MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14616-TDI/1195

MELHORIA DO SOFTWARE EMBARCADO EM SATÉLITES DO INPE: PROPOSTA PARA UM PASSO A MAIS

Primavera Botelho de Souza

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Tatuo Nakanishi e João Bosco Schumann Cunha, aprovada em 18 de outubro de 2002.

INPE
São José dos Campos
2007

681.3.06:629.783

Souza, Primavera Botelho de.
Melhoria do software embarcado em satélites do INPE: proposta para um passo a mais / Primavera Botelho de Souza. – São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2002.
138 p.; (INPE-14616-TDI/1195)

1. Software. 2. Satélites. 3. Sistema de tempo real. 4. Controle de qualidade. 5. Desenvolvimento de software. I. Título.

Aprovada pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de **Mestre** em **Computação Aplicada.**

Dr. Solon Venâncio de Carvalho
Presidente/INPE/SJCampos-SP

Dr. Tatuo Nakanishi
Orientador/UMC/M.Cruzes-SP

Dr. João Bosco Schumann Cunha
Orientador/UNIFEI/Itajubá-MG

Dr. Maurício Gonçalves Vieira Ferreira
Membro da Banca/INPE/SJCampos-SP

Drª Selma Shin Shimizu Melnikoff
Membro da Banca
Convidada POLI/USP/S. Paulo-SP

Candidata: Primavera Botelho de Souza

São José dos Campos, 18 de outubro de 2002.

Ao ***Alex***, minhas filhas *"**Luiza e Helena**",*

*meu **pai**, minha **mãe**,* pelos

momentos que deixei de estar presente.

AGRADECIMENTOS

A Deus, que ilumina o meu caminho.

Ao orientador Dr. Tatuo Nakanishi pela incansável dedicação.

Ao orientador Dr. João Bosco Schumann Cunha pela significativa colaboração durante a elaboração do trabalho.

Ao José Damião Duarte Alonso e Ronaldo Arias pelo fornecimento de documentos e informações essenciais.

Ao Dr. Hisao Takahashi e Dr. Delano Gobbi pela confiança depositada que permitiu o desenvolvimento deste trabalho.

Aos amigos das Linhas de Pesquisa LUME e FISAT pelo imenso companheirismo.

Aos amigos do curso de Mestrado sempre prestativos.

A todos os amigos que, direta ou indiretamente contribuíram para o enriquecimento deste trabalho.

RESUMO

O desenvolvimento de softwares embarcados em satélite concebidos para desempenhar funções críticas nos satélites construídos pelo INPE não tem contado com o envolvimento de especialistas em desenvolvimento de software para conduzir o processo de melhoria contínua da qualidade. Na prática, falhas poderão ocorrer resultando em insucessos, ou mal funcionamento, com enormes prejuízos econômicos, ou ainda em falhas que podem tornar um projeto longo e de custo elevado. Apesar de existirem diversos padrões propostos para o desenvolvimento de software embarcado, sendo alguns deles referências de qualidade para outras agências espaciais, na prática, torna-se inviável segui-los, visto que a evolução da atual equipe e da organização relativa à qualidade de software ainda não atingiu um nível de maturidade adequada. Com base nesta realidade, está sendo proposto um passo no processo de evolução da qualidade de software que é possível de ser absorvido e seguido pela equipe e ser o início de uma nova etapa na melhoria da qualidade do software embarcado, para que se caminhe na obtenção de produtos de software cada vez mais confiáveis e de qualidade cada vez mais previsível. Para isto, propõe-se um conjunto de atividades adicionais ao atual processo de desenvolvimento, que inclui alterações na metodologia adotada no esquema de controle de qualidade. Na construção dessa proposta foram considerados o perfil e a maturidade da equipe, a complexidade do software em questão e as condições organizacionais específicas do INPE.

ONBOARD SOFTWARE IMPROVEMENT OF INPE´S SATELLITES: PROPOSAL FOR A STEP AHEAD

ABSTRACT

The development of onboard satellite software designed to accomplish critical functions in INPE´s satellites have not counted with the collaboration of software development specialists to aid in continuous improvement of product quality. In practice, the occurrence of failures can result in insucess or malfunctioning, with significant economical losses or, yet, in a very long and expensive project. Despite of the existence of several standards in the area of onboard software development, including some that are considered quality standard by other space agencies in the world, in practice it is impossible to follow or implement them, since the present level of the organization and the development team (related to software quality) is still below the required maturity level. Based upon this scenario a step in the software quality evolution process is being proposed. This step is easy to absorb and to be implemented by the development team; it may give a new boost in the improvement of onboard software quality, aiming to obtain a more dependable, better quality software product. To achieve this goal, we propose a set of additional activities, to be added to the present development process, including modifications in the adopted methodology for quality control strategies. When developing this proposal, we considered the team profile and maturity, the software complexity to be studied and INPE´s organizational conditions.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

O programa de satélites do Instituto Nacional de Pesquisas Espaciais (INPE), iniciado com a chamada Missão Espacial Completa Brasileira (MECB), produziu nos últimos 15 anos 5 satélites, dos quais 3 encontram-se ainda em operação. Entretanto, o desenvolvimento de softwares concebidos para desempenhar funções críticas nos satélites continua sendo realizado por profissionais que não são especialistas em Engenharia de Software. Na prática, é muito grande a possibilidade de ocorrerem falhas críticas quando não se tem um processo de desenvolvimento previamente definido, um gerenciamento de projeto inadequado, ou um processo de desenvolvimento que caminha sem controle (Page-Jones, 1995).

Os métodos tradicionais baseados na geração de código, ou seja, somente na atividade de programação, correm riscos maiores de gerar erros no software produzido e podem resultar em longos períodos dedicados aos testes de detecção e correção de erros, acarretando um aumento no custo e incerteza quanto à remoção efetiva das falhas. Exemplos recentes constatam que softwares embarcados podem falhar mesmo seguindo as recomendações da Engenharia de Software (Leveson, 1993; Nuseibeh, 1997). Entre outros possíveis motivos, a imaturidade das organizações no processo de desenvolvimento constitui um fator considerável na ocorrência de falhas e insucessos que podem ser notados através de prejuízos econômicos, projetos que não atendem cronogramas e perda da missão.

Além de desempenhar funções críticas, grande parte da complexidade do software embarcado deve-se aos aspectos de tempo real do sistema, isto é, um software que deve realizar uma quantidade definida de processamento dentro de um intervalo de tempo limitado. Portanto, o desenvolvimento de software embarcado é considerado uma das tarefas mais complexas e

desafiadoras tratadas pela engenharia de software. Cuidados especiais e atividades adicionais decorrem dessa característica de maneira mais crítica do que os necessários para os sistemas ditos normais. Por exemplo, a aplicação de critérios para a quebra da complexidade, ou seja, a quebra do sistema em partes torna-se uma providência muito importante para melhorar a garantia de qualidade do software embarcado.

De uma maneira geral, o uso sistemático dos princípios de engenharia de software pode resultar em grande melhoria na qualidade do produto de software (Fairley, 1985). No entanto, ainda nos dias de hoje, poucos gerentes e profissionais questionam a necessidade de providências para melhoria da qualidade de software e muitos não estão interessados em investir em esforços necessários para adequar à sua realidade as propostas e recomendações existentes. Algumas das razões aparentes para estas situações são: gerentes que relutam em adicionar no projeto custos extras iniciais voltados à garantia da qualidade sem ter um valor concreto do retorno do investimento a ser feito; os profissionais de desenvolvimento de software acham que estão fazendo absolutamente tudo o que precisa ser feito; ninguém sabe onde colocar pontualmente na organização a responsabilidade pela qualidade e nem percebe que ela deve ser de toda a organização, assumindo cada um uma parcela da responsabilidade; todos querem evitar a “burocracia” que, segundo entendem, será introduzida no processo de melhoria da qualidade na construção do software.

Estudos e trabalhos realizados pelo SEL – *Software Engineering Laboratory* em um esforço conjunto com a NASA, através da FDD - *Flight Dynamics Division* e como o Departamento de Ciência da Computação da Universidade de Maryland, demonstraram a importância da gerência do processo para a obtenção de padrões cada vez mais elevados de qualidade. O SEL investiu durante duas décadas aproximadamente 11% de seu orçamento na melhoria do processo. Entre os resultados obtidos na definição dos processos do SEL (Basili et al., 1995; Zubrow et al., 1994) pode-se ressaltar a institucionalização

da mudança e da melhoria do processo como prática de padrões de negócios, com ênfase na tecnologia orientada à pessoa (*people-oriented*), tais como as revisões e inspeções.

Observa-se, entre as conclusões, a crescente consideração do fator humano na melhoria dos processos, contrariando a expectativa de tornar os processos independentes das pessoas. Portanto, embora uma menor ênfase tenha sido dada à automação, o resultado reafirma a necessidade de uma definição clara e adequada de técnicas e ferramentas como fator conclusivo na constatação de que um processo definido ainda necessita fortemente das pessoas preparadas para implementá-lo. Notou-se também que resultados efetivos na qualidade de software podem ser obtidos como conseqüência do investimento em melhoria do processo de desenvolvimento do software.

O tema desta dissertação foi motivado pela comparação da autora entre o desenvolvimento de dois diferentes projetos de satélite do INPE, na qual ela fez parte da equipe de desenvolvimento de software do subsistema de telemetria (segmento solo) do Satélite de Coleta de Dados (SCD-1) e do desenvolvimento e integração do software da estação dos Satélites Científicos (SACI-1 e SACI-2). O primeiro projeto contava com uma gerência composta por especialistas em Engenharia de Software, enquanto os últimos contavam com uma gerência composta apenas por especialistas na área de satélite. Notória foi a diferença para o desenvolvedor que participou do SCD1 porque se sentia mais seguro e mais confiante nos resultados alcançados em função de existirem diretrizes claras das atividades, bem como padrões bem estabelecidos a serem seguidos.

Este trabalho consiste em apresentar e discutir uma proposta de alterações no processo de desenvolvimento de software embarcado em satélites do INPE como um passo a mais na direção da melhoria da qualidade do software. A ênfase está na obtenção de um produto de software mais confiável, no estabelecimento de diretrizes para todas as atividades essenciais compatíveis

com os conhecimentos da equipe de desenvolvimento e, portanto, com chances de serem aceitas e utilizadas. A proposta poderá atuar como um modelo de referência para futuros projetos podendo ser alterado pela própria equipe de desenvolvimento no que for julgado conveniente. Além do grau de conhecimento da equipe de desenvolvimento relativo à Engenharia de Software, foram consideradas as condições ambientais existentes no INPE para a realização de projetos e a complexidade do software em questão. As atividades propostas para melhoria da qualidade abrangem dois aspectos: um deles diz respeito à metodologia utilizada no desenvolvimento de software embarcado não muito distante da que vem sendo utilizada pela equipe e um outro relacionado às atividades para a garantia de qualidade a serem realizadas ao longo do desenvolvimento, diferentemente do esquema utilizado com as atividades concentradas somente no fim do processo de desenvolvimento.

O Capítulo 2 apresenta a definição, características e processos de desenvolvimento de software para sistemas de tempo real aplicáveis ao software embarcado. São descritos os modelos de ciclos de vida para o software de tempo real, uma evolução dos métodos de projeto baseados no paradigma estruturado e métodos de projeto orientados a objeto. Por fim, é apresentado um método de projeto que define critérios para construção de uma arquitetura para sistema de tempo real, que será incluído na metodologia proposta.

O Capítulo 3 descreve de uma forma geral as características específicas do sistema em que o software embarcado está inserido. São também apresentadas as funções que o software deve desempenhar, os aspectos relevantes dos principais padrões existentes para o desenvolvimento de software crítico e um resumo do documento de padronização do INPE, utilizado como referência pela equipe de desenvolvimento. Por fim, são apresentados o perfil da equipe e o problema existente em relação à forma como os softwares embarcados estão sendo produzidos no INPE. São

abordados os aspectos de garantia de qualidade, e até que ponto o desenvolvimento é sistematizado e documentado e como as revisões são executadas.

No Capítulo 4 é discutida a proposta de evolução da garantia de qualidade, baseada na idéia de um passo a mais na direção desta garantia. É apresentada uma descrição da proposta, que inclui as alterações na metodologia para o desenvolvimento e o esquema de garantia de qualidade para obter um software mais confiável. Estão incluídas as justificativas baseadas nos conceitos de engenharia de software, adequadas ao caso específico do INPE.

No Capítulo 5 descreve-se a metodologia proposta para o desenvolvimento de forma mais detalhada. A metodologia baseia-se em técnicas estruturadas e bem conhecidas na Engenharia de Software para se manter próximo ao que vem sendo utilizado pela equipe e poder ser absorvida com segurança. Além das técnicas a serem usadas a proposta inclui os documentos que devem ser produzidos durante o desenvolvimento de um software embarcado.

O Capítulo 6 apresenta o esquema de controle de qualidade que consiste na realização de revisões técnicas, no controle de alterações e no uso de padrões. O esquema baseia-se em cada um dos produtos gerados a partir da metodologia proposta para o software embarcado, considerando os cargos definidos em função do grau de conhecimento dos membros da equipe sobre a aplicação e o ambiente existente no INPE.

O Capítulo 7 apresenta as conclusões e as perspectivas de trabalhos futuros considerando-se a evolução da qualidade dos softwares embarcados nos satélites do INPE como sendo um processo de melhoria da qualidade passo a passo de maneira contínua e consistente.

CAPÍTULO 2

SISTEMA DE TEMPO REAL E PROCESSOS DE DESENVOLVIMENTO

Por definição, sistemas de tempo real devem realizar uma quantidade definida de processamento dentro de um intervalo de tempo fixado, tais como: controlar dispositivos externos, responder a eventos externos e compartilhar tempo de processamento entre múltiplos processos (Fairley, 1985). Os sistemas de tempo real geram ações em resposta a eventos externos. Para isto, operações de controle e aquisição de dados são realizadas sob restrições de tempo e confiabilidade. Por isso freqüentemente são dedicados a uma única aplicação (Hinden e Rauch-Hinden, 1983).

O sistema de tempo real é formado por vários processos computacionais ou tarefas concorrentes. A concorrência decorre do fato de existirem tarefas assíncronas executadas em velocidades diferentes. Entretanto, de tempos em tempos, as tarefas precisam se comunicar e sincronizarem-se umas com as outras (Gomaa, 1984). Nestes sistemas as entradas chegam de forma intermitente e não estão disponíveis de início, enquanto as saídas são geradas em resposta às entradas (Yen e Wolf, 1996). O sistema tem que responder a certas entradas com restrições de tempo prescritas (Kumar et al., 1996; Xu e Parnas, 1993).

O termo **sistema de tempo real** normalmente refere-se ao sistema inteiro, que compreende a aplicação de tempo real, sistema operacional e subsistemas de E/S, que contêm dispositivos especiais para realizar interface com diversos sensores e acionadores (Gomaa, 2000). Portanto, o software desenvolvido para um sistema de tempo real é classificado como software de tempo real e está intrinsecamente ligado ao mundo real, ou seja, deve responder, monitorar, analisar e controlar elementos do mundo real de acordo com o tempo especificado no domínio do problema. Os elementos incluem um componente de coleta de dados que obtém e formata as informações provenientes de um

ambiente externo, um componente de análise que transforma as informações conforme a aplicação exige, um componente de controle e saída que responde ao ambiente externo e um componente de monitoração que coordena todos os demais componentes de forma que a resposta em tempo real possa ser mantida (Pressman, 1995).

2.1 Características dos Sistemas de Tempo Real

As características dos sistemas de tempo real são descritas como: (Selic et al., 1994):

- Tempo adequado (timing): sistemas de tempo real devem realizar sua função no tempo certo, determinado para cada contexto;
- Estrutura interna dinâmica: - sua estrutura interna deve ser dinâmica, para poderem exercer suas funções em um ambiente cujas propriedades variem no tempo;
- Reatividade: deve responder continuamente a diferentes eventos, cuja ordem de ocorrência no tempo não é sempre previsível;
- Concorrência: deve ser capaz de ativar e gerenciar múltiplas atividades simultâneas;
- Distribuição: podem ser usados múltiplos pontos computacionais cooperativos com o objetivo de executar cooperativamente uma função.

Estas características peculiares aos sistemas de tempo real geram muitas preocupações relacionadas ao projeto de tempo real, tais como: *deadlock*, exclusão mútua, coordenação entre as tarefas de tempo real ou sincronização entre processos, processamento das interrupções de sistema, manipulações

de E/S para garantir que nenhum dado seja perdido, especificação das restrições de *timing* (tempo adequado) internos e externos ao sistema e garantia da precisão e consistência do seu banco de dados.

Deadlock consiste em uma situação indesejável de impasse que ocorre quando todos os processos do sistema estão aguardando pelo término das ações ou pela liberação de recursos de outros processos para que, então, possam prosseguir. A exclusão mútua é necessária para garantir que diferentes processos não estejam utilizando simultaneamente recursos compartilhados. A sincronização é exigida para que processos concorrentes operando em diferentes velocidades possam comunicar-se nos pontos apropriados da execução de cada um.

O desempenho de um sistema de tempo real é determinado a grosso modo pelo tempo de resposta do sistema e pela taxa de transferência de dados. Tempo de resposta consiste no tempo que o sistema leva para detectar um evento interno ou externo e responder com uma ação. O primeiro fator crítico encontra-se no processamento das informações sobre o evento, pois a obtenção da resposta apropriada pode exigir a utilização de algoritmos complexos e consumidores de tempo. Outros fatores que também afetam de forma crítica o tempo de resposta são o *overhead* na troca de mensagens entre processos, o tempo despendido nesta troca, a velocidade de processamento e principalmente o acesso à memória.

A taxa de transferência de dados determina a rapidez com que os dados seriais ou paralelos, analógicos ou digitais precisam ser transferidos para dentro ou para fora do sistema. Os fornecedores de hardware especificam valores de capacidade e *timing* para características de desempenho, mas estas medidas isoladas de um modo geral pouco refletem na determinação do desempenho global do sistema de tempo real. O desempenho final depende da avaliação simultânea do desempenho dos dispositivos de E/S, de disco, latência de barramento, tamanho do *buffer* e outros a serem considerados.

O software de tempo real deve ainda garantir que dados não sejam perdidos, processando continuamente as seqüências de dados de entrada, bem como deve reagir a eventos assíncronos em que a ordem de chegada e o volume de dados não são facilmente previstos.

Exigências especiais de confiabilidade ainda são feitas a esses tipos de sistema, tais como: reinicialização, recuperação após a ocorrência de falhas e redundância interna como mecanismo de *backup.*

Por fim, um fator relacionado à confiabilidade que serve para distinguir este tipo de sistema de qualquer outro tipo é o tratamento de interrupções. O sistema deve responder a estímulos externos (interrupções) no tempo determinado pelo mundo externo. Uma vez que múltiplos estímulos ocorrem, interrupções prioritárias devem ser estabelecidas, isto é, a tarefa considerada mais importante deve ser atendida sem depender da ocorrência de outros eventos. O tratamento de interrupções não envolve somente o processamento da tarefa prioritária, mas também o restabelecimento do programa que foi interrompido. A Figura 2.1 ilustra o fluxo de processamento normal que sofre uma interrupção detectada pelo processador.

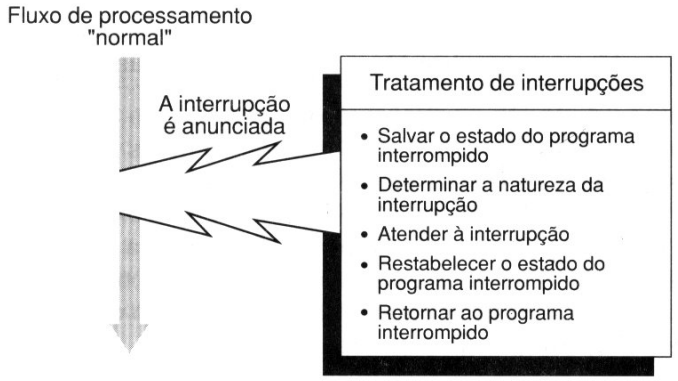

FIGURA 2.1 - Interrupções.

FONTE: Pressman (1995, p. 638).

Em virtude da sua própria natureza, o software de tempo real exige a aplicação de técnicas especiais para as atividades de análise, projeto, implementação e teste que não são necessárias em outros tipos de aplicações. Por isto, o projeto de software de sistemas de tempo real é considerado um dos mais desafiadores e complexos a ser conduzido pela engenharia de software (Pressman, 1995).

2.2 Ciclos de Vida para o Software de Tempo Real

O modelo de ciclo de vida clássico ou em cascata proposto inicialmente por Boehm foi posteriormente ampliado (Fairley, 1985) e é ainda o mais amplamente utilizado para sistemas de software de tempo real. Posteriormente, outros modelos de ciclo de vida foram propostos com o objetivo de suprir as limitações do modelo em cascata ou simplesmente atender as necessidades específicas dos sistemas de tempo real.

Por ter como base um modelo de processo de desenvolvimento idealizado, em que cada atividade é concluída antes do início da próxima atividade sem que ambas se sobreponham, o modelo em cascata leva à ocorrência de dois problemas significativos:

- Os requisitos do software não são previamente testados até que exista uma versão disponível para apresentar ao usuário;
- Uma primeira versão do software somente encontra-se disponível tardiamente dentro do ciclo de vida.

Estas limitações do modelo de ciclo de vida clássico motivaram o aparecimento de modelos para sistemas de tempo real baseados em prototipação. A primeira proposta inclui a construção de um protótipo descartável para auxiliar na especificação de requisitos de sistemas de informações interativos (Agresti, 1986; Gomaa, 1981), permitindo retratar a interface homem-máquina para o

usuário. Neste modelo, o protótipo pode ser desenvolvido depois de uma especificação preliminar de requisitos ou pode ser utilizado como uma prototipação experimental do projeto, quando é necessário determinar a eficiência de um algoritmo, adaptabilidade de um sistema operacional ou a forma que a interação homem-máquina deve assumir.

Outro modelo baseado em prototipação sugeriu um desenvolvimento incremental do sistema sob a forma de uma prototipação evolutiva (McCracken, 1982; Gomaa, 1986a). O desenvolvimento incremental consiste em um método iterativo para desenvolvimento de software em estágios. O modelo incremental de ciclo de vida apresenta como principais características: um enfoque para sistemas modulares baseado em subsistemas, produção de resultados em curto prazo para projetos complexos e ainda a eliminação da perda de controle (custo e prazo). Posteriormente, a experimentação do modelo foi realizada em um sistema controlador de robôs (Gomaa, 1986b).

As atividades do modelo do ciclo de vida utilizado como base no sistema controlador do robô (Gomaa, 1984) seguem a metodologia estruturada, mas apresenta uma diferença se comparado ao ciclo de vida de outros sistemas comerciais: a decomposição do sistema em tarefas. As atividades que compõem ciclo de vida incremental são:

1) Especificação e análise de requisitos: os requisitos do usuário são analisados e um modelo do sistema é especificado para satisfazê-los;

2) Projeto do sistema: o sistema é dividido em processos computacionais concorrentes que são chamados de tarefas e as interfaces entre elas são definidas;

3) Projeto de tarefas: cada tarefa é dividida em módulos e as interfaces entre eles são definidas;

4) Construção de módulos: o projeto detalhado, implementação e teste unitário de cada módulo são realizados;

5) Integração de tarefas e sistema: módulos são gradualmente integrados e testados para formarem as tarefas que, por sua vez, são gradualmente integradas e testadas para formarem o sistema;

6) Teste do sistema: o sistema completo ou subsistemas são testados para verificar a conformidade com a especificação funcional. Maior objetividade pode ser alcançada se os testes do sistema forem conduzidos por uma equipe de teste externa;

7) Teste de aceitação: realizado com a participação do usuário.

Embora as atividades dentro do ciclo de vida do software estejam seqüencialmente dispostas, na realidade é necessário e desejável existir alguma interação entre elas. Para satisfazer esta realidade, o desenvolvimento incremental encoraja a prototipação com o objetivo de produzir uma versão operacional do sistema o mais cedo possível dentro do ciclo de vida. Esta versão vai evoluindo até que esteja totalmente concluída.

Na experimentação (Gomaa, 1984), é construído um protótipo do sistema controlador do robô que evolui através das diversas atualizações da versão inicial. Esta prototipação contribui para verificar necessidades de desempenho, testar componentes críticos do projeto, diversos aspectos do projeto como a eficiência de algoritmos e ainda tornar disponível uma versão operacional do sistema durante as primeiras atividades do desenvolvimento, além de servir para impulsionar a equipe e a gerência. Apesar destes benefícios, é necessário cautela com protótipos que não são criados para serem descartados, visto que a qualidade do software tem que estar contida desde o início do processo, não sendo possível adicioná-la posteriormente. Em particular a arquitetura do sistema precisa ser cuidadosamente projetada e todas as interfaces especificadas.

Em função das vantagens e desvantagens citadas para cada um dos modelos, surgiu uma outra proposta de combinar dois caminhos: o da prototipação descartável e desenvolvimento incremental, isto é, um protótipo é construído para auxiliar na especificação de requisitos. Assim que os requisitos são compreendidos e a especificação é definida, um modelo de ciclo de vida incremental é adotado.

O modelo espiral de ciclo de vida (Boehm, 1988), engloba outros modelos de ciclo de vida, tais como: modelo em cascata, incremental e prototipação. Baseia-se na análise de risco, em uma filosofia evolutiva e em uma maior integração entre usuários e desenvolvedores.

O modelo em espiral encontra-se em propostas recentes para processo de desenvolvimento de software como, por exemplo, o Processo Unificado (*Unified Process*), que utiliza como notação a Linguagem de Modelagem Unificada (UML), orientada em objetos (Jacobson, Booch e Rumbaugh, 1999), para as atividades do processo. Para o Processo Unificado, o ciclo de vida de um produto de software tem um modelo em espiral, em que cada projeto constitui um ciclo de liberação do produto. Cada ciclo é dividido em 4 fases gerenciais definidas como: concepção, elaboração, construção e transição e em cada fase ocorre, em maior ou menor grau, os fluxos de trabalho, cada um caracterizado por um tema técnico específico: requisitos, análise, projeto, implementação e testes. A divisão em fases e fluxos do modelo em espiral promove uma integração maior com o usuário, porém requer sempre gestão mais sofisticada para ser previsível e confiável.

2.3 Métodos para a Atividade de Projeto do Software de Tempo Real

Durante a década de 60, programas eram implementados sem que houvesse praticamente nenhum procedimento de levantamento e análise de requisitos nem de projeto. O fluxograma era a notação gráfica utilizada com freqüência

como uma ferramenta de documentação ou para auxiliar no planejamento do projeto detalhado antes da codificação. Sub-rotinas eram criadas inicialmente para permitir que um pedaço de código fosse compartilhado e, então, chamado de diferentes partes de um programa. Este conceito tornou viável dividir um programa em módulos, onde cada módulo podia ser desenvolvido por uma pessoa e depois ser implementado como uma sub-rotina ou função. Surgiu assim a proposta de construção de sistemas modulares que, posteriormente, foi adotada como ferramenta de gerenciamento de projeto (Dahl, 1972).

No início da década de 70, a programação estruturada passou a ser amplamente utilizada. Como conseqüência, métodos de projeto de programas surgiram com o objetivo de oferecer uma sistematização para o projeto estruturado de programas, tais como: projeto *top-down* e refinamento sucessivo, cuja aceitação passou a merecer destaque.

Em meados dos anos 70, duas diferentes estratégias para projeto de software ganharam destaque: projeto orientado ao fluxo de dados e projeto estruturado de dados. O método de projeto orientado ao fluxo de dados também chamado de projeto estruturado (Yourdon, 1979) foi um dos primeiros métodos compreensivos e bem documentados, propiciando um melhor entendimento das funções do sistema ao ser considerado o fluxo de dados que atravessa o sistema. O método apresenta um caminho sistematizado para o desenvolvimento de diagramas de fluxo de dados para o sistema e o posterior mapeamento para diagramas de estrutura. O projeto estruturado introduziu os conceitos de acoplamento e coesão entre módulos, bem como critérios de avaliação da qualidade de um projeto, enfatizando os conceitos de decomposição funcional em módulos e a definição de interfaces entre módulos. A primeira parte da descrição de projeto estruturado, baseava-se no desenvolvimento do diagrama de fluxo de dados, seu refinamento e ampliação deu origem a um método compreensível de análise, conhecido por análise estruturada (DeMarco, 1978) e (Gane, 1979).

Um método de projeto alternativo foi o projeto estruturado de dados, cuja estratégia para alcançar a compreensão do sistema parte da consideração de estruturas de dados, inicialmente projetando a estrutura de dados e a seguir, projetando a estrutura do programa com base na estrutura de dados (Jackson, 1975; Orr, 1977).

No mundo da base de dados, o conceito de separação entre dados lógicos e dados físicos tornou-se chave para o gerenciamento de sistemas de base de dados. Vários métodos foram defendidos para o projeto lógico de base de dados, como o modelo de entidade e relacionamento (Chen, 1976).

Uma contribuição bastante significativa dada ao projeto de software foi a estratégia de ocultação da informação. O princípio da ocultação de informações determinava que os módulos fossem especificados e projetados de tal forma que as informações (procedimentos e dados) contidas em um módulo fossem inacessíveis a outros módulos que não necessitassem utilizar tais informações. Tornou-se um poderoso conceito de método de projeto para a construção de sistemas mais modulares e com baixo grau de acoplamento entre os módulos. A técnica foi adotada por outros métodos de projeto, tais como: projeto estruturado Mascot (Simpson, 1986) e projeto orientado a objeto (Booch, 1986). Entretanto, o maior problema apresentado pelos primeiros sistemas desenvolvidos utilizando este método é que, apesar de serem modulares, houve o emprego generalizado de dados globais. Isto tornou estes sistemas mais propensos a erros, e dificultou muito sua manutenção.

No final dos anos 70 surgiu a maior contribuição para o projeto de sistemas concorrentes e de tempo real, com a notação de Mascot (Simpson, 1979) e mais tarde o método de projeto Mascot (Simpson, 1986). O método voltava-se às necessidades dos sistemas de tempo real por defender a idéia de decomposição do sistema em tarefas ou processos concorrentes, estabelecendo o tipo de interface entre as tarefas. Embora um diagrama de rede de tarefas fosse definido, o método **não abordava como** estruturar o

sistema em tarefas e nem a estrutura individual de cada tarefa. O método baseava-se no fluxo de dados para formalizar a comunicação entre tarefas através de canais para comunicação de mensagens ou por depósitos. Estes depósitos eram módulos de ocultação que encapsulavam estruturas de dados compartilhadas. O acesso aos dados contidos nos canais de comunicação ou nos depósitos era indireto, isto é, não era realizado pela tarefa que precisava ter acesso aos dados, mas por rotinas responsáveis pela sincronização deste acesso. O sincronismo era obtido por meio de semáforos, também chamados de semáforos binários, em que uma variável lógica é utilizada para impor que somente uma tarefa de cada vez tenha acesso aos dados (exclusão mútua).

Durante os anos 80, houve um amadurecimento em termos de métodos de projeto de software com a introdução de vários métodos. A implantação de um método para redução de custo de software (Parnas, Clements e Weiss, 1984) no Naval Research Lab (NRL), permitiu que Parnas explorasse a utilização do método de ocultação de informação em larga escala ao conduzir o desenvolvimento de diversos projetos de software. Nestes projetos, foram aplicados métodos de análise estruturada e projeto estruturado para sistemas concorrentes e de tempo real, tais como: análise e projeto estruturado para sistema de tempo real - Real-Time Structured Analysis and Design (RTSAD) (Ward, 1985 ; Hatley e Pirbhai, 1988) e o Método de Projeto para Sistemas de Tempo Real - Design Approach for Real-Time System (DARTS) (Gomaa,1984). Utilizando a abordagem estruturada o método DARTS foi considerado um método representativo por oferecer mecanismos que suportam adequadamente os projetos de tempo real. O método descreve **como** estruturar o sistema em tarefas ao estabelecer critérios para divisão em tarefas concorrentes e o tipos de interface entre elas.

2.4 Evolução dos Métodos de Análise e Projeto Orientados a Objeto

A popularidade e o sucesso da programação orientada a objeto aumentou a partir do meio da década de 80. Este fato estimulou o aparecimento de diversos métodos para a atividade de projetos orientados a objeto. O Método para Desenvolvimento de Sistema de Jackson - *Jackson System Development* (JSD) (Jackson,1983) foi o precursor da modelagem de interação de objetos, considerado um aspecto essencial nos métodos de desenvolvimento orientado a objeto utilizados hoje. O JSD considera o projeto como uma simulação do mundo real, em que é enfatizada a modelagem de entidades no domínio do problema, através do uso de tarefas concorrentes. Portanto, foi um dos primeiros métodos a defender que um projeto deveria representar um modelo da realidade, sob a forma de uma rede de tarefas concorrentes, em que cada entidade do mundo real equivale a uma tarefa concorrente. Aspectos como projeto *top-down* também foram incorporados.

Todos os métodos orientados a objeto que surgiram após o JSD enfatizaram a modelagem do domínio do problema, a ocultação de informações e herança (Booch, 1991; Wirfs-Brock, 1990; Rumbaugh et al., 1991; Shlaer e Mellor, 1988; Coad e Yourdon, 1991). Foram incorporados conceitos como o de Parnas que defendeu a ocultação de informações como uma forma de projetar módulos que fossem auto-consistentes e que, conseqüentemente, pudessem ser alterados causando pouco ou nenhum impacto em outros módulos. Booch introduziu o conceito de projeto orientado a objeto (Booch, 1986), onde inicialmente foram utilizados projetos baseados em objetos com ocultação de informações. As decisões passaram a ser projetadas em objetos de tal forma que as interfaces dos objetos revelassem somente o que o usuário necessitasse saber. Em seguida, o software foi implementado em linguagem Ada e o conceito de ocultação de informações foi estendido para o conceito de classes e herança. O conceito de orientação a objeto também foi introduzido na atividade de análise do ciclo de vida. A ênfase estava em identificar objetos do mundo real contidos no domínio do problema e mapeá-los como objetos do

software. A modelagem em objetos foi classificada, inicialmente, como modelagem estática, por seguir a idéia inicial de modelagem da informação vinda do modelo entidade-relacionamento (E-R) de Chen. No modelo E-R as entidades, os atributos de cada entidade e o relacionamento entre as entidades são determinados e representados nos diagramas de E-R. Durante o projeto, o modelo E-R é mapeado para uma base de dados relacional. Durante a atividade de análise da modelagem estática, objetos no domínio do problema são identificados e modelados como classes de software e os atributos de cada classe são determinados, bem como o relacionamento entre classes (Booch, 1994; Coad e Yourdon, 1991; Rumbaugh et al., 1991; Shlaer e Mellor, 1988).

A principal diferença entre classes, na modelagem estática da orientação a objeto, e entidades, no modelo E-R consiste no fato de classes realizarem operações. De forma que a modelagem da informação além das classes de entidades persistentes que serão armazenadas em um banco de dados, estão envolvidas outras classes no domínio do problema que também deverão ser modeladas por realizarem operações. Por isto, uma modelagem da informação avançada envolve conceitos de agregação, generalização/especialização.

Os primeiros métodos propostos para análise e projeto orientados a objeto deram ênfase aos aspectos estruturais do desenvolvimento de software como a ocultação de informação e herança, mas negligenciaram aspectos dinâmicos. Uma contribuição importante partiu da Object Modeling Technique (OMT), ao ressaltar que a modelagem dinâmica era igualmente importante (Rumbaugh et al., 1991). A modelagem dinâmica era composta por um diagrama de estado, mostrando o comportamento do sistema em função do estado de objetos ativos e mais um diagrama de seqüência, mostrando a seqüência de interações entre objetos. A utilização de um diagrama hierárquico de transição de estados foi proposta inicialmente (Harel, 1988), para a modelagem de objetos ativos. Outras propostas de modelagens de objetos envolveram diagramas de transição de estado (Mellor e Shlaer, 1992),

diagramas de objetos que instanciavam o nível de interações sobre objetos (Booch, 1991) e, mais tarde, diagramas de objetos com interações numeradas seqüencialmente, que passou a ser reconhecido como diagrama de colaboração entre objetos.

Em 1992 foi introduzido o conceito de casos de uso para modelagem de requisitos funcionais do sistema (Jacobson, 1992). Para objetos que participavam em um caso de uso era utilizado o diagrama de seqüência para descrever a seqüência de interações entre eles. O conceito de casos de uso foi fundamental para todas as atividades do ciclo de vida orientado a objeto de Jacobson, pois pode também ser aplicado para sistemas que não são orientados a objeto.

Embora a OMT tenha sido a notação mais utilizada, não havia uma unificação das notações e métodos orientados a objeto antes da proposta da linguagem de modelagem unificada (UML). Porém, existiram tentativas de unificação como a proposta por Fusion (Coleman et al., 1993) e (Texel e Williams, 1997). A UML foi originalmente desenvolvida por Booch, Jacobson e Rumbaugh com o objetivo de integrar as notações das modelagens de caso de uso, modelagens estática e dinâmica (diagrama de estado e modelagem de interação de objetos). Desde então, outros metodologistas permanecem contribuindo para o desenvolvimento da UML (Cobryn, 1999; Selic, 1999). Atualmente a UML é um padrão para modelagem em objeto mantido pela *Object Management Group* (OMG) (Gomaa, 2000).

2.5 Métodos de Projeto Orientados a Objeto para Sistemas de Tempo Real

O método de projeto concorrente para sistemas de tempo real - Concurrent Design Approach for Real Time Systems (CODARTS) (Gomaa, 1993), foi construído com base nos primeiros métodos de projetos concorrentes, projetos de tempo real e projetos orientados a objeto. Estão incluídos os métodos para

redução de custos de software do Laboratório de Pesquisa Naval de Parnas (NRL) (Parnas, Clements e Weiss, 1984), projeto orientado a objeto de Booch (Booch, 1991), o JSD (Jackson, 1983) e o método DARTS baseado na estruturação em módulo de ocultação de informação e na estruturação em tarefas (Gomaa, 1984; Gomaa, 1986b).

No método CODARTS, a concorrência e o *timing* (tempo adequado) são considerados durante o projeto de tarefas e a ocultação de informação é considerada durante o projeto de módulos. Sendo assim, tarefas correspondem a objetos ativos enquanto módulos de ocultação de informação correspondem a objetos passivos. O sistema de tempo real passa a ser visto a partir de uma perspectiva estática e dinâmica. A visão dinâmica é decorrente do critério da estruturação em tarefas, enquanto a visão estática resulta do critério de estruturação em módulos de ocultação de informações. Em função destes critérios são providas diretrizes para integração de tarefas e módulos.

Outro método para projeto específico para tempo real é conhecido como OCTOPUS (Awad, Kuusela e Ziegler, 1996) e resultou da composição dos casos de uso de Jacobson, a modelagem estática de Rumbaugh e diagramas de estado. Para o projeto de tempo real, OCTOPUS enfatizou as interfaces com dispositivos externos, a estruturação em tarefas concorrentes, antecipando notações que agora pertencem a UML (Gomaa, 2000).

O método de projeto de tempo real ROOM (Selic, Gullekson e Ward, 1994) foi concebido para ser utilizado com uma ferramenta CASE - *Computer Aided Software Engineering* chamada *ObjectTime*. ROOM baseia-se em atores, que são objetos ativos modelados através de uma variação do diagrama de estado nomeada diagrama de ROOM. Este diagrama define os estados de um ator e apresenta sinais que indicam as transições entre estados e as ações realizadas durante as transições.

Similar ao diagrama de seqüência, um interessante diagrama chamado *use case map* (baseado no conceito de caso de uso), foi introduzido com o objetivo de obter uma modelagem dinâmica para produção em larga escala de sistemas (Buhr, 1996). O diagrama mostra a seqüência de interações entre objetos ou agrega objetos na forma de subsistemas em um nível de detalhes compatível com o diagrama de colaboração.

Métodos como ROOM ou OCTOPUS eram insuficientes para composição de sistemas de tempo real e orientação a objeto (Eriksson e Penker, 1998). Um método para sistemas de tempo real orientado a objeto deve prover meios para: determinar um modelo que separe processos de objetos, identificar e definir classes ativas, representar a comunicação e sincronização de classes ativas, bem como um método para mapeamento adequado do ambiente de implementação. Somente mais tarde, Douglass faz uma descrição detalhada de como aplicar a notação UML para o desenvolvimento de software de tempo real (Douglass, 1999b; Douglass, 1999a). Então, são abordados diversos tópicos, tais como: a definição de classes ativas, sistemas críticos, interação com sistema operacional, escalonamento de processos, padrões (*patterns*) e *frameworks* para tempo real, depuração e testes.

O *Concurrent Object Modeling Architetural Design with UML* (COMET) é o mais recente método de projeto de software proposto para aplicações concorrentes, distribuídas e de tempo real (Gomaa, 2000). O modelo COMET de ciclo de vida do software orientado a objeto é compatível com o Processo Unificado de Desenvolvimento de Software (USDP) (Jacobson, Booch e Rumbaugh, 1999) e com o modelo em espiral (Boehm, 1988). O processo de desenvolvimento de software é bastante interativo e baseia-se no conceito de casos de uso. Os requisitos funcionais do sistema são definidos em termos de atores e casos de uso. Para cada caso de uso é definida uma seqüência de interações entre um ou mais atores e o sistema.

Um caso de uso pode ser visto em vários níveis de detalhe. Na modelagem de requisitos, os requisitos funcionais do sistema são definidos em termos de atores e casos de uso. A atividade seguinte é a modelagem de análise, em que os casos de uso são refinados para descrever os objetos que participam e a interação entre eles. São realizadas as atividades de construção da modelagem estática, modelagem de máquina de estados finitos, estruturação em objetos e, por fim, a modelagem dinâmica do sistema. Finalizada esta atividade, inicia-se a atividade de modelagem de projeto, cujo objetivo é definir a arquitetura do software e determinar soluções para questões de distribuição, concorrência e ocultação de informação. O modelo de análise enfatiza o domínio do problema, que será mapeado para o modelo de projeto, que enfatiza o domínio da solução. O critério é estruturar o sistema em subsistemas, em que cada subsistema é considerado uma composição de objetos. Uma consideração especial é feita para o projeto de subsistemas distribuídos, por serem estes componentes configuráveis que comunicam-se uns com os outros através de mensagens. Para sistemas seqüenciais, a ênfase está em conceitos de ocultação de informação, classes e herança da orientação a objeto. Porém, para projetos de sistemas concorrentes tais como tempo real, cliente/servidor e aplicações distribuídas é necessário unir o conceito de tarefas concorrentes ao conceito de orientação a objeto.

Finalizado o projeto de arquitetura, uma construção incremental do software é iniciada. É, então, selecionado um subconjunto do sistema a ser construído em cada incremento, através da escolha dos casos de uso que estão incluídos nesses incrementos e objetos que participam de cada caso. A construção incremental abrange o projeto detalhado, codificação e teste unitário das classes do subconjunto do sistema, de forma que o software vai sendo construído gradualmente até que todo o sistema seja construído.

2.6 Critérios para Construção de uma Arquitetura de Software para Sistema de Tempo Real

Entre os diversos métodos de projetos para sistemas de tempo real citados, **aspectos relevantes** do método *Design Approach for Real-Time System* - DARTS (Gomaa, 1984) serão apresentados em função deste tornar-se adequado a esse estudo, para num primeiro passo se ter um método que aproveite a cultura do grupo (programação estruturada). O método DARTS através da abordagem estruturada, oferece mecanismos que suportam adequadamente os projetos de tempo real quando descreve como estruturar o sistema em tarefas concorrentes, estabelecendo critérios para essa quebra e definindo os tipos de interface entre elas.

Uma vez que o projeto de software de tempo real deve incorporar as características dos sistemas de tempo real, a construção do projeto envolve soluções para problemas específicos, tais como:

- Representação de interrupções e troca contextual;
- Concorrência por multitarefas e multiprocessamento;
- Comunicação e sincronização entre tarefas;
- Amplas variações nas taxas de dados e de comunicações;
- Representação nas restrições de tempo;
- Processamento assíncrono
- Acoplamento inevitável a sistemas operacionais, hardware e outros elementos externos ao sistema.

Um método de projeto de software de tempo real que suporte adequadamente todos os problemas mencionados acima deve basear-se em extensões às representações de fluxo de dados que oferecem mecanismos para o projeto de

software de tempo real. A abordagem denominada **Método de Projeto para Sistemas de Tempo Real** - *Design Approach for Real-Time System* (DARTS) permite que projetistas de sistemas de tempo real adaptem técnicas de fluxo de dados às necessidades especiais de aplicações de tempo real (Gomaa, 1984). O método pode ser imaginado como uma ampliação do método clássico de análise e projeto estruturados ao criar uma abordagem baseada na divisão do sistema em tarefas, bem como um mecanismo para definir as interfaces entre elas. O método inicia-se com a aplicação dos princípios fundamentais de análise de software, ou seja, análise do domínio de informação e partição do problema aplicados no contexto da notação de fluxo de dados. Sendo assim, medidas de qualidade de projeto como modularidade e independência funcionais são reforçadas com métodos orientados a fluxo de dados.

O método DARTS amplia os métodos de fluxo de dados, pois fornece um mecanismo para representar a comunicação e a sincronização de tarefas, uma notação para representar a dependência de estado e uma abordagem que une os métodos de fluxo de dados convencionais às características de tempo real.

2.6.1 Estruturando o Sistema em Tarefas

Uma tarefa pode ser definida com sendo um processo manual ou um processo computacional. Um processo manual consiste em uma seqüência de ações executadas por uma ou mais pessoas. Um processo computacional consiste em uma seqüência de estados ordenados no tempo, cuja passagem de um estado para outro acontece com a execução de uma instrução do programa. Ambos possuem uma característica seqüencial, por não existirem instruções executadas em paralelo. Um processo computacional é um programa em execução no ambiente de um sistema de computação, ou seja, uma entidade capaz de causar o acontecimento de eventos e que pode se comunicar e/ou se sincronizar com outros processos. O processador faz o processo

computacional avançar de um estado para o outro, podendo ser ativado, suspenso, reativado, abortado, e terminado. Se um processo computacional é ativado mais de uma vez com os mesmos dados, ele passa pelos mesmos estados, na mesma seqüência, e darão os mesmos resultados, independente de sua velocidade de execução. Quando um processo computacional é suspenso, o seu vetor de estado é armazenado de modo que ele possa ser reativado mais tarde, a partir do ponto onde parou.

O estado de um processo computacional é caracterizado por um conjunto de informações que são:

- O programa que está em execução;
- A indicação da próxima instrução a ser executada;
- Os valores das variáveis e dos dados do programa;
- Os estados e a posição de todos os equipamentos de E/S em uso.

Um sistema é constituído por um conjunto de tarefas que podem concorrer por dados, e por recursos de execução (CPU, memória, equipamentos de E/S, pessoas etc.). A concorrência ocorre porque as tarefas são executadas em velocidades diferentes, normalmente de maneira assíncrona.

Visto que cada tarefa será composta por um ou mais processos do DFD que juntos formam um programa, a cada tarefa deve-se, então, aplicar o método de projeto tradicional e construir o diagrama de estrutura de módulos. O modelo funcional de um sistema de tempo real representado em um Diagrama de Fluxo de Dados não descreve as tarefas assíncronas e concorrentes que implementam o fluxo dos dados, portanto, neste ponto é empregada uma abordagem que identifique as tarefas do sistema a partir dos processos do DFD. O fluxo de dados entre tarefas determina os requisitos de comunicação entre elas. Para determinar se os processos devem ser definidos como tarefas

separadas ou agrupados com outros processos, foram definidos os seguintes critérios (Gomaa, 1984):

Dependência de E/S - Dependendo da entrada ou da saída, uma transformação muitas vezes se limita a funcionar a uma velocidade ditada pela velocidade do dispositivo de E/S com a qual ela está interagindo. Neste caso a interação precisa ser uma tarefa separada;

Funções críticas quanto ao tempo - Uma função crítica quanto ao tempo precisa ser executada com prioridade elevada e, por conseguinte, tem que ser uma tarefa separada.

Requisitos computacionais - Uma função (ou conjunto de funções) computacionalmente intensiva pode ser executada como uma tarefa de prioridade mais baixa, consumidora de ciclos de CPU extras.

Coesão funcional - Transformações que executem um conjunto de funções estreitamente relacionadas podem ser agrupadas numa tarefa. Uma vez que o tráfego de dados entre essas funções pode ser elevado, mantê-las como tarefas separadas aumentará o *overhead* do sistema, enquanto implementar cada função como um módulo separado dentro da mesma tarefa garantirá a coesão funcional tanto em níveis modulares como de tarefas.

Coesão temporal - Certas transformações realizam funções que são levadas a efeito ao mesmo tempo. Essas funções podem agrupadas numa tarefa, de forma que elas possam ser executadas cada vez que a tarefa receber um estímulo.

Execução Periódica - Uma transformação que precisa ser executada periodicamente pode ser estruturada como uma tarefa separada que é ativada em intervalos regulares.

Tarefas compartilham um único processador ou são executadas simultaneamente em processadores distribuídos. Além disto, as informações

que devem fluir entre uma tarefa e outra, isto é, os fluxos de dados de entrada e de saída de cada tarefa devem permanecer inalterados em relação ao DFD construído. Para representar estes tipos de interface entre tarefas o método DARTS proporciona um mecanismo para manusear as informações e representar a **sincronização e a comunicação entre tarefas**. A sincronização consiste em uma forma de interferência entre tarefas, que se caracteriza pela **inexistência** de transferência de dados de uma tarefa para outra. A sincronização entre tarefas pode ocorrer de duas formas: *exclusão mútua e sinalização*. A exclusão mútua acontece quando a sincronização é necessária para que um dado recurso seja acessado simultaneamente por duas ou mais tarefas. Semáforos são utilizados para interromper uma tarefa enquanto outra (de leitura ou escrita) está acessando os dados. A sinalização acontece quando uma tarefa sinaliza à outra sobre a ocorrência de um certo evento. A tarefa que recebe o sinal sofre uma interrupção e altera o seu estado ou comportamento.

A comunicação entre tarefas consiste na comunicação por mensagens e ocorre quando uma tarefa deve **transmitir dados** a outra. Quando a tarefa que envia a mensagem aguarda uma resposta para prosseguir sua execução, a comunicação é fortemente acoplada. Quando a tarefa que envia utiliza uma fila de mensagens para acumular mensagens prosseguindo o processamento, a comunicação é fracamente acoplada.

Portanto, são definidas no método DARTS duas classes de módulos de interface (Gomaa, 1984): **o módulo de comunicação entre tarefas -** *Task Communication Module (TCM)* e **o módulo de sincronização de tarefas** - *Task Syncronization Modules (TSM).*

Um TCM é gerado por uma tarefa que se comunica utilizando primitivas de sincronização do sistema operacional para garantir o acesso adequado aos dados. Os TCMs são classificados em dois tipos: comunicação por troca de mensagens ou dados.

Módulo de comunicação por mensagem - *Message Communication Module (MCM)*: Suporta comunicação por mensagens e implementa mecanismos apropriados ao gerenciamento de uma fila de mensagens para comunicação fracamente acoplada e primitiva de sincronização para comunicação fortemente acoplada. Uma mensagem é um conjunto de informações, geralmente constituído de duas partes. Uma parte contém informações de controle e um cabeçalho com a identificação do remetente e destinatário, por exemplo. A outra parte contém as informações propriamente ditas que uma tarefa quer passar à outra.

Módulo de ocultação de informações - Information Hiding Module (IHM) - Proporciona acesso a um depósito de dados, implementa a estrutura de dados e métodos de acesso a ela por outras tarefas concorrentes. Os dados compartilhados são acessados pelas tarefas para leitura ou para leitura e escrita.

A Figura 2.2 a seguir, ilustra a notação DARTS para os mecanismos de manuseio de informações entre tarefas.

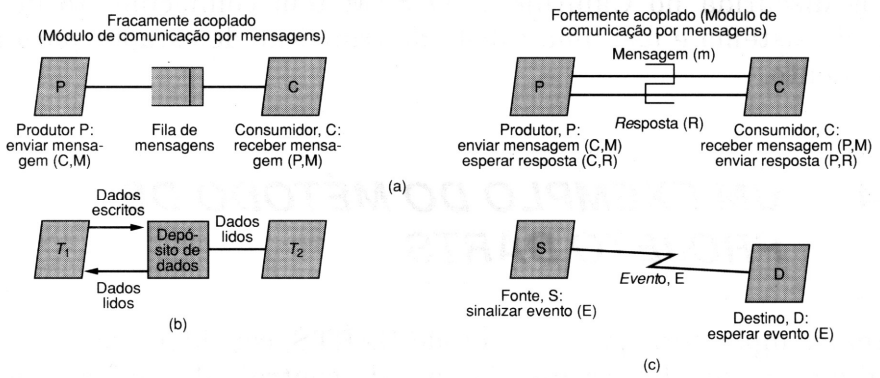

FIGURA 2.2 - (a) Comunicação por mensagens, (b) Módulo de ocultação de informações, (c) Sincronização de tarefas.
FONTE: Adaptada de Gomaa (1984 p.941 e 942).

2.6.2 Estruturando as Tarefas em Módulos

Após a definição das interfaces entre tarefas, o próximo passo é estabelecer a estrutura individual de cada uma. Cada qual corresponde a um programa seqüencial, em que um DFD é desenhado e derivado um diagrama de estrutura de módulos, conforme método de projeto (Page-Jones, 1988).

2.6.3 Implementação

Para implementar a comunicação por mensagens, podem ser utilizadas três abordagens diferentes, tais como (Gomaa, 1984):

- Um sistema operacional de tempo real pode oferecer primitivas de comunicação entre tarefas (por exemplo, [Kung, A. e Kung, R, 1985]).
- Mecanismos de comunicação entre tarefas podem ser implementados dentro do contexto de uma linguagem de programação (por exemplo, Ada).
- Um manipulador de comunicações especial pode ser criado, usando-se primitivas de sincronização oferecidas pelo sistema operacional [Simpson e Jackson, 1979].

2.7 Garantia de Qualidade do Software de Tempo Real

Garantia de qualidade do software é o nome dado ao conjunto de atividades que tem por objetivo validar e verificar o software, certificando assim, a qualidade do produto de software. O objetivo da validação do software é verificar se a equipe de desenvolvimento está construindo o sistema de acordo com as necessidades do usuário, enquanto a verificação tem por objetivo certificar se o produto de software está de acordo com as especificações.

A realização de revisões técnicas formais constitui a atividade mais valiosa para garantia de qualidade de software por ser uma atividade voltada a validação e verificação do software. Entre as principais atividades dos revisores estão:

- Revisões do projeto de tarefas e projeto de módulos: certificação da consistência com a especificação de requisitos e projeto do sistema;
- Verificação da corretude e consistência de projetos dos subsistemas;
- Revisão de módulos críticos: análise do impacto sobre o sistema no caso de alterações nas especificações de requisitos ou projeto do sistema, de forma a recomendar a aceitação ou propor solução alternativa.

Protótipos concebidos como descartáveis também podem ser usados para validação do sistema (antes do desenvolvimento), verificando se este corresponde às especificações do usuário ou para prototipação experimental do projeto.

Para aplicações de tempo real, analisar o desempenho do sistema de tempo real antes da implementação é também necessário para estimar se ele corresponde ao desempenho esperado. Desta forma problemas de desempenho podem ser detectados e eliminados nas primeiras atividades do ciclo de vida. Alguns métodos para avaliação de projetos de software utilizam modelos de enfileiramento (Menascé, Almeida e Dowdy, 1994; Menascé, Gomaa e Kerschberg, 1995; Menascé e Almeida, 1998; Menascé e Gomaa, 1998). O modelo de enfileiramento consiste em uma representação matemática que analisa a contenção, ou seja, a situação que ocorre quando dois ou mais dispositivos estão tentando se comunicar com a mesma parte de um equipamento. O barramento de contenção (*contention bus)* é um sistema de controle de comunicação onde um dispositivo tem que esperar (retardo de contenção) por um momento livre antes de transmitir dados. O retardo de

contenção (*contention delay)* corresponde ao intervalo de tempo à espera da liberação do equipamento para uso.

Modelos de simulação são utilizados, para avaliação de desempenho global, (Smith, 1990). Um modelo de simulação consiste em uma representação algorítmica de um sistema que reflete a estrutura e comportamento do sistema em relação ao tempo, pois considera explicitamente a passagem de tempo

Para análise e modelagem de sistemas concorrentes são utilizados os métodos formais como as redes de Petri (Pettit e Gomaa,1996) a rede de Petri temporizada (com restrições de tempo) (Elmstrom et al., 1993). Elas consistem em um modelo matemático dinâmico que possui uma notação gráfica contendo as posições e as transições em sistemas concorrentes.

Na atividade de teste, os aspectos que diferenciam um sistema de tempo real dos outros estão relacionados com o fato do sistema de software conter tarefas concorrentes ou por fazer interface com diversos dispositivos externos.

A maior dificuldade em testar sistemas concorrentes deve-se ao fato da execução destes sistemas não ser determinística, isto é, a resposta do sistema às entradas varia de uma forma que é difícil predizer. Tai descreve um método para a realização de testes determinísticos em sistemas concorrentes (Tai et al., 1991).

A natureza assíncrona das aplicações de tempo real acrescenta um elemento difícil na realização de teste: **o tempo**. O projetista de casos de teste não deve considerar apenas os testes de caixa branca para verificar aspectos internos do software e caixa preta para validar a especificação dos requisitos do software. Devem também ser considerados o limite de tempo e o paralelismo das tarefas que manipulam os dados. Além disto, a dependência entre o software de tempo real e o hardware em que está inserido pode causar problemas de teste. Os testes de software devem considerar o impacto das

falhas de hardware sobre o processamento do software, porém estas falhas podem ser extremamente difíceis de ser simuladas realisticamente.

Uma estratégia global para projeto de casos de teste de sistemas de tempo real consiste nos seguintes passos:

1) **Teste comportamental**: utilizando os modelos de sistema criados a partir de uma ferramenta - *Computer Aided Software Engineering* (CASE) é possível simular o comportamento de um sistema de tempo real e examinar esse comportamento como uma conseqüência à ocorrência de eventos externos. Estas atividades de análise servem como base para o projeto de casos de teste a ser realizado após a construção do software. Cada um dos eventos, tais como interrupções, sinais de controle e dados, são testados individualmente e o comportamento do sistema é examinado buscando a detecção de erros. O comportamento do modelo de sistema desenvolvido durante a atividade de análise pode ser comparado ao software executável a fim de se obter conformidade.

2) **Teste de tarefas**: cada tarefa deve ser testada individualmente, ou seja, testes de caixa branca são projetados e executados para cada tarefa. Cada tarefa é executada independentemente durante esses testes. O objetivo do teste de tarefas consiste em revelar erros de lógica ou função.

3) **Teste de integração de tarefas**: o objetivo desta atividade de teste é detectar erros relacionados ao tempo. As tarefas assíncronas que se comunicam entre si são testadas com diferentes taxas de dados e cargas de processamento para determinar se ocorrerão erros de sincronização entre tarefas. Também são testadas as tarefas que se comunicam por filas de mensagens para detectar erros na determinação do tamanho das áreas de armazenagem de dados.

Portanto, um método sistemático para testes de integração de tarefas concorrentes deve ser seguido (Gomaa, 1986b).

4) **Teste do sistema**: consiste na atividade de testar os sistemas de software e hardware integrados para verificar se o sistema como um todo atende a especificação de requisitos.

Durante os testes, vários aspectos do sistema de tempo real precisam ser avaliados (Beizer, 1995). Entre eles, estão incluídos:

- Teste funcional - Para determinar se as funções que o sistema executa correspondem às descritas na especificação de requisitos.
- Teste de estresse - Para determinar se o sistema consegue realizar e atender a toda a carga de trabalho esperada para a atividade operacional.
- Teste de desempenho - Para determinar se o sistema atende aos requisitos dentro do limite de tempo determinado.

A realização de testes de sistemas de tempo real deve contar com o apoio de simuladores de ambiente (Gomaa, 1986b), no qual o comportamento do ambiente externo é simulado, permitindo que o software seja testado em um ambiente controlado e reprodutível.

CAPÍTULO 3

DESENVOLVIMENTO DE SOFTWARE DO SISTEMA EMBARCADO EM SATÉLITE

Entre os sistemas embarcados em satélite, o computador de bordo é o sistema alvo desse estudo. O termo refere-se ao sistema de software mais o computador propriamente dito. Embora este sistema de software seja fortemente dependente do hardware no qual está inserido, aspectos específicos do hardware não serão abordados. São apresentadas as características do sistema, a criticidade das funções que deve realizar, os padrões para o desenvolvimento existentes bem como o desenvolvimento do software no INPE.

O computador de bordo é um sistema de tempo real (vide Capítulo 2), cujo comportamento é definido por sua interação com o ambiente e pela seqüência de estados. Tais sistemas estão constantemente respondendo a eventos externos (Gajski et al., 1994). Um sistema embarcado é um sistema eletrônico inserido em um dado equipamento, que realiza funções de supervisão e controle de processos, associados a um equipamento ou processo externo com o ambiente (Kündig, 1987). Nestes sistemas as entradas chegam de forma intermitente e não estão inicialmente disponíveis, enquanto as saídas são geradas em resposta às entradas (Yen e Wolf, 1996).

As funções de supervisão e controle de processos atribuídas ao computador de bordo, compreendem as funções de controle e monitoramento de pelo menos outros quatro sistemas de suporte contidos em um satélite ou segmento espacial: propulsão, energia ou suprimento de potência, comunicação de dados e controle de atitude. O sistema de propulsão é responsável pelo controle da estabilidade e orientação do satélite. O sistema de suprimento de energia é responsável pelo consumo de energia elétrica. O sistema de comunicação é dividido em canais fisicamente distintos para receber

comandos da estação de solo e transmitir dados de telemetria dos demais sistemas do segmento espacial para a estação de solo, que em conjunto com o centro de controle de satélite formam o segmento solo. No sistema de controle de atitude, em geral, é utilizado um computador de bordo dedicado ao controle da orientação e estabilidade do satélite. Para estes sistemas estão associadas tarefas de controle, tais como monitoramento e acionamento de rodas de reação e giroscópios, controle da orientação das células solares, carga de bateria, transientes de energia e temperatura, recepção e envio de dados e cálculos que auxiliam o sistema de controle de atitude.

Além dos sistemas de suporte do satélite, o segmento espacial pode conter aplicações de carga útil ou experimentos científicos, cujas tarefas de controle e envio de dados para o segmento solo são realizadas também pelo computador de bordo.

3.1 Fatores Diferenciadores do Sistema Embarcado em Satélite

Além das características de tempo real (descritas no capítulo 2), características específicas do sistema embarcado são classificadas como fatores diferenciadores (Selic et al., 1994), e descritas a seguir.

3.1.1 Distribuição

A arquitetura do sistema embarcado é definida de acordo com os requisitos da missão e as necessidades operacionais. Pode ser composta por uma única unidade (único processador) ou múltiplas unidades (vários processadores).

A maior parte dos sistemas embarcados utiliza um único processador. Entretanto, existem os sistemas distribuídos, isto é, que comportam processos que são executados em diversas CPUs conectadas por um link de comunicação (Yen e Wolf, 1996). De uma forma geral, a distribuição ocorre

quando o sistema ou aplicação concorrente é executado em um ambiente que consiste de múltiplos nós e que estão em locais geograficamente diferentes. Ou seja, a distribuição em software significa a construção de uma configuração do sistema constituída de vários nós dispersos, que estão interligados por uma rede local. Em muitos casos, vantagens são obtidas ao se construir um sistema distribuído como em caso de falha da CPU, pois o sistema poderá funcionar parcialmente, ou até normalmente se outra CPU assumir as funções. Para os satélites multimissão, em que um sistema mais complexo pode ser construído com o aumento de número de CPUs. Para o controle de muitos dispositivos (sensores) independentes com resposta crítica no tempo, em que um único processador pode não ser capaz de responder dentro de um limite de tempo adequado, então, múltiplos processadores podem ser requeridos. Para os sistemas embarcados sob controle que podem estar fisicamente distribuídos, podem existir processadores especializados que deslocam funções como aquisição de dados ou comando para os sistemas.

Com a distribuição, a complexidade aumenta entre componentes de hardware e software uma vez que os dispositivos estão interligados por uma rede, pois o desempenho passa a depender do tráfico de mensagens entre os nós e da ocorrência de eventuais perdas de link de comunicação ou de conexão com um nó de processamento. Um software de controle de processo, quando distribuído, realiza uma reconfiguração dinâmica da topologia da rede e deve suportar um aumento da carga de processamento. Neste caso, a dificuldade está em garantir o tempo de resposta, que vai depender do tipo de sistema. Porém, no caso do sistema embarcado, a distribuição não é um problema, visto que se tem o controle da carga de processamento e do tempo de resposta.

3.1.2 Concorrência

Um sistema concorrente é aquele capaz da realização simultânea de várias tarefas (Selic et al., 1994) ou aquele que contém duas ou mais linhas de controle que dependem dinamicamente uma da outra para preencher seus objetivos individuais. Os sistemas embarcados, por serem sistemas de tempo real, devem operar satisfatoriamente em ambiente concorrente (Gajski et al., 1994).

3.1.3 Confiabilidade

É a probabilidade de um sistema operar sem falha em um dado ambiente, em um dado instante ou durante um período de tempo especificado (Calvez, 1993). Considera-se que uma falha ocorre quando o sistema não realiza a tarefa requerida ou esperada durante a operação (Strauss e Ebenau, 1994; Lannino et al., 1984; Crespo et al., 1997).

3.1.4 Desempenho

A característica reativa desses sistemas exige que eles respondam no tempo às excitações provocadas pelo ambiente (Kumar et al., 1996). A determinação de desempenho do sistema embarcado depende basicamente de dois fatores: o tempo de resposta e a taxa de transferência de dados (Pressman, 1995). O tempo de resposta é o intervalo de tempo no qual um sistema deve detectar um evento, interno ou externo, e responder com uma ação. A taxa de transferência de dados consiste na velocidade com que dados (seriais ou paralelos, digitais ou analógicos) devem se mover para dentro ou para fora do sistema. Os dois fatores mencionados também determinam a categoria do sistema quanto ao tipo de desempenho: sistemas de desempenho entre baixo e moderado têm requisitos severos de custo e de forma.

3.1.5 Forma

Constitui as exigências referentes a tamanho, peso e forma (Kumar et al., 1996). A forma está fortemente vinculada aos requisitos da missão e às necessidades operacionais.

3.1.6 Consumo

Consiste no conjunto de restrições referentes ao consumo e dissipação de energia. A dissipação de energia em componentes do sistema se deve à carga e descarga de capacitores durante o chaveamento de circuitos lógicos. Nos sistemas alimentados por baterias, o fator consumo tem grande impacto sobre o dimensionamento dessas baterias e, consequentemente, no tamanho, peso e custo desses sistemas (Gajski et al., 1994). De uma maneira geral, os sistemas embarcados requerem funções específicas para desligar e ligar automaticamente parte do equipamento durante os períodos de latência para economia de energia.

3.1.7 Condições Ambientais

O computador de bordo deve ser projetado para operar em condições ambientais adversas, uma vez que estará sujeito a condições ambientais extremas com grande variação de temperatura, pressão, presença de radiação e interferência eletromagnética dos outros sistemas de bordo (Alonso, 1985).

3.1.8 Manutenção

Os sistemas embarcados requerem longos períodos de operação sem qualquer tipo de manutenção. Ao contrário de outras aplicações, esses

sistemas devem controlar o seu próprio ambiente, incluindo o consumo de energia, variação de temperatura e estabilidade (Pradhan, 1995). Portanto, para este tipo de sistema deve-se incluir requisitos de tolerância à falha, onde serão consideradas as falhas de hardware e de software, e definidas regras e mecanismos capazes de evitar que uma falha cause danos ao equipamento que está sendo controlado (Pham, 1995). As técnicas de redundância e tolerância a falhas geralmente empregadas para o software embarcado são os testes automáticos de paridade do hardware, testes das instruções ilegais, falhas de endereçamento a memória, erro de *checksum*, tempo de atuação do cão de guarda (*watch dog*), inclusão de uma tarefa específica, proteção da memória para escrita, remontagem e recarregamento da memória para mapear falhas. A correção de erros na operação pode ser automática ou por controle de solo via telecomando, quando se faz necessário o carregamento de programas ou parâmetros.

3.2 Funções do Software Embarcado em Satélite

O software embarcado consiste no software existente no computador de bordo, e que está incluído na classe de software para aplicação espacial. É executado, na maioria das vezes em computadores dedicados, construídos para uma aplicação específica. Ele realiza funções de aquisição de dados e controle dos demais sistemas de bordo. Além disto, o software deve fornecer uma resposta específica para cada um dos comandos externos recebidos, valores monitorados e estado dos dispositivos ou eventos.

O software embarcado está dividido hierarquicamente em **sistema operacional** e **aplicativos**. As principais características dos sistemas operacionais para satélite são:

- Resposta em tempo real em que a base de tempo está associada à aplicação;

- Escalonamento de processos;
- Sincronização e comunicação entre processos;
- Gerenciamento de memória;
- Gerenciamento dos dispositivos de entrada e saída;
- Tratamento e relato de erros;
- Implementação de mecanismos de tolerância a falhas por software.

Os aplicativos típicos dos sistemas a bordo de satélite realizam as seguintes funções:

- Comunicação de dados a bordo;
- Gerenciamento dos modos de operação de falhas;
- Interpretação e execução de telecomandos;
- Aquisição e processamento de telemetria;
- Processamento de dados a bordo;
- Controle de armazenamento de dados;
- Partilhamento das funções do computador do sistema de controle de atitude.

De uma forma geral, as características relevantes do software embarcado são (Alonso, 1998):

- Complexidade (internamente) por ter interações com um ambiente complexo (externamente);

- Dedicação a uma aplicação específica, exigindo que o desenvolvimento siga padrões rigorosos;

- Utilização eficiente da capacidade de recursos do hardware (capacidade de processamento, memória e dispositivos E/S);

- Requisitos de análise e projetos especiais para seu desenvolvimento, que considerem seu efeito sobre a segurança do sistema em que está embutido.

O conceito de confiabilidade do software embarcado inclui ainda a probabilidade de não ocorrer, durante um período de exposição especificado, uma falta de software que possa causar um desvio de saída que exceda a tolerância definida para um ambiente especificado (Ramamoorthy e Bastani, 1982). A falta acidental ou falta está associada ao conceito de erro e conduzem à ocorrência de falha. Uma **falha** (*failure*) ocorre quando o comportamento do sistema desvia-se da especificação; um **erro** (*error*) ocorre quando parte do estado do sistema pode levar à falha e a uma **falta** (*fault*) é atribuída a suposta causa do erro. Como exemplo, suponha que uma partícula atinja o computador de um satélite. O bit de uma célula da memória fica preso em ‘0’ (falta). Um *erro* ocorre porque o valor esperado na célula é ‘1’. Uma falha acontece quando o valor escrito provoca um desvio da especificação do sistema (percepção do usuário).

A ocorrência de falha pode resultar, desta forma, em enormes prejuízos econômicos, visto que pode alcançar o nível de controle de processos críticos. Portanto, devido a esta característica do software embarcado, devem ser identificadas e avaliadas a possibilidade de ocorrência de cada um dos eventos de risco. Se estes eventos puderem ser prontamente identificados no processo de desenvolvimento de software, é possível especificar características de projeto de software que controlem ou eliminem esses riscos. Padrões internacionais para o desenvolvimento de software para aplicação

espacial recomendam que estes eventos sejam classificados por categorias de severidade e criticalidade do efeito da falha, que pode resultar até em perda da missão. O risco (*risk*) também é definido como função da possível freqüência de ocorrência do evento indesejável, da severidade potencial das conseqüências resultantes (NASA, 1996).

Padrões internacionais recomendam ainda que a especificação deva conter os requisitos de segurança, ou seja, a elaboração de uma lista de eventos indesejáveis e as respectivas respostas esperadas do sistema. Entretanto, técnicas de análise específicas para prever a cadeia de probabilidade de que cada um dos eventos, que possam causar riscos, ocorra (Alonso, 1998) não serão discutidas neste estudo.

3.3 Padrões para o Desenvolvimento do Software Embarcado

Diversas agências espaciais, instituições acadêmicas, organizações internacionais e nacionais vêm, desde a década de 80, estabelecendo padrões de engenharia de software para as aplicações espaciais, que englobam o software embarcado. Entre os existentes, serão citados apenas aspectos relevantes dos padrões elaborados pela Agência Espacial Européia (ESA) e Agência Nacional de Aeronáutica e Espaço (NASA) utilizados, além do padrão do INPE, como referências principais para o desenvolvimento do software embarcado no INPE.

3.3.1 Padrões de Engenharia de Software da ESA

Em 1983, a ESA apresentou um modelo inicial de padronização do desenvolvimento de software com a elaboração do documento "Padrões de Engenharia de Software da ESA" (ESA-PSS-05-0) (ESA, 1983). O documento descrevia as atividades de desenvolvimento de modo qualitativo e não

abordava aspectos relacionados à segurança de software para aplicações espaciais.

Mais tarde, a ESA criou um programa de qualidade de software com a finalidade de fazer com que o processo de desenvolvimento de software (interno ou contratado) de seus sistemas atingisse o nível exigido pelas aplicações espaciais que desempenham funções críticas. Foi então elaborado um documento nomeado "Garantia do produto para sistemas espaciais da ESA" (ESA, 1988), que consistia em uma coletânea de documentos relacionados à confiabilidade, mantenabilidade e segurança. A seguir, é feita uma breve descrição dos principais documentos de análise e especificação dos requisitos da ESA específicos para as aplicações espaciais.

O documento de confiabilidade de software (PSS-01-230) (ESA, 1989d) recomenda que a especificação de requisitos de software deve estabelecer quais as técnicas de análise a serem empregadas para prever a cadeia de probabilidade de que cada um dos eventos que possam causar riscos ocorra (Alonso, 1998). Além das técnicas de análise de riscos, a especificação de requisito deve determinar as categorias de severidade de efeito da falha, que pode resultar desde a degradação do sistema até a perda da missão.

O documento de mantenabilidade de software (PSS-01-240) (ESA, 1989a) refere-se ao cumprimento da especificação e análise dos requisitos de mantenabilidade de software para equipamentos espaciais desenvolvidos para a ESA. Mantenabilidade de software é considerada a medida da facilidade e rapidez com que o software pode restabelecer o estado operacional depois de uma falha.

O documento de segurança de software (PSS-01-250) (ESA, 1989e) é baseado na especificação de uma metodologia de desenvolvimento de software que assegure que um nível aceitável de segurança do software seja conseguido.

Por fim, o documento de avaliação de confiabilidade de software (PSS-01-231) (ESA, 1989c), reforça boas práticas de projeto, que são os princípios de engenharia de software para produzir uma estrutura de software mais confiável. O documento enfatiza o uso das técnicas para a atividade de projeto de software recomendadas pela ESA, tais como: modularidade, baixo acoplamento, alta coesão e ocultação da informação, uso de linguagens de alto nível e as técnicas estruturadas no tratamento de exceção.

O documento também considera a revisão como a principal técnica recomendada para avaliação da confiabilidade de software. As revisões devem estar baseadas em listas de verificações associadas, que visam avaliar quantitativamente a confiabilidade do software.

3.3.2 Padrão de Segurança de Software da NASA

O documento "Padrão de Segurança de Software da NASA" (NSS 1740.13) (NASA, 1996) propõe uma estrutura para a garantia de segurança de software para os programas da NASA. O documento descreve as atividades necessárias para assegurar que a segurança seja incorporada ao projeto de software, desenvolvido ou adquirido pela NASA. O padrão estabelece que os requisitos de segurança devem estar relacionados com os itens definidos conforme os princípios de engenharia de software para cada atividade, como revisões e documentos específicos. Este documento é um guia suficientemente detalhado para ser utilizado como referência na escolha dos métodos, técnicas, ferramentas e até da linguagem de programação utilizados em projetos para aplicação espacial.

3.3.3 Padrões Unificados

A Cooperação Européia para Padronização Espacial - *European Cooperation for Space Standardization* (ECSS) é uma iniciativa estabelecida para desenvolver um conjunto simples e coerente de padrões amigáveis para uso em todas as atividades espaciais européias (Kriedte e El Gammal, 1995). Em 1993, várias agências espaciais nacionais européias, ESA e representantes da indústria espacial européia juntaram forças para iniciar o desenvolvimento de padrões espaciais.

O painel técnico da ECSS tem ligações com duas diferentes corporações de padronização: a Organização Internacional para Padronização (ISO) e o Comité Europeu de Normalização (CEN).

Para o ECSS, a padronização pode acontecer em vários níveis: internacional, regional, nacional e da companhia. Entretanto, é enfatizado que a padronização: somente funcionará de forma eficaz se for seguida em toda a estrutura da organização em que for aplicada.

O Padrão da ECSS consiste em uma coletânea de documentos divididos em três categorias: "Garantia do Produto Espacial", “Engenharia Espacial” e “Gerenciamento de Projeto Espacial”. Na categoria de garantia do produto, os documentos de “Garantia de Qualidade” (ECSS-Q-20A) e “Garantia do Produto de Software” (ECSS-Q-80A), consideram as revisões baseadas em listas de verificação, como a principal técnica recomendada para avaliação da confiabilidade de software. Esta categoria também inclui dois documentos específicos para o software que desempenha funções críticas: O documento “Dependabilidade” (ECSS-Q-30A), que apresenta técnicas para a prevenção de ocorrência de eventos que possam causar riscos e mantenabilidade de software e o documento “Segurança” (ECSS-Q-40A), que recomenda a realização de revisões de segurança baseadas em listas de verificação de

segurança, que devem ser incorporadas a todas as atividades do ciclo de vida do software para que seja obtida a segurança de software (Kriedte, 1996).

3.3.4 Padrão de Desenvolvimento de Software do INPE

O padrão da instituição é o relatório técnico interno nomeado Padronização de Desenvolvimento de Software (0liveira, 1985), resultado de um esforço conjunto de um grupo de trabalho integrado por pesquisadores ligados às diversas áreas do INPE envolvidas com o desenvolvimento de software. O padrão visa estabelecer normas para o desenvolvimento de software para o Centro de Controle de Satélite (CCS) e teve como base o padrão da ESA (ESA, 1983). O padrão INPE adota o modelo de ciclo de vida em cascata realimentado, que compreende seis atividades:

- Definição dos Requisitos de Software*:* elaboração de requisitos específicos do software independente de detalhes de implementação e deixando-o aberto a várias arquiteturas alternativas;

- Projeto Preliminar*:* projeto da arquitetura geral do software levando-se em consideração os requisitos da atividade anterior e elaboração do plano de implementação detalhado;

- Projeto Detalhado*:* descrição detalhada do projeto do software baseada na arquitetura definida na atividade anterior e elaboração das versões preliminares do Manual do Usuário e do Plano de Teste de Software;

- Codificação e Testes Unitários*:* envolve a codificação e depuração dos módulos do projeto detalhado e atualização dos documentos anteriores;

- Integração e Testes*:* integração do software no sistema em que está embutido, verificação da qualidade do software e preparação de sua transferência para a atividade de operação e manutenção;

- Operação e Manutenção*:* é a atividade que se segue a aceitação do software onde normalmente são feitas as operações de rotina e, excepcionalmente, o software pode ser modificado seja para corrigir erros, bem como para inserir novas funções para suprir novos requisitos.

Para cada uma destas atividades, indica-se a elaboração dos seguintes produtos de software: documento de especificação, documento de projeto preliminar, documento de projeto detalhado, codificação, plano de teste e procedimento de teste. Para cada um destes documentos o padrão descreve **o que** deve estar contido, mas não menciona **como** deve ser feito. Ele estabelece a necessidade de haver distinção entre os requisitos de software essenciais dos menos importantes; recomenda a utilização de métodos de análise e projeto estruturados, que é também recomendado por todos os padrões para desenvolvimento de software crítico, bem como define apenas qualitativamente o papel da garantia de qualidade de software.

Por ter sido elaborada há mais de uma década, esta padronização não está voltada ao software que desempenha funções críticas para aplicações espaciais. O padrão não menciona ou recomenda procedimentos específicos relacionados aos fatores de segurança, mantenabilidade ou confiabilidade associada à ocorrência de falhas por eventos indesejáveis, existentes em padrões internacionais como NASA, ESA e ECSS.

Uma proposta de alteração do documento de padronização do INPE, que inclui requisitos críticos, foi apresentada sob a forma de padronização de software crítico para aplicações espaciais (Alonso, 1998). Até o momento, no entanto, não foi oficializada qualquer alteração no documento padrão para desenvolvimento de software no INPE.

3.4 Desenvolvimento do Software Embarcado no INPE

O desenvolvimento de computadores de bordo em projetos de satélite no INPE vem sendo realizado pelo Grupo de Supervisão de Bordo (SUBORD), responsável por diversos projetos para aplicação espacial tais como: o desenvolvimento do sistema operacional e aplicativos dos Satélites de Coleta de dados (SCD) 1 e 2; desenvolvimento do projeto do Satélite Sino-Brasileiro de Recursos Terrestres - *China-Brazil Earth Resources Satellite* (CBERS) 1 e 2, o desenvolvimento de aplicativos para os Satélites Científicos (SACI) 1 e 2. A equipe SUBORD é formada por 4 membros, sendo 2 engenheiros eletrônicos e 2 bacharéis em computação. Um dos bacharéis e um dos engenheiros possuem mestrado em computação, com 15 anos de experiência em software de bordo, possuindo bastante experiência na aplicação, mas não são especialistas em engenharia de software. Para os aspectos relacionados ao hardware, é atribuído ao engenheiro mais experiente o cargo de engenheiro do sistema, responsável pela gestão de bordo. A ele são atribuídas a função de gerenciamento e a função técnica de especificação de requisitos do sistema embarcado. Para o membro da equipe com bastante experiência em software embarcado, é atribuído o cargo de projetista de software, responsável pela especificação de requisitos e projeto do software. Aos 2 membros menos experientes quanto à aplicação são atribuídos os cargos de analista-programador, responsáveis pelo projeto detalhado, codificação, testes unitários e testes de integração dos módulos.

A equipe trabalha há anos praticamente da mesma maneira, sem grandes avanços na melhoria de qualidade. Ela segue o padrão da instituição para desenvolvimento de software, apresentando para cada projeto os seguintes documentos: especificação de requisitos, projeto preliminar, projeto detalhado, codificação, plano e procedimento de teste. Porém, os documentos ainda são confeccionados manualmente. Estes documentos são encaminhados para o Grupo de Garantia de Produto do INPE (GGP) que, apesar de responsável pelo controle dos documentos, não realiza uma verificação em cada um dos

produtos entregues e, ainda, nem sempre formaliza os procedimentos de documentação das alterações. Atualmente, não existe um responsável pelo estabelecimento de padrões.

A proposta do INPE também não indica uma padronização para os documentos. Com relação à metodologia, por exemplo, somente sugere o projeto estruturado, sem abordar um modelo conceitual. Em função disto, a metodologia utilizada não inclui as técnicas de análise e projeto estruturados recomendadas por todos os padrões para desenvolvimento de software crítico. Sendo assim, os padrões não são suficientes para a garantia de qualidade e podem ser melhorados.

Quando cada atividade do desenvolvimento é considerada concluída, são realizadas revisões formais. No entanto, não existe definição de como cada revisão deve ser feita. As revisões não são realizadas por profissionais que possuem conhecimento no desenvolvimento de software, contando apenas com os profissionais da área de satélite. Durante uma revisão, um produto também não pode ser examinado com critérios por não haver padronização, o que demonstra não existir uma preocupação ou providências explícitas voltadas à reutilização. Portanto, o papel da garantia de qualidade de software não é cumprido.

Padrões como ESA, NASA e ECSS são referências de qualidade que poderiam ser utilizados em outro contexto, entretanto a equipe de desenvolvimento precisaria estar em um nível de preparação adequada para utilizar estes padrões e tentativas anteriores não resultaram em sucesso (Alonso, 1998), provavelmente porque tentou-se suprir esta diferença em um único passo. Portanto, muita coisa pode ainda ser feita para melhorar a garantia dos softwares embarcados nos satélites do INPE, de forma a incorporar as características desejáveis para esse tipo de software descritas neste capítulo.

CAPÍTULO 4

PROPOSTA DE EVOLUÇÃO DA GARANTIA DE QUALIDADE

Neste Capítulo discute-se um passo a mais para melhoria da qualidade no processo de desenvolvimento de software embarcado em satélite no Instituto Nacional de Pesquisas Espaciais (INPE). A ênfase está na obtenção de um produto de software confiável e que não pode apresentar erros. Para isto, o enfoque passa a estar na garantia de qualidade dos produtos gerados durante cada atividade do processo de desenvolvimento para, assim, obter-se o código do programa (produto final) sem erros.

Em função do perfil da equipe o passo deve ser viável, para que seja aceito e seguido. Para isto supôs-se que atuar sobre o produto surtiria mais efeito para garantir a confiabilidade do software. Portanto, o foco passa a ser o **produto** enquanto a preocupação em garantir o **processo** de desenvolvimento torna-se viável em um estágio seguinte, evitando assim, que o passo seja muito grande. Por isso, foi estabelecido um processo mais próximo do que já vem sendo feito no INPE, exceto pela novidade de incluir o método de projeto voltado ao sistema de tempo real, que propõe a quebra do sistema em tarefas ou processos computacionais (vide Capítulo 2).

Portanto, propõe-se um conjunto de atividades de desenvolvimento com certo encadeamento, instanciadas para o desenvolvimento do software embarcado no INPE. As atividades propostas para melhoria da qualidade abrangem dois aspectos: o processo de desenvolvimento propriamente dito, que inclui uma metodologia, a continuidade no uso de técnicas tradicionais e um esquema de garantia de qualidade baseado nos produtos, em que são enfatizadas as características necessárias do tipo de sistema alvo.

Propõe-se uma estratégia ou passo possível de ser seguido pela equipe, para isto, o processo de desenvolvimento está próximo ao que vem sendo utilizado pela equipe do INPE e a metodologia não poderia ser orientada a objeto como por exemplo COMET (vide Capítulo 2), visto que envolve uma mudança de paradigma.

Os produtos, ou seja, as unidades de software, tais como: documentos, modelos, rotinas e arquivos, dependem de uma metodologia que, no contexto desta proposta, serve para mostrar o esquema de garantia podendo, portanto, ser alterada. Tanto a metodologia quanto o esquema de garantia de qualidade, conforme ilustra a Figura 4.1, estão sendo propostos considerando o perfil, o grau de conhecimento da equipe, a complexidade do software em questão e as condições ambientais de projeto existentes no INPE.

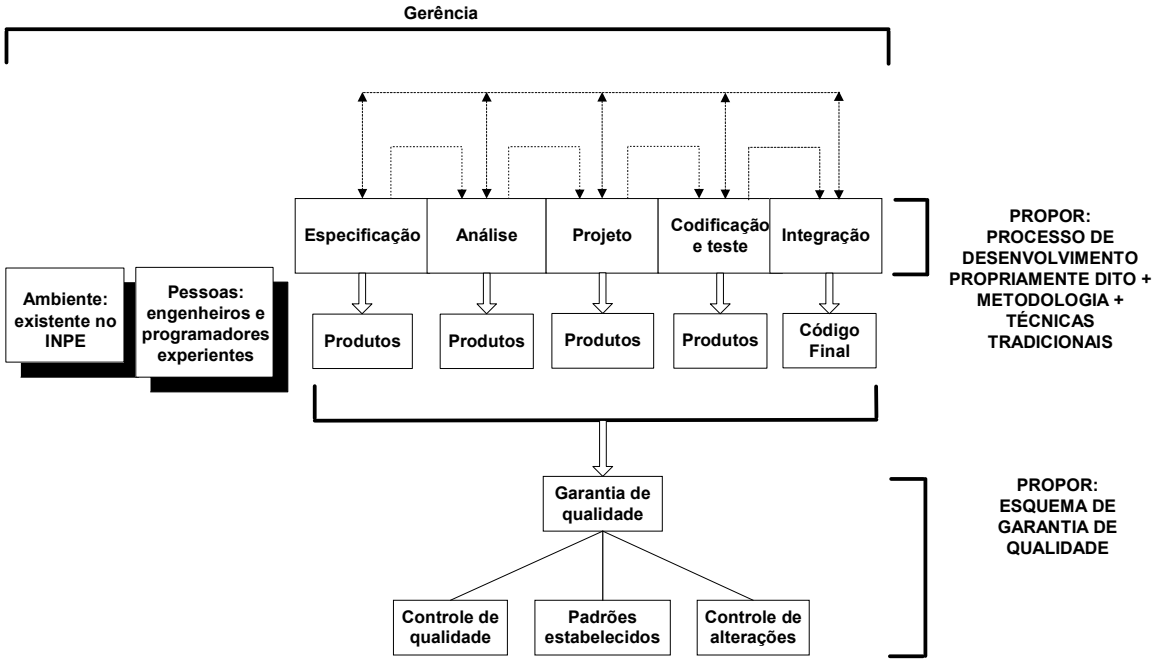

FIGURA 4.1 - Proposta para melhoria do software embarcado em satélite do INPE.

Embora neste passo a preocupação seja com os resultados, além do objetivo principal de garantir a qualidade do software embarcado, um outro objetivo

contido nesta proposta é relevante: privilegiar o reuso, tanto de documentos quanto de software.

A seguir é apresentada uma descrição detalhada do que está sendo proposto, em que estão incluídas as justificativas baseadas nos conceitos de engenharia de software e adequadas ao caso específico do INPE.

4.1 Processo de Desenvolvimento Propriamente Dito

O modelo de ciclo de vida adotado pelo padrão do INPE é o tradicional, em cascata. Embora outros modelos de ciclo de vida para software de tempo real também sejam recomendados por especialistas da área, tais como: prototipação rápida, desenvolvimento incremental e prototipação evolucionária (Gomaa, 1986a).

O passo proposto para a melhoria de qualidade baseia-se em um processo de desenvolvimento que vem sendo utilizado pela equipe do INPE, com o acréscimo de um conjunto de atividades para viabilizar a sua utilização. O modelo do ciclo de vida em cascata apresenta vantagens, tais como: as atividades e os produtos são mais facilmente identificáveis, o controle do retrabalho é mais fácil se comparado, por exemplo, ao modelo em espiral, além de facilitar a gerência da qualidade. Estas vantagens o tornam o modelo mais adequado para que o esquema de garantia de qualidade proposto tenha maior chance de ser adotado pela equipe.

4.1.1 Especificação dos Requisitos de Sistema

Para a atividade de especificação de requisitos do software, um documento textual vem sendo produzido pelo grupo de desenvolvimento de software de bordo. Embora a equipe já trabalhe com um documento textual, o que está

sendo proposto é mais rigor com o controle da evolução do documento em função da sua importância. Garantir a qualidade do documento gerado nesta atividade é garantir a base para a construção dos produtos das demais atividades. Sabe-se que requisitos são fundamentais para se construir e melhorar a qualidade de software, pois não se pode "avançar em qualidade" tratando requisitos como algo sem importância. Em função desse grau de importância, existe uma grande preocupação com o documento de especificação de requisitos, principalmente por ser fundamental o trabalho multidisciplinar, que pode torna-se difícil por envolver o trabalho conjunto de grupos formado por pessoas de diferentes áreas e experiências para produzir o resultado desejado.

Propõe-se a construção de um texto descritivo, contendo o propósito do sistema, uma descrição de como ele deve funcionar, seus principais objetivos, funções e metas, as informações de entrada, assim como as informações de saída desejadas. Deve ainda ser descrito um pouco do ambiente no qual o sistema deve operar, e com quais outros sistemas ele irá interagir. Portanto, a proposta é apresentar um modelo de documento textual contendo uma declaração completa sobre o sistema, sem que exista ambigüidade, imprecisão, inconsistência ou detalhes sobre as formas dos dados, a tecnologia utilizada para implantá-lo, as limitações de hardware e software básico. Enfim, de forma semelhante ao que existe na padronização do INPE, o documento textual deve atender ao conceito de **definição dos requisitos de software**, que consiste na elaboração de requisitos específicos do software independente de detalhes de implementação e deixando-o aberto a várias arquiteturas alternativas.

4.1.2 Análise

A equipe de supervisão de bordo segue a padronização para o desenvolvimento de software do INPE (vide Capítulo 3), que não inclui formalmente esta atividade passando da especificação de requisitos de software para a atividade de projeto preliminar ou arquitetural. A representação gráfica que vem sendo utilizada pela equipe baseia-se na principal ferramenta da análise estruturada, o Diagrama de Fluxo de Dados. Entretanto, o DFD é apresentado na atividade de projeto, contendo informações que não devem estar representadas no modelo conceitual ou lógico definido para esta atividade, pois além dos fluxos de dados ou informações, são representados dados de controle.

Propõe-se incluir uma atividade para a construção do modelo conceitual do sistema de software embarcado através da formalização das informações obtidas no modelo descritivo. Ou seja, para esta atividade de análise de requisitos do software, propõe-se à construção do modelo funcional do sistema, onde estão retratados os fluxos de informação, as divisões funcionais do sistema, os depósitos de dados e as entidades externas. Não serão representadas informações sobre detalhes físicos e fluxos de controle.

Portanto, de forma semelhante ao que já vem sendo feito, o Diagrama de Fluxo de Dados (DFD) permanece como a ferramenta gráfica utilizada, no entanto, somente nesta atividade de análise de requisitos, onde se dá maior ênfase à informação propriamente dita. A modelagem baseada em DFDs é uma técnica recomendada considerando-se que o sistema embarcado em satélites é mais voltado ao processamento do que no armazenamento de dados. Além disto, o DFD constitui uma ferramenta interessante para servir de base para a atividade de projeto. Como complementação ao DFD são criados ainda um dicionário de dados e os documentos que descrevem cada um dos processos.

4.1.3 Projeto

Para esta atividade, propõe-se a utilização do método de projeto estruturado ou orientado ao fluxo de dados acrescido de uma novidade: a decomposição do software embarcado em tarefas ou processos computacionais concorrentes. A quebra do sistema em tarefas oferece diversas vantagens, entre elas: divisão em subsistemas menores, cujo desenvolvimento é mais simplificado, facilidade na divisão de trabalho pelos membros da equipe, simplificação dos produtos gerados. Com isso ele torna-se um método de projeto que permite um passo a mais na garantia de eliminar erros.

A partir do diagrama de fluxo de dados construído na atividade de análise, o sistema é dividido em tarefas, conforme os critérios definidos pelo método de projeto DARTS (Gomaa, 1984), voltado às aplicações de tempo real (vide Capítulo 2), e que também proporciona mecanismos para o manuseio das informações entre tarefas. Cria-se, então o diagrama de tarefas, onde cada tarefa representa um programa a ser executado e que, por sua vez, deve ser decomposto em módulos, representados pelo diagrama de estrutura de módulos.

O método de projeto utilizado é orientado ao fluxo de dados ou projeto estruturado, conforme recomenda o padrão de desenvolvimento do INPE e tem sua origem em conceitos baseados na modularidade, projeto *top-down* e programação estruturada. O conceito inicial da abordagem orientada ao fluxo de dados ampliou-se ao incorporar o fluxo de dados também a atividade de projeto do sistema de software quando diferentes funções, que transformam cada fluxo foram definidas como estrutura de programa. A partir daí, o projeto orientado ao fluxo de dados tornou-se propício a ser empregado no desenvolvimento de diversos tipos de aplicação. De fato, uma vez que todo software pode ser representado por DFD, um método de projeto que faz uso desse diagrama poderia, teoricamente, ser aplicado. a abordagem, no entanto, mostra-se adequada apenas quando as informações são processadas

seqüencialmente. Sendo assim, aplicações de tempo real não estariam incluídas nesta categoria de processamento. Então, Hassan Gomaa (Gomaa, 1984) propôs de um método de projeto de software voltado as aplicações de tempo real, que incluía extensões ao projeto orientado ao fluxo de dados, isto é, mecanismos para o projeto de software de aplicações baseadas em interrupções. O método DARTS apresenta um conjunto de critérios para determinar se as funções do DFD devem ser definidas como tarefas separadas ou agrupadas com outras funções em uma única tarefa, além de implementar mecanismos de interface entre as tarefas (vide Capítulo 2). Uma tarefa representa um subsistema e pode ser definida como um processo computacional, que pode comunicar-se ou sincronizar-se com outros processos. Cada processo computacional é um programa em execução, ou seja, uma entidade capaz de causar o acontecimento de eventos. Cada programa é decomposto, então, em módulos independentes, que apresentam alta coesão e baixo acoplamento entre eles.

Após a divisão do sistema em tarefas, um plano de codificação e testes é definido, ou seja, estabelece-se uma estratégia para codificar e testar a partir do diagrama de tarefas, cada tarefa individualmente, até que esteja garantido seu funcionamento como uma unidade.

4.1.4 Atividade de Codificação e Testes Unitários

Propõe-se uma padronização para o código. Os testes unitários baseiam-se na prévia concepção dos casos de teste. À medida que forem sendo codificadas e testadas, as tarefas devem ser integradas para formarem um pacote de software. Finalizada esta etapa, deve-se realizar a integração do software com outros elementos do sistema como o hardware e outros sistemas.

4.1.5 Atividade de Integração e Testes

Os testes de integração são planejados e executados com base na prévia concepção dos casos de teste. Durante a integração das tarefas pode se fazer necessário testar o software como um todo antes mesmo de estar completo, para certificar-se de que a comunicação e o sincronismo entre as tarefas estão sendo realizados conforme previsto.

4.2 Garantia de Qualidade de Software

Cada um dos produtos de software gerado pela equipe de supervisão de bordo é repassado para outra equipe responsável pelo controle de documentos do INPE chamado Grupo de Garantia de Produto (GGP) do INPE. O GGP também é responsável pelo controle de qualidade, dos produtos modificados e de versões (vide Capítulo 3). Além do grupo de supervisão de bordo não ter controle sobre a garantia de qualidade de cada um dos produtos, aqueles que são documentos são construídos manualmente, ou seja, não se faz uso de uma ferramenta de documentação que permite reduzir em até 30% todo o esforço de desenvolvimento do software gasto em documentação. Portanto, na prática, devido às restrições de prazo, a documentação do software muitas vezes segue a engenharia de software reversa.

Com base neste quadro, propõe-se um esquema de controle de qualidade de software a ser realizado por membros da própria equipe de supervisão de bordo. Além de se considerar o estágio de conhecimento da equipe e o ambiente de projeto do INPE, o esquema está fundamentado no conceito de garantia de qualidade de software, que consiste em um padrão sistemático e planejado de ações exigidas para garantir a qualidade de software (Schulmeyer e McManus, 1987). Estas ações compreendem uma variedade de tarefas associadas a sete grandes atividades (Pressman, 1995):

- Aplicação de métodos técnicos;
- Realização de revisões técnicas formais;
- Atividades de testes de software;
- Aplicação de padrões;
- Controle de alterações;
- Medição;
- Manutenção de registro e reportagem.

Entretanto, por ser este um passo inicial para a melhoria do software embarcado em satélite do INPE, que visa garantir a qualidade do código final a partir da garantia de qualidade do produto de software, o esquema de controle de qualidade é feito para cada produto gerado ao longo do desenvolvimento. Por isto, das sete atividades mencionadas, somente três atividades fazem parte do esquema por estarem relacionadas com o produto de software: a realização de revisões técnicas como atividade básica de controle de qualidade, o controle de alterações e a aplicação de padrões. A seguir é apresentado apenas em linhas gerais o que se propõe para estas três atividades, consideradas chaves para a melhoria da qualidade. Os detalhes serão abordados durante a discussão sobre o esquema de controle de qualidade (vide Capítulo 6).

4.2.1 Controle de Qualidade

As revisões formais vêm sendo realizadas ao final de cada atividade e os participantes dependem das equipes envolvidas, podendo contar ou não com profissionais experientes externos ao projeto.

Propõe-se que o controle de qualidade, de forma semelhante ao que vem ocorrendo, seja também baseado em revisões, visto que a revisão técnica é uma atividade eficiente para verificação e validação dos produtos. Propõe-se uma seqüência de procedimentos para a realização de cada revisão em função dos produtos gerados em cada atividade de desenvolvimento. É apresentada uma descrição detalhada das ações necessárias para a realização de uma reunião de revisão a serem cumpridas pelos participantes, bem como uma descrição de itens necessários para uma lista de conferência (*checklist*), para avaliação de cada produto.

Propõe-se, também que as revisões técnicas ocorram toda vez que um produto for entregue durante cada atividade do processo de desenvolvimento e que sejam baseadas em apresentações, ou seja, na apresentação do produtor ou eventualmente de um responsável pelo produto. As apresentações acontecem durante o principal evento de uma revisão: a reunião.

Propõem-se, portanto, procedimentos que antecedem a realização de uma reunião de revisão, bem como durante e após a finalização da mesma. Enfim, diretrizes de como uma reunião deve ser conduzida, onde também estão incluídos **papéis** para os participantes experientes quanto à aplicação. Dependendo do produto, é selecionado o líder de revisão que pode ou não corresponde à figura do mediador ou moderador, controlando a reunião e evitando as dispersões. Deve ainda existir a figura de um secretário para anotar os principais pontos, bem como registrar o resultado no relatório de revisão. Alguns participantes podem cumprir ainda o papel de verificar se o produto está de acordo com os padrões estabelecidos.

Outro aspecto abordado refere-se ao ponto de partida para avaliação de cada produto a ser revisado, ou seja, um conjunto de itens básicos, ao qual o produto deve corresponder. Sendo assim, é apresentada uma lista de conferência ou lista de verificação (*checklist)*, definida para cada produto gerado em cada atividade de desenvolvimento do software embarcado.

Um *checklist* para verificar o documento de especificação de requisitos permite o controle da evolução do documento, fundamental para todo o desenvolvimento.

Um *checklist* para a atividade de projeto permite verificar a escolha de uma metodologia de projeto que proponha, por exemplo, decompor o problema em módulos pequenos para serem escritos em assembler ou C, com interfaces para outros módulos, para o sistema operacional e para o hardware. Este tipo de critério de decomposição em módulos produz um acréscimo no número de módulos e um acréscimo ainda maior no número de interfaces, causando um maior acoplamento dos componentes do software e aumentando sua complexidade, fazendo com que o custo das atividades de integração e testes cresça proporcionalmente. Neste caso conclui-se, que não se trata de uma metodologia adequada por não prover uma maneira de integrar os módulos em pacotes de nível mais alto que possam ser integrados independentemente e formar níveis mais altos de abstração, facilitando o entendimento da arquitetura do software.

Outro exemplo da importância de um *checklist* refere-se à avaliação do plano de teste, pois erros permanecem em decorrência da falta de cobertura dos testes.

4.2.2 Controle de Alterações

Procedimentos de documentação das alterações não são sempre formalizados. Portanto, as modificações no software, neste caso, têm potencial para introduzir erros ou criar efeitos colaterais que propagam erros.

O desenvolvimento de software, particularmente em seu estágio inicial, é um processo interativo, onde requisitos são bem analisados e caminhos alternativos considerados. Porém, sempre existe um momento em que um

produto é entregue, devendo ser encaminhado para uma revisão. Mediante a aprovação, a primeira versão do produto é obtida. Propõe-se que a partir desse momento, o produto passa a ser controlado e qualquer interação que se faça necessária deverá ser efetuada sob controle apropriado de alterações.

Propõe-se que o controle de alterações seja realizado conforme os seguintes procedimentos:

1) Estudo problema;

2) Impacto do problema (abrangência);

3) Proposta de soluções;

4) Quem vai realizar a alteração.

Ao ser efetuada a alteração em um produto controlado, o produto alterado passa a ser considerado um produto novo e por isso, deve ser submetido à nova revisão técnica.

4.2.3 Estabelecimento de Padrões

Atualmente, não existe na equipe um responsável por estabelecer normas e padrões nem para verificar se o produto corresponde a algum padrão.

O estabelecimento de padrões está diretamente relacionado ao controle de qualidade, visto que, sem a existência de um padrão para cada produto, a dificuldade para desempenhar as atividades de controle aumenta. Padrões têm de ser redigidos, revistos, distribuídos e atualizados para que possíveis defeitos dos padrões sejam removidos. Além disto, os desenvolvedores precisam receber treinamento e assistência ao uso de padrões.

Propõem-se modelos para o relatório de revisão, para os documentos de especificação, para cabeçalhos das rotinas, formatos de nomes lógicos para unidades de software e regras para entidades físicas, tais como estrutura de diretórios.

CAPÍTULO 5

METODOLOGIA PROPOSTA

O objetivo deste Capítulo é detalhar, através de exemplos, a metodologia proposta para o desenvolvimento do software embarcado. O sistema utilizado como exemplo, é um computador de bordo hipotético semelhante aos satélites já desenvolvidos pelo INPE, que realiza as funções de supervisão e controle de processos (vide Capítulo 3). A Figura 5.1, a seguir, descreve as principais funções de um computador de bordo e ilustra a comunicação deste com as aplicações existentes e os sistemas de suporte que compõem o satélite e a comunicação entre este segmento espacial e o segmento solo para projetos de satélite em geral.

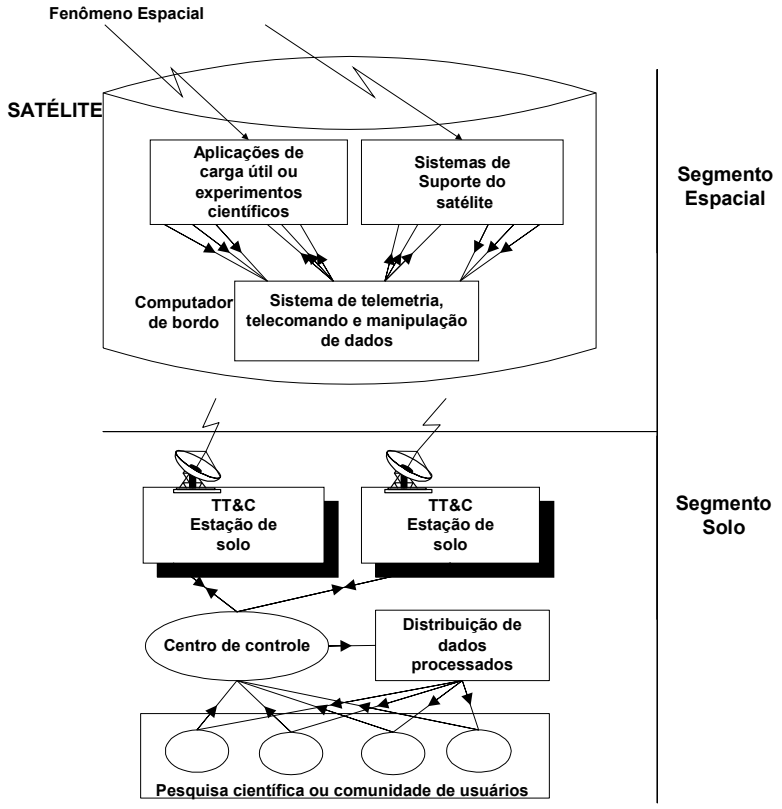

FIGURA 5.1 - Comunicação entre segmento solo e segmento espacial.
FONTE: Adaptada de Rowles e Howieson (1979, p. 470).

A metodologia aplicada no desenvolvimento do software de um computador de bordo genérico, baseia-se no modelo de ciclo de vida em cascata (vide Capítulo 4), para gerar os diversos produtos associados a cada fase e que servirão de base para mostrar o esquema de garantia de qualidade de acordo com as características desse tipo de sistema.

5.1 Especificação dos Requisitos do Sistema

A especificação de requisitos do sistema embarcado é a fase de estabelecimento dos requisitos para todos os elementos do sistema, seguida de uma definição do conjunto de requisitos para o software. Essa visão do sistema é essencial para a compreensão sobre os tipos de interface com o hardware que o software deve fazer e as restrições de desempenho associadas à missão, ambiente e as necessidades operacionais (vide Capítulo 3). Esta fase caracteriza-se pela coleta de informações sobre o sistema, que deve ser feita a partir de documentos relacionados ao projeto espacial, a missão espacial e de outras partes envolvidas, inclusive o segmento solo. São necessárias conversas entre outras equipes envolvidas no projeto e até consulta a documentação de projetos anteriores. Enfim, é a fase de definição do sistema que deve ser descrita em um texto descritivo, que servirá de base para a fase de análise de requisitos. Este texto descritivo deve conter as seguintes partes: requisitos funcionais, requisitos de desempenho, requisitos de interface, requisitos operacionais, requisitos de recursos, requisitos de verificação, requisitos de testes de aceitação, requisitos de documentação, requisitos de confiabilidade.

Os requisitos funcionais devem, por sua vez, conter: o propósito do sistema, a descrição das funções do sistema e das entidades, levantamento das interfaces, eventos e as reações do sistema.

O propósito deve ser uma declaração completa sobre a meta do sistema, não deve incluir detalhes sobre as formas dos dados ou sobre a tecnologia utilizada para implantá-lo. Em seguida, é descrito como o sistema deve funcionar, seus principais objetivos e metas (Coad, 1997), as informações que servirão de entrada assim como as informações de saída desejadas, bem como com **quem** ou com **o que** o sistema faz interface, as entidades envolvidas e os eventos relevantes, externos, que causam reação no sistema.

Uma técnica de modelagem pode ser utilizada para auxiliar no levantamento das interfaces: o diagrama de contexto. Não é necessário que seja feito, mas ajuda a fornecer dados básicos para a modelagem a ser construída na fase de análise. O diagrama deve ser construído a partir de um processo representativo (círculo) do sistema. Em seguida, devem ser listadas todas as entidades que fazem interface com o sistema, isto é, as entidades que são fontes ou receptoras de dados.

A título de exemplo, é feita a seguir uma descrição simplificada dos requisitos funcionais de um sistema do satélite, cujo objetivo é servir de base para a construção do texto descritivo.

O sistema identificado como computador de bordo tem como propósito receber e distribuir telecomandos vindos da estação de solo, enviar para estação de solo dados de telemetria adquiridos dos demais sistemas embarcados, bem como auxiliar nos cálculos necessários para controle de atitude do satélite.

O objetivo do sistema é receber telecomando e enviar telemetria para a estação solo realizando as operações de distribuição dos comandos e aquisição de dados dos demais sistemas.

Os telecomandos vindos da estação de solo obedecem a uma fila de prioridade para serem executados, atuando em cada um dos sistemas de forma imediata ou temporizada conforme critérios de urgência estabelecidos. Os dados adquiridos dos sistemas de bordo são divididos em dois tipos de

telemetria: de tempo real a ser constantemente transmitida e telemetria armazenada a ser transmitida para estação de solo somente durante o período de visada. Além destas funções, o sistema deve auxiliar no processamento do cálculo de atitude para o sistema de controle de atitude (AOCS) do satélite.

O sistema deve prover o controle sobre os diversos sistemas existentes realizando as tarefas de ajuste de data, reconfiguração do modo de operação dos sistemas, reinicialização e carregamento de parâmetros. Além disto, deve ainda monitorar e atuar no seu próprio estado de funcionamento através de mecanismos como relato de eventos e tratamento de exceções.

A entidade externa Estação de Solo envia telecomandos previamente definidos para o computador de bordo para serem executados de forma imediata ou temporizada, recebendo do mesmo, dados de telemetria armazenada e de tempo real provenientes dos demais sistemas embarcados.

As demais entidades externas envolvidas são: os Sistemas de suporte do satélite, as Aplicações de carga útil ou Experimentos científicos, onde cada um desses tem um formato próprio de telecomando e telemetria. Todos estes sistemas recebem, do computador de bordo, os respectivos telecomandos enviados pela estação de solo e enviam para o mesmo, dados de telemetria.

Ao serem recebidos pelo computador de bordo, os *frames* de telecomandos previamente definidos, enviados pela estação de solo são analisados e distribuídos conforme o formato para atuar nos sistema de suporte, carga útil ou experimentos e conforme uma fila de prioridades para serem executados de forma imediata ou temporizada. Os telecomandos são enviados somente durante o período de visada.

São enviados periodicamente, pelo computador de bordo, dados de telemetria adquiridos dos sistemas de suporte, carga útil ou experimentos para a estação de solo. Cada *frame* de telemetria contendo um formato próprio é armazenado em blocos para serem classificados, formatados e, posteriormente,

transmitidos para a estação de solo como *frames* de telemetria de tempo real, cujo formato é previamente definido. O envio periódico de *frames* de telemetria de tempo real permite o monitoramento de solo dos sistemas embarcados no satélite. Os blocos de telemetria também são compactados para, posteriormente, serem formatados como *frames* de telemetria armazenada e transmitidos para a estação de solo.

Periodicamente são realizadas tarefas de monitoramento, tais como: ajuste da data, coleta e registros de dados sobre o funcionamento dos sistemas embarcados e a ocorrência de eventos. Caso os dados sobre funcionamento de qualquer um dos sistemas embarcados indiquem problemas serão realizados testes de diagnóstico, tratamento de exceções e até uma reconfiguração do sistema.

O computador de bordo deve ainda receber *frames* de telemetria do sistema de controle de atitude (AOCS) contendo dados relacionados à atitude do satélite, tais como: efemérides e mudança de órbita para serem efetuados cálculos relacionados ao controle de atitude do satélite.

Para fechar este documento, outros requisitos como desempenho, operacionais e etc. deveriam ser mencionados, pois este documento inicial deve ser o mais completo possível. Porém, na prática, algumas informações somente estarão disponíveis ao longo do desenvolvimento do projeto do satélite como um todo. Portanto, existirão ainda outras versões do texto descritivo. De qualquer forma, esta fase de especificação de requisitos ou definição do sistema somente será considerada formalmente concluída, quando uma versão do texto descritivo completo, isto é, uma versão contendo tudo que é desejado for construída.

5.2 Análise Estruturada

Com base no documento de especificação de requisitos, ou seja, no texto descritivo do exemplo do computador de bordo genérico, é construído o modelo funcional do sistema: o Diagrama de Fluxo de Dados (DFD). O caminho mais simples utilizado para fazer um DFD foi começando pelos fluxos de dados partindo de um único processo que representou o sistema inteiro. Então, a partir da descrição dos requisitos funcionais do exemplo, uma versão do DFD de nível sistema é obtida, conforme mostra a Figura 5.2. O modelo funcional obtido retrata que o sistema embarcado em satélite é mais voltado ao processamento do que no armazenamento de dados, tornando o DFD uma técnica bastante adequada. O exemplo considera que existam os sistemas de suporte do satélite, as aplicações de carga útil ou experimentos científicos, porém para reduzir o tamanho do diagrama, será apenas representado o sistema de suporte de controle de atitude (AOCS). No diagrama são utilizados também **jargões** da área, tais como: *On board Data Handling* (OBDH) e *Housekeeping*. O OBDH refere-se à manipulação de dados de todos os sistemas a bordo do satélite, inclusive o próprio computador de bordo. O termo ***Status* do *OBDH*** mostra estado da operação e funcionamento de cada um dos sistemas a bordo. *Housekeeping* refere-se aos dados de operação e funcionamento de todos os sistemas a bordo.

Observa-se que, no DFD construído, muitas informações não são representadas, como por exemplo, “quando as coisas ocorrem”, “detalhes físicos” e nem detalhes sobre as estruturas dos dados. Os processos são considerados somente **processadores de dados** e os fluxos somente transportam **dados com valores**, não são apresentadas informações sobre fluxo de controle, isto é, sinais (*flags*) ou interrupções, bem como não apresenta informações sobre seqüência temporal. O DFD representa fluxo de dados e não fluxo de controle e deve ser visto como uma ferramenta para servir de base na fase de projeto. Aliás, o objetivo principal do DFD sistema é ainda mais amplo, ou seja, servir de base para o planejamento, pois tem-se

uma visão global do sistema, tornando-o um documento de referência para todo projeto.

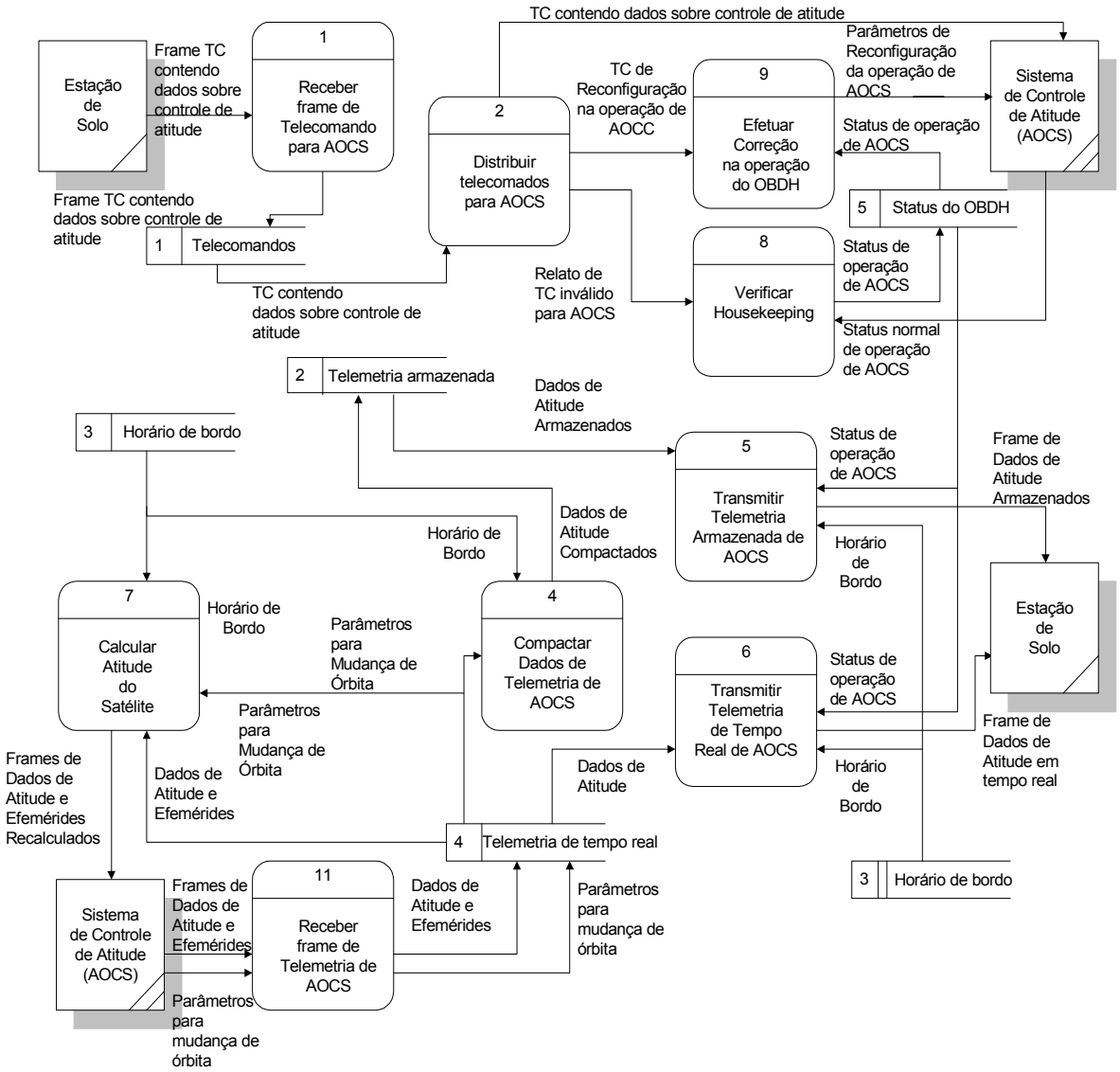

FIGURA 5.2 - Diagrama de fluxo de dados de um computador de bordo.

O próximo passo seria fazer a expansão de nível do DFD sistema. A expansão consiste em uma quebra que acrescenta detalhes, levando a um maior conhecimento sobre o sistema. Além de servir para validação do nível anterior, possibilitando um aumento na confiança no DFD inicial. Entretanto, é

importante considerar algumas regras para que esta quebra não torne o DFD complicado demais, são elas: a quebra em níveis e o balanceamento de fluxos. Para estabelecer outro nível, quebre o máximo que puder sem ficar um DFD muito complicado, em torno de 5 a 7 processos. Também deve ser avaliado que nem todos os processos precisam ser subdivididos até o mesmo nível de detalhamento. Na prática, para esse tipo de sistema, em que a complexidade não é muito extensa, uma quebra dos principais processos já será suficiente. O balanceamento de fluxos não consiste simplesmente em contar os fluxos, porque pode estar representado um fluxo complexo de dados no DFD de nível superior que precisa ser dividido em fluxos de dados mais simples de acordo com o detalhamento do processo em um nível inferior. Isto quer dizer que vários fluxos elementares podem estar reunidos para formar um fluxo contendo dados complexos e agregados representados no nível superior. Portanto, o balanceamento ocorre quando os fluxos de dados que entram e saem de um processo devem corresponder aos fluxos que entram e saem de uma figura inteira do nível imediatamente inferior que descreve aquele processo.

Para o tipo de sistema que está sendo representado, a expansão será necessária em um ou mais processos do DFD sistema relacionados as condições de erro e tratamento de exceção. As condições de erro, que são desvios devem ser tratadas dentro do diagrama de segundo nível de onde se originaram. No exemplo do computador de bordo, o DFD nível sistema contém o processo Efetuar Correção na Operação de AOCS, que se enquadra em um caso típico de tratamento de exceções, os fluxos de dados resultantes da manipulação de exceções não aparecem no DFD de nível sistema. Neste caso, a expansão permite detalhar a lógica decorrente dos desvios possíveis no caso da operação de AOCS demonstrar sinais de falha. O DFD de nível 1 originado desta expansão do processo DFD Efetuar Correção na Operação de AOCS está apresentado na Figura 5.3.

O tratamento apresentado para o caso do sistema de controle de atitude se estende a qualquer outro sistema embarcado que, por exemplo, apresente

problemas de endereçamento da memória. Os procedimentos previstos seriam desde a execução de um programa para realizar testes até a execução de um programa de reconfiguração do programa de operação. Estas possibilidades não estavam retratadas para não comprometer os objetivos do DFD sistema.

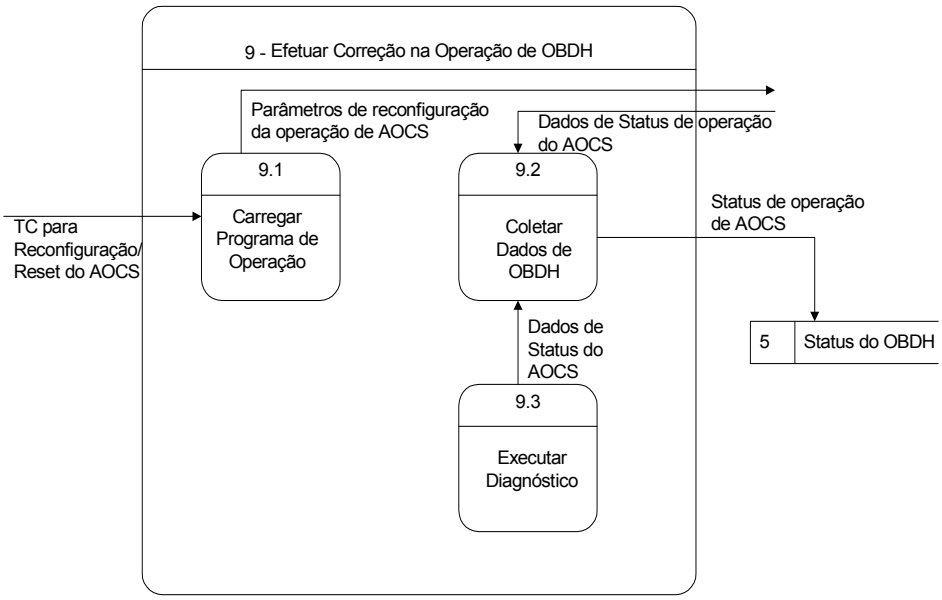

FIGURA 5.3 - Expansão do processo efetuar correção na operação de AOCS.

Embora o DFD ofereça uma visão geral dos principais componentes funcionais do sistema, não fornece detalhes sobre esses componentes. Para mostrar os detalhes de qual ou como a informação é transformada, é necessário incluir duas ferramentas de suporte textual: o dicionário de dados e uma especificação de cada processo. O dicionário de dados contém a descrição dos fluxos, estrutura de dados e processos enquanto a especificação ou mini-especificação descreve apenas as entradas e saídas existentes e um resumo da lógica envolvida no processo. Um modelo de formulário de especificação de processos é apresentado no apêndice A.

5.3 Projeto

Identificadas todas as funções existentes no sistema e os fluxos entre elas, o próximo passo do método de projeto DARTS (Gomaa, 1984; Gomaa, 1986b) é a determinação de como as tarefas concorrentes serão identificadas no diagrama de fluxo de dados.

A principal consideração na quebra em tarefas concorrentes é natureza assíncrona das funções dentro do sistema. Os processos do DFD são analisados para que sejam identificados quais deles devem ser executados seqüencialmente, e quais devem ser executados concorrentemente. Esta análise se baseia em critérios para definir se um processo DFD corresponde a uma tarefa ou deve ser agrupado com outros processos para formarem juntos uma única tarefa (vide Capítulo 2). Portanto, tarefas foram identificadas e assinaladas no DFD, como mostra a Figura 5.4. A descrição de como estas tarefas foi definida é feita a seguir:

Os processos DFD - Receber *frame* de telecomando para AOCS, Receber *frame* de telemetria de AOCS de acordo com o critério **Processo DFD dependente de E/S** seriam considerados como tarefas separadas, devido a cada uma dessas transformações estarem restritas a velocidade ditada pelo dispositivo de E/S como, por exemplo, sensores, com a qual elas estão interagindo. Entretanto, Receber *frame* de telecomando para AOCS foi agrupado com Distribuir telecomando para AOCS conforme o critério **Processos DFD com coesão funcional** para formarem uma única tarefa. Neste caso, este critério prevaleceu visto que, as duas transformações executam funções estreitamente relacionadas causando um aumento no tráfego de dados (*overhead*) entre essas funções se forem mantidas como tarefas separadas. Enquanto Receber *frame* de telemetria de AOCS foi então agrupado com Transmitir telemetria de tempo real de AOCS como uma única tarefa de acordo com o critério **Processos DFD com coesão temporal.** Neste caso, estas transformações realizam funções que são levadas a efeito ao

mesmo tempo. Um frame de telemetria que chega é escrito em um *buffer* de dados com uma capacidade bastante limitada, se os frames não forem retirados deste *buffer* um *overflow*, por exemplo, pode ocorrer. Desta forma, essas funções podem ser agrupadas numa tarefa, de forma que elas sejam executadas cada vez que a tarefa receber um estímulo.

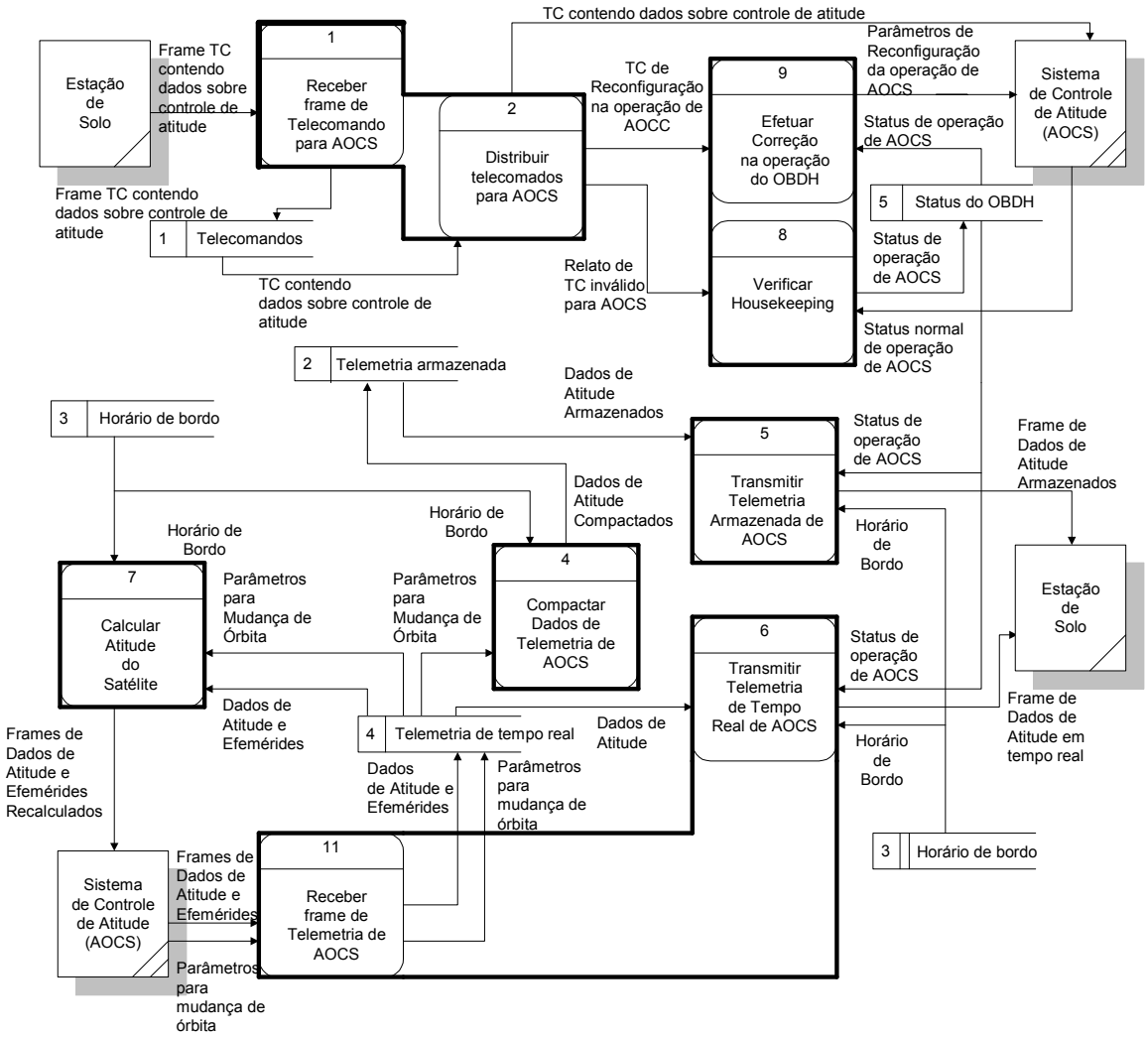

FIGURA 5.4 - DFD decomposto em tarefas.

O processo DFD - Calcular atitude do satélite requer muito processamento, sendo assim, pode aproveitar ciclos ociosos de CPU. Tornando-se, portanto,

uma tarefa ao ser aplicado o critério **Processo DFD que exigem requisitos computacionais**.

Os processos DFD - Compactar dados de telemetria de AOCS, Transmitir telemetria armazenada de AOCS executam periodicamente a compactação dos dados de telemetria de tempo real e a transmissão de telemetria armazenada de AOCS. Portanto, foram consideradas tarefas separadas por não obedecerem a mesma periodicidade apesar de ambas satisfazerem o critério **Processo DFD que têm execução periódica**.

Os processos DFD - Efetuar correção na operação de AOCS e Verificar *Housekeeping* foram agrupados como uma única tarefa de acordo com o critério **Processos DFD com coesão funcional**, uma vez que são executadas funções fortemente relacionadas como a verificação dos dados sobre o funcionamento, a necessidade de execução de testes de diagnose e a reconfiguração do sistema.

Quando um sistema está estruturado em tarefas, é possível definir se as tarefas devem ser executadas em um único processador ou devem ser distribuídas entre dois ou mais processadores. Esta decisão de projeto depende ainda de outros fatores a serem considerados, como por exemplo, à confiabilidade e o desempenho do sistema para que, então, seja definida a arquitetura mais adequada.

Definidas as tarefas, deve-se nomear cada tarefa de forma que o nome corresponda a um subsistema e, em seguida, a formalização das interfaces entre tarefas a partir das interfaces que estavam no DFD na forma de fluxo de dados ou depósito de dados. Conforme a notação DARTS, cada uma dessas interfaces deve manter os fluxos de dados de entrada e de saída de cada tarefa inalterados em relação ao DFD. As interfaces correspondem a duas classes de módulos de interface, que são **o módulo de comunicação entre tarefas** (TCM) e **o módulo de sincronização de tarefas** (TSM*)* (vide Capítulo 2).

Um TCM é sempre executado na tarefa em que foi chamado, tornando possível executá-lo concorrentemente em duas tarefas. Sendo assim, é necessário que os procedimentos de acesso como sincronização e exclusão mútua existam para garantir a consistência e o correto acesso aos dados. Este acesso adequado aos dados é garantido porque o TCM utiliza primitivas de sincronização existentes no sistema operacional, cuja implementação terá variações de um sistema para o outro, mas as funções serão similares. Os TCMs são classificados em dois tipos: **comunicação por troca de mensagens** (MCM) **ou dados** (IHM) (vide Capítulo 2).

Em DARTS, as interfaces entre tarefas são formalizadas de acordo com a forma que um fluxo de dados entre duas tarefas é tratado correspondendo a uma dos seguintes casos:

- Comunicação por fila fracamente acoplada que ocorre quando uma tarefa precisa passar informação para outra e as duas tarefas tem velocidades diferentes. A fila de mensagem é manuseada por uma MCM.

- Comunicação por mensagem com retorno fortemente acoplada que ocorre quando a informação é passada de uma mensagem para outra, sendo que a primeira não pode prosseguir até que tenha recebido um retorno da segunda. Este tipo de comunicação também é manuseado pela MCM.

- Sinalização de evento ocorre quando a notificação de um evento é necessária e nenhum dado é requerido.

Um depósito de dados que precisa ser acessado por duas ou mais tarefas é manuseado por uma IHM na qual a estrutura de dados é definida bem como o acesso à estrutura de dados

Por fim, cada tarefa que aguarda a ocorrência de evento pode necessitar de uma interface TSM.

Com base nas diretrizes descritas foram definidas as interfaces para o sistema exemplo, no entanto, a seleção dos tipos de interface foi feita de modo a corresponder também à implementação da maioria dos projetos de software embarcado desenvolvido pela equipe de supervisão de bordo (vide Capítulo 3). Para corresponder ao software embarcado que é implementado, as interfaces entre tarefas são definidas da seguinte forma: entre as tarefas que recebem e tratam os *frames* de telemetria e os *frames* de telecomando, a interface seria a comunicação por mensagens, ou seja, através do gerenciamento de uma fila de mensagens. As demais tarefas que também necessitam trocar dados, no entanto, a interface utilizada é a de módulo de ocultação de informações, onde se estabelece uma estrutura de dados e mecanismos de acesso a esses dados por não mais que duas tarefas concorrentes. Para tarefas que aguardam a ocorrência de eventos, uma interface TSM foi atribuída conforme mostra a Figura 5.5. Neste caso é importante ressaltar que deve ser dada preferência, sempre que possível, à troca de mensagens, visto que existem primitivas em qualquer sistema operacional servindo tanto para um mesmo sistema operacional como para vários sistemas operacionais com diversas CPUs (sistemas distribuídos). Entretanto, o mesmo que não se aplica ao IHM onde as estruturas de dados são específicas. Como o sistema operacional é sempre desenvolvido no próprio INPE, as primitivas de IHM são implementadas, mas se a tendência for utilizar um sistema operacional existente no mercado, a definição destas interfaces deverão ser revistas.

O diagrama de fluxo de dados é redesenhado, então, como Diagrama de Tarefas, conforme mostra a Figura 5.5, no entanto, o diagrama apresenta as interfaces que seriam definidas em função das primitivas existentes no sistema operacional desenvolvido pelo INPE.

O método DARTS assim como a análise estruturada, não representa fluxos de controle no diagrama de fluxo de dados. Entretanto, a equipe de supervisão de bordo do INPE, tem feito uso do método que utiliza extensões para sistemas de tempo real, em que são representados controle e tempo no DFD (Ward e Mellor, 1985). Neste caso, dificuldades são encontradas em distinguir fluxos de dados de controle na fase de projeto. Enquanto para DARTS a decisão entre o que é realmente dado e o que é usado como controle é adiada para a fase de estruturação das tarefas, em que o fluxo de controle passa a ter o tempo definido para a interface entre tarefas.

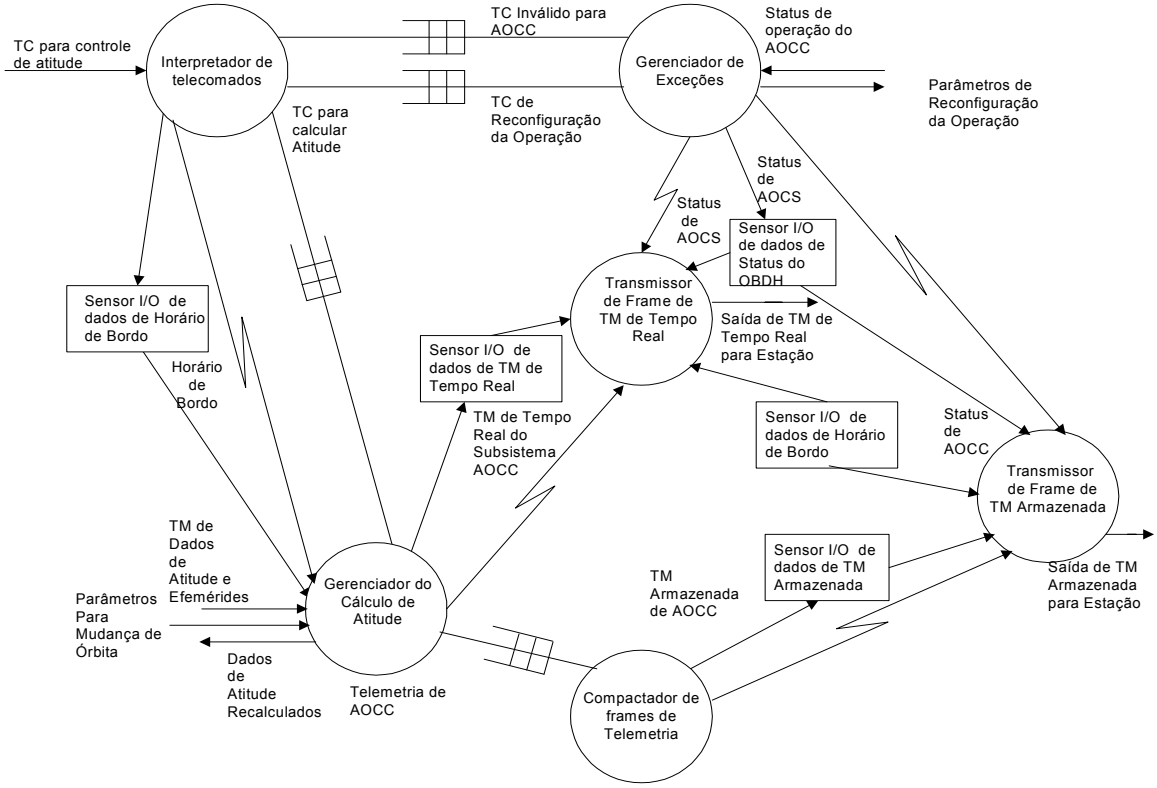

FIGURA 5.5 - Diagrama de tarefas.

Na prática, começar a fase de projeto com o DFD torna-se interessante por adiar também considerações como inicialização do sistema e manuseio de exceções para um estágio posterior, ou seja, na fase de estruturação das tarefas. Ao utilizar DARTS, surge um impulso em tornar cada transformação existente no DFD do sistema em tarefa. Isto leva a "quebra em demasia",

causando uma desnecessária complexidade em virtude da quantidade de divisões e, consequentemente, aumento das interfaces de comunicação e sincronização. Por isto, a etapa de estruturação das tarefas é crucial em DARTS, uma vez que a existência de cada tarefa é justificada.

O diagrama de tarefas da Figura 5.5 não apresenta detalhes sobre as tarefas, então, uma especificação de cada uma deve ser feita. O documento deve conter as entradas utilizadas, as saídas esperadas quando a tarefa for executada e a função que ela realiza. Um modelo de especificação das tarefas é apresentado no apêndice A.

Concluído os documentos de especificação de cada tarefa, um plano de codificação e testes deve ser definido para o sistema. O plano envolve a adoção de uma estratégia ao projetar os casos de teste de forma a verificar se o sistema de software está cumprindo corretamente as suas funções dentro de um contexto maior, isto é, se ele está operando de maneira correta quando colocado em conjunto com os elementos que interagem com ele; ou seja, com as entidades externas com que ele se relaciona.

5.3.1 Plano de Teste de Integração

Após a definição das interfaces entre tarefas, a construção do diagrama de tarefas e a especificação de cada tarefa deve-se definir um plano de testes de integração para o sistema, que determina a ordem em que as tarefas serão testadas, a seqüência em que os testes serão realizados, quais testes podem ser feitos em paralelo e os casos de testes que serão necessários. Por isto, um plano de teste é composto por duas partes: a adoção de uma estratégia para codificação e testes, e o projeto de casos de teste.

A estratégia para a atividade de teste do sistema deve focar, inicialmente, cada tarefa individualmente, garantindo que ela funcione adequadamente como uma unidade e, à medida que forem sendo implementadas e testadas, elas devem

ser integradas para formarem um pacote de software. Finalizada esta etapa, deve-se realizar a integração do software com os outros elementos do sistema como *hardware* e outros sistemas.

O projeto de casos de teste nesta fase deve restringir-se à definição dos casos de testes de integração sem se preocupar com os testes individuais de cada tarefa. Portanto, devem ser projetados casos de teste de integração que testem as interfaces entre as tarefas à medida que elas vão sendo integradas. O objetivo da integração para um sistema de tempo real é testar as **interfaces entre tarefas**. Por isso, durante a integração das tarefas pode ser necessário testar o software como um todo antes de estar completo, para certificar-se de que as tarefas já integradas até o momento estão operando de maneira correta, ou seja, para certificar-se que a comunicação e o sincronismo entre elas ocorrem como previsto.

Em seguida deve-se projetar casos de testes fundamentados na interação que o software realizará com os outros elementos que existem à sua volta como o *hardwar*e e outros sistemas, com os quais ele precisa interagir. Para isto, deve-se projetar casos de teste que verifiquem todas as informações que chegam de elementos que estão fora do escopo do software, e projetar casos de teste que verifiquem todas as informações que saem do software para outros elementos.

Por fim casos de teste devem ser projetados com o objetivo de verificar se o sistema de software está cumprindo corretamente as suas funções dentro de um contexto maior, isto é, se ele está operando de maneira correta quando colocado em conjunto com os elementos que interagem com ele; ou seja, com as entidades externas com que ele se relaciona.

Os casos de teste identificados devem ser integrados ao plano de testes do sistema, que descreve as atividades de testes a serem realizadas. Conforme um plano de teste típico, cada caso de teste identificado para o sistema deve conter as seguintes informações (Yourdon, 1990): o objetivo do teste, a

localização e cronograma do teste, procedimentos de testes e a descrição do teste. O objetivo do teste contém qual é o objetivo do teste e que parte do sistema será testada. O cronograma determina onde e quando o teste será realizado. Os procedimentos descrevem as entradas que serão introduzidas no sistema e as saídas e resultados esperados. Por fim, uma descrição de como os dados de testes devem ser preparados e submetidos ao sistema, como os resultados de saída devem ser colhidos, como os resultados dos testes devem ser analisados, e de quaisquer outros procedimentos operacionais que devam ser observados. Além disto, deve ser incluído no plano de testes a data que determinado caso de teste foi realizado, para que se possa ter controle de quais testes já foram realizados.

Um exemplo de caso de teste do software embarcado, inclui a verificação de um conjunto de tarefas que desempenha uma função representativa no sistema: a telemetria armazenada, que é composta pelas tarefas Compactador de *frame* de telemetria e Transmissor de frame de telemetria armazenada. Conforme modelo, o projeto de caso de teste é apresentado na Figura 5.6.

Para simular a entrada de *frames* contendo dados de AOCS por software, é previsto para o caso de teste o uso de um programa gerador de *frames* para, então, verificar se eles estão sendo compactados e visualizados integralmente e de acordo com o desejado.

Tipo do Teste: Teste de Integração do computador de bordo
Data da Realização do Teste: 01/12/98
Objetivo do Teste: Verificar se as tarefas Compactador de frame de telemetria e Transmissor de frame de telemetria armazenada estão enviando corretamente os dados de telemetria armazenada para a entidade externa Estação de solo.
Cronograma do Teste: Deve ser realizado após os testes de cada uma das tarefas envolvidas.
Entradas: Dados de AOCS **Saídas:** Dados para Estação de solo
Resultados Esperados: Os dados recebidos devem estar sendo compactados e formatados de acordo com a especificação para a recepção em solo.
Observações: Utilizar gerador de frames

FIGURA 5.6 - Modelo de um caso de teste de integração de tarefas.

5.3.2 Projeto das Tarefas

Conforme o método DARTS, a partir do diagrama de tarefas deve ser desenvolvido o projeto individual de cada tarefa, que representa um programa seqüencial e cada programa, um subsistema. Sendo assim, um DFD deve ser desenhado para a tarefa, que será estruturada em módulos usando o método de projeto estruturado, como ilustra a Figura 5.7 (vide Capítulo 2). Como exemplo, a tarefa Transmissor de frame de telemetria de tempo real, é utilizada para mostrar a estruturação de uma tarefa em módulos, obtendo-se assim o diagrama de estrutura.

O DFD do subsistema Transmissor de frame de TM de tempo real é, então, desenhado conforme mostra a Figura 5.8. Utilizando o método de projeto estruturado, o diagrama de estrutura dos módulos do subsistema é obtido a partir do centro de transformação do DFD. O centro de transformação é na verdade, à parte do DFD que contém as funções do sistema, e que independe da implementação particular das entradas e saídas. O propósito dessa análise de transformação é derivar um diagrama de estrutura, a partir do DFD de cada tarefa e a interface entre eles.

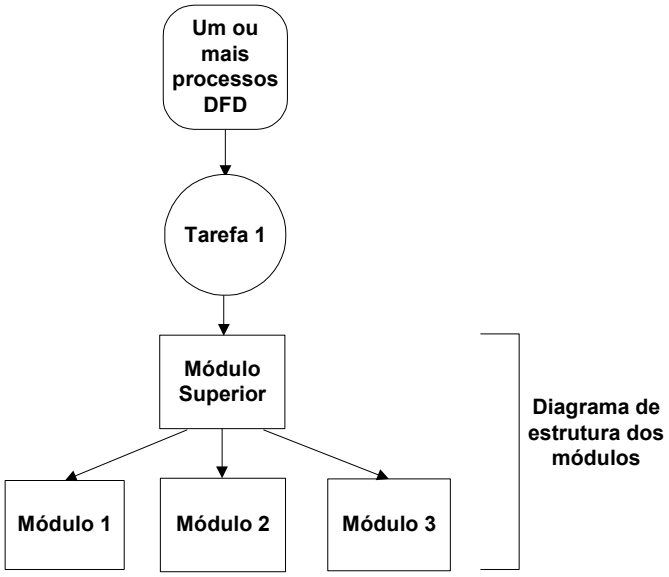

FIGURA 5.7 - Particionamento do sistema em tarefas.

A forma pela qual o centro do DFD deve ser identificado é a partir do processo que represente a entrada, acompanhando o fluxo de dados de entrada até que se alcance o ponto que não se pode mais considerar como entrada. Da mesma forma, a partir do processo que representa a saída final, acompanhar o fluxo de volta até que não se possa mais considerar saída. Quando esses pontos de maior abstração forem identificados, pode se criar um projeto centralizado por transformação, especificando um sistema que chame pela forma abstrata de entrada, realize a transformação para a forma mais abstrata de saída. O centro de transformação do DFD do Subsistema Transmissor de *frame* de telemetria de tempo real está representado na Fig. 5.8.

Cada processo do DFD, a princípio se tornará um módulo do diagrama de estrutura. Porém, os módulos são posicionados hierarquicamente, enquanto no DFD não existem superiores e subordinados. Em função disto, deve-se criar um módulo para ser o módulo superior do diagrama de estrutura. Este módulo deve representar a própria tarefa e abranger o subsistema como um todo, ou seja, deve receber o nome da própria tarefa e, assim, representar todas as funções que seus módulos subordinados realizam.

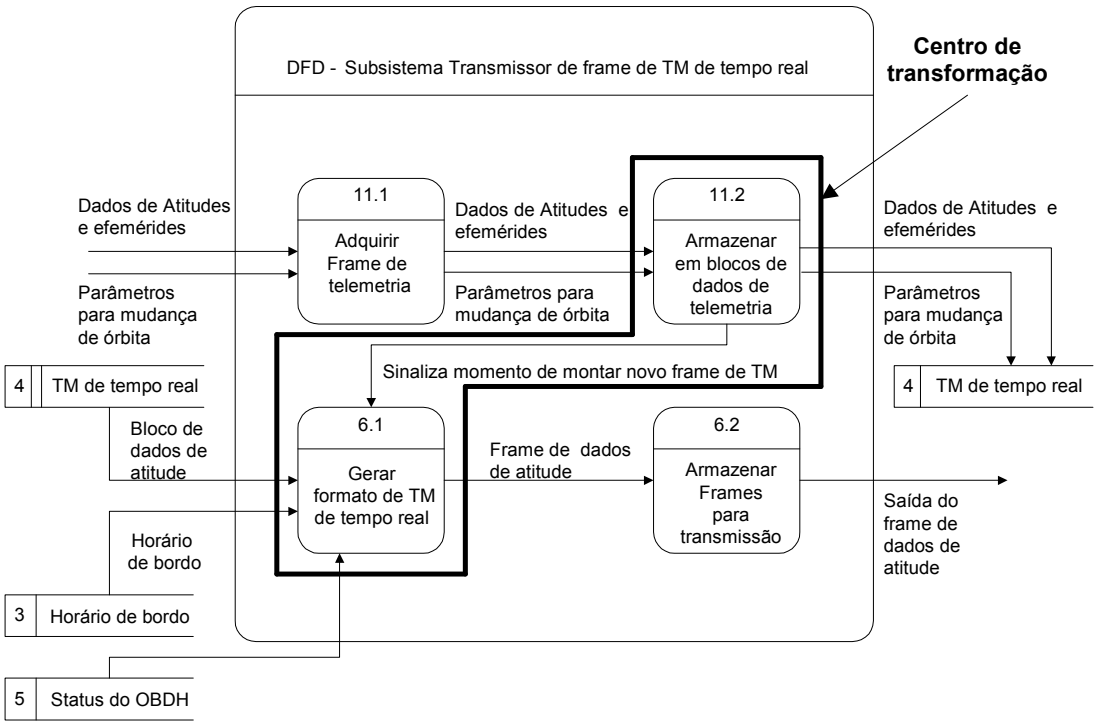

FIGURA 5.8 - Centro de transformação do DFD.

Os processos do centro de transformação do DFD também devem ser redesenhados como módulos retangulares, e serem posicionados ao centro do diagrama; também subordinados ao módulo principal. Uma primeira versão do diagrama de estrutura dos módulos para o Subsistema Transmissor de *frame* de telemetria de tempo real é apresentada na Figura 5.9.

Deve-se verificar se foram projetados módulos com alta coesão e baixo acoplamento. Alta coesão entre módulos existe quando cada um deles é constituído de elementos fortemente relacionados uns com os outros, contribuindo para execução de uma mesma função.

O acoplamento mede o grau de interdependência entre módulos, portanto, o objetivo do projeto é minimizar o acoplamento, tornando os módulos tão independentes quanto possível.

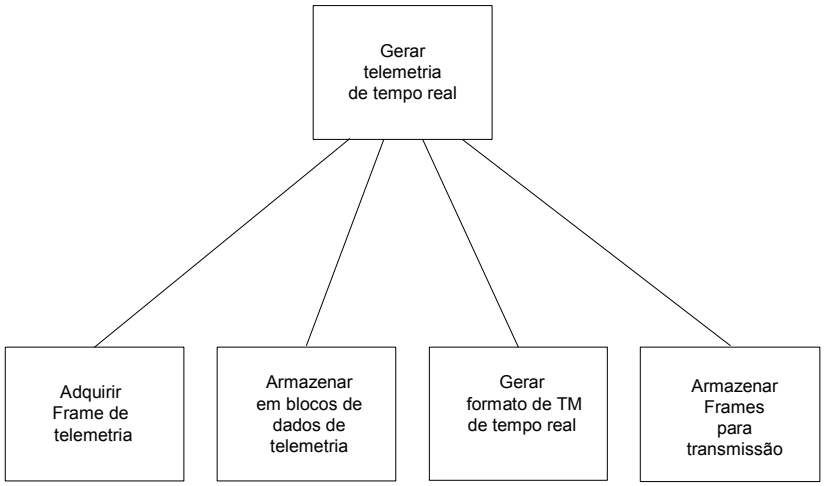

FIGURA 5.9 - Primeira versão do diagrama de estrutura dos módulos.

Pode ser necessário também durante esse processo, mudar o nome do módulo para que ele represente a função que o módulo realiza; pois o nome do módulo deve representar o conjunto de atividades de seus subordinados, enquanto que o nome do processo DFD descreve apenas a sua própria atividade.

Ao acrescentar os nós de comunicação entre os módulos deve-se lembrar que as setas que ligam os módulos a seus subordinados no diagrama de estrutura não correspondem necessariamente às setas do DFD, pois a direção do fluxo de dados não tem o mesmo significado que a direção de uma chamada.

No Subsistema Transmissor de *frame* de telemetria de tempo real, o módulo Gerar formato de TM de tempo real pode ser detalhado em novos módulos, porém os demais módulos são muito simples e, portanto, devem permanecer inalterados.

Finalizada a construção do diagrama de estrutura dos módulos deve-se especificar os módulos do diagrama. Uma especificação dos módulos contendo as entradas utilizadas, as saídas esperadas quando o módulo for processado e a função que o módulo deve executar descrita em apenas uma frase. Um modelo de especificação dos módulos é apresentado no apêndice A.

Em seguida, deve-se continuar o desenvolvimento dos subsistemas partindo para a definição do plano de codificação e teste unitário de cada módulo e de integração (Thome, 1998). Antes de começar a programação propriamente dita, deve-se detalhar sua lógica interna através de um fluxograma, que representa graficamente a lógica procedural de cada módulo.

5.4 Codificação

Nesta fase os módulos serão codificados seguindo a técnica de programação estruturada. Durante a codificação deve haver cautela quanto ao uso de variáveis globais, o ideal é que sejam evitadas. Porém, se usadas deve ser de forma moderada.

O código do módulo superior (topo) deve receber o nome do módulo que por sua vez já recebeu o nome da tarefa. Além disto, no código de cada módulo deve estar incluído um cabeçalho, cujo modelo está apresentado no apêndice A. Comentários internos também devem ser adicionados ao código.

5.5 Produtos de Software Gerados

Embora a metodologia proposta para o desenvolvimento do software embarcado tenha semelhanças se comparada a metodologia que vem sendo utilizada pela equipe do INPE, há diferenças na abordagem, que não deverão ser difíceis de serem absorvidas pela equipe. Propõe-se a confecção adequada de diferentes produtos de software, os quais servirão de base para o esquema de garantia proposto (vide Capítulo 4). Então, um resumo contendo os produtos, as atividades da metodologia proposta e as técnicas utilizadas são apresentados na Tabela 5.1 a seguir:

TABELA 5.1 - Processo de desenvolvimento de software embarcado.

Atividades	**Técnicas utilizadas**	**Produtos**
Especificação de requisitos	Modelo descritivo	Texto descritivo
Análise	Análise estruturada	Diagrama de fluxo de dados
		Dicionário de dados
		Especificação dos processos
Projeto	Método DARTS	Diagrama de tarefas
		Plano de teste
		Especificação das tarefas
		Diagrama de módulos
		Especificação dos módulos
		Plano de codificação e teste
		Fluxograma
Codificação	Programação estruturada	Codificação dos módulos

CAPÍTULO 6

ESQUEMA DE CONTROLE DE QUALIDADE

O objetivo deste Capítulo é apresentar o esquema de controle de qualidade baseado na realização de três atividades aplicadas a cada produto: a realização de revisões técnicas como atividade básica de controle de qualidade, o controle de alterações e o uso de padrões. Para descrever estas atividades, o esquema utiliza os produtos gerados a partir da metodologia proposta para o software embarcado (vide Capítulo 5). O esquema de controle considera ainda os cargos definidos dos membros da equipe e os conhecimentos sobre a aplicação e o ambiente existente no INPE (vide Capítulo 3).

6.1 Controle de Qualidade

As revisões técnicas incluem: apresentação do produtor ou responsável pelo produto, indicação do líder da revisão de acordo com o produto, na definição dos papéis para os participantes e nas listas de conferência ou listas de verificação (*checklist*), as quais podem ser consideradas como ponto de partida para avaliação de cada produto a ser revisado.

A revisão técnica, proposta como principal tipo de atividade para o controle de qualidade, consiste em reunir um grupo de pessoas para examinar o produto. Ela deverá ocorrer ao longo do processo de desenvolvimento e quando necessário, diversas vezes sobre um dado produto. Entretanto, a forma como é realizada é essencial para a sua eficácia na remoção de defeitos. A eficiência está relacionada com a forma que uma reunião de revisão é conduzida durante a apresentação e avaliação do produto. Para isto, papéis são definidos para os participantes, são eles: líder, moderador, secretário, apresentador e revisor dos padrões. O “líder” é um dos papéis que alguém necessariamente deverá

assumir, visto que ele é o responsável pela realização da revisão propriamente dita. O mesmo ocorre com o apresentador, que pode ser o produtor ou algum responsável pelo produto.

O moderador ou mediador deve preocupar-se em controlar a reunião evitando que ocorra divagação, ou atitudes inadequadas. Por envolver pessoas e egos, é necessário estar atento para que erros sejam apontados gentilmente e que jamais exista a intenção de constranger ou menosprezar, pois cada encontro deve ser descontraído e construtivo. Sua principal função é assinalar sempre que a discussão desvie do assunto, evitando assim que a reunião prossiga de forma descontrolada.

O secretário ou relator deve ser nomeado para fazer as anotações das idéias e resoluções importantes, visto que após o término da reunião nenhum dos participantes terá condição de recordar integralmente tudo que foi discutido. Os principais pontos também devem ser anotados em um quadro para servir de foco, ajudando a promover entre os participantes o entendimento comum do que está sendo dito e resolvido na reunião. Caso o assunto seja controverso, não é aconselhável que o moderador e o secretário sejam a mesma pessoa. O secretário registra ainda todas as anotações feitas durante a reunião no relatório de revisão. É imprescindível que este relatório contenha as seguintes informações: o produto que foi revisado, quem é o responsável pela revisão, quem é o produtor, o resultado da avaliação do produto e a data. Um modelo de relatório de revisão técnica é apresentado no apêndice A.

Os demais participantes devem atuar como revisores atentos em verificar o produto quanto ao conteúdo e se foi confeccionado de acordo com os padrões estabelecidos.

Todos esses papéis são estabelecidos para que os participantes trabalhem de modo cooperativo e para que a revisão seja realmente eficaz na remoção de erros. A atuação recomendada aos participantes conforme o papel atribuído a

cada um deles está colocada sob a forma de um **guia de revisão** descrito resumidamente a seguir (Pressman, 1995):

- Revisar o produto e não o produtor;
- Estabelecer e manter uma agenda;
- Limitar o debate e discordâncias;
- Enunciar áreas problemáticas, mas não solucionar todos os problemas;
- Fazer anotações;
- Limitar o número de participantes e insistir na preparação antecipada;
- Possuir um *checklist* para cada produto a ser revisado.

O líder da revisão é indicado de acordo com o grau de conhecimento sobre o produto a ser revisado. Para a realização da reunião de revisão, as seguintes atividades sob a responsabilidade do líder devem ser cumpridas:

- Definir o tipo de revisão do produto (formal ou de conteúdo);
- Definir os participantes;
- Definir os papéis dos participantes (equipe de revisão);
- Definir data e local.

Em função do tipo de revisão do produto, o líder seleciona os participantes para a reunião de revisão. Na seleção estão incluídos os membros da equipe que construíram o produto e integrantes de outras equipes de projeto do satélite que podem ser envolvidas. Entre os participantes, o líder define os papéis de forma criteriosa, avaliando, principalmente, quão polêmico é o assunto. Por fim, são estabelecidos a data e o local da reunião. Cumprida esta etapa, um material informando aos participantes sobre a reunião e o produto a

ser avaliado deve ser distribuído. Os procedimentos descritos devem ser cumpridos observando-se ainda alguns cuidados, tais como (Paula, 2001):

- Selecione somente participantes que possam realmente contribuir;
- Notifique aos participantes com antecedência;
- Confirme a presença dos participantes;
- Planeje uma reunião com duração não prolongada (típica de 2 horas).

Quando a revisão do produto foi concluída, uma análise e posterior encaminhamento dos resultados da revisão devem ser feito. O líder apresenta um dos seguintes resultados: aprovação ou rejeição do produto. Quando um produto é aprovado após uma revisão é gerada a primeira versão, que passa a ser controlado, ou seja, inicia-se o controle de versão. Caso o produto seja rejeitado é preciso avaliar se o defeito está relacionado ao conteúdo, sendo necessário retornar ao produtor ou se o defeito está relacionado com a padronização. Neste caso, nomeia-se um responsável para revisar e providenciar a conformidade necessária.

A partir da descrição, em linhas gerais, dos papéis, responsabilidades e atividades atribuídas aos participantes de uma revisão técnica, é apresentada a seguir, como esquemas de revisão para produtos específicos podem ser realizados. São listados os participantes e seus respectivos papéis, definidos conforme o cargo que desempenham dentro da equipe de supervisão de bordo ou dentro do projeto de satélite (vide Capítulo 3). Por fim, é apresentado o *checklist* para cada produto, cujo principal objetivo é garantir que a avaliação seja conduzida de acordo com uma seqüência lógica e que itens importantes não sejam esquecidos.

6.1.1 Revisão do Produto: Texto Descritivo

O objetivo de uma revisão do texto descritivo é garantir que o texto descreva o sistema de forma correta, devido à sua importância para todo o projeto. Da mesma forma que, por ocasião do levantamento para elaboração do texto, a revisão deve contar com os responsáveis dos diversos sistemas que interagem com o software embarcado. O apresentador é o engenheiro responsável pela elaboração do texto. A função de líder, neste caso, poderá ser exercida pelo responsável pelo segmento espacial por possui a visão geral de todos os sistemas sob controle do computador de bordo.

O papel de moderador deve ser exercido pelo responsável por um sistema de carga útil, por ser o participante mais imparcial entre os responsáveis pelos diversos sistemas que interagem com o computador de bordo.

O papel de secretário deve ser exercido por um membro da própria equipe, sendo o projetista de software o mais adequado. Os participantes e seus respectivos papéis são apresentados na Tabela 6.1 a seguir:

TABELA 6.1 - Participantes da revisão técnica do texto descritivo.

Produto	**Participante das revisões**	**Papéis**
Texto descritivo	Responsável pelo segmento espacial	Líder
	Responsável pelo segmento solo	Revisor
	Responsáveis pelos sistemas de suporte	Revisor
	Responsáveis pelos sistemas de carga útil	Moderador
	Engenheiro do sistema	Apresentador
	Projetista de software	Secretário

Definidos os participantes e os papéis, o líder analisa o texto descritivo durante a revisão com base no seguinte *checklist*:

- A descrição das grandes funções desempenhadas pelo sistema faz parte do texto?
- Para cada função as entradas e o comportamento estão definidos?
- As interfaces externas do sistema foram descritas de forma clara e completa?
- As condições de funcionamento do sistema foram explicitadas?
- Parâmetros de desempenho do software foram expressos no texto?
- Foram especificadas as utilizações de recurso de hardware e suas características tais como: memória, processador e barramento?
- Os tipos de teste de integração e aceitação foram definidos?
- Os prazos para o desenvolvimento foram definidos?

Caso algum item de avaliação não seja satisfeito, o texto não é aprovado. Conforme o tipo de alteração a ser feita, a revisão conta com apenas uma parte dos participantes. Como exemplo, supõe-se que o primeiro item não seja satisfeito, então, a descrição da função deve ser incluída no texto, mesmo que de forma simplificada para posterior complemento. Após inclusão da descrição, convoca-se para nova revisão apenas parte da equipe de acordo com a interface envolvida.

Naturalmente o *checklist* não deve constituir em uma peça que limite as ações dos participantes da revisão, por decisão da equipe, ela poderá ser modificada sempre que for necessário.

A primeira versão do texto se fecha num ponto em que o grupo concorda que o texto expressa todo o conhecimento sobre o sistema que se tem até o momento, exceto detalhes que não contribuem necessariamente para o entendimento do sistema. Depois de certo tempo, deve-se propor uma nova versão do texto descritivo que incorpore os novos conhecimentos, que assim sendo, é submetido à uma nova revisão.

6.1.2 Revisão do Produto: DFD - Diagramas de Fluxo de Dados

Para a revisão do DFD de nível sistemas e os DFDs resultados da primeira expansão são propostas duas revisões. A primeira é uma revisão formal somente para analisar a padronização para que a segunda tenha o foco voltado somente no conteúdo, parte mais importante da revisão.

6.1.2.1 Revisão Formal dos DFDS

Para a realização da revisão formal, é necessária somente a participação dos membros da própria equipe de supervisão de bordo, pois irão analisar se os diagramas foram confeccionados conforme os padrões estabelecidos. Os papéis a serem desempenhados nesta revisão, por membros da própria equipe, estão listados na Tabela 6.2 a seguir:

TABELA 6.2 - Participantes da revisão formal dos DFDs.

Produto	Participante das revisões	Papéis
DFD	Engenheiro do sistema	Líder e Moderador
	Projetista de software	Apresentador
	Analista-programador 1	Secretário
	Analista-programador 2	Revisor

O líder deverá fazer a avaliação da padronização do DFD sistema e da primeira expansão com base no seguinte *checklist*:

- Os processos têm nomes na forma de verbo + complemento?
- Todos os processos e depósitos de dados possuem pelo menos um fluxo de entrada e um fluxo de saída? Em caso negativo, deve-se verificar se o domínio da aplicação justifica tal situação.

- Todos os fluxos de dados possuem pelo menos uma ponta de seta?
- Os fluxos sem nome nos DFDs podem ser explicados?
- Todos os processos foram especificados de forma completa?
- Todos os processos foram descritos de forma compreensível?
- Todas as convenções e padrões gráficos do DFD sistema e da expansão foram obedecidos?

Após a revisão, caso seja necessária alguma alteração, o projetista de software se encarregará de refazer os diagramas em conformidade com a padronização, submetendo os DFDs à uma nova revisão formal.

6.1.2.2 Revisão de Conteúdo dos DFDS

Para a revisão de conteúdo dos DFDs os participantes são os mesmos da revisão do texto descritivo. Por ser o modelo funcional do sistema, a revisão deve contar com os responsáveis dos diversos sistemas que interagem com software embarcado, garantindo assim que os DFDs representem corretamente os requisitos funcionais apresentados no texto descritivo. Os participantes e seus respectivos papéis são apresentados na Tabela 6.3 a seguir:

TABELA 6.3 - Participantes da revisão de conteúdo dos DFDs.

Produto	Participante das revisões	Papéis
Texto descritivo	Responsável pelo segmento espacial	Líder
	Responsável pelo segmento solo	Revisor
	Responsáveis pelos sistemas de suporte	Revisor
	Responsáveis pelos sistemas de carga útil	Moderador
	Projetista de software	Apresentador
	Engenheiro do sistema	Secretário

O líder avalia o conteúdo do DFD sistema e primeira expansão com base no seguinte *checklist*:

- As interfaces com as interfaces externas estão completas?
- As principais funções do sistema estão corretamente representadas?
- A especificação de cada processo está correta?
- Os fluxos de dados que entram e saem de cada função estão corretos (necessários e suficientes)?
- O balanceamento dos fluxos de cada expansão se verifica?

Caso seja necessária alguma alteração, o projetista de software se encarregará de refazer os diagramas e fazer nova revisão. Dependendo da alteração, a revisão será feita com a equipe reduzida. Não havendo mais alteração, é obtida a primeira versão.

6.1.3 Revisão do Produto: Diagrama de Tarefas

Para a realização de uma revisão do diagrama de tarefas, é necessária somente a participação dos membros da própria equipe SUBORD, visto que a modelagem de projeto do software embarcado exige conhecimentos técnicos específicos de projeto. O objetivo principal dessa revisão é verificar se o projeto corresponde ao modelo funcional, que por sua vez deve corresponder à especificação de requisitos do sistema. Os papéis a serem desempenhados nesta revisão, por membros da própria equipe, estão listados na Tabela 6.4 a seguir:

TABELA 6.4 - Participantes da revisão técnica do diagrama de tarefas.

Produto	**Participante das revisões**	**Papéis**
Diagrama de tarefas	Engenheiro do sistema	Líder e Moderador
	Projetista de software	Apresentador
	Analista-programador 1	Secretário
	Analista-programador 2	Revisor

Um *checklis*t para avaliação plano de integração e teste das tarefas deve conter os seguintes itens:

- Cada tarefa foi concebida de acordo com os critérios apropriados?
- Todas as entradas e saídas de cada tarefa estão compatíveis com os DFDs ?
- Todas as interfaces de sincronização (TSM) necessárias estão representadas?

- A descrição de cada tarefa retrata realmente o que a tarefa deve realizar?
- O conjunto das tarefas corresponde ao sistema como um todo?
- Existe a concordância da equipe na definição de responsabilidade no desenvolvimento de cada uma das tarefas?
- Todas as convenções e padrões de construção do diagrama de tarefa foram obedecidos?

Caso seja necessária alguma alteração, o projetista de software se encarregará de refazer o diagrama, que deverá ser submetido à nova revisão até que não haja mais alteração a ser feita e a primeira versão, obtida.

6.1.4 Revisão do Produto: Plano de Integração e Teste

Os testes de software são também atividades de garantia de qualidade (vide Capítulo 2). Porém, a completeza e efetividade da atividade de teste podem ser melhoradas ao avaliarem a estratégia e os procedimentos de testes a serem executados. Para a realização da revisão do plano de integração e teste das tarefas, é necessária a participação dos membros da própria equipe SUBORD. Os participantes e os papéis são os mesmos definidos para a revisão do diagrama de tarefas adicionando-se o responsável pelo segmento espacial e estão listados na Tabela 6.5 a seguir:

TABELA 6.5 - Participantes da revisão do plano de integração e teste das tarefas.

Produto	Participante das revisões	Papéis
Diagrama de tarefas	Engenheiro do sistema	Líder e Moderador
	Projetista de software	Apresentador
	Analista-programador 1	Secretário
	Analista-programador 2	Revisor
	Responsável pelo segmento espacial	Revisor

Um *checklis*t para avaliação plano de integração e teste das tarefas deve conter os seguintes itens:

- Existe concordância da equipe na estratégia de integração das tarefas?
- O cronograma de realização dos testes compõe a estratégia de integração?
- Os casos de teste cobrem as funções do sistema?
- Os casos de teste verificam as interfaces externas do sistema?
- Os casos de teste verificam o funcionamento crítico do sistema?
- O plano de teste é consistente com o plano de projeto do satélite?
- Os recursos e ferramentas de teste necessários foram identificados?

O projetista de software é o responsável em fazer as alterações no plano se necessário, submetendo o plano a uma nova revisão.

6.1.5 Revisão do Produto: Diagrama de Estrutura de Módulos

Para a revisão do diagrama de estrutura de módulos, é suficiente envolver apenas parte da equipe SUBORD, isto é, basta somente que os profissionais de software participem. O apresentador é o analista-programador responsável pelo projeto da estrutura, ou seja, pelo desenvolvimento da tarefa que originou o diagrama. O líder é o projetista de software, responsável por rever todos os projetos dos módulos para certificar-se de que eles estão consistentes com o projeto das tarefas, verificando também a corretude e consistência e de cada módulo. Os participantes e os respectivos papéis para a revisão do diagrama de estrutura de módulos estão listados na Tabela 6.6 a seguir.

TABELA 6.6 - Participantes da revisão do diagrama de estrutura de módulos.

Produto	Participante das revisões	Papéis
Diagrama de estrutura de módulos	Projetista de software	Líder e Moderador
	Analista-programador 1 ou 2	Apresentador
	Analista-programador 2 ou1	Secretário

O projetista de software deve avaliar o diagrama de estrutura de módulos com base no seguinte *checklist*:

- O módulo topo do diagrama representa a tarefa de forma completa?
- O diagrama possui módulos à esquerda relacionados aos dados de entrada, os módulos à direita relacionados aos dados de saída e os do centro relacionados às transformações de dados?
- Os módulos do topo possuem características de gerenciadores e os de nível mais baixo trata aspectos físicos?
- Os módulos têm nomes na forma de verbo + complemento, ou seja, cada módulo executa uma única função?
- O tamanho de cada módulo está adequado?

- Existe módulo que recebe dados que não utiliza?
- Todos os parâmetros necessários estão presentes? Somente eles estão sendo recebidos pelo módulo?
- Os módulos de tratamento de erro estão contidos no diagrama?
- A especificação de cada módulo foi feita de forma adequada?
- Todas as convenções e padrões de construção do diagrama de estrutura de módulos foram obedecidos?
- Os casos de teste unitários e de integração estão corretos e completos?

O projetista de software é o responsável por alterações que se façam necessárias, submetendo o diagrama à nova revisão.

6.1.6 Revisão do Produto: Código Fonte

O principal objetivo da revisão do código de um módulo é verificar se ele representa de forma correta as funções do módulo. Para isto, é necessário que somente a mesma parte da equipe que participou da revisão do diagrama de estrutura, participe também da revisão do código, cumprindo os mesmos papéis. Os participantes e os respectivos papéis para a revisão de código estão listados na Tabela 6.7 a seguir.

TABELA 6.7 - Participantes da revisão técnica de código.

Produto	Participante das revisões	Papéis
Diagrama de estrutura de módulos	Projetista de software	Líder e Moderador
	Analista-programador 1 ou 2	Apresentador
	Analista-programador 2 ou1	Secretário

Um *checklist* para avaliação do código contém os seguintes itens:

- O código realiza corretamente as funções do módulo?
- Os parâmetros do módulo correspondem às entradas e saídas especificadas?
- Os códigos foram produzidos seguindo as estruturas lógicas da programação estruturada?
- Foi testado de acordo com os casos de teste?
- Os comentários existentes ao longo do código são corretos e suficientes?
- Os tipos e a declaração dos dados são apropriados?
- As constantes físicas estão corretas?
- Todas as convenções e padrões estabelecidos para o código foram obedecidos?
- Todos os casos de teste unitários previstos foram realizados com sucesso?

O analista-programador responsável pelo código se encarregará de fazer as alterações se necessário, submetendo o código alterado à nova revisão.

6.2 Controle de Alterações dos Produtos

Alterações nos produtos de software devem ser registradas e analisadas antes de serem implementadas. Os problemas e as soluções devem ser relatados aos que serão atingidos e controladas de tal forma que melhore a qualidade e

reduza a possibilidade de erro. Portanto, o controle das alterações consiste inicialmente em realizar a seguinte seqüência de ações:

1) Estudar qual é o problema;

2) Impacto do problema (abrangência);

3) Propor soluções;

4) Quem vai realizar a alteração.

A análise do impacto da alteração deve ser feita de acordo com o produto, cuja solução se faz através de uma reunião semelhante à revisão, com os participantes e papéis selecionados de acordo com o produto.

Se após a análise do impacto a alteração for recomendada, o engenheiro do sistema, que é o responsável pelo controle, é notificado. Então, o produto é liberado para o produtor. No entanto, ao efetuar a alteração, erros podem ser introduzidos ou até mesmo o resultado da alteração pode não corresponder ao esperado. Por esta razão, o produto alterado passa a ser considerado um produto novo, que deverá ser submetido a outra revisão técnica com os participantes e papéis de acordo com o produto. A Figura 6.1 mostra a lógica associada ao controle de alteração do produto.

Quando um produto é aprovado após a revisão, o engenheiro do sistema move esta versão do produto para o diretório controlado. Mediante a recomendação para alteração, o produto é retirado do diretório controlado e repassado ao produtor. Somente o engenheiro do sistema está autorizado para acesso de escrita neste diretório, ou seja, somente ele poderá incluir ou excluir um produto controlado. Os demais membros possuem acesso somente para leitura.

Dentro do diretório controlado devem ser criados subdiretórios para cada tipo de produto. Esta organização facilita o controle dos produtos produzidos,

permitindo que se tenha sempre um controle de todos os produtos e do andamento da produção (Cunha, 1997).

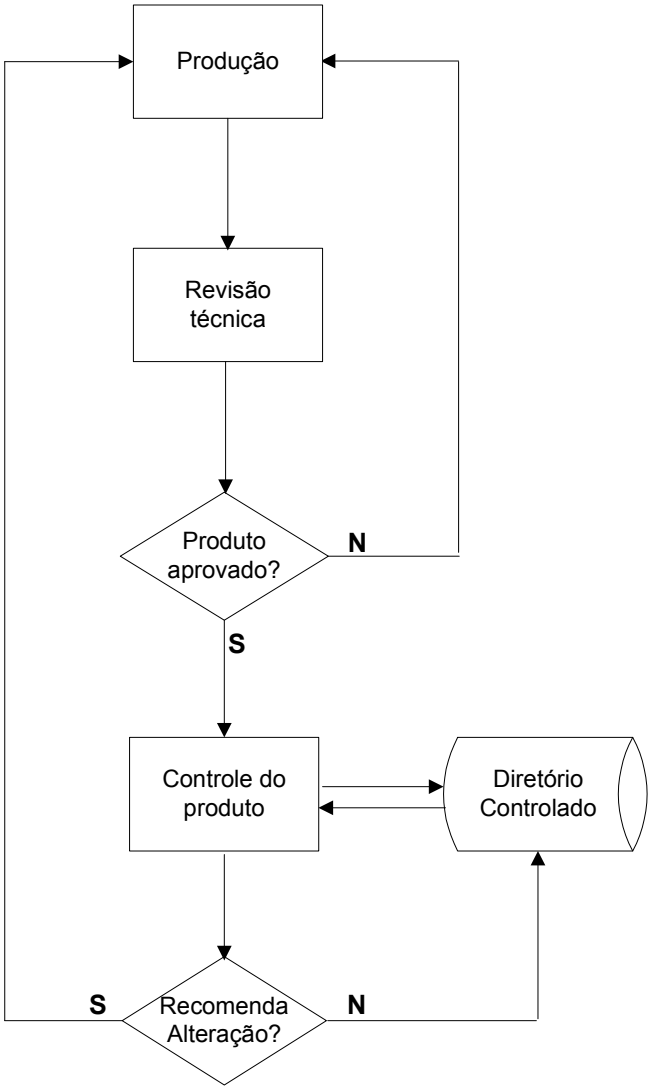

FIGURA 6.1 - Controle de alteração do produto.

Em algum momento a equipe pode constatar a necessidade do desenvolvimento de um **protótipo**, seja para completar interfaces do sistema, verificar viabilidade de requisitos funcionais ou analisar desempenho. Neste caso, o protótipo deve ser construído e utilizado dentro de um diretório somente destinado a prototipação. Evita-se, com este procedimento, uma

interferência entre essa atividade, que não faz parte da composição do sistema, e outras atividades em que os produtos são controlados.

CAPÍTULO 7

CONCLUSÃO

A motivação dessa proposta para a melhoria da qualidade do software embarcado em satélite produzido no INPE está na possibilidade de ela vir a ser aceita e seguida pelo grupo de desenvolvimento e ser aplicada num projeto real. Em função desta perspectiva, o passo não é exagerado, visto que num projeto real não se pode correr grandes riscos.

Por não ter sido ainda utilizada não existe nenhuma garantia quanto aos resultados da proposta. No entanto, mesmo sem haver comprovação espera-se que pelo menos sirva como referência para o planejamento e a evolução da qualidade, e em especial, para se obter um sistema embarcado com segurança mais previsível e trabalhando de maneira mais tranqüila. As novas atividades propostas para a equipe de bordo são baseadas no sucesso de sua utilização por outra equipe no desenvolvimento de software de controle de satélite para segmento solo. Tanto quanto o acréscimo da técnica de Hassan Gomaa no processo de desenvolvimento, como o esquema de controle de qualidade do produto foram utilizados.

Entretanto, grandes alterações como introduzir uma mudança de paradigma, por exemplo, uma mudança para uma metodologia orientada a objeto, poderá significar um passo muito grande e por demais arriscado se não houver tempo e recurso suficientes para treinar a equipe e montar um ambiente adequado para o projeto. Pode-se pensar em mudança de paradigma, mas neste caso é importante aplicá-la primeiro em um projeto piloto antes de implantar em um projeto real.

O fato do passo proposto ser relativamente pequeno, não é somente por considerar maiores e menores riscos que envolvem o software produzido, mas também para alertar a equipe que ela própria poderá ser capaz de vencer uma certa inércia e seguir passo a passo na direção da melhoria da qualidade de

desenvolvimento e da garantia do produto. Espera-se que após o primeiro passo a equipe perceba o ganho obtido com um investimento relativamente pequeno. Se isso acontecer, talvez ela venha acreditar que valerá a pena dar um passo seguinte, talvez se motive para dar um passo muito maior e muito mais ambicioso, naturalmente após avaliar se os resultados compensarão os investimentos a serem feitos.

Uma vez dado o primeiro passo, seja o proposto nesse trabalho, ou um passo diferente, é importante que exista um mínimo de controle para permitir a avaliação dos resultados e das atividades realizadas para que sirva de base para a concepção do segundo passo. Assim a melhoria da qualidade poderá ser realizada como um processo composto por um conjunto de etapas de evolução e cada uma delas constituídas de: planejamento, execução e avaliação.

Diferentes aspectos podem ser explorados em trabalhos futuros, tais como: reuso de produtos de software, revisão do processo de desenvolvimento, acrescentar e aperfeiçoar padrões (Dias, 1999; Costa, 2000), uso de ferramentas de desenvolvimento, uso de ferramentas para o trabalho cooperativo (Sant'Anna, 2000) e mudança de paradigma.

De qualquer forma, a melhoria da qualidade torna-se muito mais compreensível e viável de ser obtida se for vista como um processo de evolução a ser vencida passo após passo, sempre caminhando na obtenção de produtos de software mais confiáveis, de qualidade mais previsível e frutos de um trabalho mais sistematizado e mais produtivo.

REFERÊNCIAS BIBLIOGRÁFICAS

Agresti, W. W. **New Paradigms for Software Development**. Los Alamitos, CA.: IEEE Computer Society Press, 1986.

Alonso, J. D. D. et al. Tratamento de Erros por Software no Padrão INPE de Supervisão de Bordo. In: I Simpósio em Sistemas Tolerantes a Falhas, 30 Set.-01 Out. 1985. São José dos Campos, SP. **Anais**... São José dos Campos, SP: INPE, 1985. p. 194-200.

Alonso, J. D. D. **Guia para Padronização do desenvolvimento de Software Crítico para Aplicações Espaciais**. Dissertação (Mestrado em Ciência no Curso de Engenharia Eletrônica e Computação na Área de Informática) – Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, 1998.

Awad, M.; Kuusela, J.; Ziegler, J. **Object oriented technology for real time systems**. Englewood Cliffs, N.J.: Prentice Hall, 1996.

Basili, V.; et al. SEL's software process-improvment program. **IEEE Software**, v. 12, n. 6, p. 83-87, Nov. 1995.

Beizer, B. **Black-Box Testing:** Techniques for Functional Testing of Software and Systems. New York: John Wiley & Sons, 1995.

Boehm, B. W. A spiral model of software development and enhancement. **IEEE Computer**, v. 21, n. 5, p. 61-72, May 1988.

Booch, G. Object-oriented development. **IEEE Transactions on Software Engineering**, v. 12, n. 2, p. 211-221, Feb. 1986.

Booch, G**. Object-oriented design with applications**. Reading, MA: Addison Wesley, 1991.

Booch, G. **Object-oriented design with applications**. 2.ed. Reading, MA: Addison Wesley, 1994.

Buhr, R. J. A.; Casselman, R. S. **Use case maps for object oriented systems**. Englewood Cliffs, N.J.: Prentice Hall, 1996.

Calvez, J. P. **Embedded real-time systems**. Baffins Lane, Chichester, England, John Wiley & Sons, 1993, 647p.

Chen, P. The entity relationship model-towards a unified view of data. **ACM Transactions on Data Base Systems**, v. 1, n. 1, p. 9-36, 1976.

Coad, P.; Yourdon, E. **Object-oriented analysis**. Englewood Cliffs, N.J.: Prentice Hall, 1991.

Coad, P. **Object-Models:** Strategies, Patterns, & Applications. New Jersey: Prentice Hall, 1997, 515p.

Cobryn, C. UML 2001: a standardization odyssey. **Communications**, v. 42, n. 10, p. 29-37, Oct. 1999.

Coleman, D.; Arnold, P.; Bodoff, S.; Dollin, C.; Gilchrist, H.; Hayes, F.; Jeremaes, P. **Object-oriented development,** - the fusion method. Englewood Cliffs, N.J.: Prentice Hall, 1993.

Costa, S. R. **Objetos distribuídos: conceitos e padrões**. São José dos Campos. (INPE-7939-TDI/744). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, 2000.

Crespo, A. B.; Pasquini, A.; Jino, M.; Maldonado, J. C**.** Cobertura dos critérios potenciais-usos e a confiabilidade do software. In: Simpósio Brasileiro de Engenharia de Software, 11. Fortaleza, 13-15 out. 1997. **Anais...** Fortaleza: Universidade Federal do Ceará, 1997, p. 379-394.

Cunha, J. B. S. **Uma Abordagem de Qualidade e Produtividade para o Desenvolvimento de Sistemas de Software Complexos Utilizando a Arquitetura de Placa de Software – Softboard**. Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, 1997.

Dahl, O.; Hoare, C. A. R. Hierarchical Program Structures. In: Dahk, O.; Dijkstra, E.W.; Hoare, C.A.R. (ed.). **Structured Programming**. London: Academic Press, 1972. p. 175-220.

DeMarco, T. **Structured analysis and system specification**. ACM. Englewood Cliffs, NJ: Prentice Hall, 1978.

Dias, A. S. **Estratégias e padrões para o desenvolvimento de sistemas de software baseados em softboard**. São José dos Campos. (INPE-7262-TDI/704). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, SP, 1999.

Douglass, B. P. **Doing hard time: UML, objects, frameworks, and patterns in real time software development**. Reading, MA: Addison-Wesley, 1999a.

Douglass, B. P. **Real-time UML**. 2.ed. Reading, MA: Addison-Wesley, 1999b.

Elmstrom, R.; Lintulampi, R.; Pezze, M. Giving Semantics to SA/RT by Means of High-Level Timed Petri Nets. **Real-Time Systems**, v. 5, n. 2/3, p. 249-271, May 1993.

Eriksson, H. E.; Penker, M. **UML toolkit**. New York: John Wiley & Sons, 1998.

ESA. **Software engineering standards**. Paris: European Space Agency, Jun. 1989. (ESA PSS-05-0, Issue 1) Jun-1983.

ESA. **Product assurance requirements for ESA space systems and associate equipments**. Paris: European Space Agency, Nov. 1988. (ESA PSS-01- 21, Issue 2, Draft 8).

ESA. **Maintainability requirements analysis and specification for ESA space systems and associate equipments**. Paris: European Space Agency, Jun. 1989a. (ESA PSS-01-240, Issue 1, Draft 4).

ESA. **Reliability assessment for ESA space systems and associate equipments**. Paris: European Space Agency, Jun. 1989c. (ESA PSS-01-231, Issue 1, Draft 4).

ESA. **Reliability requirement analysis and specification for ESA space systems and associate equipments**. Paris: European Space Agency, Jun.1989d. (ESA PSS-01-230. Issue 1, Draft 4).

ESA. **Safety requirements analysis and specification for ESA space systems and associate equipments**. Paris: European Space Agency, Jun.1989e. (ESA PSS-01-240, Issue 1, Draft 4).

Fairley, R. E. **Software engineering concepts**. Singapore: McGraw-Hill, 1985.

Gajski, D.; Vahid, F.; Narayan, S.; Gong, J. **Specification and design of embedded system**. Engewood Cliffs, NJ: PTR Prentice Hall, 1994.

Gane, C.; Sarson, T. **Structured Systems Analysis:** Tools and Techniques. Englewood Cliffs, NJ: Prentice Hall, 1979.

Gomaa, H. Prototyping as a Tool in the Specification of User Requirements. International Conference on Software Engineering, 5. **Proceeding…** Los Alamitos, CA: IEEE Computer Society Press, 1981b.

Gomaa, H. **A Software Design Method for Real Time System.** Communications ACM, v. 27, n. 9, p. 938-949, Sept. 1984.

Gomaa, H. **Prototypes-keep them or throw them away?**: state of the art report on prototyping maidenhead. UK: Pergamon Infotech, 1986a, p. 41-54.

Gomaa, H. Software Development of Real Time Systems. **Communications ACM**, v. 29, n. 7, p. 657-668, July 1986b.

Gomaa, H. **Software Design Methods for Concurrent and Real Time Systems**. Reading, MA: Addison Wesley, 1993.

Gomaa, H. **Designing concurrent, distributed, and real-time applications with UML**. NJ: Addison-Wesley, 2000.

Harel, D. On visual formalisms. **CACM**, v. 31, n. 5, p. 514-530, May 1988.

Hatley, D.; Pirbhai, I. **Strategies for real time system specification**. NY: Dorset House, 1988.

Hinden, H. J.; Rauch-Hinden, W. B. Real-time systems. **Electronic Design**, p. 288-311, 1983.

Lannino, A.; et al. Criteria for software reliability model comparisons. **IEEE Trans. Software Engineering**, v. SE-10, n. 6, p. 687-691, Nov. 1984.

Jackson, M. **Principles of Program Design**. London: Academic, 1975.

Jackson, M. **System development**. Englewood Cliffs, NJ: Prentice Hall, 1983.

Jacobson, I. **Object-oriented software engineering**. Reading, MA: Addison-Wesley, 1992.

Jacobson, I.; Booch, G.; Rumbaugh, J. **The Unified Software Development Process**. Reading, MA: Addison-Wesley, 1999.

Korth, H.; Silberschats, A. **Database System Concepts**. 3.ed. New York: McGraw Hill, 1998.

Kriedte, W.; El Gammal, Y. A new approach to european space standards. **ESA Bulletin**, n. 81, p. 38-43, Feb. 1995.

Kriedte, W. **ECSS** – A single set of european space standards. Nordwijk, Nertherlands: ESTEC/ESA, 1996.

Kumar, S.; Aylor, J. H.; Johnson, B. H.; Wulf, W. A. **The codesign of embedded systems:** a unified hardware/software representation. Boston: Kluwer Academic Publishers, 1996.

Kündig, A. T. **A note on the meaning of "embedded systems"**. In: Goos, G.; Hartmanis, J., (ed.) Lectures notes in computer science. Zürich, Switzerland: Spring-Verlag, Mar. 1986. p. 1-5.

Kündig, A.; Bührer, R. E.; Dähler, J. **Embedded systems: new approaches to their formal description and design**. An advanced course. Zürich, Switzerland: Springer Verlag, 1987, p. 1-6.

Kung, A.; Kung, R. GALAXY: A Distributed Real-Time Operating System Supporting High Availability. **Proceedings…** Real-Time Systems Symposium, **IEEE**, p. 79-87, Dec. 1985.

Leveson, N.; Turner, C. An investigation of the Therac-25 accidents. **Computer**, v. 26, n. 7, p. 18-41, July 1993.

McCracken, D.; Jackson, M. Life Cycle Concept Considered Harmful **ACM Software Engineering Notes**, v. 7, n. 2, p. 28-32, 1982.

Menascé, D. A.; Almeida, V. A. F. **Capacity Planning for Web Performance: Metrics, Models and Methods**. Upper Saddle River, NJ: Prentice Hall, 1998.

Menascé, D. A.; Almeida, V. A. F.; Dowdy, L. **Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems**. Upper Saddle River, NJ: Prentice Hall, 1994.

Menascé, D. A.; Gomaa, H. On a Language Based Method for Software Performance Engineering of Client/Server Systems. **First International Workshop on Software Performance Engineering**. Santa Fe, New Mexico: Oct. 1998, p. 12-16.

Menascé, D. A.; Gomaa, H.; Kerschberg, L. A Performance-Oriented Design Methodology for Large-Scale Distributed Data Intensive Information Systems. IEEE International Conference on Engineering of Complex Computer Systems, **Proceedings…** Southern Florida: Nov. 1995, p. 6-10.

NASA. **Guidebook for Safety Critical Software**. Houston, Texas: NASA, Johnson Space Center, 1996.

Nuseibeh, B. Ariane 5: who dunnit? **IEEE Software**, v. 14, n. 3, p. 15-16, May 1997.

Oliveira, J. L. **Padronização de desenvolvimento de software**. São José dos Campos, SP: INPE, 1985.

Orr, K. **Structured systems development**. New York: Yourdon Press, 1977.

Page-Jones, M. **The Practical Guide to Structured Systems Design**. 2.ed. Englewood Cliffs, N. J.: Prentice Hall, 1988.

Page-Jones, M. **What every programmer should know about object-oriented design**. Englewood Cliffs, N.J.: Prentice Hall, 1995.

Parnas, D.; Clements, P.; Weiss, D. The Modular structure of complex systems. IEEE International Conference on Software Engineering, 1. **Proceedings...**. Orlando, Florida: Mar. 1984.

Paula, W. P. F. **Engenharia de Software, Fundamentos, Métodos e Padrões**. Rio de Janeiro, RJ, LTC-Livros Técnicos e Científicos, 2001.

Pettit, R.; Gomaa, H. Integrating Petri Nets with Design Methods for Concurrent and Real-Time Systems. IEEE Workshop on Real-Time Applications. **Proceedings…** Montreal: Oct. 1996.

Pham, H. Introduction. In: Pham, H., (org.) **Software reliability and testing**. Los Alamitos, CA: IEEE Computer Society Press, Los Alamitos, CA, 1995. p.5.

Pradhan, D. K. **Fault tolerant computer system design**, New Jersey, Prentice-hall, 1995. p. 122-124.

Pressman, R. S. **Engenharia de software**. São Paulo, SP: Makron Books do Brasil, 1995.

Ramamoorthy, C. V.; Bastani, F. B. Software reliability – status and perspectives. **IEEE Transactions on Software Engineering**, v. 8, n. 4, p. 354-371, July 1982.

Rowles, J. B.; Howieson, J. S. P. AIAA 2nd Computers in Aerospace Conference. **Collection of Technical Papers**. Los Angeles, CA.: Oct. 22-24, 1979. p. 468-471.

Rumbaugh, J.; Blaha, J. M.; Premerlani, W.; Eddy, F.; Lorenson, W. **Object-oriented modeling and design**. Englewood Cliffs, N.J.:Prentice Hall, 1991.

Sant´Anna, N. **Um ambiente integrado para o apoio ao desenvolvimento e gestão de projetos de software para sistemas de controle e satélite**. (INPE-8306-TDI/765). Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, 2000.

Schulmeyer, G. C.; McManus, J. I., **Handbook of software quality assurance**. New York: Van Nostrand Reinhold, 1987.

Selic, B. Turning clockwise: using UML in the real time domain. **Communications ACM**, v. 42, n. 10), p. 46-54, Oct. 1999.

Selic, B.; Gullekson, G.; Ward, P. **Real-time object oriented modeling**. New York: John Wiley & Sons, 1994.

Shlaer, S.; Mellor, S. J. **Object oriented systems analysis**. Englewood Cliffs, N.J.: Prentice Hall, 1988.

Mellor, S. J.; Shlaer, S.; **Object Lifecycles- Modeling the World in States**. Englewood Cliffs, NJ: Prentice Hall, 1992.

Simpson, H. The mascot method. **IEE/BCS Software Engineering Journal**, v. 1, n. 3, p. 103-120, 1986.

Simpson, H.; Jackson, K. Process Synchronization in Mascot. **The Computer Journal**, v**.** 22, n. 4, p. 332-345, 1979.

Smith, C. U. **Performance Engineering of Software Systems**. Reading MA: Addison Wesley, 1990.

Strauss, S.; Ebenau, R. G. **Software inspection process**. New York: McGrawHill, 1994. (System design and implementation series).

Tai, K. C.; Carver, R. H.; Obaid, E. E. Debugging Concurrent Ada Programs by Deterministic Execution. **IEEE Transactions on Software Engineering**, v. 17, n. 1, p. 45-63, Jan. 1991.

Texel, P.; Williams, C. **Use cases combined with Booch/OMT/UML: process and products**. Englewood Cliffs, N.J.: Prentice Hall, 1997.

Thome, A. C. **Desenvolvimento de Software por Engenheiros: Diretrizes e Metodologias**. (INPE-6891-TDI/652). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, 1998.

Ward, P. T.; Mellor, S. J. **Structured development for real-time systems**. New York: Yourdon Press, 1985. v. 1-3.

Wirfs-Brock, R.; Wilkerson, B.; Wiener, L. **Designing object-oriented software**. Englewood Cliffs, NJ: Prentice Hall, 1990.

Xu, J.; Parnas, D. L. On satisfying timing constraings in hard-real-time systems. **IEEE Transactions on Software Engineering**, v. 19, n. 1, p. 70-84, Jan. 1993.

Yen, T.; Wolf, W. **Hardware-software co-synthesis of distributed embedded system**. Norwell. Kluwer Academic Publishers, MA: 1996.

Yourdon, E.; Constantine, L. **Structured Design**. 2.ed. Englewood Cliffs, NJ: Prentice-Hall, 1979.

Yourdon, E. **Análise estruturada moderna**. Rio de Janeiro: Campus, 1990.

Zubrow, D. et al. **Maturity questionnaire. Special report CMU/SEI-94-SR-7**, Carnegie Mellon University: Software Engineering Institute, Pittsburgh, PA 15213, June 1994.

APÊNDICE A

MODELOS DE DOCUMENTOS

São apresentados modelos de documentos de especificação que compõem os produtos a serem revisados, o relatório de revisão e o modelo de cabeçalho a ser inserido no código do módulo. Uma lista de formulários apresenta cada um dos modelos a serem utilizados nas revisões, cujo objetivo é simplificar a avaliação de cada produto.

Especificação de Processo

Sistema do satélite

Processo Ref. | **Nome do Processo**

Objetivo/Função

Entradas | **Saídas**

Processos conectados por fluxo

Condições/Limitações

DFD Ref. | **Data de confecção**

Modificações

FIGURA A.1 - Especificação de processo.

Especificação de Tarefa

Sistema do satélite

Tarefa Ref. | **Nome da Tarefa**

Objetivo/Função

Entradas | **Saídas**

Interface (externas e outras tarefas)

Condições/Limitações

Autor | **Data de confecção**

Responsável pelo desenvolvimento

Modificações

FIGURA A.2 - Especificação de tarefa.

Especificação do Módulo

Sistema do satélite

Módulo Ref.	Nome do Módulo	Nome da Tarefa

Objetivo/Função

Entradas | **Saídas**

Módulos Referenciados

Condições/Limitações

Autor | **Data de confecção**

Modificações

FIGURA A.3 - Especificação de módulo.

Relatório de Revisão Técnica

Sistema do Satélite

Produto Ref.	**Produto revisado**

Objetivos

Revisão Ref.	**Projeto**

Data	**Hora**	**Local**

Equipe de Revisão:

Apresentador

Líder	**Secretário**

Participantes

Avaliação do Produto

FIGURA A.4 - Relatório de revisão técnica.

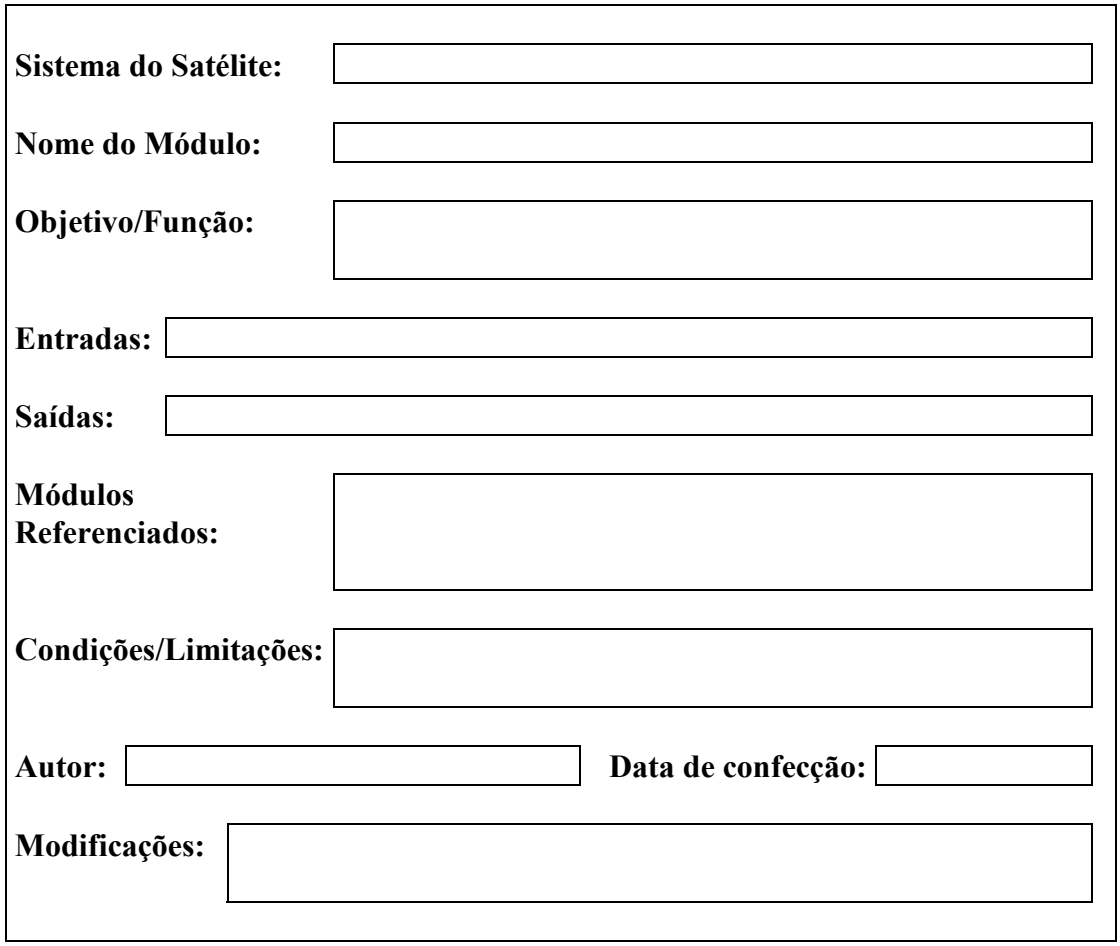

FIGURA A.5 - Cabeçalho a ser inserido na codificação do módulo.