
Pós-Graduação em Ciência da Computação

# “Adaptação do Processo de Desenvolvimento de Software para Análise de Cobertura de Código”

**Por**

***Elifrancis Rodrigues Soares***

**Dissertação de Mestrado**


Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2007

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELIFRANCIS RODRIGUES SOARES

# "Adaptação do Processo de Desenvolvimento de Software para Análise de Cobertura de Código"

*ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.*

*ORIENTADOR: PROF. DR. ALEXANDRE MARCOS LINS DE* VASCONCELOS

RECIFE, FEVEREIRO/2007

**Soares, Elifrancis Rodrigues**

**Adaptação do processo de desenvolvimento de software para análise de cobertura de código / Elifrancis Rodrigues Soares. – Recife: O autor, 2007.**

**x, 211 folhas: il., fig., tab.**

**Dissertação (mestrado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2007.**

**Inclui bibliografia e apêndices.**

**1. Engenharia de software. 2. Processo de desenvolvimento de software. 3. Qualidade de software. 4. Teste de software. I.Título.**

**005.1 CDD (22.ed.) MEI2007-049**

Dedico este trabalho a minha irmã, Francilete (in memorian), pois apesar do tempo e da distância o nosso amor é eterno.

# Agradecimentos

Acima de tudo a Deus, que sempre esteve presente na minha vida como fonte de inspiração e fortaleza, sendo meu principal apoio nos momentos de dificuldade.

Aos meus pais, Francisco e Eliete, que sempre estiveram ao meu lado com palavras de apoio e incentivo.

À minha irmã, Franciete, pelo incentivo dado desde o início. Ao meu sobrinho querido, que é minha maior fonte de descontração.

À minha noiva Joseilma, pelo amor e carinho. Sua força dada a todo o momento foi muito importante nos momentos difíceis.

Aos meus grandes amigos do grupo Rayovack, que estiveram comigo neste período muito importante da minha vida, me dando incentivo e força em todos os momentos.

Aos meus amigos conquistados durante o período do projeto, pelo incentivo e apoio,especialmente a: João dos Prazeres, Rafael Marques, Eduardo Aranha, Sidney Nogueira, Eliezez Braz, Angelo Ribeiro, Lucas Loreiro, Alexandre Campelo, José Elias, Jeremias Queiroga e Diego Madruga.

Ao Prof. Alexandre Vasconcelos, o qual como meu orientador, sempre incentivou meu trabalho, mostrando-me os melhores caminhos a serem seguidos, através de sugestões valiosas, além das oportunidades oferecidas e o apoio nos momentos difíceis.

Por fim, à Motorola, Facepe, Centro de Informática e ao C.E.S.A.R pela oportunidade de aplicar esse trabalho. Aos professores Augusto Sampaio, Sergio Cavalcanti e Paulo Borba, e ao gerente de qualidade José Lima, por todo apoio dado neste trabalho. A todos os colegas do programa CIn-Motorola pelo apoio incondicional, sempre que solicitei.

A todos, que direta ou indiretamente, contribuíram na minha formação profissional e pessoal.

# Resumo

Teste é uma atividade muito importante no processo de desenvolvimento de software, entretanto, é uma atividade cara, uma vez que ela consome uma parte considerável dos recursos de um projeto de desenvolvimento de software. Um problema encontrado na maioria dos processos de desenvolvimento de software é a ausência de uma maneira de se avaliar a efetividade dos casos de teste de unidade, que são executados no código desenvolvido. Uma possível solução para este problema é realizar testes de cobertura de código e obter métricas sobre a cobertura do conjunto de testes de unidade executados.

O presente estudo descreve um processo de desenvolvimento de software incluindo análise de cobertura de código, em que utilizamos o *Rational Unifield Process* como base para o processo proposto.

Palavras-chave: Engenharia de software, Processo de desenvolvimento de software, Qualidade de software, Testes de software, Análise de cobertura de código.

# Abstract

*Testing is a very important activity in software development process, but is also very expensive, once it consumes much of the software development process resources. A problem found in the majority of the processes of software development is the absence of a way to evaluate the unit tests executed in the developed code. A possible solution for this problem is to perform code coverage analysis and obtain metrics about set of unit tests executed.*

*The present study describes a software development process with code coverage analysis, where we use Rational Unified Process as the basis for the proposed process.*

*Keywords: Software Engineering, Software Development Process, Software Quality, Software Testing, Code Coverage Analysis.*

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviações

RUP: *Rational Unified Process*.

CIn: Centro de Informática.

UFPE: Universidade Federal de Pernambuco.

BTC: *Brazil Test Center.*

LNC: Linguagem Natural Controlada.

SWEBOK: Software *Engineering Body of Knowledge.*

IEEE: Institute of Electrical and Electronics Engineers

SQA: Software *Quality Assurance.*

BCC: *Branch Condition Coverage*.

MD/CC: *Modified Decision/Condition Coverage*.

LCSAJ: *Linear Code Sequence and Jump*.

LDRA: *Liverpool Data Research Associates*.

CM: *Configuration Management*.

IDE: *Integrated Development Environment*.

GC: Gerência de Configuração

SPEM: Software *Process Engineering Metamodel*.

UML: *Unified Modeling Language.*

OMG: *Object Management Group*.

MOF: *Meta-Object Facility*.

CR: *Change Request*.

MVC++: *Microsoft Visual* C++

# 1 Introdução

O desenvolvimento de software envolve várias atividades na sua produção, e pode haver falhas humanas na realização destas atividades. Essas falhas podem ocorrer em vários momentos do processo, principalmente no início, quando os objetivos podem ser elaborados de forma errada ou especificados imperfeitamente.

Testar é uma atividade muito importante em um processo de desenvolvimento de software. Ajuda a administrar riscos de projeto e a melhorar a qualidade do software produzido, evitando assim falhas e erros no produto durante o desenvolvimento do software. Mas testar é uma atividade muito cara e pode consumir uma quantia considerável dos recursos de um projeto. Assim, o projeto de teste deve ser bem planejado e analisado durante o processo inteiro, tornando-o mais eficiente e com menor custo.

Um problema que pode ser encontrado no processo de desenvolvimento é a ausência de uma medida quantitativa da qualidade dos casos de teste produzidos, porque, embora inspeções e revisões sejam extremamente importantes, não são muito úteis para medir a efetividade de um conjunto de testes que foi produzido. As inspeções fazem uma análise pontual no artefato a ser inspecionado, verificando apenas o padrão do código, a lógica e os comentários.

Uma técnica usada para obter essas medidas quantitativas é adotar análise de cobertura de código no processo de desenvolvimento de código, para os casos de testes elaborados na organização. Neste caso, as medidas serão obtidas enquanto os testes de unidade são executados. A análise de cobertura de código consiste em encontrar áreas do programa não executada por um conjunto de casos de testes, no capitulo dois teremos mais detalhes a respeito.

Neste capítulo, são apresentados a motivação, o escopo e os objetivos deste trabalho, bem como sua estruturação em termos de capítulos e apêndices.

## 1.1 Motivação

Atualmente, várias organizações de desenvolvimento de software desenvolvem inúmeros produtos com prazos de entrega bem apertados. Para assegurar a qualidade destes produtos, muitas organizações adotam um processo de teste rigoroso que conduz à criação e execução de diversos casos de teste para cada requisito.

Apesar do rigor na criação de casos de teste, não é possível garantir que eles cubram uma parte significativa do código a ser testado. Conseqüentemente, *bugs* podem não ser encontrados pelo processo de teste de unidade, sendo localizados apenas em fases posteriores de testes. Algumas organizações têm intensificado o trabalho de pesquisa na área de análise da cobertura de código, pois com o uso de ferramentas de análise de cobertura, pode-se medir e indicar funcionalidades e código que não estão cobertos através dos casos de teste; e isto pode dar direções para a criação de casos de teste.

Análise de cobertura de código captura informações dentro do software que está sendo testado e funciona de forma transparente quando são executados os casos do teste. Cada vez que uma função, uma linha do código, ou uma decisão lógica é executada, o analisador grava a informação. Pode-se então obter os dados estatísticos que identificam que parcelas do software foram ou não executadas [Patton, 2005]. Com estes pode-se saber:

- Que partes do software que os casos de teste não cobrem. Se um módulo específico nunca é executado, se é necessário escrever casos de teste adicionais para testar a função daquele módulo;
- Quais casos de teste são redundantes. Se dois ou mais casos de testes executam o mesmo trecho de código;
- Que novos casos de teste precisam ser criados para se ter uma melhor cobertura. Pode-se observar o código que tem uma baixa cobertura, analisando como trabalha e o que faz, e criando-se novos casos de teste que executarão mais completamente.

Um dos grandes empecilhos para a utilização da análise de cobertura de código no desenvolvimento de software é que várias organizações acreditam que estão aumentando os seus custos e o tempo de desenvolvimento ao adotarem esta técnica, consequentemente

muitas organizações preferem investir em inspeção de código e em criação de casos de testes, não avaliando a eficácia dos casos de testes já criados.

## 1.2 Escopo do trabalho

Neste trabalho, é proposto um processo de desenvolvimento de software, *Code Coverage Process*, no qual foi inserida a análise de cobertura de código no processo de desenvolvimento. O processo proposto tem como objetivo definir as atividades, os responsáveis e os artefatos gerados na implementação do código com análise de cobertura de código. O processo proposto é inspirado no *Rational Unified Process* (RUP), no qual foram adaptados quatro fluxos, alteradas e inseridas atividades e artefatos dentro do processo.

A metodologia RUP escolhida justifica-se devido à sua generecidade e por ser adaptável, bem como por reunir o melhor de várias técnicas modernas de desenvolvimento de software, sendo bem aceita tanto no meio comercial como no meio acadêmico.

A extensão dos fluxos do RUP consiste na criação de novas atividades e modificações de atividades já existentes, visando tratar características específicas da análise de cobertura de código.

Este trabalho foi realizado no *Test Research Project*, um projeto de pesquisa fruto de uma parceria entre o Centro de Informática da UFPE e o *Motorola Brazil Test Center* (BTC), onde se realizou atividades de estudo das técnicas existentes de análise de cobertura de código, bem como das ferramentas existentes no mercado. O objetivo mais amplo do projeto é contribuir com todo o processo de testes da Motorola. A intenção deste projeto é desenvolver pesquisas relativas aos processos de desenvolvimento de testes dos produtos da Motorola. Como produto desta pesquisa, a parceria pretende identificar possíveis pontos de melhoria ou investimentos nos processos. Isso inclui os seguintes objetivos específicos:

- Documentação de Requisitos – desenvolvimento de uma Linguagem Natural controlada (LNC) a ser utilizada na documentação de requisitos, com o objetivo de sistematizar a geração e seleção efetiva de casos de teste;

- Seleção de Casos de Teste – definir um procedimento para seleção de casos de teste, com o intuito de identificar efetivamente testes com potencial para descoberta de falhas na aplicação, tendo uma cobertura adequada;
- Requisitos Documentados a partir dos Testes – atualização dos documentos de requisitos a partir dos casos de teste;
- Avaliação da Suíte de Testes e Resultados – desenvolvimento de técnicas e ferramentas que possibilitem analisar parâmetros como cobertura e confiabilidade dos testes e estimar o tempo de execução dos casos de teste gerados.

A Figura 1-1 mostra o fluxo de informação do *Test Research Project*. Os quadros representam os artefatos gerados pelas atividades que estão sendo desenvolvidas no projeto. As atividades são representadas pelas setas numeradas.

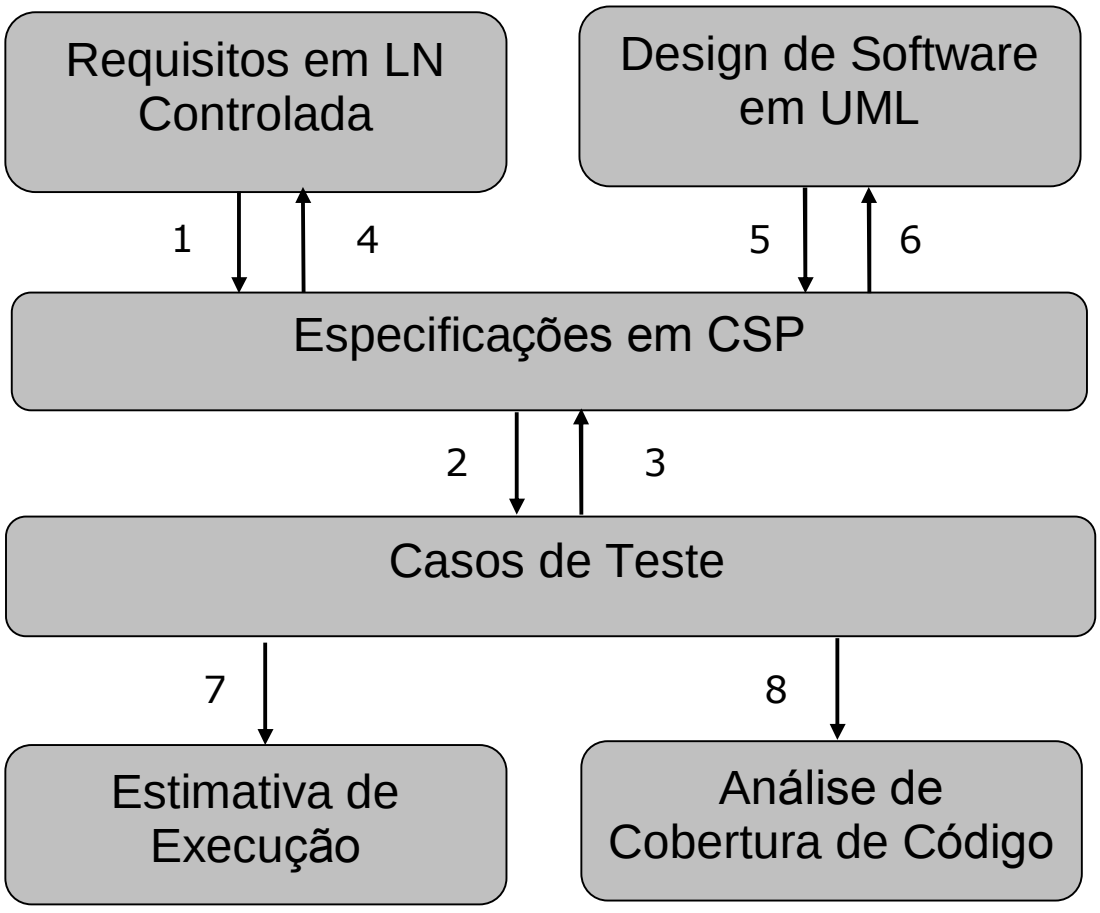



**Figura 1-1:** Fluxo de Informações do *Test Research Project*

As atividades do *Test Research Project* podem ser divididas em três grandes grupos:

- **Geração de Casos de Teste**: agrupa as atividades representadas pelas setas 1, 2 e 5, e tem como meta a geração de casos de teste a partir de documentos de requisitos e de diagramas;
- **Atualização dos Requisitos**: agrupa as atividades representadas pelas setas 3, 4 e 6, e tem como meta a atualização dos requisitos (documentos e diagramas) a partir de informações mais atuais contidas nos casos de teste;
- **Estimativas de Execução e Cobertura de Código:** agrupa as atividades 7 e 8. Recebe como entrada casos de teste e tem como meta a estimação da execução desses testes e a análise de sua cobertura de código.

O projeto surgiu da necessidade de se complementar os outros projetos do BTC. Diariamente varias equipes de desenvolvimento do BTC identificam oportunidades de melhoria no processo de desenvolvimento e execução de testes. Porém, devido à velocidade dos trabalhos e aos cronogramas apertados, estas equipes normalmente não têm tempo para explorar as possíveis soluções. O *Test Reasearch Project* aparece nesse contexto para explorar em detalhes essas oportunidades, propondo e implementando soluções para o processo de testes do BTC.

## 1.3 Objetivos do Trabalho

O objetivo maior deste trabalho é a elaboração de um processo fácil de utilizar, adaptado, através da inclusão de atividades e artefatos dentro de um processo bem consolidado, inspirado no RUP.

A proposta se dá em torno da elaboração de um processo de desenvolvimento de software, que dê suporte ao uso da análise de cobertura de código, contemplando os seguintes pontos:

- Adaptação de atividades e artefatos existentes no processo;
- Criação de novas atividades e artefatos no processo;
- Orientação de como usar a análise de cobertura de código, sem ter grandes impactos no processo normal de desenvolvimento de software;

- Utilização da técnica de análise de cobertura de código no fluxo de testes, utilizando a técnica de cobertura de código para avaliar casos de testes de caixa preta.
- Validação do processo proposto através de um estudo de caso.

## 1.4 Metodologia do Trabalho

Para a realização do objetivo proposto, inicialmente foi realizado um estudo e uma análise da técnica de cobertura de código, bem como do ambiente de desenvolvimento de software que serviu como inspiração para o trabalho. Com a aplicação em alguns projetos piloto, pôde-se identificar os pontos do processo que mereciam ser adaptados para o uso da técnica. Um dos pontos importantes do trabalho foi à inserção da técnica dentro do fluxo de testes, para que com os dados da cobertura de código possa selecionar e eliminar a redundância dos casos de testes. Por fim, com o objetivo de avaliar, melhorar, encontrar erros e validar o processo, foram realizados experimentos e avaliação qualitativa do processo.

## 1.5 Resultados e Relevância do Trabalho

Com o uso da técnica de análise de cobertura de código, apoiada pelo processo proposto, os custos relacionados a testes devem ser reduzidos, uma vez que, pode-se identificar se os casos de testes estão cobrindo o código, se existem caso de testes redundantes e se a seleção de casos de testes baseado na cobertura de código está sendo bem planejada. Possibilitando uma correção de software menos onerosa, já que as falhas são detectadas previamente, evitando assim a propagação da detecção em etapas futuras do desenvolvimento, o que tornaria mais complicado e oneroso o desenvolvimento.

## 1.6 Estrutura do Trabalho

Além deste capítulo introdutório, o trabalho consiste de mais seis capítulos e cinco apêndices. Apresentados da seguinte forma:

Capítulo 2 – Teste de Software: neste capítulo, é discutido sobre teste de software, processo de teste, técnicas de testes (caixa preta e caixa branca) e alguns tipos de teste de software. É discutido também análise de cobertura de código, alguns tipos de métricas de

cobertura de código existente, alguns trabalhos relacionados à cobertura de código na área acadêmica.

Capítulo 3 – *Code Coverage Process*: neste capítulo, apresenta-se o detalhamento das atividades e artefatos inseridos para a análise de cobertura de código.

Capítulo 4 – Extensão do RUP para incorporar o *Code Coverage Process*: apresenta uma proposta de extensão do RUP para a utilização mais apropriada da análise de cobertura de código.

Capítulo 5 – Estudo de Caso: neste capítulo, relata-se a experiência resultante do estudo de caso em torno da utilização da técnica de cobertura de código apresentada nesse trabalho, num projeto de desenvolvimento de software embarcado em uma organização real. Em seguida, é descrita a análise feita a respeito do impacto da utilização do processo proposto, através de experimentos. Por fim, é apresentada uma análise crítica do processo proposto, com o objetivo de validar a nossa proposta, no sentido de atender os objetivos estabelecidos.

Capítulo 6 - Conclusões: este capítulo conclui a dissertação com um breve resumo dos resultados obtidos, apontando algumas limitações e possíveis trabalhos futuros.

Apêndice A – Software *Process Engineering Metamodel* (SPEM): apresenta uma breve descrição do SPEM que foi utilizado para representar visualmente o processo, e atividades e artefatos inseridos no processo proposto.

Apêndice B – *Rational Unified Process*: neste apêndice, faz-se um resumo da metodologia de desenvolvimento de software RUP, através da apresentação de suas características, fases e fluxos.

Apêndice C – Atividades do RUP Estendido para *Code Coverage Proces:* apresenta as atividades, bem como seus passos correspondentes, dos fluxos de Implementação, Teste, Gerenciamento de Configuração e Mudança, e Gerenciamento de Projeto para o *Code Coverage Process.*

Apêndice D – Questionário de análise crítica do *Code Coverage Process*: apresenta o modelo do questionário utilizado para realização da análise crítica.

Apêndice E – Ferramentas de Cobertura de Código: descreve algumas ferramentas de cobertura de código disponível no mercado.

# 2 Teste de Software

Teste de software é uma atividade importante no processo de desenvolvimento de software, tendo como objetivo verificar a corretude da implementação de um sistema. Segundo [Inthurn, 2001], pode-se afirmar que teste é "uma das áreas da engenharia de software que tem como objetivo aprimorar a produtividade e fornecer evidências da confiabilidade e da qualidade do software em complemento a outras atividades de garantia de qualidade ao longo do processo de desenvolvimento de software".

A realização de testes é crucial para o desenvolvimento de software com qualidade e consome muito esforço e tempo de projeto. "Os custos com teste variam de 25% a 50% do orçamento total em muitos projetos de desenvolvimento de software." [Li, 2004].

O SWEBOK [IEEE, 2001] define teste de software como uma atividade executada para a avaliação e a melhoria da qualidade de um produto, identificando defeitos e problemas. Teste de software consiste na verificação dinâmica do comportamento de um programa através de um conjunto finito de casos de teste, apropriadamente selecionado.

## 2.1 Visão Geral

Segundo [Patton, 2005], para ter um teste eficiente, é importante ter pelo menos uma compreensão geral do processo de desenvolvimento de software. A criação de um novo produto de software pode envolver vários membros da equipe exercendo diferentes papéis e trabalhando simultaneamente dentro de um curto período de desenvolvimento.

O objetivo deste capítulo é apresentar uma introdução referente a conceitos de testes, processo, medição de software, para que possamos em seguida explanar sobre análise de cobertura de código. O objetivo é dar uma vista geral de todas as partes que entram em um produto de software e a importância de se ter qualidade no processo de desenvolvimento, realizando medições da qualidade do software.

## 2.2 Teste de Software

É uma das fases do processo que visa atingir um nível de qualidade do produto, cujo objetivo é encontrar defeitos, para que se possa corrigi-los antes da entrega. No processo de desenvolvimento de software existe a possibilidade de haver falhas humanas, mesmo que o processo esteja bem elaborado. Dentro da garantia de qualidade de software (SQA), existe o processo de teste, o qual é elaborado cuidadosamente e bem planejado, sendo sua elaboração justificada pelos custos associados às falhas de software.

Teste de software é um processo, ou uma série de processos, projetado para garantir que o código implementado faça o que foi projetado [Myers, 2004]. As principais finalidades de testes software:

- Verificar se o que foi implementado está de acordo com todos os requisitos do sistema;
- Assegurar a qualidade e a corretude do software produzido;
- Reduzir os custos de manutenção corretiva e trabalho refeito;
- Obter a satisfação do cliente com o produto desenvolvido;
- Produzir casos de teste que têm elevada probabilidade de revelar um erro, ainda não descoberto, com uma quantidade mínima de tempo e esforço;

- Indicar a qualidade e a confiabilidade do software com base na comparação do resultado dos testes com os resultados esperados;
- Verificar a correta integração entre os componentes de software.

O processo de desenvolvimento de software envolve uma série de atividades nas quais, apesar das técnicas, métodos e ferramentas empregados, erros no produto ainda podem ocorrer. Atividades agregadas sob o nome de Garantia de Qualidade de Software têm sido introduzidas ao longo de todo o processo de desenvolvimento, entre elas atividades de VV&T – Verificação, Validação e Teste, com o objetivo de minimizar a ocorrência de erros e riscos associados [Maldonado, 2000]. Os objetivos de Verificação e Validação são:

- Verificação se refere ao conjunto de atividades que garante que o software implemente corretamente uma função especifica;
- Validação se refere a um conjunto diferente de atividades que garante que o software que foi construído atendendo a especificação do cliente.

Os procedimentos de verificação e validação são complementares, não sendo mutuamente exclusivos. "Os de verificação podem ser aplicados no inicio e ao longo de todo o ciclo de vida do processo de desenvolvimento, enquanto que os de validação tendem a ocorrer de uma forma concentrada e intensiva nas fases que abrangem os testes modulares e de integração" [Inthurn, 2001].

Os testes são elaborados para encontrar erros, apesar de muitos pensarem o contrário disto, visto que, alguns têm a visão de teste bem sucedido aquele, no qual, não foi encontrado nenhum erro. Segundo [Myers, 2004], os principais objetivos de teste são:

- Executar um programa com a intenção de descobrir um erro;
- Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;
- Um teste bem-sucedido é aquele que revela um erro ainda não descoberto.

## 2.3 Técnicas de Testes

Existem os seguintes tipos de Técnica ("Abordagem") de teste:

**Técnica Funcional (“caixa preta”)**

São gerados com base nos requisitos coletados com os usuários, através de uma análise entre os dados de entrada e de saída. Os erros encontrados são: (1) funções incorretas ou ausentes; (2) erros de interface; (3) erros nas estruturas de dados ou no acesso a bancos de dados externos; (4) erros de desempenho; e (5) erros de inicialização e término. “Os métodos de teste de caixa preta concentram-se nos requisitos funcionais do software” [Pressman, 2006].

“O teste funcional concentra-se nos requisitos funcionais do software. Através dele torna-se possível verificar as entradas e saídas de cada unidade” [Inthurn, 2001]. Esta técnica trabalha com o que, e não com o está sendo realizada função. “Os dados dos testes são derivados unicamente das especificações (isto é, sem fazer uso do conhecimento da estrutura interna do programa)” [Myers, 2004].

Os testes funcionais têm duas vantagens distintas [Jorgensen, 1995]:

1. São independentes de como o software é implementado, assim se a implementação mudar, os casos de testes continuam inalterados e úteis;
2. O desenvolvimento dos casos de teste pode ocorrer em paralelo com a implementação, reduzindo desse modo o tempo total do desenvolvimento do projeto.

A desvantagem é que os casos de teste funcionais sofrem freqüentemente de dois problemas:

1. Pode haver redundâncias significativas entre casos do teste, devido a falta de uma verificação que possa comparar casos de testes com o mesmo objetivo;
2. Pode ter a possibilidade de não testar o software.

**Técnica Estrutural (“caixa branca”)**

São gerados a partir de uma análise dos caminhos lógicos possíveis de serem executados, como forma de conhecer o funcionamento interno dos componentes do software.

Os casos de teste caixa branca servem para: (1) garantir que todos os caminhos independentes, dentro de um módulo, tenham sido exercitados pelo menos uma vez; (2) exercitar todas as decisões lógicas com os valores falsos ou verdadeiros; (3) executar todos os laços nas suas fronteiras e dentro dos limites e (4) exercitar as estruturas de dados internos, garantindo a sua validade.

Na técnica estrutural utilizando a teoria de grafos, o testador pode descrever rigorosamente o que esta sendo testado, com os conceitos de teoria de grafos, teste estrutural proporciona o uso de métricas da cobertura de teste. As métricas de cobertura de teste fornecem uma maneira de indicar explicitamente o nível a que um software foi testado, e esta por sua vez, faz o gerenciamento do teste [Jorgensen, 1995].

## 2.4 Estágios de Teste

Existem diversos estágios de teste, a saber.

**Teste de Unidade**

É o processo de exercitar uma porção individual do código, um componente, determinando se o mesmo funciona corretamente. Quase todos desenvolvedores executam algum nível de teste de unidade relativo a um componente ou pedaço de código. Testes de unidade e integração são fundamentais para a entrega de um produto de software de qualidade; contudo eles são freqüentemente negligenciados, ou são implementados de uma maneira superficial [Dustin, 2002].

O teste de unidade concentra esforços na menor unidade do projeto de software, ou seja, procura identificar erros de lógica e de implementação em cada módulo do software, separadamente [Maldonado, 2000].

Segundo [Hunt, 2003], teste de unidade deve ser feito em um código escrito por um desenvolvedor que exercita uma área muito pequena, especifica da funcionalidade do

código a ser testado. Por exemplo, você poderia acrescentar um valor numa lista ordenada, então confirmaria se este valor aparece realmente no termino da lista.

Os testes de unidade verificam os seguintes pontos:

- Os componentes individuais para ter a certeza que os mesmos funcionam de forma correta;
- A interface dos componentes para a garantia de que as informações estão fluindo normalmente;
- A manipulação de dados inconsistentes;
- A inicialização das variáveis;
- Se as condições de limite estabelecido estão operando adequadamente.

Segundo [Myers, 2004], as motivações para realizar testes de unidade são: (1) um modo de administrar os elementos, no qual é focada atenção inicial em unidades menores do programa. (2) Tem a facilidade de depurar (processo de definir e corrigir um erro descoberto), pois quando é localizado um erro, é conhecido o modulo no qual ele pertence. (3) A possibilidade de paralelismo no processo de teste do programa, no qual pode testar múltiplos módulos simultaneamente.

De acordo com [Patton, 2005], esta estratégia de teste facilita isolar *bugs*. Quando um problema é encontrado em nível de unidade, o problema deve estar naquela unidade. Se *bug* é encontrado, quando múltiplas unidades forem integradas, deve-se analisar como os módulos interagem. Evidentemente, podem-se encontrar exceções, mas no geral, testar e depurar é muito mais eficientes do que testar tudo imediatamente.

Segundo [Dustin, 2002], os testes de unidade devem ser escritos em uma linguagem apropriada, capaz de testar o código ou componente em questão. Por exemplo, se o colaborador escrever um conjunto de classes C++ para resolver um problema ou uma necessidade particular, o teste de unidade provavelmente deve também ser escrito em C++ a fim exercitar as classes.

Este tipo de teste é realizado geralmente pelo desenvolvedor. Para testar um modulo geralmente se faz necessário:

- *Driver*: é o programa principal que aceitam dados para serem passados para o modulo que estar sendo testado;
- *Stubs*: são os módulos que irão substituir os módulos subordinados.

Unidade não significa apenas uma classe em uma linguagem orientada a objetos, mas também subprogramas livres, como funções em C++.

Testes de unidade e de integração são fundamentais para a entrega de um produto de software de qualidade. Se testes de unidade forem realizados corretamente, as fases de testes posteriores serão mais bem sucedidas. Para realizar um teste de unidade estruturado e repetível, programa de software de teste de unidade deve ser desenvolvido, antes ou paralelamente com o desenvolvimento do software.

Para [Dustin, 2002], os testes de unidade são considerados parte do projeto de desenvolvimento e são atualizados, junto com os requisitos e o código fonte, à medida que o projeto evolui. Descobrir defeitos enquanto um componente ainda está na fase de desenvolvimento oferece uma economia significativa de tempo e custo.

Os engenheiros de software e os programadores devem ser responsáveis pela qualidade do seu trabalho. Muitos acham que não são responsáveis por testar o código, dando como justificativa que é trabalho da equipe de testes. Na realidade, os programadores devem ser responsáveis por produzir produtos de alta qualidade que aderem aos requisitos. Liberar o código para a equipe de testes, quando o mesmo tem um número elevado de defeitos, resulta geralmente em ciclos longos de correções, podendo ser evitado com uso apropriado de testes de unidade.

Segundo [Dustin, 2002], as vantagens de se ter testes de unidade durante o desenvolvimento são as seguintes:

- O desenvolvimento será em busca de executar os testes de unidade, no qual estará atendendo cada requisito, logo o software será considerado completo quando executar com sucesso o teste de unidade;
- Fará com que o desenvolvedor focalize os esforços em satisfazer ao problema exato, tendo geralmente como resultado menos código e uma execução mais direta.

De acordo com [Dustin, 2002], muitos engenheiros de software não elaboram os testes de unidade de maneira uniforme. Padronizar e otimizar os testes de unidade pode reduzir o tempo do seu desenvolvimento e evitar formas diferentes de utilizá-los.

O desenvolvedor do componente, que escreve o teste de unidade, está em uma boa posição para atualizá-lo também, enquanto as modificações são realizadas no código. Estas alterações podem ser provenientes de melhorias e de reestruturação, de defeitos, ou de uma mudança dos requisitos. Logo, o desenvolvedor, que é responsável pelo código, também será pelo teste de unidade, resultando com isso testes atualizados e úteis.

Existem algumas ferramentas comerciais de testes, que elaboram e executam os testes de forma automática, sendo que essas ferramentas possuem muitas funcionalidades que acabam não sendo utilizadas, tendo um custo significativo para o projeto. Neste caso deve-se estudar se não é mais viável a construção de uma ferramenta feita sob medida ou, se em longo prazo, justifica comprar uma ferramenta.

**Teste de Integração**

Testa a interface entre as unidades integradas. É uma técnica sistemática, para construção da estrutura do programa, realizando, ao mesmo tempo, testes para descobrir erros associados a interfaces. “Verifica basicamente se as unidades testadas de forma individual executam corretamente quando colocadas juntas, isto é, quando integradas” [Inthurn, 2001].

**Teste de Sistema**

De acordo com [Myers, 2004], a maioria das pessoas tem o conhecimento errado sobre este tipo de teste, pois pensam que é um processo que testa a funcionalidade do sistema ou programa por completo, logo seria redundante com o processo de teste funcional. Teste de sistema tem uma finalidade em particular: comparar o sistema ou programa com os seus objetivos originais.

São testados como um todo, elementos de software integrados com o ambiente operacional (hardware, pessoas, etc.). Geralmente é um teste “caixa preta”, executado por um testador de sistemas.

O teste de sistema é uma atividade de validação usada para demonstrar que o software inteiro esta correto. O objetivo é descobrir implementações incorretas dos requisitos especificados [Inthurn, 2001].

## 2.5 Processo de Teste

O processo de teste tem como objetivo criar testes para revelar a presença de falhas. Ele define como os testes serão planejados e executados através de atividades e passos, e quando serão executados. Quando o processo é bem controlado e planejado, exige menos esforço e tem maior eficácia.

Um processo de teste incorpora algumas atividades:

- Planejamentos dos testes: os requisitos são testados e priorizados; são elabora as estratégicas de testes, cronograma, esforço e recursos exigidos;
- Projeto dos testes: elaboração dos casos de testes, bem como os seus procedimentos;
- Implementação dos testes: criação de *scripts* e componentes de teste;
- Execução dos testes: executado de forma manual e/ou automática;
- Avaliação dos testes: avalia a sua tendência de defeitos e seus critérios de sucesso.

**Caso de Teste**

Um caso de teste é um documento que descreve uma entrada, uma ação ou evento e uma resposta esperada, com intuito de determinar se o que foi especificado está sendo executado corretamente na aplicação. Um caso de teste deve conter identificação, objetivos, condições de entrada, seqüência de passos e resultados esperados.

A essência de um teste de software é determinar um conjunto de casos de teste para um item que é testado. Segundo [Jorgensen, 1995], Um caso de teste é composto por entradas e saídas, como especificado a seguir:

Entradas:

- Pré-condição: garante a condição inicial para a execução do caso de teste;

- Passos atuais: passos a serem executados, identificados pelos métodos de teste.

Saídas:

- Saída Esperada: respostas esperadas do sistema, para a entrada atual;
- Pós-condição: representa o estado final do sistema.

O ato de testar envolve estabelecer as pré-condições necessárias, fornecer as entradas do caso de teste, observar as saídas, e então comparar estas com as saídas previstas para determinar se o teste passou ou não.

## 2.6 Medição de Software

Podemos medir a qualidade do software ao longo do seu processo de desenvolvimento, e depois que o produto for entregue ao cliente e aos usuários. As métricas coletadas antes da entrega oferecem uma base quantitativa para tomadas de decisão referentes ao projeto e aos testes. Métricas coletadas, após a entrega, concentram-se no número de defeitos descobertos e na manutenibilidade do sistema, essas métricas fornecem uma indicação *post-mortem* da efetividade do processo de software.

Segundo [Goodman, 1993] define métricas de software como: "A aplicação contínua de técnicas de medições aos processos de desenvolvimento e aos produtos de software para o fornecimento de informações gerenciais, que conseqüentemente são utilizadas para a melhoria dos processos e da qualidade do produto".

Uma métrica é a medição de um atributo (propriedades ou características) de uma determinada entidade (produto, processo ou recursos). Exemplos:

- Tamanho do produto de software (ex: Número de Linhas de código);
- Número de pessoas necessárias para implementar um caso de uso;
- Número de defeitos encontrados por fase de desenvolvimento;
- Esforço para a realização de uma tarefa;
- Tempo para a realização de uma tarefa;
- Custo para a realização de uma tarefa;

- Grau de satisfação do cliente (ex: adequação do produto ao propósito, conformidade do produto com a especificação).

Segundo o [IEEE, 1992] "a utilização de métricas reduz a subjetividade na avaliação da qualidade de software, através do fornecimento de informações quantitativas a respeito do produto e do processo, além de tornar a qualidade do software mais visível". Podemos citar alguns motivos para realização da medição de software:

- Entender e aperfeiçoar o processo de desenvolvimento;
- Melhorar a gerência de projetos e o relacionamento com clientes;
- Reduzir frustrações e pressões de cronograma;
- Gerenciar contratos de software;
- Indicar a qualidade de um produto de software;
- Avaliar a produtividade do processo;
- Avaliar os benefícios (em termos de produtividade e qualidade) de novos métodos e ferramentas de engenharia de software;
- Avaliar retorno de investimento;
- Identificar as melhores práticas de desenvolvimento de software;
- Embasar solicitações de novas ferramentas e treinamento;
- Avaliar o impacto da variação de um ou mais atributos do produto ou do processo na qualidade e/ou produtividade;
- Formar uma *baseline* para estimativas;
- Melhorar a exatidão das estimativas;

Oferecer dados qualitativos e quantitativos ao gerenciamento de desenvolvimento de software, de forma a realizar melhorias em todo o processo de desenvolvimento de software.

## 2.7 Análise de Cobertura de Código

A análise de cobertura do código é um tipo de técnica usada em teste de caixa-branca, cujo objetivo é verificar como o conjunto de testes exercita partes do código. Portanto, é utilizada para averiguar a qualidade do conjunto de testes e não a qualidade do produto de software.

É utilizada principalmente na fase de teste de unidade, no qual requer a cobertura dos caminhos possíveis dentro de cada unidade do programa. As vantagens de se usar cobertura de código dentro do processo de desenvolvimento são:

- Ajuda a administrar riscos, pois fornece dados precisos referentes à cobertura do código, mediante a execução dos testes de unidade, fazendo com que as possíveis falhas nas partes em que o código não foi testado, não sejam encontradas em fases posteriores;
- Fornece informação referente a casos de testes que deveriam existir para se ter uma cobertura satisfatória ou a casos de testes redundantes;
- Suporte a garantia do processo resultando na sua melhoria, pois as falhas que deveriam ser encontradas na fase de testes de unidade não serão passadas para as demais fases de testes.

Análise de cobertura de código é o processo de encontrar código exercitado por um conjunto particular de entradas de teste, é um importante componente de desenvolvimento e verificação de software. A maioria dos métodos tradicionais de implementar ferramentas de análise de cobertura de código é baseada em instrumentação de programa. Este método ocasiona tipicamente um *overhead*, devido à inserção e a execução do código da instrumentação, e não são *deployable* em muitos ambientes de software [Shye, 2005].

## 2.8 Tipos de Cobertura de Código

A seguir serão mostrados e descritos alguns tipos de cobertura de código.

**a) *Statement Coverage***

A *statement coverage* requer a execução de cada linha do código ao menos uma única vez. É a medida mais simples de cobertura, mas é também a mais fraca, porque é insensível

às estruturas de fluxo do controle e operadores lógicos. A *statement coverage* analisa a estrutura de *loop* no código ao menos uma vez.

***b) Branch Coverage***

Em *branch coverage*, "os casos de teste são escritos para assegurar que cada expressão lógica em estruturas de controle é avaliada pelo menos uma vez verdadeiro e falso" [Lewis, 2000]. Geralmente satisfaz *statement coverage* desde que todo *statement* esteja no mesmo caminho de *branch coverage*. É conhecida também como *decision coverage, all-edges coverage, basis path coverage, decision-decision-path testing* ou *complete coverage*.

A principal desvantagem desta medida é ignorar ramos de expressões booleanas, o qual pode ser observado no fragmento de código da Figura 2-1. Esta medida pode considerar a estrutura de controle executada completamente sem a chamada para function1. A expressão é verdadeira quando condition1 e condition2 são verdadeiras, e a expressão possui resultado contrário quando condition1 é falso, não analisando a function1, devido que os operadores booleanos impossibilitam uma chamada para function1.

```
if (condition1 && (condition2 ||
function1()))
        statement1;
else
        statement2;
```

**Figura 2-1:** Exemplo de um fragmento de código para *branch coverage*

***c) Condition Coverage***

"Em *Condition Coverage,* os casos de teste são escritos para assegurar que cada condição em uma decisão assuma pelo menos uma vez todos os possíveis resultados". [Lewis, 2000] Avalia as sub-expressões independentemente e garante que toda condição dentro de uma decisão está coberta.

O método de criar casos de testes que utilizam esta técnica é construir uma tabela verdade e escrever todas as condições. Se existirem casos de testes duplicados, poderão ser eliminados. Considere o trecho do programa da Figura 2-2.

```
Do While not EOF
    read record
    if FIELD_COUNTER > 7 then
            increment COUNTER_7 by 1
    else
            if FILED_COUNTER > 3 then
                    increment COUNTER_3 by 1
            else
                    increment COUNTER_1 by 1
            endif
    endif
End_While
```

**Figura 2-2:** Exemplo de um fragmento de código para *Condition Coverage*

Fonte: [Lewis, 2000]

Os casos de testes que satisfazem a *condition coverage* são mostrados na tabela 2-1. Podemos observar que os casos de testes 2 e 3 são redundantes, logo um deles pode ser eliminado, resultando em três casos de testes.

**Tabela 2-1:** Entrada do caso de teste para *condition coverage*

| Caso de Teste | Condição | Valor |
|---|---|---|
| 1 | >7 | 8 |
| 2 | <=7 | 7 |
| 3 | >3 | 6 |
| 4 | <=3 | 3 |

**d) *Branch Condition Coverage* (BCC)**

"Em *Branch Condition Coverage*, os casos de teste são escritos para assegurar que cada decisão e condição dentro de uma decisão assumem pelo menos uma vez todos possíveis valores" [Lewis, 2000]. É composto pela união de *branch* e *condition coverage*. Tem a vantagem de ser simples. É mais forte que *branch* ou *condition coverage*, pois verifica todas as condições que podem não ser testadas com *branch coverage*.

**e) Multiple Condition Coverage**

A *multiple condition coverage* requer avaliação de toda possível combinação de sub-expressões booleanas que acontecem no código. Uma vantagem da *multiple condition coverage* é que requer um teste muito completo, pois requer 2n de casos de teste para alcançar 100% de cobertura de uma condição que contém n operandos booleanos. Tem a desvantagem de se tornar inatingível rapidamente para condições mais complexas. E também a desvantagem que o número de casos de teste pode variar muito entre condições que têm complexidade semelhante. Para algumas linguagens a *multiple condition coverage* é muito similar a *Condition Coverage*.

**f) *Modified Decision/Condition Coverage* (MD/CC)**

A *modified decision/condition coverage* (MD/CC) exige bastantes casos de teste para mostrar que cada operador booleano, em uma expressão booleana, pode afetar o resultado da decisão independentemente.

Todo ponto de entrada e de saída dentro do software é invocado pelo menos uma vez, toda condição em uma decisão no software executa pelo menos uma vez todos os possíveis resultados, toda decisão executa pelo menos uma vez todos os possíveis resultados.

**g) *Function Coverage***

*Function coverage* "verifica se cada função ou procedimento é invocado. É útil durante testes preliminares para assegurar ao menos alguma cobertura em todas as áreas do software. Teste superficial que encontra deficiências brutas em um suíte de teste rapidamente" [Cornett, 2005].

**h) *Path Coverage***

*Path Coverage* relata se cada um dos trajetos possíveis em cada função foi seguido. Um caminho é uma seqüência original do ramo de entrada da função à saída [Cornett, 2005].

É um teste de cobertura muito completo, por outro lado, tem um problema que o número de trajetos é exponencial em relação ao número de ramos, por exemplo: uma função que possui 10 *if-statement* tem 1024 trajetos a testar, se adicionamos apenas mais um *if-statement* irá dobrar a contagem para 2048. Uma outra desvantagem é que alguns trajetos são impossíveis de serem executados, devido ao relacionamento dos dados. Considere o fragmento de código da Figura 2-3. *Path coverage* considera que este fragmento contém quatros trajetos, sendo que somente dois são praticáveis: *sucess = false* e *sucess = true*.

```
if (success)
   statement1;
   statement2;
if (success)
   statement3;
```

**Figura 2-3**: Exemplo de um fragmento de código para *Path Coverage*

Fonte: [Cornett, 2005]

***i) Linear Code Sequence and Jump Coverage* (LCSAJ)**

"Um LCSAJ é uma sucessão linear de código executável que começa desde o início do programa ou de um ponto do código para o qual o controle de fluxo pode pular e/ou é concluído por um *jump* de fluxo de controle específico ou ao final do programa. Uma seqüência linear de código consiste de um ou mais blocos básicos sucessivos. Por conseguinte pode haver vários predicados que devem ser satisfeitos para que o controle de fluxo possa executar a sucessão de código linear terminando no *jump*" [Ldra, 2005].

LCSAJC é uma variação de *path coverage* que testa todos os blocos no código. Este é mais completo que *branch coverage* e evita o problema exponencial de *path coverage*, mas não evita os caminhos impraticáveis.

## 2.9 Relação entre as métricas de cobertura

Todas as métricas de cobertura são relatadas em porcentagem (entre 0 e 100). A porcentagem indica a proporção do código instrumentado que foi executado. Quando porcentagem for -1 indica uma condição de erro.

As métricas de cobertura são relacionadas no sentido de alcançar uma forma de cobertura que implica em uma ou mais outras formas de cobertura também. A Figura 2-4 mostra estes relacionamentos.

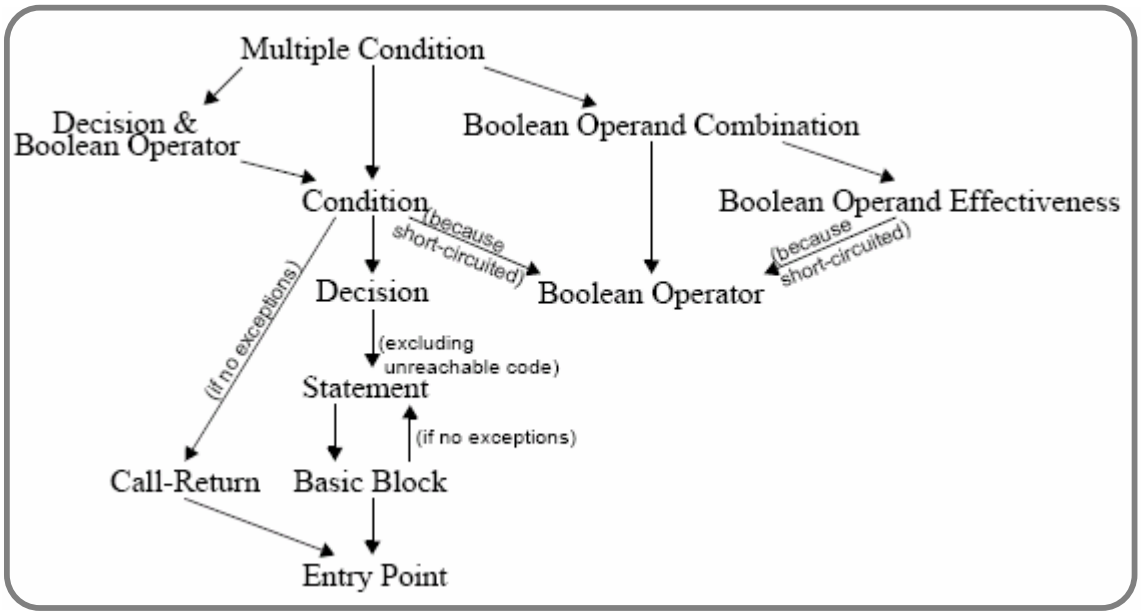



**Figura 2-4**: Relação entre as métricas de cobertura de código

Note que 100% de *decision coverage* implica em 100% de *statement coverage*, mas 100% de *statement coverage* não implica em 100% de *decision coverage*. Logo *decision coverage* é uma medida mais forte do que *statement coverage*. Na figura 2-4 podemos observar que multiple condition coverage é a forma de cobertura mais forte e a *entry point coverage* é a mais fraca.

## 2.10 Trabalhos Relacionados à Cobertura de Código

Alguns estudos foram realizados referentes à aplicação de análise de cobertura de código em projetos de desenvolvimento de software, alguns se referem aos problemas aplicar em desenvolvimentos em grande escala como em [Kim, 2003] que descreve os seguintes pontos: que a análise de grandes sistemas de softwares industriais é muito cara. A

instrumentação do código fonte é necessária, podendo ser feita em alguns módulos do sistema, mas em grande escala consome muito tempo; o ciclo de desenvolvimento de software atual é curto para se utilizar a análise de cobertura, pois existem muitas entregas de versões do sistema. Um dos motivos para não se utilizar a análise de cobertura é que maior parte da comunidade a considera como uma prática que adiciona um custo extra que não obtém retorno de investimento.

Segundo [Gokhale, 2005], no seu trabalho são mostrados alguns critérios e formas de se avaliar casos de teste. Os critérios baseados em código são denominados de baseados em caminho, no qual os casos de testes são projetados para executarem pelo menos uma vez todos os caminhos ao longo do controle gráfico do fluxo do programa. Em fluxo de dados baseado em testes o modelo gráfico é colocada informações referente à definição e utilização das variáveis no programa.

No trabalho realizado por [Elbaum, 2001], é investigado o impacto das informações provenientes da análise de cobertura de código perante a evolução do software, foram feitos dois estudos empíricos. No primeiro executou um estudo de caso em uma aplicação, que possua varias versões e suíte de testes. O segundo estudo executou uma experiência controlada na qual foram feitas simulações de alterações no software, com o intuito de verificar os resultados da análise da cobertura de código.

No trabalho de [Agrawal, 2002] é apresentada uma técnica desenvolvida para encontrar um pequeno subconjunto de comandos e pontos de decisão no programa com o intuito de que qualquer suíte de teste possa exercitar todos os comandos ou decisão do código. Essa técnica foi utilizada em alguns sistemas de software e foi comprovado um aumento no percentual de cobertura de comando e pontos de decisões em comparação a outras técnicas existentes. Outra constatação foi economia do tempo do usuário para utilização da cobertura de código e redução dos números de pontos de verificação.

No trabalho de [Piwowarski, 1993] é discutido sobre a medida de cobertura que fornece um *feedback* do produto aos desenvolvedores no qual incentivará a aumentar a cobertura adicionando novos casos de testes. Este aumento da cobertura, por sua vez, resulta em uns custos baixos na remoção de erro. Conseqüentemente, toda a avaliação do

valor requererá que um modelo do relacionamento entre a cobertura e a qualidade de produto esteja desenvolvido.

Uma ferramenta de cobertura pode ajudar para as usuárias a fazer análise provendo as seguintes operações nos modelos de cobertura: selecionando um subconjunto de tarefas de cobertura que satisfazem determinados critérios baseado em valores de atributo específicos ou combinações de valor, ou no próprio dados de cobertura; projetando o modelo de cobertura sobre um subconjunto de seus atributos de forma que alguns atributos são ignorados essencialmente; se agrupando os dados de cobertura de valores de atributo que pertencem a uma partição de atributo.

## 2.11 Considerações Finais

Neste capítulo discutiu-se sobre teste de software, processo de teste, as abordagens de caixa preta e caixa branca e alguns estágios de teste de software. Foi discutida também a análise de cobertura de código, descritos alguns tipos de cobertura existentes e apresentados alguns trabalhos relacionados à cobertura de código na área acadêmica.

O próximo capítulo apresenta em detalhes o *Code Coverage Process*, os conceitos chave importantes para o uso das técnicas de cobertura de código, as atividades inseridas nos fluxos que foram estendidos do RUP, a descrição dos artefatos que serão criados, os responsáveis envolvidos com o processo e as etapas de definição e análise das métricas e limiares de cobertura.

# 3 *Code Coverage Process*

Considerando a relevância dos testes, em todo o ciclo do desenvolvimento de software, este capítulo tem como objetivo apresentar uma solução inserindo e/ou adaptando atividades dentro de um processo de desenvolvimento de software. O *Code Coverage Process* utiliza atividades e artefatos de cobertura de código incorporados dentro de um processo de desenvolvimento de software.

Os fluxos de Implementação, Teste, Gerenciamento de Configuração e Mudança e Gerenciamento de Projeto do RUP foram adaptados para atender mais apropriadamente o *Code Coverage Process*. A adaptação se deve à realização do projeto de pesquisa, no qual inicialmente foi feito um estudo de caso para em seguida ser utilizada na elaboração deste processo.

Neste capítulo são descritas as atividades do *Code Coverage Process*, cujo propósito é atender a realização de cobertura de código inserido no processo de desenvolvimento de software, tendo como objetivo definir as atividades, os responsáveis e os artefatos gerados na implementação do código com análise de cobertura de código.

## 3.1 Visão Geral

O processo proposto, *Code Coverage Process*, aborda os desenvolvimentos iterativos, sendo guiado pela produção de artefatos. O objetivo foi tornar o processo o mais simples e genérico para atender diversos domínios e tipos de aplicações. Assim sendo, a principal contribuição deste processo é fornecer um conjunto coerente de atividades e artefatos direcionados para o desenvolvimento de testes, utilizando a técnica de análise de cobertura de código.

## 3.2 Processo de Software

Possui uma documentação, que define os seguintes aspectos: o que é feito (produto), quando (passos), por quem (responsáveis), os insumos (recursos) e o que produz (resultados). "Os passos de um processo podem ter ordenação apenas parcial, o que pode permitir paralelismo entre alguns passos" [Filho, 2001].

Na engenharia de software, o objetivo é o desenvolvimento de um produto de software. Na Engenharia de Processos, o objetivo é o desenvolvimento de um modelo de processo. No processo de desenvolvimento de software, o ponto de partida para a arquitetura de um processo é a escolha de um modelo de ciclo de vida.

Um processo de desenvolvimento define o fluxo das atividades, os artefatos e os envolvidos na realização do trabalho. Segundo [Conallen, 2000], o processo de desenvolvimento de software possui os seguintes objetivos:

- Prover direção sobre a ordem das atividades do time;
- Especificar quais artefatos devem ser desenvolvidos;
- Direcionar as tarefas de desenvolvedores e do time como um todo;
- Oferecer critérios para monitorar os produtos do projeto e das atividades.

## 3.3 Conceitos-chave

A seguir serão apresentados alguns conceitos-chave necessários para utilização do processo proposto.

### 3.3.1 Instrumentação do código

Antes de poder analisar o código fonte, a ferramenta de cobertura necessita instrumentá-lo. A instrumentação consiste em introduzir pontos de verificação em partes específicas (estratégicas) do código. Os pontos de verificação fornecerão a informação se o teste está realmente executando os trechos do código e onde foram inseridos os pontos de verificação. A instrumentação não irá alterar a execução do fluxo normal do código, ou seja, não irá alterar a seqüência de execução dos comandos e nem o resultado obtido pela computação dos comandos.

### 3.3.2 Arquivo de Histórico da Execução do Código

Algumas ferramentas de análise de cobertura de código geram um arquivo de histórico da execução do código, o qual terá as informações da execução do código fonte. O usuário deverá fornecer o local onde o arquivo será armazenado, a fim de que se possa depois capturá-lo para a geração do relatório com as métricas de cobertura.

Este arquivo de histórico de execução é gerado automaticamente quando o código instrumentado é executado utilizando os casos de testes. Neste arquivo, são registrados todos os pontos de verificação nos quais o código foi exercitado. A Figura 3-1 ilustra a geração do arquivo de histórico. No estudo de caso apresentado no Capítulo cinco (5) foi utilizada uma ferramenta de cobertura, a qual gera um arquivo de histórico.

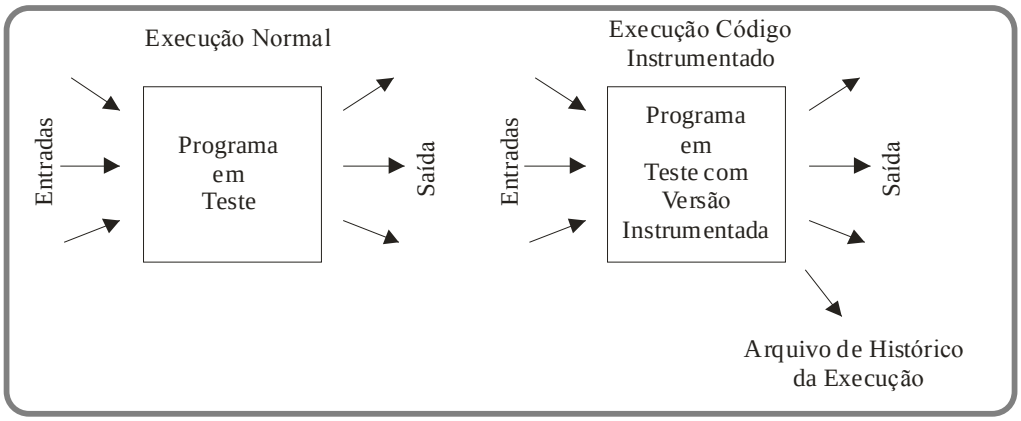



**Figura 3-1:** Geração do arquivo histórico

Outras ferramentas por sua vez geram automaticamente através de scripts o relatório de cobertura, sem a necessidade de ter arquivo de histórico da execução, bastando apenas que o usuário indique o local do repositório de armazenamento, no qual ficará o relatório. Neste caso a utilização da cobertura de código para testes que não sejam de unidade, fica inviável, devido que a maioria das ferramentas gera automaticamente o relatório de cobertura no momento das execuções dos testes de unidade.

### *3.3.3 Guidelines*

*Guidelines* são um conjunto de regras, recomendações, ou métodos de suporte a atividades e etapas. Descrevem técnicas específicas para criar certos artefatos ou a transformação de um artefato em outro. *Guidelines* podem também ser usados para validar a qualidade de um artefato na forma de um *checklist* associado com os artefatos ou para revisão de atividades.

## 3.4 Atividades de Cobertura de Código

As atividades incluídas no *Code Coverage Process*, são as seguintes: "Configuração da Cobertura"; "Instrumentação do Código"; "Geração do Relatório de Cobertura" e "Análise dos Resultados de Cobertura". Estas atividades são simples de se realizar e, dependendo da ferramenta utilizada, algumas podem ser realizadas de forma automática através de comandos (scripts). A seguir tem-se a descrição destas atividades, as quais estão ilustradas na Figura 3-2:

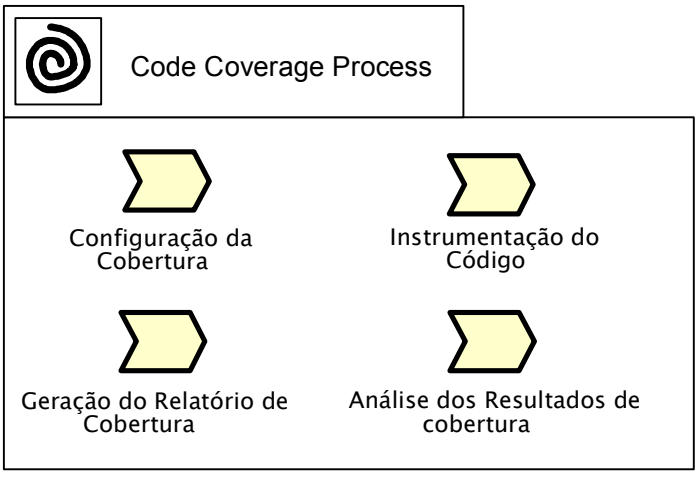



**Figura 3-2:** Atividades de cobertura de código

### 3.4.1 Configuração da Cobertura

A ferramenta de cobertura deve estar configurada no ambiente de desenvolvimento e, se possível, integrada com outras ferramentas do ambiente. Existem algumas ferramentas que possibilitam ao usuário a opção de configuração de métricas, neste caso deve-se definir e configurar os limites de cobertura desejável e os tipos de métricas que deverão ser coletadas.

**Tabela 3-1:** Configuração da Cobertura

<table>
<tr><td colspan="2">Propósito<br>Integrar a ferramenta de cobertura ao ambiente de desenvolvimento.<br>Configurar limites de aceitação das métricas de cobertura.</td></tr>
<tr><td colspan="2">Passos<br>▪ Analisar o ambiente de trabalho<br>▪ Analisar a integração com outras ferramentas<br>▪ Analisar a ferramenta<br>▪ Configurar métricas</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Ambiente de Desenvolvimento<br>▪ Guideline de Cobertura</td><td>Artefatos Resultantes:<br>▪ Guideline de Cobertura</td></tr>
<tr><td colspan="2">Responsável: Implementador</td></tr>
</table>

Dando continuidade, tem-se o detalhamento dos passos desta atividade. Alguns destes passos não serão necessários quando a ferramenta de cobertura já estiver em pleno funcionamento na organização.

**Analisar o ambiente de desenvolvimento**

Para a configuração da ferramenta de cobertura se faz necessário uma análise do ambiente de desenvolvimento que se deseja utilizar e o projeto (código e casos de teste) que será avaliado. Nem todos os projetos possuem a mesma característica específica (ambiente) de desenvolvimento, logo a análise é importante, a fim que se possa verificar se são compatíveis para integração com outras ferramentas.

**Analisar a integração com outras ferramentas**

Algumas ferramentas de cobertura têm funcionalidade de integração com ferramentas de desenvolvimento (IDE – *Integrated Development Environment*), a fim que o desenvolvedor não tenha trabalho no manuseio para utilizar as técnicas de coberturas.

Neste passo, o conhecimento de dois pontos principais é primordial: (1) o tipo de ferramenta e ambiente de desenvolvimento utilizado e (2) se a ferramenta de cobertura escolhida tem possibilidade de integração. O motivo para esta integração é facilitar ao máximo para o desenvolvedor o uso de cobertura de código, pois no ambiente no qual está desenvolvendo o código poderá utilizar a ferramenta de cobertura. No capítulo 5, referente ao estudo de caso, será detalhada a forma que foi realizada a seleção da ferramenta de cobertura e a sua integração com o ambiente de desenvolvimento.

**Analisar a ferramenta**

Realiza uma análise se a ferramenta está plenamente configurada, se os dados fornecidos para entrada estão adequados com o que à ferramenta e o projeto necessitam.

**Configurar métricas**

Seleciona métricas de cobertura, as quais se desejam analisar. Algumas ferramentas proporcionam para o usuário a funcionalidade de definir patamares referentes aos limites de aceitação das métricas de cobertura. Logo é possível especificar estes limites no momento da configuração. A vantagem é que se podem determinar limites de acordo com a característica específica de cada projeto.

Os *Guidelines* de cobertura servem de guia para a configuração da ferramenta de cobertura, principalmente no aspecto de sua integração com o ambiente de trabalho, podendo o documento ser atualizado se necessário.

### 3.4.2 Instrumentação do Código

A instrumentação consiste em inserir pontos de verificação, a fim de investigar se os locais (trechos) onde os pontos foram inseridos realmente estão sendo executados. Após ter instrumentado o código original, é importante realizar a compilação deste código, para que se possa ter uma *build* instrumentada.

Em alguns projetos o tempo de compilação é um fator crítico, pois o processo de compilação é longo, logo se deve ter o código fonte e os casos de testes estabilizados para que se possa utilizar o processo de análise de cobertura, para que não ocorra aumento do tempo e principalmente atividades sendo refeitas.

**Tabela 3-2:** Instrumentação do Código

<table>
<tr><td colspan="2">Propósito<br>Instrumentar o código que se deseja obter métricas de cobertura.</td></tr>
<tr><td colspan="2">Passos<br>▪ Selecionar os objetos para instrumentação<br>▪ Executar a instrumentação do código</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Código fonte</td><td>Artefatos Resultantes:<br>▪ Código Instrumentado</td></tr>
<tr><td colspan="2">Responsável: Implementador</td></tr>
</table>

A seguir a descrição dos passos desta atividade.

**Selecionar os itens para instrumentação**

Selecionar os itens (código e/ou componente), os quais se desejam obter métricas de cobertura dos testes realizados. Pode-se selecionar a partir do código ou até mesmo um projeto como um todo.

**Executar a instrumentação do código**

Tendo o item selecionado, basta apenas executar na ferramenta de cobertura a operação de instrumentação. Existem ferramentas de cobertura onde esta atividade é realizada de forma transparente para o usuário, gerando diretamente o relatório de cobertura no momento que a execução dos testes é finalizada. No estudo de caso proposto foi utilizada uma ferramenta que realiza esta atividade.

### 3.4.3 Geração do Relatório de Cobertura

É a atividade em que serão gerados os relatórios com a análise da cobertura do código. Dependendo da ferramenta, podem ser gerados vários tipos de relatórios, com formato textual e/ou gráfico.

**Tabela 3-3:** Geração do Relatório de Cobertura

<table>
<tr><td colspan="2">Propósito<br>Gerar os relatórios com as métricas de cobertura.</td></tr>
<tr><td colspan="2">Passos<br>▪ Selecionar o projeto onde foram executados os testes<br>▪ Gerar os relatórios com as métricas</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Arquivo de histórico da execução</td><td>Artefatos Resultantes:<br>▪ Relatórios de cobertura</td></tr>
<tr><td colspan="2">Responsável: Implementador</td></tr>
</table>

Tendo os casos de testes executados, a atividade agora será gerar as métricas. Para isso, em alguns casos (dependendo da ferramenta), se faz necessário ter o arquivo histórico da execução correspondente à execução. Os passos desta atividade são os seguintes:

**Selecionar o projeto onde foram executados os testes**

Seleciona o projeto que foi submetido aos testes, para poder gerar o relatório com as métricas definidas.

**Gerar os relatórios com as métricas**

Configurar o local do repositório no qual se deseja armazenar o relatório, bem como configurar a forma do relatório (ex.: HTML, PDF, XML). Estes passos, em alguns casos (dependendo da ferramenta), podem já ter sido realizados na atividade de configuração da ferramenta.

No momento que for selecionar e definir a ferramenta de cobertura que será utilizada no projeto da organização deve-se levar em consideração a forma do relatório de cobertura, informando os trechos que foram ou não executados pelo testes.

### 3.4.4 Análise dos Resultados de Cobertura

É a atividade em que serão analisados e verificados os resultados das métricas de cobertura. Esta atividade serve como suporte para tomada de decisões do projeto, pois é o gerente poderá tomar posicionamento a respeito do baixo nível de cobertura encontrado na análise de cobertura.

**Tabela 3-4:** Análise dos Resultados de Cobertura

<table>
<tr><td colspan="2">Propósito<br>Análise dos resultados de cobertura.</td></tr>
<tr><td colspan="2">Passos<br>▪ Verificar a consistência nos dados dos relatórios<br>▪ Analisar os pontos fortes e fracos<br>▪ Reunir os principais envolvidos no projeto<br>▪ Estabelecer metas, responsáveis e prazos para resolução dos problemas.</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Relatório de Cobertura</td><td>Artefatos Resultantes:<br>▪ Solicitação de Mudança</td></tr>
<tr><td colspan="2">Responsável: Implementador</td></tr>
</table>

**Verificar a consistência nos dados dos relatórios**

Analisar se os relatórios gerados estão de acordo com o que foi configurado na ferramenta de cobertura e se não houve problemas na sua geração, resultando em informações perdidas. Se os dados condizem realmente com o projeto que está sendo avaliado, ou seja, se as métricas de cobertura selecionadas são suficientes para analisar um projeto complexo.

**Analisar os pontos fortes e fracos**

Tendo o relatório de cobertura, deve-se analisar e fazer as seguintes considerações:

- Os pontos fortes: as métricas que foram estabelecidas para o projeto, foram alcançadas com eficiência.

- Os pontos negativos: as métricas não foram alcançadas, resultando em trecho no código que não foram testadas, correndo o risco de ter erros que não serão encontrados nos testes.

**Reunir os principais envolvidos no projeto**

Convocar as principais pessoas envolvidas no projeto, a fim que se possa obter e levantar os motivos que resultaram no nível de cobertura baixo.

**Estabelecer metas, responsáveis e prazos para resolução dos problemas.**

Tendo encontrado os problemas, é determinada a pessoa (ou equipe) que será responsável pela resolução dos mesmos. Serão definidos metas e prazos para a solução do problema. È aberta e/ou atualizada a Solicitação de Mudança para correção do problema.

## 3.5 Artefatos de cobertura de código

No momento em que as atividades de cobertura de código são realizadas, estas criam, utilizam ou alteram artefatos. Os novos artefatos são ilustrados na Figura 3-3 e descritos a seguir:

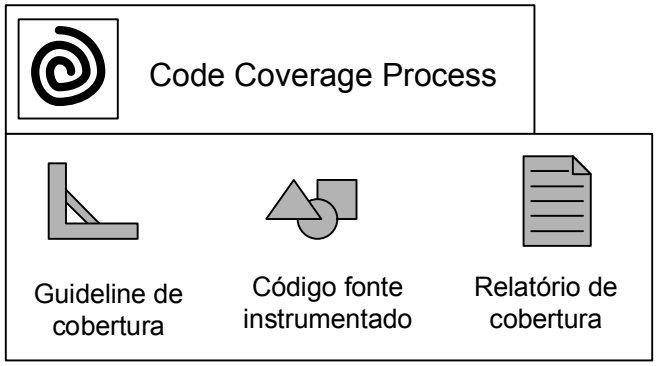



**Figura 3-3:** Artefatos gerados no *Code Coverage Process*

### 3.5.1 *Guideline* de Cobertura

No processo de desenvolvimento com cobertura de código, existe a necessidade da criação de um *guideline* que orientará o uso da ferramenta no processo. Neste guia, é importante ter os passos necessários para integração da ferramenta ao ambiente de desenvolvimento. Este artefato se assemelha a um manual da ferramenta, no entanto direcionado para o ambiente de desenvolvimento da organização e do projeto que está sendo avaliado, ao contrário do manual que acompanha a ferramenta, pois o mesmo é em muito genérico, ocasionando muitas vezes um empecilho para a sua utilização.

### 3.5.2 Código Fonte Instrumentado

É o código alterado com a inserção dos pontos de verificação, para que se possa ser realizado a análise da cobertura do código. Existem ferramentas que tornam esta atividade totalmente transparente para o usuário, fazendo com o que o mesmo não se preocupe com o código instrumentado.

Dependendo do tempo da geração da *build*, pode-se gerar uma *build* sem instrumentação, a fim de verificar se não há nenhum problema com o código original e/ou no ambiente de desenvolvimento, para então gerar a *build* instrumentada, que no caso é gerada através do código fonte instrumentado.

É preciso ter *builds* paralelas de desenvolvimento, uma oficial proveniente do código original e uma outra para o código instrumentado. A *build* do código original se tornará *build* do produto, a qual se submeterá ao controle de versão, a outra servirá apenas para realizar os casos de teste e realizar a análise de cobertura.

### 3.5.3 Relatório de Cobertura

Apresenta as métricas de cobertura do código que foi executado pelo testes de unidade. Relata a efetividade dos casos de testes de unidade. Faz-se necessário um banco de dados (repositório) com todos os relatórios de cobertura, para que se possa ter um acompanhamento da evolução dos casos de testes de unidade e do código. A partir de relatórios de cobertura, pode-se fazer uma avaliação sobre o nível de cobertura desejável para cada projeto. O relatório de cobertura se juntará a Solicitação de Mudança, a fim que se tenha um controle e monitoramento da qualidade do código e dos testes desenvolvidos.

A utilização das técnicas de cobertura de código no fluxo de testes, executada através dos testes caixa preta, ocasionará um relatório não tão condizente com a realidade, pois a geração das métricas de cobertura é realizada com base dos testes executados no código e alguns destes testes não conseguiram passar por todos os trechos no qual apenas o teste de unidade poderá verificar. Logo deverá ter uma atenção na definição de limiares e principalmente na análise dos resultados, pois a granularidade no relatório é alta.

## 3.6 Responsáveis (Papéis)

Os responsáveis que terão participação nas atividades e nos artefatos gerados pela análise de cobertura de código estão apresentados na Figura 3-4.

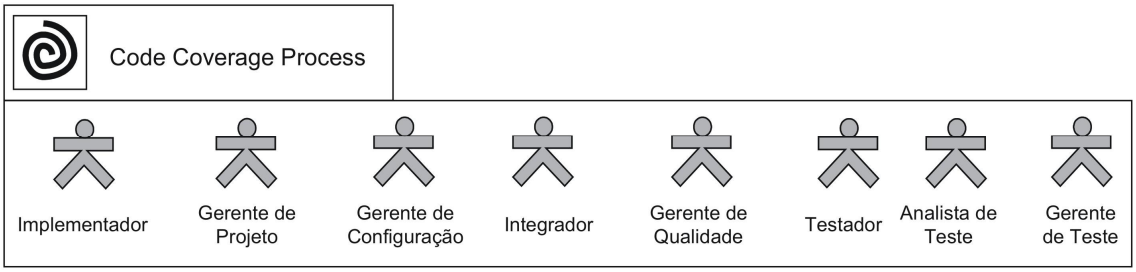



**Figura 3-4:** Responsáveis envolvidos no *Code Coverage Process*

Divide-se a interação dos envolvidos em duas partes: (1) Fluxo de Implementação e (2) Fluxo de Teste. Os fluxos de Gerenciamento de Configuração e Mudança e o de Gerenciamento de Projeto entrarão como suporte e controle.

### 3.6.1 Interação no Fluxo de Implementação

No RUP, a responsabilidade por executar os testes de unidade é do implementador e não de um testador, sendo que em algumas organizações, a tarefa de executar esses testes de unidade pode ser feita por outra pessoa, fazendo o papel de testador. No processo proposto, o implementador é considerado como sendo o responsável pela atividade.

O **implementador** terá grande envolvimento na análise de cobertura de código, pois ele será o ponto chave de todo o processo. Ele irá configurar a ferramenta no ambiente de desenvolvimento, à medida que for finalizando o conjunto de testes de unidade e submetendo o código para geração da *build*, terá a responsabilidade de instrumentar o código e coletar as informações da cobertura. Terá também a obrigação de analisar os relatórios de cobertura gerados pela ferramenta, para que possa verificar se há necessidade

de alteração e/ou criação de algum caso de teste de unidade ou alteração do código fonte. As suas atividades serão analisadas pelos resultados coletados pela análise de cobertura de código, logo a definição de métricas e limiares terá grande impacto no seu trabalho.

O **gerente de projeto** tem a responsabilidade de acompanhar, controlar e definir na medida do possível, as métricas de cobertura de código, a serem coletadas para que possa fazer o acompanhamento da evolução da qualidade dos testes de unidade desenvolvidos, bem como a qualidade do produto gerado.

O **gerente de configuração** terá que prover os recursos para utilização do processo de cobertura de código, como o local e a forma onde serão armazenados os artefatos gerados no processo. Ajudará na criação do *guideline* de cobertura, referente à utilização do ambiente de trabalho.

O **integrador** é responsável por planejar a integração, que ocorre no subsistema e no sistema. Irá criar o espaço de integração para receber os componentes testados e criará a *build* com a integração realizada, a qual será testada.

O **gerente de qualidade** acompanhará e monitorará a evolução do projeto e do processo, a partir dos resultados de cobertura. Podendo, caso seja necessário, alterar o processo. O mesmo fará também a análise dos níveis de aceitação obtidos, a fim de identificar falhas no processo e/ou na maneira que está sendo desenvolvido o código e os testes.

A Figura 3-5 mostra o implementador como sendo o responsável principal e tendo a incumbência de informar ao gerente de projeto o andamento da cobertura do código e dos testes de unidade. O implementador terá o suporte do gerente de configuração e do integrador para a execução do processo de cobertura de código. O gerente de qualidade acompanha e monitora a execução do processo, interagindo diretamente com o implementador e o gerente de projeto.

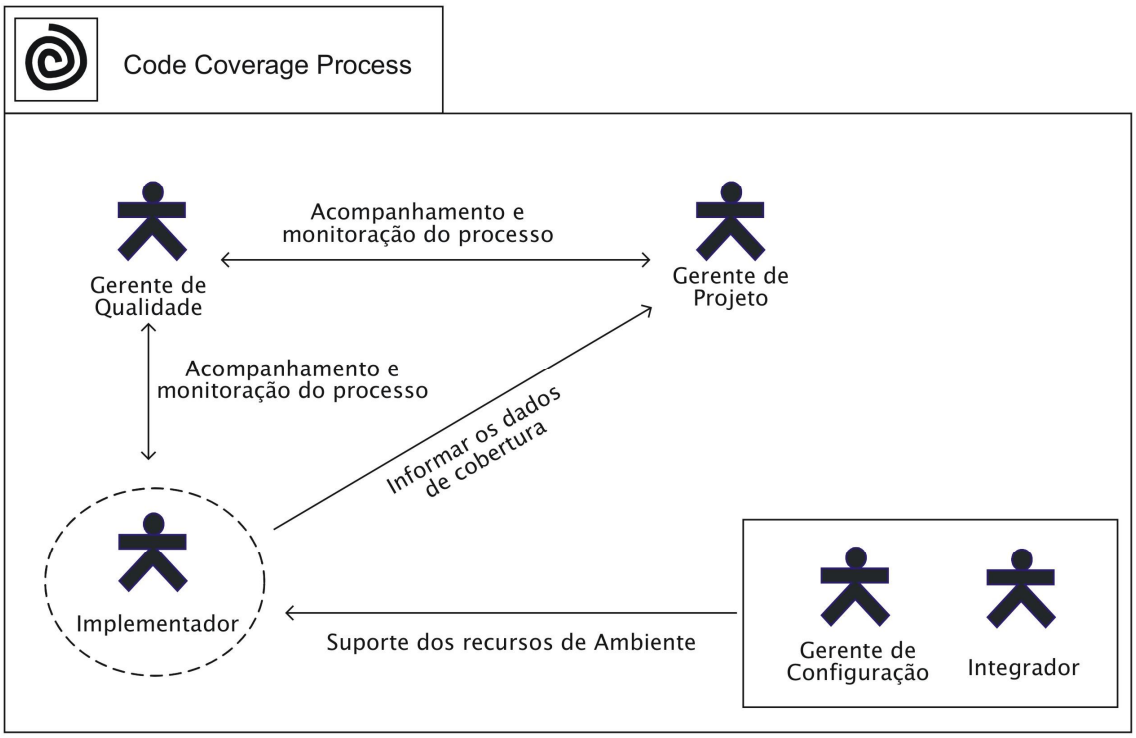



**Figura 3-5:** Interação entre os responsáveis pelo *Code Coverage Process* relacionado ao processo de implementação

### 3.6.2 Interação no Fluxo de Teste

Para utilização do *Code Coverage Process* no fluxo de Teste, será incluída a participação do testador, analista de teste e do gerente de teste.

O **testador** é responsável pelas atividades centrais do esforço de teste, que envolve conduzir os testes necessários e registrar os resultados desses testes. O testador terá a incumbência de testar a *build*, a qual foi instrumentada, a fim de que possa executar os casos de testes. Finalizando a execução, é realizada a geração dos relatórios, que dependendo da ferramenta de cobertura utilizada, pode ser feito pelo testador ou por algum membro da equipe de desenvolvimento, no qual irá receber o arquivo de histórico da execução dos casos de teste para gerará os relatórios de cobertura.

O **analista de teste** é responsável por identificar e definir os testes necessários, monitorar a abrangência dos testes e avaliar a qualidade geral obtida ao testar. Este papel também envolve a especificação dos dados de teste necessários e a avaliação do resultado dos testes conduzidos em cada ciclo de teste. Como o mesmo é responsável pela definição dos testes e pela análise do resultado, deve levar em consideração que nem todos os casos

de teste alcançarão 100% de cobertura, devido à granularidade de a cobertura ser alta, pois geralmente trabalha com testes de unidade e de integração, no qual condições e/ou trechos do código só são testados com esse tipo de testes.

O **gerente de testes** tem a responsabilidade geral pelo êxito do esforço de teste. O papel envolve garantia da qualidade do produto e dos testes, planejamento e gerenciamento de recursos e resolução de problemas que representam um obstáculo para o esforço de teste. O gerente analisará os resultados da cobertura e deve ter uma boa comunicação com os principais envolvidos com o desenvolvimento da *build* a ser testada.

O **implementador** terá a função de encaminhar a *build*, instrumentada, para o testador, a fim que o mesmo possa executar os testes. O implementador dará suporte ao testador para possíveis problemas relacionados com a *build*. Dependendo da ferramenta de cobertura utilizada, o implementador terá a responsabilidade de gerar os relatórios de cobertura através do arquivo de histórico da execução.

O **gerente de projeto** acompanhará os resultados dos testes, dando suporte ao gerente de testes e fazendo análise e monitoramento da evolução do projeto. O gerente de qualidade acompanhará e monitorará a evolução do projeto e do processo, analisando os níveis alcançados referente à cobertura do código.

O **gerente de configuração** e o integrador terão as mesmas atribuições do fluxo de Implementação, em relação à cobertura de código.

Para um melhor entendimento sobre a interação dos responsáveis no fluxo de Testes, é possível observar na Figura 3-6 que o responsável principal pelo processo é o testador, ficando o implementador e o gerente de configuração como suporte.

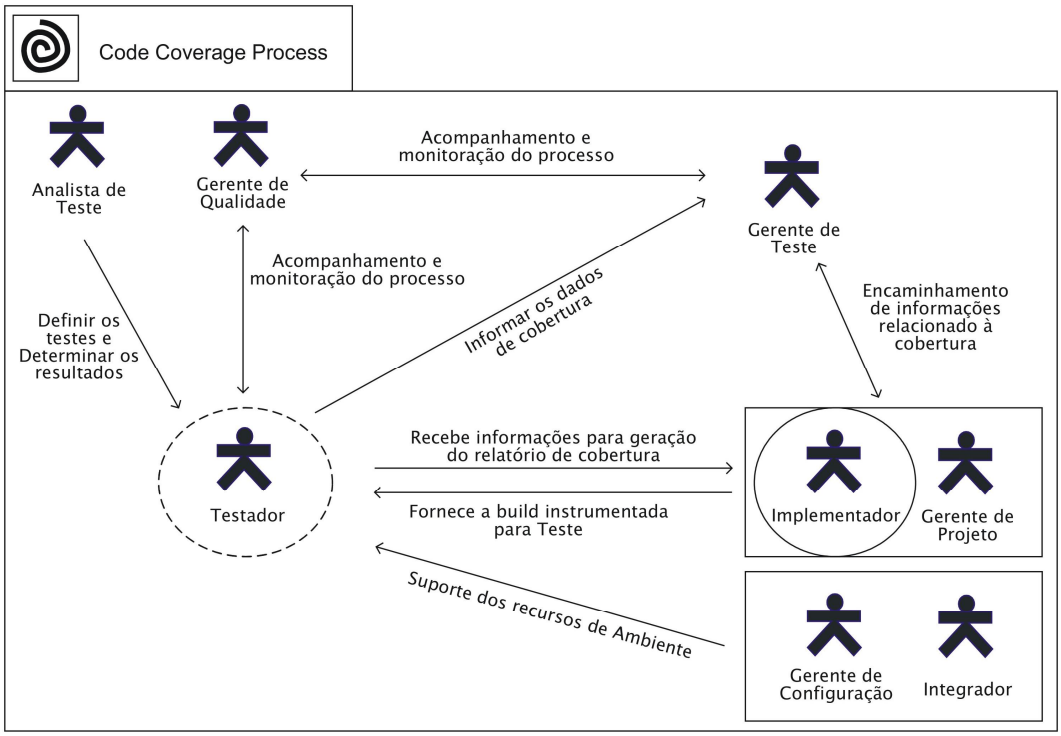



**Figura 3-6:** Interação entre os responsáveis pelo *Code Coverage Process* relacionado ao fluxo de Testes

## 3.7 Fluxo de Implementação

No desenvolvimento do software a inserção da análise de cobertura de código ocorre quando o código e os testes de unidade estão estabilizados. Para o início das atividades, referente à cobertura de código, recomenda-se que o ambiente de desenvolvimento já esteja configurado e funcionando para que não haja problemas no momento da execução das tarefas de cobertura de código. Os subfluxos que serão adaptados para o processo proposto estão apresentados na Figura 3-7 e descritos a seguir.

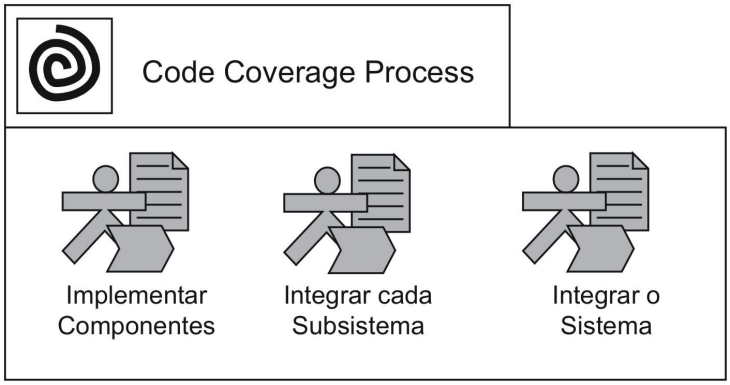



**Figura 3-7:** Subfluxos adaptados do fluxo de Implementação

**Implementar Componentes**

De forma resumida o fluxo de cobertura de código que é executado pelo implementador no fluxo de implementação, em que o mesmo irá selecionar o projeto (código e casos de teste) que se deseja verificar e analisar a sua cobertura. Na Figura 3-8 é possível ter uma visão mais detalhada do subfluxo implementar componentes.

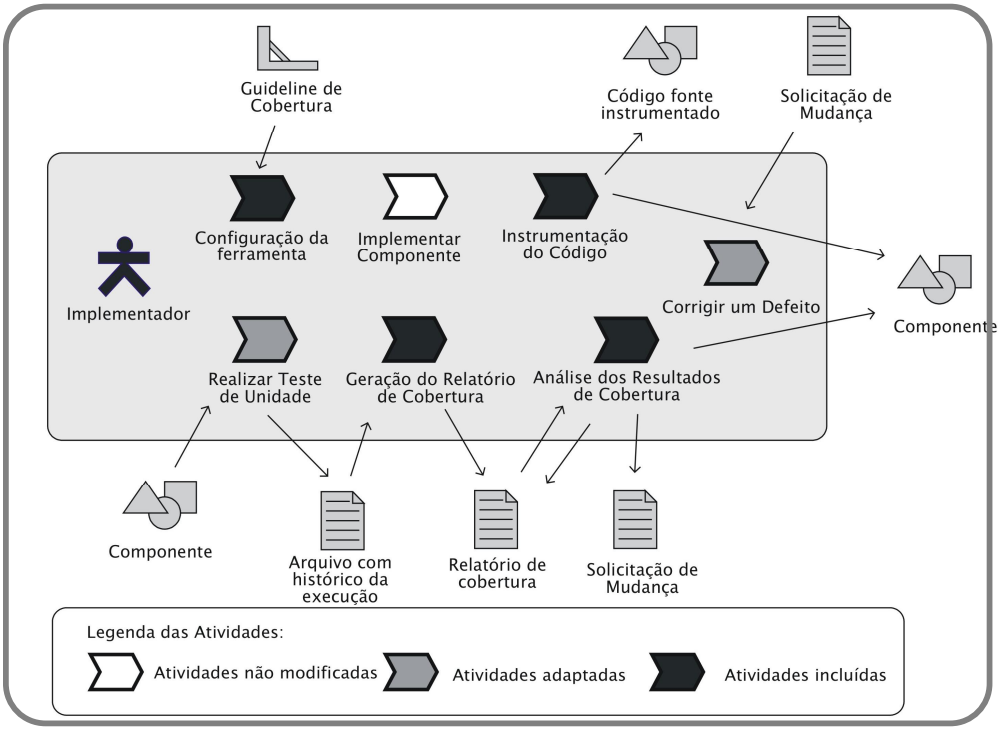



**Figura 3-8:** Subfluxo detalhado de Implementar Componentes

No *Code Coverage Process*, deve-se utilizar a ferramenta de cobertura no momento em que se tenha o código fonte e os testes de unidade estáveis, pois não se deve utilizar a instrumentação e ao mesmo tempo ficar alterando o código e/ou os testes de unidade, para que não se tenha atividade de instrumentação e de compilação sendo refeita e principalmente gerando métricas de cobertura que não condizem com a realidade, o que colocaria em dúvida da eficácia do processo.

O subfluxo Implementar Componente irá desenvolver e/ou alterar o código, sendo que é necessário configurar a ferramenta de cobertura para ter acesso ao código que esta sendo desenvolvido e para integração dela ao ambiente de trabalho.

As atividades de implementação e instrumentação, juntamente com a tarefa de compilação do código instrumentado, serão realizadas de forma seqüencial podendo ser executada paralelamente com a elaboração dos testes de unidade ou até por implementadores diferentes, sendo que ambas terão que configurar a ferramenta para o acesso ao ambiente de desenvolvimento e também para ajustar os limites de cobertura que foram especificados no projeto.

Tendo atingido sucesso em relação às métricas de cobertura especificadas no projeto com sucesso. É finalizada a tarefa, atualizando a solicitação de mudança e anexando o relatório de cobertura, a fim de se ter o monitoramento e acompanhamento do projeto. À medida que for sendo seguido o processo, pode-se rever as métricas estabelecidas, a fim de aumentar ou diminuir o limiar de aceitação das métricas de cobertura de código.

Tendo sido gerado o relatório de cobertura, começa a atividade de analisar os resultados. Dependendo do resultado do relatório com as métricas de cobertura, se faz necessário a correção (alteração) do código e/ou dos seus testes de unidade, passando em seguida para a atividade de correção de defeito, tendo sido registrado no documento de solicitação de mudança, anexando o relatório de cobertura. Ao concluir a correção, deverá ser realizada novamente a atividade de instrumentação, caso a correção tenha sido feita no código, a fim que se possam gerar novos relatórios para uma reavaliação da cobertura. Se a correção for somente aos testes, não há necessidade de instrumentar o código, precisando apenas executar os testes novamente para gerar o relatório.

Tendo atingido as métricas de cobertura especificadas no projeto, é finalizada a tarefa, atualizando a solicitação de mudança e anexando o relatório de cobertura, a fim de se ter o monitoramento e acompanhamento do projeto. À medida que for sendo seguido o processo, as métricas estabelecidas podem ser revistas, a fim de aumentar ou diminuir o patamar de aceitação.

**Integrar cada Subsistema**

O subfluxo “Integrar cada Subsistema” será realizado com uma *build* instrumentada, a fim que se possam realizar os casos de testes e em seguida gerar a métricas de cobertura, para se obter dados com o subsistema integrado e detectar se há necessidade de criar novos testes ou eliminar testes que estão redundantes.

**Integrar o Sistema**

O subfluxo "Integrar o Sistema" utilizará o código instrumentado, para o mesmo propósito da atividade anterior, ou seja, ter métricas de cobertura de código com o sistema integrado, no qual haverá a possibilidade de executar os casos de testes com base na cobertura do código.

## 3.8 Fluxo de Testes

Como já foi dito no capítulo anterior, a utilização da técnica de cobertura de código surgiu de um interesse da organização na qual foi realizado o estudo de caso. A organização propôs um desafio, teríamos que gerar *builds* instrumentadas e encaminhá-las para a equipe de testes, com intuito de executar casos de testes de caixa preta. Um outro ponto que motivou a realizar esse tipo de trabalho foi que era possível selecionar casos de testes de suítes de teste, baseado na cobertura do código. No capítulo cinco (5), referente aos estudos de caso, será apresentado em detalhes como se desenvolveu a inserção da cobertura de código dentro do fluxo de teste.

É de fundamental importância medir a qualidade dos casos de testes executados no fluxo de Teste. Claramente, o percentual de redundância de um conjunto de casos de testes, bem como a quantidade de código que é coberta pelos testes, são medidas importantes para a avaliação da qualidade da suíte utilizada. Com essa informação em mãos, é possível guiar o processo de geração de casos de testes, de forma a otimizar a cobertura dos mesmos.

O maior intuito da utilização do processo no fluxo de teste é a possibilidade de eliminar teste que por ventura sejam redundantes, e continuar com uma cobertura de código satisfatória ao projeto.

No fluxo de Teste, todos os subfluxos foram impactados com a utilização do processo conforme a mostra a Figura 3-9.

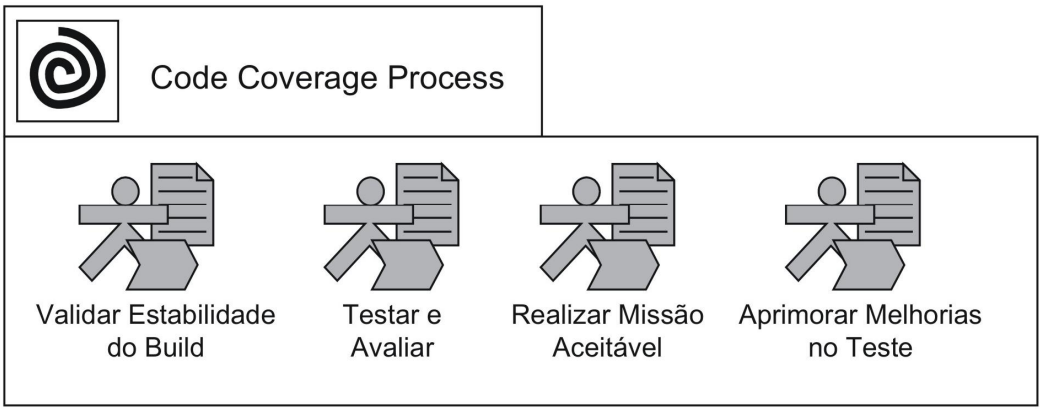



**Figura 3-9:** Subfluxos adaptados do fluxo de Testes

**Validar Estabilidade da Build e Testar e Avaliar**

Nestes dois subfluxos foi utilizada uma *build* instrumentada, a fim de realizar testes, para que com os resultados da cobertura do código, possa selecionar determinados testes de um conjunto mediante um nível de cobertura determinado.

**Realizar Missão Aceitável**

Para o *Code Coverage Process* a adaptação deste subfluxo se deve com o planejamento e a determinação do limiar de aceitação, tendo conhecimento que não é possível que alguns testes consigam ter a mesma eficácia dos testes de unidade, devido que algumas condições só são possíveis com esse tipo de teste. Neste subfluxo, se dará também uma reavaliação do limiar, podendo conter apoio de outras pessoas envolvidas no projeto e no processo.

**Aprimorar Melhorias no Teste**

Neste subfluxo, são reavaliados os casos de teste, com o intuito de selecionar ou criar testes que alcance uma maior cobertura de código.

## 3.9 Fluxo de Gerenciamento de Configuração e Mudança

A realização do processo, com a utilização da técnica de cobertura de código, terá o suporte do fluxo de Gerenciamento de Configuração e Mudança, pois novos artefatos serão criados e alguns serão adaptados, logo se faz necessário ter um controle de versão destes documentos. Outro ponto também que terá suporte deste fluxo é a questão do ambiente de trabalho, no qual o implementador necessitará configurar o ambiente para a utilização da

ferramenta de cobertura. Os subfluxo que serão adaptados estão apresentados na Figura 3-10 e descritos a seguir.

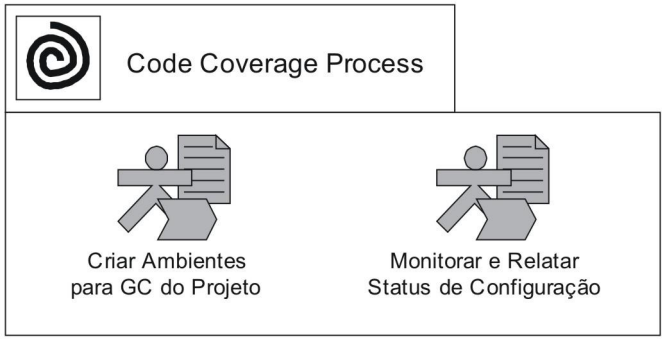



**Figura 3-10:** Subfluxos adaptados do fluxo de Gerenciamento de Configuração e Mudança

O subfluxo "Criar Ambientes para Gerência de Configuração do Projeto", armazenará o *Guideline* de Cobertura e o Relatório de Cobertura, pois os dois sofrerão mudança no decorrer do projeto.

O subfluxo "Monitorar e Relatar Status de Configuração" foi adaptado para o processo proposto, no que se refere ao documento de Métricas de Projeto, no qual foi incluída a métrica de cobertura de código.

Em todo o processo proposto, o fluxo de Gerenciamento de Configuração e Mudança dará suporte no que se refere ao ambiente de desenvolvimento e no controle dos artefatos, bem como o seu acesso.

## 3.10 Fluxo de Gerenciamento de Projeto

Neste fluxo será feita uma análise das métricas obtidas através dos relatórios de cobertura, os quais servirão de entrada para tomada de decisões referentes à qualidade do código e dos casos de testes. Os subfluxos adaptados estão apresentados na Figura 3-11 e descritos em seguida.

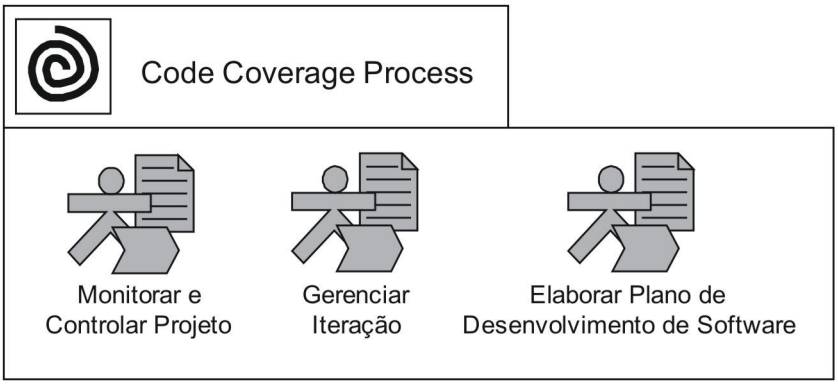



**Figura 3-11:** Subfluxos adaptados do fluxo de Gerenciamento de Projeto

**Monitorar e Controlar o Projeto**

No subfluxo "Monitorar e Controlar o Projeto", que guia o trabalho do gerente de projeto, as atividades Monitorar Status do Projeto, Relatar Status e Resolver Exceções e Problemas foram adaptadas para o *Code Coverage Process*.

Na atividade Monitorar Status do Projeto, o gerente irá, através das informações do Relatório de Cobertura, avaliar a qualidade do código e dos testes desenvolvidos.

Na atividade Relatar Status será informado, para as pessoas competentes do projeto, os dados provenientes das métricas de cobertura atingidas no projeto, com intuito de comprovar a qualidade dos testes e conseqüentemente do produto como um todo.

Na atividade Resolver Exceções e Problemas será tomado as ações e medidas corretivas relacionadas ao não cumprimento das métricas estabelecidas no projeto. O gerente de projeto irá analisar e tomará decisões cabíveis para a correção destes problemas. O gerente de qualidade deverá acompanhar e monitorar a evolução do projeto.

**Gerenciar Iterações**

No subfluxo "Gerenciar Iterações", a atividade Avaliar Iteração foi adaptada para o *Code Coverage Process*, pois o Relatório de Cobertura dará informações a respeito da qualidade dos testes desenvolvidos. Mediante essas informações se tomará ações e medidas apropriadas para as próximas iterações.

No momento de selecionar e definir a ferramenta de cobertura que será utilizada no projeto da organização, deve-se levar em consideração a forma que o relatório se apresenta, informando os trechos que foram ou não executados pelo testes.

**Elaborar Plano de Desenvolvimento de Software**

Um dos subfluxos adaptados foi "Elaborar Plano de Desenvolvimento de Software", no qual as atividades adaptadas para o *Code Coverage Process* foram: Desenvolver Plano de Métricas; Definir Processos de Controle e Monitoramento e Desenvolver Plano de Garantia de Qualidade.

A atividade Desenvolver Plano de Métricas define as metas do projeto que deverão ser seguidas e alcançadas com êxito. A atividade Definir Processos de Controle e Monitoramento define as informações e os processos que serão usados para o monitoramento e controle do andamento da qualidade e dos riscos do projeto.

Podemos pensar que a realização de medições é algo evidente, visto que a mesma irá nos orientar a quantificar e administrar o que está sendo medido. Entretanto, devemos ter atenção com a forma que os dados são coletados e principalmente interpretados, pois, sendo realizado de forma errada trará danos à organização, ao invés de benefícios, pois decisões serão tomadas e planejamentos serão realizados a fim de se ter um aprimoramento nos dados que foram medidos.

Segundo [Kan, 2003], é indiscutível que medições sejam cruciais ao desenvolvimento de qualquer ciência. A engenharia de software começa a evoluir para um caminho em que medições fazem parte dos processos básicos para se construir software.

Para as organizações é muito importante e crucial conseguirem manter um ritmo de produção mediante as constantes mudanças de tecnologia da informação, tendo mercado totalmente competitivo. As medições, neste caso, surgem como suporte para validar o nível de qualidade dos projetos, ou seja, ajudam ao gerente fornecendo informações para à tomada de decisões do projeto. Segundo [Mcgarry, 2002], medições de software fornecem informações objetivas para suportar os gerentes de projeto principalmente nos seguintes aspectos:

- Comunicação efetiva através das informações objetivas;

- Acompanhamento dos objetivos dos projetos através da medição dos processos e produtos;
- Identificação e correção dos problemas de forma antecipada, uma vez que medições viabilizam uma estratégia de gerência pró-ativa;
- Suporte na tomada de decisões críticas;
- Justificativas para a tomada de decisões.

A definição de métricas e limiares de cobertura de código é um passo muito relevante na utilização do *Code Coverage Process*. As métricas são importantes no controle e gerenciamento de projetos de software. A escolha das métricas está intimamente associada às estratégias e objetivos da organização, e vai depender do estágio de maturidade em que a mesma se encontra.

A seleção e a definição de um conjunto de métricas uma não são tarefa trivial. Na verdade, é uma tarefa custosa que demanda grande conhecimento para evitar que o seu uso não aumente ainda mais os problemas enfrentados durante o desenvolvimento de software. Escolher incorretamente uma métrica e seus limiares de aceitação pode gerar um aumento desnecessário de esforço e uma visão errada da realidade do projeto, resultando em tomada de decisões equivocadas. O correto é a realização de um ou mais projetos piloto, com o intuito de determinar métricas e principalmente atestar a ferramenta selecionada.

As etapas de definição e seleção de métricas de cobertura de código são realizadas através de cinco etapas como mostra a Figura 3-12.

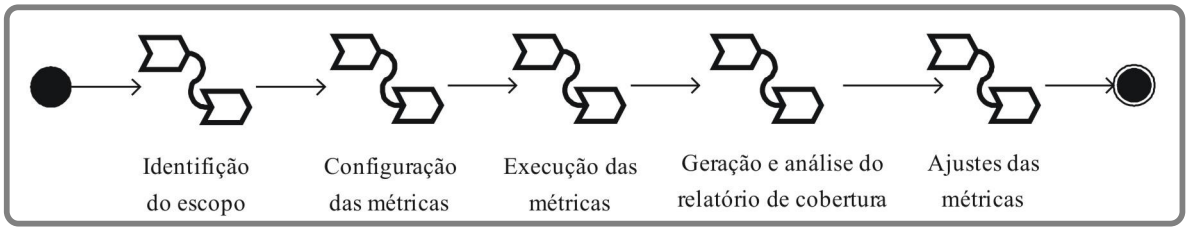



**Figura 3-12:** Etapas da definição e seleção de métricas de cobertura de código

**1) Identificação do escopo**

Esta etapa consiste em estabelecer o escopo do que será medido:

- Identificar as necessidades para medição;

- Identificar os objetivos da organização que serão atendidos, verificando se as necessidades de medição levantadas estão alinhadas aos objetivos da organização;
- Identificar os objetivos da medição, de acordo com o projeto que está sendo avaliado.

Como existem várias métricas de cobertura de código, a seleção se dá a partir daquelas que atendam os objetivos da organização que foram identificados para o projeto que será analisado.

**2) Configuração das métricas**

Uma vez identificado o conjunto de métricas, a partir da definição do escopo realizado na etapa anterior, serão configuradas na ferramenta as métricas selecionadas e seus critérios de aceitação:

- Configurar a ferramenta de cobertura, com as métricas escolhidas;
- Determinar os critérios de aceitação de cada métrica e o nível desejável de cobertura.

**3) Execução das métricas**

Com as métricas selecionadas, baseado no nível de complexidade do projeto, e a ferramenta configurada, será executado o código e/ou componente do projeto aplicando os casos de testes definidos no projeto.

**4) Geração e Análise do Relatório de Cobertura**

Tendo finalizado a execução dos casos de testes, é realizada a geração do relatório, o qual irá mostrar os trechos que estão necessitando ser cobertos. Gerado os relatórios de cobertura, é possível realizar a análise da cobertura do código, mediante os testes que foram executados, observando os níveis alcançados da cobertura de cada código em relação às métricas pré-estabelecidas.

**5) Ajustes das métricas**

Nesta análise é verificado se as métricas estabelecidas que não foram alcançadas podem realmente ser atingidas, pois dependendo do nível de complexidade do projeto

algumas métricas podem ter o limite de aceitação reduzido. Sendo encontrado esta situação são revistas as métricas de cobertura.

Na atividade de Desenvolver Plano de Garantia de Qualidade, o plano contará com as informações para o comprometimento das métricas e do processo de cobertura de código. Como foi dito anteriormente o gerente de qualidade terá uma fundamental participação, pois acompanhará e monitorar a evolução do projeto e do processo, levando em consideração os dados obtidos com as métricas de cobertura de código atingidas.

## 3.11 Considerações Finais

Neste capítulo foram apresentadas em detalhes as atividades e os artefatos do *Code Coverage Process*, cujo propósito é atender a realização da análise de cobertura de código inserido no processo de desenvolvimento de software, tendo como objetivo definir as atividades, os responsáveis e os artefatos gerados na implementação do código com análise de cobertura de código.

O RUP é um processo de software genérico para atender o desenvolvimento de vários tipos de aplicação. Entretanto, esta generecidade faz com que determinadas características para atender a uma necessidade específica, como análises de cobertura de código, não sejam devidamente contempladas. Diante deste panorama, o próximo capítulo apresenta a extensão de quatro fluxos do *Rational Unified Process* para incorporar o *Code Coverage Process*. A intenção é que qualquer organização de software, mesmo que não utilize o RUP como metodologia de desenvolvimento padrão, possa realizar as atividades referentes à análise de cobertura de código e gerar os artefatos necessários.

# 4 Extensão do RUP para incorporar o *Code Coverage Process*

O RUP é uma metodologia genérica que pode ser utilizada para desenvolvimento de sistemas em diversas áreas de aplicação. "O *Rational Unified Process* captura muitas das melhores práticas em desenvolvimento de software e as apresenta em uma forma apropriada que é satisfatória para uma gama extensa de projetos e organizações" [Kruchten, 2003].

O *Code Coverage Process* foi inspirado no *Rational Unified Process* e usa a modelagem SPEM (*Software Process Engineering Metamodel*) para representar seus artefatos e atividades. Trata-se de um processo de desenvolvimento de software com fases, disciplinas e artefatos, com o objetivo de analisar a cobertura de código, resultando numa melhoria na qualidade do software, reduzindo tempo para elaboração de casos de testes de unidade, redução de casos de testes redundantes e criação de novos casos de testes para aumentar a cobertura do código. A análise de cobertura de código tem o objetivo de ajudar a monitorar o código e torná-lo eficiente.

Desenvolvedores e testadores usam análise de cobertura de código para assegurar que todo programa, ou mais especificamente todas as linhas de um programa, foi executado pelo menos uma vez durante o processo de teste. Medidas de cobertura de código são importantes para testar e validar o código durante desenvolvimento de novas plataformas [Tikir, 2002].

Processo de software é um conjunto de etapas, formado por atividades, procedimentos e práticas utilizadas para atingir um determinado objetivo. Geralmente este objetivo está associado a um ou mais resultados e/ou produtos derivados da execução do processo.

## 4.1 Visão Geral

O propósito deste trabalho é estender alguns fluxos do RUP, aqueles que irão ser envolvidos com as novas atividades e artefatos advindos da realização do processo proposto.

## 4.2 Fluxos do RUP estendido para o *Code Coverage Process*

Na elaboração do processo proposto, foram identificados quais seriam os fluxos do RUP, nos quais teriam que inserir novas atividades ou adaptar as atividades existentes para utilização do *Code Coverage Process*. Assim, os Fluxos que serão relacionados às atividades de cobertura de código são apresentados na Figura 4-1 e descritos a seguir.

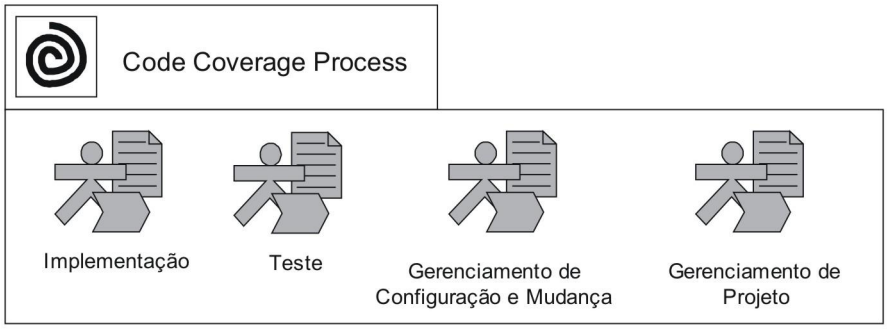



**Figura 4-1:** Fluxos estendidos para o *Code Coverage Process*

O fluxo de Implementação foi estendido porque é onde ocorre a codificação do software, bem como a geração e execução dos testes de unidade. Devido a isto uma análise de cobertura é necessária, a fim de melhorar a qualidade dos testes e verificar a cobertura do código.

O fluxo de Teste foi estendido, pois podemos utilizar a análise de cobertura de código em casos de testes de caixa preta, tendo uma ferramenta de cobertura que gere *build* instrumentada, pois com a execução dos casos de testes na *build*, se obterá as métricas de cobertura, podendo analisar os casos de testes e elimina-los caso sejam redundantes e/ou seleciona-los, baseando-se na cobertura do código.

O fluxo de Gerenciamento de Configuração e Mudança foi estendido, pois artefatos foram incorporados e alguns artefatos, já existentes, foram alterados para a utilização do processo.

O fluxo de Gerenciamento de Projeto, com as métricas obtidas pela análise de cobertura de código, algumas atividades serão do gerente, pois o mesmo utilizará as métricas para tomadas de decisões relativas ao projeto e a qualidade dos testes executados.

### 4.2.1 Fluxo de Implementação Estendido

É na Implementação que se realiza a codificação do produto, e principalmente a elaboração dos testes de unidade que irão exercitar o código que se deseja analisar. Este fluxo é o mais importante no *Code Coverage Process*, é nele que as principais atividades do processo estão inseridas; o fluxo de Teste terá a incumbência de validar *builds* e os demais fluxos envolvidos servirão para análise de resultados e suporte à execução do processo. A Figura 4-2 mostra o fluxo de Implementação, destacando os subfluxos que foram adaptados com a utilização do *Code Coverage Process*.

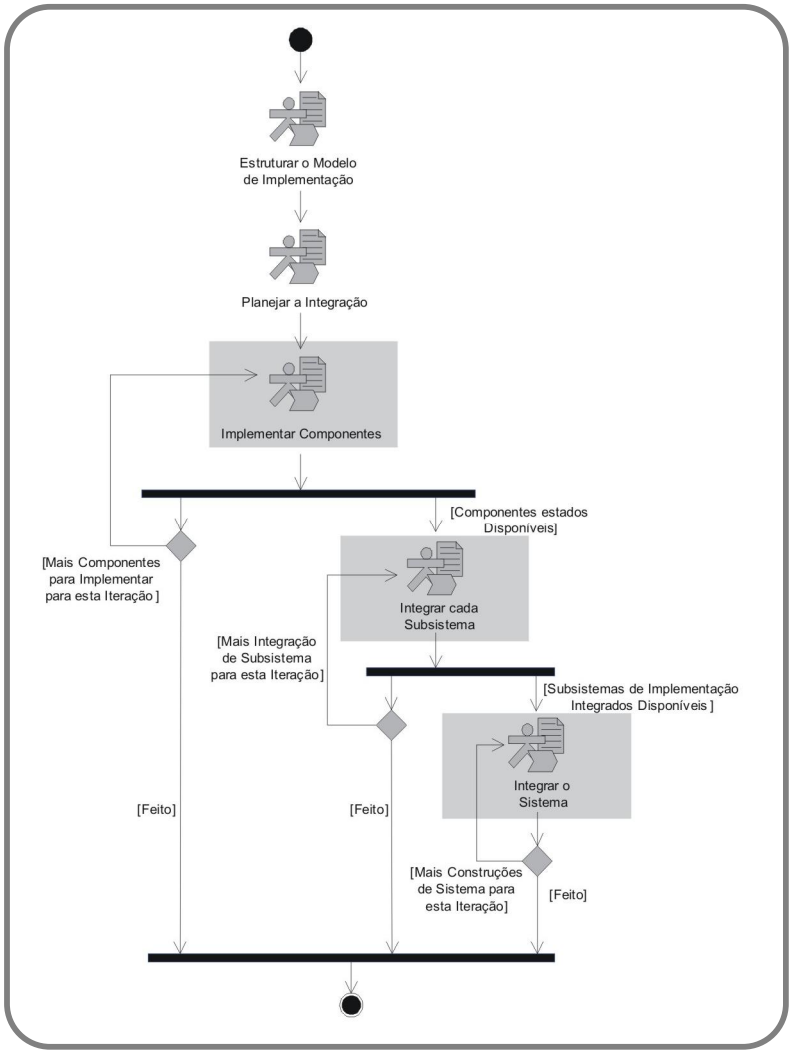



**Figura 4-2:** Fluxo de Implementação do RUP estendido

A seguir as descrições de cada subfluxo do fluxo de Implementação, são apresentadas dando destaque para o subfluxo "Implementar Componentes", "Integrar cada subsistema" e "Integrar o Sistema" os quais foram adaptados para a utilização da proposta.

A descrição das atividades de cada subfluxo que não foi adaptada para a utilização do *Code Coverage Process*, bem como os detalhamentos dos artefatos que não foram alterados pode ser encontrada na versão 2003 do *Rational Unified Process*.

**1) Estruturar o Modelo de Implementação**

A estruturação do modelo de implementação é realizada nas primeiras iterações na fase de iniciação. O objetivo é assegurar que o modelo de implementação seja organizado, definindo de que forma os subsistemas e pacotes serão organizados, levando-se em consideração suas dependências e outras características, como o uso de componentes e a atribuição da responsabilidade por componentes aos membros da equipe. Um bom modelo de organização prevê problemas de gerencia de configuração e permite que o produto seja gerado a partir de *builds* de integração sucessivamente maiores. Este subfluxo não foi modificado para o processo proposto.

**2) Planejar a Integração**

No planejamento da integração do sistema, o objetivo é determinar como será realizada a integração do sistema a partir dos seus subsistemas. O RUP adota uma estratégica de integração incremental, no qual o código é escrito e testado em pequenas partes que em seguida são combinadas para formar o todo.

No planejamento da integração do sistema é definida a forma como a integração dos componentes, a partir dos subsistemas, será realizada e a ordem da integração que será realizada na iteração atual. Este subfluxo não foi modificado para processo proposto.

**3) Implementar Componentes**

Os implementadores desenvolvem o código fonte, adaptam os componentes existentes, compilam o código e realizam os testes de unidade, à medida que as classes são implementadas. O objetivo desta atividade é produzir o código-fonte do sistema, de acordo com o Modelo de Projeto, produzido na disciplina Análise de Projeto.

Os implementadores também consertam defeitos de código e realizam testes de unidade para verificar as mudanças. Em seguida, o código é inspecionado para avaliar a qualidade e a compatibilidade com as diretrizes de programação. Na Figura 4-3, pode-se observar as atividades que foram adaptadas para o *Code Coverage Process*. A forma que foi feita à adaptação será descrita a seguir.

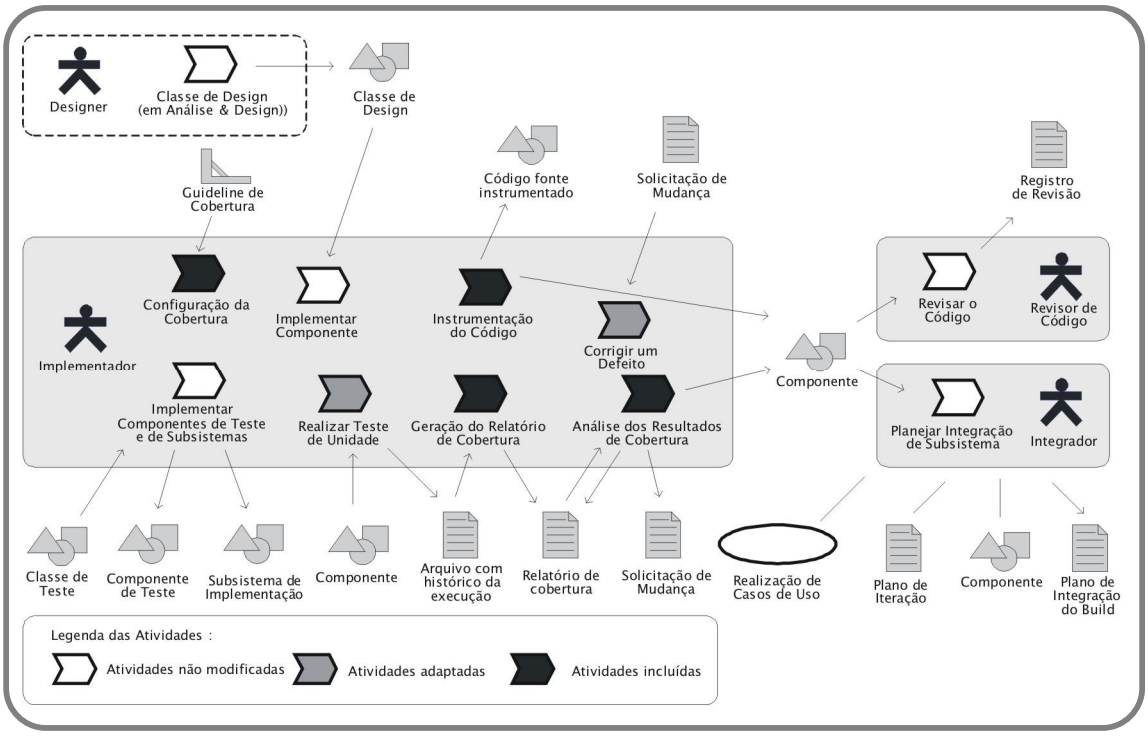



**Figura 4-3:** Subfluxo Implementar Componentes

A atividade "Corrigir Defeito" foi adaptada para o *Code Coverage Process*, de modo que o mesmo receba informações da atividade de análise dos resultados de cobertura, a fim de que possa corrigir defeitos encontrados no relatório de cobertura.

A atividade de "Realizar Teste de Unidade" foi também adaptada, pois o componente a ser testado contém o código instrumentado e dependendo da ferramenta de cobertura utilizada, será gerado um arquivo contendo o histórico da execução do teste de unidade.

As atividades, incluídas no subfluxo "Implementar Componente", propostas no *Code Coverage Process*, são as seguintes: Configuração da Ferramenta, Instrumentação do Código, Geração do Relatório de Cobertura e Análise dos Resultados de Cobertura.

A atividade **configuração da ferramenta** irá fazer a integração da ferramenta de cobertura de código ao ambiente de desenvolvimento e, se possível, a integração com

outras ferramentas do ambiente. Deve-se também definir e configurar os limites de cobertura desejáveis e os tipos de métricas que deverão ser coletadas.

A **instrumentação do código** irá inserir pontos de verificação no código, a fim de verificar se o local onde o ponto foi inserido realmente foi testado.

A **geração do relatório de cobertura** é a atividade em que são gerados os relatórios com a análise da cobertura do código. Dependendo da ferramenta, podem ser gerados vários tipos de relatórios, com formato textual e/ou gráfico.

A **análise dos resultados de cobertura** é a atividade em que são analisados e verificados os resultados das métricas de cobertura. Esta atividade serve como suporte para a tomada de decisões para o projeto, pois é a partir dela que se definirão as diretrizes para solucionar problemas relacionados à cobertura.

**4) Integrar cada Subsistema**

O objetivo deste subfluxo é integrar os componentes para formar subsistemas, os quais serão integrados para formar o sistema final. Na equipe de implementadores, deve haver um responsável pela integração dos novos componentes e pela atualização da versão do subsistema. A integração resulta em *builds* em um espaço de trabalho de integração do subsistema. A integração de cada *build* é verificada por um testador, para depois o subsistema ser liberado no espaço de trabalho de integração do sistema. O *build* gerado será instrumentado, a fim que se possam executar os testes e obter as métricas de cobertura de código. Na Figura 4-4, é possível observar que as atividades Integrar o Subsistema e Executar Conjunto de Teste foram adaptadas para o *Code Coverage Process*.

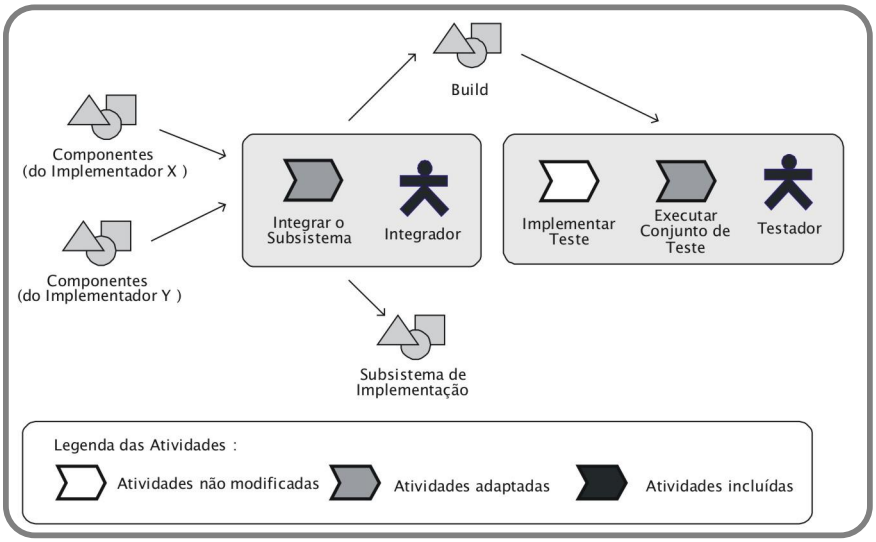



**Figura 4-4:** Subfluxo Integrar cada Subsistema

A adaptação deste subfluxo se deve ao fato de que a *build* do Subsistema de Implementação é instrumentada, com o intuito de que quando os testadores executarem os testes, será possível gerar o Relatório de Cobertura, a fim de analisar a cobertura do código.

O trabalho da integração normalmente é realizado de forma automática, o esforço manual é necessário quando a geração da *build* é interrompida. Algumas empresas utilizam a estratégia de gerar *builds* noturnas e alguns testes automáticos, com o intuito de não interromper o desenvolvimento e verificar a qualidade do produto.

Se dois ou mais implementadores estiverem trabalhando paralelamente no mesmo subsistema, o trabalho deles será integrado através de um espaço de trabalho de integração do subsistema, onde os componentes serão liberados a partir dos respectivos espaços de trabalho de desenvolvimento privado e no local onde o integrador construirá as *builds*.

**5) Integrar o Sistema**

Na atividade Integrar Sistema, os subsistemas são integrados para formar o sistema final. O sistema será testado e, uma vez aprovado, incluído na *baseline* do projeto. A integração de cada do *build* é verificada pelo testador. Após o último incremento, o sistema pode ser completamente testado por um testador. Na Figura 4-5, observa-se que as atividades Integrar o Subsistema e Executar Conjunto de Teste foram adaptadas para o *Code Coverage Process*.

Baseline é uma especificação ou produto que foi formalmente revisado e aceito, sendo modificado através apenas de procedimentos formais (solicitação de mudança).

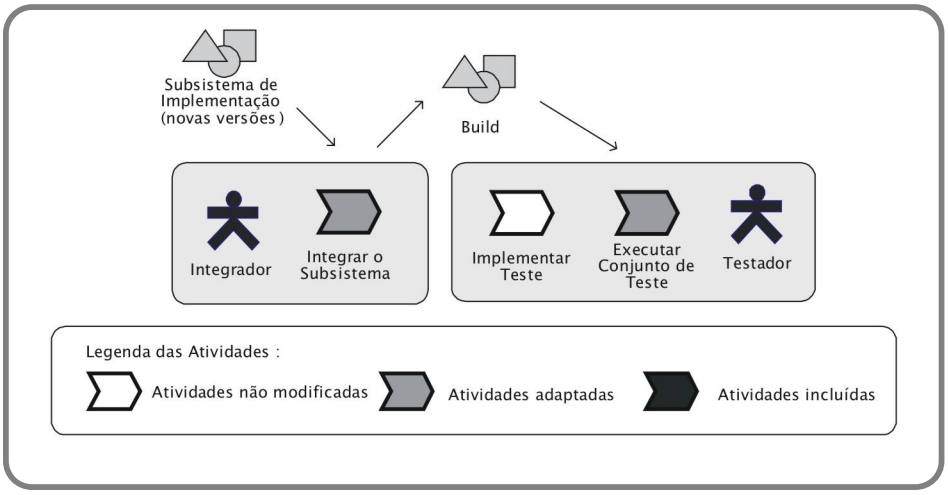



**Figura 4-5:** Subfluxo Integrar o Sistema

A adaptação deste subfluxo advém do fato que os *builds* que serão encaminhados para o testador serão instrumentados, a fim de obter as métricas de cobertura de código.

Para utilização de cobertura de código nos subfluxos Integrar o Sistema e Integrar cada Subsistema, é necessário ter uma ferramenta de cobertura que possa gerar *builds* do sistema instrumentadas.

### 4.2.2 Fluxo de Teste Estendido

O fluxo de Teste atua em diversos aspectos como um provedor de serviços para as outras disciplinas. O teste enfatiza principalmente a avaliação da qualidade do produto, realizada através de várias práticas centrais:

- Localizar e documentar defeitos de qualidade do software;
- Registrar a qualidade observada no software;
- Validar os produtos do software conforme especificados;
- Verificar se os requisitos foram implementados de forma adequada.

Uma diferença interessante entre o fluxo de Teste e os outros fluxos do RUP é que a principal finalidade do teste é localizar e expor os pontos fracos do software.

Apesar da análise de cobertura de código trabalhar com testes de unidade, os quais elaborados e executados no fluxo de Implementação, podem-se, dependendo da ferramenta de cobertura, utilizar a análise de cobertura de código no fluxo de Teste, como informação para verificação e validação da qualidade do produto, sendo que é preciso ter o cuidado ao analisar do Relatório de Cobertura, porque o relatório gera métricas baseadas em testes de unidade e no fluxo de Teste, são executados testes caixa preta. Na Seção 6.7 são detalhados os cuidados que se deve ter em relação à análise de cobertura de código, utilizando testes caixa preta.

A Cobertura de Código será de fundamental importância para medir a qualidade dos casos de testes de unidade. Claramente, o percentual de redundância de um conjunto de casos de testes, bem como a quantidade de código que é coberta pelos testes, são medidas importantes para a avaliação da qualidade da *suíte* gerada. Com essa informação, será possível orientar o processo de teste, de forma a otimizar a cobertura dos casos de teste elaborados.

A Figura 4-6 mostra o fluxo de Teste, no qual todos os subfluxos foram adaptados para utilização do *Code Coverage Process*.

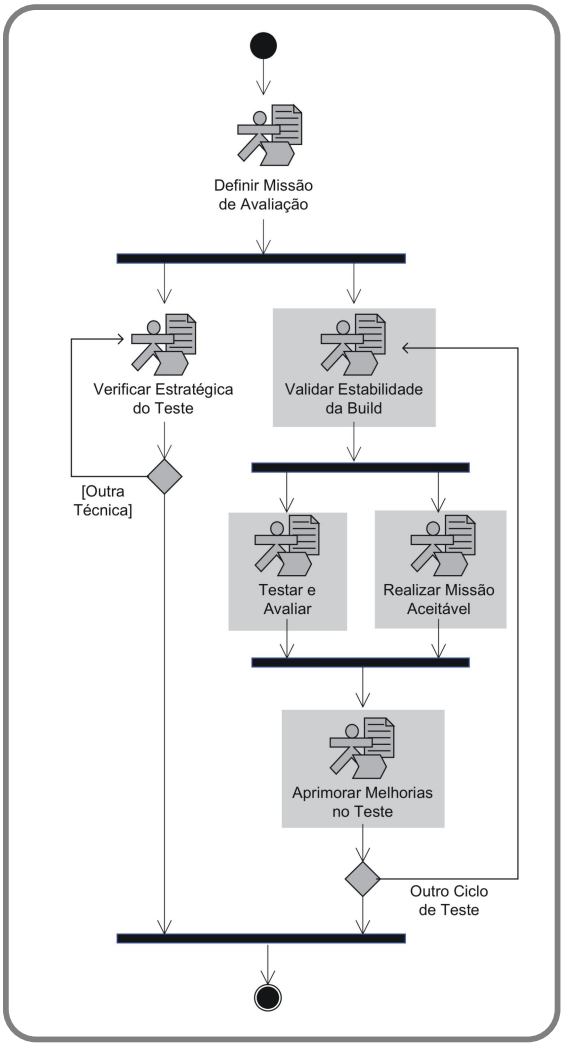



**Figura 4-6:** Fluxo de Teste do RUP estendido

O maior benefício da introdução de cobertura de código no fluxo de Teste é a verificação se um determinado teste, não sendo de unidade, poderá ou não ter eficácia na descoberta de defeito no código. Para isso, faz-se necessário executar alguns casos de teste, de um determinado conjunto, para saber quais destes testes terão possibilidade de descobrir um defeito rapidamente. Outro benefício é a possibilidade de eliminar ou selecionar casos de teste baseando-se na cobertura do código, tendo como resultado a diminuição do esforço da execução e/ou manutenção dos casos de testes.

A descrição das atividades de cada subfluxo que não foram adaptadas para a utilização do *Code Coverage Process*, bem como os detalhamentos dos artefatos, que não foram alterados, podem ser encontrados na versão 2003 do *Rational Unified Process*. A seguir são descritos cada subfluxo do fluxo de Teste.

**Definir Missão de Avaliação**

O propósito deste subfluxo é identificar o foco adequado do esforço de teste para a iteração e estabelecer consenso com os envolvidos sobre as metas correspondentes que conduzirão o esforço de teste.

Para cada iteração, deve-se:

- Identificar os objetivos, e os produtos liberados;
- Identificar uma boa estratégia de utilização de recursos;
- Definir o escopo e o limite adequados para o esforço de teste;
- Descrever a estratégica de teste que será utilizada;
- Definir como o progresso será monitorado e avaliado.

**Validar Estabilidade da Build**

O propósito deste subfluxo é validar a *build* tornando-o estável o suficiente para iniciar o teste. Esse trabalho também é denominado de teste de verificação do *build*. A Figura 4-7 mostra o subfluxo "Validar Estabilidade do *Build"* com as atividades que serão adaptadas.

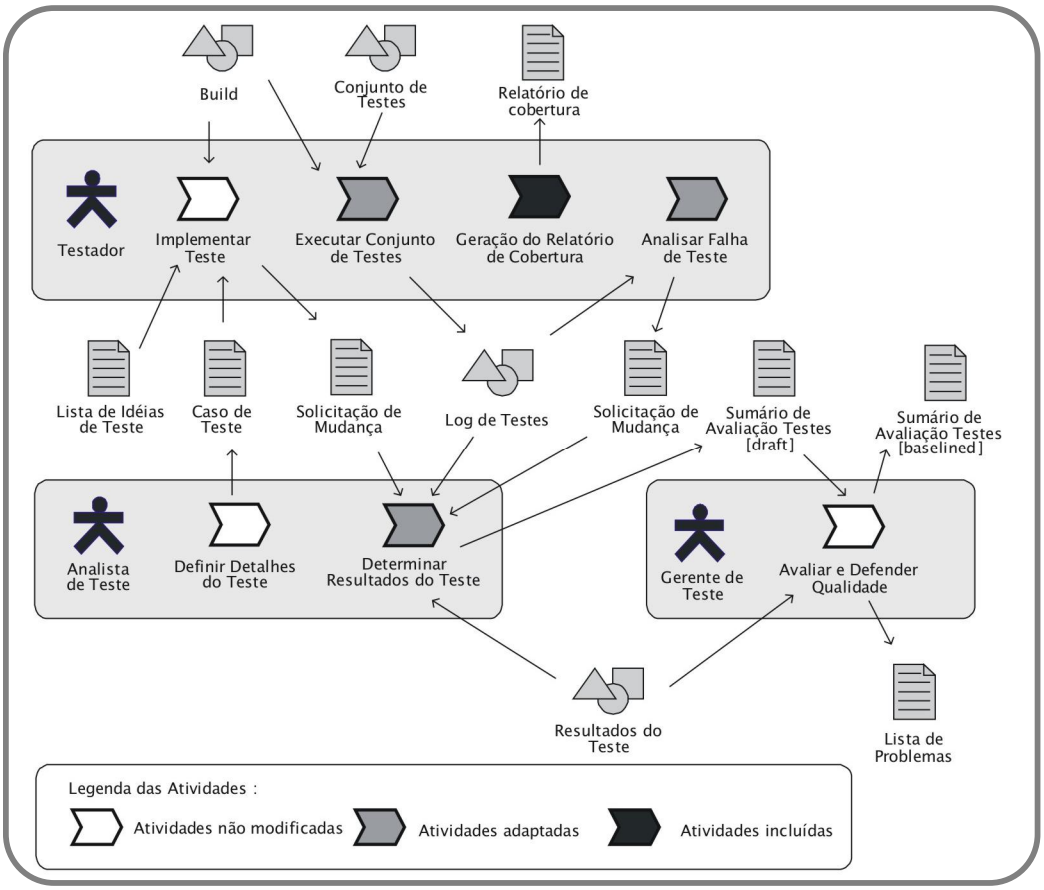



**Figura 4-7:** Subfluxo de Validar Estabilidade da *Build*

Este subfluxo será adaptado para o *Code Coverage Process,* descrita a seguir:

O **analista de teste** no momento que estiver definindo os detalhes dos testes irá identificar os pontos de observação da execução do teste, no qual se deseja observar algum aspecto do estado do ambiente de teste, e as condições nas quais deverão executar a análise da cobertura de código, para isso será utilizada uma ferramenta de cobertura para coletar as informações.

O **testador,** no momento que for realizar a atividade de executar os conjuntos de testes, terá que utilizar uma *build* que esteja instrumentada, para que possa posteriormente gerar o relatório de cobertura. Tendo o relatório de cobertura, poderá analisar os níveis de cobertura alcançados, com o objetivo de identificar onde ocorreu a falha do teste, de modo a corrigir e registrar descobertas importantes. O relatório de cobertura poderá, dependendo

da ferramenta utilizada, ser feito automaticamente à medida que os casos de teste forem executados.

O **analista de teste** terá na atividade de "Determinar os Resultados do Teste" definir a condição de validação dos resultados da cobertura de código, ou seja, determinará o limiar de aceitação da cobertura, pois se tem que levar em consideração que a ferramenta de cobertura produzirá o relatório baseado nos testes executados, sendo que neste fluxo estarão sendo utilizado casos de testes caixa preta, logo é possível ter trecho de código que só poderá ser testado na fase de teste de unidade ou código que possui uma *feature* que não faz parte do escopo da suíte de teste, ocasionando uma baixa nos resultados da cobertura do código. Para isso, faz-se necessária uma análise no código para determinar o limiar de aceitação. Esta análise poderá ser feita em conjunto com o implementador, pois o mesmo possui um maior conhecimento do que foi produzido.

O **gerente de testes** irá analisar as informações provenientes do relatório de cobertura e decidirá sobre os defeitos encontrados nos testes. Reportará os resultados da cobertura para as pessoas envolvidas (Gerente de Projeto, Implementador e Gerente de Qualidade), principalmente no que diz respeito às informações de melhoria e/ou defeitos encontrados no código. Revisará os limites de aceitação da cobertura, no intuito de possíveis ajustes, monitorando o andamento e dando suporte às mudanças que elevem a qualidade do software. O gerente de teste contará com o suporte do implementador para realizar a análise do resultado da cobertura, devido ao conhecimento do produto de desenvolvimento, analisando o código com maior precisão.

**Testar e Avaliar**

Geralmente é realizado, uma vez por ciclo de teste, este trabalho envolve a todo o esforço da execução do teste e avaliação, ou seja, a implementação, a execução e a avaliação de testes e o relatório correspondente dos *bugs* encontrados.

A Figura 4-8 mostra o subfluxo "Testar e Avaliar", com as atividades que serão adaptadas.

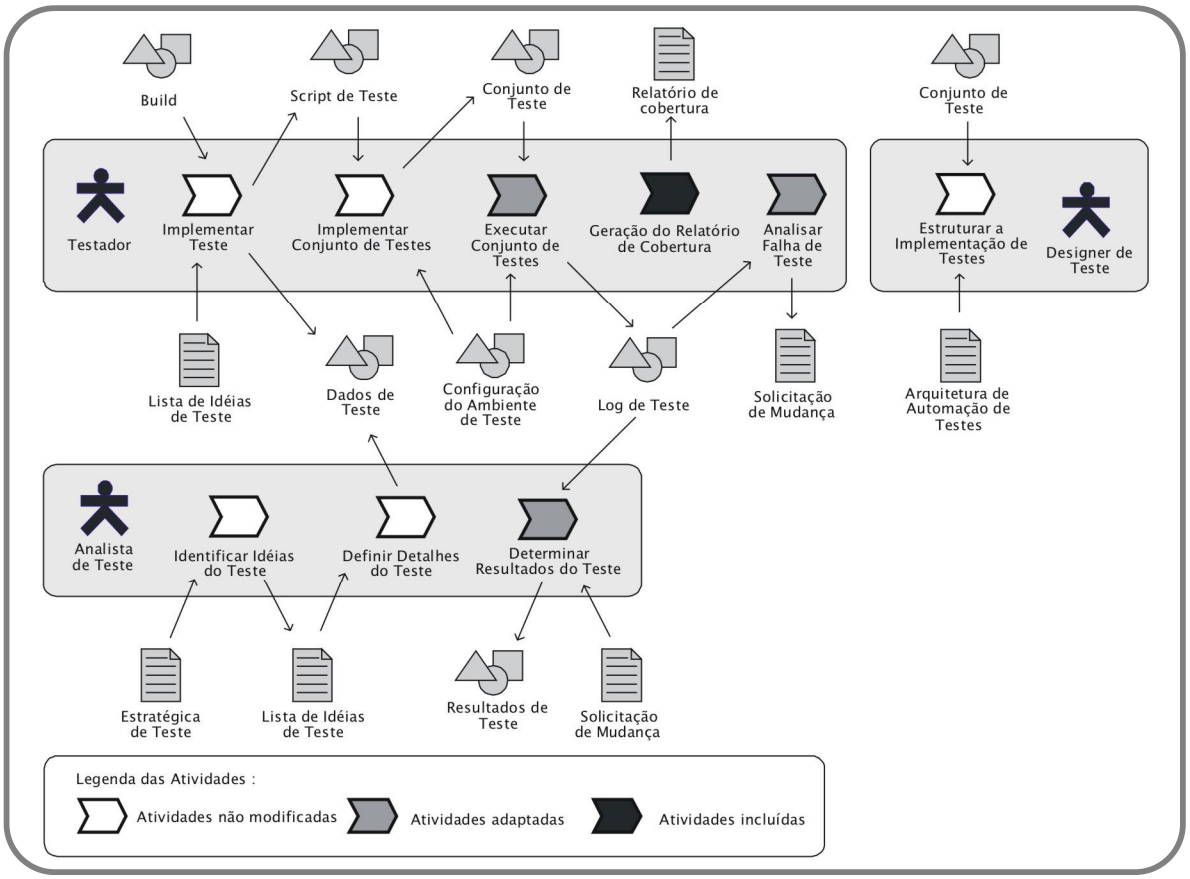



**Figura 4-8:** Subfluxo de Testar e Avaliar

A atividade de "Determinar Resultados do Teste" é realizada pelo analista de teste, e as atividades do testador que foram adaptadas e incluídas para a utilização do *Code Coverage Process*, seguem o mesmo procedimento descrito anteriormente nas atividades adaptadas e incluídas no subfluxo "Validar Estabilidade do *Build"*.

**Realizar Missão Aceitável**

O objetivo é oferecer um resultado útil da avaliação feita pelos testes para os *stakeholders*. Esse resultado é examinado de acordo com a Missão de Avaliação. Na maioria dos casos, isso significará concentrar seus esforços para ajudar a equipe do projeto a atingir os objetivos do Plano de Iteração aplicável ao ciclo de teste atual.

A Figura 4-9 mostra o subfluxo "Realizar Missão Aceitável", com as atividades que serão adaptadas.

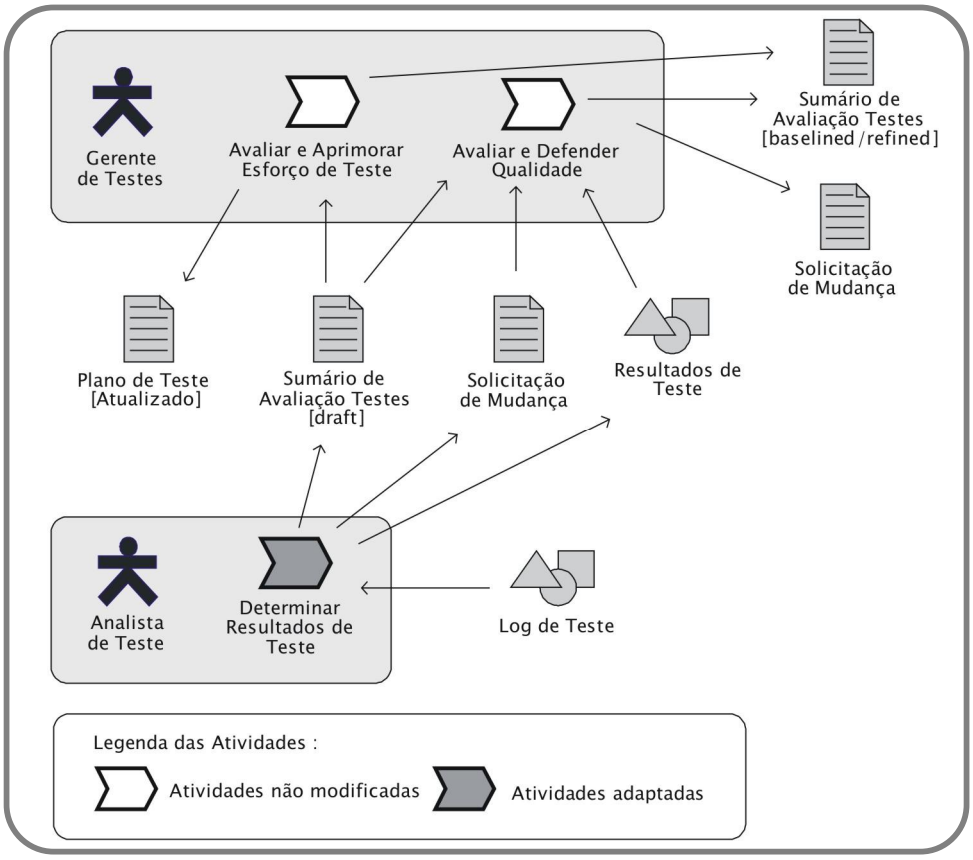



**Figura 4-9:** Subfluxo de Realizar Missão Aceitável

Atividade Determinar Resultado de Teste realizado pelo analista de teste, foi adaptada com o objetivo de planejar e definir um limiar de aceitação para a validação dos resultados obtidos no relatório da análise da cobertura do código, pois estaremos executando testes de caixa preta. Nesta atividade, é averiguado se o código do produto está sendo avaliado, a fim de determinar o limite de aceitação, para isso é necessário o suporte do implementador, pois, em muitos dos casos o analista de teste não possui conhecimento suficiente referente ao código.

A atividade "Avaliar e Aprimorar Esforço de Teste" foi adaptado, com intuito de reavaliar o limiar de cobertura imposto para o produto. Esta atividade deve ter o apoio de pessoas envolvidas em outras atividades, como o implementador e o gerente de projeto, para que se possa melhorar a qualidade dos testes.

**Aprimorar Melhorias no Teste**

O objetivo é manter e melhorar os testes. Isso é especialmente importante caso a intenção seja reutilizar as vantagens desenvolvidas no ciclo atual de teste em ciclos subseqüentes.

Para cada ciclo de teste, o foco principal do trabalho será:

- Adicionar o conjunto mínimo de testes para validar a estabilidade dos *builds* posteriores;
- Montar *scripts* de teste;
- Remover as melhorias do teste implantadas que não tenham mais finalidade ou cuja manutenção se tenha tornado dispendiosa;
- Manter configurações do ambiente de teste;
- Explorar as oportunidades para reutilização e melhorias na produtividade;
- Realizar manutenção e melhorias nos testes automatizados;
- Documentar as lições aprendidas, tanto as boas práticas quanto ás más práticas descobertas durante o ciclo de teste.

A Figura 4-10 mostra o subfluxo “Aprimorar Melhorias no Teste”, com as atividades que serão adaptadas.

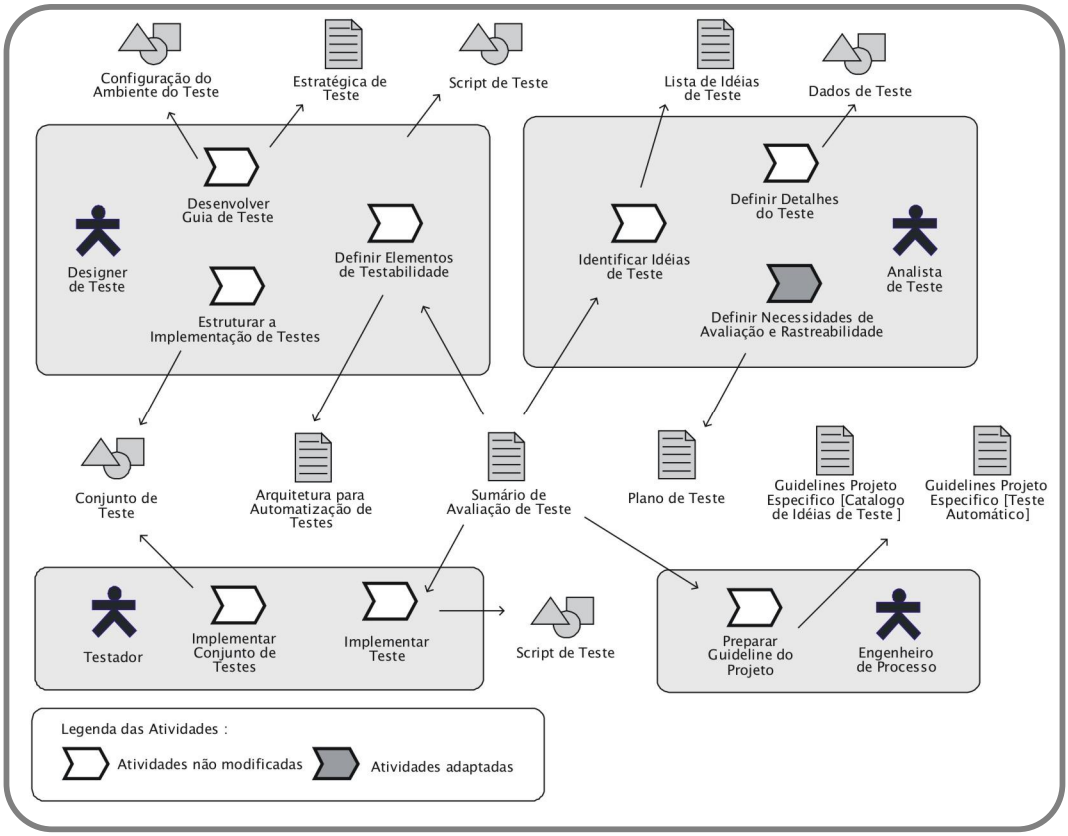



**Figura 4-10:** Subfluxo Aprimorar Melhorias no Teste

Neste subfluxo, a única atividade que foi adaptada foi a "Definir Necessidade de Avaliação e Rastreabilidade". Este subfluxo trata de um aprimoramento dos testes, ocorrendo normalmente no fim de cada ciclo de teste. Pode-se reavaliar os casos de teste, a fim de selecionar ou criar conjuntos de testes que tem um maior impacto na cobertura de código.

### Verificar Estratégica do Teste

O objetivo é demonstrar que as várias técnicas descritas na Estratégica do Teste que facilitarão o esforço do teste. O propósito é compreender as restrições e limitações de cada técnica e encontrar uma solução de implementação adequada ou encontrar técnicas alternativas que possam ser implementadas. Isso ajuda a diminuir o risco de descobrir muito tarde, no ciclo de vida do projeto, que a abordagem do teste não funciona.

Para cada iteração, o foco principal do trabalho será:

- Verificar o quanto antes se a Estratégica de Teste pretendida funcionará e produzirá resultados valiosos;
- Estabelecer a infra-estrutura básica para possibilitar e dar suporte à Estratégica do Teste;
- Obter comprometimento da equipe de desenvolvimento para fornecer e dar suporte a testabilidade exigida para atingir a Estratégica de Teste;
- Identificar escopo, fronteiras, limitações e restrições de cada técnica.

### 4.2.3 Fluxo de Gerenciamento de Configuração e Mudança Estendido

Gerencia de Configuração é o processo de identificar, organizar e controlar modificações ao software que está sendo desenvolvido. Durante o ciclo de vida do desenvolvimento são criados muitos artefatos, no qual são importantes e devem ser guardados e disponíveis para uso. Estes artefatos evoluem e, especialmente em desenvolvimento iterativo, são atualizados constantemente. "A maioria das pessoas tende a pensar que a mudança é decorrente de problemas ou de conseqüências negativas. Embora seja verdade que mudança possui aspecto negativo, como também positivo" [Baca, 2005].

A Gerencia de Configuração tem os seguintes objetivos:

- Definir o ambiente de desenvolvimento;
- Definir política para controle de versões garantindo a consistência dos artefatos produzidos;
- Definir procedimentos para solicitações de mudanças;
- Administrar o ambiente e auditar mudanças;
- Facilitar a integração das partes do sistema.

No Gerenciamento de Configuração e Mudança, toda atividade ligada à cobertura de código necessitará de um suporte e um controle dos artefatos gerados para a medição da cobertura. Desde a coleta do código, dentro do sistema de controle de versão, até o fechamento da Solicitação de Mudança da implementação ou modificação do código. A

Figura 4-11 mostra o fluxo de Gerenciamento de Configuração e Mudança, destacando os subfluxos que serão adaptados com a utilização do *Code Coverage Process*.

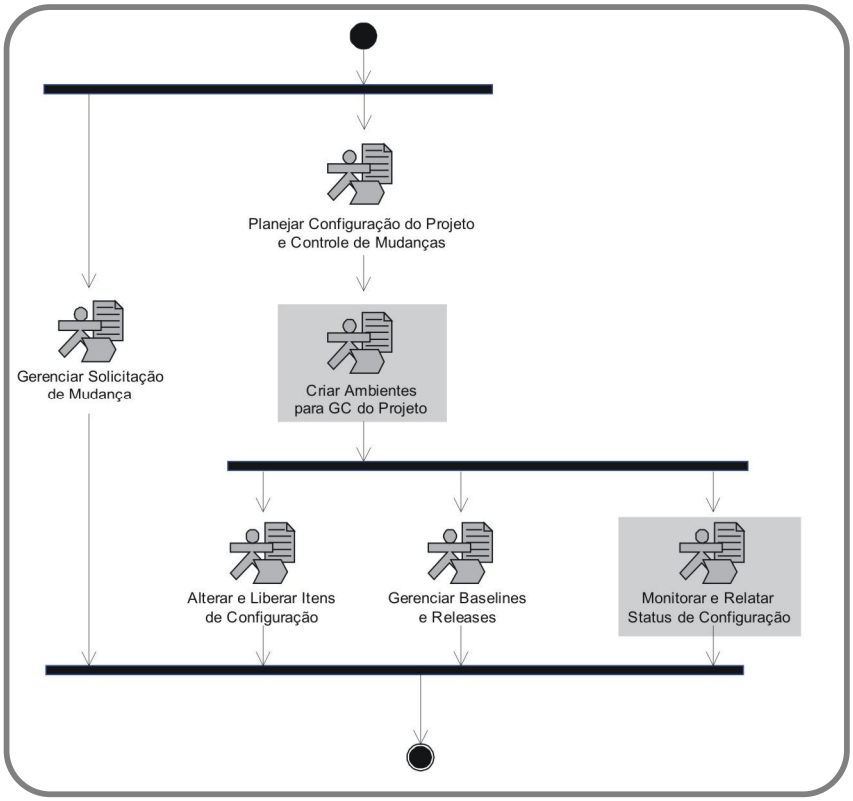



**Figura 4-11:** Fluxo de Gerenciamento de Configuração e Mudança do RUP estendido

A seguir, serão descritos cada subfluxo do fluxo Gerenciamento de Configuração e Mudança, tendo destaque para os subfluxos "Criar Ambientes para GC do Projeto" e "Monitorar e Relatar Status de Configuração", pois serão adaptados para o *Code Coverage Process*.

A descrição das atividades de cada subfluxo, que não foram adaptadas para a utilização do *Code Coverage Process*, bem como os detalhamentos dos artefatos, os quais não foram alterados, podem ser encontrados na versão 2003 do *Rational Unified Process*.

**Planejar Configuração do Projeto e Controle de Mudanças**

O objetivo é o estabelecimento de políticas para o gerenciamento de configuração do projeto e controle das mudanças realizadas no produto, documentando as informações no Plano de Gerenciamento de Configuração (GC). As políticas de Gerência de Configuração estão relacionadas com a capacidade de identificar e reportar os artefatos que foram aprovados para uso em um projeto.

**Criar Ambientes para Gerência de Configuração do Projeto**

O objetivo é estabelecer um ambiente mediante a criação e manutenção de repositórios de dados, nos quais o produto poderá ser desenvolvido, compilado e disponibilizado para manutenção e reutilização. Este trabalho faz com que os artefatos essenciais estejam disponíveis para os desenvolvedores e integradores nos diversos espaços de trabalho privados e públicos. Inicia na fase de iniciação e continua ao longo do ciclo de vida. A Figura 4-12 apresenta o subfluxo com as atividades que foram adaptadas.

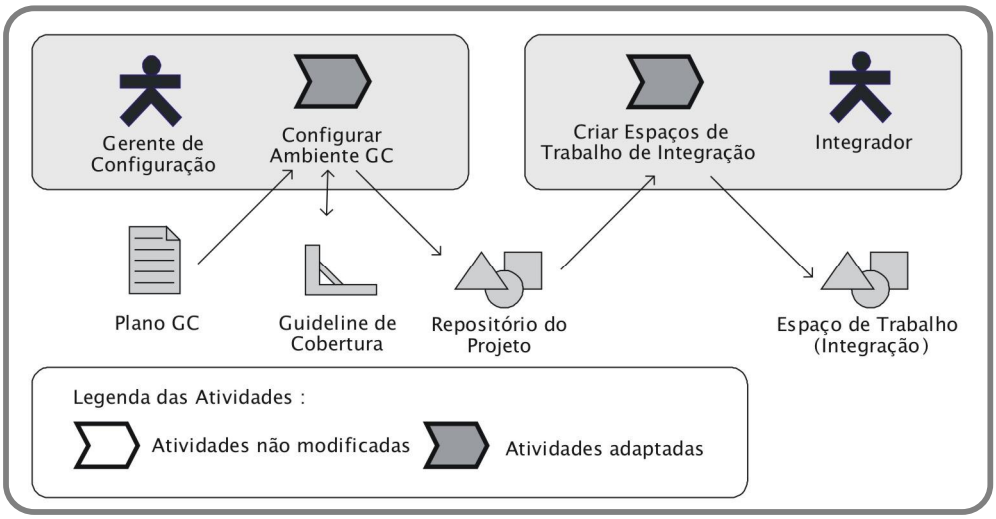



**Figura 4-12:** Subfluxo Criar Ambientes para GC do Projeto

Os artefatos *Guideline* de Cobertura e Relatório de Cobertura, produzidos nas atividades referentes à cobertura de código no processo proposto, serão controlados pelo "Gerenciamento de Controle e Mudança".

Estes dois artefatos poderão sofrer mudanças ao longo do projeto. O *Guideline* poderá sofrer alterações, devido à inserção ou modificação dos documentos relacionados a uma nova característica do produto, da ferramenta ou do ambiente de desenvolvimento. O Relatório de Cobertura, porque será monitorada a qualidade da cobertura do produto, a fim que se possa acompanhar a sua evolução da *build*. O benefício de ter um repositório que armazene os Relatórios de Cobertura é que, com o tempo pode-se realizar um estudo sobre o nível de cobertura adequado para projeto que esta sendo avaliado.

O código fonte instrumentado não necessita ter controle de versão, pois a geração de *builds* e as integrações do sistema são feitos utilizando o código original, o qual já possui

controle de mudança. Neste subfluxo não é criada nenhuma nova atividade, tendo apenas a inserção dos dois artefatos mencionados anteriormente a serem colocados no repositório do projeto.

O Repositório do Projeto deverá estar configurado e com permissão de acesso para que a ferramenta de cobertura possa utilizar o código fonte e os testes de unidade para fazer a instrumentação do código e a execução dos testes, a fim que se possa gerar o relatório de cobertura. O *Guideline* da Cobertura terá que ter informações de como acessar o código e o procedimento da integração da ferramenta ao ambiente de desenvolvimento, essas informações serão inseridas no *guideline* pelo gerente de configuração.

A vantagem de se ter um repositório dos Relatórios de Cobertura do projeto, sendo armazenado na medida em que o relatório for sendo gerado, irá auxiliar na análise das métricas que foram especificadas e que devem ser mantidas ou alteradas caso seja necessário. Essas análises poderão ser feitas por iteração.

**Alterar e Liberar Itens de Configuração**

Este subfluxo refere-se à maneira de criar um espaço de trabalho. Dentro deste espaço podem-se acessar artefatos do projeto, fazer mudanças nesses artefatos, liberar as mudanças para inclusão no produto geral. A idéia é integrar as contribuições individuais e torná-las disponível para os demais membros da equipe de desenvolvimento.

**Gerenciar *Baselines* e *Releases***

Um produto deve ter *baseline* cada vez que é liberado para o cliente. Quando um cliente tiver problema com um release liberado, a equipe de desenvolvimento ou de manutenção pode reverter o *baseline* para recriar e reparar o defeito. A freqüência e a formalidade das *releases* estão descritas no Plano de Gerência de Configuração.

**Gerenciar Solicitação de Mudança**

Processos padrão de controle de mudanças documentados têm a finalidade de assegurar, que as mudanças realizadas em um projeto, fiquem consistentes e que os envolvidos sejam notificados a respeito do estado do produto, das mudanças realizadas e do impacto de custo e programação gerados por essas mudanças.

**Monitorar e Relatar Status de Configuração**

A finalidade deste subfluxo de trabalho é:

- Determinar se o produto atende aos requisitos funcionais e físicos;
- Determinar se os artefatos necessários estão armazenados e se a respectiva *baseline* foi definida;
- Assegurar que os artefatos e as *baselines* estão disponíveis;
- Facilitar a revisão do produto por meio de relatórios e controle de defeitos;

O gerente de configuração, em conformidade com o Plano de Gerenciamento de Configuração, deve fornecer relatórios quantitativos que podem ser usados para avaliar o status do projeto. A Figura 4-13 apresenta o subfluxo "Monitorar e Relatar status de Configuração".

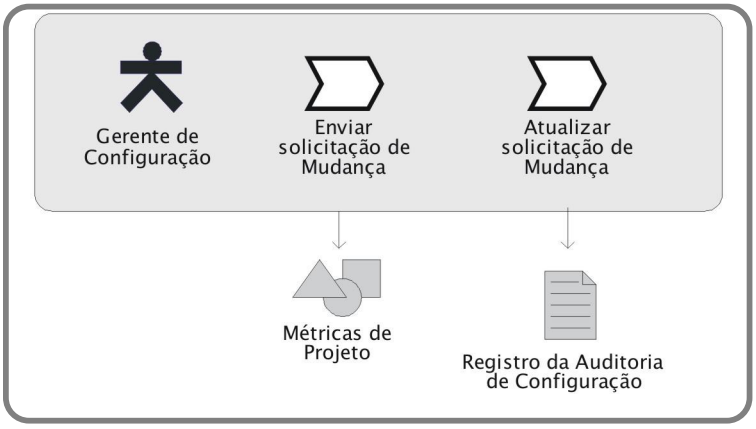



**Figura 4-13:** Subfluxo Monitorar e Relatar Status de Configuração

No *Code Coverage Process,* este subfluxo não terá adaptação ou inclusão de atividades. A única modificação neste subfluxo será no artefato Métricas de Projeto, o qual terá mudanças relacionadas com as métricas de cobertura de código, proveniente de inserção das métricas de cobertura e de ajustes realizados pelo gerente de projeto, a fim de adequar os limites que serão especificados para que o código e testes de unidade possam ser aprovados.

### 4.2.4 Fluxo de Gerenciamento de Projeto do RUP Estendido

O fluxo Gerenciamento de Projeto tem a finalidade de analisar as métricas coletadas através dos relatórios gerados pela ferramenta de cobertura, bem como tomar posicionamento e diretrizes para que a qualidade dos testes melhore, ou seja, que as métricas coletadas não fiquem abaixo do nível de cobertura estabelecido pela organização. Uma das vantagens da análise de cobertura de código é que se pode ter a noção dos casos de testes que deverão existir para assegurar a qualidade do código, com isso o gerente de projeto tem a facilidade de fazer uma estimativa dos recursos e do tempo necessário para a entrega do produto. A Figura 4-14 mostra o fluxo de Gerenciamento de Projeto, destacando os subfluxos que serão adaptados com a utilização do *Code Coverage Process*.

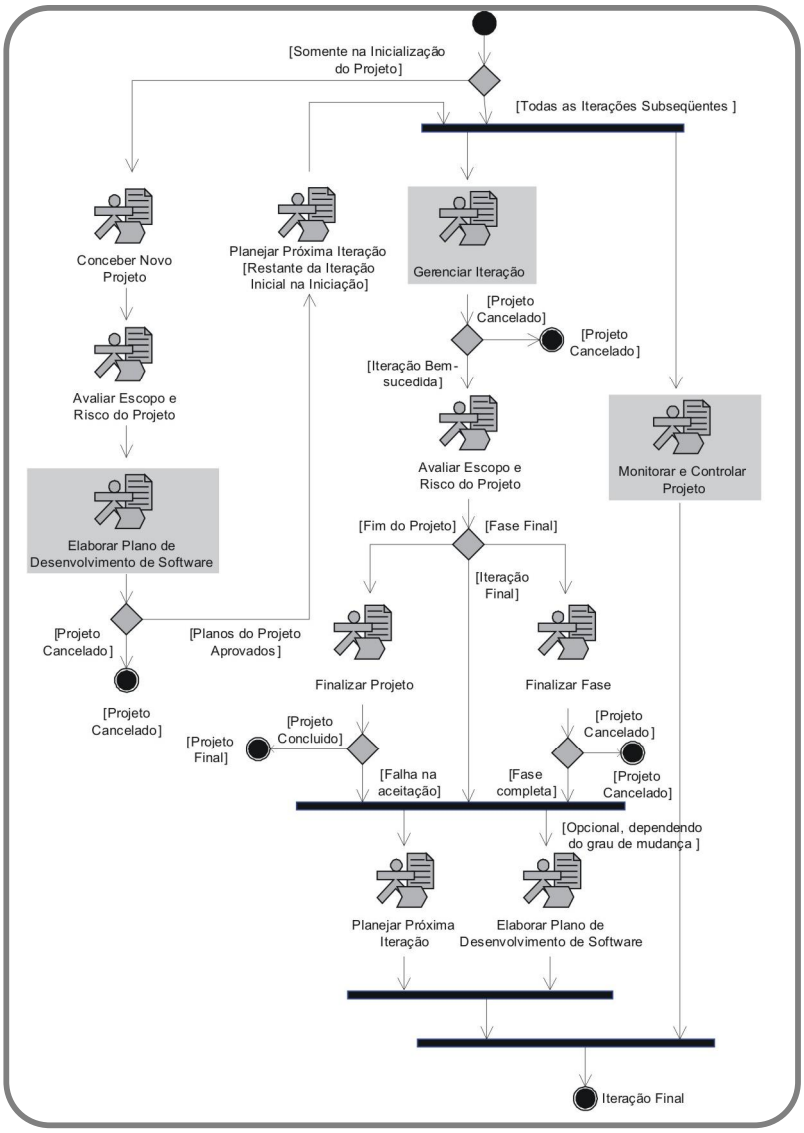



**Figura 4-14:** Fluxo de Gerenciamento de Projeto estendido

A seguir, as descrições de cada subfluxo do fluxo Gerenciamento de Projeto, são apresentadas dando destaque aos subfluxos "Elaborar Plano de Desenvolvimento de Software"; "Gerenciar Iteração e Monitorar" e "Controlar o Projeto", pois serão adaptados para o *Code Coverage Process*.

A descrição das atividades de cada subfluxo que não foram adaptadas para a utilização do *Code Coverage Process*, bem como os detalhamentos dos artefatos, que não foram alterados, podem ser encontrados na versão 2003 do *Rational Unified Process*.

**Conceber Novo Projeto**

Este subfluxo é composto das atividades para identificar e avaliar os riscos, desenvolvendo o caso de negócio. A finalidade deste subfluxo é apresentar um projeto desde a origem de uma idéia até o ponto em que é possível tomar decisões para continuar ou abandonar o projeto.

**Avaliar Escopo e Risco do Projeto**

Este subfluxo é composto das atividades "Identificar e Avaliar os Riscos" e "Desenvolver Caso de Negócio". A finalidade é reavaliar as capacidades especificadas, as características do projeto e os riscos associados para alcançá-las.

**Elaborar Plano de Desenvolvimento de Software**

Este subfluxo tem a finalidade de desenvolver os componentes e os documentos anexos ao Plano de Desenvolvimento de Software, e revisá-lo formalmente a fim de obter o aceite dos envolvidos e servir como base para o Plano de Iteração. O maior esforço na criação desses artefatos ocorre logo na fase de iniciação; depois disso, quando o fluxo de trabalho é chamado no início de cada iteração, ocorre à revisão do Plano de Desenvolvimento de Software com base na experiência da iteração anterior e no Plano de Iteração para a próxima. A Figura 4-15 apresenta o subfluxo "Elaborar Plano de Desenvolvimento de Software".

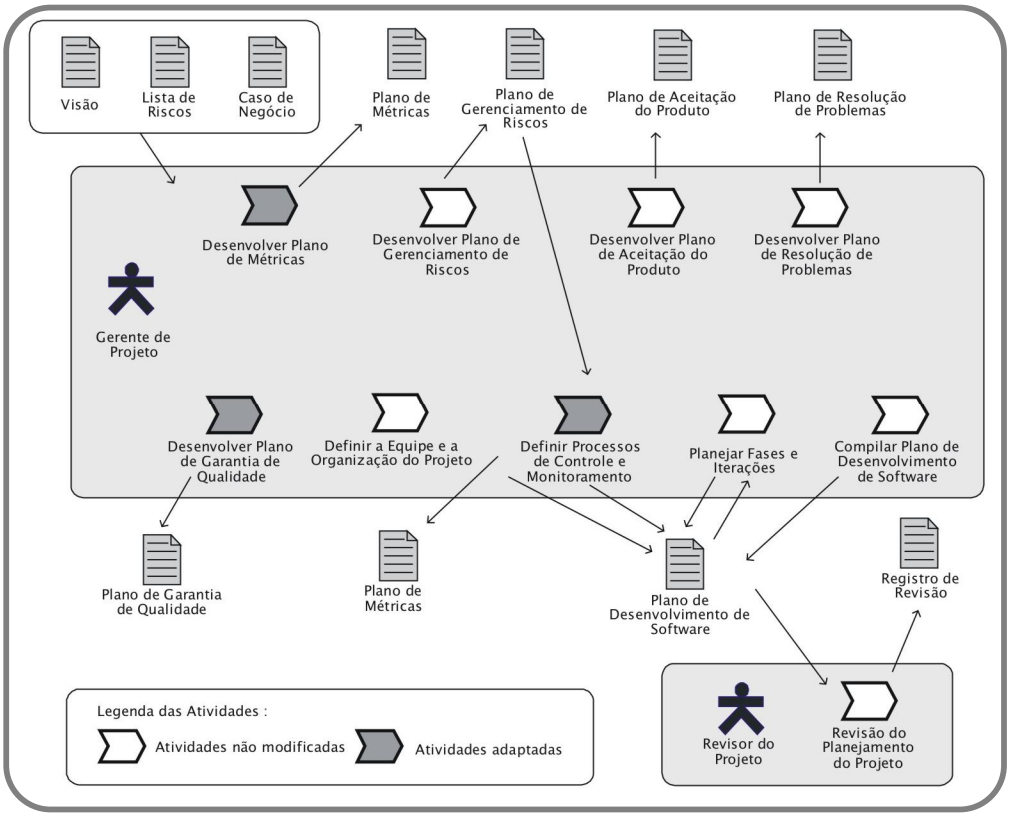



**Figura 4-15:** Subfluxo Elaborar Plano de Desenvolvimento de Software

No fluxo "Elaborar Plano de Desenvolvimento de Software", as únicas atividades que foram adaptadas para o *Code Coverage Process*, foram as seguintes: (a) "Desenvolver Plano de Métricas"; (b) "Desenvolver Plano de Garantia de Qualidade" e (c) "Definir Processos de Controle e Monitoramento". A seguir, são descritas as atividades que foram adaptadas para o *Code Coverage Process*.

**a) Desenvolver Plano de Métricas**

A atividade "Desenvolver Plano de Métricas" é realizada uma vez por projeto, na fase de iniciação, como parte da atividade de planejamento geral. O Plano de Métricas pode ser revisado, como qualquer outra seção do plano de desenvolvimento de software, durante o projeto. A finalidade desta atividade é definir metas de gerenciamento, no que se refere a qualidade, progresso e melhoria do desenvolvimento; e determinar o que precisa ser medido periodicamente para suportar essas metas.

O Plano de Métricas descreve as metas que o projeto deverá cumprir para ser concluído com êxito, bem como as métricas a serem usadas para determinar se projeto está no caminho certo.

Nesta atividade, são realizados o estudo e a definição das métricas e limiares de cobertura de código, sendo uma etapa de suma importância na utilização do *Code Coverage Process*. Esta etapa exige atenção, por proporcionar o resultado final do processo de cobertura, e é dela que partem as ações corretivas e as tomadas de decisões referentes à análise da eficácia dos testes em relação ao código. Para o estudo e definição das métricas de cobertura, o gerente de projeto poderá contar com o suporte do desenvolvedor e/ou algum membro da equipe de testes (testador e/ou analista de teste).

Para definição das métricas e limiares de cobertura de código a serem inseridos no Plano de Métricas, é necessário identificar o escopo e a complexidade do projeto que será avaliado. Como existem diversas métricas de cobertura de código (Ex.: *Statement Coverage, Condition Coverage, Function Coverage, Path Coverage*), deve-se selecionar as métricas que atendam os objetivos da organização, no caso se a ferramenta de cobertura não proporciona um número de métricas que atenda ao interesse do projeto, deve-se analisar a possibilidade de utilizar outra ferramenta de análise de cobertura de código.

**b) Desenvolver Plano de Garantia de Qualidade**

O Plano de Garantia de Qualidade é um documento formado por todas as informações necessárias para realizar as atividades de garantia de qualidade do projeto. O Plano de Garantia de Qualidade é utilizado para elaborar um programa de revisões e auditorias que verificará se o processo do projeto definido está sendo seguido corretamente.

Em relação ao processo proposto, este Plano de Garantia de Qualidade deverá conter informações para o comprometimento das métricas e do processo de cobertura de código.

Como as métricas estão ligadas à garantia da qualidade do software, o gerente de qualidade terá um papel importante nesta etapa, visto que ficará supervisionando e monitorando se as métricas e limiares realmente estão sendo seguidos, como também ficará atento aos resultados, procurando sempre questionar quando os indicadores de cobertura estiverem abaixo do que foi determinado no projeto.

**c) Definir Processos de Controle e Monitoramento**

Esta atividade irá definir as informações e os processos que serão usados para o monitoramento e controle do andamento, da qualidade e dos riscos do projeto. Como esta

atividade irá definir os "indicadores" para o projeto, o processo proposto insere mais dois passos para esta atividade, são os seguintes: seleção de métricas de cobertura e definição de métricas.

Na seleção de métricas de cobertura, utiliza-se a abordagem de McAndrews (1993), a qual consiste em identificar o escopo da atividade de medição identificando as necessidades de medição, escolhendo o conjunto de métricas que garantam ao gerente de projeto a informação de como está a real situação da qualidade dos testes gerados no projeto, resultando.

Na definição das métricas faz-se a seleção das métricas de cobertura, define-se a utilidade de cada uma em relação ao projeto, ou seja, o que aquela métrica irá proporcionar de melhoria no projeto, pois uma métrica de cobertura deve ser eficiente e agregar valor ao processo.

A maior dificuldade em definir métricas de cobertura é determinar o nível de cobertura aceitável para cada tipo métrica de cobertura (Ex.: *Statement Coverage, Condition Coverage, Function Coverage, Path Coverage*) adotada no projeto, para que se tenha uma melhor interpretação dos resultados, pois os indicadores da qualidade serão impactados, devido à utilização dos níveis de cobertura como base para análise da qualidade dos casos de testes gerados. Para se ter um nível desejável e que garanta o progresso da organização, é aconselhável fazer um estudo e uma análise dos dados históricos gerados no projeto, feito isso, tem-se uma real noção de qual nível de cobertura deverá ser estabelecido. Se não for possível fazer uma análise histórica dos dados, deve-se levar em consideração a sugestão dada pela ferramenta de cobertura utilizada no processo. É importante salientar que esse nível poderá variar de acordo com tipos e características de cada organização e/ou projeto.

As melhores fontes de informação para decidir os limites são os próprios dados da organização, pois utilizar métricas de outras organizações e/ou outros projetos, tendo nível de complexidade e característica diferentes, pode se ter uma análise dos resultados errada, logo deve-se utilizar os dados do projeto da própria organização para se determinar o nível aceitável de cobertura, principalmente utilizando bases históricas do projeto que se deseja

analisar a cobertura, realizando estas análises retroativamente, tendo como intuito obter um nível desejável de cobertura.

**Planejar da Próxima Iteração**

Este subfluxo é responsável por criar um Plano de Iteração, que consiste num plano para orientar a execução da iteração. Depois de criar o plano, podem ser necessários ajustes para o Caso de Negócio (por exemplo, se os custos são alterados, ou se o cálculo de retorno do investimento é afetado pelas mudanças de datas de disponibilidade dos recursos importantes no software).

**Gerenciar Iteração**

O subfluxo "Gerenciar Iteração" contém as atividades que iniciam, finalizam e revisam uma iteração. O objetivo é adquirir os recursos necessários para executar a iteração, alocar o trabalho que será realizado e, por fim, avaliar os resultados da iteração, que é concluída com uma revisão de aceitação, que determina se os objetivos da iteração foram alcançados.

No subfluxo de "Gerenciar Iteração" a única atividade alterada, por conta da utilização do *Code Coverage Process,* foi a atividade "Avaliar Iteração", a qual levará em consideração o Relatório de Cobertura produzido na iteração, a fim que se possa analisar a qualidade dos testes de unidade executados, podendo, se necessário, tomar ações corretivas para as próximas iterações. A Figura 4-16 mostra a atividade adaptada.

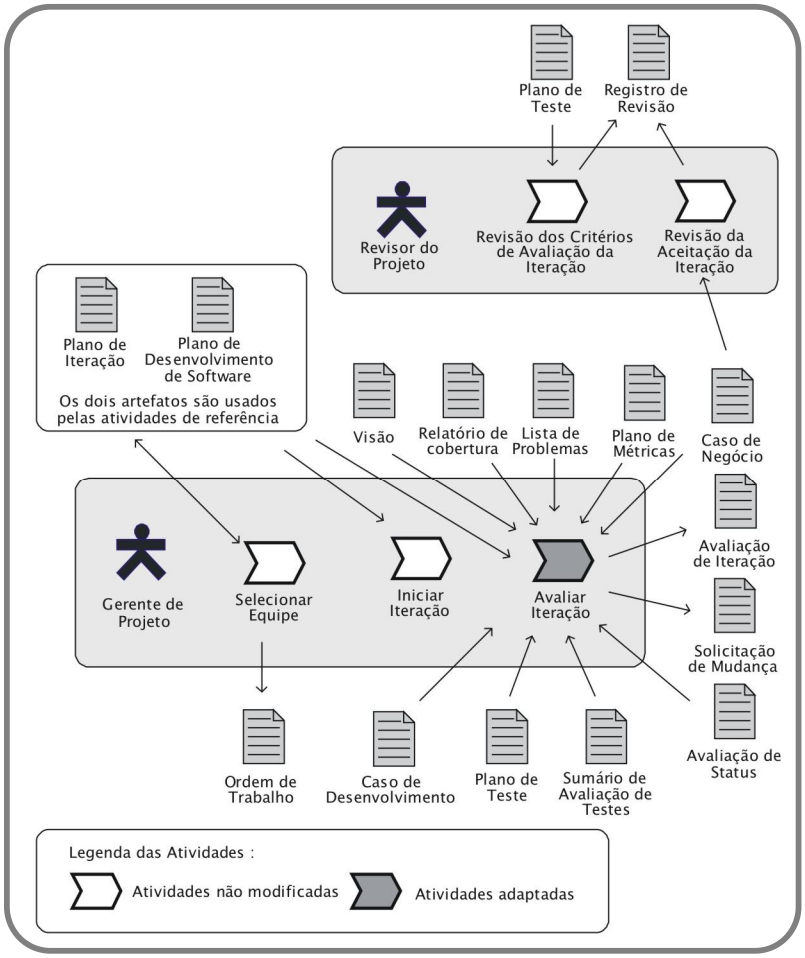



**Figura 4-16:** Subfluxo Gerenciar Iteração

A atividade "Avaliar Iteração" irá determinar se a iteração foi concluída com êxito ou apresentou defeito, capturando as lições aprendidas para modificar o projeto ou melhorar o processo. A adaptação realizada nesta atividade leva em consideração a análise do Relatório de Cobertura, a fim de que se possa obter o status da cobertura e efetuar melhorias nos testes de unidade produzidos.

O relatório de cobertura deve ser de fácil compreensão e os resultados das métricas devem ser claros e objetivos, de preferência mostrando onde realmente o código não está sendo coberto. Um bom relatório é aquele que contém sugestões para se obter uma melhor cobertura, bem como o que contém a análise crítica dos testes de unidade executados, além de propor um melhor desempenho e eficácia, principalmente apontando para os casos de testes que estão cobrindo o mesmo trecho de código e que podem ser removidos sem que a cobertura do código seja afetada.

Deve-se observar, na análise do relatório, qual parte do código que não foi coberto e se há casos de testes redundantes que podem ser removidos, reduzindo esforços na sua realização, mas não diminuindo a cobertura de código. Os relatórios de cobertura devem ser armazenados, controlados (registrados) e devem fazer parte da documentação de finalização da iteração do projeto. O objetivo é monitorar a evolução de todo o projeto e no futuro fazer uma análise das métricas utilizadas.

**Finalizar Projeto**

Este subfluxo é responsável pela atividade de encerramento do projeto, sendo executado pelo gerente de projeto. Uma avaliação do status final do projeto é elaborada para uma revisão formal entre a equipe do projeto e um representante do cliente, no qual é verificado se o produto e a documentação de suporte fornecido pelo projeto atendem aos requisitos e objetivos estabelecidos no Plano de Desenvolvimento de Software.

**Finalizar Fase**

Neste subfluxo, o gerente de projeto encerra a fase garantindo que:

- Todos os principais problemas da iteração anterior foram solucionados;
- O estado de todos os artefatos é conhecido;
- Os artefatos exigidos foram distribuídos aos envolvidos;

Uma avaliação final do status é realizada durante a revisão dos *milestones* do ciclo de vida, na qual os artefatos da fase são revistos e, se o estado do projeto estiver satisfatório, uma aprovação é fornecida para passar para a próxima fase.

**Monitorar e Controlar o Projeto**

Este subfluxo capta o trabalho diário e contínuo do gerente de projeto, abordando:

- O tratamento das solicitações de mudança aprovadas pelo gerente de configuração e a sua programação para iterações atuais ou futuras;
- O monitoramento contínuo do projeto em termos de riscos ativos e medições objetivas do andamento do projeto e da qualidade;

- A produção de um relatório regular do status do projeto, na "Avaliação de Status";
- O tratamento de questões e problemas quando eles são descobertos, através do monitoramento do status do projeto ou por outros meios, e direcionamento para a solução de acordo com o Plano de Resolução de Problemas.

O gerente de projeto necessitará de uma combinação de habilidades organizacionais, analíticas, bem como de planejamento, comunicação, gerenciamento de tempo e triagem para essa parte da disciplina. A Figura 4-17 mostra o subfluxo com as atividades que foram alteradas.

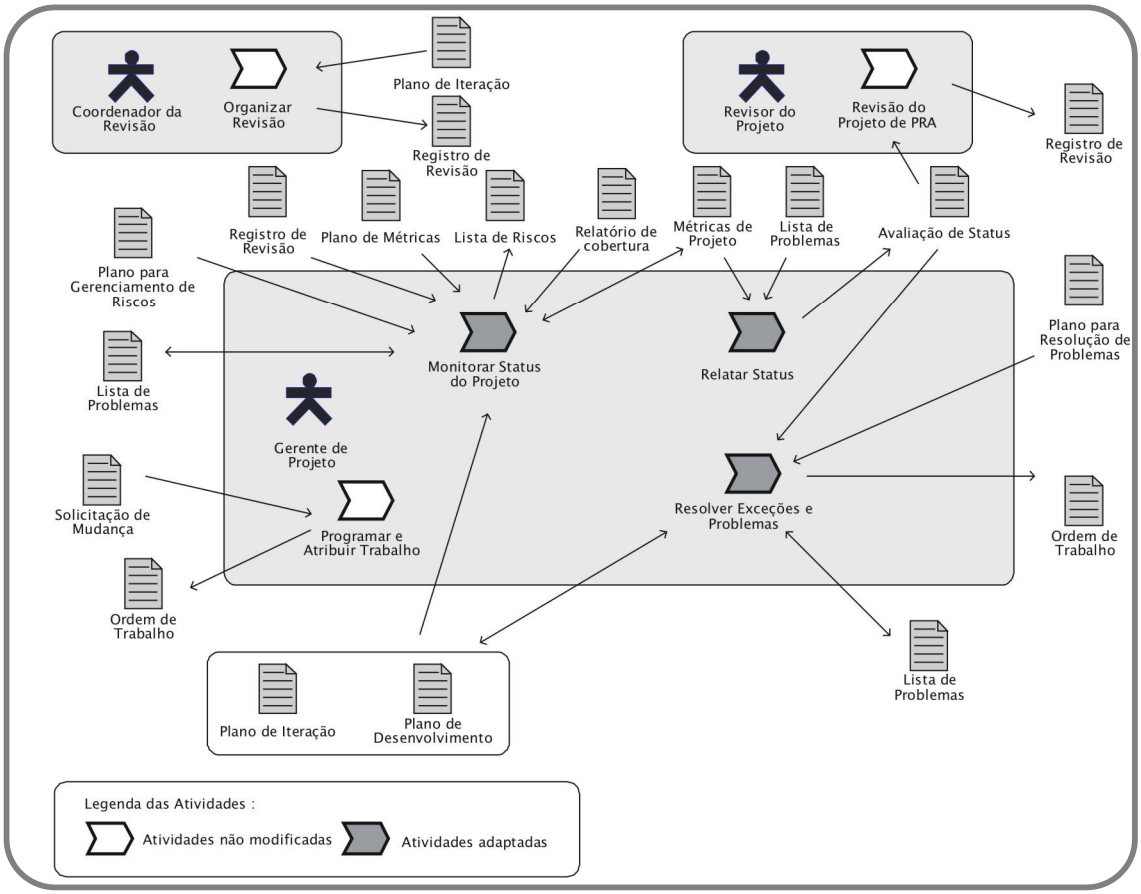



**Figura 4-17:** Subfluxo Monitorar e Controlar o Projeto

A seguir, serão descritas as atividades do subfluxo "Monitorar e Controlar o Projeto" que foram adaptadas para o *Code Coverage Process.* Estas atividades são: (a) Monitorar Status do Projeto; (b) Relatar Status e (c) Resolver Exceções e Problemas.

**a) Monitorar Status do Projeto**

O gerente de projeto captura as métricas referentes ao andamento do projeto e a qualidade do produto. Os métodos a serem usados para capturar essas métricas são descritos no Plano de Métricas do projeto.

Geralmente, os membros da equipe do projeto enviam relatórios de andamento periódicos ao gerente de projeto, fornecendo as seguintes informações:

- Esforço estimado para concluir cada pacote de trabalho pelo qual eles são responsáveis;
- Tarefas concluídas;
- Produtos liberados publicados;
- Problemas que requerem atenção da gerência (em Registros de Revisão, por exemplo). O gerente de projeto pode registrar alguns ou todos esses problemas na Lista de Problemas para verificá-los e rastreá-los posteriormente.

Com a adoção do *Code Coverage Process,* o gerente do projeto, com os dados provenientes da cobertura do código extraído pelos relatórios de cobertura irá avaliar a qualidade do código e dos testes de unidade desenvolvidos.

O gerente de projeto fará o controle e gerenciamento dos resultados coletados nos Relatórios de Cobertura, como também, fará o estudo e a análise para definição das métricas e limiares aceitáveis de cobertura.

**b) Relatar Status**

Esta atividade irá fornecer atualizações regulares ao status do projeto para fins de acompanhamento do projeto. No caso da organização de desenvolvimento de software prestar serviços para outra organização ou no caso de departamentos e/ou setores da própria empresa, se fazem necessário relatar os dados referentes às métricas de coberturas alcançadas no projeto.

**c) Resolver Exceções e Problemas**

Esta atividade irá iniciar as ações corretivas apropriadas para problemas e exceções que surgem no projeto.

Uma “Avaliação de Status” é criada na atividade “Relatar Status”. Cada uma dessas avaliações identifica problemas ou "maus resultados". Os problemas podem ser de projeto (ex.: desvios de planos, pessoal), de produto (defeitos, ambigüidade de requisitos, deficiências de tecnologia) ou riscos percebidos. A “Avaliação de Status” também identifica as exceções que ocorrem. É possível considerar exceções como problemas que constituem barreiras ao progresso do projeto (ex.: disponibilidade de equipamentos, disponibilidade dos indivíduos chave em responder a perguntas, dificuldade de tomar decisões).

Ações e medidas corretivas referentes à cobertura de código são descritas como a segue:

- O gerente de projeto deve monitorar os resultados das métricas de cobertura e conduzir o projeto de forma que esses resultados sejam mantidos dentro dos níveis definidos. Uma vez que os resultados não estejam dentro dos níveis aceitáveis, ações corretivas ou melhorias deverão ser tomadas;
- Tendo sido definido o nível aceitável de cobertura de cada métrica, e tomando como base a análise dos resultados da cobertura do código, deve-se tomar atitudes em relação ao que deve ser feito para os testes em que o nível de cobertura não foi atingido: proveniente da falta de casos de testes ou se os mesmos não executam corretamente o código. No primeiro caso, deve-se verificar no relatório se não existe nenhum caso de teste que execute aquele trecho do código, constatado isto, deve-se encaminhar para o implementador a incumbência de criar o caso de teste de unidade que esteja faltando; No segundo caso, observando que existem casos de testes de unidade, mas os mesmos não cobrem o trecho, é encaminhada para o implementador a incumbência de modificar os casos de teste de unidade para que os mesmos possam cobrir o trecho que não esteja sendo executado.

O gerente de qualidade deve acompanhar e monitorar a evolução do projeto, a partir dos resultados de cobertura, pois esta métrica irá proporcionar uma melhoria no produto e no processo. Caso não sejam atingidos os resultados esperados da cobertura, é relevante que seja feita uma reunião para se discutir a causa e até mesmo para analisar se o problema

é devido à falta de tempo hábil para realização das tarefas de desenvolvimento ou a necessidade de se ter treinamentos em relação ao desenvolvimento do código e dos testes de unidade.

## 4.3 Considerações Finais

Neste capítulo, foi apresentada a extensão de quatro fluxos do RUP, Implementação; Gerenciamento de Configuração e Mudança; Testes e Gerenciamento de Projeto, para atender mais apropriadamente *Code Coverage Process*. A principal característica destas extensões foi a adaptação e inclusão de atividades e artefatos relacionados com cobertura de código.

O próximo capítulo relata-se a experiência resultante do estudo de caso em torno da utilização da técnica de cobertura de código apresentada nesse trabalho, no projeto em uma organização real, sendo utilizado no desenvolvimento de software em sistemas embarcados. Em seguida, é descrita a análise feita a respeito do impacto da utilização do processo proposto, através de experimentos. Por fim, é apresentada uma análise crítica do processo proposto, com o objetivo de validar a nossa proposta, no sentido de atender os objetivos estabelecidos.

# 5 Estudo de Caso e Análise Crítica do Processo

Este capítulo apresenta o estudo de caso realizado numa organização de desenvolvimento de software para celulares. O estudo serviu de base para a elaboração do *Code Coverage Process*, compreendendo a utilização da técnica de cobertura de código inserida dentro de um processo de desenvolvimento de software.

Neste capítulo serão apresentados dois experimentos, o primeiro é a utilização da técnica de cobertura de código para seleção de casos de testes, utilizando dois conjuntos de casos testes para aplicação no domínio de *Messaging*; o segundo experimento é a análise do impacto da utilização da cobertura de código inserido no fluxo de Testes. Por fim, é apresentada, a análise crítica do *Code Coverage Process* realizada por um grupo de profissionais experientes na área de processo e testes de software.

## 5.1 Visão Geral

Como o material utilizado para o estudo de caso é propriedade intelectual da empresa que utilizou o processo proposto, não será possível apresentar maiores detalhes sobre os produtos, processos, padrão de documentação, etc. No entanto, a maioria das lições e experiências aprendidas constitui material não confidencial, e portanto, serão apresentadas neste trabalho.

A seguir, a Seção 5.2 apresenta o estudo de caso desenvolvido, detalhando todos os passos realizados, bem como os resultados obtidos. Na Seção 5.3, a aplicação de análise de cobertura de código no fluxo de testes. Na Seção 5.4, os experimentos realizados serão apresentados. Na Seção 5.5 são apresentadas as análises críticas realizadas para refinar e validar o processo proposto.

## 5.2 Estudo de Caso

Esta seção apresenta o estudo de caso desenvolvido para a elaboração do *Code Coverage Process*, no qual foi incorporada a técnica de cobertura de código ao processo de desenvolvimento de software durante a codificação de uma aplicação.

Inicialmente, foi preciso realizar estudos e análises para verificar a melhor forma de inserir as atividades dentro do fluxo da equipe de desenvolvimento de código, a fim de não ocasionar impactos negativos e prejudicar o andamento do desenvolvimento atual.

### 5.2.1 Ambiente do estudo de caso

A equipe de desenvolvimento, com a qual foi trabalhado o estudo de caso, tinha as seguintes características: realizava-se um procedimento de criação dos testes de unidade para código desenvolvido, mas em contrapartida não realizava análise da efetividade dos testes de unidade gerados. A única métrica que o time de desenvolvimento coletava era a quantidade de testes que passaram e falharam, apesar de não haver nenhuma garantia de que os testes de unidade passados cobriam realmente o código ou que alguns deles pudessem ser redundantes.

Durante a pesquisa das técnicas de análise de cobertura de código, foi realizada uma pesquisa de possíveis ferramentas compatíveis ao ambiente de desenvolvimento do projeto e principalmente, as que poderiam gerar dados detalhados a respeito da cobertura do código

com base dos testes de unidade realizados. Essa pesquisa foi necessária devido a enorme quantidade de ferramentas existentes no mercado, logo era necessário selecionar uma ferramenta que atendesse aos requisitos que foram levantados para a análise, a fim de obter um maior número de métricas de cobertura. As subseções a seguir descrevem o procedimento para a escolha da ferramenta de cobertura para o nosso estudo.

**Tabela 5-1: Listas de f**erramentas de análise de cobertura de código

| Ferramentas | Fornecedores |
|---|---|
| AQTime | CodeWork |
| BullseyeCoverage | Bullseye Testing Technology |
| Cantata C++ | IPL |
| Dynamic Code Coverage | Dynamic Memory Solutions |
| CTC++ | Testwell Oy (Ltd) |
| C++ Test Coverage Tool | Semantic Designs, Inc. |
| TestWorks/Coverage TCAT C/C++ | Software Research, Inc. |
| Panorama C/C++ | International Software Automation, Inc. (ISA) |
| IBM Rational PureCoverage | IBM |
| TAU/Logiscope TestChecker | Telelogic AB |
| LDRA Testbed | LDRA |
| GCT – a C coverage Tool | Testing.com |
| gcov – a Test Coverage Program | GNU |
| JCover | CodeWork |
| Clover Code Coverage for Java | Cenqua |
| JBlanket | University of Hawaii |
| GroboCoverage | Open-Source |
| Gretel | University of Oregon |
| EMMA | Open-Source |
| Insect | Georgia Institute of Technology |

### 5.2.2 Ferramenta de Cobertura

É de suma importância escolher uma ferramenta que atenda perfeitamente a organização e conseqüentemente o projeto que se deseja avaliar, pois não adianta ter um processo eficaz e não possuir ferramenta que possa dar suporte ao processo.

Como existem várias ferramentas de análise de cobertura de código no mercado, foi preciso realizar um estudo e um levantamento inicial dos requisitos necessários para atender a organização e o projeto como um todo. O apêndice E apresenta exemplos de ferramentas de cobertura de código.

### 5.2.3 Levantamento dos Requisitos para ferramenta

Inicialmente realizou-se um trabalho de levantamento dos principais requisitos organizacionais, funcionais, de ambiente de desenvolvimento, de suporte e de usabilidade.

O processo de selecionar as ferramentas iniciou-se através da criação de uma lista de requisitos técnicos, identificados para o processo de avaliação. Foi elaborada uma lista de ferramentas de cobertura de código, baseada em artigos e sites encontrados na internet, como também por indicação da organização. Em seguida, foi elaborado um subconjunto dos requisitos essenciais aplicados para seleção de algumas ferramentas que poderiam atender ao projeto:

- Requisito Funcional: fornecer métricas de cobertura para teste de caixa branca e/ou caixa preta. Provê relatórios que informem se o código está ou não coberto; Fornecer métricas efetivas, para diferentes métodos, calculando a cobertura;
- Requisito de Ambiente de Desenvolvimento: fornecer suporte à linguagem em que está sendo desenvolvido o código e ao ambiente (plataforma) utilizado pela equipe de desenvolvimento e de testes;
- Requisito de Suporte: o fornecedor da ferramenta de cobertura deverá oferecer suporte seja através de e-mail, telefone ou *chat*, incluindo manual da ferramenta;
- Requisito de Usabilidade: facilidade de instalação, instrumentação e geração de relatório.

Baseado nos requisitos estabelecidos foi realizado um estudo das ferramentas de cobertura existentes. Este trabalho foi realizado de forma organizada e bem planejada, devido à existência de diversas ferramentas de cobertura no mercado para objetivos distintos.

### 5.2.4 Seleção da Ferramenta

Para selecionar ferramenta que atendesse os objetivos da organização, foram realizados ciclos de análises e comparações das ferramentas de análise de cobertura de código pré-selecionadas. O objetivo principal para a seleção foi a escolha de uma ferramenta que pudesse operar tanto com o código, como com simuladores, emuladores e diretamente no próprio dispositivo móvel. Houve interações com alguns fabricantes ferramentas, a fim de solucionar algumas duvidas a respeito do que a ferramenta pode oferecer.

No primeiro ciclo, foram avaliadas as ferramentas para análise de cobertura de código com base nos requisitos determinados. Nesta ocasião, foi atribuída uma pontuação para cada requisito, em que o de maior prioridade tinha o maior peso. Em seguida, foi feito o calculo da pontuação de cada ferramenta, a fim de fazer uma segunda listagem de ferramentas que tiveram a maior pontuação e que atenderam o maior número de requisitos obrigatórios.

No segundo ciclo, as ferramentas de cobertura selecionadas na segunda listagem foram inseridas no ambiente de desenvolvimento, sendo selecionada uma *feature* e utilizando um simulador de dispositivo móvel para a realização dos testes e execução das ferramentas analisadas, a fim de verificar a performance e o desempenho das ferramentas no ambiente de desenvolvimento de software da organização. Para a realização deste ultimo ciclo, foi necessário ter as ferramentas de análise de cobertura de código em pleno funcionamento no ambiente de desenvolvimento de software.

Os nomes das ferramentas selecionadas não podem ser divulgados, devido a se tratar de informações confidenciais da organização. Os critérios da seleção foi o atendimento aos requisitos funcionais, de ambiente de desenvolvimento, suporte e de usabilidade que foram solicitados.

### 5.2.5 Aplicação da ferramenta de cobertura selecionada

A ferramenta selecionada foi inserida no ambiente de desenvolvimento, em que seria realizada a execução de testes reais, e seria observado o seu comportamento na geração das métricas de cobertura. Nesta ocasião, a ferramenta de cobertura passou por um projeto piloto, antes de ser efetuada a compra da sua licença.

Terminado o processo da seleção da ferramenta, o próximo passo foi aplicá-la no ambiente real de desenvolvimento, onde puderam ser coletadas métricas. Foi montada toda uma estrutura dentro do ambiente da equipe de desenvolvimento. Foi necessário conhecer todo o ambiente de desenvolvimento e interagir diretamente com desenvolvedores durante o processo. Em seguida, iniciou-se um projeto piloto, no qual foram selecionadas duas *feature*s e foi dada a continuidade à utilização do simulador do dispositivo móvel com o objetivo de verificar a eficácia da ferramenta de cobertura selecionada e principalmente colher informações referentes à inserção e adaptações de atividades para o uso da cobertura de código, e o impacto no processo de desenvolvimento com a inserção da técnica de cobertura de código.

A partir do projeto piloto foi possível analisar e registrar os benefícios da técnica de cobertura de código dentro do processo de desenvolvimento de software, como:

- Melhoria no desenvolvimento do código e dos testes;
- Identificação de casos de teste redundantes;
- Sugestão de criação de casos de teste.

Na Figura 5-1 pode-se observar como ocorre o fluxo da utilização da ferramenta de análise de cobertura de código. A ferramenta recebe como entrada o código fonte e gera a instrumentação. Em seguida, o código instrumentado é compilado, gerando uma aplicação, para que se possa executar os casos de testes. Por fim, é gerado, na ferramenta, o relatório com as métricas da análise de cobertura de código.

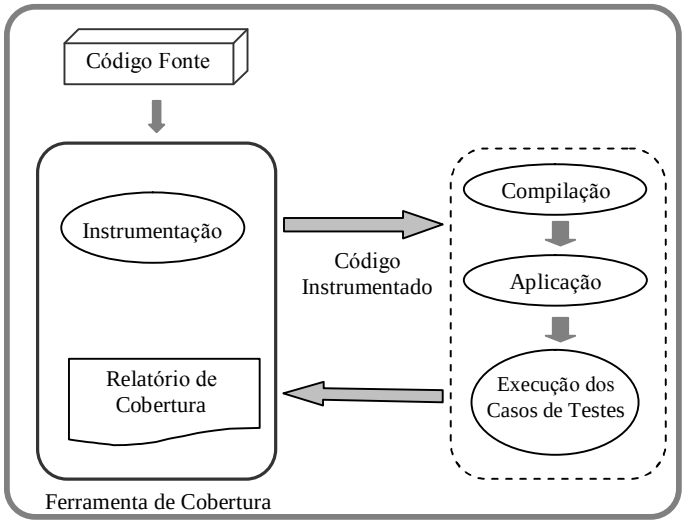



**Figura 5-1:** Fluxo de Execução

### 5.2.6 Dificuldades Encontradas e Lições Apreendidas

Alguns problemas foram encontrados no decorrer da realização do estudo de caso. A seguir serão descritas quais dificuldades foram encontradas e as soluções adotadas:

Houve problemas com arquivos que eram instrumentados, isto fez com que o tamanho do componente, cujo código foi instrumentado, aumentasse em comparação ao componente original. Esse problema ocorreu devido a uma forma errada da utilização da ferramenta de análise de cobertura de código, pois não tinha-se configurado a ferramenta de forma que instrumentasse apenas o código selecionado.

Ocorreram problemas de compreensão a respeito do ambiente de desenvolvimento, em relação à estrutura de armazenamento e à criação de *build*s. Foi preciso realizar treinamentos, bem como ter um suporte de alguns desenvolvedores. Devido a esse problema, houve certa dependência da pesquisa a equipe de desenvolvimento no começo dos trabalhos, pois os mesmos tinham um conhecimento especifico do ambiente de desenvolvimento.

As atividades de instrumentação do código e geração de relatório de cobertura tiveram que ser refeitos, pois o processo de desenvolvimento de software realizava a criação de testes de unidade paralelamente com o desenvolvimento do código e como o código mudava frequentemente, havia a necessidade de se alterar os testes e consequentemente fazer uma nova análise de cobertura de código. A solução encontrada, para este problema, foi realizar a análise de cobertura após a estabilização do código e dos testes de unidade.

Houve, também, dificuldades com a utilização e compreensão do simulador do dispositivo móvel, pois apesar do mesmo ter sido desenvolvido dentro da organização, os próprios desenvolvedores tinham problemas em utilizá-la, devido à sua instabilidade e limitações.

Como a ferramenta selecionada, nas primeiras fases do trabalho, produz uma quantidade enorme de relatórios de análise de cobertura de código e outros tipos de relatórios, foi preciso uma dedicação exclusiva à atividade de estudo dos relatórios e realizar um número maior de experimentos e testes, com intuito de entender os relatórios de cobertura de código gerados pela ferramenta, bem como escolher quais tipos de relatórios

serviria para o projeto que estava sendo analisado, isso um problema devido ao fato que não tínhamos muito tempo para realiza o estudo.

### 5.2.7 Resultados Obtidos

Durante o período do projeto piloto, várias pessoas foram envolvidas na melhoria da qualidade do software do projeto, procurando melhores resultados em relação à cobertura do código desenvolvido. Todo o projeto e envolvimento com a equipe de desenvolvimento foram considerados pela organização como um projeto real, tendo metas e prazos a serem seguidos, e reuniões semanais realizadas com intuito de informar o status do projeto.

Algumas dificuldades foram encontradas e algumas soluções foram apresentadas no processo de seleção da ferramenta de cobertura e de desenvolvimento do código. Inicialmente, foram usadas as métricas de aceitação de cobertura proposta pela ferramenta, porque não havia um histórico das execuções que permitisse criar um perfil das métricas de cobertura para os projetos a serem analisados.

Com a ferramenta de cobertura selecionada foi possível alcançar um código bem coberto, que agregasse mais qualidade, menos tempo de depuração e mais produtividade para a codificação e processo de teste. Através de conversas com alguns desenvolvedores puderam destacar os seguintes benefícios com uso da análise de cobertura de código descritos a seguir:

- Com os relatórios de cobertura, pode-se descobrir os possíveis cenários para a criação dos casos de testes de unidade, bem como saber se existem casos de testes redundantes;
- Na identificação de um determinado trecho de código, que não está sendo coberto, pode-se verificar se a razão desta falta é devido ao teste não estar passando pelo ponto de verificação ou se não existem testes;
- Quando existem vários módulos a serem testados, tendo os módulos da camada superior sido testados, pode-se reduzir a quantidade de testes da camada inferior, resultando na redução dos esforços sem perder a cobertura dos casos de testes, evitando a redundância;
- O processo de cobertura de código veio para ajudar a resolver esses problemas. Com alguns frameworks e ferramentas, é possível fazer essa

análise e verificar o quanto o código está sendo testado. Desta forma é possível saber qual pedaço do código não está sendo testado pelos testes unitários;

No projeto, surgiu a necessidade da criação de um *guideline*, que orientaria o desenvolvedor no uso da ferramenta de cobertura no processo. Neste guia, foram inseridos os passos para integração da ferramenta ao ambiente de desenvolvimento.

Com as informações colhidas no projeto piloto, pode-se elaborar uma proposta de um processo de desenvolvimento de software, com atividades, responsáveis e artefatos gerados para implementação do código, tendo como suporte a análise de cobertura de código, no qual foi empregado o RUP como base e fonte inspiradora, ou seja, foi utilizado um processo genérico, bastante conhecido e consolidado.

Após a conclusão deste projeto piloto, foram divulgados os resultados obtidos com a seleção da ferramenta de análise de cobertura de código e com o que a análise poderia beneficiar no processo de desenvolvimento e de testes, através de algumas apresentações para equipes do Brasil e do exterior, havendo, em algumas situações, comparações entre o objetivo e resultados da pesquisa a trabalhos realizados por outras equipes no mundo.

A metodologia utilizada para selecionar ferramenta de cobertura foi bastante elogiada pela organização. Através dos resultados obtidos, uma das equipes de desenvolvimento da organização efetuou a aquisição da ferramenta e começou a incorporar a análise de cobertura de código dentro do seu processo de desenvolvimento.

## 5.3 Utilização de Cobertura de Código no Fluxo de Testes

Para que se pudesse conseguir trabalhar no fluxo de testes com a análise de cobertura de código, foi necessário ter uma ferramenta de análise de cobertura de código que pudesse capturar informações de dentro da aplicação do sistema embarcado, utilizando casos de testes de caixa preta. Como na seleção da ferramenta de análise de cobertura de código, um dos requisitos para escolha era uma ferramenta que conseguisse colher métricas dentro de um sistema embarcado, logo se utilizou a mesma ferramenta do projeto piloto anterior.

Para este novo projeto piloto, teve-se como primeiro desafio a geração de uma *build* instrumentada para que pudesse ser instalada dentro do dispositivo móvel. Logo este projeto não necessitou da utilização de simuladores ou emuladores. Havia duas grandes

preocupações neste projeto piloto: a primeira era o fator da limitação de memória que existe no sistema embarcado, pois se tinha o receio que com o *build* instrumentada a aplicação não conseguisse ser inserida no aparelho; e a segunda preocupação era como seria feita a captura das informações referentes a execução dos casos de testes para em seguida gerar as métricas de cobertura.

A captura das informações veio através do arquivo de histórico da execução obtida através da implementação de uma função no código da aplicação, a qual imprimia numa tela do tipo hiper terminal do Windows os dados referentes ao arquivo de histórico. Para isso era necessário que o aparelho estivesse conectado ao computador através de um cabo USB.

Em relação ao tamanho do *build* não houve problemas, pois a mesma permaneceu com o mesmo tamanho, mesmo estando instrumentada. No entanto, surgiu uma nova preocupação que foi se o tempo de geração da *build* seria afetado em virtude do acréscimo de informações nos códigos instrumentados, mas com alguns experimentos conseguiu-se mostrar que não houve alteração no tempo da compilação.

Por fim, realizou-se a execução de todo processo de testes desde a geração da *build* até a coleta das métricas no relatório de cobertura. O trabalho foi realizado com uma *feature*, que tem como funcionalidade à comunicação através de MSN no próprio celular.

Inicialmente foi preciso, junto à equipe de desenvolvimento, entender o código da *feature*, com intuito de localizar os códigos que seriam instrumentados e onde seria inserida a função de captura da informação da execução dos casos de testes. Em seguida gerou-se a *build* instrumentada, sendo a mesma encaminhada para o time de testes responsável por aquela *feature*, a qual se localiza nos EUA. O testador realizou todos os casos de testes funcionais associados a *feature* e em seguida encaminhou os arquivos com a informação da execução de cada caso de teste. No momento da recepção dos arquivos, pôde-se gerar o relatório de análise de cobertura de código referente a *feature*.

Com a análise dos resultados através do relatório obtido, foi verificado que a cobertura da *feature* tem uma porcentagem muito baixa, chegando a alguns pontos a atingir menos de 10 por cento, a partir disto foi realizada uma investigação a fim de descobrir os motivos. A primeira constatação foi que alguns pontos no código só poderiam ser testados

através de teste de unidade, pois necessitariam testes específicos, como por exemplo, testar controle de fluxo do código. A segunda comprovação foi que como a *feature* testada esta integrada com outras *feature*s, houve trechos no código que não teriam como serem testados através dos casos de testes da suíte utilizada, pois que pertenciam à outra *feature*. Conclui-se que se deve ter cuidado no momento de analisar os relatórios de cobertura de código e também quando for determinar o limiar de aceitação da cobertura de código.

A utilização da análise de cobertura no fluxo de teste tendo gerado um *build* instrumentado e instalado diretamente no aparelho celular, sistema embarcado, foi um sucesso, sendo comemorado por todo time envolvido e elogiado por toda alta direção.

## 5.4 Impacto da utilização da Análise de Cobertura de Código

Nesta seção, serão apresentados dois experimentos realizados no trabalho. O 1º experimento trata da utilização da análise de cobertura de código na avaliação e seleção de casos de testes. O 2º experimento analisa o impacto no tempo da execução dos casos de testes caixa preta no fluxo de testes.

### 5.4.1 Primeiro Experimento – Avaliação e Seleção de Casos de Testes

Neste experimento, pôde-se não só validar e ajustar o trabalho, a meta específica era comparar os casos de teste gerados automaticamente com os testes gerados manualmente a partir dos mesmos requisitos, através da análise de cobertura de código, utilizando o processo proposto, com intuito de validá-lo e ajusta-lo se necessário.

A *feature* utilizada para este experimento consiste em um conjunto de funcionalidades que se relacionam com o armazenamento de mensagens favoritas em uma pasta específica. Ao ler uma mensagem na caixa de entrada, o usuário pode salvar essa mensagem numa pasta.

A comparação tinha o objetivo de fornecer as seguintes informações: (1) Métricas de cobertura de código, no sentido de analisar qual dos dois conjuntos (manual ou automático) de casos de teste cobriam melhor o código da *feature*. (2) Números de casos de testes redundantes, identificando qual dos conjuntos tinha menos casos de testes redundantes.

Dando continuidade ao procedimento da geração da *build* instrumentada com a *feature* foi realizado o procedimento descrito na Seção 5.2. Como dados de entrada para

execução dos testes, tem-se a seguinte quantidade: os casos de testes gerados automaticamente foram 5 (cinco) e os que foram gerados manualmente eram 6 (seis). Analisando os testes gerados, conclui-se que, mesmo a quantidade sendo diferente, o objetivo dos testes era equivalente.

Com os resultados obtidos através dos relatórios de cobertura foi realizada a análise, mostrada na Figura 5-2, na qual apresenta-se a média percentual de duas métricas de cobertura (*statement coverage* e *branch coverage*) referentes à execução dos casos de testes gerados de forma automática e manual.

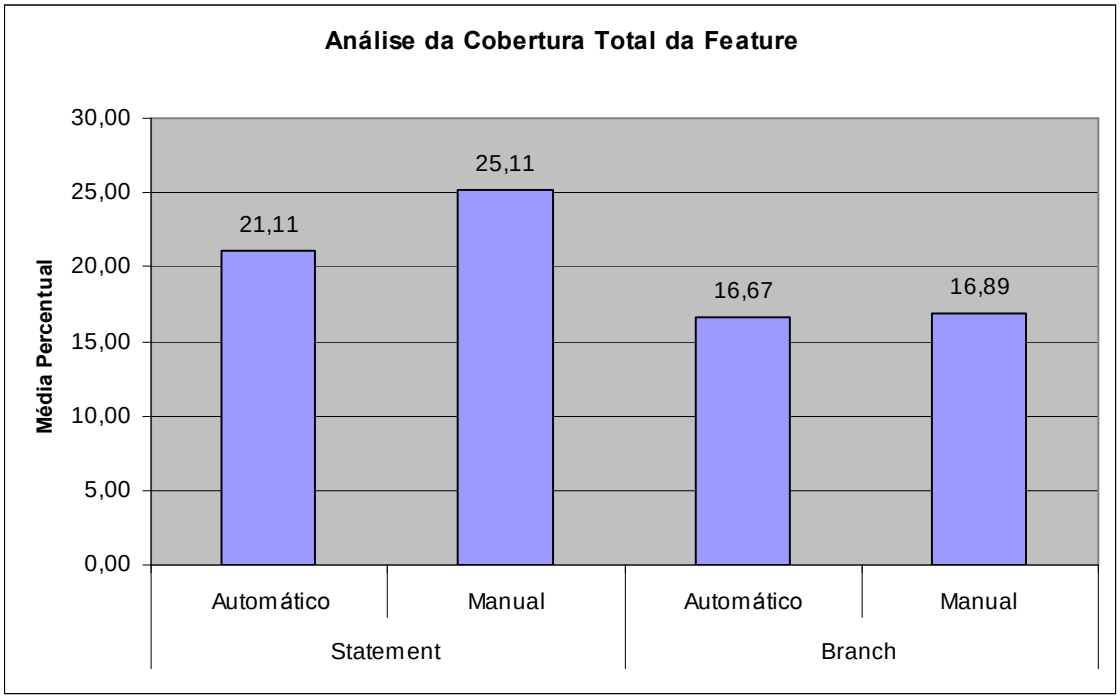



**Figura 5-2:** Métricas de Cobertura de Código proveniente da Execução dos Casos de Testes

Com os dados apresentados na Figura 5-2 pode-se verificar que os testes manuais tiveram um pequeno aumento na cobertura, logo a forma que está estruturada os casos de testes, faz com que se possa testar mais pontos do código.

Na Figura 5-3 tem-se a média percentual de casos de testes redundantes por conjunto gerado (automaticamente ou manualmente).

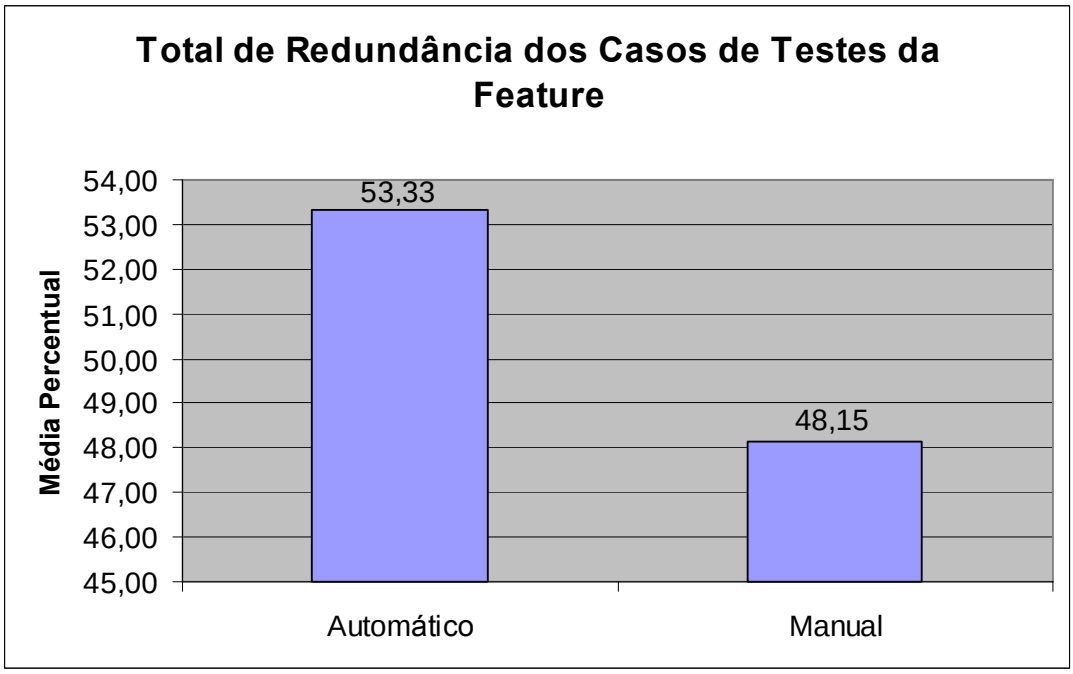



**Figura 5-3:** Total de Redundância dos Casos de Testes

Com os resultados da redundância apresentados na Figura 5-3, conclui-se que, o percentual de casos de testes redundantes do conjunto gerado automaticamente foi maior que o gerado manualmente, logo se deve revisar a estrutura e forma como estão os casos de testes automáticos para identificar o que ocasionou este fato.

A análise comparativa entre os dois conjuntos de casos de teste, utilizando a análise de cobertura de código teve êxito, mas vale salientar que dificilmente a cobertura dará 100%, devido ao fato que no código da *feature* existirem interações com outras *feature*s, que executando apenas um determinado conjunto de casos de testes que não envolva a interação, a cobertura será baixa.

### 5.4.2 Segundo Experimento – Impacto na Execução dos Casos de Testes

Foi realizado um trabalho de análise do impacto referente ao uso da técnica de cobertura de código na execução dos casos de testes, sobre uma e utilizando novos passos no procedimento da execução dos casos de testes, o que poderia ocasionar um impacto negativo no processo existente.

As novas tarefas inseridas antes da execução dos testes foram:

- Conectar o aparelho ao computador com o cabo USB;

- Abrir o hiper terminal para captura das informações da execução dos casos de testes.

Na execução dos testes:

- Verificação se o hiper terminal está registrando a captura das informações

Na finalização da execução dos casos de testes:

- Após finalizar a execução do caso de teste, é necessário salvar o arquivo que contem a informação da execução do caso de teste. Deve-se armazenar um arquivo por caso de teste, para que se possa verificar a redundância dos casos de testes.

A seguir serão apresentados gráficos que comparam a execução de casos de testes utilizando a *build* com e sem instrumentação. O experimento consistiu dos seguintes passos:

- Foi selecionada uma aplicação do celular;
- Foi selecionada uma suíte de casos de teste correspondente à aplicação a ser testada;
- Foram selecionados apenas 19 casos de teste da suíte, pois havia problemas de ambiente relacionado à operadora de celular e limitações do aparelho, pois o mesmo foi descontinuado o seu desenvolvimento;
- Cada caso de teste foi executado 10 vezes, com o intuito de se ter uma média do tempo de execução;
- Foi utilizada uma ferramenta para registrar o tempo e os dados foram armazenados numa planilha.

Pode-se constatar, na Figura 5-4, que houve um pequeno aumento no tempo para realizar o setup, ou seja, a realização da condição inicial para executar o caso de teste. Devido ao fato de que toda vez se faz necessário abrir o hiper terminal e verificar se o aparelho está se conectando ao computador.

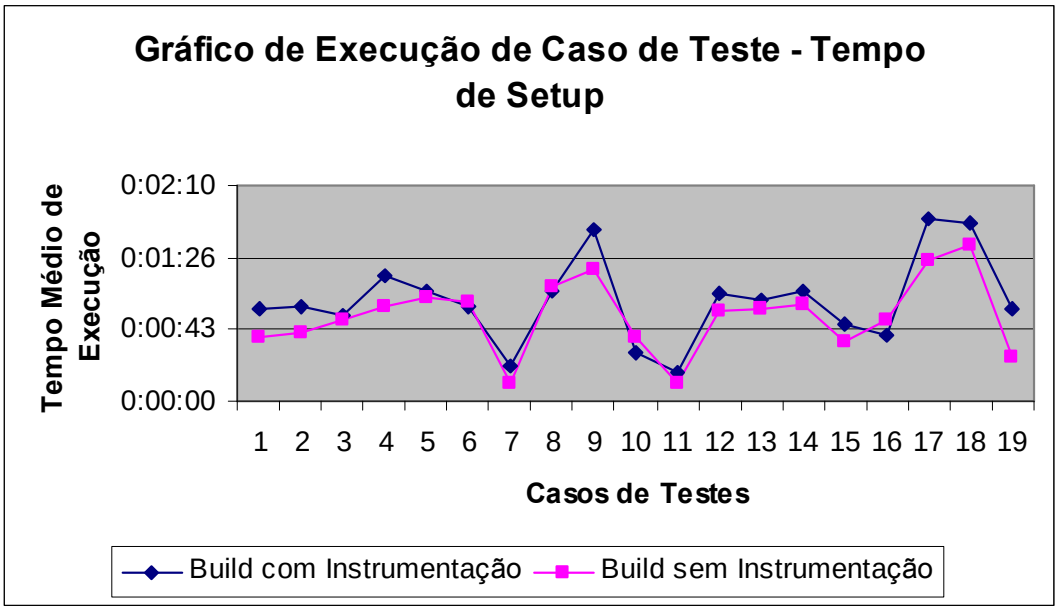



**Figura 5-4:** Tempo médio da configuração dos casos de testes para execução

Pode-se constatar, na Figura 5-5, que não houve aumento significativo no tempo quando estava sendo executando os casos de testes com a *build* instrumentada, logo a etapa de execução não sofrerá impacto. Houve apenas um ponto isolado que com instrumentação sofreu um pequeno aumento, mas nada que prejudicasse o andamento.

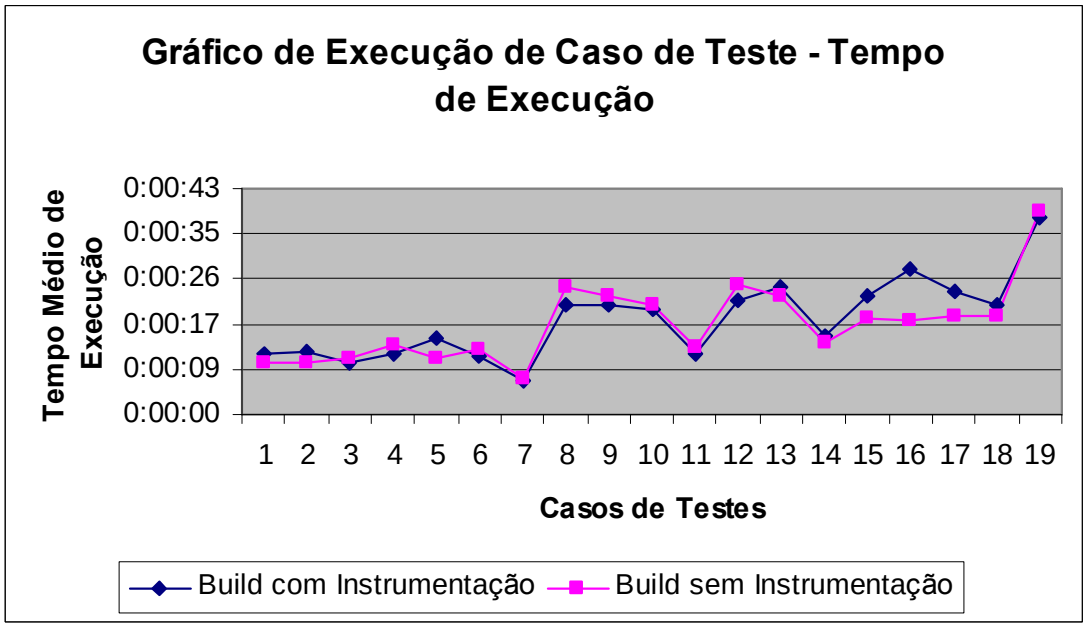



**Figura 5-5:** Tempo médio da execução dos casos de testes

Em relação à finalização da execução do caso, no qual foi considerado que o testador deveria deixar o aparelho da mesma forma que iniciou o caso de teste, ocorreu um aumento no tempo, pois foi preciso salvar o arquivo de informação e verificar se não houve dano ao arquivo.

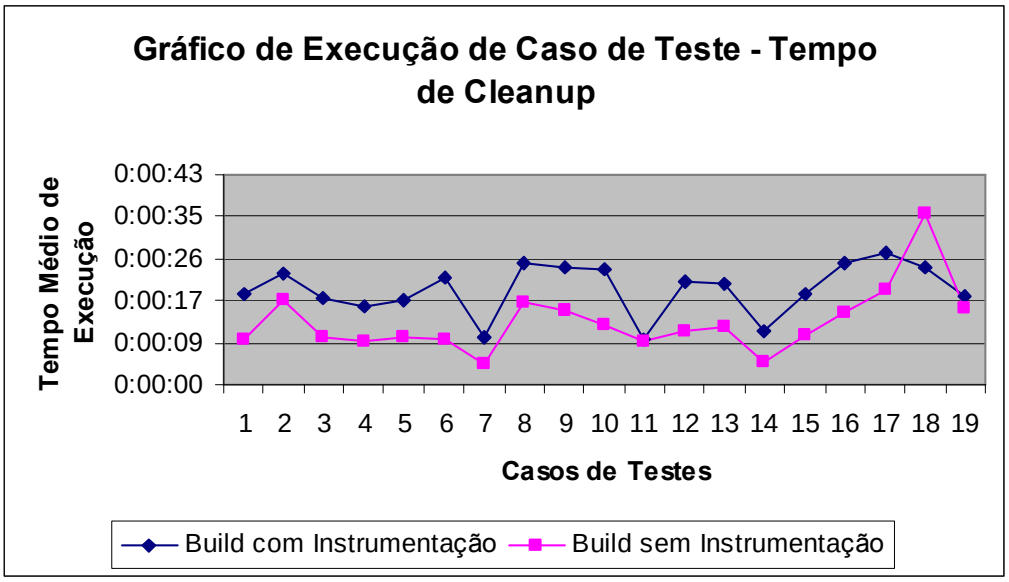



**Figura 5-6:** Tempo médio do *cleanup* dos casos de testes após execução

Pode-se observar, nas figuras 5-7 e 5-8, que a execução referente à toda etapa (setup, execução e cleanup) da execução ficou equiparado.

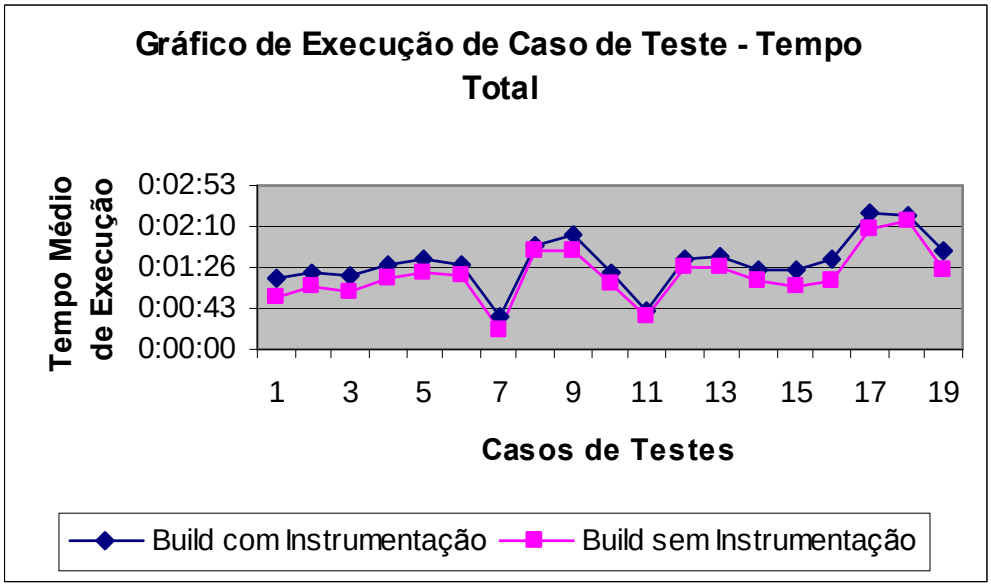



**Figura 5-7:** Tempo médio do ciclo completo da execução dos casos de testes

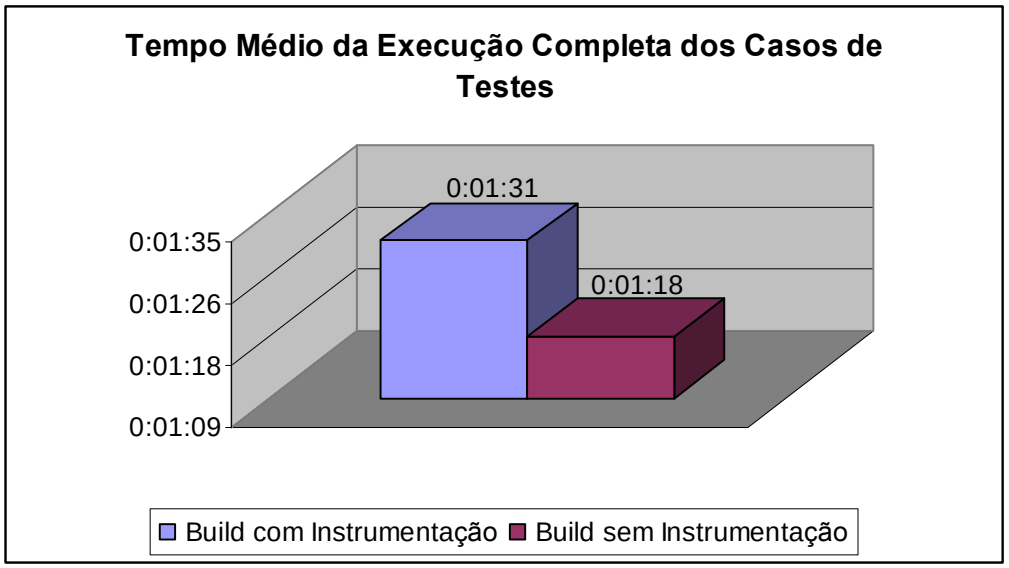



**Figura 5-8:** Tempo médio do ciclo completo da execução de todos os casos de testes

Um problema constatado foi que quando acontece alguma falha na execução dos casos de teste, o testador deverá iniciar a execução desde o início e não a partir do ponto que ocorreu a falha, pois estavam sendo registrados todos os passos no arquivo para depois gerar o relatório de cobertura e quando dava um problema na captação do log ou na execução dos testes, a informação referente à cobertura é impactada.

Como neste experimento foi utilizado um testador que já possuía conhecimento da forma como de obter o log com os dados histórico da execução dos casos de testes e também conhecimento da *feature* testada, optou-se utilizar um testador que não possuía estes conhecimentos, com o intuito de verificar o impacto.

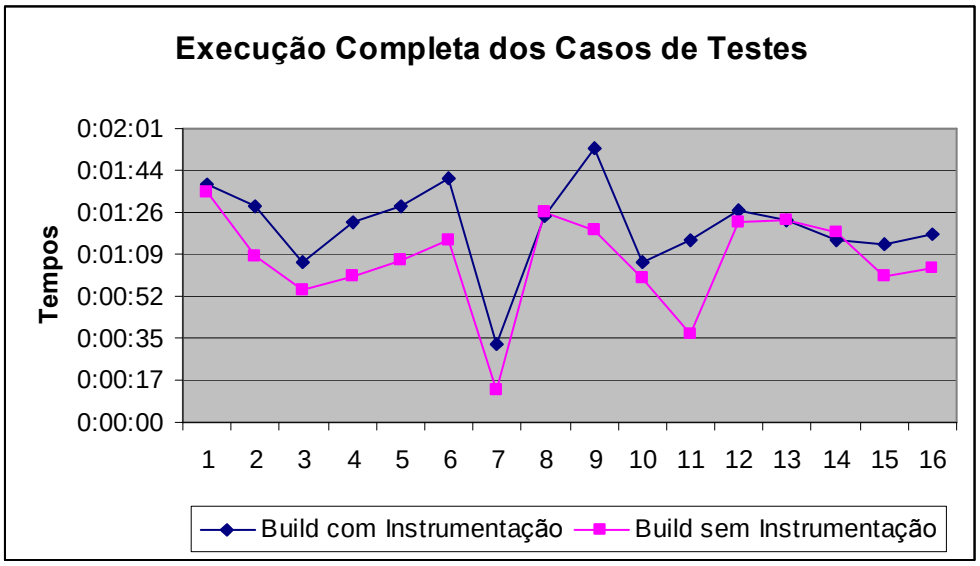



**Figura 5-9:** Tempo médio completo da execução dos casos de testes realizado por um testador experiente

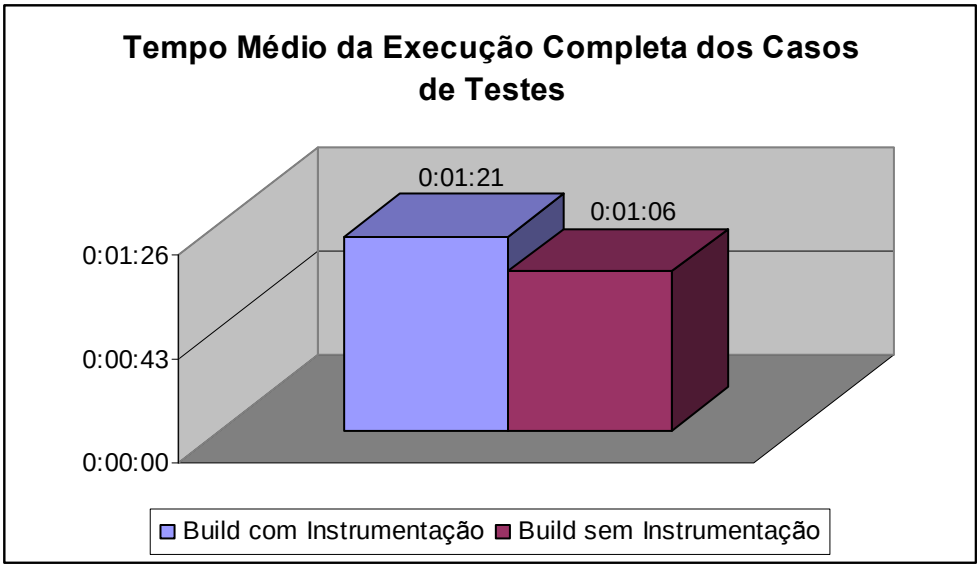



**Figura 5-10:** Tempo médio completo da execução de todos os casos de testes realizado por um testador experiente

Pode-se constatar, nas figuras 5-9 e 5-10, que em alguns casos de testes o testador teve dificuldade e em outros os tempos foram os mesmos, logo é possível afirmar que o problema estava mais atrelado ao fato do conhecimento da *feature* do que apropriadamente o uso de novas tarefas, com o intuito de gerar relatório de cobertura.

## 5.5 Análise Crítica do Processo Proposto

Ao propor um processo de desenvolvimento de código, é necessário verificar a viabilidade do mesmo, do ponto de vista de quem irá utilizar na prática (as equipes de desenvolvimento de software), para avaliar sua utilidade e seus benefícios. Em [Pfleeger, 2002] são descritas técnicas de validação de processos, métodos e ferramentas de engenharia de software. Dentre estas técnicas, as mais utilizadas são experimentos, estudos de caso, questionários e entrevistas. Questionários e entrevistas são boas técnicas para o levantamento de dados qualitativos e para saber a opinião de pessoas envolvidas em um determinado projeto.

Para obter uma maior legitimidade e imparcialidade na avaliação do *Code Coverage Process*, um questionário foi enviado a um grupo de profissionais da área de desenvolvimento de software. Tal questionário serviu para verificar a satisfação dos objetivos almejados pelo processo proposto e a possibilidade de sua utilização em metodologias tradicionais, bem como em metodologias ágeis, mesmo tendo o processo sido inspirado no RUP, uma metodologia tradicional.

O questionário foi divido em duas partes principais. A primeira parte foi utilizada para identificação do perfil dos entrevistados e a segunda parte para uma análise dos três objetivos que o processo se destina a atender e se o mesmo é aplicável às metodologias tradicionais e ágeis. Além do questionário, também foi disponibilizado o processo para que o avaliador fosse capaz de responder às perguntas claramente. A análise crítica se deu em duas fases: na primeira, avaliou-se uma versão inicial do processo, para que em seguida fosse realizado um refinamento do mesmo. Na segunda, fase buscou-se validar o refinamento realizado anteriormente.

### 5.5.1 Perfil dos Entrevistados

O perfil dos profissionais selecionados para compor o grupo de entrevistados foi escolhido de acordo com o papel desempenhado na época da pesquisa e de acordo com o conhecimento em processo e testes de software. Os papéis utilizados para identificação dos profissionais foram:

- **Engenharia** - pessoas envolvidas com aspectos técnicos do desenvolvimento;
- **Testes** - pessoas envolvidas com testes de software;

- **Qualidade** - pessoas envolvidas com a garantia da qualidade e melhoria de processos;
- **Gerência** - pessoas envolvidas com gerenciamento de prazo, escopo, custos e riscos.

O tempo de experiência profissional também foi considerado na coleta das informações, devido a sugestões de melhoria que poderiam fornecer ao processo proposto. Foi dada prioridade, principalmente, às pessoas que exercem papéis relacionados com o que foi adaptado no RUP para o processo proposto.

### 5.5.2 Perfil das Perguntas

As perguntas referentes à análise crítica do processo foram relacionadas ao processo, teste e métricas de qualidade de software, averiguando se o processo proposto contribuía para os objetivos apontados nos questionários e se poderia ou não ser aplicado a metodologias tradicionais e ou ágeis.

No final do questionário havia um espaço reservado para os entrevistados descreverem os benefícios e/ou problemas do processo proposto. A seguir o resultado do questionário é relatado para cada fase.

### 5.5.3 Primeira fase da Avaliação Qualitativa

Na primeira fase de análise, o objetivo foi coletar mais as sugestões dos entrevistados do que avaliar o processo. Com todas as sugestões coletadas foi possível realizar um refinamento do processo proposto. Nesta fase o questionário foi enviado para 12 pessoas, mas apenas 5 responderam.

São apresentados na Figura 5-11 os valores obtidos dos profissionais entrevistados na primeira fase da análise crítica: três profissionais entrevistados possuem uma atuação mais técnica na execução das tarefas de desenvolvimento de software, por atuarem na área de engenharia; 1 atua na área da garantia da qualidade; e 1 tem perfil de testador. Infelizmente não foi possível obter um entrevistado com perfil de gerência e com experiência em planejamento e acompanhamento de recursos no projeto, pois estes não entregaram em tempo o questionário aplicado.

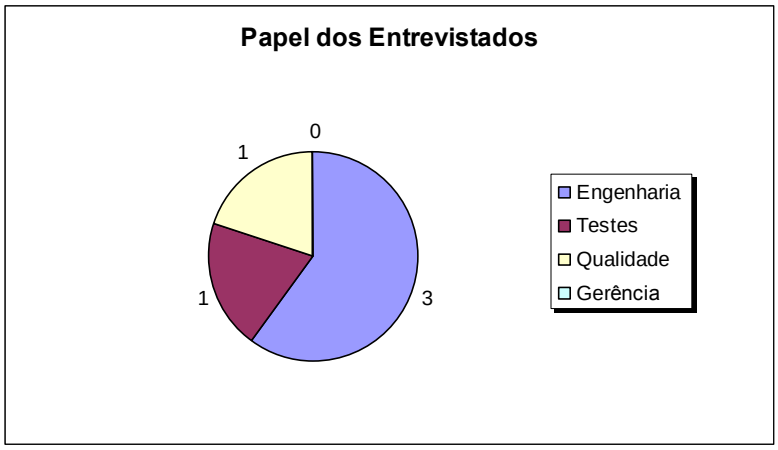



**Figura 5-11:** Perfil dos profissionais entrevistados na primeira fase

A experiência profissional é um fator bastante importante para que uma opinião seja encarada com maior confiança. O nível de experiência profissional dos participantes da pesquisa foi elevado. Como pode ser observado na Figura 5-12, três entrevistados têm mais de 5 anos de experiência profissional, acarretando uma maior confiabilidade nos resultados obtidos na primeira fase de análise crítica.

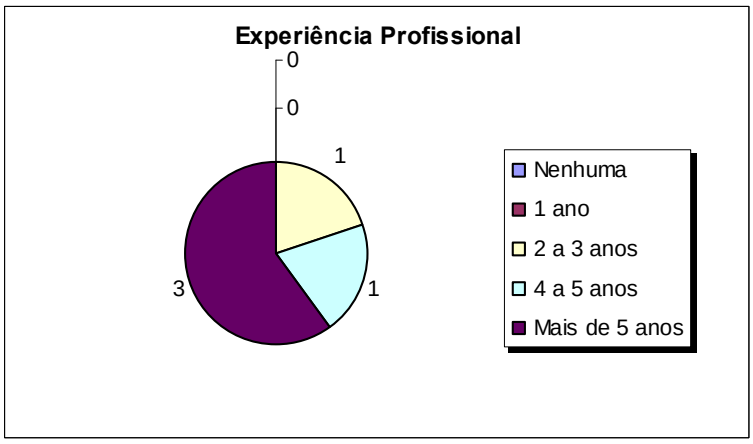



**Figura 5-12:** Experiência profissional dos entrevistados da primeira fase

O conhecimento dos profissionais em torno do processo de software apresentou três para “Médio” e dois com o conhecimento “Extenso”. Referente aos conhecimentos em tipos teste de software, pode-se observar que dois entrevistados com conhecimento médio e três extenso. Na Tabela 5-2, pode-se observar que a maioria dos entrevistados tinha conhecimento referente a processo e testes de software.

**Tabela 5-2:** Conhecimento referente a Processo e Testes de Software da primeira fase

| | **Processo de Software** | **Testes de Software** |
|---|---|---|
| Pouco | 0 | 0 |
| Médio | 3 | 2 |
| Extenso | 2 | 3 |

Na Tabela 5-3, pode-se observar que a versão inicial do processo contribui em parte ou fortemente com os quatros maiores objetivos escolhidos para avaliar o processo proposto, o qual o mesmo se adequa, que são: Processo e Teste de Software, Verificação e Validação, e Métricas de Qualidade de Software.

**Tabela 5-3:** Crítica ao *Code Coverage Process*: Resultados obtidos da primeira fase

| | **Processo de Software** | **Teste de Software** | **Verificação e Validação** | **Métricas de Qualidade de Software** |
|---|---|---|---|---|
| Contribui fortemente | 4 | 2 | 1 | 2 |
| Contribui em partes | 1 | 3 | 4 | 3 |
| Contribui fracamente | 0 | 0 | 0 | 0 |
| Não contribui | 0 | 0 | 0 | 0 |

Na Tabela 5-4 podemos observar que o processo proposto atende bem os objetivos, podendo ser utilizado para metodologias tradicionais, bem como para metodologias ágeis, mesmo não sendo o foco do processo.

**Tabela 5-4:** Crítica ao *Code Coverage Process*: Resultados obtidos da segunda fase

| | **Processo de Software** | **Teste de Software** | **Verificação e Validação** | **Métricas de Qualidade de Software** |
|---|---|---|---|---|
| Metodologias ágeis | 0 | 0 | 0 | 0 |
| Metodologias tradicionais | 2 | 1 | 1 | 1 |
| Ambos os tipos | 3 | 4 | 4 | 4 |
| Nenhuma deles | 0 | 0 | 0 | 0 |

Nas Figuras 5-13 e 5-14 é apresentado o resultado da análise crítica do *Code Coverage Process* referente à sua utilização como um todo.

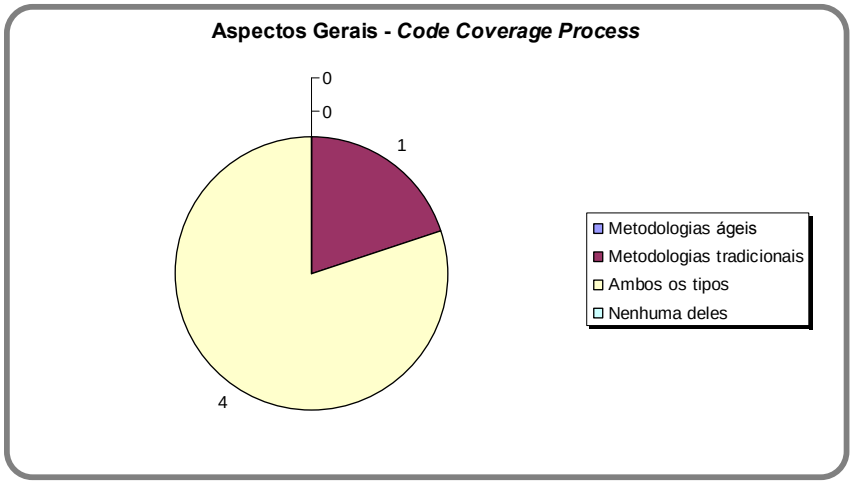



**Figura 5-13:** Aplicabilidade aos tipos de metodologias de desenvolvimento de software da primeira fase

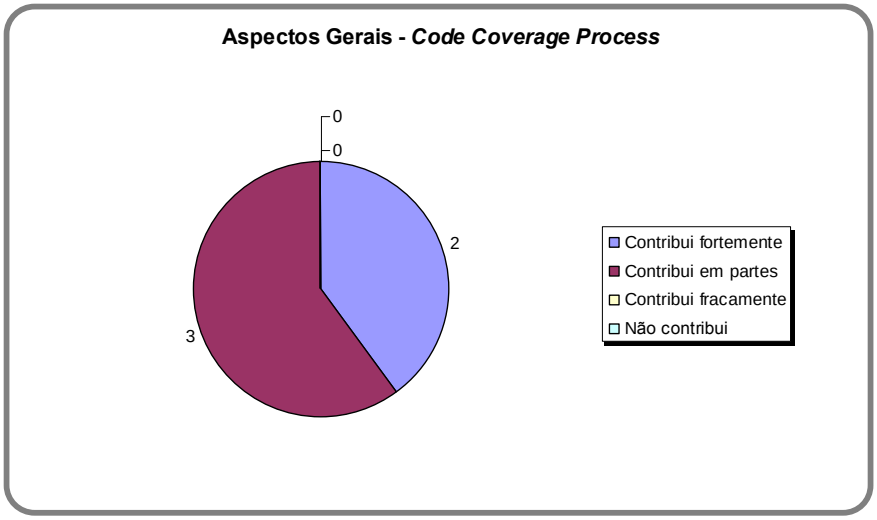



**Figura 5-14:** Contribuição ao comprimento dos objetivos na primeira fase.

Como resultado da aplicação do questionário, obteve-se um excelente *feedback* dos profissionais entrevistados que analisaram o processo. A partir das sugestões, feitas pelos profissionais entrevistados a respeito do processo, foi possível refinar o processo para numa segunda fase poder aplicá-lo novamente em uma nova avaliação, a fim de melhorar o processo e validá-lo. Com base nos resultados, pode-se observar que mesmo o processo sendo inspirado no RUP, qual é um processo tradicional, o *Code Coverage Process* pode ser aplicável também para processos ágeis; outro ponto levantado pelos profissionais entrevistados bastante relevantes é que o processo contribui em partes ou fortemente para o

cumprimento do conjunto de objetivos relacionado a processo, teste e medição de qualidade de software. Um ponto bastante negativo da análise foi à amostra ter sido pequena, visto que seis (07) profissionais não devolveram os questionários a tempo. Na próxima seção, são apresentados os resultados obtidos através da segunda fase da avaliação qualitativa.

### 5.5.4 Segunda fase Avaliação Qualitativa

Na segunda fase foi aplicada a avaliação para um maior número de profissionais, no total de dez entrevistados, com objetivo de validar o processo. Foi aplicado, mais uma vez, um questionário e o processo proposto, sendo que agora refinado, a partir das considerações e observações feitas pelos profissionais entrevistados na primeira fase. A Figura 5-15 mostra o perfil dos profissionais entrevistados e a Figura 5-16 mostra a experiência deles.

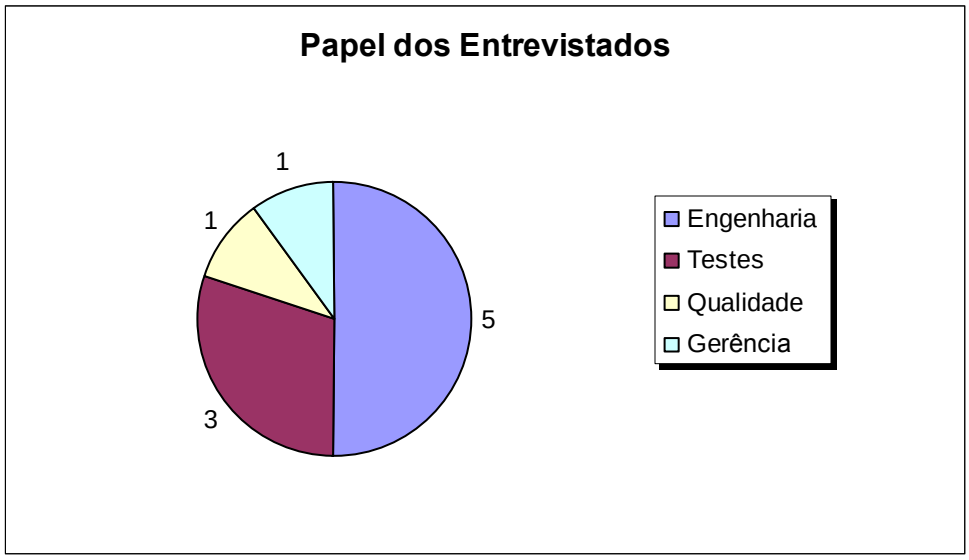



**Figura 5-15:** Perfil dos profissionais entrevistados na primeira fase


**Figura 5-16:** Experiência profissional dos entrevistados da primeira fase

Na próxima seção, são apresentados os resultados obtidos através da segunda parte do questionário de avaliação.

### 5.5.5 Crítica ao *Code Coverage Process*

Após a identificação do perfil dos profissionais entrevistados, os objetivos que o processo pretende atingir foram avaliados. Os entrevistados responderam a duas perguntas, com as quais foram coletadas informações sobre a contribuição do processo para o objetivo descrito e a possibilidade de utilização em metodologias tradicionais e/ou ágeis.

O resultado almejado era que o processo proposto contribuísse fortemente para satisfazer os seguintes objetivos: processo de teste de software (verificação e validação); e métricas de qualidade de software. Um outro resultado esperado é que o processo fosse aplicado pelo menos a metodologias tradicionais.

As Tabelas 5-5 e 5-6 a seguir apresentam um resumo do resultado obtido pela avaliação realizada. Todas as atividades e processos, segundo a opinião dos profissionais entrevistados, contribuem fortemente para a satisfação dos objetivos, bem como podem ser aplicados em metodologias ágeis e metodologias tradicionais.

**Tabela 5-5:** Contribuição do *Code Coverage Process*

| | **Processo de Software** | **Teste de Software** | **Verificação e Validação** | **Métricas de Qualidade de Software** |
|---|---|---|---|---|
| Contribui fortemente | 8 | 7 | 7 | 8 |
| Contribui em partes | 2 | 3 | 2 | 2 |
| Contribui fracamente | 0 | 0 | 1 | 0 |
| Não contribui | 0 | 0 | 0 | 0 |

**Tabela 5-6:** Utilização de metodologias no *Code Coverage Process*

| | **Processo de Software** | **Teste de Software** | **Verificação e Validação** | **Métricas de Qualidade de Software** |
|---|---|---|---|---|
| Metodologias ágeis | 1 | 0 | 0 | 1 |
| Metodologias tradicionais | 4 | 7 | 4 | 3 |
| Ambos os tipos | 5 | 3 | 6 | 6 |
| Nenhuma deles | 0 | 0 | 0 | 0 |

Por fim, foi averiguado, a partir dos questionários se a utilização do *Code Coverage Process* como um todo contribui para cumprimento do conjunto de objetivos descritos e se o processo é aplicável a metodologias tradicionais e ágeis. Assim obtiveram-se os resultados que podem ser observados na Figura 5-17 e na Figura 5-16 e novamente os resultados foram satisfatórios.

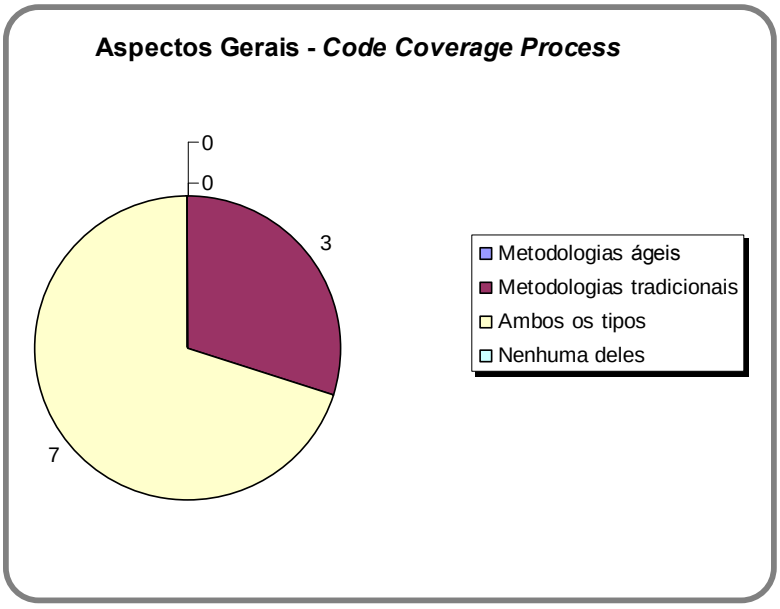



**Figura 5-17:** Aplicabilidade aos tipos de metodologias de desenvolvimento de software da segunda fase

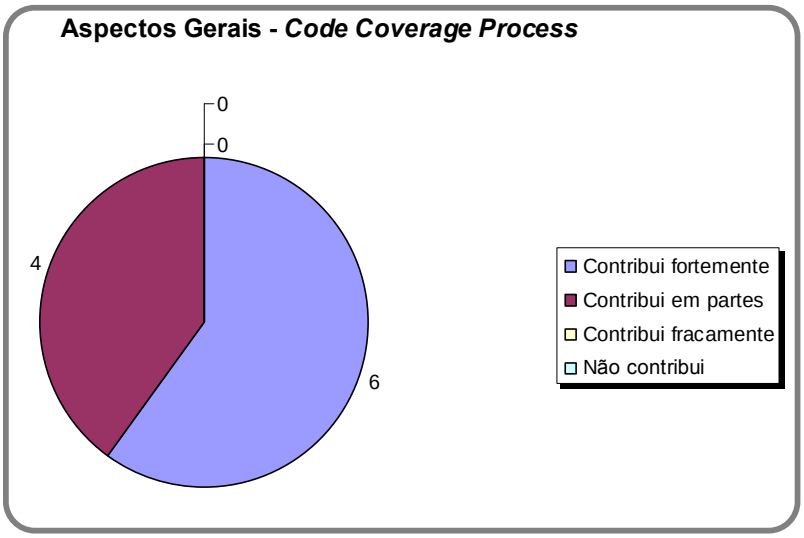



**Figura 5-18:** Contribuição ao cumprimento dos objetivos na segunda fase.

## 5.6 Considerações Finais

Neste capítulo, foram apresentados o estudo de caso e a análise crítica do *Code Coverage Process,* visando avaliar a eficácia em atingir os objetivos propostos A análise crítica foi realizada em duas fases. Na primeira fase, houve uma análise qualitativa, com aplicação do processo e de um questionário para um grupo de pessoas envolvidas com o processo, para coletar informações sobre em que o processo pode ser melhorado, contribuindo assim para um refinamento do processo. Na segunda fase, o processo e o

questionário foram enviados novamente para profissionais da área de desenvolvimento de software para evidenciar a conformidade do processo proposto e desta forma confirmar o acerto do refinamento.

# 6 Conclusão

Visando atender as necessidades de desenvolvimento de software com o suporte da análise de cobertura de código, este trabalho propôs um processo, utilizando e adequando a metodologia de desenvolvimento de software RUP (*Rational Unified Process*), mais especificamente os fluxos de Implementação, Testes, Gerenciamento de Projeto, e Gerenciamento de Configuração e Mudança.

O desenvolvimento de software com qualidade passa, necessariamente, pela execução de atividades de garantia da qualidade, sendo a atividade de testes uma das que mais importantes neste contexto. Neste capítulo são apresentadas as considerações gerais e contribuições deste trabalho, bem como os trabalhos correlatos e propostas de trabalhos futuros.

## 6.1 Principais Contribuições

A principal contribuição deste trabalho foi a concretização de um processo de desenvolvimento, inspirado no RUP, contemplando a análise de cobertura de código, no qual foram adaptadas e inseridas atividades e artefatos. A adaptação foi realizada em quatro fluxos do RUP: Implementação, Testes, Gerenciamento de Projeto e Gerenciamento de Configuração e Mudança.

O processo proposto tem como objetivo definir as atividades, responsáveis e artefatos gerados para o desenvolvimento do código tendo como suporte a análise de cobertura de código. Além disto, por ser inspirado em um processo bastante conhecido e consolidado, será de fácil entendimento.

O *Code Coverage Process* é de fácil entendimento, pois as atividades que foram inseridas são simples de execução. Além disso, o processo proposto foi inspirado num processo bastante consolidado e conhecido nas organizações de software. Pudemos realizar um estudo de caso, no qual analisamos como deveria ser o uso destas novas atividades e quais atividades deveriam ser adaptadas.

O RUP foi escolhido devido ao fato de ser genérico e adaptável e por reunir o melhor de várias técnicas modernas de desenvolvimento de software, bem como pela sua grande aceitação no meio acadêmico e comercial.

Os fluxos de Implementação, Testes, Gerenciamento de Projeto e Gerenciamento de Configuração e Mudança do RUP foram adaptados para considerar mais apropriadamente o desenvolvimento de software com a análise de cobertura de código, sem perder, contudo, as características de adequabilidade a vários tipos de aplicação e a generecidade do processo.

Com a realização do estudo de caso pudemos constatar que com a técnica de análise de cobertura de código para realizar testes que validem a efetividade dos casos de testes, o código gerado tem uma melhoria significativa na sua qualidade, resultando num ganho na produtividade do desenvolvimento e num melhor gerenciamento do projeto referente ao controle de estimativas dos testes de unidade. Outro fator é que as falhas que deveriam ser encontradas nesta fase não irão passar para fases posteriores de testes.

As atividades e artefatos sugeridos no processo são simples, pois procuramos deixar o processo genérico o suficiente e de fácil entendimento.

## 6.2 Trabalhos Futuros

Com a finalização do trabalho, tivemos a possibilidade de fazer uma análise do mesmo, a qual resultou no seguinte conjunto de propostas para a sua continuidade:

- Este trabalho foi aplicado (estudo de caso) em apenas projetos que utilizam metodologia tradicional, ficando para trabalho futuro a aplicação do mesmo para projetos de desenvolvimento de software que utilizam metodologias ágeis, a fim de saber o impacto e as interações entre as atividades propostas e as existentes no processo ágil. No entanto, com base na análise qualitativa, realizada pela aplicação de questionários, pudemos questionar se o processo proposto poderia ser utilizado em processos ágeis e a maioria das pessoas respondeu que sim.
- Realização de experimentos para definição de critérios referente aos limiares aceitáveis de cobertura de código para um determinado projeto e/ou organização;
- Implantação deste processo de cobertura de código em outras organizações, a fim que se possa verificar, analisar e validar os seus benefícios.

# Referências Bibliográficas


[Agrawal, 2002] AGRAWAL, H. Efficient Coverage Testing Using Global Dominator Graphs. Telcordia Technologies. Morristown. 2002.

[Asaf et al., 2004] ASAF, S.; MARCUS, E.; ZIV, A. Defining Coverage Views to Improve Functional Coverage Analysis. ACM SIGSOFT Software Engineering. California, USA. pp. 41-44. 2004.

[Baca, 2005] BACA, C. PMP. Project Manager's Spotlight on Change Management. Harbor Light Press. 2005.

[Bullseyecoverage, 2006] BULLSEYECOVERAGE. Bullseye Testing Technology. Disponível em: http://www.bullseye.com/productInfo.html. Último acesso: 11 de novembro de 2006.

[Cantata, 2006] CANTATA C++. IPL software Products Group. Disponível em: http://www.ipl.com/products/tools/pt400.uk.php. Último acesso: 11 de novembro de 2006.

[Clover, 2006] CLOVER Code Coverage. Cenqua Pty Ltd. Disponível em: http://www.cenqua.com/clover/. Último acesso: 11 de novembro de 2006.

[Conallen, 2000] CONALLEN, J. Building Web Applications with UML. Addison Wesley Longman Publishing Co., Inc., Boston, USA. 2002.

[Cornett, 2005] Cornett, S. Code Coverage Analysis. Bullseye Testing Technology. Disponível em: http://www.bullseye.com/coverage.html. Último acesso: 21 de Outubro de 2005.

[Dustin, 2002] DUSTIN, E. Effective Software Testing: 50 Specific Ways to Improve Your Testing. Addison Wesley. : December 18, 2002.

[Elbaum, 2001] ELBAUM, S.; GABLE, D.; ROTHERMEL, G. The impact of software evolution on code coverage information. Software Maintenance, 2001. Proceedings. IEEE International Conference on. pp. 170-179. 2001.

[Filho, 2001] FILHO, W. P. P. Engenharia de software: fundamentos, métodos e padrões. Rio de Janeiro: LTC, 2001.

[Gokhale, 2005] GOKHALE, S.S.; MULLEN, R.E. Dynamic Code Coverage Metrics: A Lognormal Perspective. Software Metrics. 11th IEEE International Symposium. pp. 33-33. 2005.

[Goodman, 1993] GOODMAN, P., Practical Implementation of Software Metrics, McGraw Hill, 1993.

[Hunt, 2003] HUNT, A.; THOMAS, D. Pragmatic Unit Testing in Java with JUnit. The Pragmatic Bookshelf. Raleigh, North Carolina Dallas, Texas, EUA. 2003.

[IEEE, 1992] IEEE - Standard for a Software Quality Metrics Methodology, IEEE Software, 1992.

[Inthurn, 2001] INTHURN, C. Qualidade e Testes de Software. Visual Books. 2001.

[Jorgensen, 1995] JORGENSEN, P. C. Software Testing – a Craftsman Approach. CRC Press, 1995.

[Kan, 2003] KAN S. H, Metrics and Models in Software Quality Engineering, Addison Wesley, 2003.

[Kim, 2003] KIM, Y. W. Efficient Use of Code Coverage in Large-Scale Software Development. Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research. Ontario, Canada, pp. 145–155. 2003.

[Kruchten, 2003] KRUCHTEN, P. Rational Unified Process, The: An Introduction. Addison Wesley Longman Publishing Co., Inc., Boston, USA, 2003.

[Ldra, 2005] LDRA. Testbed Manual. C/C++ 7.x (Windows 95\98\2K\NT\XP) Manual Revision 23 - February 2005. Liverpool: Liverpool Data Research Associates Ltd.

[Lewis, 2000] LEWIS, W. E. Software Testing and Continuous Quality Improvement. Boca Raton: CRC Press, Florida, EUA. 2000.

[Li, 2004] LI K.; WU M. Effective Software Test Automation: Developing an Automated Testing Tool. 2004.

[Maldonado, 2000] MALDONADO, J. C. et al. Introdução ao Teste de Software. In: XIV Simpósio Brasileiro de Engenharia de Software. Minicurso. João Pessoa, Brasil. 2000.

[Mcandrews, 1993] MCANDREWS D. R. Establishing a Software Measurement Process. Documento Nº CMU/SEI-93-TR-16, 1993.

[Mcgarry, 2002] MCGARRY, J.; CARD D.; Jones C.; LAYMAN B.; CLARK E.; DEAN J.; HALL F. "Practical Software Measurement: Objective Information for Decision Makers", Addison Wesley, 2002.

[Myers, 2004] MYERS, G. The Art of Software Testing. John Wiley & Sons, Inc., New Jersey, USA. 2004.

[OMG, 2005] OMG, Software Process Engineering Metamodel Specification - SPEM, Version 1.1, 2005.

[Patton, 2005] PATTON, R. Software Testing. Sams Publishing. 2005.

[Pfleeger, 2002] PFLEEGER, S. L.; KITCHENHAM, B. Principles of survey research: Principles of survey research part 4: questionnaire evaluation, Software Engineering Notes, vol 27. 2002.

[Piwowarski, 1993] PIWOWARSKI, P.; OHBA, M.; CARUSO, J. Coverage Measurement Experience During Function Test. International Conference on Software Engineering, p. 287-301, 1993.

[Pressman, 2006] PRESSMAN, R. S. Engenharia de Software. McGraw-Hill, 6º Edição, 2006.

[Rational, 2006] Rational PureCoverage. IBM Rational. Disponível em: http://www-306.ibm.com/software/awdtools/purecoverage/support/index.html. Último acesso: 11 de novembro de 2006.

[Shye et al., 2005] SHYE, A.; IYER, M.; REDDI, V. J.; CONNORS, D. A. Code Coverage Testing Using Hardware Performance Monitoring Support. Proc. of the sixth international symposium on Automated analysis-driven debugging AADEBUG'05. ACM Press. Monterey, California, USA. pp. 159-163. 2005.

[SOARES, 2006] SOARES, E. R.; VASCONCELOS, A. M. L. Adaptando o RUP para Análise de Cobertura de Código. I Simpósio Brasileiro de Testes de Software. 2006.

[SOARES, 2006] SOARES, E. R.; VASCONCELOS, A. M. L. Um Processo de Análise de Cobertura alinhado ao Processo de Desenvolvimento de Software em Aplicações Embarcadas. IV Workshop de Teses e Dissertações em Qualidade de Software, V Simpósio Brasileiro de Qualidade de Software. 2006.

[IEEE, 2001] SWEBOK - Guide to the Software Engineering Body of Knowledge – Software Engineering Tools and Methods Knowledge Area, IEEE – Trial Version 1.00 – May 2001.

[Testwell, 2006] Testwell CTC++. Testwell. Disponível em: http://www.testwell.fi/ctcdesc.html. Último acesso: 11 de novembro de 2006.

[Tikir, 2002] TIKIR, M. M.; HOLLINGSWORTH, J. K. Efficient Instrumentation for Code Coverage Testing. ACM SIGSOFT Software Engineering. Roma, Italy. pp. 86-96. 2002.

# Apêndice A - Software Process Engineering Metamodel (SPEM)

A notação Software *Process Engineering Metamodel* (SPEM) é um metamodelo utilizado para especificar, definir processos e seus componentes.

"Este metamodelo é usado para descrever um processo de desenvolvimento de software concreto ou uma família de processos de desenvolvimento de software relacionados" [OMG, 2005].

Em novembro de 2002, foi oficializado como um padrão da OMG (*Object Management Group*) e encontra-se atualmente na versão 1.1. Aborda orientação a objetos e define estereótipos UML (*Unified Modeling Language*) para modelagem de processos de software. Sua definição se baseia em uma arquitetura de quatro níveis de modelagem definida pela OMG, como mostra a Figura A-1.

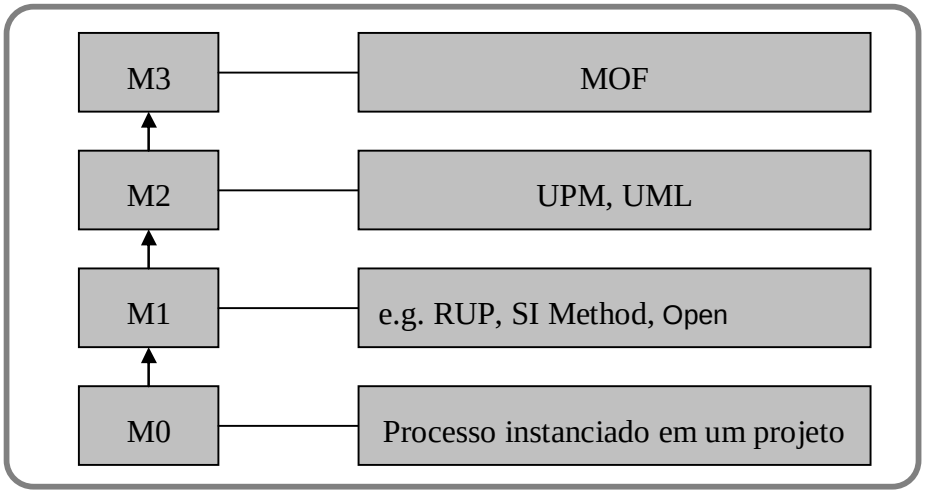


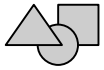

**Figura A-1:** Arquitetura do metamodelo SPEM

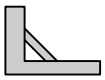
A camada M0 descreve um processo quando ele é instanciado para um projeto especifico. A camada M1 descreve um modelo de processo como o RUP, OPEN, ou XP. A camada M2 descreve o nível de metamodelo do processo, o SPEM se encaixa nesta camada, definida como a camada dos metamodelos de processo que servem de *template* para a camada M1. A camada M3 descreve o metamodelo baseado no *Meta-Object Facility* (MOF). MOF é a tecnologia utilizada pela OMG para determinar meta dados. Desta forma

o SPEM faz uso de um subconjunto da UML para representar seus elementos como um meta-modelo MOF.

Utiliza-se o SPEM através de diagramas UML Estereotipados para elementos de modelagem. A Tabela A-1 descreve alguns estereótipos do SPEM.

**Tabela A-1:** Estereótipos do SPEM

| Estereótipos | Definição | Notação |
|---|---|---|
| *WorkProduct* (Artefato) | Classe de produto de trabalho produzido, consumido ou modificado por um processo. Ex.: documento, código fonte, modelos, planos, etc. |  |
| *ProcessRole* (Papel) | É uma subclasse de *ProcessPerformer*. Ela define responsabilidade em relação a *WorkProducts* específicos e define papéis que realizam e auxiliam atividades específicas. |  |
| *Discipline* (Disciplina) | É um agrupamento coerente de elementos do processo (artefatos, papéis, atividades) cujas atividades são organizadas segundo algum ponto de vista ou tema comum (Ex: Análise e Projeto, Teste, Implementação, etc.). Notação especial para pacotes no contexto SPEM. |  |
| *Phase* (Fase) | É uma especialização de um *WorkDefinition* em que sua pré-condição define critérios de entrada e seus objetivos definem critérios de saída da fase. |  |
| *Process* (Processo) | Representa um processo completo, em toda sua extensão. |  |
| *Activity* (Atividade) | Descreve uma parte do trabalho realizado por um *Processrole.* As tarefas, operações e ações executadas ou podem ser auxiliadas por um papel. Uma atividade é constituída de elementos atômicos chamados *Steps* (passos). |  |

| | | |
|---|---|---|
| *Guidance* (Guia) | Informação mais detalhada a respeito de um *ModelElement* associado. Ex.: *Guidelines*, Técnicas, Métricas, Ferramentas, *Checklists* e *Templates*. |  |
| *Document* (Documento) | Representa um documento criado, visualizado ou modificado através de um processo. |  |
| *ProcessPerformer* | Define o realizador (executor) de um conjunto de WorkDefinitions no processo. |  |
| *WorkDefinition* (Conjunto de trabalho) | Elemento do modelo que descreve a execução, operações realizadas e transformações feitas nos "WorkProducts". |  |

# Apêndice B - *Rational Unified Process* (RUP)

O processo proposto é inspirado no *Rational Unified Process*, tendo sido escolhido devido a sua consolidação em toda a comunidade de software e por ser bastante utilizado, será de fácil entendimento, podendo ser utilizado por qualquer organização de desenvolvimento de software.

O *Rational Unified Process* (RUP) é um processo de engenharia de software que oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento [Kruchten, 2003].

Baseado em boas práticas de desenvolvimento, o RUP é um framework para definição de processos, basta instanciá-lo e definir padrões e guias específicos para a realidade de cada organização e/ou projeto.

Neste apêndice, são apresentadas as características do *Rational Unified Process* (baseado na versão 2003), bem como os fundamentos e práticas que orientam o processo de desenvolvimento de software. A proposta é a elaboração de um processo de cobertura de código alinhado ao processo de desenvolvimento de software, voltado para averiguar a efetividade dos casos de testes de unidade, eliminando os testes redundantes e proporcionando uma maior qualidade no produto.

## Conceitos-Chave

O RUP é composto a partir de alguns conceitos chave, cujos relacionamentos descrevem a estrutura do processo. Os principais conceitos são apresentados a seguir:

- Disciplina: é uma coleção de atividades relacionadas a um determinado assunto, por exemplo: testes. As atividades são agrupadas em disciplinas para organizar o processo e facilitar seu entendimento. Cada disciplina tem seus responsáveis (quem executa a tarefa), atividades (como podem executar as tarefas) e os artefatos (o que a atividade produz);
- Fluxo de Trabalho: uma seqüência de atividades de uma disciplina que produz um resultado de valor para o projeto. Um fluxo de trabalho descreve o processo, indicando, de maneira macro, a ordem em que às atividades devem ser executadas, sendo que cada atividade deste diagrama corresponde a um subfluxo;
- Subfluxo: conjunto de atividades, responsáveis envolvidos, artefatos de entrada e artefatos produzidos; utilizados para fornecer um alto nível de abstração e para estruturar um fluxo;
- Papel: podem ser visualizados como funções exercidas ao longo do processo. Os papéis não são necessariamente indivíduos: um mesmo membro da equipe pode desempenhar papéis e um papel pode ser desempenhado por várias pessoas;
- Atividade: descreve as tarefas, operações, e ações que são executadas pelo seu responsável pelo papel. Cada atividade é relacionada diretamente a uma disciplina ou *workflow* do Processo. Em geral, a atividade consiste em produzir ou atualizar um artefato. A atividade é considerada a unidade básica de trabalho, tendo duração de poucas horas e poucos dias. Uma mesma atividade pode ser executada várias vezes em um mesmo artefato, principalmente para detalhá-lo, no decorrer de uma série de iterações;
- Artefato: peça de informação que é produzida, modificada ou usada pelo processo de desenvolvimento de software; pode ser um modelo, um elemento

do modelo, um documento, código fonte ou até uma versão executável do sistema. As atividades produzem artefatos, que irão servir de entrada para outras atividades.

O RUP tem duas dimensões:

- A dimensão horizontal apresenta o tempo e trata de aspectos dinâmicos do processo, expressos em termos de fases (Iniciação, Elaboração, Construção e Transição), iterações, e marcos de referência.
- A dimensão vertical apresenta os fluxos de processo e de suporte, e trata de aspectos estáticos do processo, descritos em termos de componentes de processo (atividades, artefatos, e papéis), que logicamente agrupam as atividades de engenharia de software de acordo com sua natureza.

A Figura B-1 mostra o relacionamento entre as fases e disciplinas do RUP. Na fase de Iniciação, por exemplo, há uma grande ênfase nas disciplinas Modelagem do Negócio e Requisitos, enquanto na Construção, a ênfase nas atividades de Implementação é maior.

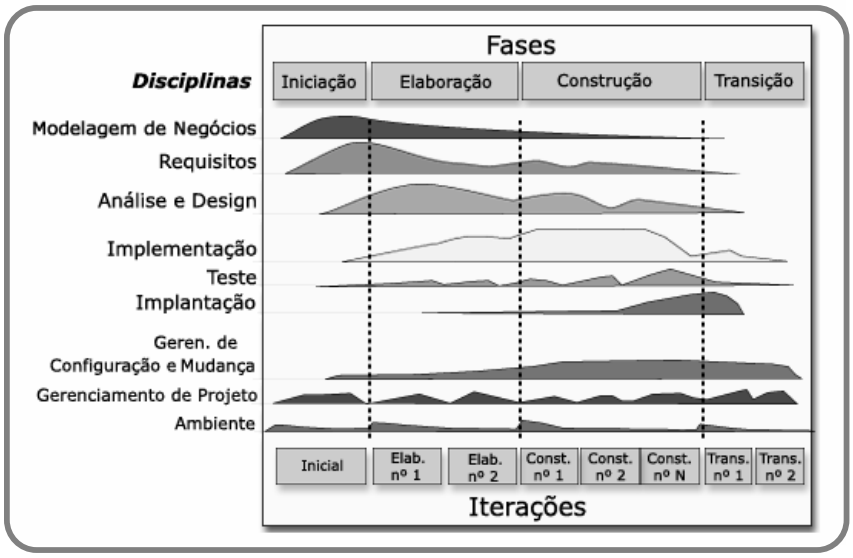



**Figura B-1:** Fluxo do Processo

## Características do RUP

O RUP é baseado em componentes, o que significa que o sistema a ser desenvolvido, será formado por componentes interconectados através de interfaces bem definidas. As principais características do RUP são as seguintes: Iterativo e Incremental; Dirigido a Casos de Uso; Centrado na Arquitetura.

**Iterativo e Incremental**

O desenvolvimento de um produto de software pode continuar por muito tempo. Por isso é prático que o trabalho seja dividido em pequenos ciclos ou miniprojetos. Em cada iteração, são identificados e especificados os Casos de Uso mais relevantes. É feita análise e projeto dos casos de uso, usando-se a arquitetura como guia. São implementados componentes que realizam o que foi projetado e no fim é verificado se os mesmos satisfazem os casos de uso selecionados. Se a iteração alcança suas metas, o desenvolvimento então, procede para a próxima iteração. Os benefícios da abordagem iterativa são:

- Os riscos são reduzidos mais cedo, pois os elementos são integrados progressivamente;
- As mudanças táticas e os requisitos variáveis possuem um melhor suporte;
- Os esforços dos desenvolvedores são mais bem planejados, pois eles trabalham mais efetivamente em direção a um resultado claro;
- A melhoria e o refinamento do produto são facilitados, resultando em um produto mais robusto;
- As organizações podem aprender a partir dessa abordagem e melhorar seus processos e a capacidade de reutilização aumenta;
- A cada versão do sistema, os usuários podem validar o mesmo e propor mudanças nos requisitos.

**Dirigido a Casos de Uso**

Um sistema de software é desenvolvido para atender necessidade dos usuários. Conseqüentemente, para se desenvolver um sistema de sucesso deve-se conhecer o que o usuário espera e necessita do sistema.

Uma possível interação entre um usuário e o sistema pode ser representada por um Caso de Uso, o qual é um pedaço de funcionalidade do sistema que retorna ao usuário um resultado de valor observável. Os Casos de Uso especificam os requisitos funcionais e juntos formam o Modelo de Casos de Uso que descreve todas as funcionalidades do sistema.

Os casos de uso capturaram os requisitos que têm valor para usuários, e direcionam todo o processo (planejamento, análise e projeto, implementação de componentes e testes), bem como, dão suporte ao desenvolvimento da Arquitetura e ajudam o desenvolvimento iterativo.

Durante a análise, projeto e implementação os casos de uso são “realizados”. No momento dos testes, verifica-se o sistema realiza o que está descrito no modelo de casos de uso.

**Centrado na Arquitetura**

A arquitetura de um sistema de software é sua organização ou estrutura de componentes significativos do sistema. Uma arquitetura pode ser repetidamente decomposta. A arquitetura é uma visão do projeto como um todo, que torna visível as características mais importantes do mesmo.

A arquitetura de software não diz respeito apenas à estrutura e organização, mas também a uso, funcionalidade, desempenho, flexibilidade, reutilização, abrangência, restrições tecnológicas e econômicas.

O RUP recomenda diversas visões de arquitetura: “Casos de Uso”, “Lógica”, “Implementação”, “Processo” e “Distribuição”. Cada visão é um subconjunto dos modelos. O documento da arquitetura deve ser uma reescrita dos modelos para torná-lo mais fácil de entender.

## Ciclo de Vida do Software no RUP

O ciclo de vida do RUP é dividido em quatro fases: Iniciação, Elaboração, Construção e Transição. Cada fase é concluída quando o *milestone* determinado é concluído. Cada ciclo termina com uma versão do produto. Cada fase é adicionalmente dividida em iterações, A Figura B-2 ilustra este ciclo.

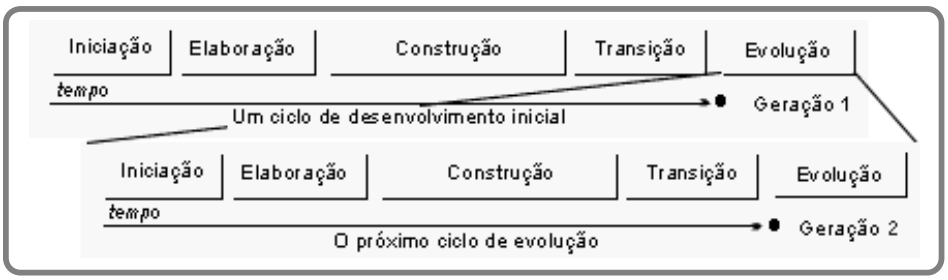



**Figura B-2:** Ciclo de vida do RUP

Os ciclos de evolução podem ser disparados por melhorias sugeridas pelos usuários, mudanças no contexto do usuário, mudanças na tecnologia adotada, reação à concorrência e assim por diante. Normalmente, os ciclos de evolução têm fases de Iniciação e Elaboração bem menores, pois a definição e a arquitetura básicas do produto foram determinadas por ciclos de desenvolvimento anteriores.

### Iniciação

Esta fase é a primeira do ciclo e tem como objetivos principais definir o escopo do sistema e condições limites, incluindo uma visão operacional, o critério de aceitação e o que é para ser esperado do produto e o que não é; discriminação dos Casos de Uso críticos do sistema e restrições; exibição e/ou demonstração de pelo menos uma arquitetura candidata. No que diz respeito ao gerenciamento do projeto, são elaboradas as estimativas iniciais de custo e de cronograma para o projeto inteiro e feito o levantamento inicial de riscos, bem como estimativas detalhadas para a próxima fase e preparação do ambiente de suporte, como instalação e configuração de ferramentas. Ao final desta fase, deve-se decidir se é viável ou não levar o projeto adiante.

Temos como saída desta fase a primeira versão de um modelo de negócios que descreve o contexto do sistema:

- Modelo de caso de uso;

- Modelo de análise e projeto;
- Arquitetura candidata;
- Protótipo;
- Lista de risco inicial e suas prioridades;
- Plano de projeto (incluindo as fases).

**Elaboração**

Esta fase captura os requisitos funcionais descritos como casos de uso. Estabelece uma arquitetura sólida para guiar as fases de construção e transição, ou seja, a arquitetura na forma executável, que demonstre a capacidade de acomodar o resto do sistema. Tem-se nesta fase o monitoramento dos riscos críticos e o seu impacto no caso de negócio. Os objetivos são a implantação de um ambiente de suporte e o detalhamento do planejamento. Temos como saída desta fase:

- Modelo de negócio completo;
- Nova versão de todos os modelos iniciados na fase de iniciação;
- Arquitetura executável;
- Descrição arquitetural;
- Lista de risco atualizada;
- Plano de projeto atualizado para as fases posteriores;
- Manual do usuário (opcional).

**Construção**

Esta fase tem como ênfase a produção de um software operacional para testes. Esta fase envolve análise, projeto e implementação dos requisitos levantados na elaboração. O desenvolvimento é monitorado pelo arquiteto para verificar se os casos de uso estão sendo corretamente acomodados na arquitetura. Esta fase possui as atividades de requisitos, análise e projeto, implementação e testes. A saída desta fase se dá através do (a):

- Plano de projeto para a próxima fase;

- Software executável;
- Descrição da arquitetura atualizada;
- Manual do usuário com detalhes suficientes;
- Caso de Negócio refletindo a situação atual.

**Transição**

Esta fase irá validar os requisitos, tentará encontrar pequenas falhas, pois as mais significativas foram descobertas e corrigidas nas fases de iniciação e elaboração. Nesta fase é fornecida informação e/ou treinamentos aos usuários. A saída desta fase se dá através do (a):

- Software executável, incluindo instalação;
- Documentação;
- Produto completo e correto;
- Manual do usuário;
- Suporte ao usuário.

A fase de Transição termina com a liberação do produto formalmente.

## Fluxos do RUP

O RUP está definido em seis fluxos, cada um tem seus responsáveis (quem executa a tarefa), atividades (como podem executar as tarefas) e os artefatos (o que a atividade produz). O propósito dos fluxos é agrupar atividades relacionadas, com o intuito de facilitar o entendimento do processo.

Basicamente, há dois tipos de fluxo no RUP: fluxos de Processo e fluxos de Suporte. Os fluxos de Processo correspondem às atividades de desenvolvimento: modelagem de negócios, requisitos, análise e projeto, implementação, testes e implantação. Os fluxos de Suporte correspondem às atividades de gerenciamento e infra-estrutura: gerenciamento de configuração e mudanças, gerenciamento de projeto e ambiente.

Dentre estes fluxos, o *Code Coverage Process* propõe alteração nos fluxos de Implementação, Testes, Gerenciamento de Projeto e Gerenciamento de Configuração e Mudança. A descrição de cada fluxo é a seguinte:

**Modelagem do Negócio**

O objetivo do fluxo de "Modelagem de Negócio" é entender a estrutura e a dinâmica da organização na qual um sistema deve ser implantado (a organização-alvo). Os problemas atuais da organização-alvo devem ser bem compreendidas e identificadas às possibilidades de melhoria.

**Requisitos**

Estabelece e mantém acordo com os clientes e outros *stakeholders*. Proporciona ao desenvolvimento de sistemas um melhor entendimento dos requisitos do sistema, define os limites do sistema, provê uma base para o planejamento das iterações e uma base para calcular custo e tempo para desenvolver o sistema, define uma interface do usuário para o sistema e focaliza as necessidades e metas dos usuários.

**Análise e Design**

Nesta disciplina os requisitos são analisados, refinados e estruturados. O objetivo é entender ao máximo as especificações dos requisitos, projetando alguns modelos que serão utilizados na disciplina de codificação através do desenvolvimento de uma arquitetura robusta para o sistema.

**Implementação**

Esta disciplina implementa os casos de uso elaborados na disciplina de Análise e *Design* em forma de componentes (por exemplo: código fonte, scripts, binários e executáveis). Nesta disciplina define-se a organização do código, em termos de subsistemas, organizando em forma de camadas. Esta disciplina testa os componentes desenvolvidos com testes de unidade.

Os principais artefatos produzidos por este fluxo são os seguintes: Modelo de Implementação, Plano Integração de *Build*, Componentes, *Build* e Subsistemas de Implementação.

O modelo de implementação é um conjunto de componentes e dos subsistemas de implementação que os contém. O plano de integração do *build* fornece um plano detalhado para integração em uma iteração. Um componente representa um trecho de um código (fonte, binário ou executável) ou um arquivo contendo informações (por exemplo, um arquivo de inicialização ou um arquivo Leia-me). *Build* é uma versão operacional de um sistema ou parte de um sistema, é uma parte integrante do ciclo de vida do software. Um subsistema de implementação é um conjunto de componentes e de outros subsistemas de implementação que é usado para estruturar o modelo de implementação, dividindo-o em partes menores.

**Teste**

O fluxo de "Teste" age em muitos aspectos como um provedor de serviço para outras disciplinas. Teste tem como foco a avaliação ou a estimativa da qualidade do produto. Esta disciplina diferencia das outras, pois é a única que busca defeito no software, enquanto as outras buscam a perfeição e construção.

**Implantação**

A "Implantação" descreve as atividades que garantem que o produto de software será disponibilizado para usuários finais [Kruchten, 2003]. É representado pela execução de várias atividades, tais como, produzir versões do software, empacotar, distribuir e instalar o software, fornecer ajuda e assistência aos usuários, planejamento e execução de beta testes, migração de software ou dados já existentes e aceitação formal.

**Gerenciamento de Configuração e Mudança**

É o processo de identificar, organizar e controlar as modificações do software que está sendo construído. É um fluxo de apoio ao projeto como um todo. Esta disciplina tem como objetivo:

- Definição de um ambiente de desenvolvimento;
- Ter política para controle de versões, garantindo a consistência dos artefatos produzidos;
- Definição de procedimentos para solicitações de mudanças;

- Administração do ambiente e auditoria das mudanças;
- Facilitar a integração das partes do sistema.

O gerenciamento de configuração e mudança controla as mudanças e mantém a integridade dos artefatos do projeto. Nela são identificados os itens de configuração, restrições a mudanças nestes itens, verificação de mudanças feitas nestes itens e a definição e gerenciamento das configurações dos itens, durante o processo de desenvolvimento. Os métodos, processos e ferramentas usados para fornecer gerenciamento de configuração e mudança para uma organização podem ser considerados como o sistema de CM (*Configuration Management*) da organização.

O sistema de Gerência de Configuração de uma organização é um conjunto de atividades que foram desenvolvidas para administrar as mudanças em todo o ciclo de vida do software, desde a iniciação até a implantação. O sistema de Gerência de Configuração contém as versões atuais e históricas dos arquivos-fonte dos artefatos de requisitos, design e implementação que definem uma determinada versão de um sistema ou de um componente do sistema.

**Gerenciamento de Projeto**

Define, estima e avalia iterações, recursos, esforços e infra-estrutura. Monitora o progresso das atividades, riscos e qualidade dos artefatos. Coordena as pessoas envolvidas no projeto.

Este fluxo foca principalmente nos aspectos importantes de um processo de desenvolvimento iterativo: gerenciamento de riscos, planejamento e monitoração do progresso. Entretanto, no RUP esta disciplina não atende a todos os aspectos de gerenciamento de projeto, tais como: gestão de pessoas (contratação, treinamento e acompanhamento), gestão de orçamento (definição e alocação) e gestão de contratos com fornecedores e clientes.

**Ambiente**

O fluxo “Ambiente” concentra-se nas atividades necessárias à configuração do processo para um projeto. Isto inclui a definição de quais atividades serão ou não executadas e de guias para execução das atividades. A meta das atividades dessa disciplina

é oferecer à organização o ambiente de desenvolvimento de software - processos e ferramentas, que darão suporte à equipe de desenvolvimento [Kruchten, 2003].

## Papéis

Os papéis podem ser visualizados como funções exercidas ao longo do processo. Uma pessoa pode assumir mais de um papel em um projeto, mas para cada papel existirá um objetivo especifico. A Figura B-3 apresenta alguns tipos de papéis, os quais são descritos a seguir.


**Figura B-3:** Papéis

**Gerente**

No RUP existem alguns tipos de gerentes com objetivos distintos que estão principalmente envolvidos no gerenciamento e configuração do processo de engenharia de software. São eles: Engenheiro de Processo, Gerente de Projeto, Gerente de Controle de Mudança, Gerente de Configuração, Gerente de Desenvolvimento e Gerente de Teste. Iremos detalhar o gerente de projeto, pois é o que tem ligação direta com o processo de cobertura de código.

O papel do gerente de projeto é de alocar recursos, analisar e determinar prioridades, coordenar interações com os clientes e usuários, e manter o time de projeto focado diretamente na meta. O gerente de projeto também estabelece um conjunto de práticas que asseguram a integridade e qualidade dos artefatos do projeto.

O gerente de projeto precisa ter algumas habilidades e experiência, dependendo do tamanho e complexidade do projeto:

- Experiência no domínio da aplicação e em desenvolvimento de software;
- Capacidade de liderança;

- Boa comunicação;
- Pró-atividade e foco em resultados.

**Analista**

No RUP existem alguns tipos de analista que estão principalmente envolvidos na elicitação e investigação dos requisitos. São eles: analista de sistema, Designer de negócios, revisor de modelo e negocio revisor de requisitos, especialista de requisitos, analista de teste e designer de interface do usuário.

O analista de sistemas conduz e coordena a elicitação de requisitos e modelagem de casos de uso esboçando a funcionalidade e delimitando o escopo do sistema. Por exemplo: estabelecendo que atores e casos de uso existem no sistema e como eles interagem. O analista deve ter o seguinte conhecimento:

- A tecnologia a ser usada no desenvolvimento do sistema;
- As técnicas de modelagem de casos de uso;
- As técnicas de análise e projeto orientado a objetos.

**Desenvolvedor**

No RUP existem alguns tipos de desenvolvedores que estão principalmente envolvidos no projeto e no desenvolvimento de software. São eles: *Designer* de Cápsula, Revisor de Código, *Designer* de Banco de Dados, Implementador, Integrador, Arquiteto de Software, Revisor de Arquitetura, Revisor de *Design*, *Designer*, *Designer* de Teste. Todos exercendo papeis diferente

O papel de implementador é responsável para desenvolver e testar componentes, conforme os padrões adotados no projeto, para integração de subsistemas maiores. O desenvolvedor também é responsável por desenvolver e executar os testes de componentes e subsistemas correspondentes.

**Testador**

O testador é responsável pelas atividades que envolvem o planejamento dos testes necessários e registro dos seus resultados. Isto cobre:

- Implementação de testes individuais;
- Configuração e execução dos testes;
- Registro e verificação dos resultados da execução dos testes;
- Análise dos erros de execução;

O conhecimento e habilidade podem variar, dependendo do tipo do teste que é executado e a fase do ciclo de vida do projeto, porém em geral, o testador deve ter as seguintes habilidades:

- Conhecimento de técnicas de teste;
- Habilidades para diagnóstico e resolução de problemas;
- Conhecimento do sistema ou aplicação que é testada (desejável).

Os testes automáticos requerem algumas outras habilidades, além das que foram mencionadas anteriormente:

- Treinamento apropriado no uso de ferramenta de automação de teste;
- Experiência em usar ferramentas de automação de teste;
- Habilidades de programação;
- Habilidades de depuração e diagnóstico.

***Stakeholder***

O *stakeholder* está definido como qualquer um que é afetado pelo resultado do projeto. Este é um papel adicional no RUP. Exemplos de *stakeholder*:

- Cliente ou representante de cliente;
- Usuário ou representante de usuário;
- Investidor;
- Acionista;
- Gerente de produção;
- Comprador;

- Desenhista;
- Testador;
- Escritor da documentação.

# Apêndice C - Atividades do RUP Estendido para *Code Coverage Process*

O *framework* apresentado neste apêndice corresponde a uma adaptação dos fluxos de Implementação, Teste, Gerenciamento de Configuração e Mudança, e Gerenciamento de Projeto do RUP para o *Code Coverage Process*.

Neste apêndice, são apresentadas as atividades, bem como seus passos correspondentes. Os fluxos foram alterados de forma a considerar as atividades e artefatos específicos de cobertura de código. Algumas atividades do fluxo original do RUP não foram modificadas e, desta forma, não serão detalhadas neste apêndice.

## Fluxo de Implementação

É na implementação que se realiza o desenvolvimento do produto, e principalmente a elaboração dos testes de unidade que irão exercitar o código que se deseja analisar. A Figura C-1 apresenta a visão geral das atividades do Fluxo de Implementação.

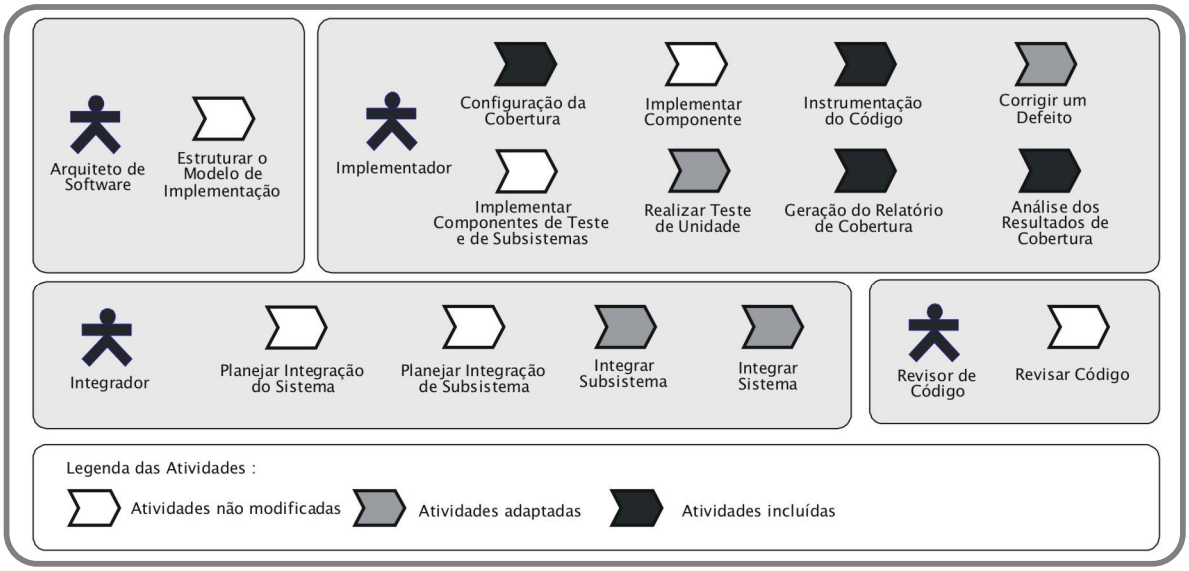



**Figura C-1:** Visão Geral das Atividades do Fluxo de Implementação

Nos parágrafos a seguir estão detalhadas apenas as atividades que foram adaptadas. As atividades que foram incluídas foram explicadas no capítulo três (3). As demais atividades e os artefatos, bem como, os conceitos e mentores de ferramentas podem ser encontrados na versão 2003 do *Rational Unified Process*.

**Atividade: Corrigir um Defeito**

<table>
<tr><td colspan="2">Propósito<br><br>Corrigir um defeito.</td></tr>
<tr><td colspan="2">Passos<br>▪ Analisar o Relatório de Cobertura<br>▪ Estabilizar o Defeito<br>▪ Localizar a Falha<br>▪ Corrigir a Falha</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Solicitação de Mudança<br>▪ Relatório de Cobertura</td><td>Artefatos Resultantes:<br>▪ Componente</td></tr>
<tr><td colspan="2">Responsável: Implementador</td></tr>
</table>

Esta atividade tem como pontos chaves a localização do defeito e sua remoção. O responsável por esta atividade é o implementador que desenvolveu o componente que se encontra com defeito. A seguir os passos desta atividade são descritos.

Analisar o Relatório de Cobertura

O relatório de cobertura servirá como informação para o implementador analisar o que ocasionou a não cobertura do código e principalmente para averiguar se o trecho do código que não foi testado está relacionado a uma falha no código. A baixa cobertura é ocasionada por três fatores principais, que são os seguintes:

- A insuficiência no número de casos de teste de unidade para executar o código;
- Problemas no código derivado do mau desenvolvimento, ocasionando na falha do teste e/ou da sua execução;
- Caso de teste mal elaborado.

O relatório de cobertura acompanhará o documento de solicitação de mudança, a fim que se possa ter o monitoramento e o acompanhamento da evolução do código e dos testes de unidade.

**Estabilizar o Defeito**

A segunda tarefa é estabilizar o defeito, a fim que ocorra de forma segura. Se isso não for possível, será muito difícil localizar a falha.

Verifique se existem casos de teste para este defeito e como se comportou a cobertura realizada por este teste, através do relatório de cobertura.

O caso de teste deve ser limitado para identificar quais os fatores nele contidos que ocasiona o defeito e quais são irrelevantes para o defeito. Para descobrir se um fator é irrelevante, altere o fator em questão e execute o caso de teste. Se o defeito ainda ocorrer, provavelmente esse fator poderá ser eliminado.

Se obtiver êxito, você deverá finalizar com pelo menos um caso de teste que cause o defeito e com alguma idéia dos fatores relacionados à ocorrência do defeito.

**Localizar a Falha**

O objetivo desta tarefa é executar os casos de teste que causam o defeito e tentar identificar em que local do código se encontra a origem da falha. Algumas formas de localizar uma falha são:

- Limite à região suspeita do código: Teste uma parte menor do código; removendo uma parte do código e executando novamente os casos de teste. Se o defeito ainda ocorrer, continue a remover partes do código. Por fim, você terá identificado onde é possível encontrar a falha;
- Use o depurador para percorrer o código linha a linha e monitorar os valores das variáveis interessantes. O relatório de cobertura poderá lhe ajudar, pois indicará o trecho que ficou faltando ser testado;
- Permita que outra pessoa revise o código.

**Corrigir a Falha**

Após a localização da falha, é o momento de corrê-la. São mencionadas a seguir algumas diretrizes a serem lembradas:

- Verifique se você compreende o problema e o programa antes de efetuar a correção.
- Efetue uma mudança de cada vez, pois a correção de falhas é em si uma atividade propensa a erros. É importante implementar as correções gradativamente para facilitar a localização da origem das falhas.

Após a correção da falha, adicione um caso de teste que verifique essa falha em particular. Observe e análise se a inserção deste caso de teste resultou no aumento da cobertura, gerando um novo relatório de cobertura.

**Atividade: Realizar Testes de Unidade**

<table>
<tr><td colspan="2">Propósito<br>Verificar a especificação de uma unidade.<br>Verificar a estrutura interna de uma unidade.</td></tr>
<tr><td colspan="2">Passos<br>▪ Realizar Testes Unitários<br>▪ Avaliar a Execução do Teste<br>▪ Verificar os Resultados do Teste<br>▪ Recuperar Testes Interrompidos</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Documento de Arquitetura de Software<br>▪ Pacote de Design<br>▪ Design da Classe<br>▪ Guia de Design<br>▪ Guia de Programação<br>▪ Especificações Suplementares<br>▪ Modelo de Design<br>▪ Caso de Teste<br>▪ Scripts de Teste<br>▪ Componentes de Teste<br>▪ Componente</td><td>Artefatos Resultantes:<br>▪ Resultados do Teste<br>▪ Componentes de Teste<br>▪ Componente<br>▪ Arquivo de histórico da execução do código</td></tr>
<tr><td colspan="2">Papel: Implementador</td></tr>
</table>

A implementação e a alteração dos componentes acontecem no contexto de gerenciamento de configuração do projeto. Os implementadores, para realizar o seu trabalho, têm um espaço de trabalho privado para desenvolvimento, no qual os elementos são criados e colocados sob o gerenciamento de configuração, ou são alterados no ciclo (realizando as tarefas de *check-out*, editar, compilar, teste de unidade e *check-in*). Após a conclusão de alguns conjuntos de componentes, o implementador liberará os componentes

novos e alterados no espaço de trabalho de integração do subsistema para integração com o trabalho de outros implementadores.

A seguir são descritos os passos da atividade Realizar Testes de Unidade, tendo em destaque as adaptações e inclusões referentes à utilização do *Code Coverage Process*.

**Realizar Testes Unitários**

Este passo tem a finalidade de executar os procedimentos de teste ou os scripts de teste (se o teste for automatizado). Devem ser executados os seguintes passos:

- Configurar o ambiente de teste para assegurar que todos os elementos necessários (como *hardware*, software, ferramentas, dados, etc.) foram implementados e estão no ambiente de teste.
- Inicializar o ambiente de teste para garantir que todos os componentes estejam no estado inicial correto para o início do teste.
- Executar os procedimentos de teste.

Algumas ferramentas geram um arquivo de histórico da execução do código, no momento da execução do teste de unidade. Logo se deve configurar na ferramenta e no ambiente de trabalho um local para o armazenamento deste arquivo. As informações para essa configuração devem estar contidas no *guideline* de cobertura.

**Avaliar a Execução do Teste**

Este passo tem a finalidade de determinar se os testes terminaram com sucesso e como desejado, bem como determinar se é necessária alguma ação corretiva.

A execução do teste é concluída ou finalizada em uma destas duas condições:

- Normal: todos os procedimentos (ou scripts) de teste são executados como esperado.
- Anormal ou prematura: os procedimentos (ou scripts) de teste não foram executados completamente ou como esperado. Quando o teste é encerrado de forma anormal, os resultados correspondentes podem não ser confiáveis. A

causa da finalização deve ser identificada e corrigida e os testes devem ser executados novamente antes da realização de atividades adicionais de teste.

Na cobertura do código se um teste teve problemas na sua execução que o impeça de continuar a sua execução, o mesmo deve ser reiniciado, pois poderá dar problemas no arquivo de histórico da execução do teste.

**Verificar os Resultados do Teste**

Determina se os resultados dos testes estão corretos e identificar ações corretivas para as falhas encontradas. Quando o teste for concluído, análise os resultados do teste para assegurar que eles sejam confiáveis e que as falhas, avisos ou resultados inesperados não tenham sido causados por influências externas, como configuração ou dados inadequados.

Nesta atividade, devem-se analisar os resultados dos relatórios de cobertura do código executado pelos casos de testes de unidade. Baseado no relatório deve-se tomar ações para que os resultados fiquem dentro do limiar de cobertura determinado nos planos de testes.

**Recuperar Testes Interrompidos**

Determina a ação corretiva apropriada para recuperar o teste com falha. Irá corrigir o problema, recuperar o teste e passará para a atividade de execução novamente.

Há dois tipos principais de testes interrompidos:

- Erros fatais: Falha no sistema; por exemplo, falhas na rede, falhas no *hardware* e erros do gênero.
- Falhas de Comando do *Script* de Teste: Relativos ao teste automatizado, quando um *script* de teste não pode executar um comando ou uma linha de código.

A descrição dos artefatos de entrada e saída desta atividade já existente pode ser encontrada na versão 2003 do *Rational Unified Process*.

Os testes interrompidos devem ser recuperados da seguinte forma:

- Determinação da causa do problema.

- Correção do problema.
- Configuração novamente do ambiente de teste.
- Inicialização novamente do ambiente de teste.
- Execução novamente dos testes.

Se tratando de cobertura de código, não pode começar o teste novamente do ponto no qual foi interrompido, pois terá impacto nas métricas de cobertura. Principalmente se o caso de teste não for de unidade, sendo realizado no fluxo de testes, devido que terá impacto no arquivo contendo o histórico da execução.

O ambiente de teste parece não responder ou é exibido em um estado não desejado, como suspenso ou com falhas.

**Atividade: Integrar Subsistema**

<table>
<tr><td colspan="2">Propósito<br>Integrar os componentes em um subsistema de implementação e liberar esse subsistema para integração do sistema.</td></tr>
<tr><td colspan="2">Passos<br>▪ Integrar Componentes<br>▪ Liberar o Subsistema de Implementação</td></tr>
<tr><td>Artefatos Informados:<br>▪ Componente<br>▪ Plano de Integração do Build<br>▪ Subsistema de Implementação</td><td>Artefatos Resultantes:<br>▪ Build<br>▪ Subsistema de Implementação</td></tr>
<tr><td colspan="2">Papel: Integrador</td></tr>
</table>

**Integrar Componentes**

De acordo com o Plano de Integração do *Build* é realizada a integração do subsistema. É aconselhável integrar as classes implementadas (componentes) segundo uma abordagem do tipo *bottom-up* incremental seguindo a ordem de dependências de compilação. A cada incremento, adicione um ou alguns componentes ao sistema.

Em relação o *Code Coverage Process*, a adaptação nesta atividade se dar através que a geração da *build* é realizada com o código instrumentado, a fim que se possam executar os testes, para coleta das métricas de cobertura de código.

**Liberar o Subsistema de Implementação**

Ao fim do ultimo incremento, no qual o subsistema de implementação estiver finalizado e os testes tiverem sido realizados no *build* em relação à integração, tendo as métricas de cobertura do código sido alcançadas, o subsistema será liberado para o espaço de trabalho de integração do sistema.

A descrição dos artefatos de entrada e saída desta atividade já existente pode ser encontrada na versão 2003 do *Rational Unified Process*.

**Atividade: Integrar Sistema**

| **Propósito**<br>Integrar os subsistemas de implementação por partes em um *build*. | |
|---|---|
| **Passos**<br>▪ Aceitar Subsistemas e Produzir *Builds* Intermediários<br>▪ Promover *Baselines* | |
| **Artefatos Informados:**<br>▪ Plano de Integração do *Build*<br>▪ Subsistema de Implementação | **Artefatos Resultantes:**<br>▪ *Build* |
| Papel: **Integrador** | |

**Aceitar Subsistemas e Produzir *Builds* Intermediários**

O integrador, tendo os subsistemas liberados no espaço de trabalho de integração do sistema e quaisquer conflitos resolvidos. O incremento de subsistemas é compilado e vinculado a um *build* intermediário, que será fornecido para o testador de modo que execute um teste mínimo de integração do sistema.

Quando esse *build* tiver sido minimamente testado, uma baseline inicial ou provisória será criada. O *build* é, então, disponibilizado para o testador para o teste completo do sistema. A natureza e o detalhamento desse teste seguirão as especificações do Plano de Integração do *Build*, e o *build* final de uma iteração estará sujeito a todos os testes definidos no Plano de Teste da Iteração.

As *builds* que será encaminhada para o testador poderão ser instrumentadas, de acordo com o interesse da organização em obter ou não métricas de cobertura de código, e/ou ao nível de complexidade do projeto, que necessitará ter essa informação.

**Promover Baselines**

Tendo o *build* aprovada nos diversos níveis de teste, as baselines associadas são promovidas de acordo.

A descrição dos artefatos de entrada e saída desta atividade já existente pode ser encontrada na versão 2003 do *Rational Unified Process*.

**Fluxo de Teste**

Como dito anteriormente, a utilização de cobertura de código no fluxo de Teste surgiu de uma iniciativa da empresa onde o trabalho foi desenvolvido. Na ocasião, surgiu o interesse da alta administração em utilizar as técnicas de cobertura, coletando os resultados para avaliar a eficácia da execução dos testes do tipo "Caixa Preta" aplicado em um *build* instrumentado. A Figura C-2 apresenta a visão geral das atividades do Fluxo de Teste modificado para contemplar a análise de cobertura.

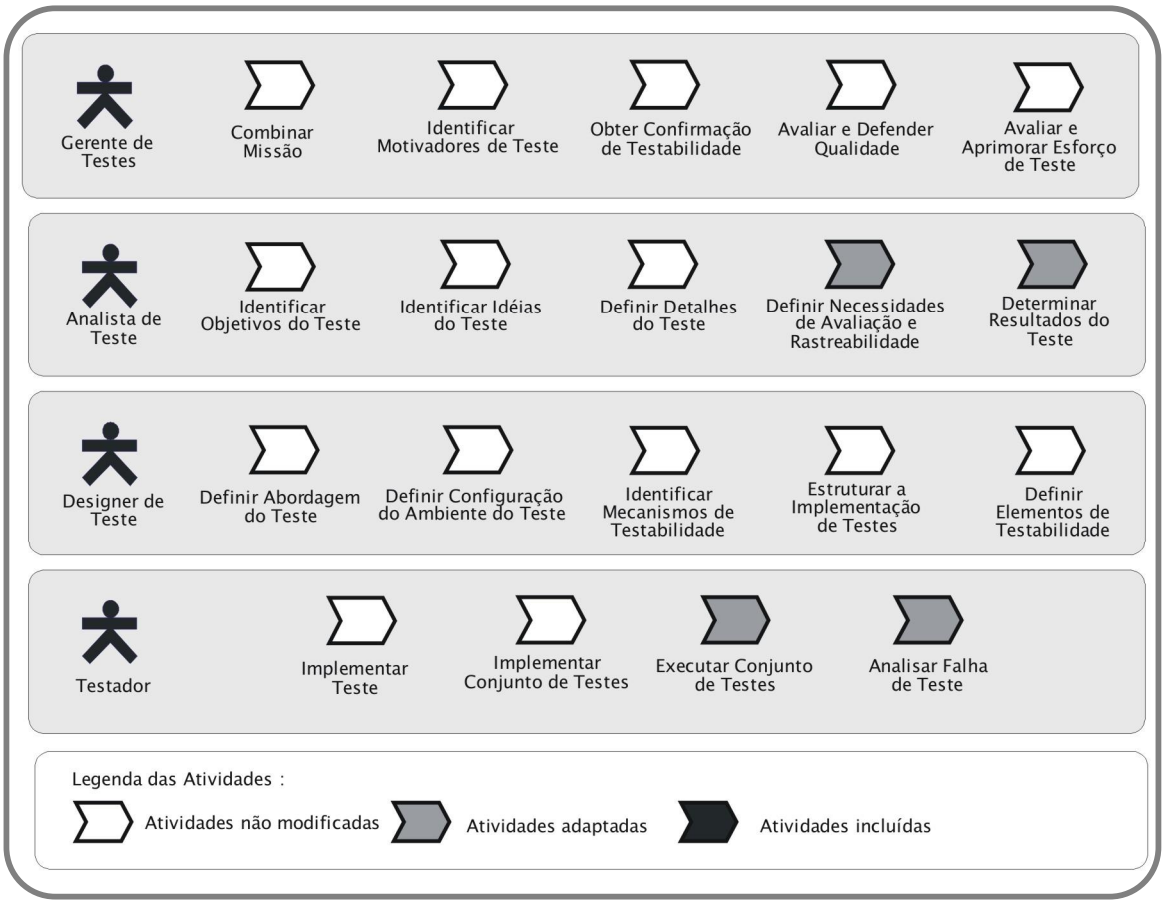



**Figura C-2:** Visão Geral das Atividades do Fluxo de Teste

Nos parágrafos a seguir estão detalhadas apenas as atividades que foram adaptadas. As atividades que foram incluídas foram explicadas no capítulo três (3). As demais atividades e os artefatos, bem como, os conceitos e mentores de ferramentas podem ser encontrados na versão 2003 do *Rational Unified Process*.

**Atividade: Executar Conjunto de Testes**

**Propósito**

Executar o conjunto de testes apropriado necessários para validar a qualidade do produto.

Capturar resultados de teste que facilitem a avaliação do produto.

**Passos**

- Configurar o ambiente de teste para o estado conhecido
- Definir opções para ferramentas de execução
- Planejar a execução do Conjunto de Testes
- Executar o Conjunto de Testes
- Avaliar a execução do Conjunto de Testes
- Recuperar a partir de testes interrompidos
    - Inspecionar os *Logs* de Teste
    - Restaurar o ambiente de teste e as opções das ferramentas
    - Corrigir erros
    - Planejar e executar novamente o Conjunto de Testes
    - Reavaliar a execução do Conjunto de Testes
- Inspecionar os *Logs* de Teste para verificar a abrangência e a precisão
- Restaurar o ambiente de teste ao estado conhecido
- Manter relacionamentos de rastreabilidade
- Avaliar e verificar seus resultados

| **Artefatos de Entrada:** | **Artefatos Resultantes:** |
|---|---|
| ▪ *Build* Instrumentada<br>▪ Componente de Teste<br>▪ Script de Teste<br>▪ Subsistema de Implementação<br>▪ Conjunto de Testes<br>▪ Plano de Teste<br>▪ Dados de Teste<br>▪ Configuração do Ambiente de Teste | ▪ *Log* de Testes<br>▪ Configuração do Ambiente de Teste<br>▪ Arquivo de Histórico da Execução do Teste |

| |
|---|
| **Freqüência:** Esta atividade é normalmente executada várias vezes por iteração. |
| **Responsável:** Testador |

A seguir é apresentada a descrição dos passos da atividade Executar Conjunto de Testes, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Configurar o ambiente de teste para o estado conhecido**

Configurar e estabelecer o ambiente de teste para assegurar que todos os componentes necessários (hardware, software, ferramentas, dados etc.) tenham sido implementados, esteja pronto no ambiente de teste e no estado correto para garantir a realização dos testes. Para a realização do *Code Coverage Process*, é necessário disponibilizar uma *build* instrumentada, para que se possa executar o conjunto de testes.

**Definir opções para ferramentas de execução**

Configurar apropriadamente as ferramentas usadas na execução do Conjunto de Testes. Dependendo do nível de sofisticação das ferramentas, várias opções podem ser usadas. A não definição apropriada dessas opções poderá reduzir a utilidade e o valor dos *Logs* de Teste e das outras saídas resultantes.

Dependendo da forma que é gerado o relatório de cobertura, é disponibilizado no ambiente e o *guideline* para a utilização da ferramenta de cobertura.

**Planejar a execução do Conjunto de Testes**

Este passo tem como finalidade determinar a hora apropriada para iniciar a execução do teste.

Em alguns casos em que no acompanhamento da execução do teste, o Conjunto de Testes poderá ser executado sob demanda. Nesses casos, a programação talvez necessite considerar as redefinições do ambiente e o trabalho dos outros testadores, bem como as diferentes equipes de teste e os outros membros da equipe que compartilham o ambiente de teste.

Contudo, nos casos em que a execução autônoma dos testes automatizados for necessária ou quando a execução de vários testes executados simultaneamente em máquinas diferentes tiver que ser planejada, algum mecanismo de programação automatizada poderá ser necessária.

Tendo em posse as informações referentes à cobertura do código em relação a iterações anteriores ou do projeto como um todo, relacionadas à execução de cada caso de teste no código, pode-se utilizar esta informação para selecionar cada caso de teste do conjunto mediante o nível de cobertura que o mesmo tem sobre o código, resultando na execução de menos casos de testes, mas permanecendo com o nível cobertura especificada.

Com a cobertura de código, pode-se reduzir a quantidade de casos de testes que têm o mesmo efeito na cobertura, ou seja, eliminar casos redundantes, mas sem perder a qualidade da cobertura.

**Executar o Conjunto de Testes**

Este passo tem a finalidade de conduzir os testes incluídos no Conjunto de Testes, monitorando e dando suporte à sua conclusão.

A execução dos Conjuntos de Testes e dos Scripts de Teste varia dependendo se os testes são, automatizados ou manuais.

- Teste automático: Os scripts de teste criados durante a atividade Implementar Teste são executados.
- Execução manual: Os Scripts de Teste estruturados e desenvolvidos durante a atividade Projetar Teste são usados para executar manualmente o teste.

Algumas ferramentas de cobertura geram um arquivo de *Log*, com isso à medida que for executado o teste é gerado um arquivo de *Log* contendo o histórico da execução, para em seguida gerar o relatório de cobertura. Com isso tem que se ter o cuidado no momento da execução para que não se perca informações no arquivo de histórico. No nosso estudo de caso, utilizamos uma ferramenta que gera um arquivo com as informações do histórico da execução do teste.

**Avaliar a execução do Conjunto de Testes**

Este passo tem a finalidade de determinar se o Conjunto de Testes foi executado com sucesso e como desejado ou se foi interrompido de forma anormal (avaliando neste caso se há necessidade de ação corretiva).

A execução do teste é concluída ou finalizada em duas situações:

- Normal: todos os Scripts de Teste são executados até a conclusão conforme planejado.
- Anormal ou prematura: os Scripts de Teste não são executados completamente conforme planejado. Quando o teste é concluído de forma anormal, é possível que os *Logs* de Teste dos quais os Resultados de Teste são derivados não sejam confiáveis. A causa do término anormal precisa ser averiguada e, se necessário, o erro deve ser corrigido e os testes devem ser novamente executados, bem como, o arquivo com histórico da execução deve ser também analisado, a fim que se possa verificar se foi gerado corretamente.

**Recuperar a partir de testes interrompidos**

Este passo tem a finalidade de determinar a ação corretiva apropriada para a recuperação a partir da execução de um Conjunto de Testes interrompido e, se necessário, corrigir o problema, recuperando e executando novamente o Conjunto de Testes. Se houver algum problema nas informações para geração do relatório de cobertura, se faz necessário também corrigir o problema e executar novamente.

Para uma recuperação a partir dos testes interrompidos, temos os seguintes Sub-passos:

**Inspecionar os *Logs* de Teste**

Inspecione os *Logos* de Teste para verificar a abrangência e a precisão. Identifique os erros ocorridos e inspecione-os. Há dois tipos principais de testes interrompidos:

- Erros fatais – Ocasionadas por falha no sistema (falhas de rede, paralisações de hardware, etc.).

- Falhas do Script de Teste - Ocasionada quando alguma parte de um Script de Teste em um Conjunto de Testes não pode ser executada conforme planejado.

Os dois tipos de ocorrência anormal durante a execução do teste podem expor os seguintes sintomas:

- Um grande número eventos inesperados ocorreu durante a execução do Conjunto de Testes.
- O ambiente de teste parou de responder, está lento ou está em um estado indesejável (por exemplo, suspenso ou paralisado).

Análise os sintomas até conseguir determinar a causa original do problema.

Devido a esses problemas é impedida a criação do arquivo com as informações do histórico da execução do teste.

**Restaurar o ambiente de teste e as opções de ferramenta**

Geralmente, é preciso fazer ajustes no ambiente de teste como parte da correção do problema. Desse modo, recomenda-se restaurar o ambiente novamente a partir de um estado conhecido antes de se efetuar qualquer correção necessária.

Se para executar e gerar o relatório do Conjunto de Testes é necessário a utilização da ferramenta de cobertura, deve-se averiguar se o ambiente da ferramenta está correto.

**Corrigir Erros**

Corrija o Script de Teste, os Dados de Teste e o ambiente de teste. Após efetuar as mudanças necessárias, salve o Script de Teste e as configurações do ambiente de teste conforme necessário.

Sendo utilizada a ferramenta de cobertura para a execução do código, então configure novamente o ambiente da ferramenta, mediante o *guideline* de cobertura.

**Planejar e executar novamente o Conjunto de Testes**

Programe novamente e execute o Conjunto de Testes. Dependendo do processo de recuperação que estiver disponível (se houver algum).

**Reavaliar a execução do Conjunto de Testes**

Confirmar se o Conjunto de Testes é executado até a sua conclusão. Em alguns casos em que são criados procedimentos de recuperação de execução de teste, talvez possa se recuperar a execução do teste a partir de um ponto intermediário. Em se tratando do uso de cobertura de código, é recomendável que se retorne o teste desde o inicio, para que não ocorra perca de informações para posterior geração do relatório de cobertura.

Se os problemas continuarem, análise as subseções de Recuperar a partir de testes interrompidos, até resolver o problema.

**Inspecionar os *Logs* de Teste para verificar a abrangência e a precisão**

Este passo tem a finalidade de determinar se a execução do Conjunto de Testes gerou informações de teste relevantes e, caso contrário, identificar a ação corretiva adequada.

Tendo a execução do teste sido inicialmente concluída, os *Logs* de Teste deverão ser revistos para verificar se são confiáveis e se as falhas relatadas, os avisos ou os resultados inesperados não foram causados por influências externas, como configuração imprópria do ambiente ou Dados de Teste inválidos.

**Restaurar o ambiente de teste ao estado conhecido**

Este passo tem a finalidade de garantir que se fez uma limpeza apropriada do ambiente após a execução do Script de Teste.

**Manter relacionamentos de rastreabilidade**

Este passo tem a finalidade de permitir a análise do impacto e gerar um relatório de avaliação dos itens rastreados.

Usando os requisitos de rastreabilidade descritos no Plano de Teste, atualize os relacionamentos de rastreabilidade conforme necessário.

Faça o rastreamento de cada caso de teste com nível de cobertura que o mesmo alcança, para que em outra oportunidade se possa conhecer e selecionar casos de teste que tenham maior impacto e eficácia na cobertura de código.

**Avaliar e verificar os resultados**

Verificar se a atividade foi concluída de forma apropriada e se os artefatos resultantes são aceitáveis.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do *Rational Unif*ied *Process*.

**Atividade: Analisar Falha de Teste**

<table>
<tr><td colspan="2">Propósito<br>Investigar os detalhes do Log de Teste e analisar as falhas que ocorreram durante a implementação e execução do teste.<br>Corrigir falhas que resultem de erros no procedimento de teste.<br>Registrar descobertas importantes apropriadamente.</td></tr>
<tr><td colspan="2">Passos<br>▪ Examinar os Registros de Teste<br>▪ Capturar dados incidentais não triviais<br>▪ Identificar erros de procedimento no teste<br>▪ Localizar e isolar falhas<br>▪ Diagnosticar sintomas e características de falha<br>▪ Identificar as possíveis soluções<br>▪ Documentar as descobertas apropriadamente<br>▪ Avaliar e verificar os resultados</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Ferramentas<br>▪ Log de Testes<br>▪ Caso de Teste<br>▪ Modelo de Análise de Carga de Trabalho<br>▪ Guia de Teste<br>▪ Dados de Teste<br>▪ Script de Teste<br>▪ Conjunto de Testes<br>▪ Configuração do Ambiente de Teste<br>▪ Relatório de Cobertura</td><td>Artefatos Resultantes:<br>▪ Solicitação de Mudança<br>▪ Script de Teste<br>▪ Conjunto de Testes<br>▪ Configuração do Ambiente de Teste</td></tr>
<tr><td colspan="2">Freqüência: Esta atividade normalmente é executada várias vezes por iteração.</td></tr>
<tr><td colspan="2">Responsável: Testador</td></tr>
</table>

A seguir é apresentada a descrição dos passos da atividade de Analisar Falha de Teste, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Examinar os Registros de Teste**

Tem como finalidade reunir e compreender o resultado dos testes executados. Iniciando com coleta dos resultados dos Registros de Teste durante a implementação e execução dos testes.

**Capturar dados incidentais não triviais**

Tem a finalidade de registrar a ocorrência de quaisquer eventos anômalos e não triviais para fins de investigação. Capture quaisquer ocorrências irregulares, mesmo que não consiga reproduzi-la e explicá-la.

**Identificar erros de procedimento no teste**

É muito comum que várias falhas sejam resultado de erros gerados durante a implementação do teste ou durante o gerenciamento do ambiente de teste. Identifique e corrija esses erros.

**Localizar e isolar falhas**

Faça uma análise da falha reproduzindo-a sob condições controladas, caso a falha não possa ser investigada sem reprodução. Use ferramentas de diagnóstico e depuração quando for necessário.

Algumas ferramentas de cobertura geram relatórios de cobertura que informam o ponto exato que está com ausência de teste. Neste caso, procure analisar o relatório para que se possam criar os casos de teste que estejam faltando ou que não estejam realizando os testes de forma adequada.

Alguns tipos de falhas só podem ser encontrados na fase de teste de unidade, pois a condição para determinado método ou função não é possível de ser reproduzida na fase de Teste.

**Diagnosticar sintomas e características de falha**

Tente diagnosticar a falha subjacente usando a experiência adquirida com incidentes similares que já ocorreram.

Se necessário e disponível, solicite suporte dos desenvolvedores, tirando proveito do conhecimento interno que eles têm do software, a fim de melhorar a análise da falha.

Existem relatórios de cobertura de algumas ferramentas que fornecem informação suficiente para criação de novos casos de teste, para que se possa complementar o Conjunto de Teste, resultando numa melhor cobertura. Essa vantagem serve também para o próximo passo "Identificar as possíveis soluções".

**Identificar as possíveis soluções**

Tem a finalidade de fornecer à pessoa responsável pela resolução da falha um entendimento melhor da natureza e do impacto da falha, além de ajudar os desenvolvedores a fornecer idéias possíveis que podem, opcionalmente, ser adotadas.

**Documentar as descobertas apropriadamente**

Apresentar a análise da falha de maneira adequada para a pessoa responsável pela resolução da falha.

**Avaliar e Verificar os Resultados**

Verificar se a atividade foi concluída de forma apropriada e se os artefatos resultantes são aceitáveis.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do *Rational Unified Process*.

Atividade: Determinar Resultados do Teste

<table>
<tr><td colspan="2">Propósito<br>Fazer avaliações resumidas e contínuas da qualidade do produto.<br>Determinar o Resultado de Teste.<br>Propor ações corretivas apropriadas para resolverem falhas na qualidade.</td></tr>
<tr><td colspan="2">Passos<br>▪ Examinar todos os incidentes e falhas do teste<br>▪ Criar e manter Solicitações de Mudança<br>▪ Analisar e avaliar o status<br>▪ Fazer uma avaliação da situação de qualidade atual<br>▪ Fazer uma avaliação dos riscos de qualidade iminentes<br>▪ Fazer uma avaliação da cobertura de teste<br>▪ Fazer um rascunho do Sumário de Avaliação de Testes<br>▪ Avisar os envolvidos das principais descobertas<br>▪ Avaliar e verificar os resultados</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Caso de Teste<br>▪ Plano de Teste<br>▪ Log de Teste<br>▪ Modelo de Análise da Carga de Trabalho<br>▪ Lista de Idéias de Teste<br>▪ Solicitação de Mudança</td><td>Artefatos Resultantes:<br>▪ Sumário de Avaliação de Testes<br>▪ Resultados do Teste<br>▪ Solicitação de Mudança</td></tr>
<tr><td colspan="2">Freqüência: Normalmente, esta atividade é executada várias vezes em cada iteração.</td></tr>
<tr><td colspan="2">Responsável: Analista de Teste</td></tr>
</table>

A seguir é apresentada a descrição dos passos da atividade Determinar Resultados do Teste, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Examinar todos os incidentes e falhas do teste**

Nesta atividade, os *Logs* de Teste são analisados para determinar os Resultados de Teste significativos em relação às diferenças entre os resultados esperados e os resultados reais de cada teste. Identifique e análise cada incidente e falha sucessivamente.

Com o relatório de cobertura, podemos examinar os pontos que não estão sendo testados.

**Criar e manter Solicitações de Mudança**

Inserir informações de solicitação de mudança em uma ferramenta de controle para avaliação, gerenciamento e resolução.

As solicitações de mudança referente à correção do código e/ou casos de teste, devem ser acompanhadas do relatório de cobertura, com intuito de ter um controle e gerenciamento da cobertura relacionada ao componente e ao projeto.

**Analisar e avaliar o status**

Tem a finalidade de calcular e fornecer as principais medidas e os indicadores do teste.

**Fazer uma avaliação da situação de qualidade atual**

Fornece um *feedback* sobre a qualidade percebida ou experimentada em relação ao produto de software. Tanto a gerência de projeto, como a gerencia de qualidade devem ser informadas sobre a situação atual dos níveis de cobertura alcançados, para que ambos possam monitorar e acompanhar a evolução da qualidade dos testes produzidos.

**Fazer uma avaliação dos riscos de qualidade iminentes**

Fornecer *feedback* sobre que outras áreas de risco possivelmente colocam em perigo o projeto.

**Fazer uma avaliação da cobertura de teste**

O RUP sugere fazer uma avaliação resumida da análise de cobertura do teste.

**Fazer um Rascunho do Sumário de Avaliação de Testes**

Comunicar os resultados do teste aos envolvidos e fazer uma avaliação objetiva da qualidade e do status do teste.

**Avisar os envolvidos das principais descobertas**

Promover e divulgar o Sumário de Avaliação conforme apropriado. Todos os envolvidos com a utilização da análise de cobertura de código devem ser informados dos resultados alcançados.

**Avaliar e verificar os resultados**

Verificar se a atividade foi concluída de forma apropriada e se os artefatos resultantes são aceitáveis.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do *Rational Unified Process*.

Atividade: Definir Necessidades de Avaliação e Rastreabilidade

<table>
<tr><td colspan="2">Propósito<br>Definir a estratégia de avaliação para o esforço de teste.<br>Definir os requisitos de rastreabilidade e cobertura.</td></tr>
<tr><td colspan="2">Passos<br>▪ Identificar os requisitos de avaliação e rastreabilidade<br>▪ Considerar as restrições<br>▪ Considerar as estratégias possíveis<br>▪ Discutir possíveis estratégias com os envolvidos<br>▪ Definir e chegar a um acordo em relação à estratégia de avaliação<br>▪ Definir os requisitos de ferramenta<br>▪ Avaliar e verificar os resultados</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Plano de Garantia de Qualidade<br>▪ Plano de Desenvolvimento de Software<br>▪ Plano de Iteração<br>▪ Plano de Gerenciamento de Requisitos<br>▪ Plano de Gerenciamento de Configuração<br>▪ Plano de Teste<br>▪ Guia de Teste</td><td>Artefatos Resultantes:<br>▪ Plano de Teste<br>▪ Guia de Teste</td></tr>
<tr><td colspan="2">Freqüência: Normalmente, esta atividade é executada várias vezes em cada iteração.</td></tr>
<tr><td colspan="2">Responsável: Analista de Teste</td></tr>
</table>

A seguir é apresentada a descrição dos passos da atividade de Definir Necessidades de Avaliação e Rastreabilidade, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Identificar os requisitos de avaliação e rastreabilidade**

Entender os produtos liberados para o processo de avaliação de software e identificar os requisitos associados.

**Considerar as restrições**

Identificar as restrições que afetarão a capacidade (ou a necessidade) de implementação dos requisitos.

**Considerar as estratégias possíveis**

Identificar e descrever uma ou mais estratégias que facilitarão o processo de avaliação necessário.

**Subtópicos:**

- Análise da Cobertura de Teste
- Análise dos Resultados de Teste
- Análise de Defeitos

**Análise da Cobertura de Teste**

As métricas de cobertura de teste e sua análise também convêm ser consideradas junto à análise de defeitos ou à análise de resultados de teste. Neste trabalho estamos focando em cobertura de código.

**Análise dos Resultados de Teste**

Algumas organizações utilizam para a análise dos resultados de teste apenas o número de resultados positivos ou negativos como uma porcentagem do número total de testes executados.

**Análise de Defeitos**

Tendo os defeitos claramente relacionados aos resultados do esforço de teste, as análises dos dados de defeito não garantem informações úteis sobre o andamento do

esforço de teste ou sobre a abrangência desse esforço. Algumas organizações através das equipes de teste e do gerente de projeto têm cometido o erro de usar a contagem de defeitos atual para medir o andamento do teste ou como medida padrão da qualidade do software desenvolvido.

**Discutir possíveis estratégias com os envolvidos**

Obter *feedback* dos envolvidos e fazer ajustes das estratégicas conforme necessário. Buscar melhorar sempre as métricas de cobertura para ter um produto e testes com alta qualidade.

**Definir e chegar a um acordo em relação à estratégia de avaliação**

Obter a aprovação dos envolvidos sobre a estratégia a ser usada. Use o *feedback* obtido nas discussões e refine a estratégia de avaliação para uma única estratégia que atenda às necessidades de todos os envolvidos.

A utilização da cobertura de código no fluxo de Teste faz necessário ter o acompanhamento da equipe de desenvolvimento, pois as métricas são geradas por unidade de código. Logo alguns resultados serão apenas entendidos com o suporte do time de desenvolvimento.

**Definir os requisitos de ferramenta**

Definir os requisitos de configuração da ferramenta que permitirão o processo de avaliação. A escolha da ferramenta de cobertura é algo muito importante, pois existem diversas ferramentas no mercado, que geram tipos de métricas e relatórios diferentes. No nosso estudo de caso, tivemos que realizar um trabalho de seleção bem detalhado, com intuito de atender a necessidade da organização no qual trabalhamos.

**Avaliar e verificar os resultados**

Verificar se a atividade foi concluída de forma apropriada e se os artefatos resultantes são aceitáveis.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do *Rational Unified Process*.

**Fluxo de Gerenciamento de Configuração e Mudança**

No "Gerenciamento de Configuração e Mudança", toda atividade ligada à cobertura de código necessitará de um suporte e um controle dos artefatos gerados para a medição da cobertura, desde a coleta do código dentro do sistema de controle de versão até o fechamento da CR (*Change Request*) de implementação ou modificação do código. A Figura C-3 apresenta a visão geral das atividades do Fluxo de Gerenciamento de Configuração e Mudança.

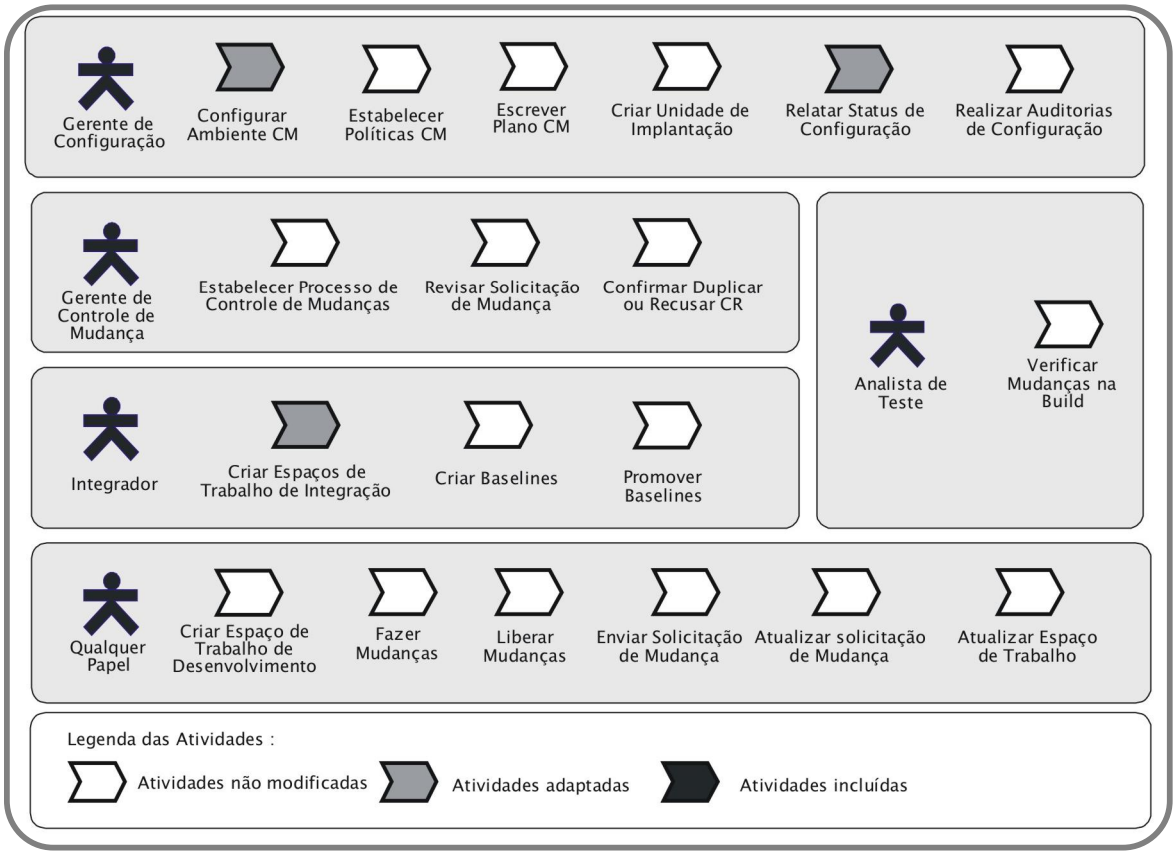



**Figura C-3:** Visão Geral das Atividades do Fluxo de Gerenciamento de Configuração e Mudança

Nos parágrafos a seguir estão detalhadas apenas as atividades que foram adaptadas. As atividades que foram incluídas foram explicadas no capítulo três (3). As demais atividades e os artefatos, bem como, os conceitos e mentores de ferramentas podem ser encontrados na versão 2003 do *Rational Unified Process*.

**Atividade: Configurar Ambiente do Gerenciamento de Configuração (GC)**

<table>
<tr><td colspan="2">Propósito<br>▪ O Propósito desta atividade é estabelecer um ambiente em que o produto possa ser desenvolvido e compilado. Isso é feito em duas partes: primeiro, configurando o ambiente do hardware e, em seguida, estabelecendo o ambiente de desenvolvimento.<br>▪ Definir o ambiente de Gerenciamento de Configuração envolve alocar recursos de máquinas (servidores e espaço em disco) e instalar as ferramentas de gerenciamento de configuração.<br>▪ Definir o ambiente de desenvolvimento envolve criar repositórios, definir a estrutura de diretórios do produto e importar todos os arquivos existentes. O ambiente inicial serve como uma baseline para o aprofundamento do trabalho de desenvolvimento.</td></tr>
<tr><td colspan="2">Passos<br>▪ Definir o Ambiente de Hardware do GC<br>▪ Mapear a Arquitetura para o Repositório<br>▪ Criar Conjunto Inicial de Elementos Versionados<br>▪ Definir Níveis de Baseline</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Modelo de Implementação<br>▪ Plano de Gerenciamento de Configuração</td><td>Artefatos Resultantes:<br>▪ Repositório do Projeto</td></tr>
<tr><td colspan="2">Responsável: Gerente de Configuração</td></tr>
</table>

A seguir é apresentada a descrição dos passos da atividade Configurar Ambiente do Gerenciamento de Configuração (GC), tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Definir o Ambiente de Hardware de Gerencia de Configuração**

Alocar os recursos de hardware necessários para instalar e configurar a Ferramenta de GC. O Gerente de Configuração trabalha com o Administrador do Sistema para alocar os recursos de máquinas e instalar os softwares necessários.

**Mapear a Arquitetura para o Repositório**

A Estrutura de Diretórios do Produto é organizada de modo lógico para assegurar um espaço reservado para todos os artefatos relacionados ao projeto. Crie um espaço reservado para cada componente que precisa ser implementado na estrutura de diretórios. Calcule os requisitos de armazenamento para os artefatos que serão desenvolvidos, e certifique-se de que haverá espaço suficiente para armazenamento.

Para a utilização do *Code Coverage Process*, se faz necessário disponibilizar a estrutura de diretórios e de arquivos, pois o código será instrumentado, bem como novos artefatos irão surgir (Por exemplo: *build* com Instrumentação, *guideline* de cobertura e relatórios).

**Criar Conjunto Inicial de Elementos Versionados**

Criar uma *baseline* inicial dos artefatos do projeto.

**Definir Níveis de *Baseline***

Uma *baseline* é uma versão única do repositório do projeto. A qualidade ou o status dessas *baseline* é indicado pelo nível. As *baselines* são lineares e podem ser promovidas ou rebaixadas de nível, de acordo com a qualidade dos itens.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do *Rational Unified Process*.

**Atividade: Criar Espaços de Trabalho de Integração**

<table>
<tr><td colspan="2">Propósito<br>A noção de espaço de trabalho de integração é semelhante à de “espaço de trabalho privado” dos implementadores, no qual as pessoas podem desenvolver e alterar o código em uma área restrita. O espaço de trabalho de integração é onde integradores de sistema e subsistema confirmam que os componentes testados e desenvolvidos separadamente podem ser construídos para que trabalhem juntos como um produto.</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Repositório do Projeto</td><td>Artefatos Resultantes:<br>▪ Espaço de Trabalho</td></tr>
<tr><td colspan="2">Freqüência: Contínua, como parte do Processo de Desenvolvimento Iterativo.</td></tr>
<tr><td colspan="2">Responsável: Integrador</td></tr>
</table>

A seguir é apresentada a descrição dos passos da atividade de Criar Espaços de Trabalho de Integração, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

Os implementadores desenvolvem componentes de acordo com o que foi definido no projeto, e passam esses componentes dos espaços de trabalho de desenvolvimento para o espaço de trabalho de integração do projeto.

Os integradores combinam os componentes liberados no espaço de trabalho de integração para produzir uma *build*. Ao criar uma nova *baseline*, o integrador precisa "bloquear" o espaço de trabalho de integração para garantir que haja um conjunto estático de arquivos e que novos arquivos não sejam liberados pelos desenvolvedores.

Há dois tipos de visão que podem ser associadas ao espaço de trabalho de integração, a visão dinâmica e a estática.

- Visões dinâmicas oferecem acesso imediato e transparente aos arquivos e diretórios contidos no repositório do projeto.

- Visão de integração deve ser uma visão dinâmica. Ela garante que o integrador tenha a última versão dos arquivos e diretórios liberados pelos desenvolvedores.

Para utilização de Cobertura de Código se faz necessário ter acesso aos arquivos, pois estaremos trabalhando diretamente com os mesmos, a fim de instrumentá-los para posterior teste e geração do relatório de cobertura.

O *guideline* de cobertura deverá conter informações de como utilizar a ferramenta de cobertura dentro do ambiente de desenvolvimento, podendo o artefato ter pontos específicos relacionados a cada projeto, em virtude de particularidades dos mesmos.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do *Rational Unified Process*.

**Atividade: Relatar Status de Configuração**

<table>
<tr><td colspan="2">Propósito<br>▪ Auxiliar as atividades de Configuração do projeto que são baseadas em um registro formalizado e relatar o status das mudanças propostas assim como o status de implementação das mudanças propostas.<br>▪ Facilitar a revisão do produto por meio de atividades de emissão de relatórios e controle de defeitos.</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Plano de Gerenciamento de Configuração<br>▪ Repositório do Projeto<br>▪ Guideline de cobertura</td><td>Artefatos Resultantes:<br>▪ Métricas de Projeto</td></tr>
<tr><td colspan="2">Responsável: Gerente de Configuração</td></tr>
</table>

Para o uso da ferramenta de cobertura se faz necessário a sua integração com o ambiente de desenvolvimento, no qual é utilizado o código do componente e os testes de unidade (demais testes de software são tratados no fluxo de Testes). Para que se tenha um melhor entendimento do uso da ferramenta aplicada ao contexto do ambiente de trabalho, é criado um *guideline* de cobertura, que tem o propósito de auxiliar os responsáveis pelo manuseio da ferramenta adequadamente. Mesmo que exista um manual de como manusear a ferramenta é necessário criar esse documento, a fim de que se tenha o procedimento específico relacionado ao ambiente de trabalho.

Qualquer alteração no *guideline*, devido à mudança ocorrida na configuração do ambiente para o uso da ferramenta de cobertura, deve ser informada.

## Fluxo de Gerenciamento de Projeto

O fluxo "Gerenciamento de Projeto" tem a finalidade de analisar as métricas coletadas através dos relatórios gerados pela ferramenta de cobertura, bem como tomar posicionamento e diretrizes para que a qualidade dos testes melhore, ou seja, que as métricas coletadas não fiquem abaixo do nível de cobertura estabelecido pela organização. A Figura C-4 apresenta a visão geral das atividades do Fluxo de Gerenciamento de Projeto.

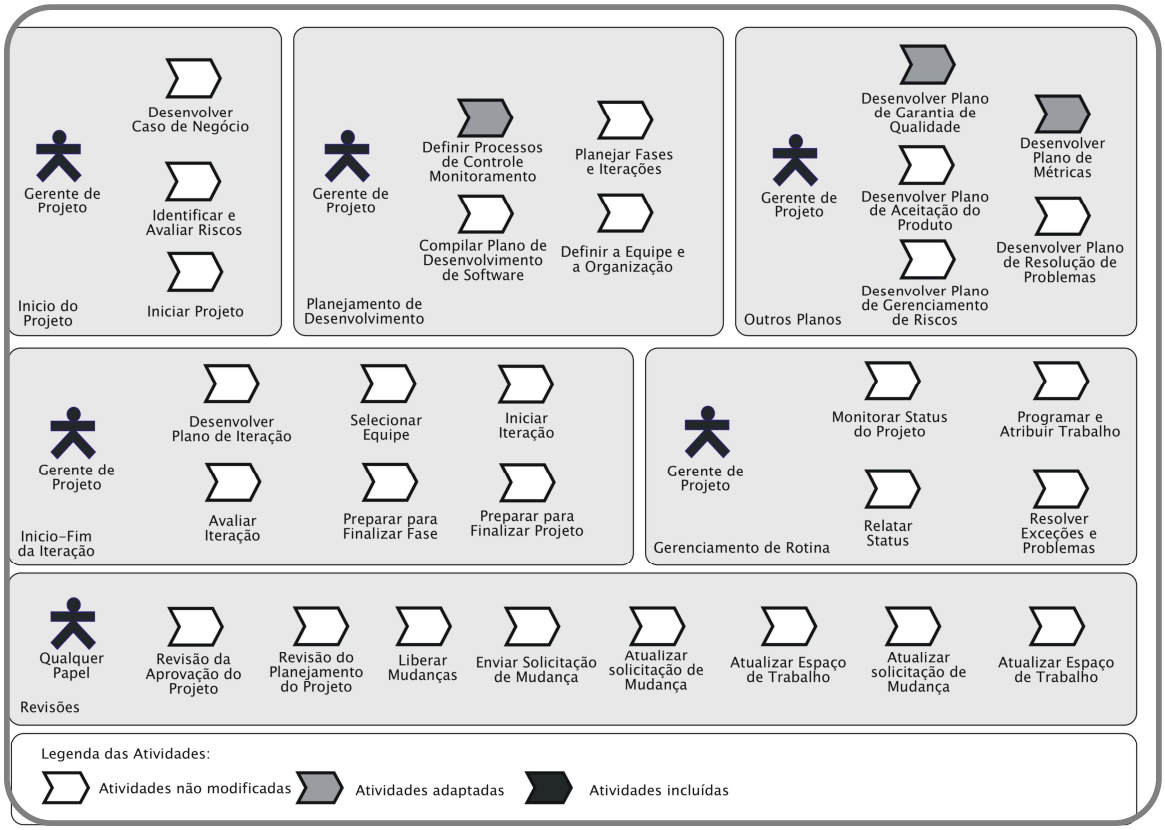



**Figura C-4:** Visão Geral das Atividades do Fluxo de Gerenciamento de Projeto

Nos parágrafos a seguir estão detalhadas apenas as atividades que foram adaptadas. As atividades que foram incluídas foram explicadas no capítulo três (3). As demais atividades e os artefatos, bem como, os conceitos e mentores de ferramentas podem ser encontrados na versão 2003 do *Rational Unified Process*.

**Atividade: Desenvolver Plano de Métricas**

<table>
<tr><td colspan="2">Propósito<br>Definir metas de gerenciamento, em termos de qualidade, progresso e melhoria.<br>Determinar o que precisa ser medido periodicamente para suportar essas metas.</td></tr>
<tr><td colspan="2">Passos<br>▪ Definir as Principais Metas de Gerenciamento<br>▪ Validar as Metas<br>▪ Definir as Submetas<br>▪ Identificar as Métricas Necessárias para Satisfazer às Submetas<br>▪ Identificar as Métricas Primitivas Necessárias para Calcular as Métricas<br>▪ Elaborar o Plano de Métricas<br>▪ Avaliar o Plano de Métricas<br>▪ Implementar os Mecanismos de Coleta</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Lista de Riscos<br>▪ Caso de Negócio<br>▪ Visão</td><td>Artefatos Resultantes:<br>▪ Métricas de Projeto<br>▪ Plano de Métricas</td></tr>
<tr><td colspan="2">Freqüência: Uma vez por projeto (atualizado a cada iteração, se necessário).</td></tr>
<tr><td colspan="2">Responsável: Gerente de Projeto</td></tr>
</table>

O Plano de Métricas descreve as metas que o projeto terá que cumprir para ser concluído com sucesso, bem como as medidas e as métricas a serem usadas para determinar se ele está no caminho correto.

A atividade Desenvolver Plano de Métricas é executada uma vez por projeto, na fase de iniciação, como parte da atividade de planejamento geral. O plano de métricas pode ser revisado, como qualquer outra parte do plano de desenvolvimento de software, durante o projeto.

A seguir é apresentada a descrição dos passos da atividade de Desenvolver Plano de Métricas, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Definir as Principais Metas de Gerenciamento**

Este passo tem a finalidade de determinar e registrar as restrições e os requisitos funcionais, não-funcionais, orçamentários e de programação importantes que precisam ser controlados.

No *Code Coverage Process*, o gerente de projeto tem que determinar o limiar de cobertura desejável que os testes devem alcançar, para isso o gerente pode contar com a participação de membros da equipe de desenvolvimento. Deve também determinar que tipo de métricas de cobertura deseja coletar no projeto.

**Validar as Metas**

O Gerente de Projeto deve revisar as metas selecionadas juntamente, com os principais envolvidos, para certificar-se de que o foco delas esteja adequado, que todas as áreas de interesse e risco sejam consideradas, que seja possível reduzir as metas em métricas coletáveis.

A determinação das métricas de cobertura deve ser revisada e os limites de cobertura revistos, a fim que se tenham condições reais de atingir os objetivos.

**Definir as Submetas**

Analisar metas complexas para determinar submetas às quais é possível aplicar métricas.

**Identificar as Métricas Necessárias para Satisfazer as Submetas**

Determinar as métricas que permitirão o controle das submetas.

**Identificar as Métricas Primitivas Necessárias para Calcular as Métricas**

Determinar as medições básicas das quais as métricas serão obtidas.

**Elaborar o Plano de Métricas**

Este passo tem a finalidade de elaborar o Plano de Métricas que irá capturar as metas, submetas.

**Avaliar o Plano de Métricas**

Verificar se o Plano de Métricas está consistente, claro, apropriado, viável e completo.

**Implementar os Mecanismos de Coleta**

Estabelecer a forma de coletar, registrar, reduzir e relatar as medições planejadas.

O gerente de projeto deve determinar para a equipe de desenvolvimento que seja feita a entrega dos relatórios de cobertura no momento da entrega de uma atividade de implementação ou manutenção de código do componente ou dos testes de unidade. O relatório acompanhará o documento de Solicitação de Mudança.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do Rational *Unified Process*.

**Atividade: Desenvolver Plano de Garantia de Qualidade**

<table>
<tr><td colspan="2">Propósito<br>Criar um plano documentado para as atividades de garantia de qualidade no projeto.</td></tr>
<tr><td colspan="2">Passos<br>▪ Verificar se Objetivos de Qualidade estão Definidos para o Projeto<br>▪ Definir Responsabilidades e Papéis para Garantia de Qualidade<br>▪ Trabalhar de Forma Coordenada com os Desenvolvedores dos Planos Referenciados<br>▪ Definir Programação e Tarefas para Garantia de Qualidade</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Plano de Desenvolvimento de Software<br>▪ Plano de Gerenciamento de Configuração<br>▪ Plano de Resolução de Problemas<br>▪ Plano de Gerenciamento de Riscos<br>▪ Plano de Métricas<br>▪ Caso de Negócio<br>▪ Visão<br>▪ Plano de Teste</td><td>Artefatos Resultantes:<br>▪ Plano de Garantia de Qualidade</td></tr>
<tr><td colspan="2">Responsável: Gerente de Projeto</td></tr>
</table>

O Plano de Garantia de Qualidade é composto de todas as informações relacionadas às atividades da garantia de qualidade do projeto. Embora grande parte das informações referenciadas por esse Plano também seja referenciada no Plano de Desenvolvimento de Software, ainda é importante desenvolver os dois planos porque apresentam finalidades distintas.

O Plano de Garantia de Qualidade é usado para elaborar um programa de revisões e auditorias que verificará se o processo do projeto definido está sendo seguido corretamente, da forma como foi definido pelos diversos planos de suporte aos quais faz referência.

Nesta atividade, o gerente de projeto define e/ou revisa o programa de Garantia de Qualidade quanto à adequação e à aceitabilidade, e trabalha de forma coordenada com os desenvolvedores dos planos referenciados.

A seguir é apresentada a descrição dos passos da atividade Desenvolver Plano de Garantia de Qualidade, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Verificar se Objetivos de Qualidade estão Definidos para o Projeto**

A organização de desenvolvimento também pode ter um conjunto padrão de metas de qualidade, em uma descrição de políticas de qualidade, que pode formar a base dessas definições.

No *Code Coverage Process*, as metas de cobertura devem estar especificadas e todos os envolvidos cientes dos compromissos que deverão realizar. A qualquer sinal de impossibilidade de não atingir os objetivos estabelecidos, os membros deverão imediatamente se pronunciar para os gerentes de projeto e de qualidade.

**Definir Responsabilidades e Papéis para Garantia de Qualidade**

O passo seguinte é estabelecer a organização, os papéis e as responsabilidades que participarão das atividades de garantia de qualidade. Os relatórios de cobertura deverão ser gerados pelo implementador, e no caso de estarem sendo realizados testes “Caixa Preta”, essa responsabilidade pode ser atribuída ao testador, dependendo da forma que a ferramenta de cobertura gere os relatórios, pois poderá ser gerado de forma automática ou dependerá de dados e configurações que o testador não tenha conhecimento suficiente para realizar.

A análise das informações das métricas de cobertura deverá ser realizada pelos principais envolvidos: gerente de projeto, implementadores e testadores. Ficando o gerente de qualidade responsável por averiguar se comprometimento estabelecido no processo está sendo seguido e principalmente se medidas estão sendo tomadas para a solução de possíveis problemas relacionados com a não realização de atividades e metas do processo de cobertura de código.

**Trabalhar de Forma Coordenada com os Desenvolvedores dos Planos Referenciados**

O Plano de Garantia de Qualidade também faz referência a outros planos que descrevem padrões do projeto e vários processos de suporte. Essas informações auxiliam a determinar os tipos de revisão de Garantia de Qualidade que serão realizadas e a respectiva freqüência. Os planos referenciados normalmente incluem:

- Plano de Documentação
- Plano de Métricas
- Plano de Gerenciamento de Riscos
- Plano de Resolução de Problemas
- Plano de Gerenciamento de Configuração
- Plano de Desenvolvimento de Software
- Plano de Teste
- Plano de Gerenciamento de Subfornecedores

**Definir Programação e Tarefas para Garantia de Qualidade**

Identifique as tarefas e as atividades de Garantia de Qualidade. Em geral, estas tarefas incluem:

- Auditoria e/ou revisão de planos do projeto para verificar se seguem o processo definido para o projeto.
- Auditoria e/ou revisão do projeto para verificar se o trabalho realizado está seguindo os planos do projeto.
- Aprovação de desvios dos processos organizacionais padrão do projeto.
- Avaliações de melhoria de processos.

A descrição em detalhes dos passos desta atividade, bem como as descrições dos artefatos utilizados pode ser encontrada na versão 2003 do Rational *Unified Process*.

**Atividade: Definir Processos de Controle e Monitoramento**

<table>
<tr><td colspan="2">Propósito<br>Definir as informações e os processos que serão usados para monitorar e controlar o andamento, a qualidade e os riscos do projeto.</td></tr>
<tr><td colspan="2">Passos<br>▪ Definir "indicadores" para o projeto<br>▪ Definir origens para indicadores do projeto<br>▪ Definir procedimento para elaboração do relatório de status da equipe<br>▪ Definir procedimento e limites para ação corretiva<br>▪ Definir procedimento para elaboração do relatório de status do projeto</td></tr>
<tr><td>Artefatos de Entrada:<br>▪ Plano de Gerenciamento de Riscos<br>▪ Plano de Desenvolvimento de Software</td><td>Artefatos Resultantes:<br>▪ Plano de Métricas<br>▪ Plano de Desenvolvimento de Software</td></tr>
<tr><td colspan="2">Responsável: Gerente de Projeto</td></tr>
</table>

A seguir é apresentada a descrição dos passos da atividade Definir Processos de Controle e Monitoramento, tendo em destaque as adaptações referentes à utilização do *Code Coverage Process*.

**Definir "indicadores" para o projeto**

Os "indicadores" do projeto são conjuntos de informações que retratam o seu andamento no plano de desenvolvimento de software. Em geral, um gerente de projeto está interessado nos indicadores que se aplicam ao escopo de trabalho, ao orçamento, à qualidade e aos riscos do projeto.

A maior dificuldade em definir métricas de cobertura é determinar o nível de cobertura aceitável para cada métrica, para que se tenha uma melhor interpretação dos resultados. Assim os indicadores de cobertura como base para análise da qualidade dos casos de testes gerados.

**Definir origens para indicadores do projeto**

Geralmente, os indicadores do projeto são medidas de projeto consolidadas, calculadas a partir de métricas primitivas mais granulares que são relatadas regularmente pela equipe do projeto. A forma como essas métricas primitivas devem ser capturadas e o processo para usá-las a fim de calcular os indicadores do projeto são definidos no Plano de Métricas do projeto.

As melhores fontes de informação, para decidir os limites para os indicadores, são os próprios dados da organização, pois os processos podem variar de organização para organização e muitas métricas não possuem critérios e processos bem definidos. Por esse motivo, comparar as métricas obtidas de organizações diferentes pode levar a interpretações erradas.

É aconselhável que se utilizem os dados da própria organização para se determinar o nível aceitável de cobertura, principalmente utilizando bases históricas do projeto que se deseja analisar a cobertura, realizando estas análises retroativamente, tendo como intuito obter um nível desejável de cobertura. Se não houver condições de realizar esse tipo de trabalho, espera-se que, com a execução de novos projetos, forme-se uma base histórica.

**Definir procedimento para elaboração do relatório de status da equipe**

Após a definição das métricas primitivas e dos indicadores do projeto, deve-se definir o procedimento que os membros da equipe do projeto deverão seguir para relatar o status e a freqüência com que deverão elaborar relatórios. Esse procedimento deverá descrever o processo para registrar o tempo nos pacotes de trabalho, informar a conclusão de tarefas, atingir os marcos do projeto e relatar problemas. Para garantir um fluxo de informações consistente, convém definir *templates* padrão para quadros de horários e relatórios de status de membros da equipe.

**Definir procedimento e limites para ação corretiva**

Quando os limites forem ultrapassados, o gerente de projeto deverá realizar uma ação corretiva para retomar o controle do projeto. Dependendo da gravidade da condição, ele poderá resolver o problema com sua própria autoridade (emitindo as ordens de trabalho

apropriadas). Se o problema estiver além da sua autoridade, ele precisará emitir uma Solicitação de Mudança e ativar o processo de controle de mudança do projeto.

**Definir procedimento para elaboração do relatório de status do projeto**

O Plano de Elaboração de Relatórios do Plano de Desenvolvimento de Software também deverá descrever o procedimento que o gerente de projeto deve seguir para informar o andamento do projeto, bem como a freqüência com que ele deverá fazer isso.

# Apêndice D - Questionário de Análise Crítica o *Code Coverage Process*

## IDENTIFICAÇÃO DE PERFIL DO ENTREVISTADO

**1 Natureza do papel que desempenha atualmente:**

☐ Engenharia (Pessoa envolvida com aspectos técnicos do desenvolvimento)
☐ Testes (Pessoa envolvida com testes de software)
☐ Qualidade (Pessoa envolvida com a garantia da qualidade e melhoria de processos)
☐ Gerência (Pessoa envolvida com gerenciamento de prazo, escopo, custos e riscos).

**1.1 Experiência Profissional:**

☐ Nenhuma
☐ 1 ano
☐ 2 a 3 anos
☐ 4 a 5 anos
☐ Mais de 5 anos

**1.2 Conhecimento em processo de software** (Ex. XP, RUP).

☐ Pouco
☐ Médio
☐ Extenso

**1.3 Conhecimento em testes de software** (Ex. *Unit Tests, Feature Test*).

☐ Pouco
☐ Médio
☐ Extenso

# ANÁLISE CRÍTICA DO *CODE COVERAGE PROCESS*

Nesta etapa, são descritos perguntas com intuito de verificar a adequação aos objetivos que o *Code Coverage Process* se propõe a alcançar. Desta forma para responder tais questionamentos deve ser feita uma comparação entre o *Code Coverage Process* e os objetivos a serem alcançados que estão descritos neste documento.

## 1. Processo de Software

**Objetivos a serem alçados**

Conjunto de etapas, formadas por atividades, procedimentos e práticas utilizadas para atingir um determinado objetivo. Geralmente este objetivo está associado a um ou mais resultados e/ou produtos derivado da sua execução. O processo de desenvolvimento de software possui os seguintes objetivos:

- Prover direção sobre a ordem das atividades do time;
- Especificar quais artefatos deve ser desenvolvido;
- Direcionar as tarefas de desenvolvedores e do time como um todo;
- Oferecer critérios para monitorar os produtos do projeto e das atividades.

**Questionamentos relacionados**

- ***Code Coverage Process*** (Atividades, responsáveis e artefatos gerados).

  **a. Em relação à contribuição aos objetivos acima citados.**

  ☐ Contribui fortemente
  ☐ Contribui em partes
  ☐ Contribui fracamente
  ☐ Não contribui

  **b. Em relação à sua aplicabilidade aos tipos de metodologias de desenvolvimento de software. É possível dizer que pode ser utilizado em:**

  ☐ Metodologias ágeis
  ☐ Metodologias tradicionais
  ☐ Ambos os tipos
  ☐ Nenhuma deles

- **Seleção da Ferramenta** (Levantamento dos requisitos da organização, ciclos de análises das ferramentas e aplicação de projeto piloto).

**a) Em relação à contribuição aos objetivos acima citados.**

☐ Contribui fortemente
☐ Contribui em partes
☐ Contribui fracamente
☐ Não contribui

# 2. Teste de Software

**Objetivos a serem alçados**

O processo de teste tem como objetivo criar testes para revelar a presença de erros. Ele define como os testes serão planejados e executados através de atividades e passos, e quando serão executados. Quando o processo é bem controlado e planejado, exige menos esforço e tem maior eficácia.

**Questionamentos relacionados**

- ***Code Coverage Process*** (Atividades, responsáveis e artefatos gerados).

**a. Em relação à contribuição aos objetivos acima citados.**

☐ Contribui fortemente
☐ Contribui em partes
☐ Contribui fracamente
☐ Não contribui

**b. Em relação à sua aplicabilidade aos tipos de metodologias de desenvolvimento de software. É possível dizer que pode ser utilizada em:**

☐ Metodologias ágeis
☐ Metodologias tradicionais
☐ Ambos os tipos
☐ Nenhuma deles

## 2.1. Verificação e Validação

**Objetivos a serem alcançados**

O objetivo da verificação se refere ao conjunto de atividades que garante que o software implemente corretamente uma função especifica.

O objetivo da validação se refere a um conjunto diferente de atividades que garante que o software que foi construído é rastreável às exigências do cliente.

**Questionamentos relacionados**

- **Técnicas de Cobertura de Código** (*statment coverage, branch coverage, Linear Code Sequence and Jump* – LCSAJ e demais técnicas de cobertura).

  **a. Em relação à contribuição aos objetivos acima citados.**

  ☐ Contribui fortemente
  ☐ Contribui em partes
  ☐ Contribui fracamente
  ☐ Não contribui

  **b. Em relação à sua aplicabilidade aos tipos de metodologias de desenvolvimento de software. É possível dizer que pode ser utilizada em:**

  ☐ Metodologias ágeis
  ☐ Metodologias tradicionais
  ☐ Ambos os tipos
  ☐ Nenhuma deles

- **Ferramenta de Cobertura** (Geração de relatórios eficientes para apresentar a melhoria no código, indicação de casos de testes redundantes e que devem ser criados).

  **a. Em relação à contribuição aos objetivos acima citados.**

  ☐ Contribui fortemente
  ☐ Contribui em partes
  ☐ Contribui fracamente
  ☐ Não contribui

  **b. Em relação à sua aplicabilidade aos tipos de metodologias de desenvolvimento de software. É possível dizer que pode ser utilizada em:**

  ☐ Metodologias ágeis
  ☐ Metodologias tradicionais
  ☐ Ambos os tipos
  ☐ Nenhuma deles

# 3. Métricas de Qualidade de Software

**Objetivos a serem alçados**

Podemos medir a qualidade do software ao longo do seu processo de desenvolvimento, e depois que o produto for entregue ao cliente e aos usuários. As métricas coletadas antes da entrega oferecem uma base quantitativa para tomadas de decisão referente a projeto e os testes. Métricas coletadas, após a entrega, concentram-se no número de defeitos descobertos e na manutenibilidade do sistema, essas métricas fornecem uma indicação post-mortem da efetividade do processo de software.

**Questionamentos relacionados**

- **Definição de métricas e limiares de cobertura** (seleção, definição, análise de métricas e ações e medidas corretivas).

**a. Em relação à contribuição aos objetivos acima citados.**

☐ Contribui fortemente
☐ Contribui em partes
☐ Contribui fracamente
☐ Não contribui

**b. Em relação à sua aplicabilidade aos tipos de metodologias de desenvolvimento de software. É possível dizer que pode ser utilizada em:**

☐ Metodologias ágeis
☐ Metodologias tradicionais
☐ Ambos os tipos
☐ Nenhuma deles

# 4. Aspectos Gerais

A utilização do *Code Coverage Process* como um todo, pode-se assumir que:

**a. Em relação à contribuição ao comprimento do conjunto de objetivos citados.**

☐ Contribui fortemente
☐ Contribui em partes
☐ Contribui fracamente
☐ Não contribui

**b. Em relação à sua aplicabilidade aos tipos de metodologias de desenvolvimento de software. É possível dizer que pode ser utilizada em:**

☐ Metodologias ágeis
☐ Metodologias tradicionais
☐ Ambos os tipos
☐ Nenhuma deles

**Descreva Benefícios/Problemas do Processo e Técnica de Cobertura de Código**

# Apêndice E - Exemplos de ferramentas de cobertura de código

A seguir são descritas algumas ferramentas de cobertura de código disponíveis no mercado. Cada ferramenta apresentada tem o fornecedor, URL, descrição geral, tipo de licença, suporte a linguagem e ambiente de desenvolvimento.

- BullseyeCoverage

| | |
|---|---|
| Nome: | BullseyeCoverage |
| Fornecedor: | Bullseye Testing Technology |
| URL: | http://www.bullseye.com/ |
| Descrição Geral: | BullseyeCoverage é um analisador de cobertura de código para C++ e C que informa quanto do código fonte foi testado. Pode usar esta informação para concentra o esforço de teste e definir áreas que precisam ser re-testadas.<br>Algumas características de ferramenta são:<br>▪ Cobertura de função lhe dá uma visão geral e cobertura de condition/decision lhe dá uma melhor visão<br>▪ Integrado com Microsoft Visual Studio<br>▪ Incluí ou exclui qualquer parte de seu código do projeto<br>▪ Código fonte incluído em tempo real |
| Tipo de Licença: | Por usuário |
| Suporte de Linguagem: | C, C++ |
| Suporte de Ambiente: | Windows, Unix |

- Cantata C++

| | |
|---|---|
| Nome: | Cantata C++ |
| Fornecedor: | IPL |
| URL: | http://www.iplbath.com/products/tools/pt400.uk.php |
| Descrição Geral: | Cantata++ foi projetada em volta dos requisitos da linguagem C/C++ para produzir uma ferramenta que permite para desenvolvedor executar com eficiência testes de unidade e de integração.<br><br>Algumas características da ferramenta são:<br><br>▪ Teste de unidade e integração: no computador e no target.<br>▪ Análise de cobertura integrada: *statement*, *decision*, MC/DC, ponto de entrada e métricas de *call-return*<br>▪ Suporte completo para: ANSI C, ISO C++ e EC++<br>▪ Análise dos resultados de cobertura em forma de gráficos<br>▪ Análise estática: complexidade de código e métricas de tamanho<br>▪ Integração: É integrada com outras ferramentas de desenvolvimento, incluindo de*bugs*gers, simuladores/emuladores. |
| Tipo de Licença: | Por usuário |
| Suporte de Linguagem: | C, C++ e java |
| Suporte de Ambiente: | Windows, Solaris, Linux |

- LDRA Testbed

| | |
|---|---|
| Nome: | LDRA Testbed |
| Fornecedor: | LDRA |
| URL: | http://www.ldra.co.uk/pages/testbed.asp |
| Descrição Geral: | LDRA Testbed que pode ser aplicada nos domínios de teste por Análise Estática e Análise Dinâmica.<br>▪ Análise estáticas do código: complexidade e estrutura do código.<br>▪ Análise dinâmica envolve execução com dados de teste, por uma versão instrumentada do código fonte. O requisito mais freqüente é alcançar níveis específicos de cobertura de controle de fluxo. Há muitos níveis de cobertura que podem ser informados usando a ferramenta LDRA: *statement, branch/decision*, LCSAJ, MC/DC e cobertura dinâmica de fluxo de dados.<br>A ferramenta possui seu próprio IDE, onde é possível criar a lista de arquivos a serem instrumentados. Esta ferramenta cria uma copia do arquivo original com a instrumentação. No momento que for executar o código instrumentado é gerado um arquivo histórico para em seguida com esse arquivo a ferramenta puder gerar o relatório de cobertura. A ferramenta oferece métricas de análise estática para o código fonte utilizando a verificação de padrões, além de oferecer análise dinâmica, com dados da cobertura do código.<br>Os relatórios gerados são gráficos e textuais disponíveis em html e em formatos ASCII. Além disso, é possível comparar dados históricos de teste e os visualizar como relatórios gráficos. Finalmente, também é possível escolher as métricas de cobertura no relatório e o limiar para o critério |

| | |
|---|---|
| | de falha/sucesso. |
| Tipo de Licença: | Por usuário |
| Suporte de Linguagem: | C, C++, C#, Ada83, Ada95, Java, Visual Basic, Intel® Assemblers, Motorola® Assemblers, Texas Instruments® Assemblers, PowerPC® Assemblers, Cobol, Coral66, Fortran, Pascal, PL/Mx86, PL/1, Algol |
| Suporte de Ambiente: | Windows, Unix, Unisys |

- Clover Code Coverage

| | |
|---|---|
| Nome: | Clover Code Coverage for Java |
| Fornecedor: | Cenqua |
| URL: | http://www.cenqua.com/clover/ |
| Descrição Geral: | Clover é uma poderosa e altamente configurável ferramenta de análise de cobertura de código. Descobre seções de código que não está sendo exercitado adequadamente pelos testes de unidade.<br><br>Algumas características de ferramenta são:<br><br>▪ Plugins completamente integrado para NetBeans, Eclipse, IDÉIA de IntelliJ, JBuilder e JDeveloper. Este plugins permitem medir e inspecionar o resultado da cobertura sem sair do IDE;<br><br>▪ Integração com projetos que usam o Ant apache. Integração fácil a sistemas legados com interface em linha de comando e API;<br><br>▪ Cobertura detalhada de métodos, *statment* e *branch*;<br><br>▪ Relatório em formato de HTML, PDF, XML ou em Swing GUI. |
| Tipo de Licença: | Por usuário, Servidor, Equipe, Open-Source |
| Suporte de Linguagem: | Java, Net |
| Suporte de Ambiente: | Windows, Linux |

- Rational PureCoverage

| | |
|---|---|
| Nome: | IBM Rational PureCoverage Component – IBM Rational PurifyPLus |
| Fornecedor: | IBM |
| URL: | http://www-306.ibm.com/software/awdtools/purify/features/ |
| Descrição Geral: | IBM Rational PureCoverage para Linux e Unix, um componente do conjunto da Rational PurifyPlus. É uma personalização da ferramenta de análise de cobertura de código para aplicações Windows que foram implementados em C/C++ ou Java.<br><br>Rational PureCoverage automaticamente detalha área testada ou não do código e visualiza a análise dos dados, permitindo ao desenvolvedor criar testes mais eficiente .<br><br>Algumas características da ferramenta são:<br><br>▪ Automaticamente detecta pontos não testados;<br><br>▪ Integrado com Rational Purify, Rational Quantify, Visual Test, e ClearQuest. |
| Tipo de Licença: | Por usuário |
| Suporte de Linguagem: | C++, C, C#, .Net, Java |
| Suporte de Ambiente: | Windows, Unix, Linux |

- Testwell CTC++

| | |
|---|---|
| Nome: | Testwell CTC++ |
| Fornecedor: | Testwell Oy (Ltd) |
| URL: | http://www.testwell.fi/ctcdesc.html |
| Descrição Geral: | CTC++ é uma ferramenta de análise dinâmica para código implementado em C e C++. Como uma ferramenta de cobertura, CTC++ mostra toda cobertura de *Modified Condition/Decision Coverage* (MC/DC).<br><br>Algumas características da ferramenta são:<br><br>▪ Fornece alguns tipos de análise de cobertura. Por exemplo: Function coverage, *decision, statement, condition e Modified Condition/Decicion Coverage*.<br><br>▪ Suporte para teste no target<br><br>CTC++ usa o seu próprio compilador para criar a versão instrumentada do código fonte ou arquivos do tipo de objeto, utilizando a linha de comando ou o MVC++. Essas duas estratégias podem gerar uma versão instrumentada do código fonte, substituindo o original; ou gerar uma versão instrumentada de arquivos do tipo objeto. |
| Tipo de Licença: | Por usuário |
| Suporte de Linguagem: | C, C++ |
| Suporte de Ambiente: | Solaris, Linux |

Dissertação de Mestrado apresentada por **Elifrancis Rodrigues Soares** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Adaptação do Processo de Desenvolvimento de Software para Análise de Cobertura de Código"**, orientada pelo **Prof. Alexandre Marcos Lins de Vasconcelos** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Alexandre Cabral Mota
Centro de Informática / UFPE

Prof. Jones Oliveira de Alburquerque
Departamento de Estatística e Informática / UFRPE

Prof. Alexandre Marcos Lins de Vasconcelos
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 27 de fevereiro de 2007.

**Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO**
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.