Capítulo 4. Desenvolvimento de Sistemas Críticos

4.1. Definições

Um sistema é formado por diversos componentes, que podem ser de diferentes naturezas. Cada um desses componentes realiza uma tarefa específica, normalmente interagindo com os outros para alcançar um objetivo comum. Pode-se ainda considerar cada componente, como um sistema menor ou um subsistema.

Um **Sistema Crítico** apresenta restrições relacionadas a um determinado fator como, por exemplo: financeiro, tempo, capacidade do equipamento e segurança. Os Sistemas Críticos podem ser de 3 tipos:

- Sistemas críticos quanto à segurança, nos quais uma falha na sua execução pode causar um acidente com danos à vida, à propriedade ou ao meio ambiente.
- Sistemas críticos quanto à missão, onde uma falha resulta na deficiência de algum objetivo; e
- Sistemas críticos quanto aos negócios, onde uma falha gera grandes perdas financeiras.

Sistemas de Tempo Real realizam interação constante com o ambiente, aos quais se impõe restrições em relação ao tempo. Já os **Sistemas Embutidos (Embarcados)** são subsistemas componentes de um sistema maior, geralmente eletrônico ou eletromecânico.

Até a década de 1970, havia relutância em utilizar-se o computador para o controle de sistemas críticos. Por ser o mesmo muito complexo e pouco dominado até então, o computador era ainda pouco utilizado para exercer funções críticas.

O sistema controlador das entradas de ar supersônicas no motor do Concorde é considerado como um dos primeiros sistemas críticos computadorizados a exercer funções críticas de controle.

Os principais argumentos determinantes para a agregação de Computadores aos sistemas Críticos são os que propiciam: o aumento de desempenho; a eficácia na execução de tarefas; a redução de custos; e a versatilidade de sua utilização.

4.2. Computadores Componentes de Sistemas Críticos

Um computador somente contribui para a ocorrência de um acidente, quando ele faz parte de um sistema crítico, principalmente ao exercer função de apoio ao controle. Além disso, a inclusão do software torna o sistema mais complexo, podendo também adicionar erros sutis e difíceis de localizar-se.

Um software componente de um sistema pode contribuir para a ocorrência de acidentes de duas formas distintas: fornecendo valores errados ou fora do tempo ao sistema; e falhando ao reconhecer erros de outros componentes, que necessitem de tratamento especial.

O software componente de um computador inserido em um sistema crítico, denomina-se software crítico. Leveson analisa as consequências da utilização dos Softwares Críticos da seguinte forma:

> **"Antes que o Software fosse usado em Sistemas Críticos quanto a Segurança eles frequentemente eram controlados por dispositivos mecânicos e eletrônicos (não-programáveis) convencionais. Técnicas de segurança de Sistemas são projetadas para lidar com falhas aleatórias nesses Sistemas (não-programáveis). Erros humanos em projetos não são considerados uma vez que todas as falhas causadas por erros humanos podem ser completamente evitadas ou suprimidas antes da entrega e operação"** .

A maioria dos softwares críticos é também de Tempo Real. Um software deste tipo deve responder dentro de restrições de tempo bastante precisas, gerando determinada ação de acordo com eventos externos.

Softwares de Tempo Real geralmente realizam funções de controle do Sistema e troca de dados sob rígidas restrições de tempo, confiabilidade e segurança, de maneira que o resultado de suas execuções possa ser aproveitado por outras partes do sistema.

O Software de Tempo Real necessita de um componente de monitoração para coordenar todos os demais e assegurar a execução do Sistema em Tempo Real. Pode-se também empregar técnicas de Processamento Paralelo e Programação Concorrente para implementar este tipo de aplicação de Software.

4.3. Falhas de Software

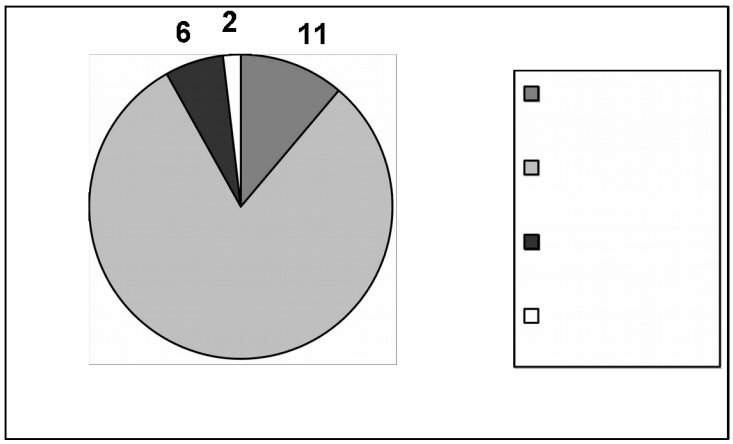
Denomina-se **Defeito** de Software, qualquer deficiência existente no código fonte do seu programa. Se alguma entrada do sistema faz com que uma linha de código contendo defeito seja executada ocorre um **Erro**, que pode constituir-se em uma **Falha**.

A ocorrência de uma falha pode levar o Software a atingir um **Estado Inseguro** (Hazard), de forma que, caso o processamento continue, e não haja um tratamento adequando, ocasione um acidente. Define-se Estado Inseguro como um estado do sistema que combinados a condições externas podem levar a acidentes

Esse tipo de falha ocorre até que o defeito seja corrigido, pois o mesmo não caracteriza-se como um evento aleatório. Isso impede que se apliquem no Software, com propriedade, as técnicas de avaliação e garantia de Segurança de Sistemas Eletrônicos Analógicos e Mecânicos.

Os defeitos de Software ocorrem devido a imperfeições no seu desenvolvimento, falhas de Hardware, ou ainda a interação do computador com outros componentes do Sistema.

A figura a seguir mostra um gráfico da distribuição das principais fontes de defeitos para o Software.



Observa-se no gráfico da figura anterior que grande parte dos defeitos, decorre da especificação de requisitos de forma incorreta e de imperfeições no projeto. É importante ressaltar também que, aumentando-se a complexidade do software, a quantidade de erros cresce exponencialmente.

A próxima tabela mostra os principais defeitos encontrados no desenvolvimento de um software, fornecendo uma boa fonte de pesquisa para a identificação de possíveis defeitos.

Tipos de Defeitos	Subtipos
Funcionais	Requisitos Incorretos
	Completeza dos Requisitos
	Verificabilidade dos Requisitos
	Documentação dos Requisitos
	Mudanças de Requisitos
Implementação de Funções	Acertos na Implementação
	Completeza das Características
	Completeza de Condições
	Domínio de Variáveis
	Mensagens
	Condições de exceção
Estruturais	Controle de fluxo do Programa
	Processamento
Dados	Definição, estrutura e declaração de dados
	Tratamento e acesso aos dados
Implementação	Edição de Programas
	Violação de Padrões de Programação
	Documentação de Programas
Integração	Interfaces Internas do Sistema
	Interfaces Externas do Sistema
Execução do Teste	Projeto do Teste
	Execução do Teste
	Documentação do Teste
	Completeza do Teste
Arquitetura do Software	Uso do Sistema Operacional
	Recuperação de Falhas
	Diagnóstico Incorreto
	Desempenho

Ao analisar a tabela, pode-se concluir que todos os defeitos funcionais relacionam-se aos requisitos do Sistema, representando grande parte dos possíveis defeitos de seu Software, de acordo com a Figura. Estes defeitos, bem como os estruturais e de dados, originam-se na fase de Análise do Software.

Os outros tipos de defeitos provêm das demais fases do desenvolvimento do Software, não existindo nenhuma fase que não possa contribuir para a geração de defeitos. Conclui-se então que um processo que vise garantir a segurança de um

Software deve consistir em uma abordagem completa do ciclo de desenvolvimento de um Software.

Como a maior parte dos defeitos de Software originam-se nas fases de Análise e Projeto, pode-se concentrar maior parte dos esforços para diminuição dos mesmos nas primeiras fases do ciclo de desenvolvimento, procurando detectar o quanto antes os defeitos do Software. Essa procedimento reduz o esforço, o gasto e o tempo necessário para correção dos defeitos.

4.4. Perdas Relacionadas a Falhas de Sistemas

A segurança de sistemas (System Safety) tornou-se uma preocupação dos projetistas, a partir do final da década de 1940. O seu principal impulso ocorreu com o desenvolvimento de mísseis durante a década de 1950, que passaram a exigir novos métodos e técnicas para controlar as situações perigosas associadas aos sistemas de armamentos. A Segurança de um sistema pode ser avaliada sob duas perspectivas: intencional (Security) e ocasional (Safety).

Por meio da utilização de abordagens científicas e administrativas, e obedecendo à restrições de eficácia, tempo e custo, busca-se reduzir a possibilidade da ocorrência de acidentes nos Sistemas. Um acidente é um evento não planejado que causa perdas (financeira, equipamento, danos ao meio ambiente, mortes ou doenças).

Um acidente geralmente origina-se da combinação de vários fatores. Ele é um mecanismo dinâmico, ativado por um Estado Inseguro ou Perigoso (hazard), que flui pelo sistema como uma sequência lógica de eventos consecutivos e/ou concorrentes, até que ocorra uma perda. Devido a esse fato, podem existir diversas opções para interromper essa sequência de eventos, a fim de evitar acidentes.

Deve-se observar que a incidência de acidentes em um sistema relaciona-se diretamente com a complexidade do mesmo e com o grau de interação de seus componentes.

4.5. Processo de Desenvolvimento

Os usuários sempre esperam um sistema seja seguro, disponível e confiável. Contudo, em sistemas “comuns” (não-críticos), os usuários podem estar dispostos a aceitar a ocorrência de algumas falhas. No entanto, algumas aplicações têm requisitos muito altos de confiabilidade, exigindo o emprego de técnicas específicas de engenharia de software.

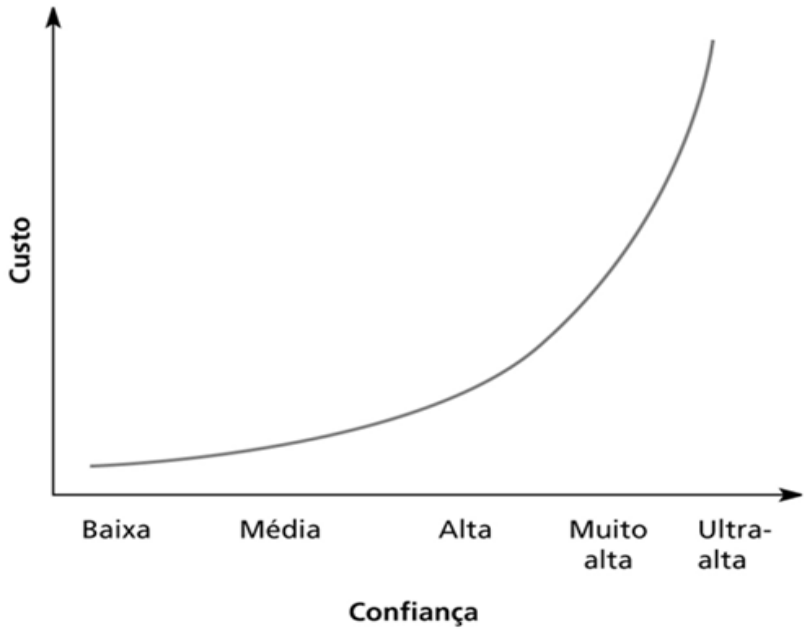
Para atingir uma alta confiabilidade, pode-se empregar técnicas para:

- Prevenção de defeitos: emprega-se técnicas para identificação de defeitos no projeto do software;
- Remoção de defeitos: aplica-se técnicas de verificação e de validação para remover defeitos em um sistema antes que seja entregue; e
- Tolerância a falhas: desenvolve-se o sistema de forma que os erros no software entregue não resultem em falhas em tempo de execução. Envolve Detecção, Avaliação, Recuperação e Reparação de Falhas. Para tanto, pode-se empregar redundância e a diversidade para garantir a execução dos sistemas. Na implementação de sistemas críticos distribuídos é comum o emprego de redundância para a garantia de segurança.

Para assegurar um número aceitável de defeitos de software é importante ter um processo de desenvolvimento bem definido. Para tanto, algumas atividades podem ser empregadas para auxiliar na prevenção e remoção de defeitos:

- Inspeções de requisitos
- Gerenciamento de requisitos
- Verificação de modelos
- Inspeções de projeto e de codificação
- Análise de programas
- Planejamento e gerenciamento de teste
- Gerenciamento de configuração
- Uso de arquiteturas adequadas

No entanto, deve-se avaliar criteriosamente a necessidade do emprego destas técnicas, pois as mesmas podem acrescentar um custo adicional ao processo de desenvolvimento.



Capítulo 5. Engenharia de Software Sala Limpa

5.1. Introdução

A Engenharia de Software Sala Limpa propõe o uso integrado de processos de desenvolvimento tradicional, de verificação de programas e da estatística para produzir software de alta qualidade.

O termo sala limpa (Cleanroom) é originário das salas limpas do desenvolvimento de hardware, onde se prioriza uma disciplina rigorosa de engenharia e o foco na prevenção de defeitos, ao invés da sua correção. Este processo foi proposto nos anos 1980, sendo voltado principalmente para desenvolvimento e certificação de software de maior complexidade.

Nos processos tradicionais de desenvolvimento os erros são inevitáveis, o que pode gerar retrabalho. A Engenharia de Software Sala Limpa procura eliminar os erros ainda durante o projeto, empregando uma rigorosa inspeção do programa.

O processo Sala Limpa pode ser empregado para o desenvolvimento de sistemas novos ou legados, sendo utilizado em geral, para sistemas embutidos e de tempo real.

É um processo independente de linguagem de programação, de ambiente, que permite a prototipação, a orientação a objetos e também o reuso, o qual deve observar a certificação da confiabilidade de cada componente.

As caraterísticas que diferem o Sala Limpa de outros métodos são:

- Especificação Formal: elabora-se um modelo de transição de estados, que mostra as respostas do sistema a estímulos externos. Este modelo é utilizado como uma especificação de sistema.
- Desenvolvimento incremental: o software é desenvolvido e validado em incrementos, que são especificados pelo cliente, no início do processo;
- Verificação estática: o software é verificado utilizando-se rigorosas inspeções de software. Não existe nenhum processo de teste de unidade ou de módulo para os componentes do código; e
- Teste estatístico do sistema: O incremento de software integrado é testado estaticamente para determinar sua confiabilidade. Os testes

estáticos baseiam-se em um perfil operacional que é desenvolvido em paralelo com a especificação do sistema.

Este processo é empregado no desenvolvimento de software para organizações como NASA e DoD, onde foram observados:

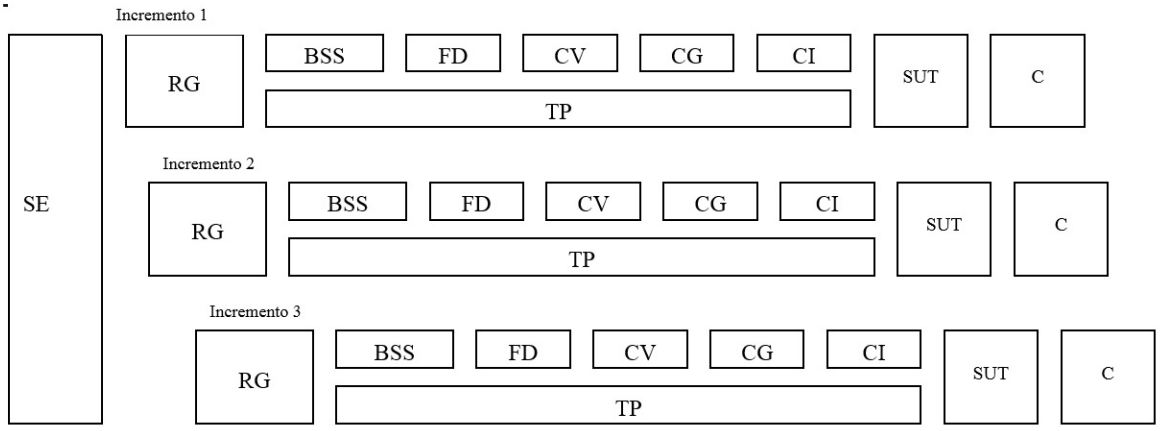
- Aumento da qualidade, com a construção de softwares com muitos poucos erros, sem o aumento do custo de desenvolvimento;
- Aumento de produtividade, desenvolvendo-se mais código em menos tempo;
- Alto ROI (return-on-investment), por meio da redução dos custos de desenvolvimento.

No entanto, algumas críticas podem ser feitas:

- Metodologia muito teórica, matemática e radical para o desenvolvimento de software comercial;
- A substituição de testes unitários por verificação de correção e controle estatístico de qualidade é muito diferente do método costumeiramente empregado pelos desenvolvedores de software; e
- A indústria de software ainda não possui um nível de maturidade ideal para aplicar o processo Sala Limpa.

5.2. Fases do Processo

Na Engenharia de Software Sala Limpa é empregada uma versão especializada do modelo incremental, na qual um conjunto de pequenos incrementos de software é desenvolvido por equipes menores. Quando o incremento é certificado, ele é integrado ao software, aumentado a funcionalidade do mesmo. A próxima figura apresenta o processo.



•CG (geração de código)
•CI (inspeção de código)
•SUT (teste estatístico de uso)
•C(certificação)
•TP (planejamento de Teste)
•SE (engenharia de sistemas)
•RG (coleta de requisitos)
•BSS (especificação de estrutura de blocos)
•FD (projeto formal)
•CV (verificação de correção)

Onde:

- Coleta de requisitos (RG) – desenvolve-se uma descrição mais detalhada dos requisitos;
- Especificação da estrutura de blocos (BSS) - emprega um método de especificação que faz uso de uma estrutura de blocos na especificação funcional (Function Specification Process), visando “isolar e separar a definição criativa do comportamento, dados e procedimentos, em cada nível”.
- Projeto formal (FD) - extensão da especificação (caixa-preta) na qual a mesma é refinada iterativamente;
- Verificação de correção (CV) - atividades de verificação de correções rigorosas sobre o projeto e subsequentemente sobre o código.
- Geração de Código, inspeção e verificação (CG , CI) - tradução da especificação para uma linguagem de programação;
- Planejamento de Teste Estatístico (TP) – a partir da análise do uso do software, uma sequência de casos de teste é projetada;
- Teste estatístico de uso (SUT) - usa-se uma amostra estatística de todas as execuções do programa com base no padrão de uso do software (o que é mais usado é mais testado); e

- Certificação (C) - depois de completadas as verificações, inspeções e testes de uso, e corrigidos os erros, o incremento é certificado para ser incorporado no software.