PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Firmino dos Santos Filho
Brasil
firminosantos@totumcontinens.com.br

Abstract

Este artigo discute como aplicar um processo de desenvolvimento de software, baseado no processo iterativo, baseado nas melhores práticas de desenvolvimento de software, práticas comercialmente aprovadas.

Índice

1. INTRODUÇÃO

1.1. Objetivo

Estabelecer processo de desenvolvimento de software de produto baseado no processo iterativo, e que incorpore as melhores práticas de desenvolvimento de software, práticas comercialmente aprovadas, tanto a nível de métodos como procedimentos (definição do processo em termos de padrões de engenharia de software e métodos).

1.2. Escopo

Aplicável a todo desenvolvimento de software e firmware (assim denominado o software embarcado) de produto, incluindo anteprojeto, software de teste, análise de software existente, documentação de software desenvolvido por empresas contratadas e transferência de tecnologia de software, independentemente da extensão ou complexidade.

1.3. Definições e Abreviações

ESD	..	Especificação detalhada ou suplementar de software
DCC	..	Diagrama de classes conceitual
DCF	..	Diagrama de classes (modelo físico)
DSQ	..	Diagrama de seqüência descrevendo as iterações entre objetos.
DES	..	Diagrama de estados
DAS	..	Documento de Arquitetura, descrevendo a Arquitetura do sistema.
DIS	..	Documento de Implementação
CWT	..	Code Walk-Through
PTE	..	Plano de Teste
CTE	..	Casos e Procedimentos de Teste
RTR	..	Relatório de Teste com os resultados
PTH	..	Procedimento de teste de homologação
SCT	..	Scripts de Teste (Componentes de Software de Teste)
DHP	..	Documento de Histórico do Projeto

2. REFERÊNCIAS

[1] UML - Unified Modeling Language – linguagem padronizada pela OMG ("Object Management Group", consorcio sem fim lucrativos que produz e mantém especificações para a indústria de informática).

[2] Design and Code Inspections to Reduce Errors in Program Development, Fagan; 1976

[3] Gerenciar um projeto é realizá-lo através do uso de processos [PMI, 2000].

[4] The Unified Software Development Process – Jacobson, Booch, Rumbaugh.

[5] The Complete Guide to Software Testing – Bill Hetzel.

3. Conceitos - Melhores Práticas de Software

Um *processo de desenvolvimento de software* é uma série de atividades necessárias para transformar os requisitos do usuário em um sistema de software. Um *processo* efetivo provê diretrizes para o desenvolvimento eficiente de software com qualidade. Um processo bem definido permitirá resolver o paradoxo de software, maior qualidade com rápida disponibilização para o mercado. O processo adequado captura e apresenta as melhores práticas que o estado atual da arte permite:

3.1. Desenvolvimento Iterativo e Incremental

Desenvolvimento Iterativo é a técnica que é usada para entregar a funcionalidade de um sistema em uma série sucessiva de liberações de modo a realizar crescentemente o sistema desejado (também conhecido como *Elaboração progressiva*). Os riscos mais críticos devem ser focados nas fases iniciais do projeto de modo a aumentar previsibilidade a evitar descarte e retrabalho.

3.2. Gerenciamento de Requisitos (GR)

GR é definido como uma abordagem sistemática para extrair, organizar e documentar os requisitos do sistema e um processo que estabelece e mantém um acordo entre o cliente e os projetistas, de modo a controlar as mudanças de requisitos do sistema. O gerenciamento de requisitos envolve a tradução dos pedidos dos clientes em uma série de necessidades fundamentais dos clientes e características de sistema. Estes por sua vez são detalhados em especificações funcionais e não funcionais. Estes requisitos mais específicos são os requisitos de software [2]. Uma valiosa ferramenta pode ser usada para detalhar requisitos funcionais: *Use Case* (Casos de Uso). Um Caso de Uso descreve uma sucessão de ações, executada pelo sistema, que resulta em um valor para o usuário. *Casos de Uso* servem como uma representação UML para os requisitos do sistema. Quando é dito que um processo é dirigido a Casos de Uso *(Use-case driven)* significa que o processo de desenvolvimento segue um fluxo – segue uma série de fluxos de trabalho que derivam dos casos de uso. Casos de Uso são especificados, casos de uso são projetados, e no final casos de uso são a fonte a partir do qual o projetista de teste constrói os casos de teste (test cases).

3.3. Foco na Arquitetura

Extensas análises dos requisitos, do projeto, implementação e atividades de avaliação são executadas antes da implementação completa esteja em foco. O foco inicial do projeto de implementar e testar a arquitetura deve preceder o desenvolvimento em larga escala e teste de todos os componentes. Em sistemas de software de grande porte, a obtenção dos requisitos de qualidade não só depende de práticas a nível de código, mas também de toda a arquitetura do software. Assim, está entre os principais interesses do desenvolvedor, determinar, na ocasião em que a arquitetura de sistema de software for especificada, se o sistema terá as qualidades desejadas.

3.4. Modelagem Visual (UML) e Métodos Orientados a Objetos

A orientação a objetos prove conceitos, notações e métodos para a produção de um modelo de um sistema. Um *modelo* é uma abstração do sistema, especificando o sistema modelado a partir de um certo ponto de vista e a um determinado nível de abstração. Mostra as aspectos principais de um sistema em uma perspectiva particular e esconde os detalhes não essenciais. Pode-se destacar os seguintes benefícios na utilização de Modelos:

- Facilita o entendimento de sistemas complexos.
- Permite a exploração e comparação de alternativas de projeto a um baixo custo.
- Formação dos fundamentos da implementação.
- Captura precisa dos requisitos.
- A orientação a objetos simplifica o desenvolvimento de modelos tendo em vista que as notações orientadas a objeto podem criar modelos mais próximos do mundo real.

Modelamento Visual é o uso de notações gráficas e textual de modo a capturar o projeto de software. A notação UML permite-se elevar o nível de abstração, ao mesmo tempo que se mantém uma rigorosa sintaxe e semântica.

A UML (Unified Modeling Language) é a linguagem sucessora da análise orientada a objetos e métodos de projeto. UML é a terceira geração de linguagem de modelos visuais que define as construções e relações de sistemas complexos. A UML é uma linguagem padronizada para modelagem de software – uma linguagem que permite a visualização, especificação, construção e documentação dos artefatos de sistemas intensivos de software. A linguagem UML permite a comunicação clara e precisa entre

os diversos membros do time de desenvolvimento. A linguagem UML permite ao desenvolvedor a visualização do produto de seu trabalho em diagramas padronizados. Provendo anotações formalizadas para capturar e visualizar as abstrações de software, o principal impacto da tecnologia orientada a objetos está na redução do tamanho total do que precisa ser desenvolvido.

3.5. Trabalho de equipe

Um projeto de software bem sucedido precisa ter equilíbrio entre um sólido talento e pessoas altamente qualificadas nas posições chaves. Trabalho de equipe é muito mais importante que a soma dos indivíduos. Todas as pessoas em organizações de desenvolvimento de software têm que ter uma meta comum: entregar um produto de alta qualidade no prazo e dentro do orçamento previsto.

3.6. Ferramentas de apoio

As ferramentas e ambiente usados no processo de desenvolvimento de software têm um efeito linear na produtividade do processo. Durante cada ciclo de desenvolvimento de software as ferramentas provêem apoio de automatização crucial, de modo a permitir o desenvolvimento dos artefatos de engenharia de software através de *mudanças graduais e controladas*. Ferramentas e ambientes de desenvolvimento devem ser vistos como componentes primário para a automatização do processo e melhoria continua.

3.7. Verificação continua da qualidade

Existe uma propriedade famosa no desenvolvimento de software: é muito mais barato corrigir defeitos durante o desenvolvimento que os corrigir depois do desenvolvimento. Nos ciclos de vida iniciais as métricas devem ter uma forte contribuição para a avaliação da qualidade, quando esforços para melhorar a qualidade de software são muito mais efetivos. A coleta das métricas deve ser automatizada e não intrusiva, ou seja não deve interferir com as atividades dos desenvolvedores.
A avaliação do software deve ser uma coleta de tópicos não usuais (geralmente chamado de métricas de software), que possui uma ampla gama de modelos para predizer a qualidade do projeto de software. Deve-se utilizar as métricas e indicadores de software para medir o progresso e a qualidade de cada artefato produzido durante o processo de desenvolvimento de software.
Você não pode desenvolver um produto de software com qualidade ou melhorar o processo sem ter como medir (medir no sentido de avaliar sua qualidade e estágio de desenvolvimento) este software. As medições obtidas do software devem ser analisadas de modo a identificar pontos fracos do processo definido e prover uma forma de se obter uma melhoria do processo utilizado.
Métricas de software são usadas afim de monitorar e avaliar os seguintes aspectos do projeto:
" *Progresso* em termos de tamanho e complexidade.
" *Estabilidade* em termos de taxa de mudança na implementação, tamanho ou complexidade.
" *Modularidade* em termos da extensão da mudança.
" *Qualidade* em termos do número e tipo de erros.
" *Maturidade* em termos da freqüência de erros.
" *Recursos* em termos de recursos despendidos contra os planejados.

3.8. Gerenciamento da configuração de software (SCM, Software configuration management)

SCM é uma disciplina da engenharia de software que inclui as ferramentas e técnicas (processos ou metodologia) utilizadas pelas empresas para gerenciar as mudanças de seus ativos de software.
IEEE 828-1998 diz sobre SCM:
" *SCM constitui uma boa prática de engenharia para todos os projetos de software, independentemente da fase do desenvolvimento, ou se for protótipo rápido, ou manutenção em andamento. Aumenta a confiança e qualidade do software pelas seguintes razões:*

- *Prove estrutura para identificação e controle da documentação, código, interfaces, e bancos de dados para apoiar todas as fases do ciclo de vida.*
- *Apoiando uma metodologia escolhida para desenvolvimento / manutenção que se ajuste aos requisitos, normas, políticas, organização e filosofia de gerenciamento.*
- *Produz gerenciamento e informação de produto relativo ao estado da versão do produto de software utilizado como referencia, controle de mudanças, testes, liberações, auditorias, etc."*

3.9. Padrões para Codificação (Coding Standards)

A eliminação de todos os erros está, pelo menos em ambientes industriais atuais, bem além da capacidade tecnológica de software disponível. Não obstante a experiência sugere que a densidade de erros de códigos liberados possam ser reduzidos pela metade através da utilização de processos internos de verificação apoiados por ferramentas analíticas adequadas. Isto não requer mudanças de paradigmas ou idiomas. É necessário a determinação na adoção de técnicas comprovadas. De todas as formas de controle de qualidade de software, a inspeção de código é sem dúvida a mais efetiva. Quando apoiado por boas ferramentas, o custo unitário típico de identificação de um erro estático é até duas vezes menor que o custo da identificação através de métodos dinâmicos. Para realizar o trabalho de inspeção é necessário um padrão de codificação (coding standard) definindo requisitos que o código a ser inspecionado deve satisfazer. Um padrão de codificação são regras que governam o uso de uma linguagem de programação. Complementa o padrão da linguagem definindo características de uso aceitável e o inaceitável. Características de uso inaceitável conduzem ao erro ou a má interpretação. Características de uso aceitável evitam situações dúbias ou problemáticas. Isto não garante que o código esteja livre de erros, porque sempre se pode alcançar uma implementação imaculada da coisa errada. Um padrão de codificação ajuda a que se:

- evite uso de características indefinidas,
- evite uso de características não especificadas,
- evite uso de características definidas pela implementação,
- se proteja contra erros de compilação,
- se proteja contra erros comuns de programação,
- limite a complexidade do programa,
- estabeleça uma base objetiva para revisão de seu código.

Evitando usos indefinidos, não especificados e definidos pela implementação, efetivamente você limita o uso da linguagem de programação a um subconjunto que é inequivocamente definido. Isto evita os pontos fracos linguagem. Este processo não é a prova de falhas mas o ajudará a prevenir erros. Se você examinar relatórios de erros gerados pelo compilador, não só o seu próprio, mas também de outros desenvolvedores, para a mesma linguagem, você obterá uma idéia razoável dos tipos de coisas que podem ser mal implementadas. Você pode evitar essas características da linguagem ou pelo menos sujeitar seu o uso a uma clara e razoável justificação.

3.10. Teste Automatizado

"Teste Automatizado" está automatizando o processo manual de teste ainda hoje em uso. O propósito real das ferramentas de teste automatizadas são automatizar os testes de regressão. Isto significa que você tem que ter ou tem que desenvolver casos de teste detalhados que sejam reproduzíveis, e este conjunto de testes sejam executados toda vez que haja uma mudança no programa de modo a assegurar que esta mudança não produza conseqüências não intencionais.
Automatizar um teste significa termos uma ferramenta que reproduz todos os estados possíveis da interface de uma função ou objeto e ao mesmo tempo avalia a resposta recebida ou o estado de saída. Deste modo um teste de um objeto manualmente que

demandaria dias para percorrer todos os seus estados, com o uso de um processo automático pode ser implementado em minutos e pode ser repetido todas as vezes que houver uma alteração de software.
No processo de automatização do teste pode-se seguir dois caminhos alternativos: comprar uma ferramenta disponível no mercado ou se desenvolver pequenos componentes de software para esta função.

As melhores práticas são um conjunto de técnicas aprovadas comercialmente para o desenvolvimento de software que, quando usadas de modo combinado, golpeiem as causas básicas dos problemas de desenvolvimento de software.

4. DETALHAMENTO DO PROCESSO

4.1. Ciclo de desenvolvimento de Software

O processo de desenvolvimento de software utiliza um ciclo de desenvolvimento de software *iterativo e incremental* (série sucessiva de liberações ou série sucessiva de iterações especificadas no planejamento do projeto). Deverá existir um planejamento para cada iteração e este planejamento contém os requisitos do sistema que deverão ser implementados na fase respectiva. Os requisitos de maior risco deverão ser implementados nas primeiras iterações.
A cada final de iteração (fase) está associado a obtenção de uma revisão interna de software.

Cada *iteração* (ou fase) é composta das seguintes etapas:

- Análise de Requisitos
  Na 1ª iteração, elaboração da especificação detalhada baseada na Especificação do Produto, nas seguintes, refinamento dos requisitos.
- Análise Orientada a Objetos (UML)
  A análise visa transformar as informações obtidas no levantamento de requisitos em um modelo através de diagramas de caso de uso e diagramas de classe. Os diagramas resultantes da fase de análise é também conhecido como Modelo Conceitual. Como regra geral se o cliente não entende um determinado conceito, então provavelmente este não é um conceito.
- Projeto
  Nesta fase se define a arquitetura do sistema de software.
  O projeto do sistema é executado através do refinamento dos diagramas criados na fase de análise ou até mesmo através da construção de pseudocódigo.
- Implementação
  Refere-se à codificação e à integração de todas as funcionalidades especificadas. Os componentes e objetos são codificados através de uma linguagem de programação. Baseado no modelo visual (UML) o desenvolvedor codifica o modelo. No processo de codificação o desenvolvedor deverá adicionar comentários ao código em quantidade suficiente para a compreensão do mesmo, possibilitando a criação da documentação do software durante o processo de desenvolvimento e não ao final.
- Teste
  Através da aplicação das melhores práticas de software procura-se garantir a qualidade do processo de desenvolvimento de software e por conseqüência do produto final.
  Apesar dos métodos, ferramentas e procedimentos utilizadas, falhas no software ainda podem ocorrer. Assim a etapa de teste é importante para a identificação e eliminação de falhas.

Estas etapas são executadas em todas as fases, mas com diferente ênfase:

Nas fases iniciais dá-se ênfase na análise dos requisitos e conceitos principais do projeto, de modo a se obter uma clara e precisa noção do problema. Nas fases

subsequentes deve-se dar ênfase no detalhamento do projeto e no grosso da implementação.

Durante cada fase do ciclo de desenvolvimento de software são gerados diversos registros, denominados artefatos. Os artefatos gerados estão descritos nos itens de descrição de cada fase do desenvolvimento.

O modelo incremental tem como premissa a resolução dos riscos porque é centrado na arquitetura, incremental e orientado a casos de uso.

Nota: *No meio e no final de cada fase deverá haver uma revisão de modo similar a revisão de código conforme descrito no item 4.1.4.1.*

4.1.1. Análise de Requisitos

Esta atividade consiste em obter um claro entendimento da especificação de software. Nesta fase do processo são efetuadas análises da especificação de modo a se garantir que seja obtido um entendimento correto da especificação seja alcançado. Efetua-se uma análise critica dos requisitos e se são factíveis. Como resultado desta atividade uma Especificação detalhada ou suplementar de software será elaborada com uma definição do escopo do projeto. Cada item da especificação será avaliado segundo os critérios definidos no item 4.1.1.1.

4.1.1.1. Critérios de avaliação e aceitação dos requisitos

- Definição clara e propriamente definido.
- Completo
- Consistência com outros itens
- Identificação única.
- Possível de ser implementado
- Requisito possível de ser verificado e testado.

4.1.1.2. Artefatos

O relatório produzido nesta fase do desenvolvimento deve conter os seguintes itens:

- Especificação detalhada ou suplementar de software baseada na Especificação de Produto, incluindo requisitos funcionais e não funcionais; seções e subseções conforme Anexo A.
- Diagramas e textos de Casos de Uso;
- Diagramas de Seqüência, caso seja necessário para complemento da Especificação.
- Diagramas de Atividades, como complementação dos casos de uso.

4.1.1.2.1. Referencia cruzada

A especificação detalhada deverá conter uma tabela que correlacione os itens da Especificação do Produto com a especificação detalhada de software, bem como uma correlação dos itens de especificação com os módulos e modelo de software implementado (sendo esta ultima correlação criada e mantida ao longo do desenvolvimento).

4.1.2. Análise Orientada a Objetos (UML)

A metodologia de projeto orientada a objeto produz uma arquitetura de software baseada em objetos manipulados pelo sistema ou sub-sistema. Um projeto orientado a objetos tenta criar um modelo aproximado da realidade visto que este representa os objetos do mundo real e operações sobre estes ou suas ações externas.

Um objeto é uma entidade que possui as seguintes características:

- Estado que representa seu valor corrente
- Uma descrição da ação recebida ou requerida de outros objetos
- Um nome que pode ser referenciado
- Visibilidade controlada para e dos outros objetos

- A habilidade de ser visto de dois modos:
  - Vista externa ou comportamental
  - Vista interna ou de implementação

No projeto orientado a objeto a estrutura resultante tende a ser definida em camadas de abstração onde cada camada representa coleções de objetos com visibilidade limitada para outros objetos.

4.1.2.1. Analise: Definição da Estrutura dos Objetos

Uma vez que o comportamento externo do sistema está definido, o analista deve identificar os objetos chave, classes e seus relacionamentos dentro do sistema (este procedimento também é conhecido como extração dos conceitos do sistema a partir dos requisitos).

Passos básicos na Análise Orientada a Objetos:

- Identificar os Objetos
- Identificar a Associação dos Objetos
- Agrupar os Objetos em Classes
- Identificar e Classificar os relacionamentos das Classes
- Identificar o comportamento dos Objetos e Classes
  - Tipos de comportamento: simples, por estado e continuo.
  - Diagrama de estados
  - Diagrama de seqüência.
  - Diagrama de colaboração.
- Agrupar Classes em Domínios
- Validar Classes e Objetos

4.1.2.2. Artefatos

O Artefato produzido nesta fase do desenvolvimento deve conter os seguintes itens:

- Diagrama de classes (conceitual) que realizam casos de uso, contendo a das classes de controle, interface e as classes que contém dados permanentes (não voláteis). As classes relacionadas a detalhes da implementação não são modeladas nesta etapa.
- Diagrama de colaboração entre os objetos identificados.

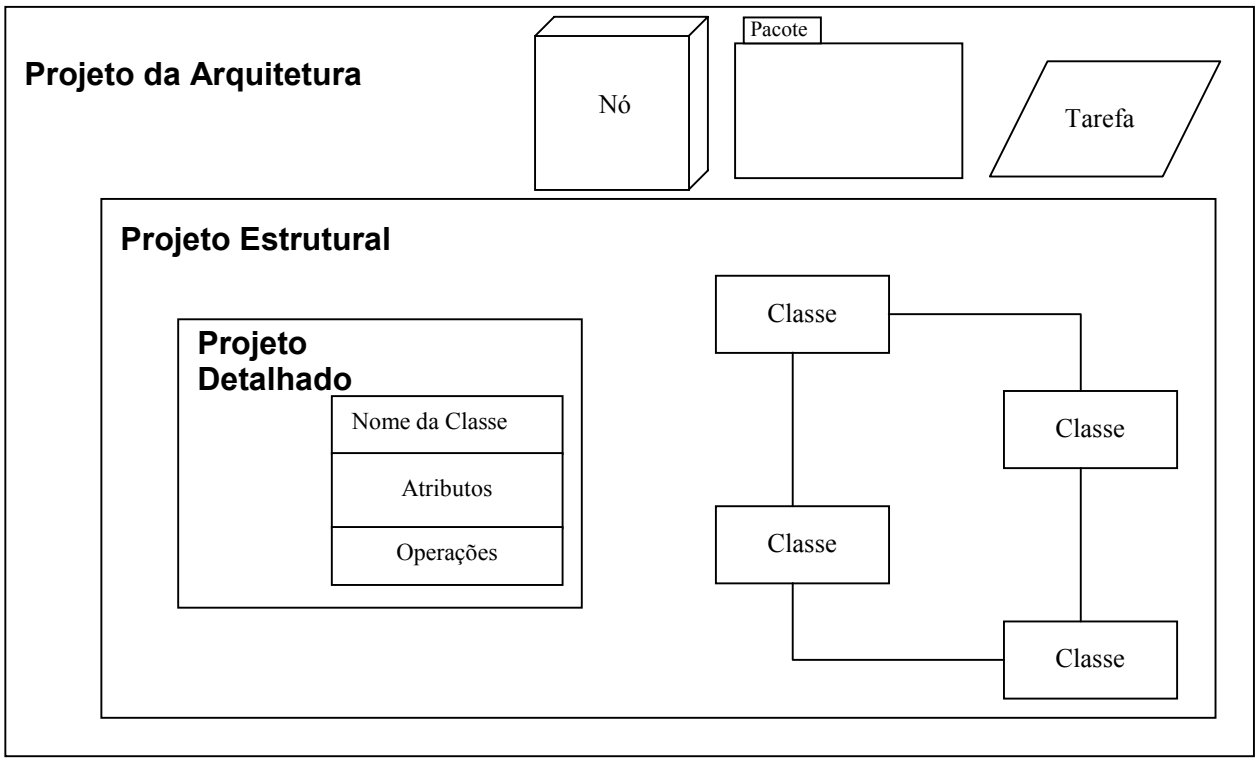
Nota: *O projeto orientado a objetos independe da linguagem de programação, sendo valido também para C e Assembler.*

4.1.3. Projeto do Software

O Projeto define uma solução particular baseada no modelo de análise. O projeto pode ser subdividido em três partes: *projeto da arquitetura do software, projeto estrutural e projeto detalhado*.

Fase do Projeto de SW	Escopo	O que é definido
Projeto da Arquitetura	• Sistema • Processador	• Numero e tipo de processadores; • Pacotes de objetos sendo executados em cada processador; • Meio de comunicação entre processadores e protocolos. • Modelo de concorrência e estratégia de comunicação entre "threads"; • Níveis (ou camadas) de software; • Política global de tratamento de erros.
Projeto Estrutural	• Objetos (inter-relacionamento)	• Instancias de padrões de projeto de múltiplos objetos colaborando entre si. • Containers e classes e objetos definidas nesta fase do projeto.
Projeto Detalhado	• Objetos (internamente)	• Definição do algoritmo do objeto. • Estrutura de dados; • Funções membro.

4.1.3.1. Projeto da Arquitetura



O modelo de análise identifica objetos, classes e relacionamentos mas não define como estes são organizados em uma estrutura em larga escala. O projeto da arquitetura identifica as estratégias chaves para a organização em larga escala do sistema em desenvolvimento. Estas estratégias incluem o mapeamento dos pacotes de software aos processadores, tipos de comunicação e seleção do protocolo, modelo de concorrência e definição dos “threads” (tarefas).
As estratégias do projeto podem ser baseadas em arquiteturas padrões utilizadas ou definidas em sistemas similares. O projeto da arquitetura do software consiste no processo de projeto dos subsistemas, pacotes, tarefas e suas interconexões.

Os subsistemas são constituídos de pacotes e tarefas. Pacotes podem conter sub-pacotes, porém seus componentes primários são objetos e classes que implementam conceitos importantes dentre de um dado domínio. A generalização de classes estão sempre dentro de um único pacote.
Os subsistemas são freqüentemente organizados em níveis ou camadas, tais como:

- Aplicação
- Interface do usuário
- Comunicação
- Sistema Operacional
- Abstração do Hardware.

4.1.3.2. Projeto Estrutural

O projeto estrutural é organizado primariamente na descoberta e uso de padrões de colaboração dos objetos. O projeto estrutural preocupa-se com a adição e organização das classes de modo a suportar uma estratégia particular de implementação. No processo de definição do projeto estrutural o modelo obtido na fase de análise é adicionado de objetos facilitadores de modo a implementar um projeto em particular. Um conjunto de classes e objetos colaborando entre si é denominado *mecanismo*. Um sistema de tempo real pode ter centenas de mecanismos operando concorrentemente.

Mecanismos que podem ser utilizados nesta fase:

- *Observador:* uma única fonte de informação age como servidor para múltiplos clientes, que devem ser atualizados autonomamente quando o valor dos dados são alterados.
- *Transação*: utilizado quando comunicações de alta confiabilidade é necessária em meios não 100% confiáveis.
- *Container:* gerencia uma coleção e prove acesso a operações.
- *Interfaceamento:* separação explicita entre interface e implementação.
- *"Rendezvous":* sincronização de tarefas concorrentes.
- *Padrão baseado no comportamento de estado:* implementação por maquina de estados.

A atividade de análise resulta em classes de análise que representam elementos conceituais que representam o comportamento do sistema. Na fase de projeto, as classes de análise evoluem para diversos tipos de elementos de projeto:

- *Classes,* que representam funcionalidades em níveis mais detalhados.
- *Subsistemas,* que representam funcionalidades em nível macro.
- *Classes ativas,* que representam "threads" de controle do sistema, permitindo o modelamento de concorrência.
- *Interfaces*

❖ Identificação de Formas de Comunicação

Adicionalmente, na fase de projeto devemos identificar:

- Eventos, que especificam ocorrências de interesse no tempo e espaço e requerem resposta do sistema.
- Sinais, que representam mecanismos assíncronos utilizados para a comunicação de certos tipos de eventos dentro do sistema.

❖ Identificação de Concorrência

Os requisitos de concorrência do sistema devem ser considerados nesta fase. Como resultado, devemos identificar as classes ativas que representam as "threads" de controle do sistema.

Como exemplo, podemos dizer que "threads" de controle podem ser necessárias nas seguintes condições:

- Objetivos distintos entre diferentes áreas do software.
- Melhor utilização da CPU quando da existência de atividades suspensas.
- Priorização de atividades.
- Suporte adequado a subsistemas.

❖ Identificação de Mecanismos de Comunicação entre Processos.

Temos os seguintes típicos mecanismos de comunicação:

- Memória Compartilhada, com ou sem semáforos para garantir a sincronização.
- Rendezvous, especialmente quando diretamente suportado pela linguagem.
- Semáforos, utilizados para bloquear acesso simultâneo a recursos compartilhados.
- Passagem de mensagens, ponto a ponto ou multiponto.
- "Mailboxes"

4.1.3.3. Projeto Detalhado

Nesta fase os objetos são detalhados e define-se o projeto da sua estrutura interna. Nesta fase define detalhes tais como forma de armazenamento utilizada para os atributos (*estrutura de dados*); como as associações são implementadas; *operações* que serão providas pelo objetos; seleção dos *algoritmos* internos e a definição do tratamento de *exceção* interna ao objeto.

Mecanismos que podem ser utilizados nesta fase:

- *Persistência:* objetos responsáveis por dados permanentes (database) ou estados permanentes.
- *Concorrência:* forma de concorrência dos objetos.
- *Segurança:* definição de formas de controle de acesso aos componentes do sistema.
- *Tratamento de Erros:* conceitos e critérios a serem utilizados nos tratamentos de erros. Os erros devem ser identificados nas camadas de baixo nível, para seu tratamento deve ser reservado para as camadas de alto nível.

4.1.3.4. Artefatos

Os seguintes artefatos são produzidos nesta fase do desenvolvimento:
- Diagrama de classes (modelo físico);
- Diagrama de seqüência descrevendo as iterações entre objetos;
- Diagrama de estados;
- Documento de Arquitetura, descrevendo a Arquitetura do sistema.

4.1.4. Implementação

Os seguintes artefatos são produzidos nesta fase do desenvolvimento:
- Diagrma dos subsistemas implementados e seus dependentes, interfaces e conteúdo.
- Componentes e objetos codificados através de uma linguagem de programação.
- Componentes executáveis;

4.1.4.1. "Code Walk-through"

4.1.4.1.1. Conceitos

O conceito de inspeção de código foi introduzido por M. Fagan como descrito em [2]. Este é um dos métodos que são eficientes na detecção de defeitos e melhoria da qualidade. A combinação da técnica de inspeção de código com a técnica de "code walkthrough" permite ampliar a eficiência enquanto reforça a transferencia de "know-how" dentro do projeto.
Não se deve revisar todo o código mas apenas uma seleção baseada na importância e complexidade das partes e na falta de experiência do desenvolvedor. Pelo menos 10% de todo o código deverá ser revisado.
Uma das formas de promover a transferencia de "know-how" entre os subprojetos é a indicação de um desenvolvedor de outro subprojeto para participar da inspeção.

4.1.4.1.2. Objetivos

- Reforçar a pratica das melhores técnicas de software de modo a obtenção de um desenvolvimento de software eficiente;
- Suportar a transferencia de "know-how" entre membros do projeto;
- Impor estratégias de programação comum dentro do projeto;
- Verificação da maturidade do projeto de software e sua implementação.

4.1.4.1.3. Procedimento

Deverão participar da revisão:
- Autor
- 1 a 2 revisores (também desenvolvedores)
- O moderador, que além da função de revisão deverá ponderar a aplicação das mudanças;

Material:

- Modelo Visual e código fonte.
- Apenas um componente contendo de 10 a 20 classes deverá ser revisado por vez.

Atividades antes da revisão:

- Gerente do projeto indica os desenvolvedores (revisores e moderador) que deverão participar do “walkthrough” .
- O autor prove aos participantes o material a ser revisado, modelo e código fonte, e indica os principais pontos do foco da revisão.
- Os participantes devem analisar o material previamente e criar uma lista de pontos a serem discutidos durante o “walkthrough”.

Atividades da revisão:

- O autor deverá expor para os revisores as partes de interesse do projeto, enquanto os participantes apresentaram suas questões criticas.
- O “code walkthrough” deverá iniciar pelas partes criticas do projeto. A inspeção deverá iniciar pelas interfaces.
- O autor registra os problemas achados e as decisões do grupo quanto as alterações a serem implementadas em relatório técnico (WTR).
- A revisão deverá ter um tempo máximo de duração de 2horas ± 30 minutos.

4.1.5. Projeto de Teste do Software

Nesta fase define-se um plano de teste com a definição dos requisitos de teste e as estratégias a serem utilizadas, bem como a definição dos casos de teste (a serem derivados dos casos de uso definidos na etapa de especificação). O projeto de teste objetiva mensurar e gerenciar as ações que serão executadas durante toda a fase de teste.

Os propósitos do projeto de teste são:

- Identificar e descrever os casos de teste;
- Identificar e estruturar os procedimentos de teste, especificando como executar os casos de teste.

Os Artefatos produzidos nesta fase do desenvolvimento são:

- Plano de Teste (PTE).
- Casos e Procedimentos de Teste (CTE).
- Relatório de Teste com os resultados. (RTR).
- Procedimento de teste de homologação. (PTH).
- Scripts de Teste (Componentes de Software de Teste) (SCT).

4.1.5.1. Tipos de Teste

4.1.5.1.1. Teste de Objetos

O comportamento de um objeto é dado pelo valor de seus atributos e este valor pode ser alterado pela seqüência de mensagens e resposta que envia ou recebe.
Durante o projeto destes testes, deve-se identificar todos os estados do objeto (ou valor de seus atributos) e as mensagens (ou estímulos) que conduzem o objeto a estes estados. O projeto do teste também deve incluir o teste de todas as mensagens a serem enviadas ou recebidas.
Nos casos de projetos que utilizem linguagem não orientada a objeto, este teste será substituído pelo teste de “procedures”.

4.1.5.1.2. Teste de Integração

O teste de integração verifica se os objetos testados de forma individual continuam se comportando corretamente quando integrados. A integração dos objetos deve ser realizada de forma incremental.

4.1.5.1.3. Teste Funcional

O teste funcional é focado nos requisitos funcionais do software. Os casos de testes funcionais são derivados dos casos de uso definidos durante a fase de levantamento dos requisitos. Durante o projeto destes testes, deve-se identificar todos os estados e estímulos de cada função e criar casos de teste para cada situação identificada.

4.1.5.1.4. Teste de Regressão

Antes de finalizar uma iteração é necessário garantir que nenhum componente desenvolvido em fases anteriores foi danificado e que continua operando corretamente, através dos testes de regressão. Alguns casos de teste das fases anteriores podem ser reaproveitados com alterações adequadas para implementação dos testes de regressão.

4.1.5.1.5. Teste de Validação

Este testes visa garantir que as características funcionais e de desempenho estão em conformidade com as especificações.

4.1.5.1.6. Teste de Tolerância a Falhas

O teste de tolerância a falhas é um teste de sistema que força o software a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada. Um Sistema tolerante a falhas pode ou não ser parte da especificação do sistema.

4.1.5.1.7. Teste de Segurança

O teste de segurança tem por objetivo garantir que o sistema se comporte adequadamente mediante tentativas ilegais de acesso. O teste de segurança tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão de acesso indevidos.

4.1.6. Verificação da qualidade

A cada final de iteração (fase) deverá também estar associado uma avaliação da versão gerada confrontada com a respectiva especificação.
As medições obtidas do software (métricas) devem ser analisadas de modo a identificar possíveis pontos fracos da versão obtida.
As Métricas de software devem ser utilizadas afim de monitorar e avaliar os seguintes aspectos do projeto:
" *Progresso* em termos de tamanho e complexidade.
" *Estabilidade* em termos de taxa de mudança na implementação, tamanho ou complexidade.
" *Modularidade* em termos da extensão da mudança.
" *Qualidade* em termos do número e tipo de erros.
" *Maturidade* em termos da freqüência de erros.
" *Recursos* em termos de recursos despendidos contra os planejados.

Esta avaliação será utilizada para se efetuar eventuais correções no projeto, prazo ou recursos necessários, além de subsidiar estimativas de prazo, custo e recursos necessários a eventuais mudanças de especificação.

4.1.6.1. Métricas Principais

SETE METRICAS PRINCIPAIS			
	Métrica	Objetivo	Perspectivas
Indicadores Gerenciais	Trabalho e Progresso	Iteração Planejada Planejado vs. Real	Nº de Classes, UCs, SLOC, Casos de Teste, Nº de Alterações.
	Custo Orçado e gastos	Visão Financeira, Planejado vs Real	Custo mensal, Horas Gastas mês, Porcentagem do orçamento gasto.
	Dinâmica de Recursos Humanos	Recursos Planejados vs Real	Pessoas adicionadas ou retiradas do Projeto mensalmente.
Indicadores de Qualidade	*Fluxo de Mudanças* (número de SAs abertas e fechadas durante o ciclo de vida) e *Estabilidade* (relação entre SAs abertas e SAs implementadas).	Indicativo de convergência de cronograma.	Solic.Alteração (SA) abertas vs fechada, por tipo (0,1,2..), versão, componente, subsistema.
	Quebra (extensão média de mudança, dimensão do software que necessitou retrabalho) e *Modularidade* (Quebra média através do tempo).	Convergência, Retrabalho Software	Nº de SLOC por mudança, por tipo (0,1..), por versão, componente, subsistema.
	Retrabalho (custo médio da mudança, que é composto dos recursos para analisar, resolver e retestar a mudança em relação a versão anterior) e *Adaptabilidade* (definida como o Retrabalho através do tempo).	Convergência, Retrabalho Software	Nº de horas média gasto por mudança, (0,1..), por versão, componente ou subsistema.
	MTBF (é o tempo médio de uso do software entre falhas)e *maturidade* (definido como o MTBF através do tempo).	Cobertura do Teste Adequação e robustez ao uso.	Contagem das falhas, Nº de horas de teste até a falha, por versão, componente ou subsistema.

A utilização destas métricas não excluem a necessidade ou a viabilidade de utilização de métricas adicionais convenientes ou necessárias ao processo.
SLOC significa número de linhas de código. SA solicitação de alteração

4.1.6.2. Forma de Avaliação dos Artefatos

Inspeções, Revisões e "Walkthroughs" são técnicas focadas na avaliação dos artefatos e são metodos eficazes de melhoria da qualidade e produtividade do processo de desenvolvimento. Devem ser conduzidas em reuniões formais, com um atuando como facilitador e outro como relator. A cada final de etapa, os artefatos produzidos serão avaliados por uma das três técnicas.

Revisão
Reunião de revisão formal de um determinado artefato (ou grupo de artefatos) apresentado ao cliente (interno ou externo), para comentários e aprovação.

Inspeção

Técnica de avaliação formal no qual artefatos são examinados em detalhe por um ou mais desenvolvedores (que não sejam os autores), com a intenção de identificar erros, violações dos padrões de desenvolvimento ou outros problemas.

Walkthrough

Processo de revisão no qual o desenvolvedor apresenta o artefato do qual foi autor enquanto os demais membros apresentam questões e fazem comentários sobre a técnica utilizada, estilo, possíveis erros, violações de padrões de desenvolvimento e outros problemas.

4.2. Sistema de Controle de Versão.

Através da utilização de uma ferramenta (de software) para o controle de versão, e a utilização de um repositório centralizado para armazenar e gerenciar o acesso a todos os arquivos produzidos durante o desenvolvimento, obtêm-se os seguintes resultados:

Controle de Versão: múltiplas revisões do mesmo arquivo podem ser salvas, e antigas revisões estarão sempre disponíveis, de modo controlado.

Desenvolvimento concorrente: múltiplos desenvolvedores podem editar sua cópia própria do mesmo arquivo, prevenindo colisões entre desenvolvedores, prevenindo múltiplas edições do mesmo arquivo ao mesmo tempo.

Gerenciamento de versão liberada: permite rastrear qual arquivo faz parte de qual revisão liberada.

Rastreamento: rastreamento automático das alterações dos arquivos fontes ao longo do curso de desenvolvimento.

Auditoria: é possível controlar quem modificou qual arquivo, quando ocorreram as alterações e quais foram as alterações.

Armazenamento: redução dos requisitos de armazenamento dos dados.

Um sistema de gerenciamento de configuração de software é requisito da ISO/IEC 15504 e CMMI.

4.2.1. Identificação das Versões

Inicialmente devemos subdividir o conceito de versões em “Releases” (liberações de versões para o cliente) e revisões internas (não liberadas para cliente) utilizadas para controle dos ciclos de teste.

4.2.1.1. “Releases”

As liberações para o cliente terão numeração **“R.rr”**. . A primeira liberação para o cliente será o release 1.00.
Revisões que alterem a operação do sistema ou corrijam problemas críticos serão identificados pela alteração no digito mais significativo (R) da revisão; Revisões que corrijam problemas não críticos serão identificados pela alteração no digito menos significativo (rr, de 00 a 99).

4.2.1.2. Revisões de uso interno

As revisões internas terão numeração complementar ao último release (ou último release em que foi baseado a revisão interna), ou seja “**R.rr.lll**”. Se o último release for a versão 1.05 (por exemplo), as revisões internas serão 1.05.001, 1.05.002 e assim sucessivamente. A cada novo release, a numeração da revisão interna reinicia de 001.

No inicio do projeto quando ainda não houver liberações para o cliente, inicia-se da versão 0.00.001. Durante o desenvolvimento deverá ser registrado pelo menos uma revisão interna por semana.

4.2.1.3. Controle dos arquivos associados aos “Releases”

Cada arquivo de software associado ao “Release” deverá estar clara e inequivocamente relacionado a revisão liberada, de modo a possibilitar a reconstituição do arquivo executável a partir dos arquivos fontes e arquivos de projeto.

4.2.1.4. Tarja de Identificação dos arquivos

Cada arquivo versionado (liberado ou com revisão interna) receberá uma tarja. A seguinte regra será utilizada para a criação desta tarja:

NomedoSistema_Componente_RR_rr_III

A utilização da tarja nos arquivos do projeto de software permitirá relacionar inequivocamente cada arquivo ao “Release” ou versão interna.

4.2.2. Histórico do Projeto

Durante todo desenvolvimento do projeto deverá ser criado e mantido um histórico de criação e modificações do projeto, contendo:

```
/* Copyright (c) 2004 FSF Company
 *                    Todos os Direitos reservados.
 *
 * Autor(es):
 * Data de Criação:
 * Histórico de Modificações:
 * 01-Jan-2003, Autor, Índice da alteração, Artefato, Descrição da
alteração.
 * 01-Fev-2003, Autor, Índice da alteração, Artefato, Descrição da
alteração.
 */
```

onde artefato pode ser qualquer documento, diagrama ou arquivo de código fonte.
Deverá ser criado um documento de histórico para cada subsistema, componente ou aplicativo.

4.3. Padrão de codificação

Durante o processo de codificação (criação do código fonte), deve-se seguir um padrão de codificação de acordo com a linguagem de programação utilizada.

ANEXO A: Seções da Especificação do Produto do subsistema de software

1. Objetivo

1.1. Convenções

1.2. Escopo

2. Documentos Aplicáveis

2.1. Referências

2.2. Glossário

3. Utilização do Produto

3.1. Perspectiva do Produto

3.2. Funções do Produto

3.3. Componentes do Produto

3.4. Ambiente de Operação

3.4.1. Sistema Operacional

3.5. Restrições do Projeto e de Implementação

3.6. Dependências

4. Requisitos

4.1. Requisitos da interface externa

4.1.1. Interface do Usuário

4.1.2. Interface com dispositivos

4.1.3. Interfaces de Hardware

4.1.4. Interfaces de Software

4.1.5. Interfaces de Comunicação

4.2. Características do Sistema.

4.2.1. Diagrama de Casos de Uso.

4.2.2. Descrição dos Casos de Uso.

4.2.3. Cenários: Diagrama de Seqüência, Estado e Atividades.

4.3. Requisitos não funcionais

4.3.1. Requisitos de Desempenho

4.3.2. Requisitos de segurança do usuário

4.3.3. Requisitos de segurança de acesso ao sistema

4.3.4. Atributos de qualidade do software

4.3.5. Documentação do usuário

4.3.6. Requisitos de uso

4.3.7. Requisitos de confiabilidade

4.3.8. Requisitos de manutenção

4.3.9. Requisitos de escala do sistema

4.4. Outros Requisitos

5. Métodos de Ensaios

6. Outros