UNIVERSIDADE FEDERAL DE SANTA CATARINA
Departamento de Informática e Estatística
Ciências da Computação

Suporte à rastreabilidade de informações ao longo das fases de desenvolvimento de software

por

Carlos Alexandre Matias
Ricardo Joselito Winck

Professor Doutor Ricardo Pereira e Silva
Orientador
Florianópolis, 19 de Fevereiro de 2003.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
Departamento de Informática e Estatística
Ciências da Computação

Suporte à rastreabilidade de informações ao longo das fases de desenvolvimento de software

por

Carlos Alexandre Matias
Ricardo Joselito Winck

Professor Doutor Ricardo Pereira e Silva
Orientador

Suporte à rastreabilidade de informações ao longo das fases de desenvolvimento de software

Professor Doutor Ricardo Pereira e Silva
Orientador

Professor Walter de Abreu Cybis
Membro da Banca Avaliadora

Professora Patrícia Vilain
Membro da Banca Avaliadora

1 Resumo

O tema da pesquisa está relacionado a área de Engenharia de Software, mais precisamente ao desenvolvimento de processos de gerenciamento de software. O trabalho foi fundamentado no modelo CMMI (Capability Maturity Mode Integration), recentemente lançado pelo SEI (Software Engineering Instutute). O modelo tem por objetivo aumentar a qualidade e a produtividade no desenvolvimento de sistemas e softwares, integrando as duas disciplinas.

Como produto de software, decorrente deste estudo, apresenta-se uma solução ao controle e a rastreabilidade automáticos e bi-direcional dos requisitos ao longo do ciclo de desenvolvimento de software. A ferramenta foi batizada pelo nome de 'tracer', pois ela vai fazer o rastreamento adequado, acompanhando os requisitos em todas as fases do projeto e gerando a matriz de rastreabilidade.

2 Sumário

3 Lista de Tabelas

4 Lista de Figuras

5 Acrônimos e Abreviaturas

- CMM – Capability Maturity Model
- CMMI – Capability Maturity Model Integration
- IPD-CMM - Integrated Product Development Capability Maturity Model
- SEI – Software Engineering Institute
- SECM - System Engineering Capability Model
- UFSC - Universidade Federal de Santa Catarina

6 Introdução

Esta pesquisa tem por objetivo o desenvolvimento de uma ferramenta para auxiliar o gerenciamento do processo de desenvolvimento de software, mais especificamente para auxiliar o gerenciamento dos requisitos. Apresenta-se, inicialmente, uma visão geral do modelo CMMI (Capability Maturity Model Integration) e a área específica onde a ferramenta irá dar suporte. Esta apresentação tem por objetivo contextualizar e delimitar o escopo do trabalho. Em seguida, mostra-se a ferramenta de software, as conclusões e possíveis melhoramentos, além de outras considerações.

6.1 Tema

Desenvolvimento de uma ferramenta para dar suporte ao gerenciamento do processo de desenvolvimento de software na área de gerenciamento de requisitos.

6.2 Delimitação do Tema

O foco do tema é o desenvolvimento de uma ferramenta para dar suporte a seguinte prática específica: manter a rastreabilidade bi-direcional entre os requisitos e os planos de projeto, e também entre os requisitos e os produtos de trabalho de sofwares (p.e., código, testes). Esta prática específica é um componente do modelo CMMI, e deve ser realizada para satisfazer o componente área de processo gerenciamento de requisitos, também coberta pelo modelo. Estes conceitos serão melhores explicados ao longo do trabalho.

6.3 Objetivo Geral

Desenvolvimento de uma ferramenta para automatizar a rastreabilidade de requisitos ao longo das etapas de desenvolvimento de software.

6.4 Objetivos Específicos

- Criar uma ferramenta para gerar a matriz de rastreabilidade entre requisitos e os planos de projeto e produtos de trabalho de forma automática;
- Apresentar o novo modelo CMMI;
- Descrever a área de processo de gerenciamento de requisitos, na qual a ferramenta está inserida.

6.5 Motivação/Justificativa

O tema da pesquisa aproveita as atividades de desenvolvimento de software realizadas na Universidade Federal de Santa Catarina em parceria com a Motorola Industrial Ltda, sob a forma de estágio.

Durante as atividades do estágio, deparou-se com a dificuldade da geração manual da matriz de rastreabilidade de requisitos, que faz o acompanhamento dos requisitos ao longo do ciclo de desenvolvimento de software. Por outro lado, a experiência mostrou os grandes benefícios da adoção de um processo de melhoramento, no caso proprietário da Motorola, baseado no modelo CMM Software [CMM 94].

Além da qualidade dos produtos, reconhecida pelos clientes, houve um fato, durante as atividades do estágio, que comprovou a eficácia do uso de processos. Iniciado o projeto 3G Parsers na UFSC, o mesmo foi transferido para um site da Motorola, localizado na França. Bastaram duas semanas de trabalho para que o *handover* fosse completado. Após isto, quase não houve consultas e os trabalhos seguiram seu curso normalmente. Se não fosse a farta documentação existente, este processo seria bastante traumático e caro.

Por outro lado, observaram-se algumas limitações e entraves, que podem desestimular o uso de tais processos. Dentre estas, destaca-se os custos homem/hora associados à burocratização. Estes custos são decorrentes da geração, inspeção e manutenção de documentos. Outro fator é a presença de trabalho monótono em excesso, que pode chegar a 65 %, ou mais, das atividades de desenvolvimento.

Sendo assim, a relevância deste trabalho esta no fato de que, a automatização da geração de documentos, representa um potencial fator de redução de custos, de aumento da qualidade e produtividade dos produtos de trabalho, incentivando, ainda, o uso de processos disciplinados pelas empresas de informática. Atualmente, o desenvolvimento de software ainda é realizado de maneira caótica na maioria das empresas do setor.

Outro ponto de destaque é o fato de que a ferramenta é genérica o suficiente para ser utilizada em qualquer tipo de processo, seja um complexo, como os gerados a partir do modelo CMMI, seja em processos mais simples, como o XP (Extreme Programming) [WAK 00].

A novidade deste trabalho está relacionada à teoria associada ao tema. Recentemente, foi lançado o modelo CMMI [CCR 02] [CSR 02]. O modelo faz a integração de outros modelos CMMs existentes e fornece uma interface única. Além da integração das disciplinas de engenharia de software e sistema, como veremos adiante, ele estabelece um *framework* para que as empresas possam criar os seus próprios modelos, a partir de suas próprias necessidades, tamanho e área de domínio.

7 CMMI (Capability Maturity Model Integration)

7.1 Introdução

Apresenta-se, a seguir, uma visão geral do modelo CMMI (Capability Maturity Model Integration), sua estrutura, representações, disciplinas e componentes. O objetivo é mostrar o contexto e o escopo do trabalho.

O modelo realiza a integração dos seguintes modelos CMM [CSR 02]:

- Capability Maturity Model for Software V2, draft C (SW-CMM V2C);
- EIA Interim Standard 731, System Engineering Capability Model (SECM);
- Integrated Product Development Capability Maturity Model, draft V0.98 (IPD-CMM);

Patrocinado pelo Departamento de Defesa Americano e contando com a participação de dezenas de empresas, a integração dos modelos CMM surgiu da necessidade de uma melhor integração entre as disciplinas de sistemas e software. A realidade tem mostrado que a participação dos requisitos de software nos sistemas tem aumentado muito nos últimos anos. No projeto do avião B-2, o software representava 65 % dos requisitos totais, no projeto do F-22, 80 % [PHI 02].

CMMI é um modelo, e como tal, é uma simplificação aproximada da realidade. É usado como ponto de referência, fornecendo diretrizes para o melhoramento dos processos das organizações. Sendo assim, CMMI não é um processo em si ou a descrição de um processo. Ele responde a pergunta o que fazer, e não como fazer ou quem fará. Um processo é o ponto de partida para uma organização alcançar um desenvolvimento sustentado. Em grande medida, a qualidade dos produtos e serviços depende da qualidade do processo para produzi-los e mantê-los. Os benefícios alcançados são inúmeros. Os mais importantes são:

- O aumento da previsibilidade de prazos e orçamentos;
- Redução do tempo do ciclo de desenvolvimento;
- Aumento da produtividade com redução da taxa de erros.

O modelo é importante por que [PHI 02]:

- Fornece um ponto de partida para o desenvolvimento de um processo;

- Permite que se estabeleça uma linguagem comum e uma visão compartilhada na organização;
- Permite que se aproveite o benefício das experiências anteriores de toda uma comunidade de desenvolvimento;
- Fornece um *framework* para se priorizar ações.

Na subseção seguinte, apresentam-se os critérios básicos de como escolher um modelo CMMI.

7.2 Escolhendo um modelo CMMI

A escolha de um modelo CMMI depende das necessidades de cada organização. Isto implica que existem muitos modelos CMMI disponíveis. Basicamente, o foco pode estar centrado na disciplina de sistemas ou de software. Se o campo de conhecimento está voltado a sistemas, o modelo define a representação contínua. Se for voltada a disciplina de software, pode-se adotar a representação em estágio. Entretanto, é possível se construir um modelo que seja uma combinação das duas representações.

Assim sendo, o modelo CMMI fornece um *framework*, a partir do qual cada organização escolherá o modelo que melhor atenda as suas expectativas.

Uma vez escolhido o modelo, têm-se as diretrizes para o estabelecimento e melhoramento de um processo da organização.

Abaixo segue uma visão resumida das principais vantagens da representação contínua e em estágios e o foco das disciplinas de engenharia de sistemas e de software. As disciplinas de desenvolvimento de produtos e processos integrados e de contratação de fornecedores também são abordadas. A primeira possui uma representação híbrida das representações contínua e em estágios e a segunda está contida em ambas.

É importante salientar que a escolha do melhor modelo, a ser usado em cada organização, deve ser baseada em um julgamento profissional.

As fontes de cada representação foram os seguintes modelos:

- Software CMM: em níveis;
- SECM: contínua;
- IPD CMM: híbrida;

A representação em estágios tem como vantagens:

1. Fornece um guia para a implementação através de:
   - Grupos de área de processo;

- De uma seqüência de implementação;

2. Fornece uma estrutura familiar para quem esta migrando do CMM software;

A representação contínua tem como vantagens:

1. Fornece mais flexibilidade para quem deseja focar-se em uma área de processo específica, de acordo com as metas e objetivos do negócio;
2. Fornece uma estrutura familiar para quem já trabalha com sistemas de engenharia;

Atualmente, estão disponíveis os seguintes campos de atuação, ou simplesmente, disciplinas, que podem ser cobertas pelo framework CMMI:

- Engenharia de software;
- Engenharia de sistemas;
- Desenvolvimento de produtos e processos integrados;
- Contratação de fornecedores;

O modelo é bastante flexível e prevê que outras disciplinas poderão ser acrescidas ao modelo.

O foco de cada campo do conhecimento é descrito sucintamente a seguir:

- A engenharia de sistemas cobre o desenvolvimento de um sistema como um todo, que pode ou não incluir software. O foco da engenharia de sistemas é transformar as necessidades e expectativas dos clientes em produtos, serviços e suporte durante o ciclo de vida do produto.
- A engenharia de software cobre o desenvolvimento de sistemas de software. O foco é sobre a aplicação de uma abordagem sistemática, disciplinada e que pode ser medida quantitativamente para o desenvolvimento, operação e manutenção de software.
- A disciplina de desenvolvimento de produtos e processos integrados tem seu foco sobre as pessoas. É uma abordagem sistemática que tem por objetivo conseguir uma colaboração relevante dos atores envolvidos no projeto, durante o ciclo de desenvolvimento do produto.
- Quando o produto ou serviço desenvolvido é muito complexo, é necessária a contratação de fornecedores externos. O acompanhamento destes fornecedores é o foco da disciplina de contratação de fornecedores.

Baseada no tamanho, campo de domínio e estrutura, cada organização poderá, assim, criar o seu próprio modelo. Os manuais da SEI [CCR 02] [CSR 02] contém informações detalhadas de como proceder a melhor escolha do modelo. Os conceitos aqui apresentados foram baseados nos referidos manuais. Na próxima seção apresenta-se a estrutura do CMMI, para uma visualização do contexto do projeto.

7.3 Estrutura do CMMI

A figura 1 ilustra a estrutura do modelo CMMI [PHI 02]. Nela observa-se uma estrutura de um só modelo e duas representações. No lado esquerdo da figura tem-se a representação em estágios, que deriva do CMM software (SW-CMM V2C). No direito, a representação contínua, derivada do CMM de engenharia de sistemas (SECM). Uma organização que deseja uma evolução como um todo, deverá focar-se num modelo baseado em níveis de maturidade (representação em estágios). Nada impede, porém, que queira tornar-se capaz na área de engenharia, por exemplo. Sendo assim, usará a representação contínua, ou ambas.

Desta forma, os diversos modelos CMMI são desenvolvidos.

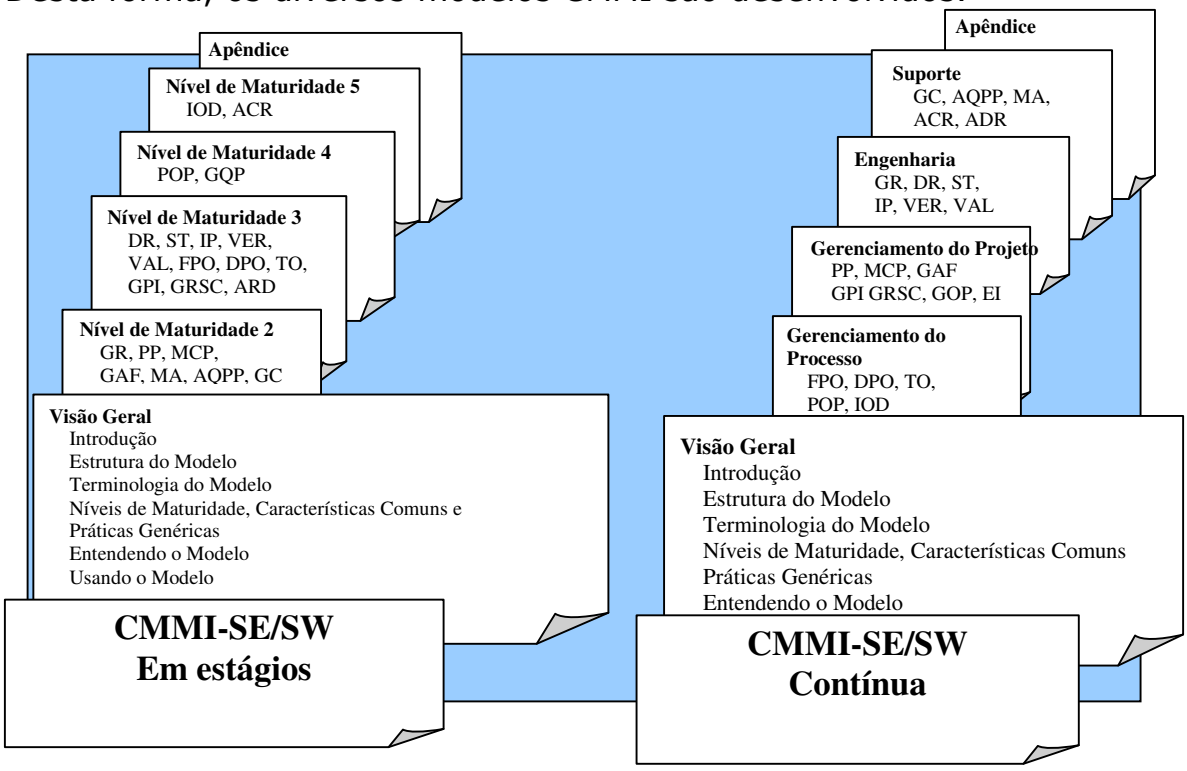


Figura 1: Estrutura do Modelo CMMI

A seguir veremos cada uma das representações.

Representação em estágios

A representação em estágios organiza as áreas de processo em níveis de maturidade, para dar suporte e guiar o processo de melhoramento. Cada nível de maturidade representa um caminho evolutivo, e envolve a organização como um todo. Para alcançar o nível desejado, a organização deverá cumprir todos os requisitos estabelecidos para cada área de processo, definidas para aquele nível. Por exemplo, uma organização será classificada como nível 2 (gerenciado) se satisfizer as seguintes áreas de processo (ver figura 1):

- gerência de requisitos (GR);
- planejamento de projeto (PP);
- monitoramento e controle do projeto (MCP);
- gerenciamento de acordo com fornecedores (GAF);
- medidas e análise (MA);
- assegurar a qualidade do produto e do processo (AQPP);
- gerenciamento de configuração (GC);

A tabela 1 mostra a legenda de cada uma das siglas presentes na figura 1, bem como um quadro demonstrativo dos níveis, seu foco e as áreas de processo relacionadas a cada nível de maturidade.

Nível	***Foco***	***Área de Processo***
5 Otimizado	**Melhoramento contínuo do Processo**	**- Inovação organizacional e desenvolvimento (IOD)** **- Análise casual e resolução (ACR)**
4 Quantitativamente Gerenciado	**Gerenciamento quantitativo**	**- Performance Organizacional do Processo (POP)** **- Gerenciamento Quantitativo do Projeto (GQP)**
3 Definido	**Processo padronizado**	**- Desenvolvimento de Requisitos (DR)** **- Solução Técnica (ST)** **- Integração do Produto (IP)** **- Verificação (VER)** **- Validação (VAL)** **- Foco no Processo Organizacional (FPO)** **- Definição de Processo Organizacional (DPO)** **- Treinamento Organizacional (TO)** **- Gerenciamento de Projeto Integrado (GPI)** **- Gerenciamento Integrado de**

		Fornecedores (GIF) **- Gerenciamento de riscos (GRSC)** **- Análise de Decisão e Resolução (ADR)** **- Ambiente Organizacional para Integração (AOI)** **- Equipe Integrada (EI)**
2 Gerenciado	**Gerenciamento básico do projeto**	**- Gerenciamento de Requisitos (GR)** **- Planejamento do Projeto (PP)** **- Monitoramento e Controle do Projeto (MCP)** **- Gerenciamento de Acordo com Fornecedores (GAF)** **- Medidas e análises (MA)** **- Assegurar a qualidade do produto e processo (AQPP)** **- Gerenciamento de configuração (GC)**
1 Inicial		

Tabela 1: Áreas de processo por nível de maturidade na representação em estágios.

A figura 2, abaixo, ilustra a estrutura do CMMI para a representação em estágios. Cada elemento será descrito a seguir.

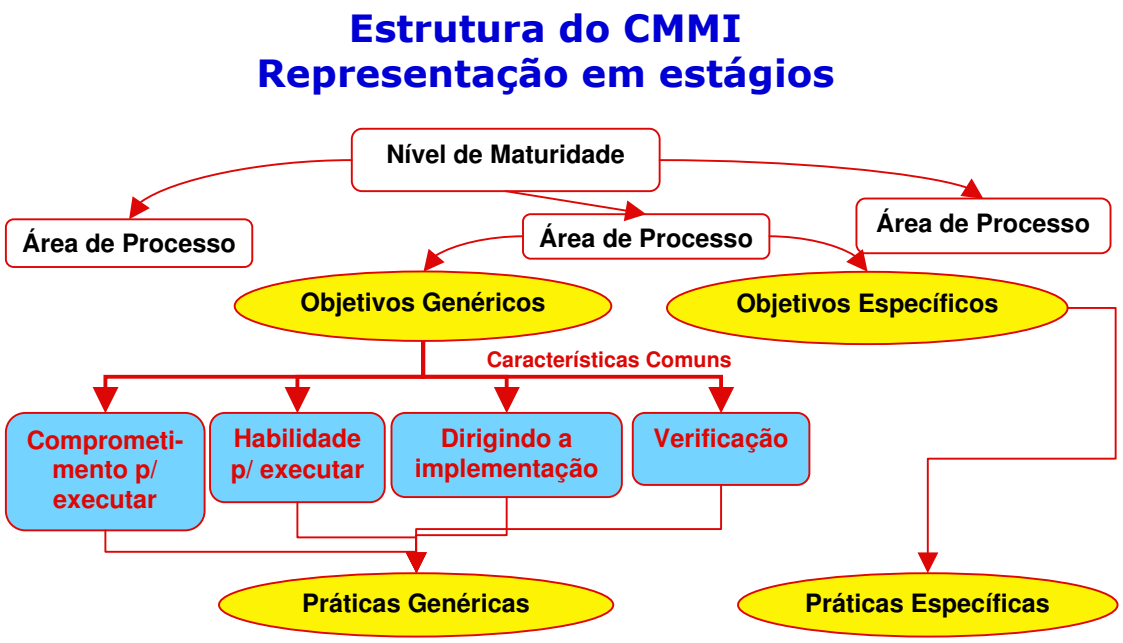


Figura 2: Estrutura da representação em estágios do modelo CMMI.

Níveis de Maturidade

O SEI [CSR 02] define nível de maturidade como um platô evolucionário bem definido de um processo de melhoramento. No modelo CMMI há cinco níveis de maturidade, onde cada camada é uma fundação para um contínuo processo de melhoramento. São numerados de um a cinco:

1. Inicial
2. Gerenciado
3. Definido
4. Quantitativamente Gerenciado
5. Otimizado

Nível de maturidade consiste em um conjunto de áreas de processos e são medidos pela realização de objetivos específicos e genéricos que são aplicados nestas áreas. A seguir descreve-se sucintamente cada nível de maturidade.

Nível de maturidade 1: inicial

Neste nível os processos são ad hoc ou caóticos. Geralmente não se consegue repetir os sucessos do passado. Os processos são abandonados em tempos de crise. As organizações não fornecem um ambiente estável.

Nível de maturidade 2: gerenciado

Os projetos da organização têm assegurado que os requisitos são gerenciados e que processos são planejados, executados, medidos e controlados. A organização possui práticas que são repetidas em tempo de estresse. A organização tem todos os objetivos específicos e genéricos realizados para as áreas de processo do nível de maturidade 2. Comprometimentos são estabelecidos entre atores importantes e há revisões quando necessário.

Nível de maturidade 3: definido

Caracteriza-se pela existência de um processo bem definido e entendido por todos e são descritos na forma de padrões, procedimentos, ferramentas e métodos. A diferença para o nível de

maturidade 2 é de que os padrões do processo são mais rigorosamente definidos e se estendem por toda a organização, e não apenas por um projeto. No nível 3 os processos são gerenciados de forma mais pró-ativa, usando-se um entendimento dos inter-relacionamentos e das medidas das atividades do processo. É possível assim fazer previsões qualitativas no processo.

Nível de maturidade 4 : quantitativamente gerenciado

A evolução do nível 3 para o nível 4 é que ocorre uma previsibilidade da performance do processo. No nível 4, a performance do processo é controlada usando-se estatísticas e outras técnicas quantitativas. Neste nível, as previsões são quantitativas.

Nível de maturidade 5: otimizado

Os processos são melhorados continuamente baseados em valores quantitativos observados em de causas comuns de mudanças no processo. Estes melhoramentos na performance do processo podem ser derivados de melhoramentos tecnológicos incrementais ou inovadores, sempre levando em consideração os objetivos de negócios da organização.

Um nível fornece uma fundação necessária para a implementação de um processo em um próximo nível de maturidade. Processos de mais alto nível podem ser sacrificados se não apoiados na disciplina fornecida pelos níveis anteriores. Da mesma forma, em tempo de crise, processos de mais alto nível podem não ser consistentes, se aplicados em organizações de mais baixo nível de maturidade.

Componentes do modelo CMMI

Estes são os principais componentes do modelo CMMI [CCR 02] [CSR 02] e são comuns a representação contínua e em estágios. A figura 2 mostra como eles se relacionam. Na seção 2.3 veremos um exempo de cada um dos componentes abaixo para a área de processo de gerenciamento de requisitos.

Área de processo – uma área de processo é definida por um conjunto de práticas relacionadas (genéricas e específicas) a uma determinada área da organização (gerenciamento dos requisitos). Quando executadas coletivamente, estas práticas satisfazem um

conjunto de objetivos que realizam um melhoramento significativo nesta área.

Objetivos Específicos – referem-se a uma área de processo e estabelece uma característica específica que descreve o que deve ser implementado para satisfazer a área de processo em questão (gerenciar requisitos).

Práticas Específicas – É uma atividade que é considerada importante para satisfazer um objetivo específico associado (Obtendo um entendimento dos requisitos).

Características Comuns – Quatro características comuns organizam as práticas genéricas de cada área de processo.

1. **Comprometimento para executar**: criar políticas e responsáveis pelos esforços de melhoramento.
2. **Habilidade para executar**: assegurar que o projeto e/ou a organização tenha os recursos necessários para adotar o processo de melhoramento.
3. **Dirigindo a implementação:** coletar, medir e analisar os dados relacionados ao processo.
4. **Verificação**: verificar se as atividades do projeto e/ou da organização estão de acordo com os requisitos, processos e procedimentos.

Objetivos Genéricos – Estes objetivos são chamados 'genéricos' por que a mesma declaração de um objetivo aparece em múltiplas áreas de processo. São necessários no modelo e são usados para avaliar se uma área de processo está sendo satisfeita (institucionalizar um processo gerenciado).

Práticas Genéricas – Fornecem institucionalização para assegurar que um processo associado a uma área de processo seja efetivo, repetível e duradouro (Atribuir responsabilidades).

A seguir, será descrita sucintamente a representação contínua do CMMI.

Representação contínua

A representação contínua utiliza seis níveis de capacidade, perfis de capacidade, estágio alvo e estágio equivalente como princípios organizacionais para o modelo de componentes. Agrupa as áreas de processo por afinidade de categoria e define níveis de capacidade de melhoramento de processo dentro de cada área.

A tabela 2, abaixo, mostra o agrupamento das áreas de processo por categoria.

Categoria	*Área de Processo*
Gerenciamento do Projeto	**- Planejamento do Projeto** **- Monitoramento e Controle do Projeto** **- Gerenciamento de Acordo com Fornecedores** **- Gerenciamento de Projeto Integrado** **- Gerenciamento Integrado de Fornecedores** **- Gerenciamento de riscos** **- Gerenciamento Quantitativo do Projeto** **- Equipe Integrada**
Suporte	**- Medidas e análises** **- Assegurar a qualidade do produto e processo** **- Gerenciamento de configuração** **- Análise casual e resolução** **- Ambiente Organizacional para Integração** **- Análise de Decisão e Resolução**
Engenharia	**1.1.1.1.1.1.1 - Gerenciamento de Requisitos** **- Desenvolvimento de Requisitos** **- Soluções Técnicas** **- Integração do Produto** **- Verificação** **- Validação**
Gerenciamento do processo	**- Performance Organizacional do Processo** **- Inovação organizacional e desenvolvimento** **- Foco no Processo Organizacional** **- Definição de Processo Organizacional** **- Treinamento Organizacional**

Tabela 2: Categorias por áreas de processo.

A figura 3 ilustra a estrutura do CMMI para a representação contínua.

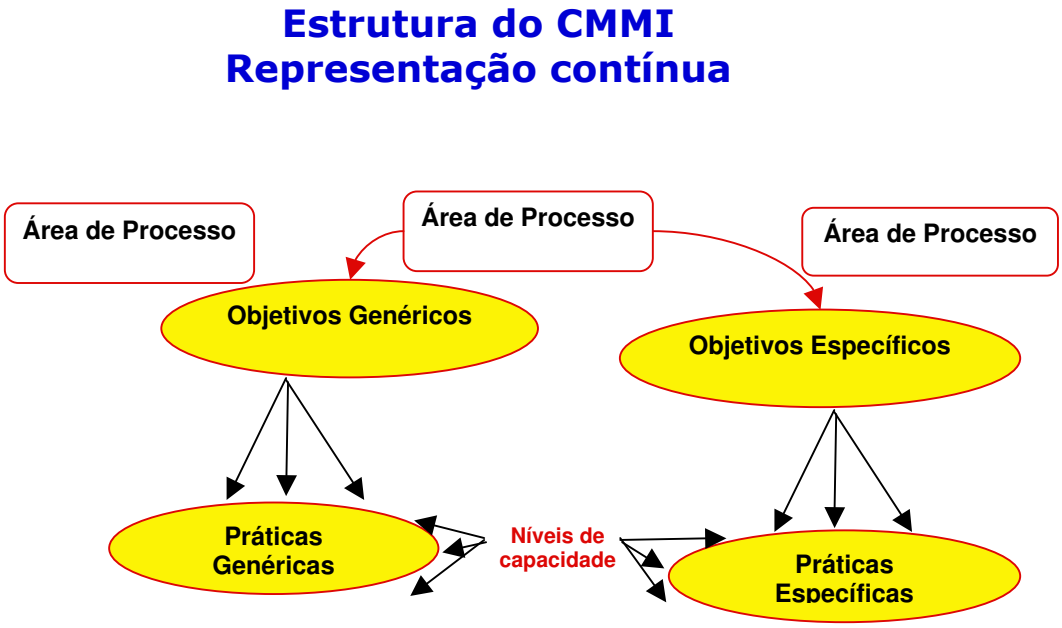


Figura 3: representação contínua do modelo CMMI.

Níveis de capacidade têm seu foco sobre o crescimento das habilidades para executar, controlar e melhorar a performance das organizações em uma determinada área de processo que a organização considera importante. Por exemplo, um organização que tem seu foco no desenvolvimento de projetos técnicos, possivelmente, dará ênfaze a categoria de engenharia. Ao contrário dos níveis de maturidade que se referem a toda a organização, os níveis de capacidade se referem à uma determinada área de processo. Os níveis de capacidade são cumulativos e cada nível serve como fundação para o nível seguinte, como na representação em estágios. Nos manuais da SEI [CCR 02] há uma descrição detalhada de cada nível de capacidade. A seguir uma descrição do que deve ser realizado em cada área de processo para se atingir o nível de capacidade desejado [PHI 02]:

0. Incompleto : Não executado, imcompleto
1. Executado : Trabalho executado
2.Gerenciado : Adoção de uma política, seguimento de planos e processos, aplicação adequada de recursos, atribuição de responsábilidades e autoridade, treinamento de pessoas, aplicação de

gerência de configuração, monitoramento, controle e avaliação do processo e revisão com gerenciamento

3. Definido : Os processo de projeto seguem os padrões dos processos da organização, o processo é entendido quantitativamente e o processo é percebido como um ativo da organização

4. Quantitativamente Gerenciado : A performance do processo é mensurável, processo estável, controles gráficos

5. Otimizado : prevenção de defeitos, melhoramentos próativos, inserção de inovações tecnlógicas e desenvolvimento

A figura 4 [PHI 02], abaixo, ilustra um exemplo de um perfil de capacidade por área de processo (GR, DR, ST). A capacidade e uma área de processo pode ser representada por um ponto em duas dimensões. O eixo da capacidade indica o 'quão bem' a atividade é executada. O eixo da área de processo indica 'o que' fazer. Na figura há o exemplo do nível de capacidade para gerenciamento de requisitos, onde GR é a área de processo de gerenciamento de requisitos, DR, desenvolvimento de requisitos, ST, soluções técnicas. A altura da barra vertical, indica o mais alto nível de capacidade na área de processo. Na figura, a área de soluções técnicas (ST) está avaliada como tendo capacidade de nível 2 (executado).

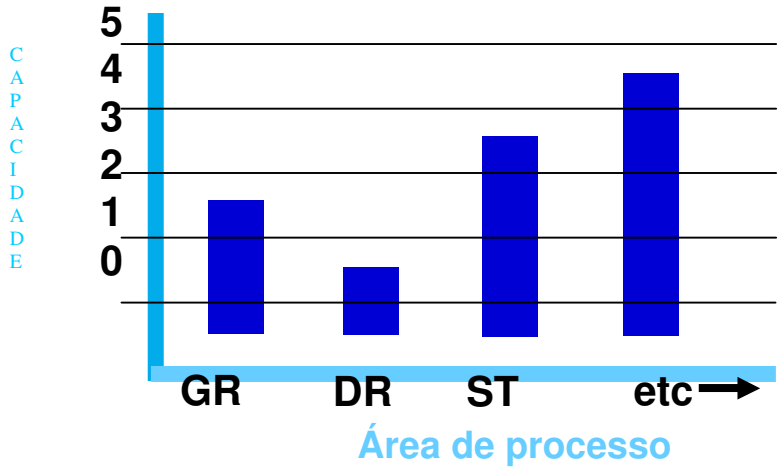


Figura 4: exemplo de nível de capacidade de gerenciamento de requisitos.

A seguir, mostra-se a área de processo de gerenciamento de requisitos, que fundamentou o desenvolvimento do aplicativo alvo deste trabalho.

Gerenciamento de requisitos

O objetivo deste projeto é fornecer um suporte ao processo de melhoramento de software. O escopo escolhido foi a área de processo de gerenciamento de requisitos, conforme necessidade levantada e já mencionada anteriormente. Mais especificamente ainda, o suporte diz respeito à prática específica de manter uma rastreabilidade bi-direcional dos requisitos. Para um melhor entendimento, e também para exemplificar o modelo CMMI, apresenta-se em mais detalhe estes componentes do modelo [CSR 02]. Como a disciplina associada ao projeto é a engenharia de software, foi escolhida a representação em estágios.

Área de processo – Gerenciamento de requisitos.

Representação – Em estágios.

Propósito – O propósito do gerenciamento de requisitos é gerenciar os requisitos dos componentes e produtos do projeto e identificar inconsistências entre estes requisitos e os planos e produtos de trabalho.

Objetivos Genéricos e Específicos:

Objetivo Específico 1 - Gerenciar requisitos: Requisitos são gerenciados e inconsistências com os planos do projeto e produtos de trabalho são identificadas.

Objetivo Genérico 2 - Institucionalizar um processo gerenciado: O processo é institucionalizado como um processo gerenciad**o,** ou seja, a organização utiliza planos e processos na execução dos projetos, sem contudo, possuir um processo próprio bem definido e institucionalizado. Ocorre apenas o gerenciamento.

Objetivo Genérico 3 - Institucionalizar um processo definido (exigido para os níveis 3 e 4 acima): O processo é institucionalizado como um processo definido, ou seja, existe um processo definido, documentado e que é seguido.

A tabela a seguir relaciona práticas com os objetivos genéricos e específicos listados acima. Atribuiu-se a seguinte legenda: PE são práticas específicas, PG são práticas gerais. O primeiro número indica o

nível de maturidade onde a prática é exigida e o segundo o número da prática. Por exemplo: PE1.1 se refere à primeira prática específica para o nível um de maturidade.

Gerenciamento de Requisitos	
Práticas Específicas	**Práticas Genéricas**
PE1.1: Obtendo um entendimento dos requisitos **PE1.2: Obtendo o comprometimento para os requisitos** **PE1.3: Gerenciando as mudanças de requisitos** **PE1.4: Mantendo a rastreabilidade bi-direcional dos requisitos** **PE1.5: Identificando inconsistências entre os produtos de trabalho e os requisitos**	**PG1.1: Executar práticas base** **PG 2.1: Estabelecer uma política organizacional** **PG 2.2: Planejar o processo** **PG 2.3: Fornecer recursos** **PG 2.4: Atribuir responsabilidades** **PG 2.5: Treinar pessoas** **PG 2.6: Gerenciar a configuração** **PG 2.7: Identificar e envolver os atores relevantes** **PG 2.8: Monitorar e controlar o processo** **PG 2.9: Objetivamente avaliar a aderência** **PG 2.10: Revisar o status com a mais alta gerência**

Tabela 3: práticas genéricas e específicas associadas ao gerenciamento de requisitos.

Como o objetivo deste relatório não é descrever o modelo CMMI em profundidade, descreve-se, apenas, a prática específica associada ao projeto. No caso é a prática PE 1.4, sublinhada na tabela 3:

PE 1.4 Mantendo a rastreabilidade bi-direcional dos requisitos: Mantendo a rastreabilidade bi-direcional entre os requisitos e os planos de projeto e produtos de trabalho.

A intenção desta prática específica é manter a rastreabilidade bi-direcional dos requisitos para cada nível de decomposição do produto. Quando os requisitos são bem gerenciados é possível acompanhá-los desde o mais baixo nível de especificação até o código fonte. A rastreabilidade também cobre os relacionamentos dos requisitos com outras entidades, como produtos intermediários e finais, mudanças no documento de projeto, planos de testes. Esta prática é, também, fundamental na condução do impacto de mudanças importantes nos requisitos sobre os planos do projeto, atividades e produtos de trabalho.

Produtos de trabalho típicos

1. Matriz de rastreabilidade de requisitos
2. Sistema de acompanhamento de requisitos

Sub práticas

1. Manter a rastreabilidade dos requisitos para assegurar que o código fonte derivado dos requisitos seja documentado.
2. Manter a rastreabilidade dos requisitos de um requisito para o seu requisito derivado tanto quanto para a alocação de funções, objetos, pessoas, processos e produtos de trabalho.

3. Manter a rastreabilidade horizontal de função para função e através das interfaces.

4. Gerar a matriz de rastreabilidade de requisitos

Este é, pois, o escopo do presente trabalho e o contexto onde foi desenvolvido. O suporte, através da geração automática da matriz de rastreabilidade dos requisitos é mais do que necessário. A tarefa de acompanhar os requisitos de forma manual é uma prática difícil, trabalhosa e sujeita a erros. Ainda mais se levarmos em consideração que, em alguns casos, pode-se ter milhares de requisitos.

Em seguida, descreve-se a ferramenta que dará este suporte de forma automática ao gerenciamento de requisitos.

8 Protótipo do aplicativo

A idéia do desenvolvimento desta ferramenta surgiu da observação de que o acompanhamento dos requisitos, ao longo do ciclo de desenvolvimento de software, quando feita manualmente, é tarefa bastante onerosa. Sua aplicação se encaixa em organizações que fazem uso de processos de gerenciamento de software, como o CMMI. A seguir, descrevem-se as fase de desenvolvimento do protótipo desenvolvido: a análise de domínio, o projeto e a implementação.

8.1 Análise do Domínio

A ferramenta tem por finalidade mapear automaticamente e bi-direcionalmente os requisitos de um projeto de desenvolvimento de software, levantados na fase de análise de requisitos, em relação às fases subseqüentes.

Além disso, o aplicativo possibilitará a configuração das fases do ciclo de desenvolvimento de software, de acordo com o processo de cada organização. Isso se deve principalmente às características diferenciadas de cada projeto de desenvolvimento e também ao tamanho da empresa ou do grupo de desenvolvimento.

Podemos citar o exemplo de uma pequena empresa que deseja colocar o seu software o mais rápido possível no mercado. Assim, essa empresa utilizará um processo constituído de uma fase de requisitos e outra de código (veja figura 5). Isso possibilita uma maior agilidade no desenvolvimento porque o processo não requer uma documentação rigorosa, mas ao mesmo tempo pode dificultar a manutenção do mesmo. Por outro lado, uma empresa multinacional necessitará de um processo constituído de uma fase de requisitos, uma fase de projeto e uma última fase de testes (veja figura 6). Isso se deve principalmente à maior complexidade do projeto em questão e também à diversidade de pessoas envolvidas no mesmo. Sendo assim, a mesma possuirá um projeto bem documentado possibilitando a migração do mesmo entre diversos grupos de desenvolvimento e também facilitando a sua manutenção.

As figuras abaixo exemplificam a configuração de processos dinâmicos, voltados à necessidade do projeto em questão.

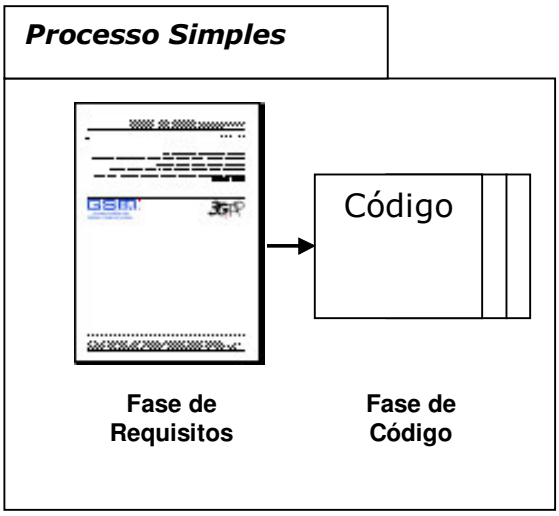


Figura 5: Processo simples de desenvolvimento de Software

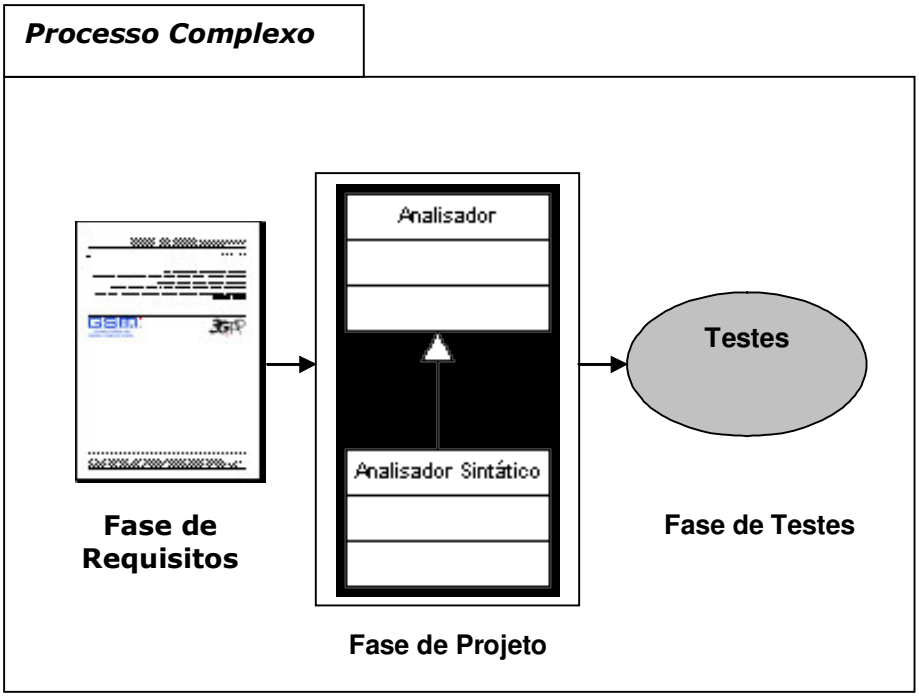


Figura 6: Processo complexo de desenvolvimento de Software

O rastreabilidade de requisitos deve seguir uma linguagem padronizada. Assim, a identificação e a descrição dos requisitos de cada fase do processo de desenvolvimento irá seguir a seguinte linguagem:

- O identificador de requisito será representado pelo símbolo

#XXX_NNNNN, XXX representa a fase do processo e NNNNN representa o número do requisito. Exemplo: fase de requisitos (#REQ_00001), fase de projeto (#PRJ_ 00001), fase de testes (#TST_00003).

- A descrição de requisito será representada pelo símbolo #<YYYYYYYYYYYYYYYYY>, YYYYYYYYYYYYYYYYY deve ser a descrição do requisito. Exemplo: fase de requisitos (#< Decodificar mensagens binárias em mensagens textuais>), fase de projeto (#< Faz o parsing de tokens binárias em tokens textuais >), fase de testes (#< Verifica se as mensagens binárias são traduzidas em texto>).
- O identificador de requisito mapeado ou relacionado será representado pelo símbolo ##XXX_NNNNN, XXX representa a fase do processo e NNNNN representa o número do requisito. Exemplo: fase de requisitos (##REQ_00001), fase de projeto (##PRJ_ 00001), fase de testes (##TST_00003).

Após definirmos a simbologia de nossa linguagem podemos passar para a sua estrutura sintática. Essa será descrita através da utilização de um exemplo de rastreabilidade do requisito 1 ao longo das fases do processo de desenvolvimento complexo (veja figura 6). Além disso também mapearemos o requisito de projeto 1 na fase subseqüente. Veja exemplo abaixo:

- Documento de requisito:

#REQ_00001 #< Decodificar mensagens binárias em mensagens textuais>

- Documento de projeto

#PRJ_00001 #< Faz o parsing de tokens binárias em tokens textuais >
##REQ_00001

- Documento de Teste

#TST_00001 #< Verifica se as mensagens binárias são traduzidas em texto>
##REQ_00001
##PRJ_00001

O produto final da ferramenta é uma matriz que faz o rastreamento dos requisitos ao longo das fases do processo de desenvolvimento (veja tabela 1). Esta matriz irá responder às seguintes perguntas:

- Quais os impactos de uma mudança de um determinado requisito?

- Onde um requisito foi mapeado e/ou implementado?
- Todos os requisitos foram cobertos e/ou implementados?
- O requisito é mesmo necessário?
- Quais os testes utilizados na verificação de um requisito?

Fase de Requisitos	Fase de Projeto	Fase de Testes
#REQ_00001	#PRJ_00001	#TST_00001

Tabela 4: Matriz de Rastreamento dos Requisitos

8.2 Projeto

Utilizou-se a metodologia de programação UML [RUM 94] [JAC 92] na documentação e descrição da arquitetura da fase de projeto do aplicativo.

A necessidade de configuração de processos dinâmicos dentro de um projeto de software acarretou na especificação de uma classe "Project" que simboliza o projeto em execução. Este possuirá um processo, representado pela classe "Process", que por sua vez será constituído por uma seqüência de fases, representadas pela classe "Phase" (veja figura 7).

Assim, podemos criar objetos dinâmicos que representem o projeto de desenvolvimento em questão e também o seu respectivo processo de desenvolvimento de software.

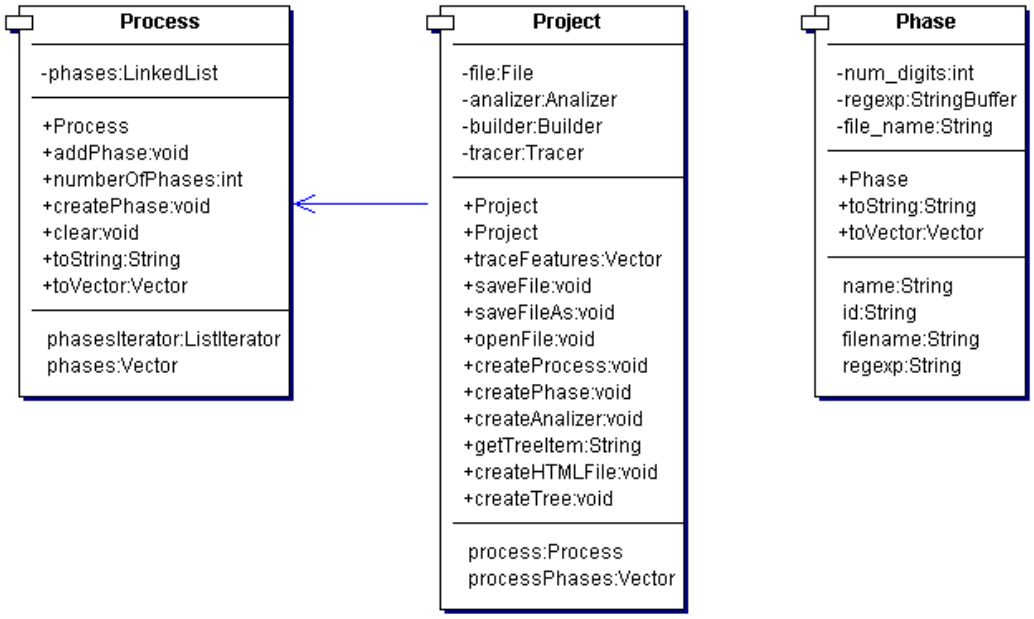


Figura 7: Especificação das classes Project, Process e Phase.

Um outro ponto a ser abordado é a especificação dos documentos de cada fase do processo. Sendo assim, definiu-se um outro conjunto de classes para simbolizar o documento (classe "Document") relacionado a sua respectiva fase do processo. O documento possuirá um conjunto de requisitos, que será representado pela classe "Feature", os quais são constituídos pelo identificador do requisito, a descrição do mesmo e também pelos requisitos mapeados e/ou relacionados (veja figura 8).

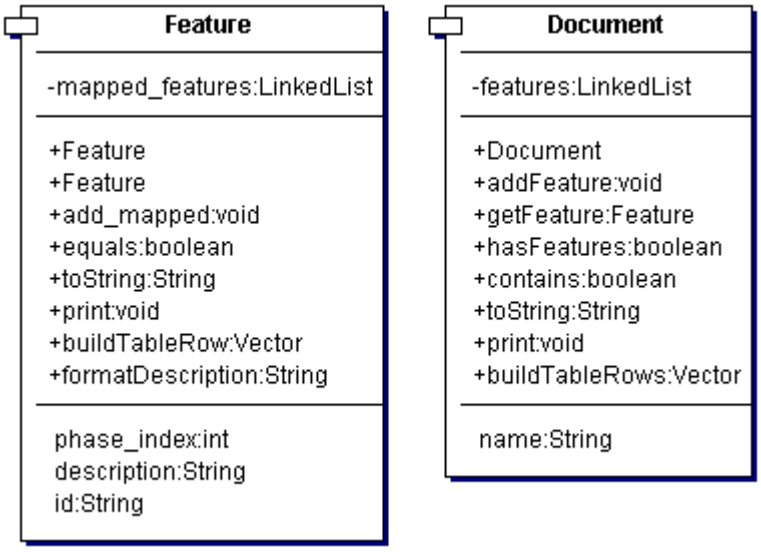


Figura 8: Especificação das classes Document e Feature.

A análise e extração dos símbolos definidos pela linguagem (classe "Grammar") do aplicativo serão realizadas através de um analisador léxico (classe "Analizer"). Esse irá prover os símbolos necessários para que o montador (classe Builder) possa criar os documentos de cada fase do processo e posteriormente conectar os requisitos relacionados e mapeados através das fases do processo (veja figura 9).

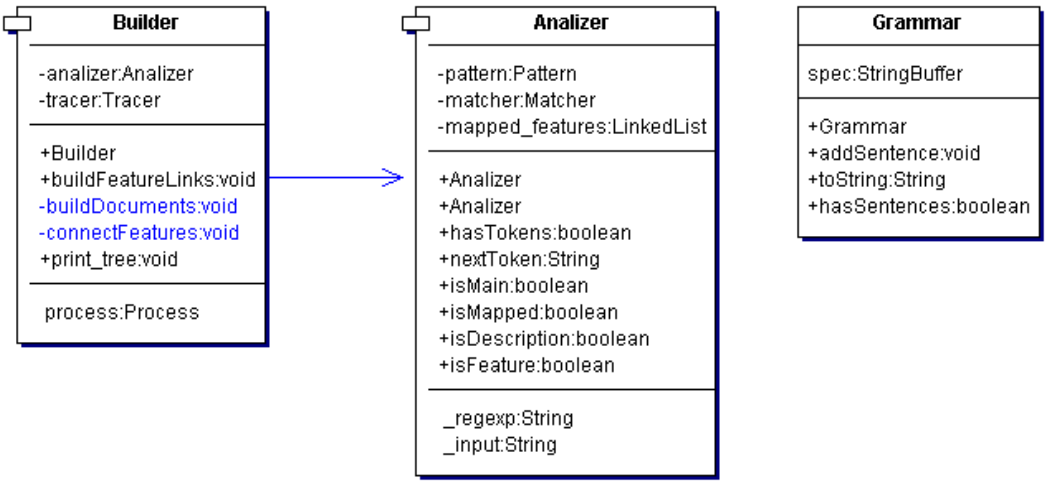


Figura 9: Especificação das classes Grammar, Analizer e Builder.

O analisador e o montador são o coração do aplicativo. Eles são responsáveis pela identificação dos requisitos de cada fase do processo e posteriormente pelo rastreabilidade dos requisitos ao longo do processo. Essa dinâmica de montagem dos documentos e rastreabilidade dos requisitos estão especificados nos diagramas de seqüência das figuras 10 e 11 respectivamente:

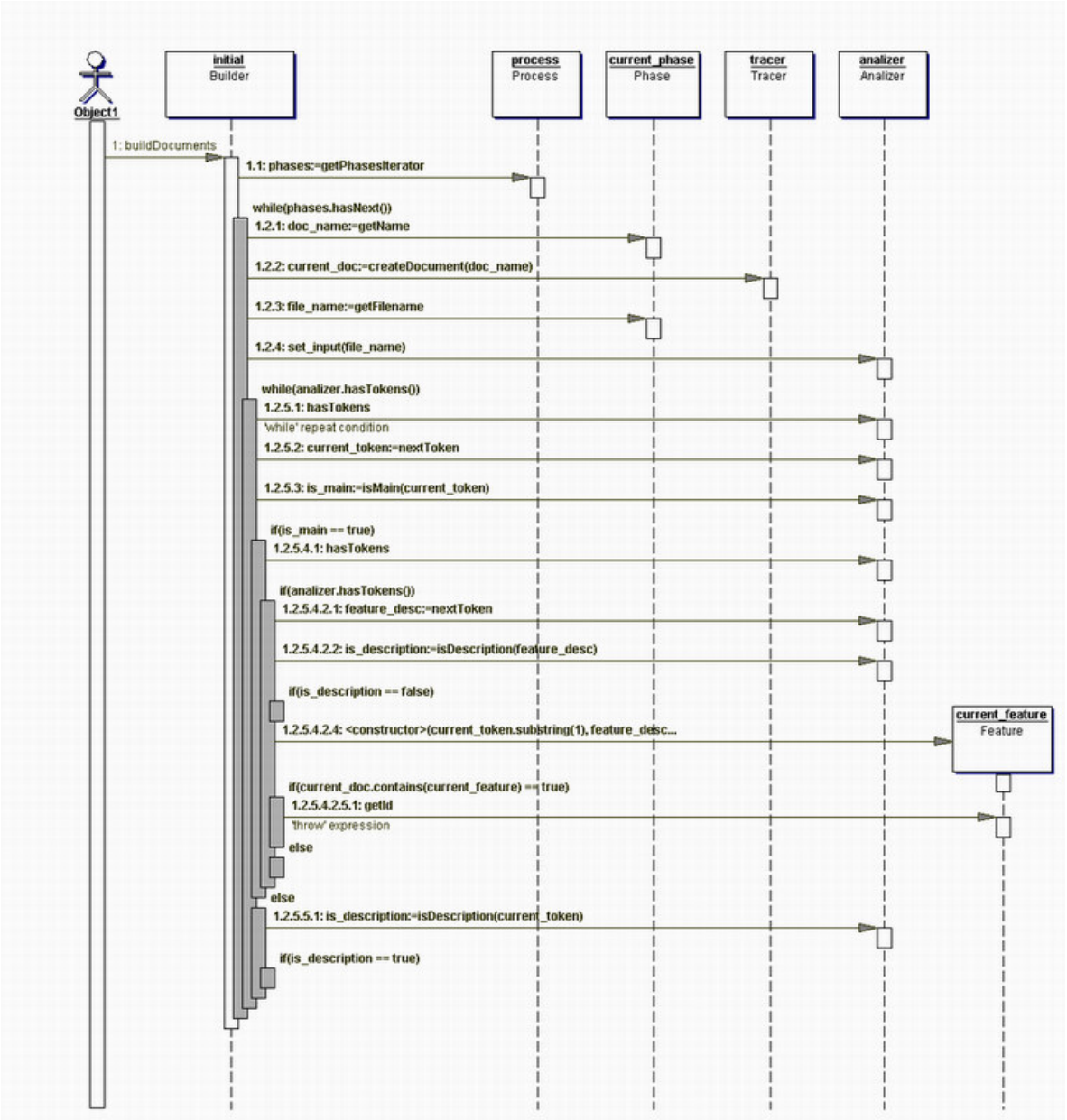


Figura 10: Diagrama de seqüência identificação de requisitos.

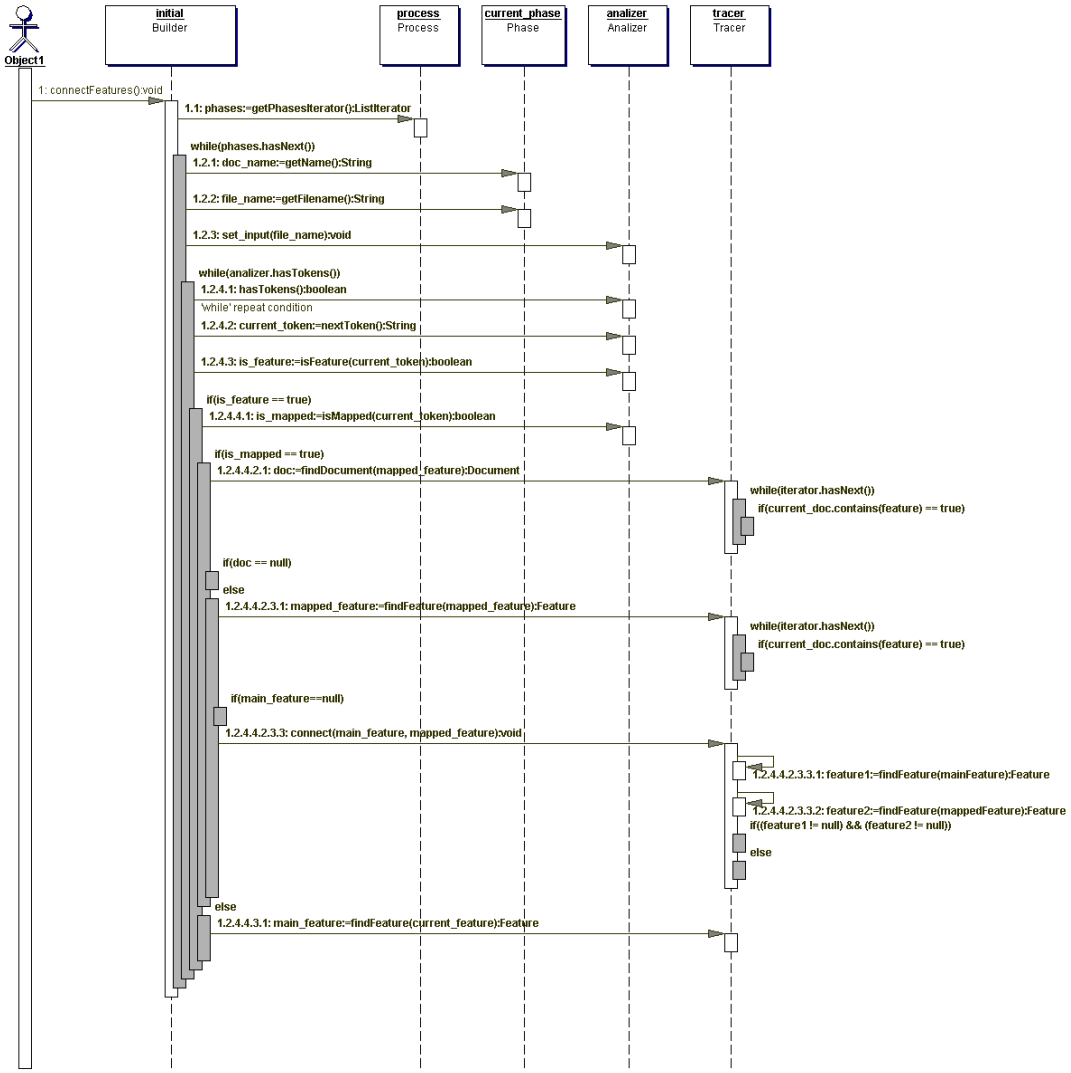


Figura 11: Diagrama de seqüência rastreabilidade de requisitos.

8.3 Implementação

O aplicativo implementado é constituído de uma interface principal onde podemos criar um novo projeto ou abrir um projeto previamente configurado (veja figura 12).

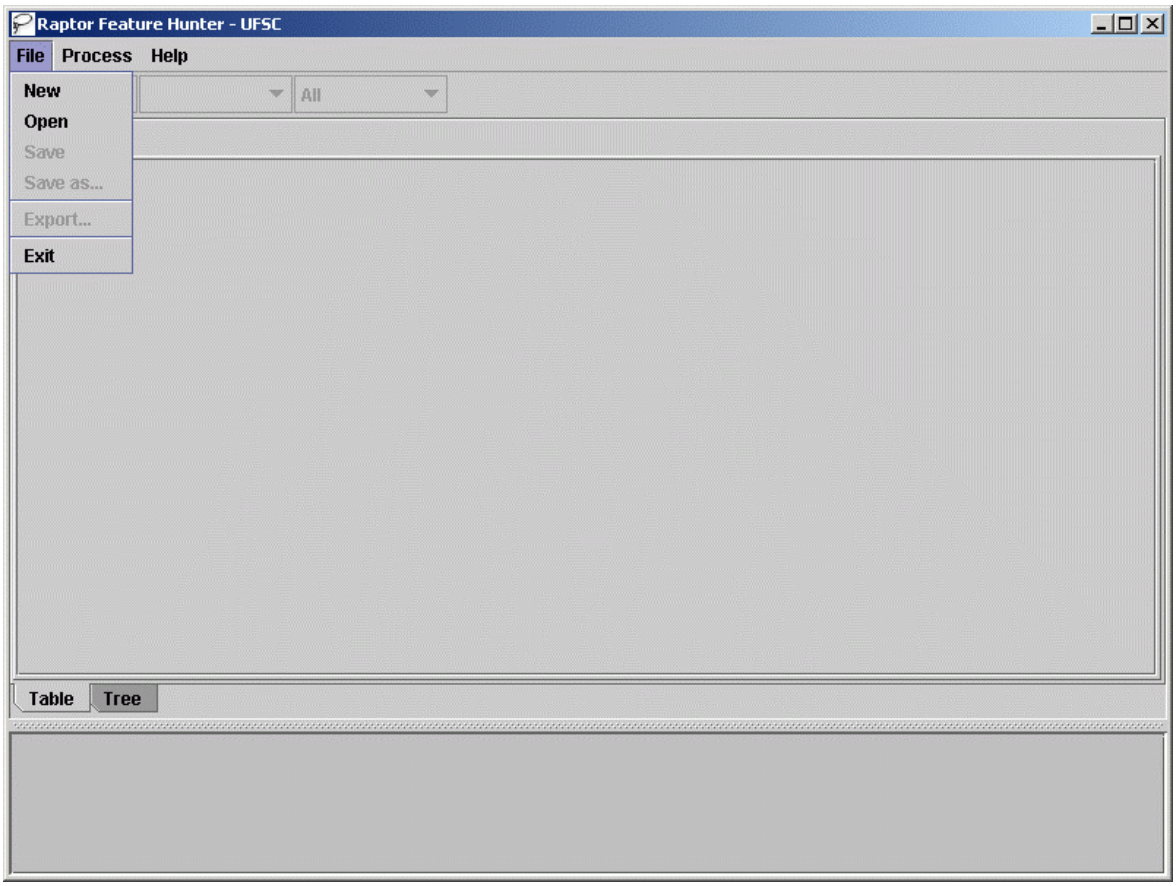


Figura 12: Interface inicial.

Após a abertura ou a criação de um novo projeto, pode-se iniciar a configuração das fases do ciclo de desenvolvimento que se deseja mapear. Para tanto, é preciso utilizar a interface de edição de processos (veja figura 13) a qual é acessada através do item de menu "Process" da interface principal.

O editor de processos permite a inserção e a retirada de fases do processo de software. Além disso, é possível configurar o nome de cada fase (Ex: Requisitos), o identificador de requisitos (Ex: #REQ_00001 – O identificador será REQ), a quantidade de números do mesmo (Ex: #REQ_00001 – 5 números) e o diretório que contém os documentos

relacionados com a sua respectiva fase.

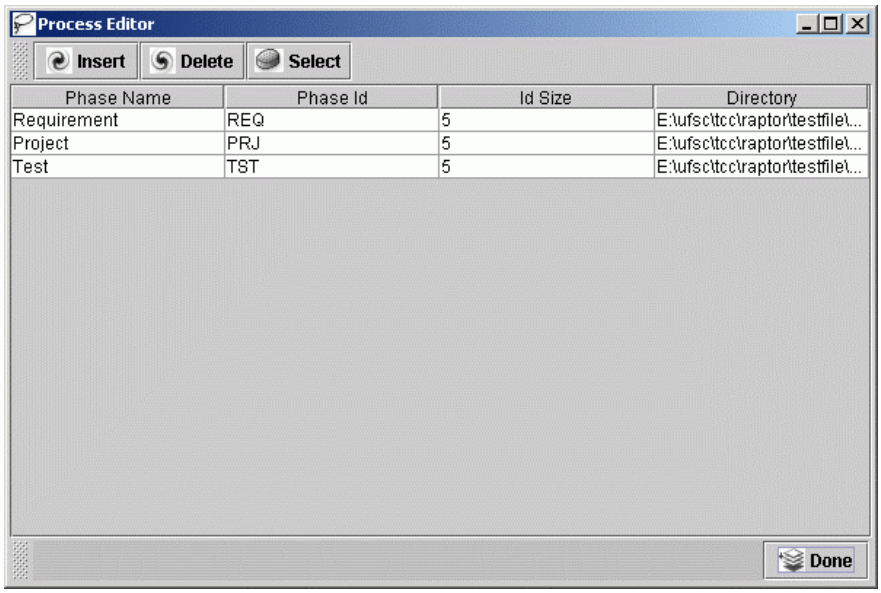


Figura 13: Interface de edição de processo.

Com o processo configurado pode-se criar a tabela (veja figura 14) e a árvore (veja figura 15) de rastreabilidade dos requisitos de uma determinada fase do processo ao longo das fases subseqüentes do mesmo.

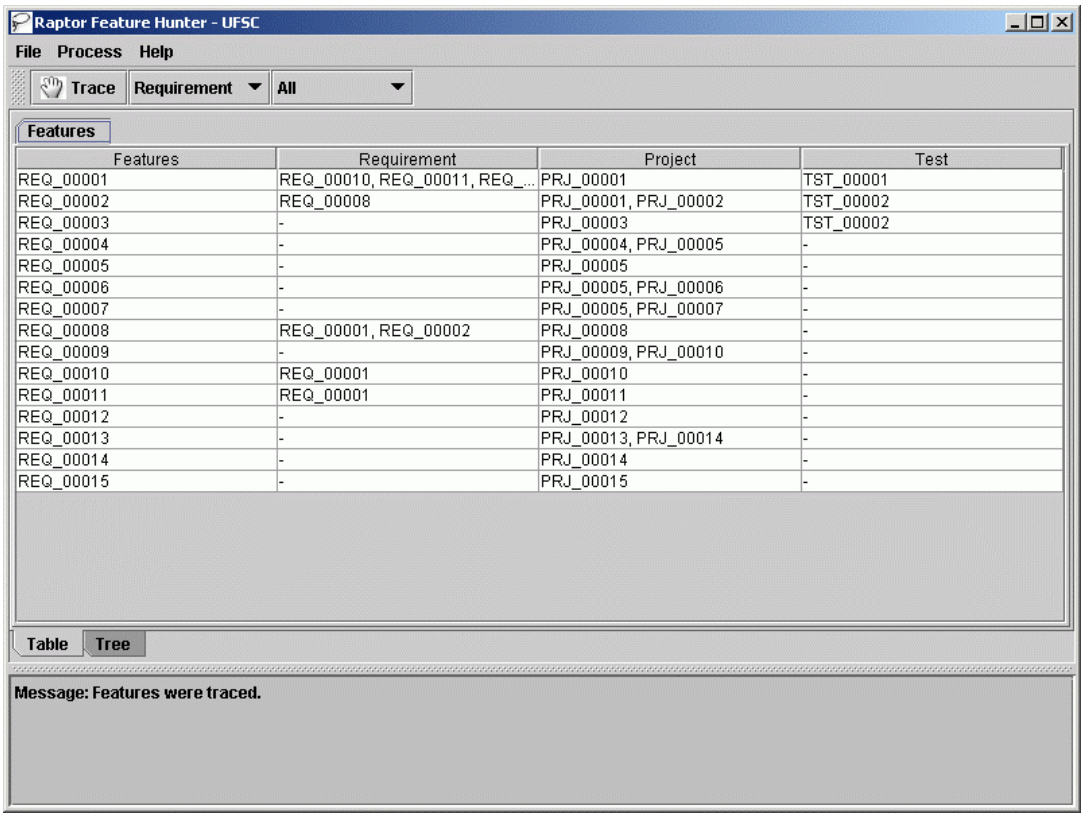


Figura 14: Interface da tabela de rastreabilidade dos requisitos.

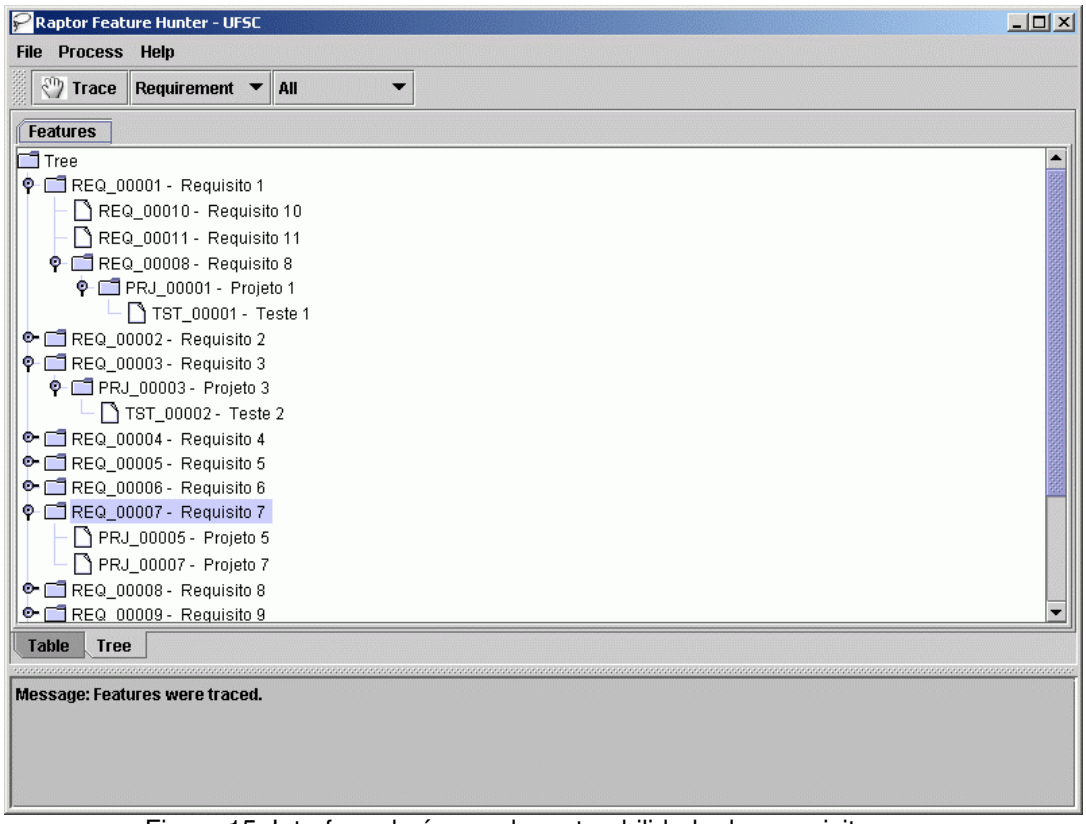


Figura 15: Interface da árvore de rastreabilidade dos requisitos.

A seção seguinte demonstra a utilização da ferramenta Tracer em um projeto de desenvolvimento real da própria ferramenta Tracer.

4.3.1 Testes (Estudo de caso real)

Apresenta-se nessa seção um exemplo real do aplicativo Tracer com o objetivo de elucidar a aplicablidade da ferramenta em um projeto de desenvolvimento de software real.

Após a definição do processo de desenvolvimento de software e suas respectivas fases (Ex: Fase Requisitos, Fase Projeto e Fase Código), a primeira tarefa a ser realizada é a criação dos documentos do projeto. Na maioria dos casos utiliza-se apenas um único documento em cada fase do processo, mas isso não impede que utilize-se múltiplos documentos (Ex: 2 documentos escritos no aplicativo Word que identificam os Requisitos do projeto).

Os documentos de requisitos, projetos e código do aplicativo Tracer podem ser escritos utilizando-se qualquer editor capaz de salvar o arquivo no formato html. Nesse exemplo, utiliza-se o aplicativo Microsoft Word na criação dos mesmos (Veja figura 16). Os documentos de requisitos, projeto e código estão descritos no anexo 1.

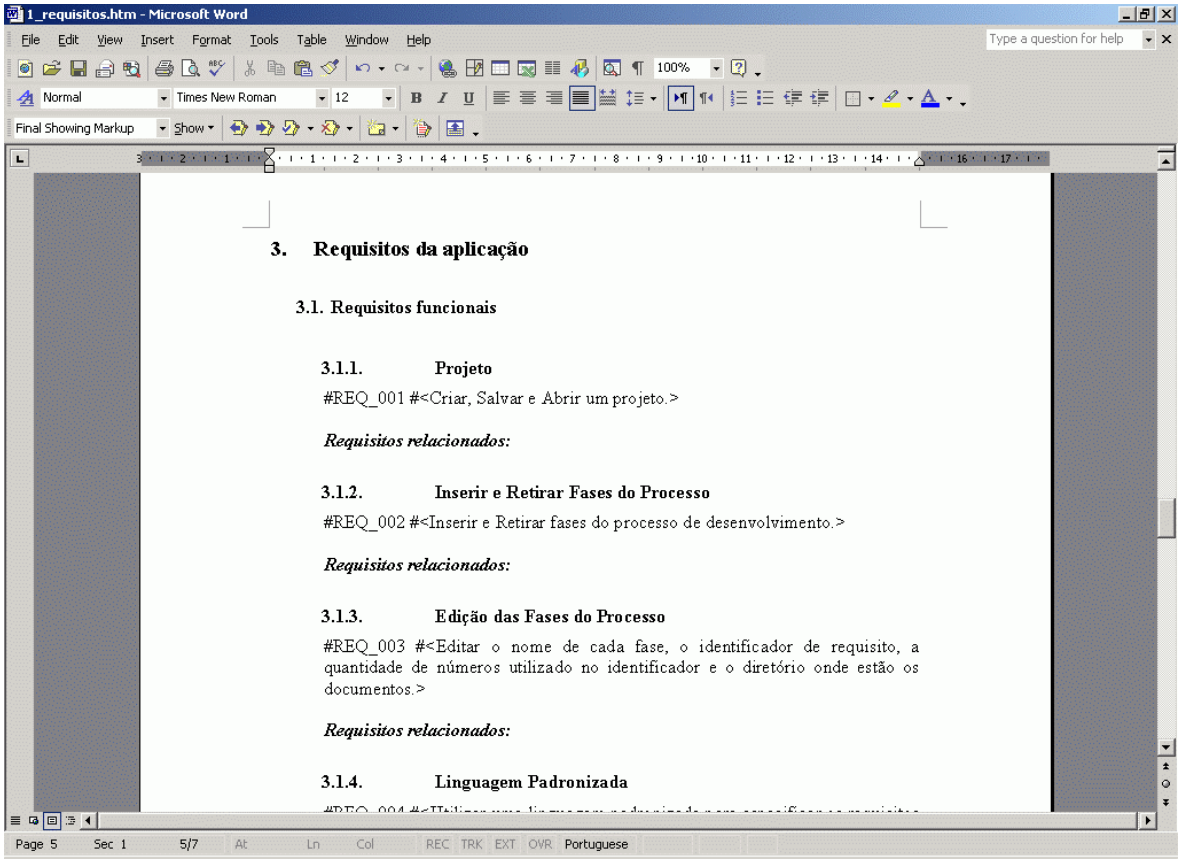


Figura 16: documento de requisitos do aplicativo Tracer.

Após ser feita a documentação do projeto (Ex: Documento de Requisitos, Projeto e Código), deve-se armazenar os documentos em uma estrutura de diretórios, onde cada diretório representa uma única fase do processo de desenvolvimento (Ex: O diretório "c:\tracer\exemplo\1requirement" armazena os documentos da fase de Requisitos).

Nesse caso cria-se 3 diretórios (Ex: "c:\tracer\exemplo\1requirement", "c:\tracer\exemplo\2project" e "c:\tracer\exemplo\1code") que representam as fases do processo de desenvolvimento do aplicativo Tracer e onde posteriormente armazena-se os respectivos documentos do projeto (Ex: Documento "1_requisitos.htm" é armazenado em "c:\tracer\exemplo\1requirement"). A árvore de diretórios está ilustrada na figura 17.

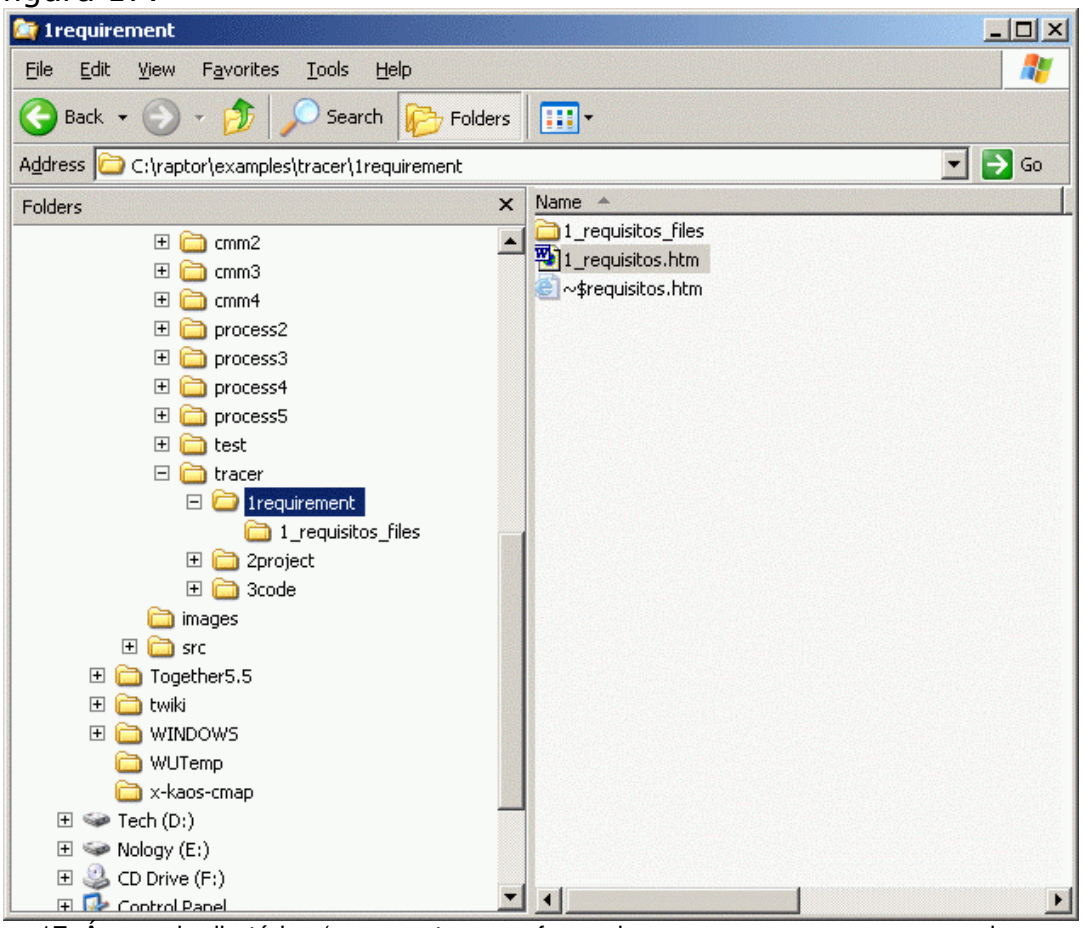


Figura 17: Árvore de diretórios (representa-se as fases do processo e armazena-se os documentos do projeto).

Em sequência à criação dos documentos e ao armazenamento dos

mesmos em uma árvore de diretórios, passa-se a utilizar o aplicativo Tracer.

Assim, o primeiro passo contitui-se em executar o aplicativo Tracer e em seguida clicar em "File->New" (veja figura 12) onde seleciona-se o diretório principal do projeto (Ex: "C:\raptor\examples\tracer") e o nome do arquivo de projeto (Ex: tracer). Finalmente, cria-se o arquivo de projeto "C:\raptor\examples\tracer\tracer.rpt" do aplicativo Tracer.

O segundo passo resume-se em editar as fases do processo de desenvolvimento. É necessário acessar o editor de processos através do item de menu "Process->Create". Neste editor, deve-se inserir as três fases (Ex: Requisitos, Projeto e Código) do processo do aplicativo Tracer.

Além disso, configura-se cada fase do processo com o nome de cada fase do processo (Ex: Requisitos), o identificador de requisitos de cada fase do processo (Ex: REQ para identificadores do tipo REQ_XXXXX), o número de dígitos utilizados no identificador (Ex: O identificador REQ_00001 utiliza 5 dígitos) e a localização dos documentos de cada fase (Ex: Documentos da fase de requisitos estão localizados em "c:\tracer\exemplo\1requisitos"). Após a edição das fases do processo clica-se no botão "Create" a fim de criar o processo de desenvolvimento.

As fases de desenvolvimento do projeto do aplicativo Tracer estão ilustrados na figura 18.

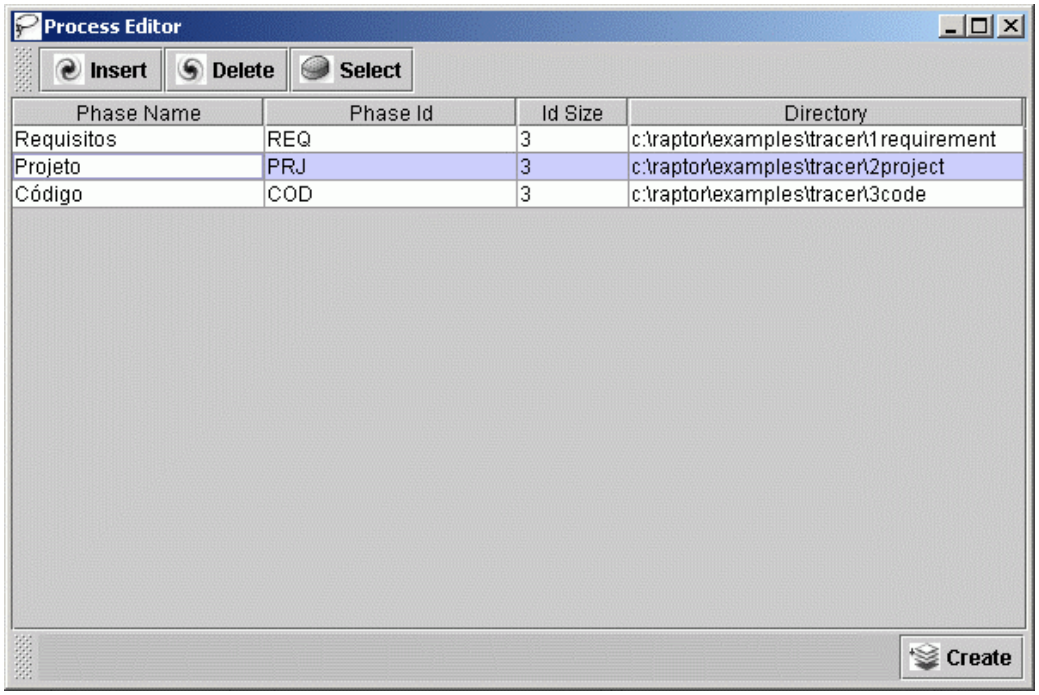


Figura 18: Fases de desenvolvimento (Requisitos, Projeto e Código).

Após a configuração do processo, pode-se realizar o mapeamento automático e bidirecional dos requisitos de uma determinada fase (Ex: Requisitos, Projeto ou Código) do processo de desenvolvimento. Mas, antes deve-se selecionar a fase que deverá ser mapeada ao longo do processo. A figura 19 demonstra a seleção da fase "Requisitos" a ser mapeada.

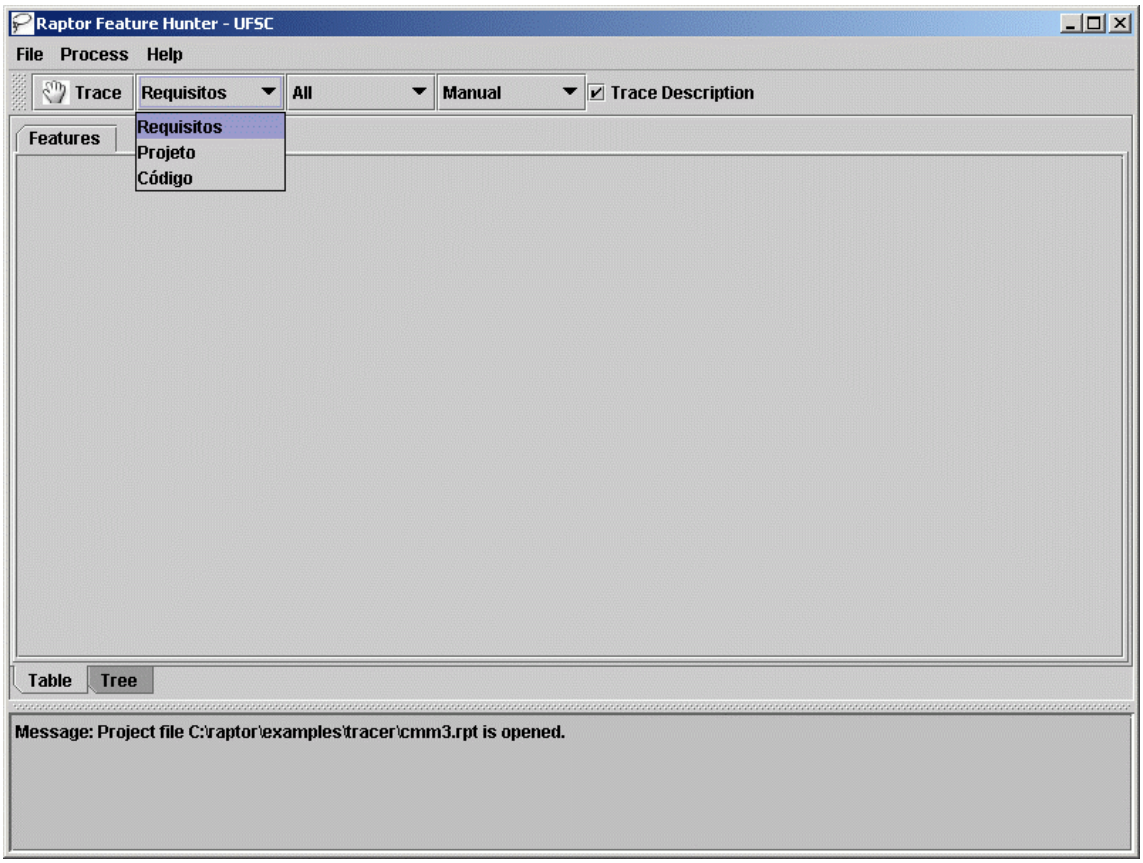


Figura 19: Seleção da fase "Requisitos" a ser mapeada ao longo do processo.

Além da fase a ser mapeada, configura-se o modo de seleção dos requisitos a serem cobertos na tabela e na árvore de traceabilidade de requisitos. Nesse caso, configura-se o modo "All – Mapear todos os requisitos" (Veja figura 20). Os modos de mapeamento estão listados a seguir:

- Mapear todos os requisitos (Ex: All);
- Mapear apenas os requisitos mapeados em todas as fases subseqüentes (Ex: Traced);
- Mapear apenas os requisitos que não são mapeados em pelo menos uma fase subseqüente (Ex: Untraced).

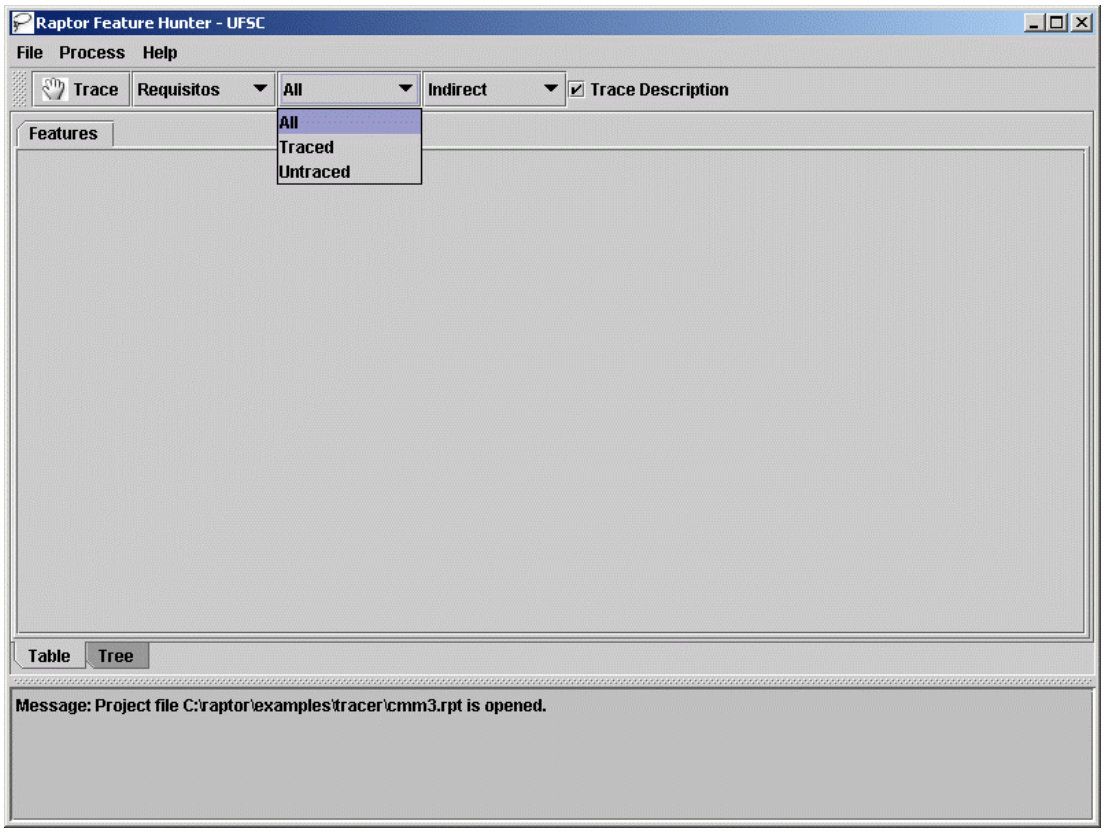


Figura 20: Seleção do modo de mapeamento "All – Mapear todos os requisitos".

A ferramenta também possui as seguintes funcionalidades de configuração de mapeamento de requisitos:

- ***mapeamento manual de requisitos:***

Ex: "Manual" - Apenas os requisitos especificados pelo usuário são mapeados. O requisito #REQ_001 da fase de "Requisitos" foi mapeado no requisito #PRJ_001 da fase de "Projeto". O requisito #PRJ_001 da fase de "Projeto" foi mapeado no requisito #COD_001 da fase de "Código". Nesse caso o #REQ_001 será mapeado apenas na fase de "Projeto". Veja tabela de traceabilidade de requisitos abaixo:

Requisitos	**Projeto**	**Código**
REQ_001	PRJ_001	-

- ***mapeamento automático e indireto de requisitos***

Ex: "Indirect" - Todos os requisitos especificados são mapeados indiretamente. O requisito #REQ_001 da fase de "Requisitos" foi mapeado no requisito #PRJ_001 da fase de "Projeto". O requisito #PRJ_001 da fase de "Projeto" foi mapeado no requisito #COD_001 da fase de "Código". Nesse caso o #COD_001 será mapeado nas fase de

"Projeto" e de "Código". Veja tabela de traceabilidade de requisitos abaixo:

Requisitos	Projeto	Código
REQ_001	PRJ_001	COD_001

No exemplo de caso real do aplicativo configura-se o modo de mapeamento "Indirect" como exposto na figura 21.

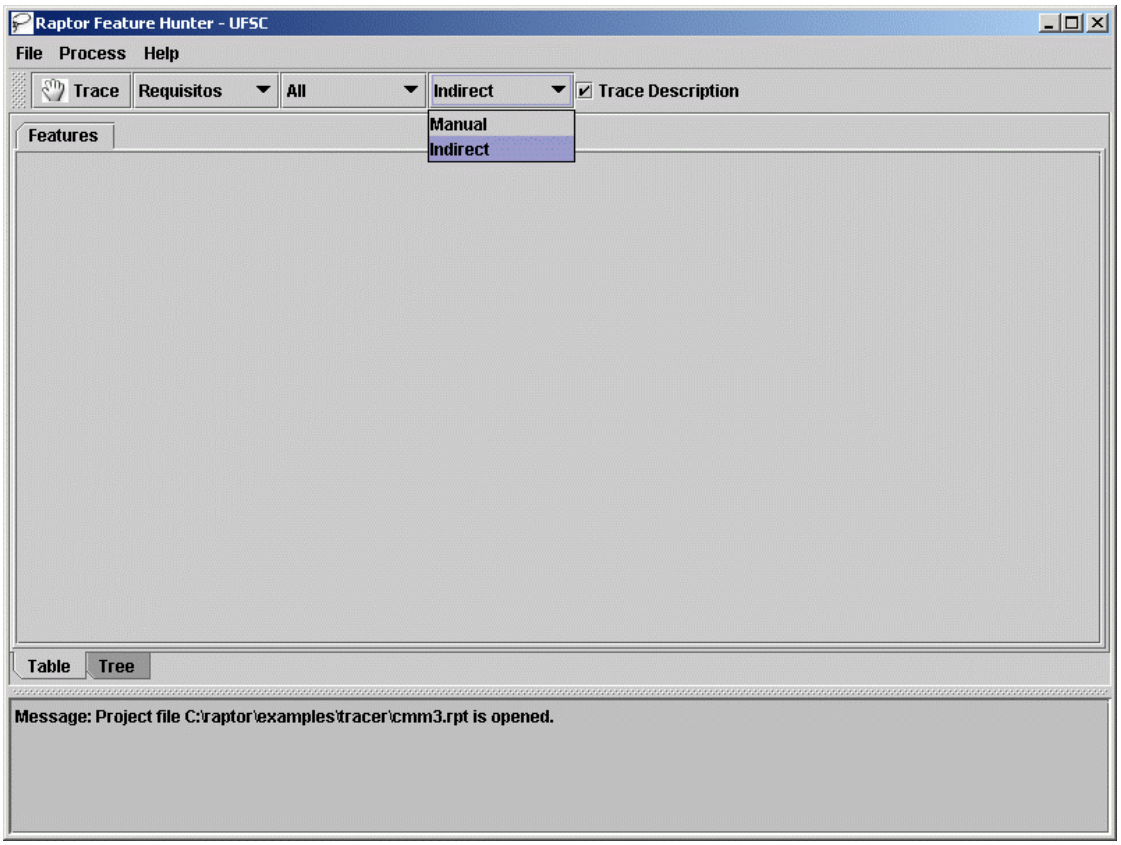


Figura 21: Seleção do modo de mapeamento "Indirect".

Após a configuração da ferramenta, pode-se partir para a criação da tabela e da árvore de mapeamento automático e bidirecional de requisitos. Para tanto, clica-se no botão "Trace". As figuras 22 e 23 ilustram os mapeamentos de requisitos respectivamente. A fim de demonstrar o mapeamento bidirecional, seleciona-se a fase de "Projeto" que será mapeada ao longo das fases subseqüentes e realiza-se o mapeamento da mesma como ilustrado na figura 24.

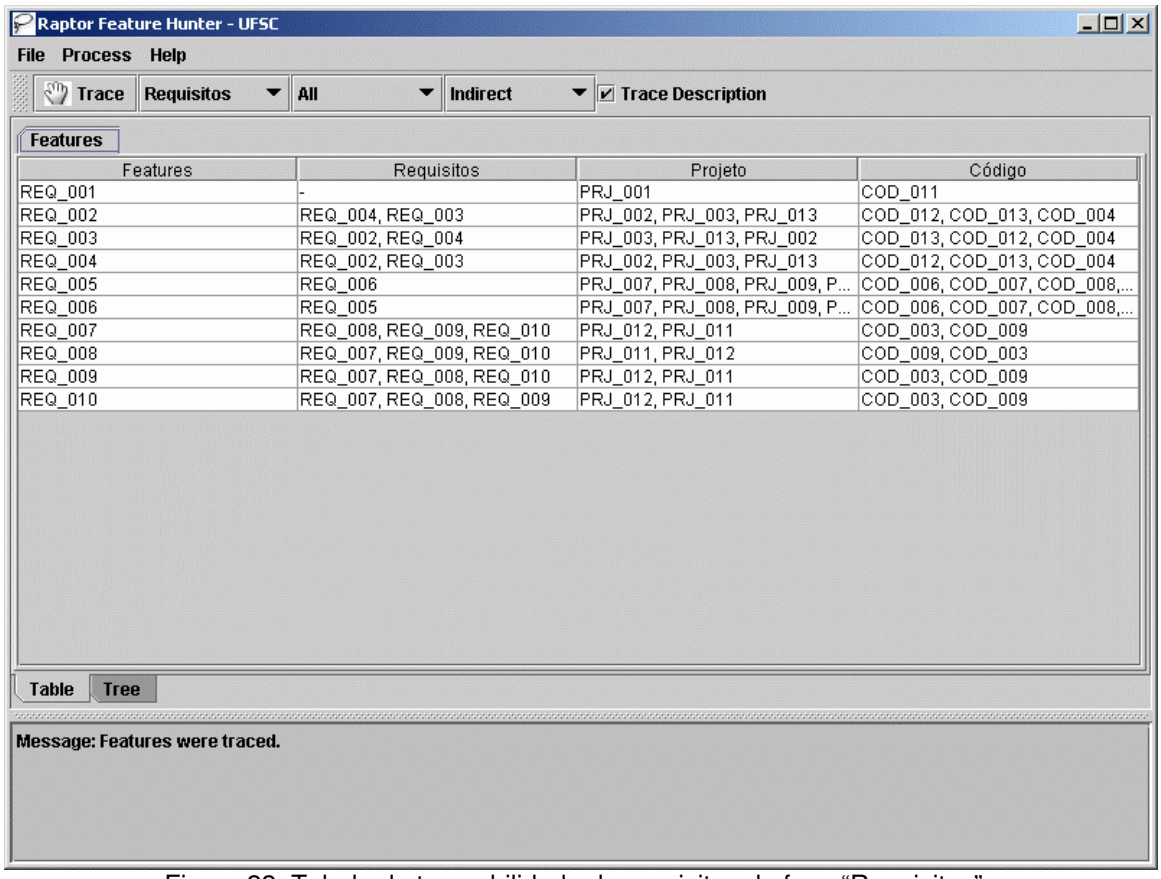


Figura 22: Tabela de traceabilidade de requisitos da fase “Requisitos” .

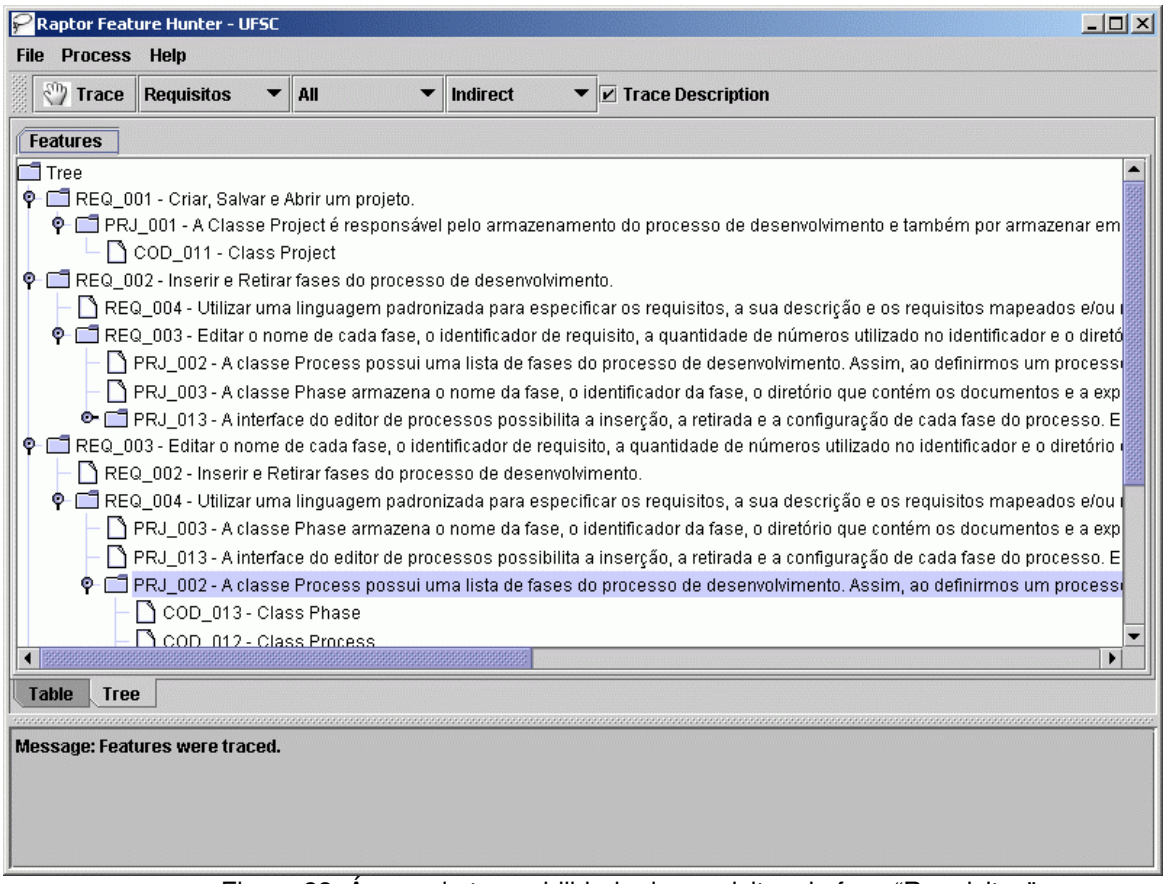
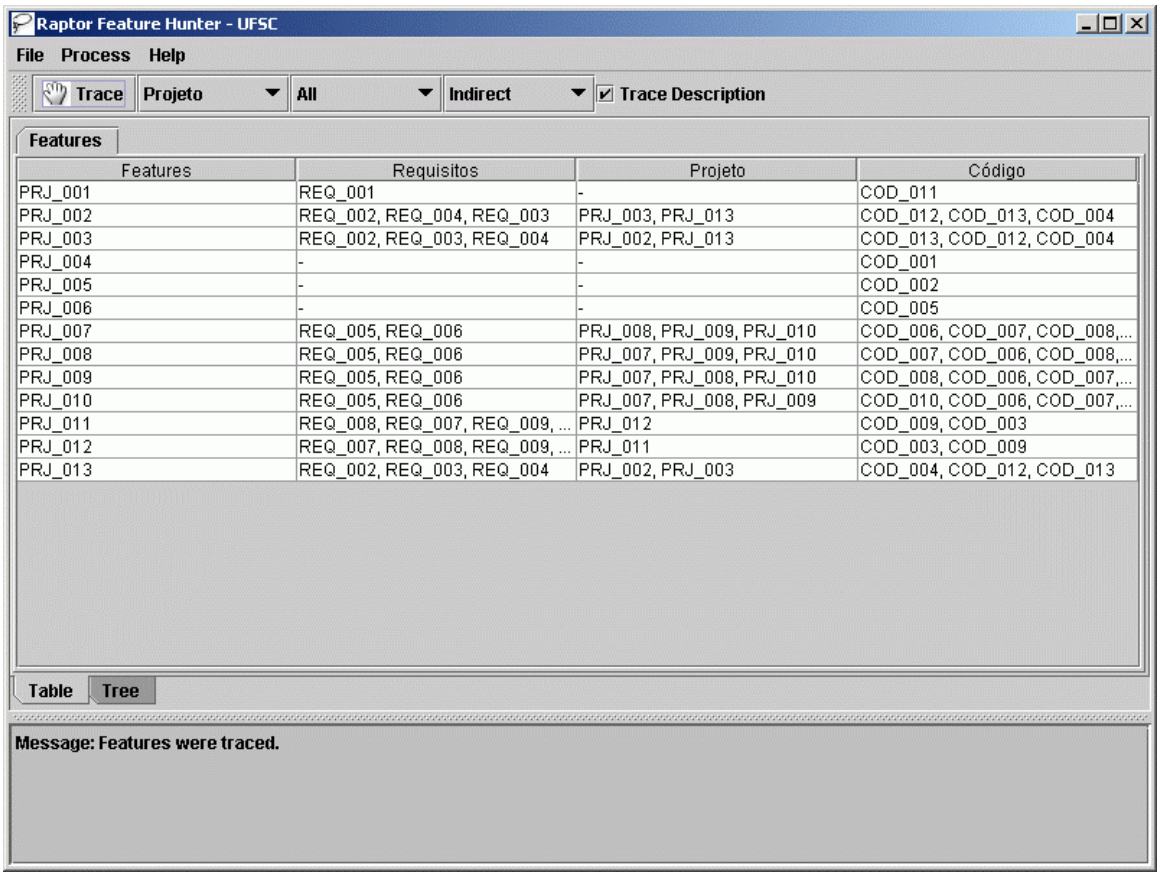


Figura 23: Árvore de traceabilidade de requisitos da fase “Requisitos” .



Features	Requisitos	Projeto	Código
PRJ_001	REQ_001	-	COD_011
PRJ_002	REQ_002, REQ_004, REQ_003	PRJ_003, PRJ_013	COD_012, COD_013, COD_004
PRJ_003	REQ_002, REQ_003, REQ_004	PRJ_002, PRJ_013	COD_013, COD_012, COD_004
PRJ_004	-	-	COD_001
PRJ_005	-	-	COD_002
PRJ_006	-	-	COD_005
PRJ_007	REQ_005, REQ_006	PRJ_008, PRJ_009, PRJ_010	COD_006, COD_007, COD_008,...
PRJ_008	REQ_005, REQ_006	PRJ_007, PRJ_009, PRJ_010	COD_007, COD_006, COD_008,...
PRJ_009	REQ_005, REQ_006	PRJ_007, PRJ_008, PRJ_010	COD_008, COD_006, COD_007,...
PRJ_010	REQ_005, REQ_006	PRJ_007, PRJ_008, PRJ_009	COD_010, COD_006, COD_007,...
PRJ_011	REQ_008, REQ_007, REQ_009, ...	PRJ_012	COD_009, COD_003
PRJ_012	REQ_007, REQ_008, REQ_009, ...	PRJ_011	COD_003, COD_009
PRJ_013	REQ_002, REQ_003, REQ_004	PRJ_002, PRJ_003	COD_004, COD_012, COD_013

Figura 24:Tabela de traceabilidade de requisitos da fase “Projeto” .

Após a criação da tabela de traceabilidade de requisitos, pode-se armazenar a mesma em um arquivo no formato html. Para isso, seleciona-se o item de menu “File->Export” e na seqüencia digita-se o nome do arquivo html a ser salvo. A figura 25 ilutra a tabela de mapeamento de requisitos da fase “Projeto” no formato html.

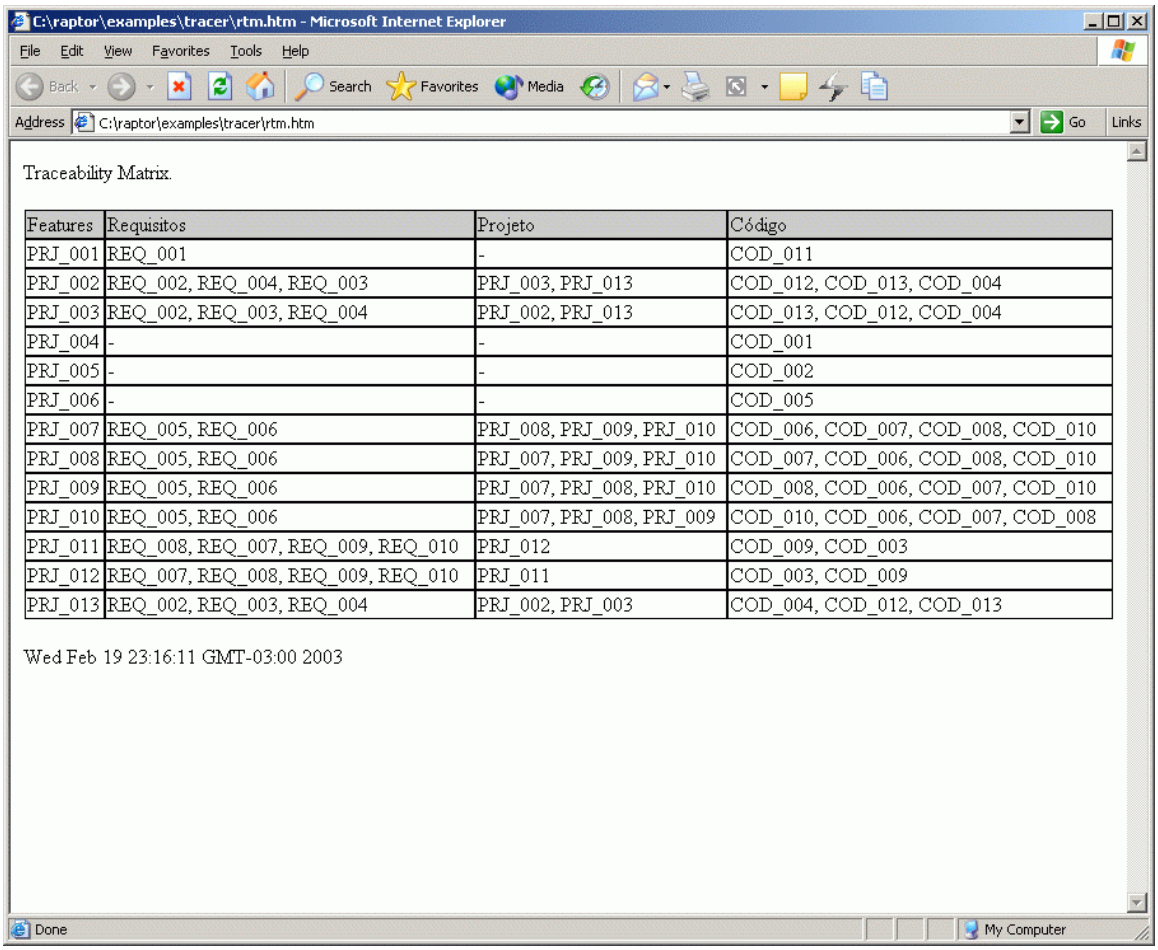


Figura 25:Tabela de traceabilidade de requisitos da fase “Projeto” convertida em html.

Com isso, demonstrou-se a aplicabilidade da ferramenta Tracer em um projeto real de desenvolvimento de software.

9 Conclusões

Pela experiência prática adquirida no estágio UFSC/Motorola, entendeu-se a importância e o significado, primeiro do uso de um processo disciplinado como ferramenta indispensável ao desenvolvimento de produtos e serviços de qualidade e com produtividade. Por outro lado, percebeu-se a forte necessidade de se automatizar cada etapa de controle e geração de documentos o tanto quanto possível. Consumindo grande parte das horas de trabalho e sendo consideradas 'não produtivas', as tarefas burocráticas, embora essenciais, ainda provocam muitas resistências por parte de analistas e programadores.

A questão da burocracia envolvida nos processos de melhoramentos das organizações, também nos faz levantar uma outra questão, já abordada por Marins e outros, que é o perigo do engessamento das organizações. Presas a normas, práticas e procedimentos muito rígidos e que podem consumir grande parte das horas de trabalho, as organizações correm o risco de não darem a atenção devida a atividades que gerem produtos e serviços inovadores. O risco é que um meio se torne um fim por si mesmo.

Diante do exposto acima, concluímos que é fundamental a automatização de tarefas burocráticas e sugerimos que o modelo CMMI adote a prática da inovação como uma disciplina nova, a ser incorporada ao modelo. O objetivo é o de diminuir os riscos da chamada 'camisa de força', presente nos processos de melhoramento.

10 Referências Bibliográficas

[CMM 94] Carnegie Mellon University / Software Engineering Institute. The Capability Maturity Model: Guidelines for improving the software process. Pittsburgh: Ed. Addison –Wesley, 1994. 451p.

[CCR 02] Carnegie Mellon University / Software Engineering Institute. Capability Maturity Model Integration (CMMISM) – Continuous Representation, Version 1.1, 2002, 645p.

[CSR 02] Carnegie Mellon University / Software Engineering Institute. Capability Maturity Model Integration (CMMISM) – Staged Representation, Version 1.1, 2002, 639p.

[JAC 92] Jacobson, Ivar et al. Object-Oriented Software Enginneering – A Use Case Driven Approach. Harlow: Ed. Addison - Wesley, 1992, 528p.

[RUM 94] Rumbaugh, James et al. Modelagem e Projetos Baseados em Objetos. São Paulo: Ed. Campus, 1994, 652p.

[PHI 02] Phillips, Mike. CMMI V1.1 Tutorial: E-SEPG. Carnegie Mellon University / Software Engineering Institute. Pittsburgh, PA. <http://www.inf.ufsc.br/~ricardo/download/cmmi>, acessado em 21 de dezembro de 2002.

[WAK 00] Wake, William C. Extreme Programming Explored. Halow: Ed. Addison – Wesley, 2000. 158p.

11 Anexos

11.1 Documento de Requisitos do aplicativo Tracer

Universidade Federal de Santa Catarina

Projeto: *Tracer*
Aplicação*: Tracer System*

Especificação de Requisitos do Tracer

Identificador do Documento: REQ
Versão (draft): 1.0
Data: 2/2/2003

Histórico do documento

Versão	Autor(es)	Data	Ação	Esforço (em horas, especificado por autor)
1.0	Carlos Matias Ricardo Winck	2/2/2003	Criação do documento	4 horas

Conteúdo

Introdução

Objetivo do desenvolvimento

Este documento tem como objetivo identificar os requisitos do projeto Tracer.

Definições, abreviaturas

UFSC - Universidade Federal de Santa Catarina

Referências

Nenhuma.

Localização

/tracer/examples/tracer/1requirement/1_requisitos.htm

Visão geral do sistema

Arquitetura da aplicação

O aplicativo Tracer consiste principalmente em montar a matriz de traceabilidade de requisitos de forma automática e bidirecional.

Premissas de desenvolvimento

Utiliza-se a linguagem de programação java no desenvolviemnto do aplicativo Tracer.

Requisitos da aplicação

Requisitos funcionais

Projeto

#REQ_001 #<Criar, Salvar e Abrir um projeto.>

Requisitos relacionados:

Inserir e Retirar Fases do Processo

#REQ_002 #<Inserir e Retirar fases do processo de desenvolvimento.>

Requisitos relacionados:

Edição das Fases do Processo

#REQ_003 #<Editar o nome de cada fase, o identificador de requisito, a quantidade de números utilizado no identificador e o diretório onde estão os documentos.>

Requisitos relacionados:

Linguagem Padronizada

#REQ_004 #<Utilizar uma linguagem padronizada para especificar os requisitos, a sua descrição e os requisitos mapeados e/ou relacionados.>

Requisitos relacionados:

Mapeamento Bidirecional e Automático

#REQ_005 #<A ferramenta permitirá o mapeamento de todos os requisitos de todas as fases do processo de forma bidirecional e automática.>

Requisitos relacionados:

Mapeamento N Fases

#REQ_006 #<Mapear os requisitos de todas as fases do processo de desenvovimento.>

Requisitos relacionados:

Modos de Mapeamento

#REQ_007 #<Definir 3 formas de mapeamento de requisitos: Todos os requisitos, apenas os requisitos mapeados, apenas os requisitos não mapeados.>

Requisitos relacionados:

1.1.1. Matriz no formato html

#REQ_008 #<A matriz de traceabilidade das fases do processo será armazenada em um arquivo no formato HTML.>

Requisitos relacionados:

Requisitos de interface

Interface tabela de traceabilidade

#REQ_009 #<A matriz de traceabilidade das fases do processo deve ser mostrada em uma interface gráfica, sendo representada por uma tabela onde as colunas representam as fases do processo e os requisitos mapeados.>

Requisitos relacionados:

Interface árvore de traceabilidade

#REQ_010 #<A matriz de traceabilidade das fases do processo deve ser mostrada em uma interface gráfica, sendo representada por uma árvore>

Requisitos relacionados:

Restrições de projeto

Suporte de desenvolvimento

Por exemplo, usar UML com Rational Rose, Delphi, Oracle etc.

Plataforma de execução

Por exemplo, Windows 2000 (anteriores, posteriores?) etc.

Padrões de modularização

DLLs, nomes de arquivos (executáveis, de dados etc) etc.

Robustez, integridade e segurança

Condições adversas sob as quais a aplicação deve manter-se em operação (rede inoperante, esgotamento do tempo de resposta a alguma solicitação etc)

Integridade de arquivos de dados em panes elétricas, interrupção abrupta de procedimentos etc.

Política de classificação de usuários, com especificação de restrições de acesso a dados e funcionalidades etc.

Performance

Tempos de resposta

Manutenção

Restrições de tempo e esforço para proceder alterações emergenciais referentes a modificação de legislação ou identificação de bugs

11.2 Documento de Projeto do aplicativo Tracer

Universidade Federal de Santa Catarina - UFSC

Projeto: *Tracer*
Aplicação*: Tracer System*

Especificação de Projeto do Tracer

Identificador do Documento: PRJ
Versão (draft): 1.0
Data: 2/2/2003

Histórico do documento

Versão	Autor(es)	Data	Ação	Esforço (em horas, especificado por autor)
1.0	Carlos Matias Ricardo Winck	2/2/2003	Criação do documento.	4 horas

Conteúdo

Introdução

1.2. Objetivo do desenvolvimento

Este documento tem como objetivo identificar a arquitetura do projeto Tracer, identificando os seus principais componentes também a interface com o usuário.

1.3. Definições, abreviaturas

UFSC - Universidade Federal de Santa Catarina

1.4. Referências

Especificação de Requisitos do Projeto Tracer

1.5. Localização

/tracer/examples/tracer/2project/2_project.htm

Visão geral do sistema

1.6. Arquitetura da aplicação

O aplicativo Tracer consiste principalmente em um analisador que faz o parsing dos requisitos contidos em um documento. Esses são repassados para um montador o qual analisa sintaticamente os requisitos do documento e posteriormente monta a estrutura dos documentos.

Em seguida, o linkador conecta / traça os requisitos ao longo do processo de desenvolvimento. Após realizar o mapeamento, a tabela e a árvore de traceabilidade da fase selecionada é mostrada na interface gráfica do aplicativo Tracer.

1.7. Premissas de desenvolvimento

Utiliza-se a metodologia UML na descrição do aplicativo Tracer.

Projeto da aplicação

Processos

O uso de inúmeros processos de desenvolvimeto, devido principalmente às diferentes realidades vivenciadas, requeriu a especificação de uma classe *Project* que representa o projeto em execução e também as classes *Process* e *Phase* que representam o processo de uma determinada organização e suas respectivas fases (veja figura 1).

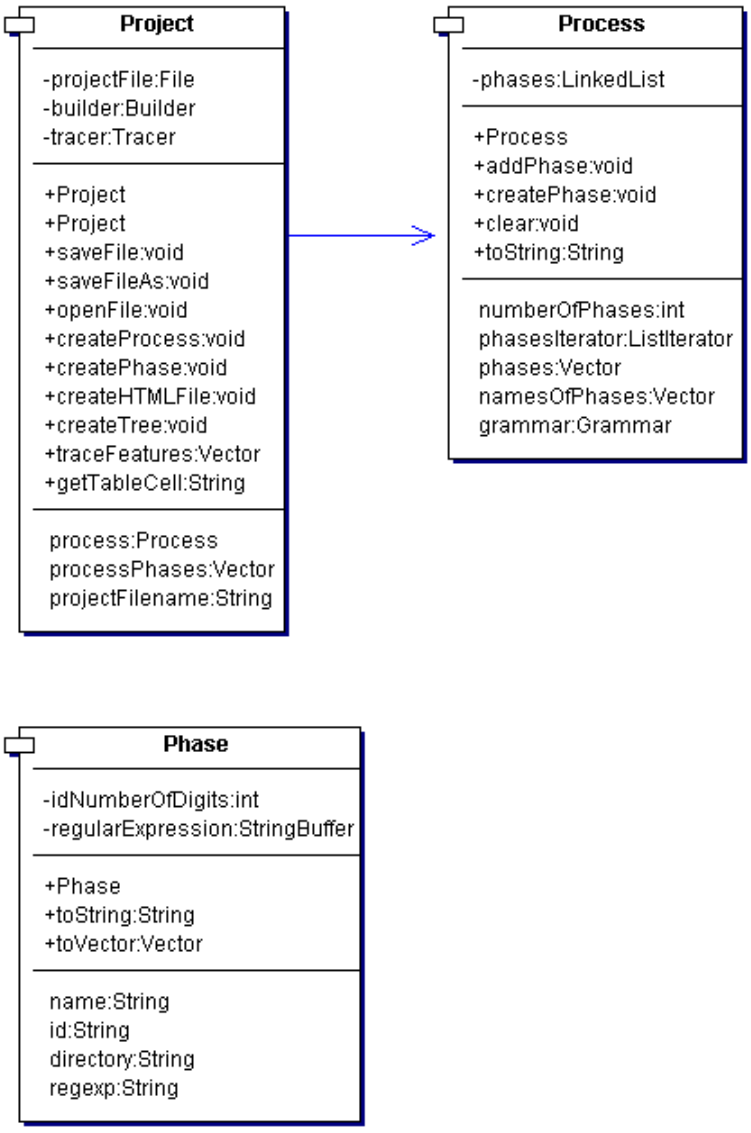


Figura 1: Diagrama das classes Project, Process e Phase.

Classe Project

#PRJ_001 #<A Classe Project é responsável pelo armazenamento do processo de desenvolvimento e também por armazenar em arquivo os dados pertinentes ao projeto. Esses incluem principalmente os dados de cada fase do processo.>

Requisitos Relacionados/Mapeados:

##REQ_001

Classe Process

#PRJ_002 #<A classe Process possui uma lista de fases do processo de desenvolvimento. Assim, ao definirmos um processo, armazenamos nessa lista a sequência de fases especificada. Além disso, também fornece a gramática a ser utilizada para fazer o parsing dos documentos.>

Requisitos Relacionados/Mapeados:

##REQ_002

Classe Phase

#PRJ_003 #<A classe Phase armazena o nome da fase, o identificador da fase, o diretório que contém os documentos e a expressão regular utilizada no parsing dos documentos.>

##REQ_002

##REQ_003

Requisitos Relacionados/Mapeados:

Documentos

Os documentos de uma determinada fase do processo de desenvolvimento estão representados nas classes *Document* e *Feature* (veja figura 2). Mesmo se utilizarmos vários documentos em uma determinada fase, esses serão representados por um único objeto da classe Document. Sendo assim, os documentos de uma mesma fase não podem incluir identificadores de requisitos idênticos.

Classe Document.

#PRJ_004 #<A classe Document é responsável pelo amazenamento de todos os requisitos encontrados em uma determinada fase do processo. Além disso, também realiza operações de ordenamento dos mesmos.>

Requisitos Relacionados/Mapeados:

Classe Feature.

#PRJ_005 #<A classe Feature armazena todas as informações pertinentes a um requisito de um documento. Essas incluem o seu identificador, a descrição do requisito e uma lista com todos os requisitos mapeados e/ou relacionados.>

Requisitos Relacionados/Mapeados:

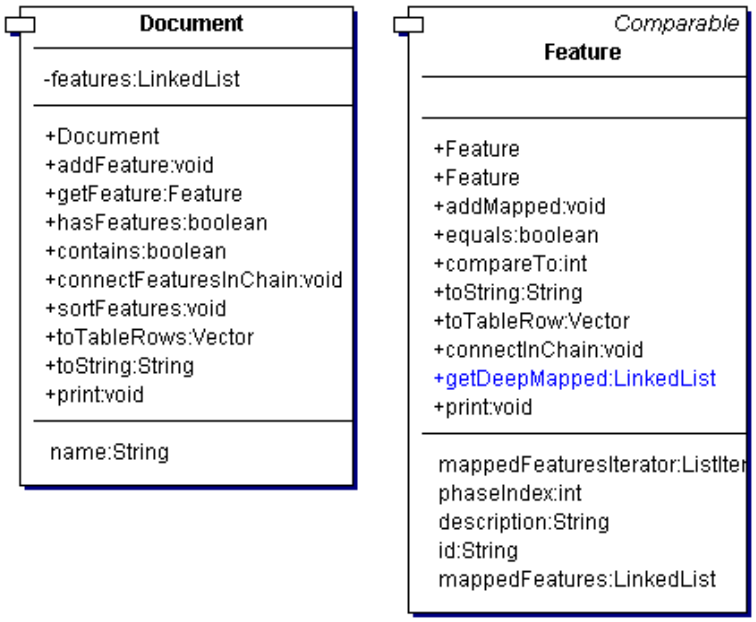


Figura 2: Diagrama das classes Document e Feature.

Tracer

Utilizaremos um analisador, classe *Analizer*, para fazer o parsing dos tokens nos dodumentos e também um montador, classe *Builder*, para verficar a estrutura sintática dos tokens fornecidos pelo analisador. Esse utiliza uma gramática, classe *Grammar*, fornecida pelo objeto da classe Process (veja figura 1) para fazer o parsing dos tokens.

O fato de traçarmos os requisitos ao longo das fases do processo, levou à especificação de um linkador, classe *Tracer*. Este é responsável por conectar os requisitos e montar a tabela de traceabilidade de requisitos, classe *Table*, resultante (veja figura 3).

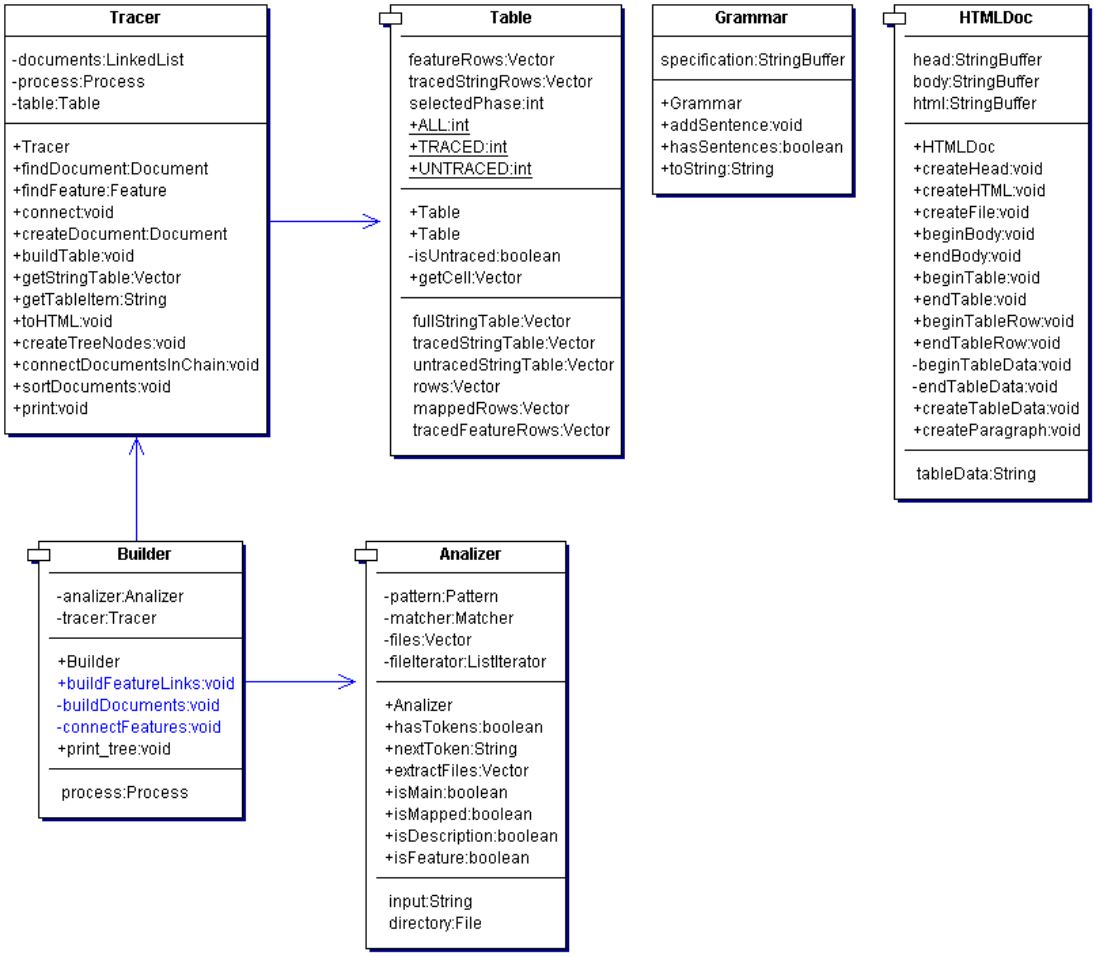


Figura 3: Diagrama das classes Analizer, Builder, Tracer, Table, Grammar e HTMLDoc.

Classe Analizer.

#PRJ_006 #<A classe Analizer é responsável pelo parsing dos tokens da linguagem definida de acordo com as fases do processo de desenvolvimento. Por ouro lado, o analisador verifica o tipo do token>

Requisitos Relacionados/Mapeados:

Classe Builder.

#PRJ_007 #<A classe Builder constrói todos os documentos e seus respectivos requisitos e também realiza a análise sintática de todos os tokens fornecidos pelo analisador. Posteriormente, os envia para o linkador a fim de conectar os requisitos e montar a tabela de traceabilidade de requitos.>

Requisitos Relacionados/Mapeados:

##REQ_005
##REQ_006

Classe Tracer.

#PRJ_008 #<A classe Tracer conecta o requisito mapeado, colocando-o na lista de requisitos mapeados do requisito principal. A operação inversa também é efetuada a fim de possibilitar o mapeamento bidirecional dos requisitos. Além disso, monta e fornece a tabela de traceabilidade de requisitos de uma determinada fase.>

Requisitos Relacionados/Mapeados:

##REQ_005
##REQ_006

Classe Table.

#PRJ_009 #<A classe Table conecta o requisito mapeado, colocando-o na lista de requisitos mapeados do requisito principal. A operação inversa também é efetuada a fim de possibilitar o mapeamento bidirecional dos requisitos.>

Requisitos Relacionados/Mapeados:

##REQ_005
##REQ_006

Classe Grammar.

#PRJ_010 #<A classe Grammar armazena as sentenças fornecidas pelas fases, classe Phase, do processo, classe Process, e posteriormente as envia ao analisador.>

Requisitos Relacionados/Mapeados:

##REQ_005
##REQ_006

Classe HTMLDoc.

#PRJ_011 #<A classe HTMLDoc é utilizada para montarmos a tabela de traceabilidade de requisitos no formato html e em seguida gravar em um arquivo do sistema.>

Requisitos Relacionados/Mapeados:

##REQ_008

Interfaces

Modelamos duas interfaces com usuário. A primeira (Veja figura 4) constitui-se em uma janela principal onde cria-se um novo projeto e também mostra-se ao usuário a tabela e a árvore de traceabilidade dos requisitos. A segunda (Veja figura 5) possibilitará a configuração do processo de desenvolvimento em questão, como a inserção e retirada de fases do processo.

Interface do Tracer

#PRJ_012 #<A interface principal do projeto Tracer permite salvar, renomear e abrir um arquivo de projeto. Além disso mostra a árvore e a tabela de traceabilidade de acordo com o modo selecionado, All, Traced e Untraced, sendo possível exportar a última para um arquivo html. Através dessa janela, podemos também executar o editor de processos.>

Requisitos Relacionados/Mapeados:

##REQ_007
##REQ_008
##REQ_009
##REQ_010

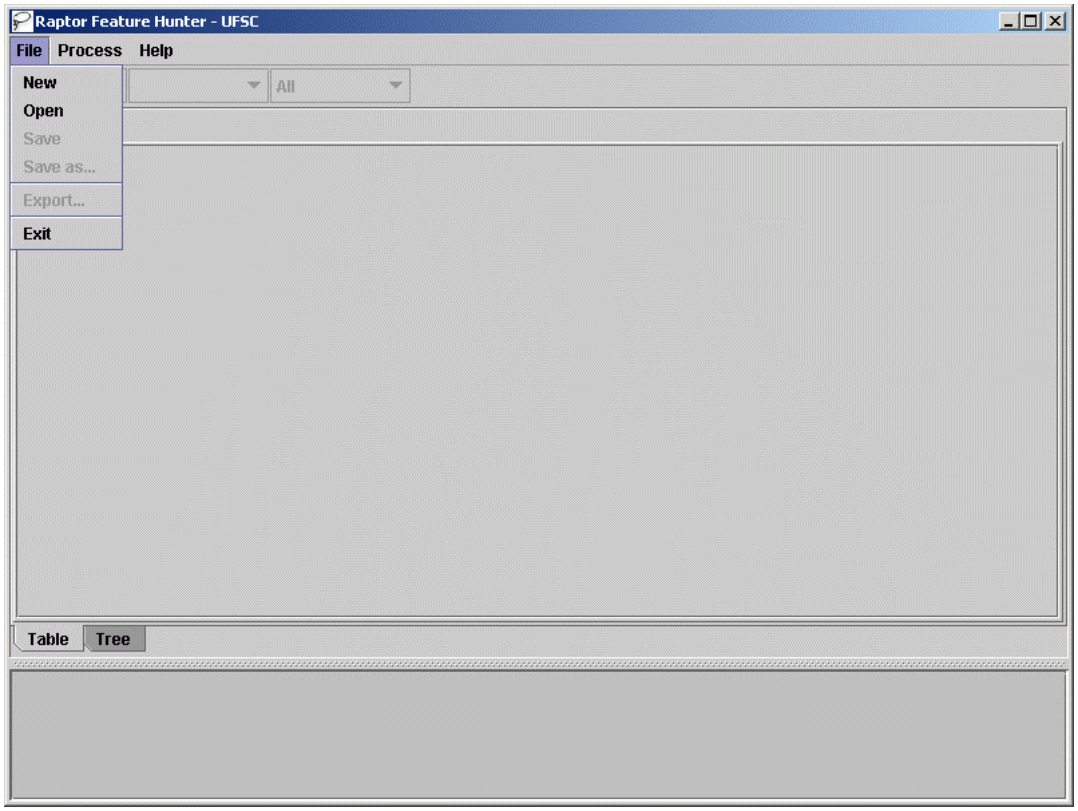


Figura 4: Interface do Tracer.

Interface do Editor de Processos

#PRJ_013 #<A interface do editor de processos possibilita a inserção, a retirada e a configuração de cada fase do processo. Esta inclui a definição do nome de cada fase, o seu identificador, o número de dígitos utilizado no identificador e o diretório onde estão os documentos.>

Requisitos Relacionados/Mapeados:

##REQ_002

##REQ_003

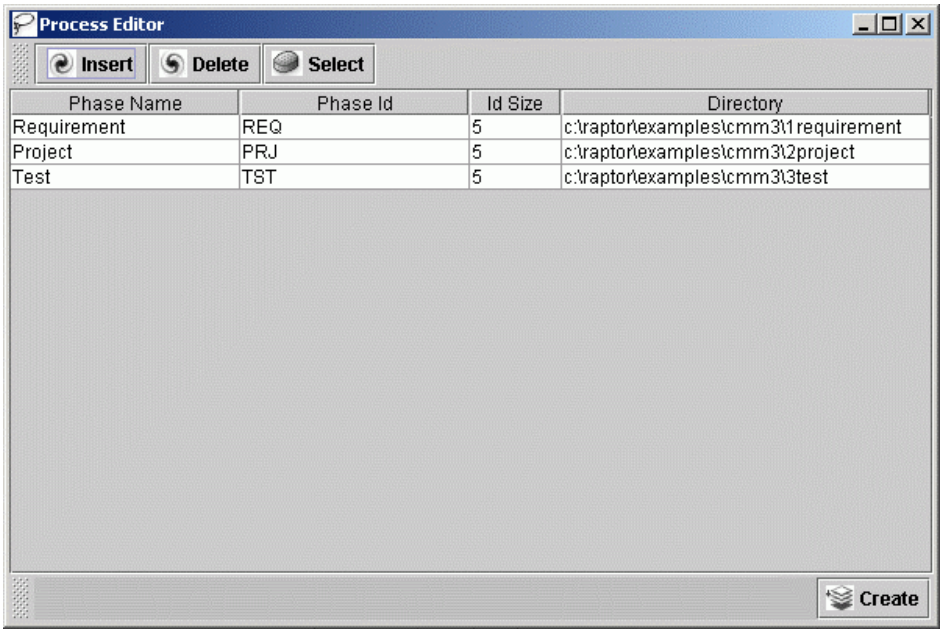


Figura 5: Interface do Editor de Processos.

11.3 Documento de Código do aplicativo Tracer

```
package project;

import java.util.*;

import parser.Grammar;

/**
 * #COD_012 #<Class Process>
 * ##PRJ_002
 * ##REQ_004
 * Process that stores phases
 * @author Carlos Matias / Ricardo Winck
 */
public class Process {
	/**
	 * phases of the current process
	 */
	private LinkedList phases;
	/**
	 * grammar to be loaded into analizer
	 */
	private Grammar grammar;

	/**
	 * Builds a new process
	 */
	public Process()
	{
		phases = new LinkedList();
	}

	/**
	 * Insert a new phase in the end of the process
	 */
	public void addPhase(Phase new_phase)
	{
		phases.addLast(new_phase);
	}

	/**
	 * Creates a new phase and inserts it in the end of the process.
	 */
	public void createPhase(String name, String id, int num_digits,
String file_name)
	{
		Phase new_phase = new Phase(name, id, num_digits, file_name);
		addPhase(new_phase);
	}

	/**
	 * Returns the process number of phases
```

```
	 * @return int
	 */
	public int getNumberOfPhases()
	{
		return phases.size();
	}

	/**
	 * Returns the process phases iterator
	 * @return ListIterator
	 */
	public ListIterator getPhasesIterator()
	{
		return phases.listIterator();
	}

	/**
	 * Returns the process phases
	 * @return Vector
	 */
	public Vector getPhases()
	{
		Vector phases_temp = new Vector();
		ListIterator iterator = phases.listIterator();
		while (iterator.hasNext())
		{
			Phase phase = (Phase) iterator.next();
			phases_temp.add(phase.toVector());
		}
		return phases_temp;
	}

	/**
	 * Returns the names of phases
	 * @return Vector
	 */
	public Vector getNamesOfPhases()
	{
		Vector phases_temp = new Vector();
		ListIterator iterator = phases.listIterator();
		while (iterator.hasNext())
		{
			Phase phase = (Phase) iterator.next();
			phases_temp.addElement(phase.getName());
		}
		return phases_temp;
	}

	/**
	 * Returns the grammar to be parsed along the process phase
documents.
	 * @return Grammar
	 */
	public Grammar getGrammar() {
```

```
            grammar = new Grammar();
            ListIterator iterator = phases.listIterator();
            while (iterator.hasNext()) {
                  Phase current_phase = (Phase) iterator.next();
                  String sentence = current_phase.getRegexp();
                  grammar.addSentence(sentence);
            }
            if (grammar.hasSentences())
            {
                  grammar.addSentence("(#&lt;([^(#)]*)&gt;)");
            }
            return grammar;
      }

      /**
       * Remove all phases of the current process
       */
      public void clear() {
            phases.clear();
      }

      /**
       * Convert the process to a string representing the process phases.
       * @return String
       */
      public String toString() {
            StringBuffer buffer = new StringBuffer();
            ListIterator iterator = phases.listIterator();
            while (iterator.hasNext())
            {
                  Phase phase = (Phase) iterator.next();
                  buffer.append(phase.toString());
            }
            return buffer.toString();
      }
}

package parser;

import java.io.*;
import java.util.*;
import javax.swing.tree.*;
import project.Phase;
import project.Process;
import docs.*;

/**
 * #COD_007 #<Class Tracer>
 * ##PRJ_008
 *
 * <p>Store all documents.</p>
 * <p>Link features.</p>
 * <p>Link features in chain.</p>
 * @author Carlos Matias / Ricardo Winck
```

```
 */
public class Tracer {
	/**
	 * list of documents.
	 */
	private LinkedList documents;

	/**
	 * process
	 */
	private Process process;

	/**
	 * Mapped feature table
	 */
	private Table table;

	/**
	 * Builds a new Tracer
	 * @param process process to be traced
	 */
	public Tracer(Process process) {
		documents = new LinkedList();
		this.process = process;
	}

	/**
	 * Looks for the document that contains the specified feature.
	 * @param Feature feature - feature to be searched in the documents
list
	 */
	public Document findDocument(Feature feature) {
		ListIterator iterator = documents.listIterator();
		boolean isContained = false;
		Document current_doc = null;
		while (iterator.hasNext()) {
			current_doc = (Document) iterator.next();
			if (current_doc.contains(feature) == true) {
				isContained = true;
				break;
			}
		}
		if (isContained == false)
			current_doc = null;
		return current_doc;
	}

	/**
	 * Looks for a feature inside the documents list.
	 * @param Feature feature - feature to be searched in the documents
list
	 */
	public Feature findFeature(Feature feature) {
		ListIterator iterator = documents.listIterator();
```

```
            Document current_doc = null;
            Feature current_feature = null;
            while (iterator.hasNext()) {
                  current_doc = (Document) iterator.next();
                  if (current_doc.contains(feature) == true) {
                        current_feature =
current_doc.getFeature(feature);
                        break;
                  }
            }
            return current_feature;
      }

      /**
       * Links two features
       * @param mainFeature main feature
       * @param mappedFeature mapped feature
       */
      public void connect(Feature mainFeature, Feature mappedFeature)
            throws Exception {
            Feature feature1 = findFeature(mainFeature);
            Feature feature2 = findFeature(mappedFeature);
            if ((feature1 != null) && (feature2 != null)) {
                  feature1.addMapped(feature2);
                  feature2.addMapped(feature1);
            } else {
                  throw new Exception("Feature not found.");
            }
      }

      /**
       * Creates a new Document
       * @param name document name
       * @return Document
       */
      public Document createDocument(String name) {
            Document doc = new Document(name);
            documents.add(doc);
            return doc;
      }

      /**
       * Creates the feature traceability table
       * @param selectedDocument phase to be traced
       */
      public void buildTable(int selectedDocument) throws Exception {
            Document document = (Document)
documents.get(selectedDocument);
            if (document.hasFeatures() == false) {
                  String docName = document.getName();
                  throw new Exception(
                        "Document \""
                              + docName
```

```
                              + "\" does not contain features to be
traced.");
            }
            int numberOfPhases = process.getNumberOfPhases();
            Vector rows = document.toTableRows(numberOfPhases);
            table = new Table(rows, selectedDocument);
      }

      /**
       * Returns the feature string table
       * @param traceMode feature trace mode
       */
      public Vector getStringTable(int traceMode) throws Exception {
            Vector rows;
            switch (traceMode) {
                  case Table.ALL :
                        rows = table.getFullStringTable();
                        break;
                  case Table.TRACED :
                        rows = table.getTracedStringTable();
                        break;
                  case Table.UNTRACED :
                        rows = table.getUntracedStringTable();
                        break;
                  default :
                        throw new Exception("Invalid Trace Mode.");
            }
            return rows;
      }

      /**
       * Returns a table cell as a string
       * @param row table row
       * @param col table col
       * @return String
       */
      public String getTableItem(int row, int col) {
            Vector tableCol = (Vector) table.getCell(row, col);
            ListIterator colIterator = tableCol.listIterator();
            String id = null;
            String desc = null;
            StringBuffer buffer = new StringBuffer("Features
Description:\n");
            int size = tableCol.size();
            while (colIterator.hasNext()) {
                  Feature feature = (Feature) colIterator.next();
                  id = feature.getId();
                  desc = feature.getDescription();
                  if (colIterator.nextIndex() == size)
                        buffer.append("  " + id + " - " + desc);
                  else
                        buffer.append("  " + id + " - " + desc + "\n");
            }
            if (id == null)
```

```
                buffer.append("  There are no features!!!");
            return buffer.toString();

      }

      /**
       * Save the mapped features table into a html fle.
       * @param filename output html filename
       */
      public void toHTML(String filename) throws Exception {
            String width =
                  new String(String.valueOf(100 /
(process.getNumberOfPhases() + 1)));
            HTMLDoc html = new HTMLDoc();
            html.createHead("Traceability Matrix.");
            html.beginBody();

            html.beginTable();
            html.beginTableRow();
            html.createTableData("Features", width, "CCCCCC");

            ListIterator phaseIterator = process.getPhasesIterator();
            while (phaseIterator.hasNext()) {
                  Phase phase = (Phase) phaseIterator.next();
                  html.createTableData(phase.getName(), width, "CCCCCC");
            }

            Vector mappedRows = table.getMappedRows();

            ListIterator tableIterator = mappedRows.listIterator();
            while (tableIterator.hasNext()) {
                  Vector col = (Vector) tableIterator.next();
                  ListIterator colIterator = col.listIterator();
                  html.beginTableRow();
                  while (colIterator.hasNext()) {
                        String buffer = (String) colIterator.next();
                        if (buffer.equals("") == true)
                              buffer = "-";
                        html.createTableData(buffer, width, "FFFFFF");
                  }
                  html.endTableRow();
            }

            html.endTableRow();
            html.endTable();
            html.createParagraph(new Date().toString());
            html.endBody();
            html.createHTML();

            html.createFile(filename);
      }

      /**
       * Build interface JTree
```

```
	 */
	public void createTreeNodes(DefaultMutableTreeNode top) {
		Vector rows = table.getTracedFeatureRows();
		ListIterator tableIterator = rows.listIterator();
		while (tableIterator.hasNext()) {
			Vector newRow = new Vector();
			Vector col = (Vector) tableIterator.next();
			ListIterator colIterator = col.listIterator();
			DefaultMutableTreeNode newNode = null, oldNode = null;
			while (colIterator.hasNext()) {
				Vector features = (Vector) colIterator.next();
				ListIterator featIterator =
features.listIterator();
				oldNode = newNode;
				while (featIterator.hasNext()) {
					Feature feature = (Feature)
featIterator.next();
					String id = feature.getId();
					String desc = feature.getDescription();
					if (id.equals("") == true)
						id = "-";
					newNode = new DefaultMutableTreeNode(id + "
- " + desc);
					if (colIterator.nextIndex() == 1)
						top.add(newNode);
					else
						oldNode.add(newNode);
				}
			}
		}
	}

	/**
	 * Creates indirect connection between features.
	 */
	public void connectDocumentsInChain() {
		ListIterator documentsIt = documents.listIterator();

		while (documentsIt.hasNext()) {
			Document currentDocument = (Document)
documentsIt.next();
			currentDocument.connectFeaturesInChain();
		}
	}

	/**
	 * Sorts alphabetically documents features
	 */
	public void sortDocuments() {
		ListIterator documentsIt = documents.listIterator();

		while (documentsIt.hasNext()) {
			Document currentDocument = (Document)
documentsIt.next();
```

```
                currentDocument.sortFeatures();
            }
        }

        /**
         * Debug function
         */
        public void print() {
            ListIterator iterator = documents.listIterator();
            while (iterator.hasNext()) {
                Document doc = (Document) iterator.next();
                doc.print();
                System.out.println("");
            }
        }
}


package parser;

/**
 * #COD_005 #<Class Analizer>
 * ##PRJ_006
 *
 * <p>Parses and returns the tokens contained in the file.</p>
 * <p>Check whether there are more tokens.</p>
 * <p>Manage the input files of each phase.</p>
 * @author Carlos Matias / Ricardo Winck
 */

import java.util.*;
import java.util.regex.*;
import java.io.*;
import java.nio.*;
import java.nio.charset.*;
import java.nio.channels.*;

public class Analizer {
        /**
         * Pattern used to parse the input file according to the regular
expression
         */
        private Pattern pattern;

        /**
         * Matcher that contains the parsed tokens
         */
        private Matcher matcher;

        /**
         * Input files related to a single phase.
         */
        private Vector files;
```

```
	/**
	 * Input files iterator to a single phase.
	 */
	private ListIterator fileIterator;

	/**
	 * Builds a new analizer
	 * @param regularExpression - regular expression used to parse the
input files. e.g. "(#&lt;([^(#)]*)&gt;)"
	 */
	public Analizer(Grammar grammar) {
		pattern = Pattern.compile(grammar.toString(),
Pattern.DOTALL);
	}

	/**
	 * <p>Sets a new input filename.</p>
	 * <p>Initializes pattern and matcher.</p>
	 * @param filename - current filename.
	 */
	private void setInput(String filename) throws Exception {
		File f = new File(filename);
		FileInputStream fis = new FileInputStream(f);
		FileChannel fc = fis.getChannel();

		// Get a CharBuffer from the source file
		ByteBuffer bb =
			fc.map(FileChannel.MapMode.READ_ONLY, 0, (int)
fc.size());
		Charset cs = Charset.forName("8859_1");
		CharsetDecoder cd = cs.newDecoder();
		CharBuffer cb = cd.decode(bb);

		// Run some matches
		matcher = pattern.matcher(cb);
	}

	/**
	 * <p>Sets a new directory.</p>
	 * <p>Initializes the input buffer with the first file.</p>
	 * @param directory - directory that stores the phase files.
	 */
	public void setDirectory(File directory) throws Exception {
		files = extractFiles(directory);
		fileIterator = files.listIterator();
		if (fileIterator.hasNext()) {
			File nextFile = (File) fileIterator.next();
			setInput(nextFile.getPath());
		} else
			throw new Exception("Empty directory: " +
directory.getPath());
	}

	/**
```

```
	 * Checks whether there are any more tokens.
	 * @return boolean
	 */
	public boolean hasTokens() throws Exception {
		if (matcher.find() == true) {
			return true;
		} else {
			// Check whether there are more files
			while (fileIterator.hasNext() == true) {
				// Get next file
				File nextFile = (File) fileIterator.next();
				// Set the input buffer to be parsed
				setInput(nextFile.getPath());
				// Check whether there are more tokens
				if (matcher.find() == true)
					return true;
			}
			return false;
		}
	}

	/**
	 * Gets next token.
	 * @return String
	 */
	public String nextToken() {
		return matcher.group();
	}

	/**
	 * <p>Extracts all files contained in depth the specified
directory.</p>
	 * @param directory - directory that stores the phase files.
	 * @return Vector
	 */
	public Vector extractFiles(File directory) throws Exception {
		Vector list = new Vector();
		if (directory.isDirectory() == true) {
			File[] files = directory.listFiles();
			for (int i = 0; i < files.length; i++) {
				File currentFile = files[i];
				if (currentFile.isDirectory() == true)
					list.addAll(extractFiles(currentFile));
				else {
					list.add(currentFile);
				}
			}
		} else {
			list.add(directory);
		}
		return list;

	}
```

```
	/**
	 * Check whether token is a main feature identifier.
	 * @param token - main featur identifier
	 * @param id - phase id. e.g. REQ
	 * @return boolean
	 */
	public boolean isMain(String token, String id) {
		return !token.startsWith("##")
			&& !token.startsWith("#&lt;")
			&& (token.indexOf(id) != -1);
	}

	/**
	 * Check whether token is a mapped feature identifier.
	 * @param token - mapped feature identifier
	 * @return boolean
	 */
	public boolean isMapped(String token) {
		return token.startsWith("##");
	}

	/**
	 * Check whether token is a feature description.
	 * @param token - feature description
	 * @return boolean
	 */
	public boolean isDescription(String token) {
		return token.startsWith("#&lt;");
	}

	/**
	 * Check whether token is a feature.
	 * @param token - feature
	 * @return boolean
	 */
	public boolean isFeature(String token) {
		return !token.startsWith("#&lt;");
	}
}
```