenta auxiliar) ou não (metodologia para melhora o processo do CIn). Nesse contexto foi identificada a necessidade de modelagem do processo, a qual possibilita melhor adaptação de novos membros, visibilidade geral do projeto e menor dependência dos profissionais existentes. Tal modelagem foi baseada na especificação do SPEM (*software process engineering metamodel*) que utiliza a linguagem de modelagem de software UML para modelagem de processo de desenvolvimento.

5.1. Requisição de Mudanças (*Change Request*)

Para que seja mantido o controle de versão, é necessária adoção de alguns métodos. Dessa forma, sempre que for detectada alguma falha, ou se for identificado a possibilidade/necessidade de mudança na interface do software e nos documentos de caso de teste, é realizado um procedimento oficial. Esse procedeimento é bem documentado de forma que seja avaliado e reproduzido por outras pessoas designadas para resolver tal problema. Todo

esse procedimento é chamado de CR (*Change Request*) que em português significa requisição de mudança.

Quando uma CR seja aberta são preenchidos alguns tópicos, tais como o responsável pela possível falha, a data de abertura e o defeito, entre outros. Essa requisição de mudanças é alocada para algum membro de algum time dependendo do escopo, finalmente o encarregado pela resolução da CR deve analisá-la reproduzindo o defeito. Existem duas possibilidades, a primeira é que o responsável pode concluir, baseado no documento de requisito, que o comportamento encontrado não representa mais um defeito e sim uma alteração evolutiva do software ou, realmente, o comportamento encontrado pelo testador está errado. Nesse caso o responsável pela CR deve permanecer com a CR aberta para que o time de desenvolvimento resolva o defeito encontrado, o qual deve ser resolvido na próxima versão da *build*.

5.2. Inspeção de documentos

A atividade de inspeção de documentos tem como objetivos realizar um trabalho direcionado na detecção de erros encontrados no documento, para que esses sejam corrigidos posteriormente, evitando gasto de tempo excessivo na correção dos trabalhos. Bem como, reduzindo ao máximo o número de erros no documento que será publicado.

Nessa atividade, deve ser formada uma equipe contendo, basicamente, o autor do trabalho, um moderador, um testador (no nosso contexto), um leitor e um escritor, todos têm a função de inspecionar o documento, além disso, uma pessoa pode assumir mais de uma tarefa.

Após a formação da equipe, que deve ser indicada pelo autor, cada membro deve receber um convite via *email* no mínimo 24 horas antes do horário de inicio da realização do evento, esse espaço de tempo precedente é utilizado por cada membro para que eles realizem uma preparação, fazendo uma análise prévia do documento inteiro ou de um trecho que tenha sido alterado. A duração de cada inspeção deve ser de no máximo 2 horas, pois estudos realizados por profissionais da área indicam que se excedendo esse período, a realização do trabalho passa a ter produtividade muito baixa. Ao

final de cada inspeção, deve ser agendada a realização da próxima, caso não tenha sido suficiente para finalizar o documento. No período entre as inspeções, o autor deve realizar o retrabalho das alterações apontadas pela equipe.

Para que não seja perdida qualquer informação contida nos documentos, seja ela correta ou não, deve existir um controle de versão. Dessa forma, o documento que está sendo alterado tem versão diferente do documento inspecionado, que por sua vez tem versão diferente do documento retrabalhado. Até o fechamento da versão do documento que deve ser revisado pelo autor e pelo moderador, caso seja encontrado muitos erros nessa última etapa, pode ser marcada uma nova inspeção, evento raro que deve ser evitado por todos desde a fase de alteração/construção do documento. A seguir é descrita a função de cada membro da equipe na atividade.

5.3. Projetar Casos de Teste

O desenvolvimento de software consiste num processo dinâmico, que sofre alterações em todo seu ciclo de vida, novos requisitos surgem e são modificados durante todo tempo. Dessa forma, é necessário que o processo de testes acompanhe essa constante evolução, ou seja, com as mudanças dos requisitos o projeto de testes também deve mudar, acompanhando as novas versões do software para evitar trabalhos perdidos, tais como detecção de defeitos inexistentes, que desencadeariam todo um procedimento, mobilizando várias pessoas inutilmente.

Quando é iniciado o desenvolvimento de uma nova *feature*, paralelamente é elaborado e projetado um documento de testes, que deve conter os casos de testes necessários para testar tal *feature*. Esse documento chamado de FTD (*Feature Test Document*), assim como as primeiras versões de softwares, é publicado com muitos defeitos ou de forma incompleta, sendo necessária a constante atualização do mesmo.

Quando é identificada a necessidade de alteração ou até criação de um novo caso de teste, o testador deve abrir uma CR, conforme descrito anteriormente. Esse evento pode ser ocasionado, basicamente por três

motivos, os requisitos foram alterados e o caso de teste não é mais adequado para tal situação; por está ocorrendo uma falha que o caso de teste não consegue mais detectar e, ainda, quando a falha é encontrada no teste exploratório. Esse tipo de teste é realizado pelo exclusivamente pelo bom senso do testador, ou seja, as falhas são procuradas no *phone* sem apoio de qualquer caso de teste.

Em qualquer situação, o testador deve abrir uma CR contra o documento de caso de teste, apontando a modificação necessária. E todo o procedimento de CR, análise do problema, inspeção de documentos e retrabalho deve ser seguida, conforme descrito neste trabalho.

Durante a atividade de análise do problema o responsável deve utilizar a versão atualizada do documento de requisitos como base.

5.4. Instalação das Versões de Software

A bateria de teste deve ser executada sempre que for lançada uma nova *build,* que por sua vez, é uma versão atualizada de um software, as novas versões devem corrigir erros encontrados anteriormente. Assim devemos instalar completamente a nova *build,* ou apenas atualizá-la, dependendo de quão radicais sejam as alterações. Seja qual for a situação, existe algumas ferramentas utilizadas, pela motorola, nesse processo e que devem ser utilizadas através de um PC conectado ao *phone* por um cabo serial.

5.5. Modelagem do processo de messaging do Cin

A modelagem de um processo é um tipo de documentação necessária para que todos os colaboradores tenham uma visão geral da organização, e estejam completamente inseridos no processo. Novos integrantes serão inseridos mais facilmente, e o fato de todos conhecerem o processo possibilita que a organização se torne mais estável, pois qualquer integrante pode identificar algum problema, de qualquer natureza e sugerir ações para melhorar o processo, deixando de ser uma atividade apenas da alçada da gerência.

Realizamos essa modelagem baseada no SPEM, descrito anteriormente, e utilizamos a ferramenta *Rational Rose*. O processo de message foi modelado em três macro-atividades, denominadas disciplinas, segundo a terminologia adotada pelo SPEM, as quais agrupam um conjunto de atividades análogas.

Primeiramente, foi realizada uma entrevista com o gerente do projeto, que conhece bem todo o processo, em seguida entrevistamos os responsáveis por cada atividade desenvolvidas no CIn.

Com base nas informações coletadas, agrupamos as atividades de acordo o seu objetivo, ou seja, focamos um produto e selecionamos as atividades diretamente relacionadas no desenvolvimento desse produto. Bem como, os *ProcessRoles* que são responsáveis pelas respectivas atividades. Feito esse levantamento, associamos os *ProcessRole* com suas respectivas atividades e construímos os diagramas de caso de uso. Para construção do diagrama de classes, acrescentamos os WorkProcuts consumidos, produzidos ou modificados por cada atividade. Finalmente, construímos o diagrama de atividades, o qual contém toda a seqüência de execução das atividades, com todos seus componentes envolvidos.

Após todos os diagramas construídos, construímos um documento, descrevemos cada atividade e ilustramos cada situação através dos diagramas. Os resultados dessa modelagem não podem ser exibidos nesse trabalho por conter informações sigilosas, entretanto, foi permitida a apresentação do diagrama de atividades referente à disciplina de *test*, exibido na figura 6.

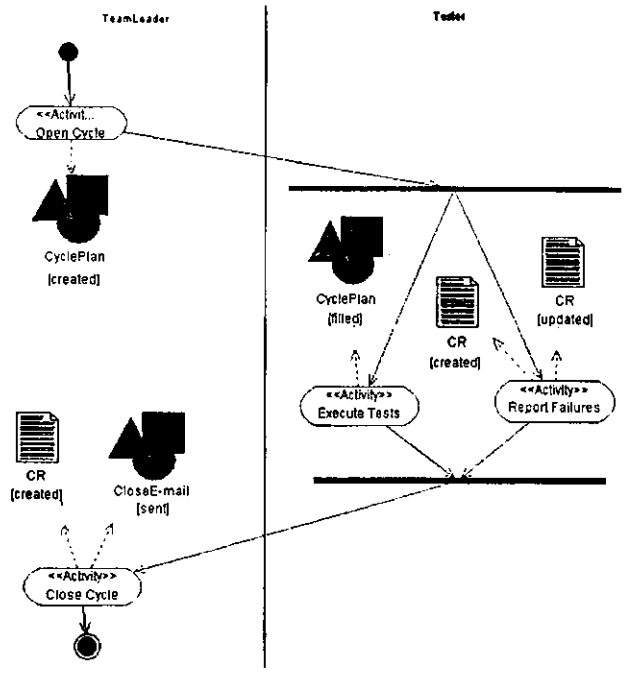

Figura 7: Diagrama de atividades do processo de message

5.6. Relatório das atividades

Todas as atividades realizadas por todos os integrantes do CIn devem ser relatadas individualmente através de uma ferramenta, o *AllNetic Working Time Tracker*, que é a ferramenta padrão definida para a reportagem de tempo de atividades realizadas no projeto Cin-STP.

Para reportagem de tempo deve ser seguida a estrutura padrão de atividades definida pelo projeto. As atividades foram definidas pelo gerente do projeto em conjunto com os líderes de equipe.

Ao reportar uma atividade, o usuário deve procurar identificar na estrutura do AllNetic uma descrição que seja compatível com a sua necessidade. O usuário não deve excluir ou alterar uma descrição, ou incluir novas atividades na estrutura. Necessidades de novas atividades não contempladas pela estrutura padrão devem ser comunicadas ao gerente de projetos para que sejam tomadas as devidas ações corretivas.

As atividades padrão do projeto estão estruturadas hierarquicamente. A reportagem de tempo deve ser realizada nas atividades de nível mais baixo da estrutura. Ao reportar a atividade, deverá ser informada na descrição do período reportado a seguinte informação:

- Uma descrição da atividade realizada. Exemplo: Manutenção da CR no. LIBxx999adad
- Data para atividade. Exemplo: 24/05/2003.
- Percentual já realizado da atividade. Este percentual deve ser o percentual acumulado desde o início da realização da atividade. Por exemplo, se no primeiro dia foi realizado 10% da atividade e no segundo dia foi realizado mais 20%, ao reportar a atividade do segundo dia, deverá ser informado 30%. Exemplo: Realizado 10%.

AllNetic Working Time Tracker. Report for Nilo Sergio.

Summary

First date: segunda-feira, 6 de setembro de 2004, 00:00:00
Last date: domingo, 12 de setembro de 2004, 23:59:59
Total time: 23:58:18

Detailed

Task	Begin	End	Duration
1. 1. 1.Cin-STP Messaging Project Actitivies	6/9/2004, 00:00:00	12/9/2004, 23:59:59	23:58:18
1.1. '1.C.Support Activities'	6/9/2004, 00:00:00	10/9/2004, 17:05:04	13:02:11
1.1.1. '1.C.Support Activities' \ '1.C.3.Prepare/Ministre Training'	8/9/2004, 14:00:49	10/9/2004, 16:11:16	12:08:23
Preparando/Apresentando o proceso do CIn aos novos bolsistas	8/9/2004, 14:00:49	8/9/2004, 18:00:00	03:59:11
Treinamento de FTD aos novos bolsistas	9/9/2004, 16:00:34	9/9/2004, 18:01:18	02:00:44
Treinamento de FTD aos novos bolsistas	10/9/2004, 08:02:34	10/9/2004, 12:02:54	04:00:20
Treinamento de FTD aos novos bolsistas	10/9/2004, 14:03:08	10/9/2004, 16:11:16	02:08:08
1.1.2. '1.C.Support Activities' \ '1.C.4.Prepare team environment'	6/9/2004, 00:00:00	10/9/2004, 17:05:04	00:53:48
Preparando ambiente para execussão de testes no FIGI	10/9/2004, 16:11:16	10/9/2004, 17:05:04	00:53:48
1.2. '1.D.Project Training'	6/9/2004, 00:00:00	12/9/2004, 23:59:59	06:00:03
1.2.1. '1.D.Project Training' \ '1.D.2.Self Study / On The Job'	6/9/2004, 00:00:00	12/9/2004, 23:59:59	06:00:03
Estudando enquanto não sou alocado em alguma atividade	9/9/2004, 08:00:36	9/9/2004, 12:00:39	04:00:03
Estudando enquanto não sou alocado em alguma atividade	9/9/2004, 14:00:34	9/9/2004, 16:00:34	02:00:00
1.3. '1.G.Improductive'	6/9/2004,	12/9/2004,	00:54:56

	00:00:00	23:59:59	
Tentando colocar o phone em serviço	10/9/2004, 17:05:04	10/9/2004, 18:00:00	00:54:56
1.4. '1.8.Test Execution'	6/9/2004, 00:00:00	12/9/2004, 23:59:59	04:01:08
1.4.1. '1.8.Test Execution' \ '1.8.R.Review test results'	6/9/2004, 00:00:00	8/9/2004, 12:01:36	04:01:08
Estudando enquanto não sou alocado em alguma atividade	8/9/2004, 08:00:28	8/9/2004, 12:01:36	04:01:08

Nesse capítulo foi apresentada de maneira geral toda atividade executada neste trabalho, a seguir, serão relatadas as configurações finais.

6. Conclusão

Neste trabalho, verificamos que a área de execução de testes de software é uma realidade e está sendo inserida em todas as empresas que produzem dispositivos que usam software, ou seja, praticamente todas as empresas. Pois é fundamental para garantia da qualidade dos produtos. Baseado na nova necessidade do mercado foi desenvolvido um trabalho totalmente relacionado com a área explicitada, pois o CIn é responsável pelos testes de software das *feature*'s de *message* da Motorola. Dessa forma realizamos todas as atividades relacionadas desde planejamento, projeto de caso de teste até a sua execução.

Uma experiência importante é o fato de trabalhar com uma multinacional, que precisa implantar um processo pesado e burocrático para que possa funcionar. Um processo desse tipo tem impacto forte sobre o desempenho das atividades, pois para qualquer atividade por mais simples que seja, deve ser registrada e todo procedimento adotado pelo processo deve ser seguido. No entanto, é necessário já que colaboradores do mundo inteiro têm acesso a todos os produtos.

Além de toda experiência adquirida no contexto de teste de software com o objetivo de assegurar a qualidade desses produtos, foi possível a utilização de várias ferramentas computacionais e o estudo em diversas novas tecnologias, as quais possibilitam qualificação indispensável ao profissional da área.

O planejamento de testes é uma atividade, principalmente, gerencial e de tomadas de decisões, dessa forma exige do executor grande poder de liderança e capacidade de identificar o perfil da equipe. O gerente de testes é responsável pela alocação de pessoas, estabelecimento de prazos e prioridades e a definição do escopo de teste. Essa atividade está longe da objetividade e o profissional da área tem que estar preparado para tomar decisões bastante subjetivas. Pode-se perceber facilmente essa observação na simples utilização de um software por diferentes pessoas, alguns terão mais facilidade que outros ao interagir com o sistema através de sua interface.

Fazendo-se algumas analogias com o curso e considerando os projetos desenvolvidos, qualquer engenheiro está bem preparado, já que os fazemos

todo o tempo, na verdade, projetar é intuitivo para um engenheiro, a diferença é que fazemos trabalhos bastantes objetivos, através da aplicação de fórmulas, conceitos de engenharia, mas é claro que também existe um alto grau de subjetividade, onde será definido o melhor projeto, nos projetos de software são visto sobre um ponto de vista diferente, mas totalmente adaptável. Com relação às pressões de prazo, também estamos bastante preparados, talvez a maratona de provas e relatórios seja mais difícil.

Com relação às atividades gerenciais, é necessário ter muita destreza e está sempre atento a tudo que acontece como informações novas, possíveis problemas, motivação dos membros da equipe para determinadas atividades, estudo de impacto e riscos referentes à implantação de determinados sistema, processo ou metodologia, e a mais difícil de todas que é o relacionamento pessoal. E todo esse leque de habilidades não é adquirido na universidade ou em algum curso específico, mas a soma de todo tipo de experiência, pessoal, acadêmica e profissional.

Do ponto de vista técnico, não houve maiores problemas, com a formação mais generalista que temos foi possíveis absorver facilmente os conceitos das novas tecnologias existentes no mercado, mas houve certa desvantagem com relação a alunos de outras faculdades. A maneira como o curso é ministrado é eficiente, mas precisa ser mais dinâmico, principalmente com relação as disciplinas específicas. Além dos conceitos básicos de determinadas áreas, é necessário a utilização das tecnologias que estão surgindo no mercado. E aplicação dessa sistemática talvez fosse seguida se na ementa de todas as disciplinas específicas existisse um tópico no qual o aluno estudasse algum tema de uma tecnologia utilizada no mercado, e esse tópico deveria contemplar maior parte da disciplina. Existe uma deficiência muito grande nos processo de recrutamento, seria muito importante a adoção de uma disciplina voltada pra esse quesito, ensinando como preparar currículos e se comportar nas entrevistas.

A área de analise de falhas esta em constante ascensão no mercado e demanda bastantes vagas, pois existem poucos funcionários qualificados. Já existe estudo sobre a aplicação de testes de certificação na área, visando medir e melhorar a qualidade desses profissionais. Portanto, ao profissional que se interessar pela área, existem grandes chances de ser inserido no

mercado, é uma atividade relativamente simples, mas de grande responsabilidade exigindo espírito investigativo, paciência, perseverança, firmeza de opinião, pois a partir de um trabalho de teste bem planejado, teremos um produto com boa aceitação no mercado, ou prejuízos para a empresa, com a necessidade de recalls (chamadas das empresas aos usuários para reparar defeitos em determinado produto), que além de caro, denigre a imagem da empresa.

7. Referências Bibliográficas

- [1] Pressman, R. S. **Engenharia de Software**, Mc Graw Hill, 5ª Edição, 2001.
- [2] Furlan, J. D. Modelagem de Objetos através de UML, Makron Books, São Paulo, 1998.
- [3] Sommverville, I. *Engenharia de Software.* 6a. Edição. São Paulo: Addison-Wesley, 2003.
- [4] Rocha, A. R. C., Maldonado, J.C. E Weber, K. C. *Qualidade de Software: Teoria e Prática.* São Paulo: Prentice-Hall, 2001.
- [5] McGregor, J. D.; korsons, T. D., Integrated Object-Oriented Test end Development Process, 1994.
- [6] http://www.omg.org/docs/ptc/04-04-02.pdf, 23/05/2004.
- [7] http://www.omg.org/docs/formal/02-11-14.pdf, 18/05/2004.
- [8] Lewis, W. E.; Software Testing and Continuous Quality Improvement; Auerbach, New York, 2000.
- [9] Cantor, R. M.; Object-Oriented Project Management; Wiley; Toronto;1998.
- [10]http://www.sei.cmu.edu/pub/documents/93.reports/pdf/tr25.93.pdf, 20/05/2004.
- [11] http://www.tc176.org/Interpre.asp, 03/07/2004.