PROCESSOS PARA SISTEMAS DE SOFTWARE DE SEGURANÇA CRÍTICA

Maurício Peixoto Pacheco da Rocha

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profª. Ana Regina Cavalcanti da Rocha, D.Sc.

Prof. Guilherme Horta Travassos, D.Sc.

Dr. Gustavo Henrique Alves Martins, PhD.

Profª. Sandra Camargo Pinto Ferraz Fabbri, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 2004

ROCHA, MAURICIO PEIXOTO PACHECO DA

Processos para Sistemas de Software de Segurança Crítica [Rio de Janeiro] 2004

VIII, 141, p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2004)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Processo de Software
2. Segurança Crítica
3. Ambientes de Desenvolvimento de Software Orientados à Organização

I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PROCESSOS PARA SISTEMAS DE SOFTWARE DE SEGURANÇA CRÍTICA

Maurício Peixoto Pacheco da Rocha

Março/2004

Orientadora: Ana Regina Cavalcanti da Rocha

Programa: Engenharia de Sistemas e Computação

O crescente emprego de software em sistemas feitos pelo homem e o papel cada vez mais importante que esses sistemas desempenham na sociedade, tornam, cada vez mais relevante, a questão de como construir sistemas de software que funcionem conforme esperado e que não causem danos. Essa questão assume especial importância em todos aqueles sistemas nos quais um mau funcionamento do sistema pode ter conseqüências muito sérias, ou até catastróficas, como a perda de vidas humanas, danos ao meio ambiente, danos materiais expressivos, ou ainda, danos à reputação. Esse tipo de sistema costuma ser denominado sistema de segurança crítica (*safety critical system*).

A história recente de acidentes nesses sistemas mostra que muitos acidentes têm ocorrido porque os engenheiros de software desconhecem ou não aplicam princípios básicos e já conhecidos de segurança de sistemas, ou utilizam abordagens inapropriadas para esse tipo de sistema.

Neste trabalho, os conceitos básicos de uma abordagem apropriada foram organizados e um conjunto de atividades de segurança foi proposto. Essas atividades de segurança foram agrupadas em um programa de segurança, a ser mapeado em um processo de desenvolvimento de sistemas, que contenha o restante das atividades usuais de desenvolvimento e manutenção de sistemas.

Esse programa de segurança foi registrado na Estação TABA, a fim de facilitar a futura configuração e instanciação de ambientes de desenvolvimento para organizações que desenvolvam sistemas de software de segurança crítica.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PROCESSES FOR SAFETY CRITICAL SOFTWARE SYSTEMS

Maurício Peixoto Pacheco da Rocha

March/2004

Advisor: Ana Regina Cavalcanti da Rocha

Department: System and Computing Engineering

The increasing use of software in man made systems and the increasingly important role these systems play in society, make all the more important the question concerning how to build software systems that will work as expected and will not cause damage. This question assumes special relevance in all those systems in which a bad system functioning may have very serious, or even catastrophic consequences, such as loss of human life, damage to environment, property loss, or even damage to reputation. This kind of system is frequently called safety critical system.

The recent history of accidents in these systems shows that many accidents in these systems happen because software engineers do not know or do not apply known basic system safety principles, or are using inappropriate approaches for this kind of system.

In this thesis, the basic concepts of an appropriate approach were organized and a set of safety activities was proposed. These safety activities were grouped in a safety program, to be mapped into a system development process, which contains the rest of the usual system development and maintenance activities.

This safety program was registered in TABA station, in order to easy the configuration and instantiation of development environments for safety critical systems development organizations.

À minha família

AGRADECIMENTOS

À minha família, por todo amor, apoio e incentivo, em especial, minha mãe e a Carolina.

A Karina, Gleison, Mariano, Sávio e Sômulo, pela ajuda e troca de conhecimentos.

Aos professores Guilherme Travassos e Sandra Fabbri, e ao Comandante Gustavo Henrique, por participarem da banca.

À minha orientadora Ana Regina, pela dedicação, orientação e amizade.

ÍNDICE

CAPÍTULO I - INTRODUÇÃO 9

1.1 Objetivo da tese 9
1.2 Estrutura da tese 9

CAPÍTULO II - PESQUISA DO ESTADO ATUAL 11

2.1. Exemplos de sistemas de segurança crítica e acidentes 11
2.1.1. Indústria aeronáutica 11
2.1.2. Indústria ferroviária 14
2.1.3. Indústria automobilística 15
2.1.4. Indústria naval 16
2.1.5. Indústria nuclear 18
2.1.6. Medicina 19
2.1.7. Sistemas de informação 22
2.1.8. Infra-estruturas de larga escala 25
2.1.9. Ampliação de domínios 26
2.2. Uso de software e suas dificuldades 28
2.2.1. Alto grau de inovação 28
2.2.2. Complexidade dos projetos 29
2.2.3. Comportamento discreto 31
2.3. Conclusão 35

CAPÍTULO III - CONCEITOS 36

3.1. Diferença entre *safety* e *security* 36
3.2. Abordagens para segurança 37
3.2.1 Confiabilidade e dependabilidade 38
3.2.2 Segurança de sistemas 39
3.2.3 Escolha da abordagem apropriada 40
3.3 Sistema 43
3.4 Sistema de segurança crítica 45
3.5 Falha 47
3.6 Acidente e incidente 48
3.7 Perigo 49
3.8 Risco 51
3.9 Avaliação e aceitação do risco 52
3.10 Princípios básicos de segurança 54
3.11 Conclusão 55

CAPÍTULO IV - ANÁLISE DE PERIGOS 57

4.1 Processo de Análise de Perigos 57
4.2 Modelos de acidentes 61
4.2.1 Noção de causa 63
4.2.2 Modelagem do erro humano 66
4.2.2.1 Psicologia cognitiva 67
4.2.2.2 Atos inseguros 73
4.2.2.3 Acidentes organizacionais 74

4.3 Técnicas de análise de perigos 77
4.3.1 Tipos de técnicas 77
4.3.2 Exemplos de técnicas 78
4.3.2.1 Checklists 78
4.3.2.2 Failure Mode and Effect Analysis (FMEA) 79
4.3.2.3 Fault Tree Analysis (FTA) 80
4.3.2.4 Hazard and Operability Analysis (HAZOP) 82
4.3.2.5 Análise de perigos da operação 83
4.4 Conclusão 83

CAPÍTULO V - PROGRAMA DE SEGURANÇA NO CONTEXTO DO DESENVOLVIMENTO E MANUTENÇÃO DO SOFTWARE 84

5.1 Proposta de um Programa de Segurança 84
5.2 Atividades do Programa de Segurança no Contexto do Desenvolvimento e Manutenção do Software 94
5.2.1 Processo de Desenvolvimento 94
5.2.2 Processo de Manutenção 96
5.2.3 Processo de Operação 98
5.3 Técnicas de Engenharia de Software sugeridas para realização das atividades do Programa de Segurança 100
5.4 Exemplo de utilização do processo: Sistema de Navegação Eletrônica em Hidrovias 102
5.5 Conclusão 106

CAPÍTULO VI - APOIO AO DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE DE SEGURANÇA CRÍTICA NA ESTAÇÃO TABA 107

6.1 Introdução 107
6.2 Estação TABA e Ambientes de Desenvolvimento de Software Orientados à Organização 108
6.2.1 Configuração e Instanciação de Ambientes na Estação TABA 110
6.2.1.1 Processo de Configuração de Ambientes na Estação TABA 111
6.2.1.2 Processo de Instanciação de Ambientes na Estação TABA 112
6.3 Desenvolvimento de Software de Segurança Crítica na Estação TABA 113
6.3.1 Configuração de um Ambiente 113
6.3.2 Instanciação de um ADSOrg 121
6.4 Conclusão 124

CAPÍTULO VII - CONCLUSÕES E PERSPECTIVAS FUTURAS 125

7.1 Conclusões 125
7.2 Perspectivas futuras 127

REFERÊNCIAS BIBLIOGRÁFICAS 129

ANEXO I – PROCESSO MÍNIMO PARA DESENVOLVIMENTO DE SISTEMAS DE SEGURANÇA CRÍTICA 137

CAPÍTULO I - INTRODUÇÃO

1.1 Objetivo da tese

O objetivo fundamental desta tese é pesquisar, organizar e disponibilizar o conhecimento sobre como construir sistemas de software de segurança crítica.

Esse conhecimento é muito importante porque sistemas de software de segurança crítica estão sendo cada vez mais construídos, devido às inúmeras vantagens econômicas que o software oferece, e muitos acidentes evitáveis têm ocorrido porque os engenheiros de software ignoram, ou não aplicam, princípios básicos de segurança de sistemas, ou usam abordagens que são inapropriadas para esse tipo de sistema.

Para atingir este objetivo, os conceitos básicos de uma abordagem apropriada foram organizados e um programa de segurança, contendo somente atividades de segurança, foi proposto. Essas atividades de segurança devem ser combinadas com as atividades usuais de desenvolvimento de sistemas de segurança crítica.

O programa de segurança foi registrado na Estação TABA, a fim de facilitar a configuração e instanciação de ambientes de desenvolvimento para organizações que desenvolvam sistemas de software de segurança crítica. Esta abordagem fundamenta-se, portanto, nos conceitos de Segurança de Sistemas, Gerência do Conhecimento e de Ambientes de Desenvolvimento de Software Orientados à Organização.

1.2 Estrutura da tese

No capítulo 2, faremos uma revisão do estado atual da prática. Consideraremos exemplos de sistemas de software de segurança crítica e alguns acidentes e incidentes já ocorridos nesses sistemas, ilustrando a importância atual dessa questão. Mostraremos como esses sistemas vêm assumindo papéis cada vez mais importantes na sociedade, estando presentes de forma vital em quase todos os seus segmentos. Em seguida, consideraremos porque software vem sendo usado nesses sistemas e quais as dificuldades que seu emprego apresenta.

No capítulo 3, caracterizaremos o que entendemos por segurança nesta tese. No contexto da engenharia de sistemas, há dois tipos de segurança, fato este reconhecível

pela existência, no idioma inglês, de duas palavras distintas para segurança: *safety* e *security*. Esta tese enfoca *safety*. Em seguida, consideraremos as abordagens atualmente praticadas para construção de sistemas de segurança crítica e escolheremos a melhor abordagem para o tipo de sistema que é objeto desta tese: sistemas de software de segurança crítica. Definida a abordagem, apresentaremos seus conceitos e princípios básicos.

No capítulo 4, destacaremos o processo de análise de perigos. A análise de perigos está no coração de qualquer programa de segurança. Os modelos de acidente e as técnicas de análise de perigos serão vistos. Um estudo mais detalhado sobre erro humano também foi incluído nesse capítulo.

No capítulo 5, proporemos um programa de segurança. que possa ser mapeado num processo de desenvolvimento de sistemas. Ilustraremos como seria o mapeamento das atividades do nosso programa de segurança em um processo de desenvolvimento de sistemas, que contenha o restante das atividades usuais de desenvolvimento e manutenção de sistemas de segurança crítica. No Anexo 1, apresentamos um processo mínimo já contendo as atividades mapeadas.

No capítulo 6, consideraremos ambientes de desenvolvimento de software orientados à organização (ADSOrg), e a Estação TABA, que permite configurar e instanciar esses ambientes. Esses ambientes apóiam a atividade de engenharia de software, possibilitando a gerência do conhecimento que poderá ser útil aos engenheiros de software ao longo dos projetos de uma organização. Realizaremos a configuração e instanciação de um ambiente de desenvolvimento de software orientado para organizações que desenvolvam sistemas de software de segurança crítica.

Finalmente, no capítulo 7, apresentaremos as conclusões deste trabalho e algumas perspectivas futuras.

CAPÍTULO II - PESQUISA DO ESTADO ATUAL

2.1. Exemplos de sistemas de segurança crítica e acidentes

Alguns exemplos de sistemas de segurança crítica que empregam software, e alguns acidentes nesses sistemas, são considerados a seguir.

2.1.1. Indústria aeronáutica

O TCAS (*Traffic Allert and Collision Avoidance System*) [1] é um sistema computadorizado utilizado a bordo de aeronaves que tem por objetivo principal manter uma separação mínima entre as aeronaves, evitando colisões. As principais motivações para o desenvolvimento do TCAS foram a expectativa de continuação da tendência de aumento do tráfego aéreo mundial e a crescente dificuldade em continuar a controlá-lo pelos meios tradicionais.

O TCAS a bordo de uma aeronave consulta *transponders* de controle de tráfego aéreo de aeronaves na sua vizinhança e escuta as respostas desses *transponders*. Um *transponder* é um dispositivo que responde a sinais interrogatórios de rádio. Analisando as respostas dos *transponders*, o sistema determina qual aeronave representa uma ameaça de colisão potencial e fornece conselhos na tela para o piloto. Por exemplo, o sistema pode emitir alerta de tráfego próximo (*traffic advisories*) ou mesmo sugerir uma mudança de altitude para resolver conflitos (*resolution advisories*).

Grande parte dos aviões comerciais de passageiros modernos usa TCAS.

Em julho de 2002, no sudeste da Alemanha, ocorreu uma colisão entre um avião comercial russo e um cargueiro da DHL, ambos equipados com TCAS. Ladkin [2] sugeriu que esse acidente poderia não ter ocorrido se os aviões não estivessem equipados com TCAS, apresentando inclusive, um conjunto de cenários nos quais esse sistema poderia provocar uma situação perigosa ao invés de evitá-la. Por exemplo, não é surpreendente que possa haver variação entre a reação dos pilotos aos conselhos do sistema. Um dos pilotos pode reagir bruscamente, outro pode reagir mais lentamente, outro, ainda, ignorá-los, às vezes justificadamente segundo seu ponto de vista.

Johnson [3] cita um incidente com TCAS interferindo com o sistema de alerta de proximidade do solo (GPWS) de uma aeronave. Em uma determinada situação, o TCAS

gerou recomendações para uma aeronave descer à taxa máxima, o que foi feito pelo piloto, até que o GPWS da aeronave começou a alertar para proximidade do solo. Se a tripulação tivesse continuado seguindo o seu TCAS, a aeronave teria caído. Curiosamente, o controlador de tráfego da área afirmava que naquele momento não havia tráfego na área para justificar o conselho do TCAS.

O grau de utilização de software na aviação tem aumentado. Segundo Hermann [4], nos aviões mais novos, 80-90% das funções são controladas por software. Por exemplo, conforme Storey [5], nos aviões A320 da AIRBUS, as superfícies de controle de vôo primárias são controladas pelo piloto via *joystick*, através de uma tecnologia denominada *fly-by-wire*. Nesses aviões, simplesmente não há mais a coluna de direção tradicional. Os comandos do piloto são interpretados por computadores - o processador principal é um INTEL 386 - que repassam, ou não, comandos apropriados para os atuadores físicos da aeronave. Desta forma, mesmo querendo, o piloto não consegue, por exemplo, produzir um *stall* (perda de sustentação), porque a lógica dos computadores não vai repassar a manobra perigosa para os atuadores físicos.

O BOEING 777 também utiliza um sistema computadorizado de controle das superfícies de vôo primárias, mas seus projetistas optaram por manter a coluna de direção tradicional [8]. A tecnologia *fly-by-wire* é usada também nos ônibus espaciais da NASA e alguns caças mais modernos como o F-16 e o F/A-18. A NASA teria realizado o primeiro vôo *fly-by-wire*, sem *backup* mecânico, em 1972, com um F-8 modificado [9].

Segundo Storey [5], a decisão de colocar tanta dependência no sistema computadorizado não foi leviana. Segundo ele, estatísticas sugerem que 60% dos acidentes com aeronaves envolvem algum aspecto de falha humana. Além disso, em aviões mais modernos, um *stall* pode ser irrecuperável.

Retrospectivamente, as taxas de acidente em aeronaves com alta tecnologia são, em geral, menores que aquelas de aviões convencionais comparáveis [6]. Entretanto, de acordo com Reason [7], é óbvio que o desejo de evitar seres humanos falíveis não é a única razão do aumento da automação. Um incentivo muito poderoso tem sido a disponibilidade de capacidade computacional barata nos últimos anos e o uso de tecnologia de ponta poder oferecer vantagens comerciais consideráveis. Um sistema *fly-*

by-wire também dá aos projetistas mais flexibilidade na configuração, tamanho e colocação dos componentes. Um sistema *fly-by-wire* também seria menor, mais confiável e, em aviões militares, bem menos vulnerável a danos de batalha.

Segundo Reason, quando o consórcio europeu, AIRBUS INDUSTRIE, foi formado no início dos anos 70, a fatia européia do mercado de jatos de transporte comercial era próxima de zero. A fim de competir com os fabricantes americanos dominantes, a AIRBUS resolveu usar a última tecnologia no projeto de *cockpits* automatizados. Apesar dos outros fabricantes, tais como BOEING e MCDONNEL DOUGLAS, terem seguido a tendência, AIRBUS continuou a ser pioneira na divisão do trabalho entre pilotos e sistemas de gerência de vôo automatizados. Vencer a competição requeria uma política agressiva de "ser diferente". A filosofia radical de automação do *cockpit* conquistou para eles um lugar substancial no mercado, mas também criou várias dificuldades novas com fatores humanos em *cockpits* muito automatizados.

Em 1993 [10, 11], em Varsóvia, o computador de um AIRBUS demorou a iniciar a frenagem após o pouso da aeronave, em função de uma combinação de fatores, entre eles, erros na especificação da lógica do sistema. Durante a descida, a pista estava molhada e teria ocorrido aquaplanagem. A lógica do sistema requeria um valor mínimo de velocidade rotativa na roda para que o programa fizesse uma transição de modo ar para modo terra e iniciasse automaticamente a frenagem, como é comum nesse tipo de aeronave. Uma outra condição na lógica do sistema impedia que o piloto demandasse mais potência do reversor para tentar corrigir e parar a aeronave naquela situação.

Nesse acidente, a aeronave ultrapassou o limite da pista, indo bater num alto banco de terra, inconvenientemente construído no final da pista. A aeronave pegou fogo e duas pessoas morreram. O piloto foi oficialmente considerado culpado por ter descido muito rápido, porém uma análise mais detalhada revelou que ele teria sido levado a descer mais rápido por ter sido avisado para esperar a ocorrência de *windshear*, de acordo com um relatório meteorológico, que mais tarde foi descoberto estar defasado. Para compensar a possível situação de menor sustentação devido ao *windshear*, o piloto teria decidido descer mais rápido, para maior estabilidade e sustentação no pouso.

De acordo com Leveson [12], no caso desse acidente, culpar o piloto leva a desviar a atenção para as demais razões, como o fato do relatório meteorológico estar defasado, o projeto do sistema computadorizado de frenagem, a permissão para pousar em pista molhada e a decisão de construir um banco alto no final da pista. Em sistemas complexos, acidentes são complexos, e devemos resistir à tendência de identificar uma causa única, ou apenas as causa mais próximas, se quisermos ser mais eficazes na prevenção de futuros acidentes.

Ladkin mantém um *site* com várias análises de incidentes com aviões comerciais relacionados a computadores em [13].

2.1.2. Indústria ferroviária

A indústria ferroviária também ocupa uma posição de destaque no desenvolvimento de sistemas de segurança crítica já há algum tempo. O sistema que controla o metrô de Paris, freqüentemente é citado como exemplo de aplicação bem sucedida de métodos formais (método B do matemático Abrial).

Um outro exemplo de sistema de segurança crítica na indústria ferroviária é o TVM430, sistema de controle do trem "bala" francês TGV (*Trains a Grand Vitesse*). O TVM430 é um sistema composto de módulos em terra e no trem - processador MOTOROLA 68020 do primeiro MACINTOSH colorido, programado em linguagem Ada.

A velocidade dos trens é tão alta – por volta de 300 km/h - que o condutor não poderia ler de forma confiável os sinais tradicionalmente dispostos ao longo dos trilhos. O TGV utiliza, então, um sistema no qual a informação de sinalização é transmitida através dos trilhos como sinais elétricos, que são capturados por antenas colocadas embaixo do trem. A informação é processada por computadores a bordo e exibida numa tela para o condutor.

Os projetistas desse sistema preferiram manter o controle definitivo do trem com o condutor humano, em virtude da falta de adaptabilidade do sistema para situações eventualmente não antecipadas. Dirigir um TGV é uma tarefa inteiramente manual, mas o sistema de sinalização mantém uma vigilância rígida para garantir a máxima segurança. Por exemplo, se a exibição de sinais na tela para o condutor falha, e a falha é

detectada, o trem é parado automaticamente. Se o condutor ultrapassa uma determinada velocidade máxima, o trem também pode ser freado ou parado automaticamente. Um subsistema de gravação passiva registra todo o processo, semelhante a uma caixa-preta de aeronaves [14].

No TGV, a repartição de tarefas entre o homem e a máquina vai além do "separa quem faz melhor o quê". Modernamente, reconhece-se que a repartição de tarefas entre o homem e a máquina é um problema bem complicado. Segundo Leveson [15], realizar a repartição de tarefas apenas com base nas habilidades relativas de humanos e computadores pode não resultar em uma combinação ótima. O desempenho de uma tarefa depende não apenas das condições presentes, mas também daquilo que o operador acabou de fazer. Além disso, aptidões, estilos cognitivos e atitudes variam entre as pessoas e variam na mesma pessoa com o tempo, por exemplo, a pessoa pode ficar cansada.

A repartição de tarefas no TGV considera as possibilidades de cooperação, ou simbiose, entre o homem e a máquina. O sistema pode supervisionar o desempenho humano e vice-versa. Um incidente ocorrido em 2003 com um trem "bala" japonês ilustra bem a importância dessa abordagem cooperativa. Conforme foi relatado no Fórum de Riscos [16], o condutor do trem dormiu nos controles por 8 minutos, enquanto o trem ia a 270 km/h. O computador interveio e parou o trem 100 metros após a estação. O condutor ainda foi encontrado dormindo.

O TGV e o trem "bala" japonês operam há muito tempo sem acidentes fatais, estabelecendo importantes recordes de segurança. O TGV opera sem acidentes fatais, desde o início do serviço em 1981, se considerada somente a rede mais moderna [14]. O ICE (*InterCity Express*) alemão já esteve envolvido em uma grande tragédia em 1998, quando 100 pessoas morreram num grande descarrilamento, mas o acidente foi devido a uma falha mecânica [17].

2.1.3. Indústria automobilística

O uso de sistemas eletrônicos em veículos tem aumentado significativamente nos últimos anos e deve continuar a aumentar, segundo um guia [20] da MISRA (*Motor Industry Software Reliability Association*). Esses sistemas eletrônicos dependem, cada vez mais, de software. Talvez o exemplo mais conhecido seja o sistema anti-blocagem

(ABS) que impede que as rodas travem numa freada, proporcionando uma frenagem mais segura. Outro exemplo, são os sistemas de navegação, que vão falando (literalmente) para o motorista o caminho que ele deve seguir para chegar num local programado.

Curiosamente, já há um relato no Fórum de Riscos sobre um acidente envolvendo um sistema desse tipo, no qual um BMW foi parar dentro de um rio [21] porque o motorista prestou mais atenção ao sistema que à sua volta. Em outro incidente, o Ministro de Finanças da Tailândia teve que ser resgatado de sua limusine BMW porque o computador de bordo parou. Após a falha do computador, nem os fechos das portas, vidros elétricos e ar condicionado funcionavam. O ministro tailandês e seu motorista ficaram presos dentro do carro abafado. Apesar de todos os seus esforços, passaram-se dez minutos até que pudessem chamar a atenção de um guarda próximo, que os liberou quebrando um dos vidros do carro com uma marreta [22].

Nos carros, o conceito *fly-by-wire* dos aviões se torna *drive-by-wire*. No caso dos carros, o que parece ter começado a impulsionar essa tecnologia teriam sido os padrões de emissão de poluentes cada vez mais rígidos. A *drive-by-wire* diminuiria a quantidade de partes moventes, peso do veículo, requisitos de manutenção, entre outros. No BMW 750i, por exemplo, o acelerador já é controlado indiretamente (*throttle-by-wire*). A BMW tem testado novos modelos com freios e até direção indiretos, respectivamente, *brake-by-wire* e *steer-by-wire*.

2.1.4. Indústria naval

Navios mais modernos estão cada vez mais empregando sistemas de navegação baseados em carta eletrônica (*ECDIS – Eletronic Chart and Display System*). Uma carta eletrônica pode ser uma carta *raster*, quando é uma réplica *scaneada* de uma carta de papel oficial emitida por um *bureau* hidrográfico, ou uma carta vetorial baseada em banco de dados geográficos. Os regulamentos marítimos internacionais determinam que só uma carta vetorial, e somente ela, poderia substituir uma carta tradicional de papel.

Na tela de um ECDIS, é possível ver o navio sendo posicionado automaticamente na carta eletrônica, à medida que ele se desloca, graças à tecnologia do GPS (*Global Positioning System*), bem como informações de diversos sensores. Um

ECDIS pode, também, enviar comandos para o piloto automático do navio de acordo com uma rota previamente programada.

A carta vetorial é bem mais sofisticada que a *raster* porque uma carta vetorial não é uma imagem fixa, mas uma imagem construída sob demanda a partir de objetos georeferidos em um banco de dados. Desta forma, é possível adaptar a imagem da carta na tela às necessidades correntes do usuário, ligando e desligando camadas vetoriais. Quando a carta vetorial gira na tela, para manter o rumo para cima, ou quando se comanda *zoom*, a imagem da carta vetorial continua legível e pode acrescentar, ou suprimir, detalhes automaticamente.

Um dos objetivos da tecnologia ECDIS é que a carta eletrônica substitua a tradicional carta de papel, na esperança de que isso vá proporcionar, entre outros benefícios, maior segurança para navegação. Por exemplo, além do comandante poder acompanhar em tempo real a posição do navio em uma carta, se for usada uma carta vetorial com profundidades, é possível fazer o sistema verificar automaticamente o perigo de encalhe pela carta. Por outro lado, uma carta eletrônica pode ser mantida atualizada por satélite enquanto o navio ainda se encontra no oceano. Muitos acidentes já ocorreram porque a carta de papel estava desatualizada e o navio veio a colidir com um perigo não cartografado.

Sistemas de rastreamento também têm sido empregados para acompanhar a posição das frotas em terra. Num futuro próximo, os navios devem passar a ser equipados com sistema AIS (*Automatic Identification System*) no qual cada navio transmite a sua posição em modo *broadcast*, estabelecendo algumas semelhanças com o TCAS da aviação, comentado anteriormente.

Ao analisar acidentes na indústria naval, Perrow [23] chamou a atenção para o fato de que essa indústria é caracterizada por pressões de produção que minam a eficácia de todas as tentativas de "consertos tecnológicos", incluindo ecobatímetros, radar, sistemas anti-colisão, comunicações por satélite, etc. Pressões de produção minam as defesas oferecidas por esses dispositivos de segurança e aumentam a pressão para usá-los para reduzir despesas operacionais, ao navegar mais rápido ou mais direto. A regulação dessa indústria também sucumbe, facilmente, a pressões econômicas.

De acordo com Perrow, quando o radar foi introduzido pelos ingleses, na Segunda Guerra Mundial, ele foi considerado um grande avanço que, inclusive, teria ajudado aos aliados ganhar a guerra. Assim que atingiu uma razoável confiabilidade e acurácia, começou a ser usado em navios mercantes. Ainda hoje, o termo "alvo" (*target*) é usado para se referir aos navios que aparecem plotados na tela do radar. Entretanto, segundo Perrow, a taxa de colisão não diminuiu com o radar, podendo até ter aumentado. O que certamente aumentou foi a velocidade dos navios equipados com radar. Várias colisões de navios já ocorreram com os radares funcionando.

Embora colisões entre navios não sejam tão comuns, quase todas são bem evitáveis, já que há evidências de que as colisões requereram manobras para acontecerem. Nos círculos marítimos, isso é conhecido como "colisões apoiadas por radar". Um exemplo foi a colisão, em 1981, entre o cargueiro de containers *Lash Atlântico* de 820 pés e 30000 toneladas e o *Hellenic Carrier*, um fretador grego da metade do tamanho do primeiro. O dano, incluindo a remoção do óleo das praias foi de $8.5 milhões.

2.1.5. Indústria nuclear

Computadores e software também já são bastante usados na indústria nuclear. Acredita-se que eles possam trazer muitas vantagens para segurança e redução de custo. Computadores e software podem oferecer um monitoramento mais sofisticado do reator e iniciar o *shutdown* automático quando as condições de operação se tornam perigosas. Eles também podem ser usados para realizar simulações de acidentes, treinamento de operadores ou, conforme [8] para organizar a profusão de alarmes que costumam existir numa sala de controle do reator, amenizando o que é conhecido na área como "efeito árvore de natal".

A usina nuclear de Darlington, Canadá, é uma das mais modernas do mundo. Segundo Leveson [27], ela foi a primeira usina nuclear a obter uma licença para um sistema completamente computadorizado para *shutdown* do reator. Estima-se que a verificação do software empregado nos subsistemas de *shutdown* do reator da usina tenha custado aproximadamente $4 milhões de dólares canadenses (dados de 1994). Havia dois subsistemas, SDS1 escrito em FORTRAN (7000 linhas) e SDS2 escrito em PASCAL (13000 linhas) e mais 6000 linhas de *assembly* distribuídas em ambos. Após

um longo e caro esforço de verificação, a autoridade reguladora canadense (AECL) entendeu que os sistemas estavam adequados e autorizou a sua utilização, porém, também determinou que o software precisaria ser re-projetado no longo prazo [34]. Parnas e Leveson participaram dessa verificação.

2.1.6. Medicina

Em Medicina, marca-passos cardíacos eletrônicos representam um outro exemplo de uma aplicação de segurança crítica envolvendo software. Eles salvam vidas e são implantados nos pacientes para controlar o ritmo cardíaco, devendo funcionar de forma segura por vários anos. Pode haver software tanto embutido no marca-passo quanto no seu programador externo. Segundo Knight [24], entre 1990 e 2000, 52 avisos do FDA (*Food and Drug Administration*), agência que controla medicamentos e alimentos dos Estados Unidos, foram emitidos nesta indústria: 35 relativos a problemas no hardware e 10 relativos a problemas no software. Em 79% dos avisos, intervenção clínica (verificar, analisar ou trocar o dispositivo) foi recomendada.

De acordo com um estudo publicado no jornal Física em Medicina e Biologia, do Instituto de Física (IOP), a nova geração de celulares digitais (ex: GSM) pode interferir com muitos tipos de marca-passos. Marca-passos podem confundir os sinais dos telefones móveis com os sinais elétricos do próprio coração, causando o mal funcionamento do marca-passo. Marca-passos equipados com filtro de cerâmica seriam imunes ao problema [25].

Máquinas de rádio-terapia sofisticadas são outro exemplo de dispositivo de segurança crítica na Medicina e que emprega software. Uma série de acidentes ocorreu com uma dessas máquinas – a THERAC-25. Leveson [26] realizou uma extensa análise desses acidentes.

Segundo Leveson [15], entre junho de 1985 e janeiro de 1987, uma máquina de rádio-terapia controlada por computador, denominada THERAC-25, levou à *overdose* significativa, e em alguns casos fatal, de seis pacientes até que o defeito fosse reconhecido pelo fabricante. Hermann [4] sugere que casos de sub-dosagem menos investigados também podem ter ocorrido.

Grande parte do software do THERAC-25 era similar ao do seu predecessor, o THERAC-20. Esse software tinha erros que podiam se manifestar em certas circunstâncias. Porém, no THERAC-25, o software também foi projetado para assumir maior responsabilidade pela segurança e alguns mecanismos de proteção por hardware, usados no THERAC-20, foram removidos do novo projeto. Como resultado, em determinadas situações em que o THERAC-20 queimaria alguns fusíveis, o THERAC-25 irradiou, fatalmente, alguns pacientes [26]. Por algum tempo, o fabricante insistiu que era impossível, ou muito improvável, haver erros no software, e mais pessoas foram afetadas, até que o problema foi admitido.

Leveson [27] nota que, cada vez mais, mecanismos existentes de segurança implementados em hardware têm sido eliminados e substituídos por computadores, monitoração e controle. Engenheiros têm decidido que mecanismos em hardware não justificam os gastos, ou no caso de um avião, o peso extra.

Em 2001, no Panamá, novos acidentes ocorreram relacionados a software usado em rádio-terapia.

É muito importante considerar bem as implicações da substituição de hardware por software. Um estudo preliminar detalhado pode ser necessário. Um exemplo de estudo desse tipo, extremamente minucioso, pode ser encontrado em [28], que analisa o uso das tradicionais tiras de papel no controle de tráfego aéreo (*flight paper strips*) para melhor entender as implicações de segurança antes de aumentar o grau de automação num sistema assim. Note que a introdução da automação por si só pode gerar mudanças no ambiente que podem dificultar prever o seu real impacto. Por outro lado, pode haver um grau de automação ótimo, não se devendo automatizar sempre cada vez mais.

A "eliminação" de humanos do controle de sistemas a fim de reduzir erros (ou latências) pode levar a um paradoxo. Bainbridge [30] lembra que, com freqüência, o humano não é eliminado mas deslocado para uma posição de supervisão. Bainbridge chama isso de ironia da automação. Se a automação é mal projetada (por um humano!), o operador pode perder a noção do estado do processo sendo supervisionado e quando precisa intervir, em geral quando a situação já é bem difícil, pouco pode fazer ou intervém de maneira desastrosa.

Muitos concordam que envolvimento manual deve ser mantido em um "certo nível" – aqui surgem controvérsias sobre o ponto certo - a fim de atualizar o modelo mental do operador do sistema. Pouco envolvimento do humano pode levá-lo a ficar menos alerta ou vigilante, complacente ou depender demais da automação. A possibilidade de humanos intervirem também é fundamental para poder lidar com situações perigosas que não tenham sido antecipadas pelos projetistas.

Uma área da Medicina que tem se beneficiado bastante da automação é a hemodiálise. Segundo Ronco [29], o advento de computadores na hemodiálise modificou profundamente a forma como as máquinas de hemodiálise eram concebidas.

A máquina de diálise, até então baseada em componentes hidráulicos com controles elétricos mínimos, começou a ser implementada com hardware e software. Em outras palavras, passou a ser possível modificar parâmetros de tratamento dialítico durante a sessão de acordo com instruções específicas pré-estabelecidas, o que representou um aumento da qualidade do tratamento e sobrevida dos pacientes. As inovações tecnológicas abriram caminho para a realização de sistemas e dispositivos dedicados à medição contínua e em tempo-real de sinais físico-químicos provenientes dos pacientes em diálise. Já se pensa em adotar também mecanismos de *biofeedback.* Através desses mecanismos, a máquina pode intervir automaticamente, alterando parâmetros da sessão, por exemplo, se uma situação anormal é antecipada. O objetivo seria reduzir a latência intervencional (se realizada por um humano) e obter um tratamento de alta qualidade que é constante e automaticamente ajustado.

Já o APACHE é um sistema clínico de apoio à decisão e tenta predizer as chances que um paciente muito crítico tem, a fim de orientar e formalizar a conduta em relação a esse paciente. O APACHE atribui um nível a cada paciente para apoiar a decisão dos médicos. Quanto maior o nível APACHE, pior seria a situação do paciente, mas já há muitos casos de pacientes que sobreviveram após terem tido altos níveis no APACHE [31].

Em tese, sistemas especialistas em Medicina poderiam permitir que o conhecimento de um especialista fosse utilizado para uniformizar julgamentos, ou ainda, por um médico menos treinado, ou operando o sistema em regiões mais remotas e carentes. Porém, é fundamental que o médico usuário possa entender e criticar a

sugestão do sistema (cooperação e não substituição), porque, devemos lembrar, que ela reflete apenas parcialmente a opinião do médico especialista.

Kletz [32] sugere que falhas perigosas em sistemas especialistas, em geral, podem advir não só de deficiências no conhecimento do especialista que serviu de fonte do conhecimento, mas também na transposição desse conhecimento para a máquina. Kletz e muitos outros praticantes e pesquisadores em segurança crítica parecem ser bem cautelosos quanto ao uso de inteligência artificial em sistemas de segurança crítica.

O HSE (*Health and Safety Executive*), instituição inglesa que cuida de questões de segurança e saúde, publicou um estudo sobre os aspectos de segurança em sistemas especialistas. Ele identifica cinco perigos potenciais nesses sistemas [33]:

i. diferença entre a especificação e implementação;
ii. discrepâncias ontológicas, ou seja, a coerência interna da representação;
iii. extrapolação além do envelope de operação segura;
iv. falsa precisão;
v. respostas humanas inadequadas para comportamento não previsto ou não compreendido do sistema, já que o humano pode ficar muito fora do *loop*.

2.1.7. Sistemas de informação

Um sério, e infelizmente comum, engano que pode ocorrer é achar que sistemas de informação não seriam de segurança crítica porque só lidam com informação e não controlam nada físico. Na verdade, a imaterialidade já é uma característica do software. Software por si só nunca pode causar um dano direto, mas pode contribuir para que um acidente ocorra. É na interação com os outros componentes de um sistema completo (incluindo hardware e humanos) que o software pode contribuir para um acidente. Por exemplo, um sistema que recebe chamados de emergência e despacha ambulâncias para socorro. Se esse sistema contém um erro no seu algoritmo de rotas, ou não suporta uma demanda realista, ele pode orientar de forma errada, ou deixar de orientar uma ambulância, e a pessoa que necessita de socorro mais imediato sofrer as conseqüências de uma demora no atendimento. No início da década de 90, essa situação aconteceu em Londres (sistema LAS – *London Ambulance System,* bem descrito em [35]) e pelo menos uma pessoa morreu.

Leveson [36] cita um exemplo de uma aplicação de monitoramento da camada de ozônio. Registrar dados para análise não tem implicações diretas para segurança, mas a detecção do buraco na camada de ozônio no Pólo Sul foi atrasada por vários anos porque a redução era tão grande que o computador da NASA, analisando os dados, estava suprimindo a informação, haja vista ter sido programado para supor que desvios tão grandes deveriam ser erros e os dados descartados. A lógica de desprezar valores julgados absurdos é comum em toda aplicação de sensores.

Ainda de acordo com Leveson [15], em 1979, um erro foi descoberto em um programa usado no projeto de reatores nucleares e seus sistemas de refrigeração de apoio. O erro que ele causou no projeto de reatores comprometeu a credibilidade na capacidade, comumente exigida em usinas nucleares, do reator resistir a terremotos. Em conseqüência, a NRC (*Nuclear Regulatory Comission*) interrompeu temporariamente a operação de 5 usinas nucleares nos Estados Unidos.

Segundo Hermann [4], em 1989, um sistema de gerência de um banco de sangue liberou (por erro do sistema) um plasma infectado com HIV.

Em novembro de 2002, a rede do *Beth Israel Deaconess Medical Center* (Boston, E.U.A) falhou de forma impressionante [37]. Durante 4 dias, o hospital teve que funcionar como na década de 70, aparentemente, sem prejuízo da segurança dos pacientes. O sistema inteiro do centro médico falhou, congelando informações sobre prescrições, exames de laboratório, históricos de pacientes e contas. A crise começou quando um pesquisador instalou software para analisar dados e uma grande quantidade de informação começou a fluir pela rede. Gradativamente, o desempenho da rede foi piorando até tornar as operações inviáveis. Apesar do evento iniciador ter sido removido prontamente, a normalidade só foi restaurada após 4 dias, com a participação de "forças tarefa" da CISCO. Uma causa possível foi sugerida na Internet [38] e teria a ver com protocolos de roteamento – *spanning tree protocol loop*. A rede do BETH, embora de certa forma restrita a uma organização, é uma rede importante, circula por volta de 40 *terabytes* de informação diariamente e, ironicamente, teria até recebido um prêmio em 2001 de melhor rede nacional (americana) na área de saúde.

Ferramentas e ambientes de desenvolvimento usados para desenvolver softwares de segurança crítica podem passar a ser de segurança crítica, dependendo da sua

contribuição para obtenção de um produto seguro. Por exemplo, um compilador pode ter *bugs* que tornem o código objeto gerado diferente do especificado e programado em alto nível. Algumas normas definem critérios para seleção e combinação de ferramentas, como a norma inglesa para sistemas de segurança crítica [39], por exemplo.

Ao comparar sistemas operacionais de código aberto, tipo Linux, contra sistemas operacionais de código fechado, tipo Windows, Wichmann [40] sugere que afirmativas tais como "Windows é inseguro", a rigor não têm sentido. Segurança é uma questão do sistema como um todo. A utilização, ou não, de um determinado componente de software em um sistema não é nem suficiente para provar, nem para negar sua segurança. Em particular, o uso de um sistema operacional precisa ser considerado no contexto do sistema maior do qual ele faz parte. Entretanto, em muitos casos, o sistema operacional desempenha um papel chave na determinação da segurança do sistema, podendo, por exemplo, facilitar a verificação da segurança, ou possuir propriedades que serão úteis para garantia da segurança.

Em geral, as ferramentas disponíveis no mercado não foram desenvolvidas com o rigor de um sistema de segurança crítica. Alguns fabricantes proíbem, ou tentam se eximir da responsabilidade pelo uso de suas ferramentas no desenvolvimento de sistemas de segurança crítica, entre eles temos fabricantes de compiladores famosos, como por exemplo, BORLAND e MICROSOFT. Uma forma de compensar isso, em sistemas mais críticos, tem sido utilizar ferramentas especiais, mais caras, quando elas estão disponíveis, ou usar várias versões de um mesmo tipo de ferramenta para obter maior garantia. Por exemplo, no sistema de controle do BOEING 777, três compiladores diferentes foram usados [8].

Entretanto, o uso de software desenvolvido por terceiros em sistemas de segurança crítica, quer seja, software de prateleira, software previamente desenvolvido ou ferramentas, pode dificultar a verificação da segurança em função da falta de informação disponível sobre esses produtos, gerando impacto na nossa capacidade de avaliar a segurança do sistema.

Usar diversidade ainda é uma estratégia polêmica, pois podem existir pontos de falha comuns nas "diversas" ferramentas. Segundo Leveson [41], em 1990 um

matemático relatou um *bug* em um fórum de discussão do programa MACSYMA, um programa amplamente utilizado, que computa funções matemáticas. Esse programa computava incorretamente a integral de 0 a 1 de $\sqrt{x+(1/x-2))}$ como -4/3, ao invés do valor correto que seria 4/3. Outros participantes do fórum ficaram curiosos e foram testar o mesmo problema em outros pacotes matemáticos. O resultado foi que 4 pacotes (MACSYMA, MAPLE, MATHEMATICA, e REDUCE) apresentaram o mesmo erro e somente 1 (DERIVE) apresentou a resposta correta. Esses pacotes matemáticos foram todos desenvolvidos separadamente, em linguagens de programação diferentes e mesmo em países diferentes e foram amplamente usados durante anos.

Alguns dos primeiros receptores de GPS (Global Positioning System) também apresentavam os mesmos *bugs* em algumas condições de contorno. Por exemplo, quando a latitude da posição passava de 90^0, vários receptores não mudavam de hemisfério. Alguns até travavam nessa situação.

2.1.8. Infra-estruturas de larga escala

Sistemas computadorizados já apóiam, ou controlam, virtualmente grandes infra-estruturas sociais. Sistemas de infra-estrutura de larga escala têm sido construídos no mundo e no Brasil. Por exemplo, no Brasil, temos o recente Sistema de Pagamentos Brasileiro (SPB).

Nesses sistemas de infra-estrutura circulam informações vitais para o funcionamento da sociedade. No SPB circulam hoje todos os pagamentos acima de R$5000,00. Essas informações precisam circular em segurança e em tempo hábil. Vários outros sistemas, alguns críticos também, estão interligados a esses sistemas. O potencial para problemas em larga escala é muito grande devido ao alto grau de acoplamento que pode estar envolvido.

De acordo com um artigo da INFORMATIONWEEK BRASIL [49], em 2002, após um ano de operação, já trafegavam 43 mil ted's, o equivalente a R$ 8,4 bilhões por dia, no SPB. No balanço de um ano de SPB, algumas lições foram aprendidas. Muitos especialistas defendem que o maior erro, do lado dos grandes bancos, principalmente de varejo, foi terceirizar tudo com grandes fornecedores. O BRADESCO, por exemplo, teve que rever sua estratégia quando percebeu a fragilidade da solução adquirida no

mercado. O SPB-2 deve integrar, além das instituições financeiras, as empresas e pessoas físicas ao sistema, mas ainda há muita discussão sobre a questão da exigência de certificados e custos.

O Comitê Basel (Suíça) sobre Supervisão Bancária recentemente acrescentou o risco operacional à fórmula que um banco determina o mínimo de capital regulatório que ele precisa alocar para se proteger de contingências. A fórmula, previamente, só calculava o risco de crédito. Segundo o comitê, risco operacional deve incluir desde fraudes não detectadas até roedores comendo as linhas de energia, e falhas de segurança [50]. No Brasil, o Banco Central tem realizado auditorias nos bancos para avaliar o estado da prática em tecnologia de informação.

O Brasil também foi o primeiro país a realizar uma votação eletrônica de escala nacional em 2000 e vem realizando essa atividade com sucesso desde então. Os Estados Unidos têm tido uma série de problemas com votação eletrônica. Aparentemente, eles não reconhecem o sucesso brasileiro e insistem na importância da recontagem de votos, que fazemos apenas por amostragem. O problema da votação eletrônica parece ser especialmente complicado porque, de acordo com Mercuri [51], a privacidade é crítica para uma eleição justa, necessária para prevenir a coerção do eleitor, intimidação e venda de votos. Mas manter a privacidade do eleitor parece conflitar com a necessidade de garantir que os votos vieram de eleitores legítimos, votando não mais que uma vez. Na prática, isso acaba obrigando a confiar demais nos fabricantes.

2.1.9. Ampliação de domínios

Se no passado era mais comum que sistemas de segurança crítica ficassem restritos a nichos bem específicos (por exemplo, aviação, medicina ou indústria nuclear), hoje nota-se uma ampliação de domínios e muitos sistemas passam a ser de segurança crítica, muitas vezes sem a adequada percepção desta situação por parte de seus projetistas e usuários, conforme sugere Littlewood [42].

Segundo Lutz [43], avanços tecnológicos futuros e o mercado devem promover o surgimento de ainda mais aplicações de segurança crítica.

Dunn [44] sugere que as perdas combinadas, em termos de sofrimento humano e danos materiais, nessas aplicações consumidas em massa podem exceder de longe

àquelas dos sistemas mais convencionais, que são potencialmente mais catastróficos, mas que existem em menor quantidade. Por exemplo, o NIST (*National Institute of Standards and Technology*) publicou recentemente um relatório no qual ele estima o custo nacional (EUA) dos defeitos em software como algo em torno de $59 bilhões por ano [45].

Segundo Littlewood [42], nas áreas de aplicação mais cotidianas e insuspeitas, tradicionalmente menos associadas à segurança crítica, ainda há uma grande barreira a ser vencida com relação aos fabricantes de software passarem a considerar a confiabilidade e segurança dos seus produtos como vantagem de mercado a ser ganha. Mas, gradualmente, parecem estar ocorrendo algumas mudanças.

Chillarege [46] argumenta que organizações bem sucedidas são aquelas que adaptam seus processos para entregar produtos que respondam aos valores do mercado. Freqüentemente, recursos são limitados e oportunidades transientes. A organização prudente reconhece suas limitações e garante que seus produtos satisfaçam, pelo menos, aqueles valores em mais alta demanda. Assim, o mercado e nós mesmos temos grande influência sobre a qualidade dos produtos. O mercado atual ainda parece valorizar bastante produtos cheios de recursos, complexidades e, inevitavelmente, *bugs*.

No caso de sistema de segurança crítica, conforme observa Leveson [47], a tolerância para com acidentes vem diminuindo na medida que os sistemas vêm assumindo um papel cada vez mais importante na sociedade e seus poderes de causar dano aumentam.

Como consumidores, temos um papel muito importante a desempenhar, qual seja, tornar um real investimento em segurança desejável para os fabricantes. Schneier [48] lembra que somos todos consumidores de segurança e precisamos ser melhores consumidores de segurança. É bem razoável argumentar que o discurso sobre *trustworthness* da MICROSOFT é, em parte, *marketing*, e que segurança de fato só será obtida se usuários e projetistas se tornarem mais conscientes quanto à importância da segurança.

2.2. Uso de software e suas dificuldades

Em [52], Parnas *et al.* consideram alguns motivos que tornam atrativo o emprego de software. Por exemplo, software permite colocar no sistema mais lógica, a um custo mais baixo e em menos espaço, que o hardware equivalente. Software, em princípio, é também mais fácil de alterar. Software parece viabilizar novos tipos de sistemas. Máquinas que eram fisicamente impossíveis, ou impraticáveis de construir se tornam viáveis. Nesta tese, por exemplo, já vimos o exemplo dos avanços nas máquinas de hemodiálise.

A tendência em usar cada vez mais software deve continuar e então, precisamos saber como usar essa tecnologia sem ter que expor as pessoas a riscos indevidos. Precisamos conhecer o que pode sair errado, as soluções e as grandes questões em aberto dessa área.

Quando sistemas de segurança crítica utilizam software, o seu desenvolvimento e verificação parecem ficar especialmente mais complicados [5, 15, 53]. Entre os motivos mais comumente citados, e que serão brevemente considerados aqui, estão: (1) o alto grau de inovação e (2) complexidade dos projetos envolvendo software e o (3) comportamento discreto do software.

2.2.1. Alto grau de inovação

Petroski [54] afirmou que entender o que é a engenharia e o que os engenheiros fazem é entender como falhas podem acontecer e como elas podem contribuir mais que os sucessos, para o avanço da tecnologia. As lições aprendidas dos desastres podem fazer mais para avançar o conhecimento de engenharia, que todos as máquinas e estruturas bem sucedidas no mundo.

Entretanto, segundo Leveson [15], o acelerado ritmo de inovação atual parece reduzir as oportunidades de se utilizar experiência acumulada do passado e o potencial de dano de nossos artefatos tem aumentado. Em geral, a tolerância para com acidentes vem diminuindo dada a gravidade dos mesmos. Muitos sistemas estão passando a ser controlados pela, comparativamente nova, tecnologia de software e sistemas inéditos e complexos, alguns tornados viáveis graças ao software, estão sendo desenvolvidos.

Hermann [4] lembra que, no passado, muitos sistemas de segurança crítica dependiam de princípios bem estabelecidos de confiabilidade e segurança, por exemplo, das engenharias elétrica, mecânica ou civil, acumulados ao longo de vários anos (e muitos acidentes). Porém, ainda não dispomos de uma base de conhecimento comparável para sistemas de software.

Ao comparar a situação da tecnologia de software, hoje, com a das máquinas a vapor de alta pressão no século XIX, Leveson [27] destacou o tema recorrente do risco e oportunidade de uma grande inovação tecnológica. No caso das máquinas a vapor, a utilização de cada vez mais pressão nas caldeiras, para obtenção de maior potência, resultou em vários acidentes terríveis decorrentes de explosão das caldeiras. Considerando as lições aprendidas dessa história da tecnologia das máquinas a vapor de alta pressão, Leveson sugere que há, basicamente, duas formas básicas para lidar com esse tipo de questão.

A primeira é aplicar ainda mais um princípio de engenharia, de centenas de anos, que sobrevive ao teste do tempo: manter as coisas simples e aumentar a complexidade daquilo que estamos tentando construir lentamente, à medida que aprendemos com a experiência. A segunda é que nós reduzamos, drasticamente, nosso entusiasmo e confiança nos computadores. A primeira opção parece ser a mais factível.

2.2.2. Complexidade dos projetos

A suposta flexibilidade que o software oferece pode levar à incorporação de muita funcionalidade supérflua, motivado pela crença de que esse acréscimo é barato. Leveson chama esse fenômeno de "maldição da flexibilidade" do software [15].

Por exemplo, o sistema operacional WINDOWS atual tem 50 milhões de linhas de código e cresce em torno de 20% a cada *release*. Ele é mantido por quase 10000 pessoas, vem em 34 línguas, é vendido em 137 países, e tem que funcionar com mais de 190000 dispositivos – diferentes modelos de câmeras digitais, impressoras, etc.

Software parece não ter os mesmos tipos de limites naturais que os materiais das outras engenharias. Na realidade, segundo Leveson [55], limites existem mas eles seriam menos óbvios e mais relacionados às habilidades humanas – o limite de complexidade com a qual o humano pode lidar. Leveson sugere que devemos nos impor

uma disciplina e controlar a complexidade, assim como melhorar a comunicação entre os engenheiros.

De acordo com Dijkstra [56], apesar de sabermos do problema da complexidade descontrolada, não sabemos que grau de simplicidade pode ser obtido (complexidade intrínseca vs. complexidade acidental), nem em que grau a complexidade intrínseca de todo o projeto precisa aparecer nas interfaces. Em software, a própria noção de complexidade não está bem estabelecida. Métricas para complexidade já foram propostas, mas elas não são consensuais.

A funcionalidade excessiva pode aumentar a complexidade do sistema e tornar difícil entender e raciocinar sobre as suas propriedades. No caso de sistemas de segurança crítica, nos quais é necessário antever propriedades de segurança do sistema antes dele ser testado ou entrar em operação, pode ser difícil determinar se o objetivo de segurança foi atingido.

Quando o software é modificado pode ser difícil saber se a modificação vai produzir um efeito indesejado, porque, se o sistema é complexo, poucos entendem seu funcionamento de forma completa e partes do sistema podem estar acopladas de forma não antecipada.

Os operadores também podem ter dificuldade em controlar todas as situações normais e anormais em um sistema muito complexo. Se o sistema é muito acoplado, os operadores não têm muita margem de manobra.

Freqüentemente, o projetista acaba deixando para o operador humano tudo aquilo que não conseguiu automatizar ou, então, tenta "eliminar" o humano para reduzir a vulnerabilidade do sistema ao erro humano. Mas, em muitos casos, o humano não é eliminado, mas deslocado para uma posição de supervisão. Se a automação é mal projetada, quando o humano tem que intervir, ele pode estar muito alheio ao processo. Por exemplo, é comum ocorrerem situações em que o operador, simplesmente, não sabe o que o computador está fazendo. Essas situações constituem um exemplo de um fenômeno mais geral, conhecido na área por confusão de modo (*mode confusion*). Em sistemas de segurança crítica é muito importante analisar o potencial de confusão de modo (*mode confusion potential*) de uma interface homem-máquina.

Quando um helicóptero SEA KING caiu sobre o *deck* do destróier canadense *Iroquis* [57] os dois primeiros sistemas de controle de incêndio falharam e apenas um terceiro funcionou. O segundo sistema falhou porque o operador não sabia que o mesmo botão que acionava o sistema também o desligava. O operador apertou o botão, seguidas vezes, num total de 6 vezes e estava, sem que percebesse, ligando e desligando a cada nova tentativa.

2.2.3. Comportamento discreto

Um terceiro fator que parece contribuir para tornar o uso de software mais complicado, é que software apresenta um comportamento discreto, ou seja, que assume certos valores e não é contínuo.

Software pode falhar em qualquer lugar do espaço de entradas. Ao contrário do que acontece com os materiais físicos das engenharias tradicionais, em software não há a noção de um comportamento contínuo para uma faixa de estímulos (intervalo de entradas), porque o comportamento do software é discreto, depende do programa feito por uma pessoa e não de uma lei física externa que nos permita apelar para continuidade de comportamento.

Finelli *et al.* [58] mostraram, e esse é um resultado extremamente importante nessa área e que deveria ser mais conhecido pelos praticantes, que quantificar confiabilidade do software na região ultracrítica (taxa de falha por hora $< 10^{-7}$) é inviável porque é necessário um tempo de testes proporcional ao tempo que se deseja observar a operação sem falhas ($\sim 10^{7}$ horas, para uma confiança de 99%). Assim, demonstrar uma taxa de falha de um software na ordem de 10^{-9} falhas por hora iria requerer mais de 114000 anos de teste ou 1000 sistemas em paralelo (independentes) durante 100 anos! Isso tem a ver com o fato de estarmos lidando com sistemas discretos. Extrapolar a partir de testes incompletos, não é considerado justificável devido à falta de continuidade.

Littlewood [59] sugere 10^{-3}, ou até 10^{-4}, como sendo números plausíveis para confiabilidade de software em aplicações críticas, mas por outro lado, lembra que, em geral, essas estimativas costumam ser fornecidas sem a declaração complementar do intervalo de confiança. Qual a confiança para declarar $p = 10^{-3}$? É a mesma para declarar 10^{-4} ?

Leveson [15] afirma que em muitas análises de segurança por árvore de falhas que ela já viu, a probabilidade de falha do software é, arbitrariamente, fixada em 0 (não tem falha), ou em número magicamente derivado. Ainda há a questão de quão realistas são o ambiente de testes e as premissas usadas nas avaliações, geralmente não explicitadas.

Devido ao fato das falhas de software serem sempre causadas por erros no projeto – se o software tem um defeito, sempre que a condição de manifestação do defeito for verificada, ele se manifestará - e por isso, serem ditas sistemáticas, uma alternativa tem sido analisar, matematicamente, modelos do sistema, através de análises ditas estáticas, assim denominadas por serem realizadas sobre um modelo e não sobre o software em execução. No caso de sistemas de segurança crítica, essas análises são usadas para mostrar que comportamentos considerados perigosos não podem ocorrer.

A matemática pode, de fato, nos ajudar a saber sobre as propriedades daquilo que está em construção. De acordo com Ladkin [60], avaliar e garantir a segurança de artefatos e procedimentos é, em grande parte, uma questão de prever o que pode acontecer no futuro. Segundo o filósofo Heidegger, *ta mathémata* significa para os gregos aquilo que o homem sabe de antemão quando observa aquilo que é e na sua relação com as coisas.

De acordo com a FAA (*Federal Aviation Administration*) [61], métodos formais consistem em um conjunto de técnicas e ferramentas baseadas em modelagem matemática e lógica formal que são usadas (em engenharia de software) para verificar, matematicamente, que o projeto e implementação do sistema satisfazem propriedades funcionais e de segurança.

Wordsworth [62] afirma que métodos formais não são métodos para desenvolver software sem ter que enfrentar decisões difíceis, mas métodos para registrar adequadamente essas decisões uma vez tomadas. Além disso, eles nos permitem explorar as conseqüências das decisões antes de torná-las irrevogáveis, ou muito caras para reverter, e eles podem sugerir algumas decisões que possam passar despercebidas. Eles nos ajudam, a cada estágio, a nos perguntarmos que decisões foram tomadas e quais foram postergadas, e, talvez, encorajarão a registrar os motivos para tomar, ou postergar, decisões.

Jones [63] sugere que a possibilidade de raciocinar sobre programas de computadores foi sentida como necessária já pelos pioneiros da computação eletrônica, tais como Von Neumann ou Goldstine. Até mesmo Babagge escreveu sobre a verificação das fórmulas inseridas nos cartões operacionais. Assim, a necessidade de raciocinar sobre programas e sua correção parece ter surgido junto com a criação dos primeiros programas.

Entretanto, a verificação da correção de programas pode ser impossível e um desperdício, de acordo com De Millo *et al.* [64] e Fetzer *et al.* [65]. Leveson [15] prefere dizer que ou é impossível, ou ainda está muito longe.

Segundo Fetzer, é importante fazer a distinção entre algoritmos e programas. Algoritmos até poderiam ser provados corretos por serem estruturas lógicas bem definidas e estáticas, guardando alguma semelhança com os teoremas da Matemática. A despeito da dificuldade e confiablidade dessas provas, elas até podem ser obtidas. Por outro lado, os programas que implementam os algoritmos num computador, dependeriam de relações causais com vários outros objetos para produzir as saídas desejadas, além de estarem sendo constantemente alterados, e por isso, jamais poderiam ser garantidos no sentido absoluto. Fetzer *et al.* chegam a recomendar que não sejam desperdiçados recursos tentando o impossível.

Um exemplo marcante dessa dificuldade de raciocinar sobre o comportamento do software pode ser ilustrado por um programa, de poucas linhas, aparentemente simples, mas para o qual até hoje não foi provado se ele sempre termina para qualquer entrada N, conforme afirma Harel [66]. O problema está em aberto há mais de 60 anos. Esse exemplo é considerado um dos problemas de formulação mais simples, porém ainda insolúvel, da matemática. O programa está descrito na figura 1.1.

```
Programa "3x+ 1"
    Receba N
    Enquanto N não for 1
        Se N for par, faça N = N / 2
        Se N for ímpar, faça N = 3 * N + 1
    Fim-Enquanto
```

Fig. 2.1. Programa "3x + 1"

A execução para N=7, é: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. O fim é atingido. Mas esse programa pára para todo N ?

Além de ser complicado provar a correção de programas, ainda que se prove, quem prova a prova ? No caso do programa "3x + 1", é possível restringir o N de entrada e testar o programa nessa faixa restrita para obter uma garantia limitada de parada, mas e se essa faixa precisar ser grande ? Quanto tempo de teste será necessário ?

Alguns autores sugerem que a formalização ajudaria a impor uma disciplina, ou rigor, que levaria o analista a considerar o sistema em maior profundidade e, atualmente, considera-se que o maior benefício de métodos formais seja oriundo da disciplina que eles impõem no especificador, que seria levado a pensar em mais detalhes, devido à necessidade de formalização. Métodos formais também permitem automatizar parte das análises de segurança. Por exemplo, um programa pode verificar, automaticamente, se todo estado de um diagrama de estados tem uma resposta definida para todo evento do sistema, ou se uma propriedade permanece invariante num *loop*. Um dos grandes benefícios de métodos formais pode ser a introdução de uma disciplina no desenvolvimento de software, proporcionando especificações mais completas e consistentes.

Entretanto, nem tudo poderá ser formalizado. Leveson [67] argumenta que, embora métodos formais sejam cruciais no desenvolvimento de sistemas de segurança crítica, a maior parte de uma especificação será informal. Além disso, aquele tipo de método formal que usa notação matemática que obscurece a especificação pode ser mais prejudicial que benéfico porque se a linguagem de especificação é obscura para aqueles que não são matemáticos, mas que são especialistas do domínio da aplicação, pode ser difícil para esses especialistas "leigos" participarem da importante revisão dos requisitos.

Erros na especificação de requisitos são mais caros de corrigir e representam grande parte dos problemas em sistemas de segurança crítica. Seria extremamente prejudicial se uma barreira de linguagem fosse estabelecida aí.

Segundo Leveson [15], além de critérios mais ou menos formalizáveis como completude e consistência, cuja verificação pode ser automatizada em certo grau,

sempre vão existir critérios de validação específicos do domínio que vão requerer a participação de especialistas humanos.

2.3. Conclusão

Neste capítulo, consideramos exemplos de sistemas de software de segurança crítica e alguns acidentes e incidentes já ocorridos nesses sistemas, ilustrando a importância atual dessa questão. Mostramos como esses sistemas vêm assumindo papéis cada vez mais importantes na sociedade, estando presentes de forma vital em quase todos os seus segmentos. Conforme afirma Johnson [114], em um certo sentido, lucro e *confiança* são os lubrificantes primários da economia do mundo moderno.

Também consideramos porque software vem sendo usado nesses sistemas e quais as dificuldades que seu emprego apresenta. Consideramos três espécies de dificuldades em particular: (1) o alto grau de inovação e (2) complexidade dos projetos envolvendo software e o (3) comportamento discreto do software.

No próximo capítulo iremos caracterizar segurança e escolher a abordagem mais apropriada para construção desses sistemas de software de segurança crítica, bem como apresentar os principais conceitos da abordagem escolhida.

CAPÍTULO III - CONCEITOS

3.1. Diferença entre *safety* e *security*

As palavras em inglês *safety* e *security* são, em geral, traduzidas ambas para segurança, mas em sistemas críticos, elas significam coisas diferentes. Esse problema não é exclusivo da nossa língua. Mesmo em inglês, que parece ser uma das poucas línguas que dispõe de duas palavras para segurança, é razoável supor que um leigo não saberia distinguir bem os termos.

Safety pode ser definida como a ausência de acidentes [15]. Acidentes, por sua vez, são eventos indesejados que podem envolver a perda de vida humana, danos ao meio ambiente ou danos materiais expressivos. Acidentes podem ocorrer mesmo devido a ações bem intencionadas ou, ainda, quando tudo está funcionando conforme especificado. Além disso, um sistema pode falhar, ou parar em um estado seguro, de forma que segurança não é a mesma coisa que confiabilidade.

Ladkin [60] nota que um acidente pode ser praticamente qualquer coisa. Usualmente, estaremos mais preocupados se a operação do sistema pode matar ou machucar alguém, mas pouco da engenharia de segurança depende desse ser o comportamento, em particular, que será considerado um acidente.

De acordo com Lamport [72], existem, em geral, dois tipos de propriedades que alguém freqüentemente deseja que um programa (concorrente) satisfaça:

- propriedades de segurança (*safety properties*): que afirmam que alguma coisa "ruim" nunca acontece, ou seja, que o programa nunca entra em um estado indesejado;
- propriedades de vivacidade (*liveness properties*): que afirmam que alguma coisa boa eventualmente acontece, ou seja, que o programa eventualmente entra em um estado desejado.

Já *security* enfoca ações maliciosas. O seu foco primário, tradicionalmente, tem sido prevenir o acesso não autorizado à informação [15].

Uma alternativa de tradução, porém pouco utilizada, seria usar *segurança-inocuidade* para *safety* e *segurança-confidencialidade* para *security*, considerando os

termos em francês usados pelo IFIP WG 10.4 (*Dependable Computing and Fault Tolerance)*, respectivamente *securité-innocuité* e *securité-confidentialité*. Uma outra possibilidade de tradução seria usar *segurança contra acidentes* para *safety* e *segurança contra ameaças* para *security*, conforme sugere Rogério de Lemos [68].

No que diz respeito a este trabalho, é importante destacar que o enfoque do mesmo é em segurança contra acidentes (*safety*) e sistemas de segurança crítica (*safety critical systems*). Assim, daqui em diante, a distinção só será feita quando necessária e falaremos simplesmente em segurança.

Cabe ressaltar que a distinção entre os termos é importante porque: (1) sistemas críticos envolvendo ambas as qualidades críticas vêm sendo construídos e (2) apesar de haver bastante possibilidade de compartilhamento de técnicas, como as qualidades não são a mesma coisa, as técnicas, eventualmente, podem conflitar.

Por exemplo, segundo Knight [69], existe um grande potencial para ataques do tipo negação de serviço nas comunicações da aviação. O TCAS, por exemplo, depende dessas comunicações. Assim, nesse caso, ambos os tipos de segurança seriam necessários: segurança para evitar colisão entre aeronaves e segurança das comunicações entre os *transponders* das aeronaves, já que a informação dos *transponders* é fundamental para o sistema calcular e recomendar as manobras para aumentar a separação.

Rushby [70] nota que, a despeito de algumas incompatibilidades, de uma forma geral, e talvez, a não ser no caso da segurança forte (*strong security*) que não toleraria canais cobertos (*covert channels*), as técnicas para ambos os tipos de segurança seriam amplamente compatíveis.

Além de potencialmente conflitar com outros objetivos de um sistema, segurança em excesso também pode gerar dependência excessiva ou estagnação (muita coisa não pode acontecer). O avião mais seguro não voa.

3.2. Abordagens para segurança

Mesmo dentro da área de segurança (*safety*) diferentes abordagens são usadas para construção de sistemas. Há, também, uma tendência internacional no sentido de definir uma estratégia global para sistemas de segurança crítica, ainda que

especificidades por indústria sejam entendidas como necessárias. Essa parece ser a abordagem da recente norma internacional IEC 61508. Acontece que a abordagem da IEC 61508 não é consensual.

Se considerarmos a sugestão de Rushby [70] de que haveria quatro abordagens para sistemas críticos: dependabilidade, segurança (*safety*) de sistemas, segurança (*security*) e tempo real; para sistemas de segurança crítica haveria duas: dependabilidade e segurança de sistemas. A abordagem da dependabilidade está bem descrita em [71] por Laprie. A abordagem da segurança de sistemas em [15] por Leveson. Rushby comenta ambas em [70].

A seguir, serão consideradas as abordagens da dependabilidade e da segurança de sistemas, e definida a escolha de abordagem da tese.

Basicamente, essas duas grandes abordagens dentro da área de segurança, a da dependabilidade e a da segurança de sistemas, se dividiriam na interpretação da relação entre falhas, perigos e acidentes.

3.2.1 Confiabilidade e dependabilidade

O termo *dependabilidade* tem sido usado para reunir, sob um único conceito global e abstrato, várias qualidades (*safety*, *security*, confiabilidade, disponibilidade e outras).

Segundo Rushby [70], o termo foi criado por Jean-Claude Laprie na tentativa de escapar dos significados técnicos especializados que, segundo ele, ficaram associados a termos tais como confiabilidade, e ter um termo para uma abordagem geral, que pudesse reunir diversas qualidades e técnicas que tinham sido, até então, consideradas em separado. Dependabilidade é uma medida do grau de confiança que se poderia depositar no serviço oferecido por um sistema.

A preocupação com dependabilidade parece querer responder à crescente conjugação de qualidades críticas nos novos sistemas.

No grupo da dependabilidade, uma falha é definida em relação à função, ou serviço do sistema e não à sua especificação [71]. Um sistema também falha quando ele atende à sua especificação, mas não atende ao seu propósito. Nesse caso, a especificação estaria errada e precisaria ser corrigida. Esse grupo pode argumentar que,

dessa forma, eles mantêm o usuário como a referência da falha e não um documento de especificação. Segundo Littlewood [42], definir confiabilidade em relação à especificação pode ter conseqüências paradoxais: se os requisitos estiverem imprecisos, ou até nem especificados, a confiabilidade não poderia ser medida. Assim, a referência tem que ser o usuário. Falhas ainda podem ser classificadas em benignas ou catastróficas.

O grupo de dependabilidade mantém uma associação forte entre falhas e acidentes, e assim, afirma que um sistema seguro é aquele livre de falhas catastróficas.

3.2.2 Segurança de sistemas

A abordagem da segurança de sistemas é oriunda da Guerra Fria. No final da Segunda Guerra Mundial, os americanos começaram a se preocupar com a possibilidade dos mísseis intercontinentais explodirem nos seus silos e precisaram criar novos conceitos em segurança, já que eles perceberam que os problemas que estavam sendo experimentados não tinham mais a ver com confiabilidade de componentes e sim com interações não antecipadas entre os componentes de um sistema.

Na abordagem da segurança de sistemas há uma certa dissociação entre falha e acidente. Nem todo perigo, definido como uma situação que pode levar ao acidente, é causado por falha e nem toda falha resulta em um perigo [15].

Um sistema confiável nem sempre é seguro e vice-versa. Por exemplo, uma pistola pode ser muito confiável, porque ela sempre atira quando solicitada, mas insegura na mão de uma criança. Por outro lado, um sistema pode ser muito seguro e pouco confiável, basta que ele falhe em um estado seguro (*fail-safe*). Por exemplo, sinais de trem que falham na posição "parar", por gravidade.

De acordo com Ladkin [70], confiabilidade e segurança podem estar relacionadas quando a segurança depende do funcionamento de um dispositivo de proteção. Mas Leveson [15] frisa que dispositivos de proteção deveriam ser os últimos recursos a serem empregados na ordem de precedência da segurança, devendo-se dar preferência a eliminar os perigos.

Segundo Rushby [70], o enfoque desse grupo é em segurança (acidentes, conseqüências externas) e não confiabilidade (falhas, serviço do sistema), já que essas

qualidades podem conflitar. Além disso, Leveson [15] afirma que o serviço prestado pelo sistema pode ser a fonte mesma das conseqüências indesejáveis. A maioria dos acidentes envolvendo software está relacionada a erros nos requisitos do software. Nessa abordagem, a noção de perigo (*hazard*) assume maior importância que a noção de falha.

Segundo Leveson [73], a abordagem da segurança de sistemas não requer que os componentes do sistema demonstrem ultra confiabilidade, apenas que um conjunto de comportamentos perigosos não ocorram. Para software essa distinção é crítica: é relativamente muito mais fácil projetar software para prevenir determinados comportamentos que garantir que ele sempre fará a coisa "certa". Na verdade, essa última meta pode ser impossível.

Ainda segundo Leveson [73], muitos acidentes têm ocorrido porque os engenheiros de software desconhecem ou não aplicam princípios básicos e já conhecidos de segurança de sistemas, ou utilizam abordagens inapropriadas para sistemas de segurança crítica.

3.2.3 Escolha da abordagem apropriada

Essas diferenças conceituais vão acabar por influenciar as práticas em ambos os grupos. Esses grupos seriam irreconciliáveis ?

Para o grupo de dependabilidade, segurança vai depender de quanto podemos confiar em funções de segurança, daí o termo integridade de funções de segurança que é a chance de falha de uma função de segurança. As funções de segurança do sistema precisam ser elicitadas e alocadas aos componentes (hardware, software e humanos) do sistema.

Já para o grupo de segurança de sistemas, segurança é uma propriedade emergente do sistema (ao invés de componentes) operando no seu ambiente. Grande ênfase é dada na identificação e eliminação dos perigos para obter segurança. Nem tudo pode, ou precisa, ser quantificado. Há uma preferência por análises qualitativas. Muitas vezes, simplesmente saber que um perigo existe pode ser suficiente para eliminá-lo. Os perigos poderiam ser eliminados através de um projeto seletivo, por exemplo, evitando materiais ou condições perigosas. Quando não é possível eliminar o perigo, outras

estratégias de mitigação devem ser adotadas, mas seguindo sempre a chamada ordem de precedência para segurança: (1) eliminar o perigo, (2) reduzir o perigo, (3) controlar o perigo ou (4) reduzir o dano. Grande ênfase é dada na identificação e eliminação dos perigos para obter segurança. Uma técnica conhecida como análise de perigos é central nessa abordagem.

Aparentemente, o primeiro grupo, da dependabilidade, concordaria (claro) e também atuaria no sentido de eliminar os perigos, antes de definir as funções de segurança, e assim, não veria as abordagens como mutuamente excludentes. Note que se o perigo é eliminado, não há mais necessidade de uma função de segurança, mas elas sempre seriam necessárias porque, realisticamente, nem sempre será possível eliminar todos os perigos. Apesar disso, é interessante notar que a IEC 61508, de certa forma alinhada com a abordagem da dependabilidade, não diz muita coisa sobre a eliminação de perigos, conforme alertou Leveson, que considera essa norma um grande retrocesso [74].

O grupo da abordagem da segurança de sistemas não acha que as abordagens sejam combináveis porque considera segurança e confiabilidade, qualidades distintas e que podem até conflitar, lembrando, ainda, que nossos recursos são finitos e nossas escolhas não são tão gratuitas assim. Avanços tecnológicos recentes teriam alterado o perfil dos acidentes nos sistemas mais complexos que estamos construindo ou pretendemos construir. Isso seria ainda mais forte naqueles sistemas que empregam software.

Segundo Leveson [15], a construção de muitos dos sistemas complexos de hoje requer a integração de partes construídas por entidades ou contratados separados. Mesmo que cada contratado, ou grupo, adote medidas para embutir qualidade em seus componentes, a combinação de subsistemas em um sistema introduz novos modos de falha e perigos que não estão aparentes quando as partes são consideradas separadamente.

Leveson sugere que as qualidades, ou pelo menos a segurança, seja considerada em separado para melhor controlá-la, entendê-la, alocar recursos limitados e estabelecer prioridades. Leveson argumenta que quando é utilizada uma única medida global do tipo *dependabilidade*, que agrupa qualidades que podem conflitar, é possível aumentar

essa *dependabilidade* e, ao mesmo tempo, diminuir a segurança (*safety*) sem que esteja aparente que esse aumento do risco ocorreu.

Por exemplo, em algumas aplicações, o sistema simplesmente não pode parar, se ele pára, a segurança diminui, por não haver um estado seguro de parada. O sistema pode precisar ter que continuar funcionando ainda, que suavemente degradado (*graceful degradation*). Nesse caso, além de segurança, é necessária alta disponibilidade. Mas, técnicas para obtenção de degradação suave podem aumentar muito a complexidade do sistema e acabar por diminuir a segurança, que se beneficia da simplicidade, determinismo e previsibilidade. A abordagem da dependabilidade poderia tornar esses tipos de conflito menos visíveis, segundo Leveson.

No início da década de 80, Perrow [23] sugeriu o conceito de "acidentes normais" para referenciar acidentes em sistemas complexos que ocorreriam devido a interações não antecipadas, e talvez não antecipáveis, entre os componentes do sistema. Os componentes estariam fazendo exatamente o que foi especificado e, talvez, até o esperado, mas o resultado do conjunto é desastroso devido a características do sistema, daí o termo acidentes normais (para aquele sistema). Quando o acidente é investigado, o papel do contexto e causas raízes bem longe do espaço e tempo do acidente (condições latentes) podem ser reconhecidos. Em suma, em sistemas complexos, acidentes tendem a ser complexos.

Assim, o segundo grupo olha para o primeiro grupo como obsoleto e talvez perigoso. Técnicas para aumentar a confiabilidade (das funções de segurança) poderiam diminuir a segurança. É melhor eliminar perigos por construção, através de um projeto mais simples e bem pensado, que adicionar dispositivos de proteção a um projeto terminado. Dispositivos de proteção adicionados a um projeto terminado podem aumentar a complexidade do sistema (maior acoplamento e complexidade das interações) e diminuir a segurança.

Outra crítica contra a abordagem da dependabilidade é que ela poderia levar a perda de tempo tentando obter e medir integridade, quando esse mesmo tempo poderia estar sendo melhor empregado para identificar e eliminar perigos ainda não conhecidos. Assim, a abordagem da dependabilidade poderia levar à perda do foco do que interessa

para a segurança – os perigos. Dúvidas também poderiam ser colocadas com relação a até que ponto seríamos capazes de medir a integridade de software e de pessoas.

Rushby [70] observa que a abordagem da dependabilidade pode ser mais adequada naquelas aplicações nas quais não há alternativa segura ao serviço normal (por exemplo *fly-by-wire*) e a da segurança de sistemas, nas aplicações nas quais existem eventos indesejados específicos (p. ex. sistemas de detonação de armas e a detonação inadvertida).

Leveson [15] afirma que a abordagem da segurança de sistemas é mais disseminada nos Estados Unidos do que na Europa, onde a abordagem para segurança tende a depender mais de confiabilidade.

Esta tese se baseia na abordagem da segurança de sistemas porque considera essa abordagem diretamente focada nos perigos que precisamos evitar em um sistema.

Assim, a tese considera segurança como uma qualidade bem definida - ausência de acidentes ou eventos indesejáveis, diferente de dependabilidade, confiabilidade, segurança (*security*) ou disponibilidade (*availability*), e que precisa ser tratada cedo e durante todo o projeto, quando os perigos podem ser mais facilmente eliminados, e de forma independente, uma vez que ela potencialmente conflita com outras qualidades.

3.3 Sistema

Sistema é um conjunto de componentes (software, hardware, pessoas) que atuam em conjunto como um todo para atingir um objetivo comum. O estado do sistema em um determinado instante de tempo é o conjunto de propriedades relevantes que descrevem o sistema nesse instante. O ambiente do sistema é um conjunto de componentes que não são parte do sistema mas cujo comportamento pode afetar o estado do sistema [15].

Segundo Ladkin [60], um método característico de estipular a fronteira entre o sistema e o ambiente é considerar que características do universo se pode ter mais ou menos controle, e quais não, e tomar a decisão baseado nisso.

A fim de que os componentes do sistema atinjam os objetivos globais do mesmo, não apenas as interfaces entre os componentes precisam ser consistentes, mas os

componentes precisam estar em acordo na ordenação das ações. Dois conceitos básicos em teoria de sistemas são: emergência e hierarquia.

Segurança é uma propriedade emergente de um sistema operando em seu ambiente. Não tem sentido falar da segurança de um componente, mas sim de como um componente pode contribuir para ocorrência de um acidente.

Segundo Allen [75], a arquitetura ou estrutura de um sistema define o sistema em termos de seus componentes e suas interações. Fornece um modelo de alto nível do sistema que suprime detalhe de implementação, permitindo ao arquiteto se concentrar nas análises e decisões que são mais cruciais para a estruturação do sistema a fim de satisfazer seu propósito. Permite tratar propriedades ao nível de sistema, como segurança.

Quando a noção de arquitetura de um sistema específico é generalizada, surge o conceito de estilo de arquitetura. Por exemplo, um estilo de arquitetura para sistemas de tempo real é conhecido como executor cíclico. Um executor cíclico é responsável por executar um conjunto de comandos, ou acionar um conjunto de componentes de forma periódica e determinística.

A arquitetura do sistema é importante não apenas para a facilidade de implementação e teste do sistema, mas também para mantê-lo. Tem havido uma ênfase cada vez maior na Engenharia de software sobre a importância de se pensar sobre a arquitetura de um sistema. Pfleeger afirma que a qualidade da arquitetura de um sistema pode fazer, ou quebrar, um sistema [76]. Por exemplo, uma arquitetura pode permitir que mudanças no projeto sejam acomodadas com facilidade, ou não.

Muitas medidas de segurança podem ser implantadas ao nível da arquitetura do sistema. Por exemplo, em sistemas de segurança crítica é recomendável que os componentes do sistema possuam baixo acoplamento. Se os componentes estão fortemente acoplados, problemas podem se espalhar rapidamente pelo sistema e dificultar a reversão ou controle de uma situação problemática. As interações entre os componentes do sistema também devem ser de preferência simples, planejadas e previsíveis. Desta forma, pode ficar mais fácil entender o comportamento do sistema. Os componentes críticos devem ser simples e isolados do resto do sistema a fim de simplificar análises de perigo e reduzir o esforço para demonstrar sua segurança.

Entretanto, segundo Allen [75], a arquitetura de um sistema, freqüentemente, aparece em descrições de sistemas como diagramas "caixas e linhas". A falta de informação limita severamente a utilidade desses diagramas. Sem informação específica sobre as interfaces entre os componentes, e sem informação suficiente para determinar o significado da composição de diferentes elementos, os implementadores são forçados ou a adivinhar as intenções do projetista, praticamente refazendo o trabalho de projeto, ou a continuamente consultar o desenvolvedor original, tornando esta pessoa um gargalo no desenvolvimento e criando uma fonte significativa de riscos. Um modelo formal pode prover a base para uma análise rigorosa e justificável de propriedades críticas do sistema [75].

Uma base formal não significa que a forma da linguagem tenha que ser matematicamente obscura. Seria possível e crucial ter uma linguagem próxima dos usuários ou especialistas da aplicação que poderão validar a arquitetura proposta e, ao mesmo tempo, essa linguagem ter uma base formal que permita introduzir rigor matemático na descrição e automatizar as análises. Leveson [77] sugere que a RSML (*Requirements Specification and Modelling Language*), parcialmente baseada em Statecharts, proposta por Harel, é um exemplo de uma linguagem assim. A UML tem sido criticada na área de sistemas de segurança crítica, em parte porque ainda não tem uma semântica formalmente definida.

3.4 Sistema de segurança crítica

Segundo Leveson [36], um sistema é considerado de segurança crítica se um mal funcionamento do sistema pode envolver dano para vidas humanas, propriedades, ou para o ambiente. Alguns autores, tais como Kjellén [78], incluem reputação também. Uma caracterização mais detalhada sobre como deve ser entendido esse termo "mau funcionamento" será feita mais adiante.

Leveson [77] afirma que em sistemas de segurança crítica, segurança pode ser o objetivo, ou parte dos objetivos, de um sistema, ou ainda, parte das restrições que um sistema precisa respeitar. Quando é parte dos objetivos, uma das funções do sistema é garantir que acidentes não ocorrem. Por exemplo, segurança como objetivo de um sistema anticolisão utilizado a bordo de uma aeronave. Quando é parte das restrições, o sistema precisa atingir seu objetivo, pode ser segurança de novo, sem que acidentes

ocorram. Por exemplo, o mesmo sistema anticolisão do exemplo anterior não deveria sugerir uma manobra para o piloto que conflitasse com a gerência do espaço aéreo, realizada pelo controle de tráfego aéreo em terra.

Segundo Hermann [4], software em sistemas de segurança crítica pode ser de dois tipos básicos:

- software de segurança crítica (*safety-critical software*);
- software relacionado à segurança (*safety related software*).

Software de segurança crítica realiza ou controla funções que, se mal executadas, podem infligir danos. Software relacionado à segurança realiza ou controla funções que são ativadas para prevenir ou minimizar o efeito de um mau funcionamento de um sistema de segurança crítica. Algumas normas, entretanto, utilizam o termo software relacionado à segurança para ambos os tipos, por exemplo a IEC 61508.

A fim de auxiliar na determinação se um sistema é ou não é de segurança crítica, algumas normas, como por exemplo a norma australiana [53], definem algumas categorias de sistemas de segurança crítica. Por exemplo, qualquer sistema que controle ou parcialmente controle o movimento de um veículo (navio, avião, etc) é um sistema de segurança crítica. Qualquer sistema que coleta, armazena, manipula, e relata ou exibe dados que possam ser de natureza de segurança crítica é um sistema de segurança crítica.

Um sistema que não controle ou interaja com quaisquer dispositivos físicos, com exceção da plataforma na qual ele é executado, e que não processe dados críticos, não é de segurança crítica.

Quando persiste a dúvida se um sistema é ou não é de segurança crítica, a solução é conduzir uma análise de perigos preliminar (*preliminary hazard analysis*). A análise de perigos será vista mais adiante.

Em sistemas de segurança crítica, o termo mal funcionamento precisa ser entendido de forma um pouco mais abrangente, não estando unicamente associado à ocorrência de falhas. A seguir, serão vistas as noções de falha, acidente, incidente, perigo e risco.

3.5 Falha

Uma falha do sistema ou de um componente ocorre quando o serviço prestado pelo sistema ou componente difere do serviço especificado. Confiabilidade é a probabilidade que um componente ou sistema irá realizar a sua função pretendida de forma satisfatória por um tempo determinado e sob circunstâncias ambientais estipuladas [15].

Uma falha é dita sistemática quando é devida a erros de projeto. O sistemático vem da constatação de que dada uma condição específica, a falha sempre vai se manifestar até que o projeto seja corrigido. Software não desgasta, apesar de ele poder "degradar" após muitas manutenções e, portanto, todas as falhas de software são sistemáticas [15]. Os erros lógicos num projeto de software podem ocorrer porque: (1) o software foi escrito para satisfazer requisitos incorretos ou (2) os requisitos não foram implementados corretamente.

De acordo com Leveson [15], a imensa maioria dos acidentes envolvendo software está relacionada a erros nos requisitos. Poucos acidentes ocorrem devido a erros de codificação. Além disso, a grande maioria dos acidentes em que software estava envolvido aconteceram quando o software estava fazendo exatamente ou mais do que estava especificado (erros de comissão) e não por ele ter deixado de fazer o que estava especificado (erros de omissão).

Uma falha também pode ser transiente ou aleatória, causada, por exemplo, por distúrbios no ambiente ou desgaste de peças ao longo do tempo. As falhas do hardware podem ser dos dois tipos citados.

Falha, ou erro humano, será discutida mais adiante nesta tese. O "erro" humano é particularmente interessante porque, conforme a psicologia cognitiva, os mesmos mecanismos cognitivos que provocam o erro são aqueles que nos permitem resolver problemas complexos. Erro humano faz parte do comportamento humano. Outro aspecto importante com relação ao erro humano é a influência do ambiente no comportamento humano.

3.6 Acidente e incidente

Um acidente é um evento indesejado e não planejado, não necessariamente inesperado, que envolve algum dano para vidas, ambientes ou propriedades.

Ladkin [60] nota que um acidente pode ser praticamente qualquer coisa. Usualmente, estaremos mais preocupados se a operação do sistema pode matar ou machucar alguém, mas pouco da engenharia de segurança depende desse ser o comportamento em particular que será considerado um acidente.

Um incidente é um evento que não envolve perda (ou uma perda mínima) mas com potencial para perda sob circunstâncias diferentes.

Em 1929, Heinrich publicou um clássico estudo de análise de 50000 acidentes industriais, concluindo que para cada dano sério que ocorreu após uma exposição a um perigo, houve 29 danos menores e 300 incidentes envolvendo nenhum dano relatável. Ele sugeriu que milhares de incidentes menores (*near misses*) estariam ocorrendo também, hipótese que ficou conhecida como Pirâmide de Heinrich. Assim, um sistema de relato de incidentes deveria ser utilizado como uma poderosa arma para segurança, já que, em geral, acidentes serão precedidos por muitos alertas.

Atualmente, não é mais aceita a razão universal entre acidentes e incidentes (1:300) sugerida por Heinrich, mas ainda é dada muita importância ao relato de incidentes, já que eles nos permitem fazer alguma coisa antes que acidentes sérios ocorram.

Falhas podem não resultar em acidentes e acidentes podem ocorrer mesmo quando todos os componentes do sistema operam com sucesso. Acidentes sérios já ocorreram com todos os componentes do sistema funcionando exatamente conforme especificado.

Um acidente pode ocorrer devido a interações "disfuncionais", não planejadas e perigosas entre os componentes de um sistema, incluindo pessoas interagindo com a máquina. Leplat [citado em 15], por exemplo, sugeriu dividir essas interações em dois tipos: (1) deficiências na articulação dos subsistemas e (2) falta de amarração (*linkup*) entre os elementos de um sistema. No primeiro caso, teríamos, por exemplo, zonas de

conflito. No segundo caso, teríamos, por exemplo, problemas com a tramitação de uma ordem.

Segundo Leveson [15], segurança é um complexo problema sócio-técnico. Os eventos que levam a um acidente podem formar uma complexa combinação de falha no equipamento, manutenção deficiente, problemas de instrumentação e controle, ações humanas, e erros de projeto. Particularmente, o "erro" humano não pode ser entendido com profundidade se ele é considerado isolado do ambiente no qual ele ocorre. Salvaguardas para prevenir erro humano que não consideram esses fatores serão limitadas em efetividade.

Muitas investigações de acidentes tendem a ignorar fatores organizacionais, parando quando encontram alguém (operador) ou alguma coisa para culpar [47]. Isso impede que ações mais eficazes sejam tomadas para prevenir futuros acidentes.

Assim, o enfoque em segurança é em situações perigosas ou, simplesmente, perigos, e, eventualmente, em falhas. Se o enfoque fosse apenas em falhas, muitos acidentes potenciais poderiam deixar de ser considerados.

3.7 Perigo

Um perigo (*hazard*) do sistema é um evento ou situação do sistema que, junto com outras condições do ambiente, vai levar a um acidente.

Em um sistema de portas automatizadas de um trem, os seguintes perigos podem ser identificados [15]:

- trem começar a andar com as portas abertas
- porta abrir com o trem em movimento
- porta abrir com o trem não alinhado com a plataforma da estação
- porta fechar com alguém atravessando a porta
- porta que fecha com obstrução não re-abre ou a porta reaberta não torna a fechar
- portas não podem ser abertas para uma evacuação de emergência.

Dado um perigo, a ocorrência ou não do acidente, em geral, vai depender de condições fora do alcance do projetista (condições do ambiente). Uma condição

perigosa pode anteceder várias configurações de acidente possíveis. O enfoque é no perigo, porque é o que está ao alcance do projetista eliminar ou controlar.

Uma análise de perigos permite identificar perigos, mecanismos, causas e possíveis conseqüências em maior detalhe e o sistema ser projetado para eliminar ou controlar essas situações (mitigá-las). Em um certo sentido, o projetista do sistema não pode evitar diretamente os acidentes, mas pode preveni-los, evitando as situações perigosas identificadas que podem levar a um acidente. No exemplo do diagrama de estados, o projetista deve tentar impedir que os estados perigosos do sistema sejam atingidos.

Severidade de um perigo é definida como o pior acidente possível que pode resultar do perigo dadas as piores condições do ambiente. A chance de ocorrência do perigo pode ser classificada qualitativa ou quantitativamente. A combinação de severidade das conseqüências e chance de ocorrência do perigo é, freqüentemente, denominado nível de perigo.

De acordo com a MIL-STD-882, severidade de um perigo pode ser classificada em:

- catastrófico;
- crítico;
- marginal;
- desprezível.

Já a chance de ocorrer, de acordo com a MIL-STD-882, pode ser classificada em:

- freqüente;
- provável;
- ocasional;
- remota;
- improvável.

A severidade do perigo e a sua chance de ocorrer podem ser combinadas em uma matriz, permitindo avaliar o nível de perigo. Nível de perigo é um dispositivo para

ajudar a determinar como priorizar o esforço para lidar com os perigos. Uma matriz de nível de perigo pode ajudar a decidir que perigos tratar e como.

No caso de software, ou sistemas que empregam muito software, nem sempre é possível determinar o nível de perigo, devido à dificuldade (ou impossibilidade) de se determinar probabilidade de falha (na região ultra crítica) e também ao alto grau de inovação dos projetos.

Leveson [12] argumenta que severidade apenas é freqüentemente adequada para se determinar o que fazer e quanto gastar em eliminar ou mitigar um perigo. Em alguns casos, segundo ela, é possível também estabelecer um critério qualitativo para se avaliar o nível de perigo potencial para auxiliar melhor nas decisões, mas isso vai depender de cada sistema.

Por exemplo, a MIL-STD-882C sugere que, no caso de software, severidade seja combinada com grau de controle desempenhado pelo software, ao invés da probabilidade de ocorrência do perigo. Como graus de controle desempenhados pelo software, podemos tomar a seguinte classificação adotada por Leveson em [15]:

- Grau I: computador fornece informação e conselho para o operador, talvez lendo sensores diretamente;
- Grau II: computador lê e interpreta dados de sensores para o operador;
- Grau III: computador interpreta e exibe dados para o operador e emite comandos;
- Grau IV: computador assume controle total sobre o processo com o operador supervisionando.

3.8 Risco

Risco, conforme sua utilização na abordagem da segurança de sistemas, é um número calculado em função da severidade e da probabilidade de ocorrência de um acidente e que serve para efetuarmos comparações.

Risco (conseqüência/tempo) =

Freqüência (eventos/tempo) x Magnitude (conseqüência/evento)

A definição acima foi obtida de Bahr [89].

A engenharia de segurança de sistemas coloca alta prioridade na eliminação de perigos por construção do sistema. O risco não precisa ser quantificado para que alguma medida possa ser tomada. Em muitos casos, um conhecimento mais qualitativo sobre os perigos é melhor que tentar quantificar, com cada vez mais precisão, perigos já conhecidos. Alguns fatores podem ser de difícil quantificação (por exemplo, fatores humanos), mas não devem ser ignorados.

Entretanto, a quantificação do risco não é inteiramente descartada. Segundo Leveson [15], a quantificação do risco pode ajudar a demonstrar que mudanças no projeto são necessárias para reduzir o risco e mostrar a razão de decisões de compromisso entre objetivos conflitantes.

3.9 Avaliação e aceitação do risco

Pfleeger *et al.* [79] lembram que nunca será possível terminar a análise de perigos, porque nunca se terá absoluta certeza que foram identificadas todas as formas nas quais as coisas podem sair errado. Entretanto, dado que os orçamentos e prazos são finitos, é importante gastar recursos com sabedoria na análise de perigos. O princípio guia deve ser aumentar a confiança de que o sistema vai continuar a fornecer serviços essenciais e não vai sofrer um acidente sério, no contexto de eventos plausíveis, dentro de custos razoáveis. Esses julgamentos devem ser realizados, de preferência, por vários especialistas.

Collins *et al.* [80] abordam a questão subjacente do "quão bom é bom o suficiente" e sugerem, entre outros princípios, o teste da publicidade, que consiste, basicamente, em perguntar se as nossas decisões soariam ultrajantes se precisassem ser defendidas por um público bem informado.

Rushby [citado em 43] sugere que, dado que os recursos são finitos, as análises mais grosseiras, que vão encontrar os problemas mais aparentes, devem ser realizadas primeiro. Análises cada vez mais sofisticadas poderiam ir sendo realizadas até o fim dos recursos alocados.

A norma internacional IEC 61508 recomenda a adoção do princípio conhecido por *ALARP* (*At Least As Reasonably Practible*) para aceitação de riscos. Segundo esse princípio, um risco pode ser intolerável, negligível ou cair na região ALARP. Quando o

risco cai na região ALARP, ele só será aceito se for demonstrado que todo esforço razoável foi feito para reduzi-lo e não é mais prática uma redução adicional, tendo em vista o custo *vs.* benefício. Além da noção de razoável no ALARP ser uma noção subjetiva, críticos do ALARP, ainda dizem que as medidas mais caras para segurança podem provocar a formação de argumentos baseados no ALARP para que sejam evitadas. As medidas mais baratas tenderiam a passar sem muito debate [8].

Perrow [23] sugere que a avaliação de riscos tem seus próprios riscos. Segundo ele, esse é um campo muito sofisticado no qual modelos matemáticos predominam ao lado de conceitos mais nebulosos, tais como o princípio ALARP. Nessa área há do xamanista ao cientista. Não seria exagero dizer que além da função de informar e aconselhar sobre os riscos, a avaliação de riscos também tem o papel, se o risco precisar ser assumido, de legitimá-lo e acalmar as pessoas. Nesse caso, a questão subjacente nem seria risco, mas uma questão de poder: o poder de uma elite impor riscos a muitos, para o benefício de poucos.

Littlewood [59] argumenta que é muito raro ser capaz de estabelecer, com completa certeza, que um argumento de dependabilidade sobre um produto é verdadeiro. Ainda assim, na falta de certeza, nenhum padrão ou regulamento parece requerer que o nível de confiança seja afirmado, ou mesmo qualquer orientação sobre como um tal nível seria calculado. Que nível de confiança é necessário para que um argumento seja aceito ? Quando se pretende dizer que a probabilidade de falha de um software é $p = 10^{-3}$, qual a confiança necessária para afirmar isso ? É a mesma para $p = 10^{-4}$? Qual a confiança necessária para afirmar um $p = 10^{-9}$? Chegar a $p = 10^{-9}$ com confiança é factível retrospectivamente ? E antes do fato ?

Leveson [15] sugere que os mais importantes fatores causais em termos de prevenção de acidentes são freqüentemente imensuráveis, ou de difícil medição, e o que não pode ser medido pode, por essa razão, acabar ficando de fora das análises. Na abordagem da segurança de sistemas, tempo é mais bem empregado identificando novas situações perigosas, que quantificando com cada vez mais precisão perigos já conhecidos. A aceitação de riscos, freqüentemente envolve questões trans-científicas e portanto, não é uma questão apenas de engenharia.

Em indústrias reguladas, a argumentação junto a uma autoridade reguladora (governamental) será obrigatória para se obter autorização para operar, ou modificar o sistema. Esse argumento precisará ser mantido durante toda vida do sistema. Quando acidentes ocorrem, a autoridade de segurança tem que ser comunicada.

Internacionalmente, verifica-se uma tendência em direção ao seguinte modelo. O projetista sendo responsável por argumentar sobre a segurança do sistema junto a uma autoridade reguladora e esta última sendo responsável por avaliar a argumentação, tendo em vista proteger o interesse público.

Redmill *et al.* [81] lembram que os riscos, eventualmente aceitos, devem ser revistos periodicamente, para que se utilize de novas tecnologias, conceitos e condições que permitam eliminar, ou melhor controlar, os perigos. De acordo com Leveson [15], a entrada em operação do sistema inevitavelmente revela situações não antecipadas e as análises de perigo devem ser revistas nesse momento.

3.10 Princípios básicos de segurança

Muitos princípios e recomendações podem ser aplicados no projeto de sistemas de segurança crítica e é muito importante que eles sejam conhecidos e utilizados. Alguns acidentes parecem ter ocorrido porque os engenheiros de software não conheciam, ou não utilizaram princípios básicos de engenharia de segurança.

Este item apresenta uma lista de doze princípios, conforme eles foram reunidos pela norma australiana [53]. Eles foram divididos em princípios gerenciais e técnicos, o que reflete as duas importantes dimensões da solução do problema da segurança.

Os princípios gerenciais são os seguintes:

1. Sistemas devem ser considerados de segurança crítica a não ser que demonstrado o contrário;
2. Questões de segurança devem ser abordadas cedo no ciclo de vida de desenvolvimento e acompanhadas de forma permanente;
3. Garantia da segurança requer visibilidade, tanto do produto, quanto do processo;
4. Segurança é mais bem obtida através de uma diversidade de pessoas, habilidades e técnicas;

5. Segurança requer o comprometimento e cooperação de todas as partes;
6. Argumentos de garantia de segurança requerem verificação e revisão independentes;
7. Garantia de segurança deve ser transferível (comunicável);
8. Segurança deve ser demonstrada por meio de argumento de segurança auditável e repetível.

Os princípios técnicos são os seguintes:

1. Segurança é mais bem obtida usando tecnologias testadas e provadas, quando possível;
2. Segurança é mais bem obtida por meio de um processo interativo, contínuo e evolucionário;
3. Segurança é mais bem obtida se funções críticas são tão simples quanto possíveis;
4. Garantia de segurança requer, nos casos mais críticos, o uso de métodos formais.

3.11 Conclusão

Neste capítulo, caracterizamos segurança e estabelecemos a diferença entre *safety* e *security*, explicitando porque é importante fazer a distinção. Em seguida, consideramos a melhor abordagem para sistemas de software de segurança crítica (*safety critical software systems*) e apresentamos os principais conceitos e princípios básicos da Segurança de Sistemas.

A abordagem da segurança de sistemas é mais indicada para sistemas de software de segurança crítica porque na maioria dos acidentes envolvendo software, este estava fazendo exatamente o que era requerido, e os acidentes ocorreram devido a interações disfuncionais entre os componentes do sistema. Nessa situação, não há muito sentido em aumentar a confiabilidade do software. Faz mais sentido tratar segurança como uma propriedade emergente do sistema, analisando e eliminando situações perigosas do mesmo.

No próximo capítulo, veremos o componente fundamental dos programas de segurança orientados segundo a abordagem da Segurança de Sistemas que é o Processo de Análise de Perigos.

CAPÍTULO IV - ANÁLISE DE PERIGOS

4.1 Processo de Análise de Perigos

De acordo com Pfleeger *et al.* [79], a análise de perigos é um conjunto de técnicas sistemáticas, porém informais, voltadas para expor estados do sistema potencialmente perigosos. A análise facilita a identificação de estratégias de prevenção ou mitigação, ou seja, ela expõe causas prováveis de problemas de forma que possam ser aplicadas técnicas para prevenir o problema ou suavizar suas conseqüências prováveis.

Leveson [15] afirma que, subjacente a qualquer análise de perigos, existe um modelo de acidente, explícito ou não, que determina o que é um acidente e o seu mecanismo. Modelos de acidente serão analisados no item seguinte.

As análises de perigos são comumente realizadas em três estágios:

- Análise de Perigos Preliminar (*PHA*)
- Análise de Perigos ao nível de Sistema (*SHA*)
- Análise de Perigos ao nível de Subsistema (*SSHA*)

No início do projeto, é importante realizar uma análise de perigos preliminar (*PHA*) para identificar funções críticas e já delinear alguns perigos do sistema. Desta forma, considerações de segurança são incorporadas cedo no projeto.

Em alguns países e indústrias, os perigos particulares a serem analisados e a profundidade e tipos de análise podem estar estabelecidos por autoridades reguladoras ou pela legislação. Para muitos perigos e sistemas, a análise pode consistir meramente em comparar o projeto com vários padrões e códigos que foram desenvolvidos ao longo do tempo para lidar com perigos conhecidos.

Entretanto, na medida que nova tecnologia é desenvolvida e sistemas crescem em tamanho, novos perigos surgem e a possibilidade de introduzir perigos aumenta. Sistemas com projetos novos e complexos podem requerer procedimentos analíticos documentados, sofisticados e formais. O Fórum de Riscos na Internet (Risks Forum) também pode ser uma fonte de consulta sobre problemas que ocorreram em sistemas no passado ou que já estão sendo antevistos.

À medida que mais informações sobre o projeto se tornam disponíveis, as análises podem ser aprofundadas. Análises de perigos podem ser realizadas ao nível de sistema (*SHA*) ou de subsistema (*SSHA*). Subsistema pode ser: software, hardware ou pessoas, ou uma combinação deles. No caso de pessoas, as análises também podem ser denominadas análises de tarefas do operador (*operator task analysis*) ou análise de segurança operacional.

A análise de perigos ao nível de sistema (*SHA*) determina como modos normais ou de falha dos componentes operando em conjunto podem afetar a segurança do sistema. A *SHA* não considera apenas os componentes de hardware de um sistema, mas hardware, software e pessoas interagindo. Ela pode ser usada para identificar tarefas humanas de alto-risco e erros, potencialmente, de segurança crítica do operador. Essa informação deve ser usada no projeto da interface homem-máquina e em procedimentos operacionais para reduzir erros e prover meios para os operadores responderem de forma apropriada à falhas e erros da automação.

A análise de perigos de subsistema (*SSHA*) procura identificar como a operação ou falha de um componente individual afeta a segurança geral do sistema – incluindo desempenho normal, degradação operacional, falha funcional, função não pretendida e função inadvertida (função correta, mas na hora ou ordem errada). No caso do hardware é importante considerar a influência das emissões eletromagnéticas (*EMI*), ou outros fatores externos que podem afetar a segurança. Os processadores usados em aplicações de segurança crítica costumam ter um *clock* bem mais baixo que os processadores topo de linha disponíveis para os usuários comuns.

Em qualquer análise de perigos, modos de operação anormais ou pouco freqüentes também devem ser considerados. É importante também considerar todas as fases do ciclo de vida e, eventualmente, identificar a fase na qual o perigo ocorre.

Um erro comum é deixar de fora das análises fatores que são de difícil quantificação (software e pessoas). Mas nem tudo pode, ou precisa, ser quantificado. Análises qualitativas podem ser feitas. Muitas vezes, simplesmente saber que um perigo existe pode ser suficiente para eliminá-lo. Uma abordagem qualitativa permite que as considerações sobre segurança iniciem cedo e considerem mais fatores que uma

abordagem quantitativa que precisa gerar números de risco embasados em dados suficientes.

Vários autores [15, 32, 79] enfatizam que a análise de perigos freqüentemente requer uma equipe com diversos conhecimentos e visões complementares. Kletz [32] lembra que o sucesso de qualquer análise de perigos depende fortemente do conhecimento e experiência dos participantes. É extremamente benéfico incluir projetistas, mantenedores e também os operadores (usuários) como participantes das análises, porque eles podem contribuir com diferentes perspectivas.

Segundo Leveson [15], operadores, por exemplo, podem contribuir muito com a experiência operacional e o conhecimento mais prático. Se os operadores participam das análises, eles também se tornam mais conscientes dos perigos e do que está sendo feito para segurança, e desta forma, podem cooperar melhor.

O processo de análise de perigos precisa ser contínuo e iterativo, conforme a figura 4.1. A figura mostra que o esforço de segurança deve se estender por todo o ciclo de vida do sistema. A identificação e análise dos perigos começam no estágio conceitual do projeto e vão até a retirada de serviço do sistema. As análises devem ser aprofundadas na medida que mais informação fica disponível. Ações precisam ser tomadas para implementar os resultados das análises e verificar que eles foram implementados corretamente. Mudanças no ambiente e manutenções do sistema devem motivar a revisão das análises.

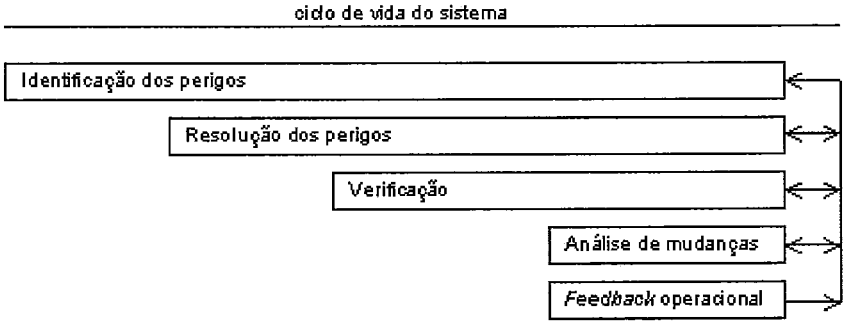

Fig. 4.1 Processo de análise de perigos [15]

A resolução dos perigos deve obedecer uma ordem de precedência.

1. **Eliminar o perigo**: eliminar o perigo por um projeto seletivo. Por exemplo, não utilizar materiais perigosos ou não permitir operações perigosas.

Se o perigo não pode ser eliminado...

2. **Reduzir o perigo**: se não é possível eliminar completamente, reduzir a chance da situação ocorrer, utilizando, por exemplo, mecanismos de *lockin*, *lockout* ou *interlock. Lockins* tornam difícil ou impossível sair de um estado seguro, por exemplo, a trava de segurança de uma pistola. *Lockouts* tornam o acesso a um estado perigoso difícil ou impossível, por exemplo, uma cerca. *Interlocks* impõem uma seqüência de eventos, por exemplo, para entrar nos Estados Unidos é necessário um visto da embaixada.

Se o perigo ou situação perigosa ocorre...

3. **Controlar o perigo**: se a situação perigosa ocorrer, controlar o perigo. Uma forma de controlar o perigo é detectá-lo e mudar para um estado seguro, o mais rápido possível. Controles podem ser passivos ou ativos. Dispositivos de controle passivo não requerem uma ação positiva para se tornarem efetivos (por exemplo, sinais de ferrovia que por gravidade quebram na posição "fechado"). Dispositivos ativos requerem algum acionamento (por exemplo, sistemas *sprinkler* para incêndios precisam detectar fumaça)

.

Se o acidente ocorre...

4. **Reduzir o dano**: se a situação se transformar em um acidente ou um perigo não previsto ocorrer, reduzir o dano ou possibilidade de dano. Redução do dano pode assumir a forma de (1) dispositivos de alerta, treinamento e procedimentos de emergência, ou (2) isolamento de sistemas perigosos de centros populacionais.

Note que uma combinação de medidas pode ser adotada, pois elas não são excludentes. Por exemplo, podemos reduzir a chance de um perigo ocorrer e ao mesmo tempo dotar o sistema de dispositivos para controlar o perigo caso ele ocorra. Por outro lado, o isolamento de sistemas perigosos de centros populacionais pode oferecer proteção para perigos específicos não identificados ou pouco compreendidos.

Os resultados das análises precisam ser documentados. O registro de perigos (*hazard log*) do sistema permite registrar e acompanhar a resolução dos perigos do

sistema durante todo o ciclo de vida. Por exemplo, os perigos são identificados, assim como, o que será feito para controlar um determinado perigo, os perigos postos de lado (*waived*), métodos de verificação, quem ficou responsável, se a recomendação foi cumprida, as premissas utilizadas nas análises.

Um formato comumente adotado para o registro de perigos é considerado a seguir [89]. Ele consiste em registrar para cada perigo:

- Número de controle;
- Sistema;
- Subsistema;
- Elemento;
- Descrição do perigo;
- Fatores causais potenciais;
- Efeitos potenciais;
- Severidade;
- Chance de ocorrência;
- Risco;
- Mitigação;
- Efeito da mitigação no risco;
- Referências;
- Verificação;
- Resultados da verificação;
- Status do controle;
- Notas.

4.2 Modelos de acidentes

Segundo Leveson [15], um modelo de acidente é uma abstração que ajuda ao analista de um acidente ou perigo considerar determinados aspectos em detrimento de outros, julgados menos importantes. Dessa forma, uma parcela reduzida da realidade pode ser explorada. Por outro lado, os modelos de acidentes podem forçar que certos aspectos, relevantes para o modelo, sejam considerados, evitando que as análises ignorem esses aspectos.

Para Kjellén [78], modelos de acidentes são representações simplificadas dos acidentes que ocorrem na vida real. Um dos importantes objetivos de introduzir um modelo de acidentes é estabelecer um entendimento compartilhado na organização de como e porque acidentes ocorrem.

Modelos de acidentes, em geral, são usados para três propósitos distintos, de acordo com Leveson [82]:

- entender e investigar acidentes que já ocorreram;
- prevenir acidentes futuros;
- avaliar o risco associado a uma atividade, uso de um produto, ou operação de um sistema.

Leveson sugere a seguinte classificação para os modelos de acidentes [82]:

- modelos dominó ou evento simples (1 causa = culpa);
- modelos de cadeias de eventos (> 1 causa);
- modelos hierárquicos (causas raízes);
- novos modelos.

Nos modelos dominó ou evento simples o foco tende a ser encontrar culpado. São modelos mais legalistas e moralistas.

Nos modelos de cadeia de eventos, os acidentes são resultado de um ou mais eventos ou fatores contribuintes. Esses modelos são os mais comumente usados.

Nos modelos hierárquicos, o nível mais baixo descreve o mecanismo do acidente. Num segundo nível, consideram-se as condições ou falta de condições que permitiram que os eventos do primeiro nível acontecessem. Num terceiro nível estariam as causas raízes ou fatores sistêmicos que afetam classes gerais de acidentes; são fraquezas que não apenas contribuíram para um acidente sendo investigado, mas que também podem afetar futuros acidentes.

Muitos modelos estão baseados no conceito de causa. Modelos mais recentes tentam superar as limitações desse conceito, ou pelo menos superar suas interpretações mais tradicionais. Em novos modelos, estão os modelos baseados em teoria de sistemas, ou em noções mais amplas de causalidade, nos quais os acidentes são resultado de

interações entre os componentes do sistema (humanos e máquinas). Acidentes podem ocorrer mesmo que não ocorram falhas. Esses modelos são os mais recentes.

4.2.1 Noção de causa

A noção ou conceito de causa tem sido muito importante em segurança. Informalmente, a causa de alguma coisa é aquilo que faz ela acontecer.

Aristóteles (384 AC – 322 AC) acreditava que nós não conhecemos alguma coisa até que nós tenhamos vislumbrado o seu "porquê" ou a sua causa original. Para Aristóteles, a causa é qualquer coisa que se possa reunir para explicar porque uma coisa é do jeito que ela é. Ele identificou quatro tipos de causas [115]:

- *causa materialis*: material de que é feita uma coisa;
- *causa formalis*: a forma ou essência de uma coisa;
- *causa finalis*: o propósito de uma coisa;
- *causa efficiens*: o que traz uma coisa à luz.

Após a Idade Média, com a revolução moderna e o sucesso das explicações mecânicas dos fenômenos, nós passamos a considerar a *causa efficiens* como o padrão para causalidade, ao ponto de nem considerarmos mais a *causa finallis* – o propósito de uma coisa - como causalidade.

Hume (1711-1776) definiu causa como um objeto, seguido por outro, e onde todos os objetos similares ao primeiro são seguidos por objetos similares ao segundo. Ou, em outras palavras, para um primeiro objeto ser causa, se ele não tivesse estado, o segundo nunca teria existido [113].

Ainda de acordo com Hume, nós nos acostumamos tanto a ver dois eventos juntos que concluímos que um é *causado* pelo outro. Mas não há uma razão lógica para essa crença. Não há evidência *a priori* de que amanhã teremos sol como tivemos hoje. Ironicamente, Hume reconhece na sua obra que, apesar do seu "ceticismo causal", não seria nada sábio "se atirar pela janela". Segundo ele, precisamos ser modestos nas nossas pretensões; e até descobrir nós mesmos a dificuldade antes que ela nos seja imposta. Desta forma, poderemos tirar algum mérito de nossa ignorância [113].

Em [15], Leveson afirma utilizar uma definição de causa, baseada em John Stuart Mill (1806-1873), que definiu causa como um conjunto de condições suficientes.

Causa é um conjunto de condições, cada uma delas sendo necessária, e, em conjunto, suficientes para o evento ocorrer.

A observação de que em uma explicação causal não tem apenas uma causa provável, mas que normalmente muitos fatores causais explicam a ocorrência de um evento, e que não se pode distinguir entre fatores "mais necessários" e "menos necessários", freqüentemente é atribuída a John Stuart Mill.

Nietzche (1844-1900), na sua luta por libertar-nos dos nossos úteis, porém desgastados, amuletos metafísicos, insiste que a causa é uma falsificação útil. A metafísica, segundo ele, tenta explicar o mundo pelo fio da causalidade, e como essa busca não tem fim, a metafísica acaba sempre precisando postular algo como *causa sui* (causa de si mesmo), ou seja, Deus.

Nietzche lembra que a causa não aperta e empurra até que ela efetua o seu fim. Deve-se usar causa e efeito apenas como conceitos puros, ou seja, como ficções convencionais para o propósito de designação e comunicação – não para explicação, porque a causa não explica nada, só descreve. O efeito não segue da causa, não há uma regra de "lei". Somos nós mesmos que estabelecemos a causa [83].

Segundo Nietzche [84], nós somos confrontados por um contínuo, do qual nós extraímos um punhado de pedaços, da mesma forma que nós percebemos o movimento apenas como pontos isolados e, então, inferimo-lo, sem na verdade vê-lo. A repentinidade com a qual muitos efeitos surgem nos confunde; mas, para ele, seria repentino para nós, humanos. Nesse momento de repentinidade, há um infinito número de processos que nos confunde. Um intelecto que pudesse ver causa e efeito como um contínuo e um fluxo, e não, como nós, em termos de divisão arbitrária e desmembramento, iria repudiar o conceito de causa e efeito e negar toda condicionalidade. Por causa e efeito, nós não explicaríamos nada, apenas descreveríamos com mais sutileza aquilo que o leigo e investigador de culturas mais remotas veria apenas como coisas isoladas. A causa é uma ficção regulativa que nós impomos ao mundo para melhor sobreviver nele.

Nietzche propõe o método genealógico, para avaliação e crítica de fenômenos complexos. No método genealógico, é importante "perder o respeito pela origem", indo além, reconhecendo que nada acontece de forma repentina, mas que os eventos que

mudam o mundo "caminham na ponta dos pés" e admitem várias interpretações. O que foi considerado uma causa pode ser visto como efeito e é possível identificar uma diversidade de motivações latentes, concorrentes, imbricadas e concomitantes. Quanto mais pontos de vista nós lançarmos sobre um fenômeno, mais conheceremos sobre, mas nunca o conheceremos por inteiro, porque conhecer requer simplificar.

Karl Popper (1902-1994) [85] sugere que nós nascemos com expectativas; com "conhecimento" que, apesar de não ser válido a *priori*, é psicologicamente ou geneticamente a *priori*, ou seja, anterior a toda experiência observacional. Uma das mais importantes dessas expectativas é a esperança de encontrar regularidade. Ela está ligada com a propensão nata de procurar regularidades, ou com a necessidade de encontrar regularidades, como nós poderíamos observar do prazer da criança que satisfaz essa necessidade. A esperança "instintiva" de encontrar regularidades, que é psicologicamente a *priori*, corresponderia muito proximamente à "lei da causalidade" (tudo pode ser casualmente explicado) que Kant acreditou ser parte do nosso aparato mental e ser válida a *priori*. Segundo Popper, ela não é válida a *priori* porque ela pode falhar: nós podemos construir um ambiente (letal) que, comparado com o nosso ambiente cotidiano, fosse tão caótico que nós falhássemos completamente em encontrar regularidades. Ambientes desse tipo foram usados em experiências com animais.

Karl Popper [86] diz, então, que o princípio da causalidade é um princípio metafísico, o qual ele não rejeita nem aceita. Ao invés disso, ele o propõe como uma regra metodológica, muito parecida com o princípio, qual seja, que nunca abandonemos nossas tentativas de explicar casualmente todo evento que possamos descrever.

A fim de prevenir futuros acidentes, nós começamos a procurar por causas. Essa procura pode ser mais ou menos sofisticada, indo desde a busca por uma causa única até níveis de causa, conforme ocorre em modelos de acidente hierárquicos. Nesses modelos, primeiro procura-se estabelecer o mecanismo do acidente – basicamente a seqüência de eventos – depois, as condições que estavam presentes e que viabilizaram o acidente e, finalmente, as chamadas causas raízes dos acidentes, muitas vezes de ordem organizacional.

Entretanto, em um novo modelo de acidentes, recentemente proposto, Leveson [47] sugere que a ênfase das análises de perigos deveria mudar de "causa", que pode ter

uma orientação limitada, legalista ou voltada para culpa, para um entendimento de acidentes em termos de "razões", ou seja, porque os eventos e erros acontecem; uma idéia atribuída a Rasmussen.

Noções mais tradicionais e limitadas de causa, como causa e efeito linear, podem estar mais preocupadas em encontrar alguém ou alguma coisa para culpar e simplesmente fazer "um acerto", ou "um remendo". Essas noções levam a ignorar fundações nos acidentes que podem ter sido estabelecidas muitos anos antes do acidente. Causas mais bem entendidas tendem a ser destacadas, tais como: falha de um componente ou erro do operador. Segundo Leveson, os modelos tradicionais tendem a ignorar: (1) fatores sociais e organizacionais nos acidentes, (2) acidentes de sistema e erros de software, (3) erro humano, e (4) adaptação ao longo do tempo.

Rasmussen [88] afirma que, para ele, razões são usadas para explicar escolhas humanas entre alternativas durante o projeto do sistema ou, em geral, escolhas entre atividades alternativas na interação com o ambiente. Já as causas explicam ou descrevem a liberação de eventos em seqüências temporais. Ele dá um exemplo do funcionamento de um relógio antigo de parede (*grandfather clock*) para melhor ilustrar a diferença. O funcionamento do relógio não é "causado" pela gravidade. A razão do seu funcionamento é a necessidade (humana) de um movimento uniforme para medir o tempo.

De acordo com Leveson [112], a Engenharia deve se concentrar em identificar os fatores que: (1) são mais fáceis e viáveis de serem alterados e (2) cuja alteração irá prevenir amplas classes de acidentes.

4.2.2 Modelagem do erro humano

O erro humano tem sido um fator importante nos acidentes. A forma mais simples, porém muito pouco eficaz para prevenção de acidentes futuros, consiste em culpar pessoas pelos acidentes e eliminá-las. Essa abordagem é ineficaz dado que todo mundo erra. Nos itens seguintes será visto como, partindo da proposta da Psicologia Cognitiva de que o erro humano está ligado de forma indissociável ao comportamento humano, o erro humano pode ser estudado racionalmente. Mecanismos ainda mais eficazes de segurança podem ser introduzidos se considerarmos as influências do meio no comportamento humano.

4.2.2.1 Psicologia cognitiva

Segundo Reason [90], para a Psicologia Cognitiva, o erro humano sempre vai existir. O erro humano não é considerado algo irracional, misterioso. Os mesmos mecanismos (cognitivos) que levam o ser humano a errar são também aqueles que levam o ser humano a conseguir resolver complexos problemas. Erro e acerto brotam da mesma raiz. Por exemplo, para resolver um problema complexo, muitas vezes recorremos à simplificações. Essas simplificações podem nos permitir continuar a perseguir a solução, evitando que fiquemos imobilizados pela complexidade. Mas essas simplificações também podem nos levar a desconsiderar dados relevantes do problema e nos desviar da solução. De forma similar, a tendência de usar informação do passado, independente da sua relevância para circunstâncias presentes ou futuras, pode levar a um erro de estimativa, mas pode também ser vista como a grande habilidade de humanos modelar regularidades do passado e aplicá-las no futuro.

Além disso, Reason acredita que o erro humano não é nem abundante nem variado como o seu vasto potencial possa sugerir. Não apenas erros seriam bem mais raros que as ações corretas, mas eles também tenderiam a assumir um número surpreendentemente limitado de formas. Essa é a hipótese básica lançada por Reason em [90].

Ele afirma que um erro só ocorre em ações que se tenha uma intenção prévia (plano). Se não havia uma intenção prévia, então, se está diante ou de um ato involuntário ou de um ato espontâneo, sobre os quais não haveria sentido em tecer considerações sobre erro. Nenhuma ação humana estaria intrinsecamente errada; ela estará errada ou não em relação a uma intenção prévia.

Assim, um erro ocorre quando uma seqüência planejada de atividades mentais e físicas falha em atingir seus resultados pretendidos. Logo, os erros podem ser de planejamento ou de execução do plano.

Os erros de planejamento são chamados equívocos (*mistakes*) e podem ocorrer na seleção dos objetivos ou meios para atingi-los. São os menos entendidos, difíceis de detectar e portanto, os mais perigosos. Algumas pessoas podem ter dificuldade em perceber que algo está errado se elas estão seguindo exatamente o plano.

Os erros de execução do plano podem ser deslizes (*slips*) ou lapsos (*lapses*). Deslizes são mais observáveis e externados como aquelas ações que não saíram conforme planejado, enquanto que lapsos geralmente são reservados para aqueles erros mais ocultos, que não se manifestam no comportamento, ligados à memória e, muitas vezes, só percebidos por quem os comete.

A figura 4.2 resume essas últimas considerações. Note que a Psicologia Cognitiva (Searl) faz distinção entre intenção para agir, que seria anterior à ação, e intenção na ação.

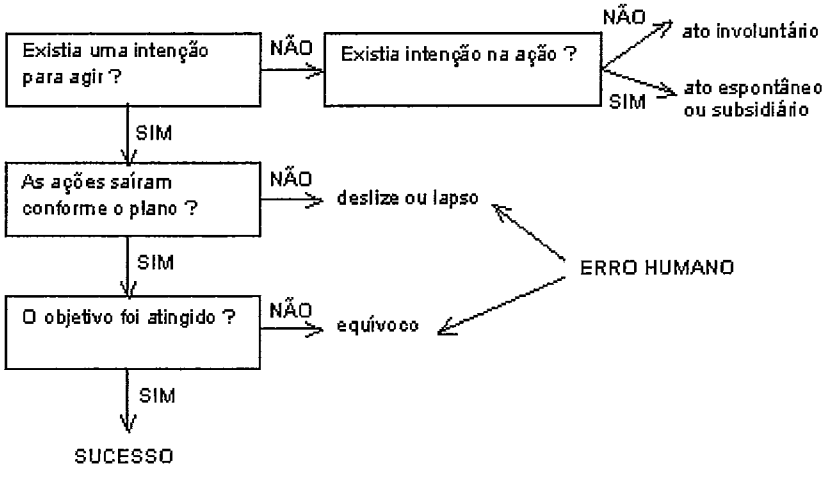

Fig. 4.2 Quando ocorre erro humano [90]

A Psicologia Cognitiva vai tentar explicar esses tipos de erros, considerando os diversos níveis de desempenho cognitivo nos quais eles ocorrerem: nível baseado em habilidades (*skill-based*), baseado em regras (*rule-based*), baseado em conhecimento (*knowledge-based*).

Essa classificação dos níveis de desempenho cognitivo foi, segundo Reason [90], sugerida por Rasmussen que estudou a resolução de problemas em voz alta e postulou os níveis cognitivos. Os níveis estão ligados à familiaridade com a tarefa sendo desempenhada que pode ser desde habitual até inteiramente nova. Por exemplo, se eu conheço bem uma situação ou problema, o meu plano para lidar com ele tende a estar

certo, mas ainda posso cometer deslizes ou lapsos na execução do plano. Se eu conheço mal uma situação, posso cometer erros já no plano mesmo.

A figura 4.3 relaciona os modos de controle cognitivo e as situações [92].

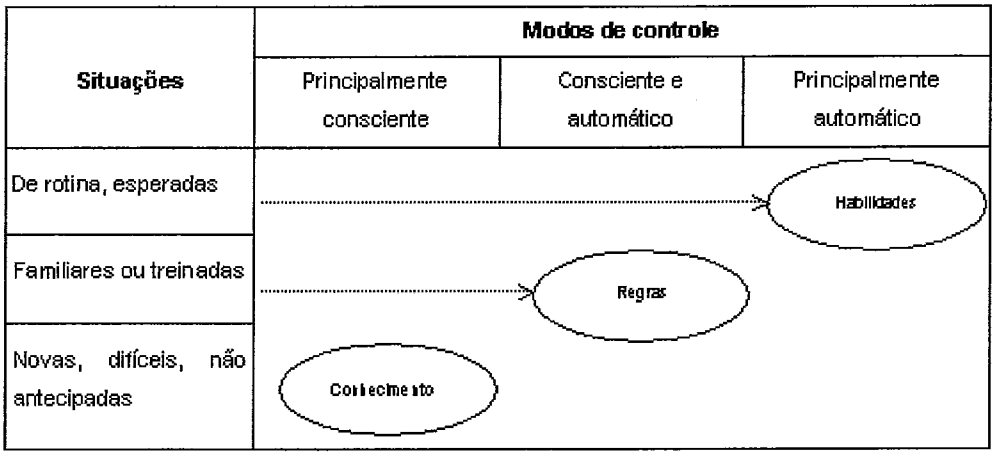

Fig. 4.3 Níveis cognitivos para cada tipo de situação [92]

No nível de desempenho baseado em habilidades, o comportamento humano é governado por padrões armazenados em instruções pré-programadas (hábito). No nível de desempenho baseado em regras, problemas familiares são resolvidos através da utilização de regras armazenadas do tipo se-situação-então-ação, também chamadas de heurísticas na Inteligência Artificial. No nível de desempenho baseado em conhecimento, situações novas, não familiares, vão requerer o planejamento de ações *online*, usando processos analíticos conscientes e conhecimento armazenado.

À medida que nos tornamos especialistas em um problema, ganhando mais experiência, o controle cognitivo será exercido mais no nível de habilidade (*skills*) que no nível de conhecimento (*knowledge*). O especialista é mais automático que o leigo porque o leigo precisa se debruçar e estudar o problema que é novo para ele. Entretanto, na prática, os três níveis de controle cognitivo sempre estarão todos mais ou menos presentes.

Reason [90] sugere uma lista de modos de falha de cada nível cognitivo. As tabelas a seguir, 4.4, 4.5 e 4.6, vão mostrar para cada nível cognitivo: o tipo de atividade que invoca aquele nível cognitivo, as características daquele nível, os tipos de erros naquele nível (deslizes, lapsos ou equívocos) e os modos de falha daquele nível. Após

cada tabela, os modos de falha são descritos de forma sucinta. Reason acredita que, considerando esses modos de falha, será possível neutralizar os efeitos indesejados dos erros humanos ao invés de tentar eliminá-los.

Nível cognitivo baseado em habilidades	
Tipo de atividade: ações de rotina	
Características: rápido, intuitivo, automático, inconsciente	
Tipos de erros: deslizes e lapsos	
Modos de falha	
Desatenção	Excesso de atenção
(1) Deslizes dupla-captura (2) Omissões após interrupções (3) Intencionalidade reduzida (4) Confusões perceptuais (5) Erros de interferência	(6) Verificações fora do tempo

Tabela 4.4 Modos de falha do nível de habilidades [90]

Esses modos de falha podem ser descritos de forma sucinta assim:

(1) Deslizes dupla-captura: a atenção fica dividida com uma preocupação interna ou algum fator de distração externo;

(2) Omissões após interrupções: lapsos que levam a retornar para o ponto errado da atividade corrente após uma interrupção;

(3) Intencionalidade reduzida: intenção enfraquece às vezes porque passa muito tempo entre a formulação da intenção e a hora que a ação é executada;

(4) Confusões perceptuais: confundir coisas parecidas;

(5) Erros de interferência: 2 planos ou 2 atividades de um mesmo plano podem se tornar emaranhadas na luta pelo controle dos atuadores;

(6) Verificações fora do tempo: verificações supérfluas quando era melhor deixar no "piloto automático", por exemplo, tocar piano prestando atenção nos dedos.

Nível cognitivo baseado em regras	
Tipo de atividade: resolução de problemas	
Características: aplicação de regras ou heurísticas (se-condição-então-ação)	
Tipos de erros: equívocos com regras	
Modos de falha	
Má aplicação de boas regras	Aplicação de regras ruins
(1) Primeiras exceções (2) Sinais, contra-sinas e não-sinais (3) Sobrecarga de informação (4) Força da regra (5) Regras gerais são mais fortes (6) Redundância (7) Rigidez	(8) Deficiências de codificação das regras (9) Deficiências na ação

Tabela 4.5 Modos de falha do nível de regras [90]

Esses modos de falha podem ser descritos de forma sucinta assim:

(1) Primeiras exceções: custa uma exceção para descobrir que uma regra não era tão geral;

(2) Sinais, contra-sinas e não-sinais: todos ao mesmo tempo causam confusão sobre a condição atual;

(3) Sobrecarga de informação: muita informação causa confusão sobre a condição atual;

(4) Força da regra: regras que costumam ser bem sucedidas têm mais chance de serem re-aplicadas;

(5) Regras gerais tendem a ser mais fortes: regras mais gerais têm mais força porque aparecem mais no mundo;

(6) Redundância: percepção de que os sinais são redundantes leva a passar a descartar informação, mas com isso, sinais contra não esperados são ignorados;

(7) Rigidez: para uma pessoa que só tem um martelo todos os problemas se parecem com um prego;

(8) Deficiências de codificação das regras: regras incompletas ou imprecisas;

(9) Deficiências na ação: problemas na ação sugerida pela regra.

Nível cognitivo baseado em conhecimento
Tipo de atividade: resolução de problemas
Características: modo consciente, restrito em capacidade, lento, seqüencial, sujeito a erros mas potencialmente bem esperto
Tipos de erros: equívocos
Modos de falhas
(1) Seletividade
(2) Limitações de "área de trabalho"
(3) "O que não está a vista, não é considerado"
(4) Viés de confirmação
(5) Excesso de confiança
(6) Revisão tendenciosa
(7) Correlação ilusória
(8) Efeito Halo
(9) Problemas com causalidade
(10) Problemas com complexidade

Tabela 4.6 Modos de falha do nível de conhecimento [90]

Esses modos de falha podem ser descritos de forma sucinta assim:

(1) Seletividade: atenção é dada às características erradas ou não é dada às certas;

(2) Limitações de "área de trabalho": área de trabalho da consciência é limitada;

(3) "O que não está à vista, não é considerado": ignorar aquilo que não está imediatamente presente; representações incompletas podem ser piores que nenhuma representação;

(4) Viés de confirmação: impede que a hipótese corrente seja abandonada em face de evidências contraditórias;

(5) Excesso de confiança: ao avaliar a correção do seu conhecimento; tendência em justificar o curso de ação escolhido focando em evidências que o favorecem e ignorando sinais contraditórios. A resistência para abandonar ou modificar um plano é especialmente forte quando:

- o plano é muito elaborado;
- o plano foi o produto de trabalho considerável e investimento emocional e o seu completamento estava associado a uma redução marcante de tensão ou ansiedade;

- o plano é o produto de várias pessoas, em especial, quando é um grupo pequeno e de elite;
- o plano tem objetivos escondidos, ou seja, quando ele é concebido subconsciente ou conscientemente para satisfazer um conjunto de necessidades e motivos.

(6) Revisão tendenciosa: o material é revisto superficialmente, em fragmentos, ao invés da parte relevante ser revisada sistematicamente;

(7) Correlação ilusória: deficiências em detectar diversos tipos de covariação (p.ex. somente as previsíveis);

(8) Efeito Halo: a falha em generalizar nossos juízos a respeito de alguém para com seus comportamentos específicos, avaliando-os definitivamente como sendo bons e corretos ou maus e incorretos; a expectativa influencia a percepção;

(9) Problemas com causalidade: por exemplo, tendência em simplificar excessivamente as causas; procurar culpados.

(10) Problemas com complexidade: *feedback* atrasado, consideração insuficiente dos processos no tempo, dificuldades com processos que se desenvolvem de forma exponencial, pensar em termos de séries lineares de causa e não em redes de causa, flerte temático ou o seu contrário, o enquistamento.

4.2.2.2 Atos inseguros

Reason [90] reconhece, além dos aspectos cognitivos (mais internos), a importância do contexto social nas ações humanas e numa tentativa de integrar as psicologias cognitiva e social, propõe os conceitos de violação e ato inseguro. Uma violação é um desvio deliberado – mas não necessariamente repreensível – daquelas práticas consideradas necessárias (por projetistas, gerentes, agências reguladoras) para manter a segurança.

As violações podem ser classificadas em: violações de rotina, excepcionais ou sabotagem. Violações e erros são genericamente denominados atos inseguros. A figura 4.7 organiza os conceitos apresentados até aqui.

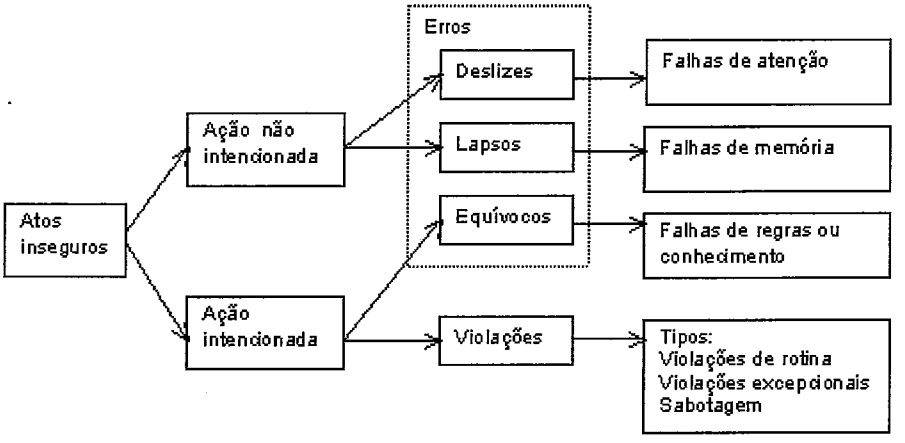

Fig. 4.7 Atos inseguros [90]

As violações de rotina são cometidas devido à: (1) tendência natural humana de tentar o caminho do menor esforço ou (2) ambiente indiferente que não pune violações nem premia observância. As violações excepcionais são menos específicas, podendo, por exemplo, serem cometidas por necessidade.

Violações também podem ocorrer em função da tentativa dos operadores do sistema de manter a segurança do sistema. Após analisar importantes acidentes recentes, Reason sugere que ao invés de serem os principais instigadores de acidentes, operadores tendem a ser os herdeiros de defeitos do sistema criados por projetos pobres, instalação incorreta, manutenção deficiente e decisões ruins da gerência. Isso pode ser especialmente agravado naqueles sistemas opacos ou que tentam eliminar o humano, porque o operador muitas vezes não sabe o que está acontecendo ou quando tem que intervir, a situação já é bem difícil.

4.2.2.3 Acidentes organizacionais

Reason gradativamente amplia o foco do indivíduo para organização, e ele realiza uma segunda generalização.

Reason sugere duas categorias de erros: (1) **erros ativos**, cujos efeitos são quase que sentidos imediatamente, e (2) **erros latentes** cujas conseqüências adversas podem ficar dormentes no sistema por um longo tempo, só se tornando evidentes quando eles combinam com outros fatores que ameaçam as defesas do sistema. Por exemplo, talvez

o primeiro sinal de alerta de que poderia haver problemas com a vedação dos tanques de combustível da Challenger foi em 1977, quando por ocasião dos primeiros testes, o fabricante já teria descoberto problemas, mas optou por persuadir a NASA que estava tudo bem. O acidente foi em 1986.

Em geral, erros ativos estão associados com a performance de operadores de "linha de frente": pilotos, controladores de vôo, oficiais de navios, tripulações, e assemelhados. Erros latentes, por outro lado, têm mais chance de serem gerados por aqueles cujas atividades estão longe no tempo e espaço da interface de controle direto: projetistas, tomadores de decisão em alto-nível, construtores, gerentes e pessoal de manutenção. Assim, descobrir e neutralizar essas falhas latentes terá um benefício bem maior que esforços localizados para minimizar erros ativos.

A figura 4.8 apresenta um modelo geral de acidentes nas organizações proposto por Reason [91].

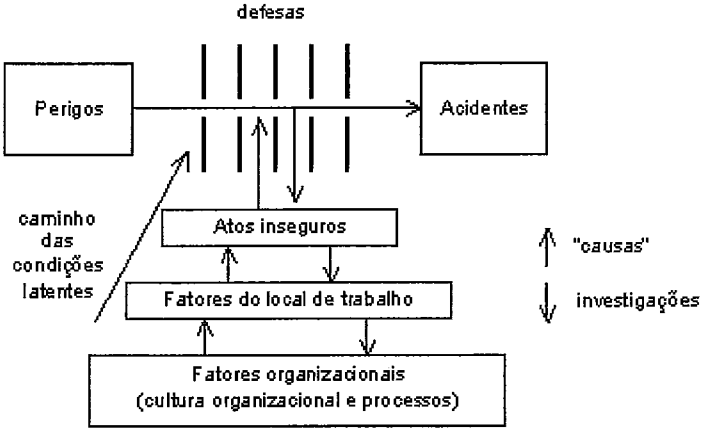

Fig. 4.8 Modelo geral para acidentes organizacionais [91]

A teoria estabelece que as conseqüências negativas das decisões de alto-nível (por exemplo, orçamentos inadequados, planejamento deficiente, pressões de prazo, etc) são transmitidas através de vários caminhos departamentais e organizacionais até diferentes locais de trabalho. Lá, elas criam condições (fatores) locais que promovem atos inseguros. Muitos desses atos inseguros serão cometidos, mas apenas alguns irão penetrar nas defesas para gerar perdas. As falhas latentes estão associadas com defesas fracas ou inexistentes [91]. Elas vão minando as defesas até que uma falha ativa e um acidente ocorre.

Reason *et al.* [92] consideram a organização uma fronteira arbitrária mas prática para considerações de segurança. Eles lembram que se quisermos buscar a origem de tudo acabaremos fazendo considerações sobre o *Big Bang*. O perímetro da organização é, segundo eles, não só um ponto de corte prático – até onde algo efetivo pode ser feito - mas também deveria ser o ponto focal, já que, segundo ele, as decisões tomadas em alto nível representam o ponto de início comum de todos os caminhos levando aos acidentes.

Subjacente à forma de fazer as coisas (os processos) de uma organização, há a cultura organizacional.

A cultura organizacional pode ser definida como um conjunto raramente articulado e amplamente inconsciente de crenças, valores, normas, e premissas fundamentais que a organização faz sobre ela mesma, a natureza das pessoas em geral, e seu ambiente. Na prática, a cultura é um conjunto de "regras" que governam o "comportamento aceitável" dentro e fora da organização. A cultura organizacional compreende aqueles atributos e crenças que tanto emergem quanto determinam a forma que uma organização realiza seus processos de negócio.

A cultura de segurança de uma organização é a parte da cultura da organização que determina o comprometimento, o estilo e a proficiência dos programas de segurança da organização. Por exemplo, a cultura de segurança pode determinar que modelos de acidente vão ser usados nas investigações.

Reason [92] acredita que a cultura de segurança pode ser objeto de uma engenharia social. Os componentes críticos dessa cultura seriam:

- cultura de relato de incidentes e acidentes;
- cultura de justiça que recompensa quem colabora mas também estabelece uma linha entre comportamentos aceitáveis e inaceitáveis;
- cultura de flexibilidade que permite que a organização se reconfigure em situações de crise e depois reverta para o seu modo tradicional;
- cultura de aprendizagem.

Esses componentes seriam combinados para gerar uma cultura informada que para ele é o mesmo que cultura de segurança na organização.

Finalmente, Reason cita o papel do contexto nacional determinando a cultura organizacional.

Segundo Leveson [47], fora do perímetro da organização também existem fatores que influenciam a segurança, determinados pela indústria, governo, agências reguladoras governamentais, tribunais, companhias de seguro ou associações de usuários. Recentemente, Leveson apresentou um novo modelo de acidentes – STAMP (*Systems Theoretic Accident Modeling and Processes*) - que tenta dar mais visibilidade a esses outros fatores.

4.3 Técnicas de análise de perigos

As técnicas de análise de perigos estão baseadas, consciente ou inconscientemente, nos modelos de acidentes, que definem porque um acidente ocorre. Todas as técnicas e modelos de análise de perigos conhecidos possuem importantes limitações. É fundamental estar ciente das limitações de cada técnica para que uma combinação adequada de técnicas possa ser utilizada.

Nos itens seguintes, veremos como as técnicas podem ser classificadas e exemplos de algumas técnicas específicas que podem ser empregadas na construção de sistemas de segurança crítica.

4.3.1 Tipos de técnicas

Segundo Leveson [15], as técnicas de análise de perigos freqüentemente envolvem buscas que são realizadas de acordo com algumas técnicas. As técnicas de busca podem ser classificadas em:

- encadeadas para frente (*forward*) ou indutivas
- encadeadas para trás (*backward*) ou dedutivas

Uma busca encadeada para frente (*forward search*) parte de um evento iniciador, ou uma condição, e acompanha o seu desdobramento no tempo. Por exemplo, uma busca para determinar como a perda de uma superfície de controle de um avião irá afetar o seu vôo.

Em uma busca encadeada para trás (*backward search*), o analista começa com um estado perigoso e determina os eventos ou estados precedentes ou causas. Esse tipo

de busca é útil em investigações das causas de acidentes e em eliminar perigos por instalação de controles para eliminar eventos predecessores.

Buscas encadeadas para trás ou para frente são ordenações cronológicas de eventos no tempo. Uma segunda categorização de técnicas de busca é:

- de cima para baixo (*top-down*)
- de baixo para cima (*bottom-up*)

Um evento, conjunto, tarefa, ou sistema pode ser dividido em eventos, condições, tarefas ou subsistemas mais básicos em uma busca de cima para baixo (*top-down*). Quando a busca é de baixo para cima (*bottom-up*), componentes são combinados de formas diferentes para determinar o resultado. Um exemplo de uma busca de cima para baixo é identificação de todas as formas que a potência possa ser perdida. Uma busca de baixo para cima pode examinar o efeito de uma falha de uma bateria individual no sistema como um todo.

4.3.2 Exemplos de técnicas

A seguir, são consideradas de forma sucinta algumas técnicas específicas de análise de perigos.

4.3.2.1 Checklists

De acordo com Leveson [15], *checklists*, ou listas de conferência, são uma forma de passar experiência para que não seja necessário que cada projeto aprenda novamente as lições do passado e inicie toda análise de perigos do zero.

Por exemplo, Leveson sugere o seguinte para auxiliar o processo de identificação de perigos [12]:

(1) Usar experiência histórica de segurança, lições aprendidas, relatório de problemas, análises de perigos e arquivos de acidentes e incidentes;

(2) Consultar listas publicadas, *checklists*, padrões, e códigos de prática;

(3) Examinar fontes de energia básicas, fluxos, itens de alta energia, materiais perigosos (combustíveis propelentes, lasers, explosivos, substâncias tóxicas, e sistemas de pressão);

(4) Examinar problemas de interface potenciais tais como incompatibilidades, possibilidade de ativação inadvertida, contaminação, e cenários ambientais adversos;

(5) Rever requisitos básicos e de missão incluindo ambientes nos quais a operação será realizada. Considerar todos os usos do sistema, todos os modos de operação, todos os ambientes possíveis, e todos os momentos da operação;

(6) Examinar a interface homem-máquina;

(7) Examinar fases de transição, modos de operação não-rotineiros, mudanças no sistema, mudanças no ambiente técnico e social, e mudanças nos modos de operação;

(8) Usar investigação científica das propriedades físicas, químicas e outras do sistema;

(9) Pensar todo o processo, passo a passo, antecipando o que pode dar errado, como se preparar para tal, e o que fazer se o pior acontecer.

4.3.2.2 Failure Mode and Effect Analysis (FMEA)

FMEA (*Failure Mode and Effects Analysis*) é a sigla para análise de modo e efeito de falhas. Conforme seu nome indica, esta técnica considera de que modo uma falha pode ocorrer e quais os efeitos dessa falha.

Segundo Leveson [15], essa técnica enfatiza funcionamento correto e não perigos. Os perigos descobertos serão aqueles decorrentes de falhas. A técnica é baseada em um modelo de acidentes denominado cadeia de eventos, no qual os eventos iniciadores são falhas de componentes individuais. Assim, é importante conhecer bem e, previamente, os modos de falha dos componentes.

O procedimento para realizar um FMEA é o seguinte:

1. Dividir o sistema em componentes (lógicos ou físicos)
2. Considerar os modos de falha em potencial de cada componente
3. Postular efeitos desses modos de falha nos demais componentes e no sistema
4. Determinar a severidade e freqüência dessas falhas

Por exemplo, em um sistema de navegação por GPS, poderia ser considerado como a falha do receptor do sinal GPS (*Global Positioning System*) afetaria o sistema de navegação.

Num sistema desse tipo, o GPS informa a posição, latitude e longitude, hora referida ao meridiano de Greenwich, velocidade e rumo de uma embarcação como resultado de complexos cálculos trigonométricos efetuados sobre sinais enviados por pelo menos três satélites GPS. Há vários satélites orbitando em torno da Terra para fornecer uma cobertura adequada, mas em determinadas situações, os sinais dos satélites podem apresentar muito ruído ou serem bloqueados por obstáculos naturais, como por exemplo, uma montanha. Isso seria um modo de falha. Nessas situações, um sistema de navegação baseado em GPS perderia a informação de posição e não teria mais como executar as principais funções de navegação que dependem dessa informação, como, por exemplo, dar sugestões de manobra.

Assim, o efeito de um modo de falha do receptor GPS levou a contemplar uma situação perigosa no sistema. De posse dessa informação, sabe-se que algo precisará ser feito para dar conta de uma falha do receptor GPS. A fim de mitigar esse perigo, uma decisão de projeto poderia determinar que se o GPS falhar, um subsistema de navegação estimada assumiria a navegação, baseado nos últimos dados válidos recebidos do GPS, e o operador seria imediatamente avisado do evento.

4.3.2.3 Fault Tree Analysis (FTA)

FTA (*Fault Tree Analysis*) é a sigla para análise por árvore de defeitos. Uma FTA pode ajudar na determinação de eventos ou combinação de eventos que vão levar a um perigo. Ela é uma técnica para analisar causas de perigos e não identificar perigos. O evento no topo da árvore (perigo) já tem que ter sido identificado por outras técnicas. Cada nível da árvore relaciona eventos mais básicos que são necessários e suficientes para causar o problema mostrado no nível acima dele. Assim, cada nível da árvore mostra a mesma coisa que o nível superior, só que em maior detalhe.

A árvore ilustrada na figura 4.9 representa uma análise do perigo de explosão de um tanque.

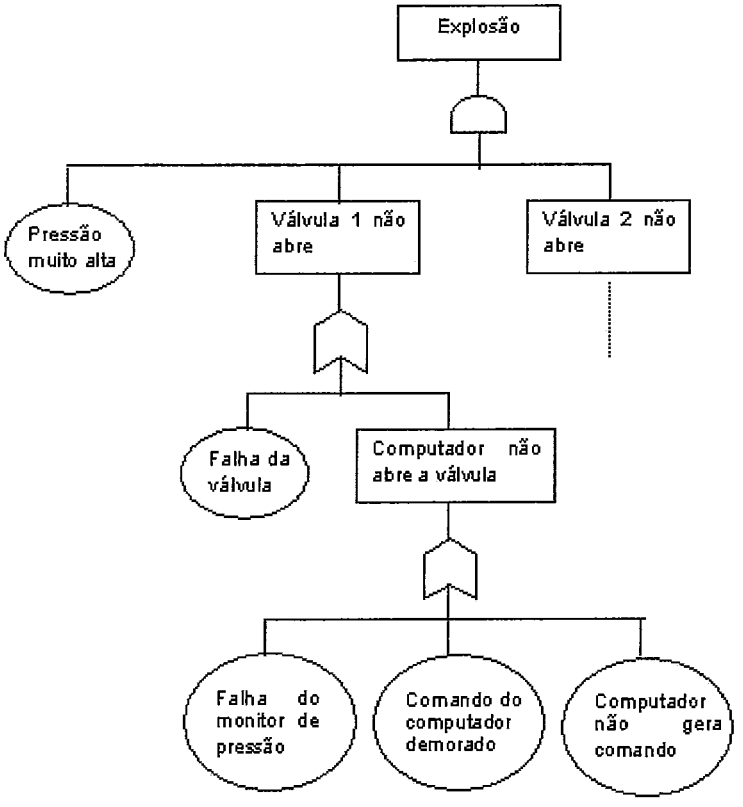

Fig. 4.9 FTA parcial da explosão de um tanque

O processo de FTA tem 4 passos básicos:

1. Definição do sistema;
2. Construção da árvore;
3. Análises qualitativas: por exemplo, determinar conjuntos de eventos básicos que são necessários para causar o evento topo, chamados *minimal cut sets;*
4. Análises quantitativas: por exemplo, combinar probabilidades para obter a probabilidade do evento topo, desde que se possa assumir independência entre eventos.

Knight [24] afirma que essas árvores podem crescer muito. Há representações de árvores de falha que podem ocupar o chão de uma grande sala.

Leveson [1] sugere uma notação alternativa, que "verticaliza" a árvore, usada na especificação de requisitos do TCAS. A estrutura hierárquica da árvore passa a ser

representada através da identação do texto. As linhas delimitadas por <> denotam folhas da árvore (não ramificam mais).

4.3.2.4 Hazard and Operability Analysis (HAZOP)

HAZOP (*Hazard and Operability*) é a sigla para estudos de operabilidade e perigos. HAZOP é uma técnica oriunda da indústria química onde ela é aplicada em análise de segurança de plantas químicas (processos químicos), mas o seu emprego tem sido estendido para outras indústrias.

Segundo Leveson [15], HAZOP é uma abordagem sistemática e qualitativa para análise de perigos. Partindo de um modelo do sistema a ser analisado, a técnica encoraja pensamento criativo sobre os perigos existentes. Para reduzir a chance que alguma coisa seja esquecida, uma sistemática é empregada. Um líder deve conduzir os estudos. Para guiar a busca, a técnica parte de um modelo daquilo que será analisado (uma planta) e utiliza um conjunto de palavras-guia (NÃO, MAIS, MENOS, INVERTIDO, etc). Antes de iniciar a análise, esse conjunto de palavras-guia deve ser estabelecido e a interpretação de cada palavra-guia deve ser fixada.

Por exemplo, ao analisar um fluxo de dados, o analista seria confrontado com perguntas do tipo:

- e se NÃO houvesse esse fluxo ? É perigoso ? O que causaria a interrupção deste fluxo ?
- e se MAIS dados fossem passados ?
- e assim por diante.

Esse processo é repetido para cada palavra-guia e para cada atributo do modelo. Claro que algumas combinações de atributo e palavra-guia podem não fazer sentido. Há variações de orientação sobre documentar absolutamente tudo, ou só aquelas combinações que façam sentido.

O poder da técnica está na sua simplicidade e facilidade de aplicação. Ela não se concentra apenas em falhas, mas tem o potencial de encontrar eventos perigosos mais complexos e suas causas. Segundo Leveson [15], HAZOP é capaz de elicitar perigos em projetos novos e perigos que não foram considerados previamente.

As desvantagens da técnica são o tempo e o esforço requerido – ela requer trabalho intenso – e as limitações impostas pelo padrão de busca (desvios). HAZOP depende muito do julgamento dos engenheiros realizando a avaliação.

Leveson [15] lembra que HAZOP cobre perigos causados por desvios do processo, que é certamente mais compreensivo e inclusivo que considerar falhas apenas, mas ainda deixa de fora perigos que possuem fatores determinantes mais estáveis sem envolver desvios.

Redmill [81] detalha o emprego dessa técnica em sistemas eletrônicos programáveis. Segundo ele, HAZOP é, talvez, a técnica mais poderosa para identificação e análise de perigos.

4.3.2.5 Análise de perigos da operação

A análise de perigos da operação se baseia na análise de atividades (*task analysis*) que quebra os procedimentos operacionais em operações menores, e procura por dificuldades na execução das operações menores ou do plano todo. Em geral, ela é conduzida de uma forma bastante informal. Uma outra alternativa, similar a uma análise de modos de falha de equipamentos, é considerar os modos de erro humano, conforme vimos neste capítulo.

4.4 Conclusão

Neste capítulo vimos o Processo de Análise de Perigos. Esse processo requer conhecimento, modelos e técnicas de análise de perigos.

Todas as técnicas e modelos de análise de perigos conhecidos possuem importantes limitações. É fundamental estar ciente das limitações de cada técnica para que uma combinação adequada de técnicas possa ser utilizada.

No próximo capítulo, proporemos um conjunto de atividades de segurança a ser mapeado em um processo de desenvolvimento de sistemas.

CAPÍTULO V - PROGRAMA DE SEGURANÇA NO CONTEXTO DO DESENVOLVIMENTO E MANUTENÇÃO DO SOFTWARE

5.1 Proposta de um Programa de Segurança

Neste capítulo, reunimos as atividades de segurança em um conjunto de atividades de segurança, ou programa de segurança, a ser mapeado em um processo de desenvolvimento de sistemas, que contenha o restante das atividades usuais de desenvolvimento e manutenção de sistemas de segurança crítica. Para cada atividade de segurança, forneceremos uma descrição, os participantes e os artefatos consumidos e produzidos.

Segurança é uma propriedade emergente do sistema operando no seu ambiente. Desta forma, os requisitos de segurança do software devem ser derivados dos requisitos de segurança do sistema que, por sua vez, devem ser derivados das análises de perigos do sistema operando em seu ambiente.

Também é importante considerar de que forma o comportamento, normal ou anormal, de um componente do sistema pode afetar a segurança do sistema. Por exemplo, o código precisa ser analisado porque, em geral, ele pode conter detalhes de implementação que podem não ser relacionados aos requisitos do software (funções não pretendidas) e que podem apresentar comportamento perigoso.

Assim, o programa de segurança que propomos está centrado na análise de perigos, conduzida, principalmente, do sistema em direção às partes, mas, também, considerando de que forma o comportamento de uma parte pode influenciar a segurança do sistema.

Utilizamos um registro de perigos do sistema que nos permite acompanhar um perigo desde sua identificação até a sua eliminação, ou aceitação como risco. No final de um ciclo do processo, temos condição de avaliar o que já foi feito para cada um dos perigos identificados. No final de um projeto, temos como saber que perigos foram considerados e o que foi feito para controlá-los. Isso nos permite avaliar a segurança do sistema antes dele entrar em operação.

O projeto do sistema deve seguir as recomendações das análises de perigos, princípios e normas aplicáveis e o sistema deve ser projetado de tal forma que possamos verificar e validar as recomendações de segurança.

À medida que o sistema vai sendo construído, seus perigos são identificados, registrados e acompanhados em um Registro de Perigos. O formato do registro foi visto no capítulo 4. Ao final do desenvolvimento do sistema, será possível verificar que perigos foram considerados durante a sua construção e o status da resolução desses perigos. Em indústrias reguladas, o argumento de segurança deve ser construído a partir do Registro de Perigos. Após a entrada em operação, o sistema deve continuar a ser acompanhado para que a sua segurança seja mantida, novos perigos sejam identificados e no final de sua vida útil, o sistema seja retirado de serviço com segurança.

As atividades do programa de segurança serão apresentadas a seguir. Essas atividades combinadas irão contribuir para construção de um sistema que seja argüivelmente seguro. Elas não são executadas seqüencialmente. Elas serão mapeadas nas atividades em um processo de desenvolvimento de sistemas, que contenha o restante das atividades usuais de desenvolvimento e manutenção de sistemas de segurança crítica. Algumas atividades serão executadas várias vezes.

Atividades do Programa de Segurança

1. **Implementar Programa de Segurança**
2. **Planejar Segurança**
3. **Aprovar Plano de Segurança**
4. **Acompanhar e Atualizar Plano de Segurança**
5. **Análise de Perigos Preliminar**
6. **Análise de Perigos do Sistema**
7. **Análise de Perigos do Software (Subsistema)**
8. **Análise de Perigos na Operação**
9. **Projetar para Segurança**
10. **Manter Segurança durante o Desenvolvimento**
11. **Análise de Perigos ao Nível do Código**
12. **Estabelecer Rastreabilidade dos Perigos**
13. **Avaliar Segurança do Sistema**

14. Planejar Verificação de Segurança

15. Verificar Segurança

16. Certificar Sistema

17. Acompanhar Sistema após Entrada em Operação

18. Manter Segurança durante a Manutenção

19. Atualizar Base de Dados de Segurança

20. Treinamento de Segurança

21. Avaliar o Programa de Segurança

A atividade Avaliar Programa de Segurança deveria ser realizada no contexto de um Processo de Melhoria de acordo com a ISO/IEC 12207.

A seguir, detalhamos as atividades do programa de segurança proposto.

Implementar Programa de Segurança

Descrição: Implementar e manter um programa de segurança na organização a fim de garantir que a segurança seja considerada em todos os projetos e mantida durante todo o ciclo de vida dos sistemas.

Participantes:

Toda organização

Produto:

Política de Segurança da Organização

Uma cultura de segurança estabelecida na organização

Base de Dados de Segurança

Planejar Segurança

Descrição: O Plano de Segurança deve detalhar como o trabalho de segurança será executado em um projeto específico. Ele deve ser feito no início do projeto e mantido atualizado ao longo do mesmo, à medida que mais informação se torna disponível, ou mudanças ocorrem. As partes envolvidas na execução do plano devem participar do planejamento para se obter comprometimento.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Produto:

Plano de Segurança

Aprovar o Plano de Segurança

Descrição: Verificar se o Plano de Segurança está de acordo com a Política de Segurança da Organização ou, no caso de indústrias reguladas, às exigências do regulador.

Participantes:

Gerente de Projeto

Autoridade governamental (certificador), se for o caso

Produto:

Plano de Segurança aprovado

Acompanhar e Atualizar o Plano de Segurança

Descrição: Executar, verificar a execução, e manter o plano atualizado ao longo de um projeto. Verificar o que já foi feito para cada um dos perigos identificados. Um Relatório de Progresso pode ser emitido a fim de facilitar o acompanhamento.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Produto:

Atividades executadas conforme o Plano de Segurança

Plano de Segurança atualizado

Relatório de Progresso

Análise de Perigos Preliminar

Descrição: Identificar, analisar e priorizar perigos do sistema como um todo, incluindo hardware, software e procedimentos manuais. Definir requisitos de segurança de alto nível.

Participantes:

Gerente de Projeto

Engenheiro

Especialista no domínio

Operador

Produto:

Perigos eliminados

Perigos não eliminados identificados e armazenados no Registro de Perigos

Requisitos de segurança de alto nível definidos

Análise de Perigos do Sistema

Descrição: Examinar interfaces entre subsistemas para avaliar a segurança do sistema operando como um todo, incluindo hardware, software e procedimentos manuais. Considerar também as influências do ambiente. Refinar requisitos de segurança e mapear para os componentes individuais (incluindo operadores).

Participantes:

Gerente de Projeto

Engenheiro

Especialista no domínio

Operador

Produto:

Perigos eliminados

Novos perigos identificados e armazenados no Registro de Perigos

Perigos detalhados no Registro de Perigos, incluindo causas, efeitos, recomendações e métodos de verificação

Requisitos de segurança refinados até o nível dos componentes

Análise de Perigos do Software (Subsistema)

Descrição: Identificar como o software (incluindo COTS) pode contribuir para um acidente. Avaliar conformidade dos requisitos do software com os requisitos de segurança do sistema.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Produto:

Perigos eliminados

Novos perigos identificados e armazenados no Registro de Perigos

Perigos detalhados no Registro de Perigos, incluindo causas, efeitos e recomendações

Requisitos de segurança do software verificados quanto à conformidade para com os requisitos de segurança derivados do sistema

Análise de Perigos na Operação do Sistema

Descrição: Avaliar os perigos introduzidos pelos procedimentos operacionais, e eliminá-los ou mitigá-los, e projetar a interface homem-máquina.

Participantes:

Gerente de Projeto

Engenheiro

Especialista no domínio

Especialista em fatores humanos

Operador

Produto:

Perigos eliminados

Novos perigos identificados e armazenados no Registro de Perigos

Perigos detalhados no Registro de Perigos, incluindo causas, efeitos e recomendações

Procedimentos manuais de operação tornados mais seguros

Projeto seguro da interface homem-máquina

Projetar para Segurança

Descrição: Projetar o software respeitando a ordem de precedência da segurança (*) e princípios básicos de segurança. Rever padrões, especificações, regulamentos, guias e recomendações das análises de perigos realizadas previamente. Software deve ser projetado para ser verificável (determinismo, identificação e separação de funções críticas, simplicidade e desacoplamento).

Participantes:

Gerente de Projeto

Engenheiro

Produto:

Projeto considerando resultados das análises de perigos e normas

(*) de acordo com a ordem de precedência de segurança devemos (nesta ordem):

i. eliminar o perigo;
ii. usar dispositivos de proteção;
iii. usar dispositivos de alerta;
iv. definir procedimentos especiais ou realizar treinamento.

Manter a Segurança durante o Desenvolvimento

Descrição: À medida que o projeto avança, mudanças podem ocorrer. A segurança precisa ser mantida. Rever e atualizar análises de perigo, avaliando impacto das mudanças.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Produto:

Segurança mantida após mudança

Análise de Perigos ao Nível do Código

Descrição: O código precisa ser analisado e testado porque, em geral, ele pode conter detalhes de implementação que podem não ser relacionados aos requisitos do software (funções não pretendidas) e que podem apresentar comportamento perigoso.

Participantes:

Gerente de Projeto

Engenheiro

Produto:

Perigos eliminados

Novos perigos identificados e armazenados no Registro de Perigos

Perigos detalhados no Registro de Perigos, incluindo causas, efeitos e recomendações

Código modificado para maior segurança

Estabelecer Rastreabilidade de Perigos

Descrição: Estabelecer uma ligação entre um perigo, desde o momento no qual ele é identificado, até a aceitação de sua resolução por uma autoridade, passando pela sua implementação. Isso é muito importante para garantir que as medidas implementadas de segurança sejam bem compreendidas e não sejam desfeitas após uma manutenção.

Participantes:

Gerente de Projeto

Engenheiro

Produto:

Matriz de rastreabilidade

Avaliar Segurança do Sistema

Descrição: Avaliar e aceitar se o sistema está seguro o suficiente antes de iniciar sua instalação, testes ou a sua operação, avaliando se o risco residual é aceitável.

Participantes:

Gerente de Projeto

Autoridade de aceitação na organização

Produto:

Argumento sobre a segurança do sistema

Planejar a Verificação de Segurança

Descrição: Planejar como as medidas de segurança serão verificadas. Durante esta atividade pode-se descobrir que certas medidas são de difícil verificação e precisam ser revistas. Os resultados dos testes devem ser documentados. Os testes têm que ser repetíveis, de forma que quando houver uma alteração no projeto eles possam ser re-executados com facilidade.

Participantes:

Gerente de Projeto

Engenheiro

Produto:

Plano de Segurança atualizado

Verificar a Segurança

Descrição: Verificar medidas de segurança no software, hardware e procedimentos manuais, através de análises sobre modelos (análises estáticas) ou testes (análises dinâmicas). As análises dinâmicas devem complementar as análises estáticas, já que nem todos os perigos poderão ser considerados apenas a partir dos modelos do sistema. Certas propriedades do sistema não poderão ser modeladas adequadamente e é importante que sejam avaliadas, tais como, tempo de resposta. Testar condições de uso anormais e sobrecarga. Quando os custos de teste forem proibitivos, eles podem ser substituídos por testes em laboratórios ou simulações. Registrar resultados das verificações no Registro de Perigos.

Participantes:

Gerente de Projeto

Verificador independente

Engenheiro

Produto:

Recomendações de segurança verificadas

Certificar Sistema

Descrição: Certificar o sistema perante uma autoridade governamental ou reguladora.

Participantes:

Gerente de Projeto

Autoridade governamental ou autoridade reguladora

Produto:

Sistema certificado

Acompanhar Operação

Descrição: O trabalho de segurança deve se estender por todo o ciclo de vida do sistema porque nem todos os perigos poderão ser antecipados e mudanças podem

ocorrer no sistema, ou no seu ambiente. Realizar auditorias e inspeções. Investigar acidentes e incidentes de forma a prevenir futuros acidentes.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Usuário

Produto:

Relatórios de acompanhamento, incidentes e acidentes

Manter a Segurança Durante a Manutenção

Descrição: Quando mudanças ocorrem no sistema ou seu ambiente, a segurança precisa ser mantida. Essas mudanças precisam ser detectadas e o impacto das mudanças na segurança do sistema analisado. O mesmo deve ser feito com relação aos pedidos de mudança. Rever e refazer análises de perigos.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Produto:

Segurança mantida após mudança

Atualizar a Base de Dados de Segurança

Descrição: Manter a Base de Dados de Segurança com lições aprendidas, incidentes e acidentes.

Participantes:

Gerente de Projeto

Produto:

Base de Dados de Segurança atualizada

Treinamento de Segurança

Descrição: Realizar treinamento de segurança. O treinamento de segurança inclui: o pessoal envolvido na operação, teste e suporte e o pessoal envolvido no projeto

e instalação do sistema. Tópicos: princípios e métodos de segurança, teoria do domínio, procedimentos de segurança do sistema e resultados das análises de perigos.

Participantes:

Gerente de Projeto

Engenheiro

Operador

Produto:

Pessoal treinado

Avaliar o Programa de Segurança

Descrição: Avaliar a adequação do programa de segurança. Sugestões: (i) observar a taxa de acidentes e incidentes ao longo do tempo; (ii) comparar sistemas com e sem o programa de segurança; (iii) avaliar quantos perigos foram corrigidos pelo pessoal de segurança (bem) antes que acidentes ocorressem e (iv) examinar casos nos quais recomendações de segurança não foram seguidas e um acidente ocorreu.

Participantes:

Gerente de Projeto

Produto:

Programa de Segurança Melhorado

5.2 Atividades do Programa de Segurança no Contexto do Desenvolvimento e Manutenção do Software

As atividades do Programa de Segurança devem ser realizadas durante o processo de desenvolvimento e manutenção dos sistemas de segurança crítica. Assim sendo, a seguir descrevemos as atividades dos processos de desenvolvimento, manutenção e operação da Norma Internacional ISO/IEC 12207 para Processos de Ciclo de Vida de Software, e mostramos como as atividades do Programa de Segurança que definimos na seção anterior se inserem neste contexto.

5.2.1 Processo de Desenvolvimento

O processo de desenvolvimento da Norma ISO/IEC 12207 contém as atividades e tarefas do desenvolvedor. O processo contém as atividades para análise de requisitos,

projeto, codificação, integração, testes, instalação e aceitação relacionadas aos produtos de software. Pode conter atividades relacionadas ao sistema, se pertinente.

Na Tabela 5.1, a seguir, listamos as atividades do processo de desenvolvimento da ISO/IEC 12207 e em que atividades devem ser realizadas as atividades de segurança, definidas na seção anterior.

Atividades do processo de desenvolvimento da ISO/IEC 12207	**Atividades do Programa de Segurança a serem realizadas**
Implementação do Processo	Implementar Programa de Segurança Planejar Segurança Aprovar Plano de Segurança
Análise de Requisitos do Sistema	Acompanhar e Atualizar Plano de Segurança Análise de Perigos Preliminar
Projeto da Arquitetura do Sistema	Acompanhar e Atualizar Plano de Segurança Análise de Perigos do Sistema Análise de Perigos na Operação Planejar a Verificação da Segurança Treinamento de Segurança
Análise de Requisitos do Software	Acompanhar e Atualizar Plano de Segurança Análise de Perigos do Software (Subsistema) Planejar a Verificação da Segurança Treinamento de Segurança
Projeto do Software	Acompanhar e Atualizar Plano de Segurança Projetar para Segurança Manter a Segurança durante o Desenvolvimento
Codificação e Testes do Software	Acompanhar e Atualizar Plano de Segurança Análise de Perigos ao Nível do Código Planejar a Verificação da Segurança

	Manter a Segurança durante o Desenvolvimento
Integração do Software	Análise de Perigos ao Nível do Código
Teste de Qualificação do Software	Acompanhar e Atualizar Plano de Segurança Avaliar a Segurança Verificar a Segurança
Integração do Sistema	Análise de Perigos do Sistema
Teste de Qualificação do Sistema	Acompanhar e Atualizar Plano de Segurança Avaliar a Segurança Verificar a Segurança
Instalação	Acompanhar e Atualizar Plano de Segurança Avaliar a Segurança Manter a Segurança durante o Desenvolvimento
Apoio à Aceitação	Acompanhar e Atualizar Plano de Segurança Avaliar a Segurança (pré-operação) Treinamento de Segurança Estabelecer Rastreabilidade de Perigos Certificar Sistema Atualizar Base de Dados de Segurança

Tabela 5.1 – Programa de Segurança no Contexto do Processo de Desenvolvimento

5.2.2 Processo de Manutenção

O processo de manutenção da Norma ISO/IEC 12207 contém as atividades e tarefas do mantenedor. O processo é ativado quando o produto de software é submetido a modificações no código e na documentação associada devido a um problema, ou à necessidade de melhoria ou adaptação. O objetivo é modificar o produto de software existente, preservando a sua integridade. Esse processo inclui a migração e a retirada de serviço do produto de software. O processo termina com a retirada de serviço do produto de software.

Na Tabela 5.2, a seguir, listamos as atividades do processo de manutenção da ISO/IEC 12207 e em que atividades devem ser realizadas as atividades de segurança, definidas na seção anterior.

Atividades do processo de manutenção da ISO/IEC 12207	**Atividades do Programa de Segurança a serem realizadas**
Implementação do Processo	Implementar Programa de Segurança Planejar Segurança Aprovar Plano de Segurança
Análise do Problema e da Modificação	Acompanhar e Atualizar Plano de Segurança Manter a Segurança durante a Manutenção Atualizar Base de Dados de Segurança
Implementação da Modificação	Acompanhar e Atualizar Plano de Segurança Manter a Segurança durante a Manutenção
Revisão e aceitação da Modificação	Acompanhar e Atualizar Plano de Segurança Manter a Segurança durante a Manutenção Avaliar a Segurança (pré-testes) Certificar Sistema
Migração	Avaliar a Segurança (pré-migração) Treinamento de Segurança Acompanhar e Atualizar Plano de Segurança Manter a Segurança durante a Manutenção Acompanhar Operação
Descontinuação	Acompanhar e Atualizar Plano de Segurança Manter a Segurança durante a Manutenção (do sistema do qual este era parte) Atualizar Base de Dados de Segurança

Tabela 5.2 – Programa de Segurança no Contexto do Processo de Manutenção

5.2.3 Processo de Operação

A operação de um sistema irá inevitavelmente revelar novos e não esperados modos de operação perigosos e procedimentos deverão ser atualizados ou novos procedimentos deverão ser criados. Acidentes e incidentes devem ser investigados.

Em princípio, o sistema deveria operar somente dentro dos limites para os quais foi projetado e testado. Por exemplo, na aviação existe o conceito de "envelope de vôo" (*flight envelope*) que determina faixas seguras para uma série de parâmetros operacionais numa determinada fase do vôo. Nos aviões mais modernos, um computador pode estar monitorando e contribuindo para que o envelope seja automaticamente mantido ou ainda registrar violações para análise futura. Os pilotos de aviões comerciais de algumas companhias recebem "cartas" quando chegam aos seus destinos se tiverem ultrapassado algum limite operacional.

Na indústria nuclear, são empregados dispositivos de *shutdown* automáticos para desligar o reator se as condições normais de operação são violadas.

Em geral, as premissas utilizadas nas análises de perigo devem se tornar pré-condições para operação do sistema, ou então, as análises devem ser revistas se as premissas nas quais elas estão baseadas forem irreais. Por isso, é importante que as premissas usadas nas análises sejam explicitadas. Essas premissas podem ser premissas sobre o ambiente e sobre os operadores, como por exemplo, assumir que o operador vai ter um determinado perfil ou habilidade. Durante a operação do sistema, um mecanismo para obter *feedback* é fundamental para avaliar se as análises de perigos foram realistas. Mesmo que acidentes não ocorram, é importante aproveitar a experiência operacional para revisar as análises de segurança.

Operadores precisam ser estimulados (recompensados) a relatar problemas no sistema. Eles são uma fonte inestimável de informação. Um exemplo notável de um sistema para coletar relatos de problemas é o ASRS – *Aviation Safety Reporting System* usado na aviação. A fim de promover o relato, o ASRS adota a seguinte filosofia:

- o instituto de pesquisa responsável pelo sistema de relato (NASA) está separado da agência reguladora (FAA);
- confidencialidade e anonimato para relatores;

- re-alimentação rápida e concreta para a comunidade relatora (pilotos e controladores de tráfego);
- facilidade de preenchimento e envio dos relatos.

Pequenos incidentes podem representar valiosos sinais de alerta que permitem corrigir os produtos e processos. Por exemplo, alguns meses após o acidente do Columbia, houve muita discussão sobre um *email* interno na NASA de um engenheiro praticamente antevendo a tragédia, mas que teve sua importância diminuída no decorrer da missão. O Fórum de Riscos na Internet também pode ser uma fonte de consulta sobre problemas que ocorreram em sistemas no passado ou já estão sendo antevistos.

Auditorias de segurança periódicas também são importantes porque podem ocorrer mudanças no ambiente do sistema, procedimentos operacionais serem ligeiramente alterados e, desta forma, os modelos sobre os quais as análises de segurança foram realizadas ficarem desatualizados. Acidentes também podem ocorrer porque manutenções em princípio "temporárias" e de baixa qualidade se tornam permanentes, devido, por exemplo, a pressões para "manter a produção".

O treinamento de um operador de um sistema de segurança crítica deve ser mais abrangente e profundo. O operador precisa ter um bom entendimento do processo sendo automatizado. Os operadores precisam entender os aspectos de segurança do projeto, incluindo os perigos e o que foi feito para mitigá-los. Assim, o operador entenderá e considerará o resultado potencial de remover ou substituir controles, alterar procedimentos prescritos, e da falta de atenção aos aspectos de segurança crítica ou operações.

Treinamento dos operadores para situações de emergência com simuladores também é importante porque os simuladores podem trazer bem mais realismo ao treinamento. A usina nuclear de Angra dos Reis, por exemplo, dispõe de um centro avançado de simulação que reproduz a sala de controle do reator real. A indústria de aviação também nos fornece um outro exemplo de preocupação freqüente com treinamento dos pilotos em simuladores de vôo.

Na Tabela 5.3, a seguir, listamos as atividades do processo de operação da ISO/IEC 12207 e em que atividades devem ser realizadas as atividades de segurança, definidas na seção anterior.

Atividades do processo de operação da ISO/IEC 12207	**Atividades do Programa de Segurança a serem realizadas**
Implementação do Processo	Implementar Programa de Segurança Planejar Segurança Aprovar Plano de Segurança
Teste Operacional	Avaliar a Segurança (pré-testes) Certificar Sistema
Operação do Sistema	Treinamento Acompanhar operação Atualizar Base de Dados de Segurança
Suporte ao usuário	Acompanhar e Atualizar Plano de Segurança Manter a Segurança durante a Manutenção Treinamento

Tabela 5.3 – Programa de Segurança no Contexto do Processo de Operação

5.3 Técnicas de Engenharia de Software sugeridas para realização das atividades do Programa de Segurança

Nesta seção sugerimos algumas técnicas de Engenharia de Software que são potencialmente úteis na realização das atividades do programa de segurança proposto.

Atividades do Programa de Segurança	**Técnicas de Engenharia de Software sugeridas**
Aprovar Plano de Segurança	Walkthrough Inspeções do Plano
Análise de Perigos do Sistema	Execução de Especificações Formais

Análise de Perigos do Software	Execução de Especificações Formais Software Fault Tree Analysis (SFTA) Testes
Projetar para segurança	Modularidade (isolar funções críticas. módulos críticos simples) Determinismo Diversidade Software Diversidade de Dados Tolerância a falhas
Manter a Segurança durante o Desenvolvimento	Testes de Regressão
Manter a Segurança durante a Manutenção	Análise de Impacto Testes de Regressão
Análise de perigos ao nível do código	Uso de linguagens de programação seguras ou subconjuntos seguros de linguagens comerciais Inspeções de código Programação de Assertivas
Estabelecer rastreabilidade de perigos	Gerência de Configuração
Verificar segurança	Testes Análise estática Verificação Independente Prova de Teoremas
Acompanhar sistema após entrada em operação	Log de Operações

Tabela 5.4 – Técnicas de Engenharia de Software sugeridas

5.4 Exemplo de utilização do processo: Sistema de Navegação Eletrônica em Hidrovias

A fim de ilustrar a utilização desse processo, iremos considerar, a seguir, o exemplo de um sistema de navegação eletrônica em hidrovias.

De uma maneira geral, esse sistema compreende um computador, um receptor GPS para obter a posição da embarcação, cartas eletrônicas e um programa de navegação que lê a posição do receptor GPS e plota a embarcação na carta eletrônica em tempo real. Desta forma, um sistema desse tipo, constitui um auxílio à navegação, permitindo que a tripulação acompanhe melhor a navegação ao saber onde está navegando. Em geral, as cartas náuticas eletrônicas exibem as profundidades na região. No caso de hidrovias em rios, a carta precisa ser bem atual porque o leito do rio muda muito.

Neste exemplo, destacaremos algumas atividades do processo que são específicas de sistemas de segurança crítica.

Análise de Perigos Preliminar

Conforme vimos anteriormente, na Análise de Perigos Preliminar interessa identificar, analisar e priorizar perigos do sistema como um todo, incluindo hardware, software e procedimentos manuais, bem como, definir requisitos de segurança de alto nível.

Na navegação em hidrovias e mesmo na navegação, em geral , há um perigo muito importante a ser considerados que é o perigo de encalhe. A atividade de análise de perigos facilita a identificação de estratégias de prevenção ou mitigação, ou seja, ela expõe causas prováveis de problemas de forma que possam ser aplicadas técnicas para prevenir o problema ou suavizar suas conseqüências prováveis. Na tabela 5.5, ilustramos o resultado parcial dessa análise, em termos de uma entrada a ser guardada no Registro de Perigos do sistema.

Sistema	Sistema de Navegação Eletrônica em Hidrovias
Subsistema	Software de carta eletrônica
Nome do perigo	Águas rasas
Efeito	Encalhe da embarcação
Severidade	Catastrófico. O pior cenário envolve morte. Se cargas perigosas estiverem sendo tranportadas, podem ocorrer danos ao meio ambiente
Chance de ocorrência	Ocasional
Causas	Uma causa é a carta eletrônica estar desatualizada e a tripulação estar dependendo exclusivamente do sistema para navegar, sem utilizar meios complementares para segurança da navegação
Mitigação	O software deve sempre mostrar a data e edição da carta eletrônica em uso
Verificação	Conduzir um teste em campo para avaliar o perfil de uso do software pela tripulação. Utilizar uma carta eletrônica muita antiga para o local e verificar se a tripulação percebe e em quanto tempo percebe
Notas	Verificado em dd/mm/aaaa por XXX. Fechado

Tabela 5.5 – Exemplo de um registro de perigos em um sistema de navegação eletrônica

Planejar Segurança

O Plano de Segurança deve detalhar como o trabalho de segurança será executado em um projeto específico. Ele deve ser feito no início do projeto e mantido atualizado ao longo do mesmo, à medida que mais informação se torna disponível, ou mudanças ocorrem. As partes envolvidas na execução do plano devem participar do planejamento para se obter comprometimento.

A seguir apresentamos um esboço de um Plano de Segurança.

Plano de Segurança

I. Considerações gerais

- Introdução
- Escopo e propósito
- Objetivos
- Padrões aplicáveis
- Relato de Progresso
- Documentação e Relatórios

II. Organização de Segurança

- Qualificação de Pessoal e Tarefas
- Organização funcional
- Equipes e Força de Trabalho
- Canais de comunicação
- Responsabilidade, autoridade e acompanhamento
- Responsabilidades de sub-contratados
- Coordenação
- Grupos de Segurança
- Interfaces do Programa de Segurança com outras disciplinas

III. Cronograma de Programa de Segurança

- Marcos e pontos de verificação críticos
- Datas de início e término de atividades
- Procedimentos de revisão

IV. Critérios de Segurança

- Definições
- Identificação e disseminação
- Classificação e priorização dos perigos
- Precedência da segurança
- Critérios de projeto
- Requisitos contratuais especiais

V. Dados de Segurança

- Requisitos de dados
- Sistema de Relato e Acompanhamento de Perigos
- Uso de dados de segurança

VI. Análise de Perigos

- Análise de Perigos Preliminar
- Análise de Perigos do Sistema
- Análise de Perigos de SubSistema (incluindo software)
- Análise de Perigos do Sistema Operante
- Integração de Análises de Subcontratados com a Análise Global de Perigos do Sistema
- Mapeamento dos Perigos do Sistema em Subsistemas

VII. Verificação

- Testes relacionados à segurança
- Demonstrações especiais
- Procedimentos de revisão e feedback

VIII. Auditoria

IX. Operações

- Procedimentos de Emergência e Contingência
- Atividades de Controle de Configuração
- Treinamento

X. Relato e Investigação de Acidentes e Incidentes durante Operações

XI. Atividades Especiais de Segurança

- Segurança de Instalações e Equipamentos

Projetar para Segurança

Conforme vimos anteriormente, o software deve ser projetado respeitando-se a ordem de precedência da segurança e princípios básicos de segurança. Padrões, especificações, regulamentos, guias e recomendações das análises de perigos realizadas previamente devem ser revistos. Software deve ser projetado para ser verificável (determinismo, identificação e separação de funções críticas, simplicidade e desacoplamento).

O padrão internacional IEC 61174 (Electronic chart display and information system - Operational and performance requirements, methods of testing and required test results, 2001) define os requisitos mínimos, métodos de teste e resultados requeridos para ECDIS, e é usado internacionalmente no processo de certificação de produtos. O capítulo 6 do IEC 61174 trata dos métodos de teste e resultados esperados. Por exemplo, o item 6.7.5 requer que no modo de monitoramento da rota, quando a embarcação está navegando ao longo de uma rota criada no sistema, qualquer janela sobreposta na área de exibição da carta seja removível ou possa ser removida da área de exibição da carta.

O projeto de um sistema de navegação deve ser guiado pelo padrão IEC 61174.

5.5 Conclusão

Neste capítulo definimos as atividades de um Programa de Segurança tendo como base a revisão bibliográfica dos capítulos anteriores. Em seguida, inserimos as atividades deste processo no contexto dos processos de desenvolvimento, manutenção e operação de software da Norma Internacional ISO/IEC 12207 para Processos de Ciclo de Vida de Software, identificando em que momento cada uma delas deve ser executada. No próximo capítulo, inserimos este conhecimento no meta-ambiente da Estação TABA de forma a mostrar como ele permite, a partir de agora, a configuração e instanciação de ambientes para desenvolvimento de software de segurança crítica.

CAPÍTULO VI - APOIO AO DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE DE SEGURANÇA CRÍTICA NA ESTAÇÃO TABA

6.1 Introdução

O conceito de Ambientes de Desenvolvimento de Software (ADS) surgiu na década de 70 e tem evoluído desde então. Um Ambiente de Desenvolvimento de Software é definido como sendo um sistema computacional que provê suporte para o desenvolvimento, reparo e melhorias em software e para o gerenciamento e controle dessas atividades [97]. Para tal, o ADS contém um repositório com todas as informações relacionadas com o projeto ao longo do seu ciclo de vida. Além disso, possui ferramentas que oferecem apoio às várias atividades técnicas e gerenciais passíveis de automação que devem ser realizadas no projeto. Travassos [98] enfatiza que o ADS deve se preocupar com o apoio às atividades individuais e ao trabalho em grupo, o gerenciamento do projeto, o aumento da qualidade geral dos produtos e o aumento da produtividade, permitindo ao engenheiro de software acompanhar o projeto e medir a evolução dos trabalhos através de informações obtidas ao longo do desenvolvimento.

Com a definição de ADS e a preocupação de que software não deve ser desenvolvido de forma *ad hoc*, surgiram pesquisas em ambientes centrados em processo [99], nas quais se defende a idéia de ferramentas integradas e incorporadas a um processo de desenvolvimento de software específico. Um processo de desenvolvimento de software é um conjunto bem definido e ordenado de atividades, somado aos recursos utilizados e produzidos e ao conjunto de ferramentas e técnicas para apoio à realização destas atividades [76].

ADS diferem um dos outros de acordo com a natureza de seu repositório, escopo das ferramentas fornecidas e/ou da tecnologia adotada. Pesquisas na área têm explorado diferentes aspectos em relação às ferramentas de apoio. Atualmente, o maior desafio está na construção e integração de ferramentas que possam ser adaptadas para uso em

novos contextos e de ferramentas para dar suporte ao processo de adaptação e integração.

6.2 Estação TABA e Ambientes de Desenvolvimento de Software Orientados à Organização

A COPPE possui um grupo de pesquisa em ambientes de desenvolvimento de software responsável pela definição e criação da Estação TABA [96]. Este projeto surgiu no início da década de 90 e consiste de um meta-ambiente de desenvolvimento de software capaz de gerar, através de instanciação, outros ADS. Ao longo desses anos de trabalho o conceito de ADS evoluiu para a definição de ADS com suporte à utilização de informações sobre o conhecimento do domínio de aplicação durante o desenvolvimento [102] e, mais recentemente, para a utilização de conhecimento organizacional [108].

Para melhorar a maneira como as organizações desenvolvem e mantêm software, é fundamental melhorar a maneira como elas administram o conhecimento requerido para a realização desta atividade. Desenvolvedores de software precisam que todo o conhecimento relevante para a realização de suas atividades esteja facilmente disponível, o que inclui conhecimento sobre o domínio, diretrizes e melhores práticas organizacionais, técnicas e métodos de desenvolvimento de software, experiências anteriores com o uso destas técnicas e métodos e também com o processo de software [108].

ADSOD (Ambientes de Desenvolvimento Orientados ao Domínio) buscam assegurar a disponibilidade e o entendimento de conhecimento sobre domínios de aplicação. No entanto, o conhecimento necessário, para um esforço de engenharia de software eficiente, extrapola o conhecimento sobre o domínio, envolvendo normas e diretrizes organizacionais, melhores práticas, relatos de experiências, entre outros. Reconhecendo que as organizações que desenvolvem e mantêm software lidam de forma intensa com diversos tipos de conhecimento, os Ambientes de Desenvolvimento de Software Orientados à Organização (ADSOrg) representam uma evolução dos ADSOD. Têm por objetivo apoiar o gerenciamento completo do conhecimento requerido em uma atividade de engenharia de software, evitando que este conhecimento

fique disperso ao longo da estrutura organizacional e, conseqüentemente, sujeito a dificuldades de acesso e, mesmo, a perdas.

A motivação para a construção de ADSOrg surgiu de duas constatações:

- duas ou mais organizações podem desenvolver software para um mesmo domínio com processos, interesses e características muito distintas; e
- o conhecimento do domínio não é o único conhecimento importante para apoiar desenvolvedores de software em suas atividades. Outros conhecimentos também são extremamente importantes e úteis para os desenvolvedores, como: diretrizes e melhores práticas organizacionais, lições aprendidas com o uso de processos, métodos e técnicas de software etc.

ADSOrg possuem os seguintes objetivos:

- prover para os desenvolvedores de software o conhecimento acumulado pela organização e relevante no contexto do desenvolvimento de software;
- e apoiar o aprendizado organizacional neste contexto.

Baseado nestas constatações, as funções originais da Estação TABA foram revistas e ampliadas. Suas funções atuais são [108]:

(1) Auxiliar o engenheiro de software na configuração do ambiente mais adequado para apoiar o desenvolvimento e a manutenção de software em uma organização específica (Ambiente Configurado), considerando seu processo de software e a gerência do conhecimento organizacional relevante neste contexto;

(2) Auxiliar o engenheiro de software na instanciação de ambientes de desenvolvimento de software para projetos específicos (caso a configuração de um ambiente para organização não seja possível ou considerada adequada);

(3) Auxiliar os gerentes de projeto na instanciação de ambientes de desenvolvimento de software para projetos específicos a partir de um Ambiente Configurado;

(4) Auxiliar o engenheiro de software de empresas cujo negócio é o desenvolvimento e a manutenção de software para diversos clientes na especialização de processos da sua empresa de acordo com as particularidades de um cliente específico;

(5) Auxiliar o engenheiro de software a implementar ferramentas necessárias aos ambientes;

(6) Apoiar, através dos ambientes instanciados, o desenvolvimento e a manutenção de software, bem como a gerência destas atividades;

(7) Permitir a execução do software na própria Estação, pelo menos para fins de teste.

6.2.1 Configuração e Instanciação de Ambientes na Estação TABA

A configuração de um ambiente na Estação TABA é feita a partir da definição de um processo de desenvolvimento que se caracteriza pela descrição de uma seqüência de atividades, suas ferramentas de apoio, produtos de software gerados e recursos consumidos [109].

A definição de um Processo Padrão segue o modelo proposto por Oliveira [110]: definição de um processo padrão para a organização, especialização de acordo com o paradigma de desenvolvimento e instanciação para o projeto específico.

O Processo Padrão é um processo básico abstrato que serve como guia para definição de processos concretos que são executados pela organização durante um projeto específico. A partir do Processo Padrão definido para organização, diferentes processos de software podem ser especializados de acordo com as características de cada tipo de software. Para ser utilizado em um projeto específico, o processo especializado mais adequado deve ser instanciado para atender às características do projeto específico.

O ambiente configurado contém o Processo Padrão da organização e os processos especializados para os paradigmas de desenvolvimento utilizados na organização. Este ambiente irá armazenar e prover todo o conhecimento capturado pela organização e importante para o desenvolvimento e a manutenção do software. Um ambiente configurado contém, ainda, a ferramenta AdaptPro para apoiar a instanciação de processos com a capacidade de gerar ambientes de desenvolvimento de software para

projetos específicos [115], os ADSOrg, a partir dos processos instanciados. Os ambientes instanciados são os ambientes que apóiam o desenvolvimento dos produtos de software e contêm outras ferramentas de apoio ao desenvolvimento. Estes ambientes têm, também, acesso ao conhecimento da organização armazenado no repositório da organização. A figura 6.1 mostra o esquema utilizado para geração de ambientes configurados para organizações específicas e instanciação de ADSOrg a partir da Estação TABA.

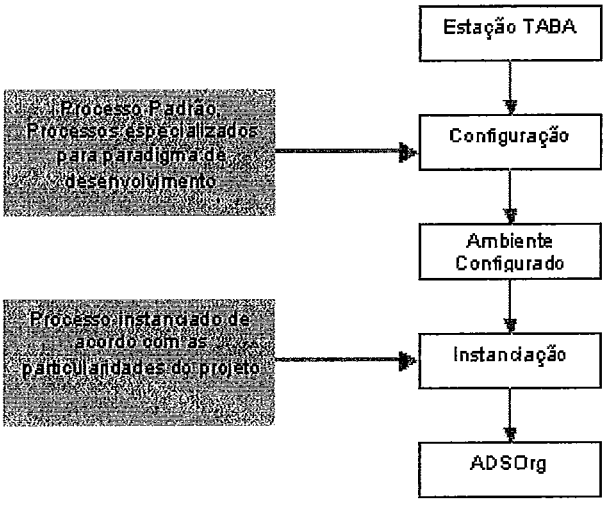

Fig. 6.1 Esquema para geração de ambientes a partir da Estação TABA

6.2.1.1 Processo de Configuração de Ambientes na Estação TABA

O objetivo deste processo é orientar a configuração, a partir do meta-ambiente da Estação TABA, de um ambiente para uma organização capaz de acumular informações, conhecimento e experiências relevantes para os seus projetos de software e de instanciar ambientes de desenvolvimento de software de acordo com as características de cada projeto.

De uma forma resumida, as atividades deste processo são as seguintes:

1. Caracterizar a organização
2. Identificar a Cultura Organizacional
3. Identificar Objetivos para Configuração do Ambiente
4. Elaborar Proposta de Configuração do Ambiente
5. Definir Escopo da Configuração

6. Definir Processo Padrão
7. Definir Processo Especializado
8. Editar Processo Padrão
9. Editar Processo Especializado
10. Definir Teoria do Domínio
11. Descrever Tarefa Genérica
12. Incluir Novas Ferramentas nos Processos
13. Gerar Ambiente Configurado

Este processo está apoiado pela ferramenta Config e ao final de sua execução tem-se um ambiente configurado TABA para uma organização específica.

6.2.1.2 Processo de Instanciação de Ambientes na Estação TABA

Para ser utilizado em um projeto específico, o processo especializado mais adequado deve ser instanciado para atender às características do projeto. O processo de instanciação apresentado a seguir descreve as atividades necessárias para se realizar a adaptação (instanciação) de um processo especializado para um projeto específico, dentro do contexto de um Ambiente Configurado.

O resultado da instanciação é um processo instanciado que constitui o Plano do Processo de Desenvolvimento ou Manutenção para um projeto específico. O Processo de Instanciação é composto das seguintes atividades [115]:

1. Caracterizar Projeto
    1.1. Definir Projeto
    1.2. Identificar Características do Projeto
2. Planejar Processo
    2.1. Incluir Atividade do Tipo de Software
    2.2. Definir Modelo de Ciclo de Vida
    2.3. Mapear Atividades do Ciclo de Vida
    2.4. Excluir Atividades Não Pertinentes
    2.5. Incluir Novas Atividades
    2.6. Selecionar Técnicas de Avaliação
    2.7. Selecionar Ferramentas
3. Instanciar ADSOrg

Este processo está apoiado pela ferramenta AdaptPro e após sua execução tem-se um ADSOrg disponível para apoiar o desenvolvimento de um produto de software.

6.3 Desenvolvimento de Software de Segurança Crítica na Estação TABA

Neste item iremos ilustrar como seria a configuração e instanciação de um ambiente de desenvolvimento na Estação TABA para uma empresa que desenvolva sistemas de segurança crítica.

6.3.1 Configuração de um Ambiente

Nesta seção, mostramos passo a passo a configuração de um ambiente na Estação TABA para uma organização que só desenvolve software de segurança crítica, dando destaque às atividades do processo de configuração que evidenciam a especificidade destes sistemas com a inclusão das atividades do Programa de Segurança descrito no capítulo anterior.

Para configuração do ambiente foi usada a ferramenta Config que apóia o processo de configuração.

Caracterizar a organização

No meta-ambiente, a caracterização da organização faz parte de um grupo de atividades "Contextualizar Configuração". Após clicarmos no ícone correspondente a "Caracterizar Organização", no lado esquerdo, surge uma tela para entrarmos com as informações relevantes do lado direito, conforme a figura 6.2. Acrescentamos a micro-empresa XYZ à lista de organizações cadastradas, definimos a indústria, tamanho da organização e domínios de conhecimento de negócio e apoio ao negócio.

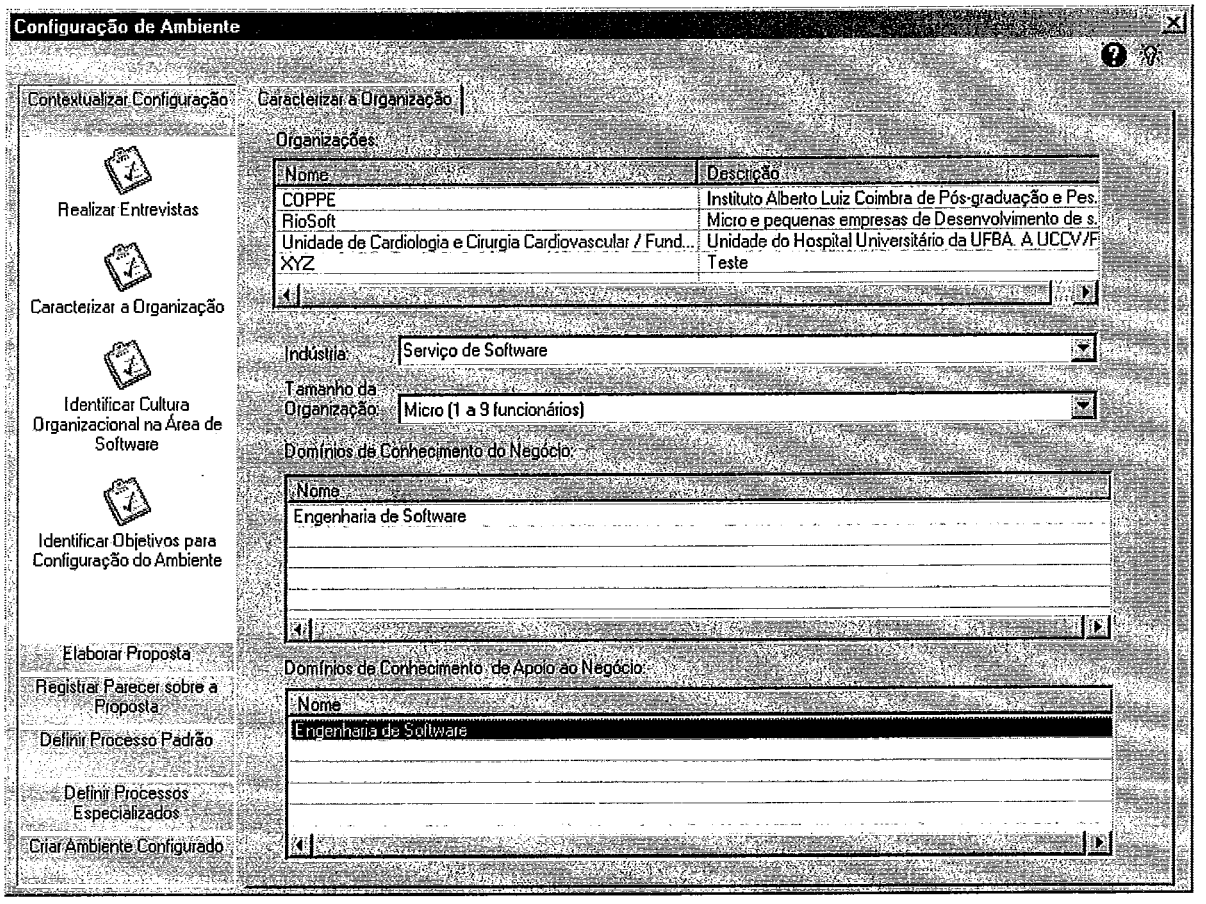

Figura 6.2 – Caracterização de uma organização no meta-ambiente

Identificar a Cultura Organizacional

A próxima atividade do processo de configuração que realizamos foi "Identificar a Cultura Organizacional na Área de Software" (figura 6.3). Os projetos da organização são realizados através de equipes formadas por um engenheiro de software da própria organização e engenheiros, analistas de sistemas e programadores de uma empresa parceira nesses projetos. Esses softwares são desenvolvidos sob encomenda para uso de terceiros.

A empresa desenvolve sistemas de informação geográfica, e, em especial, sistemas de carta eletrônica, ambos considerados software de segurança crítica. Esta informação será, posteriormente, utilizada para inclusão das atividades do Programa de Segurança nas atividades do Processo Padrão da Organização.

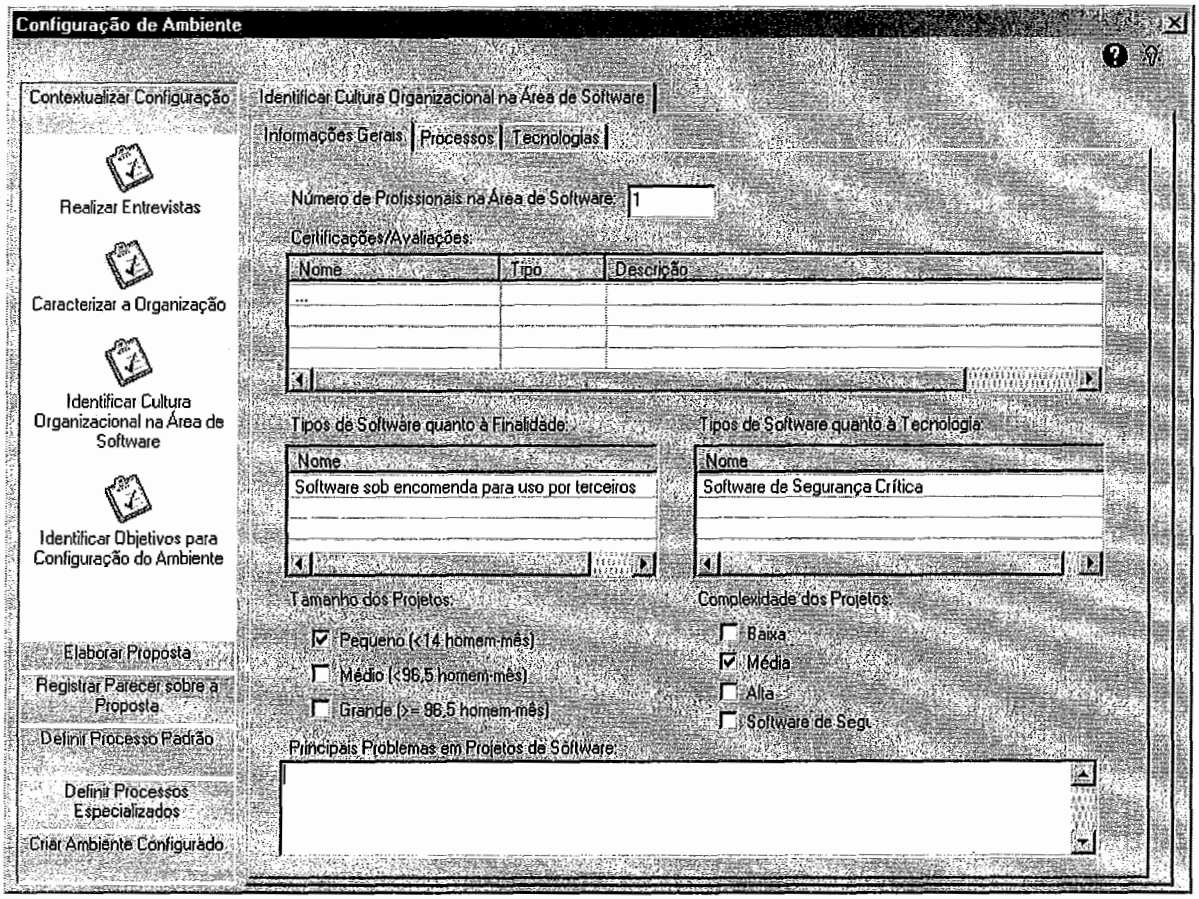

Figura 6.3 – Identificação da Cultura Organizacional na Área de Software

Na aba "Tecnologias", da tela anterior, definimos que a organização desenvolve software utilizando o paradigma Orientado a Objetos (figura 6.4). Essa informação será, posteriormente, utilizada para a especialização do processo.

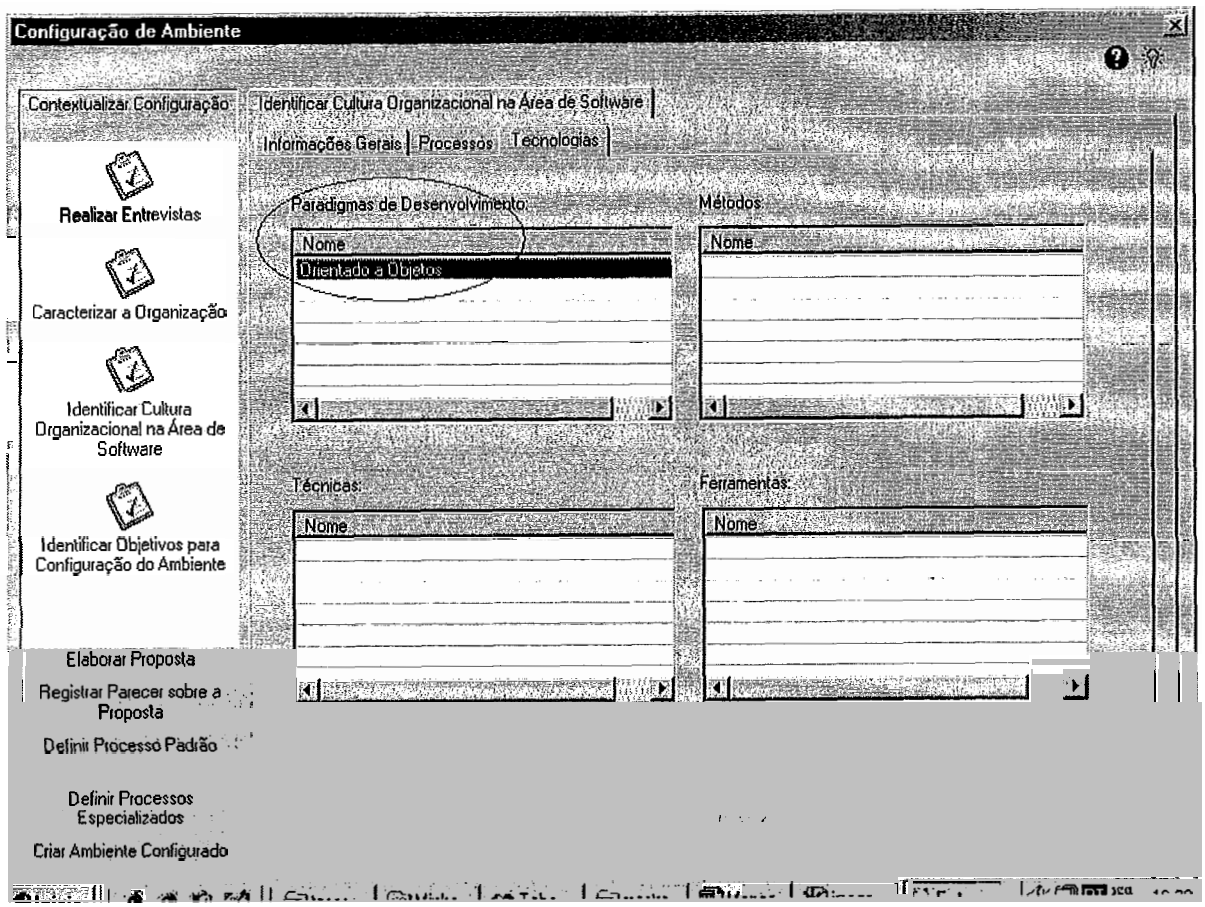

Figura 6.4 – Identificando Paradigma de Desenvolvimento

Definir Processo Padrão

No meta-ambiente da Estação TABA, a definição do processo padrão de uma organização compreende um conjunto de atividades dais quais executamos as seguintes: "Caracterizar Processo Padrão", "Incluir Atividades da ISO/IEC 12207" e "Incluir Atividades do Tipo de Software".

A figura 6.5 mostra a caracterização do Processo Padrão, no qual informamos nome e selecionamos os processo de ciclo de vida que iremos definir: desenvolvimento e manutenção.

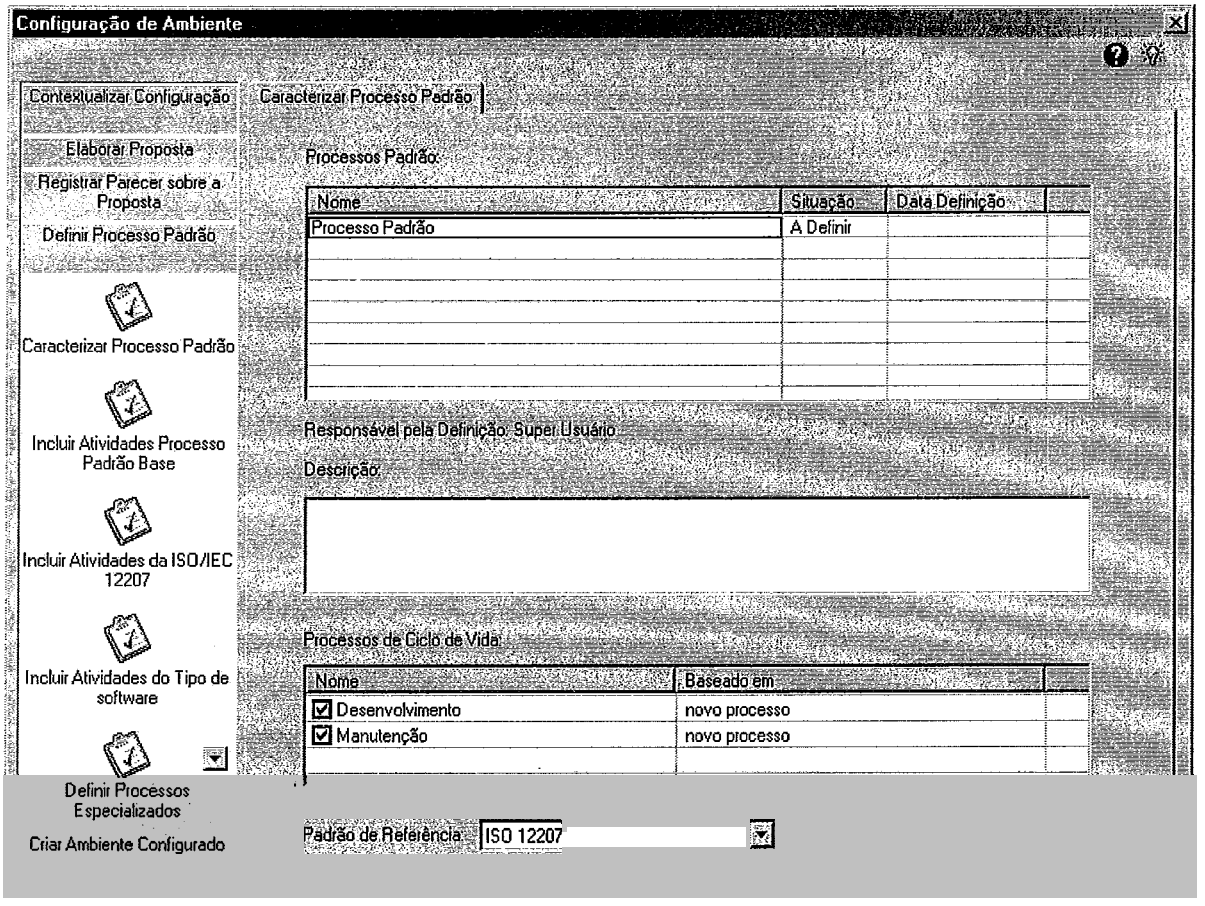

Figura 6.5 –Caracterização do Processo Padrão

A figura 6.6 mostra a tela para incluir atividades da ISO/IEC 12207. Nesta tela, selecionamos o processo de desenvolvimento e incluímos todas as atividades do processo de desenvolvimento da Norma ISO/IEC 12207.

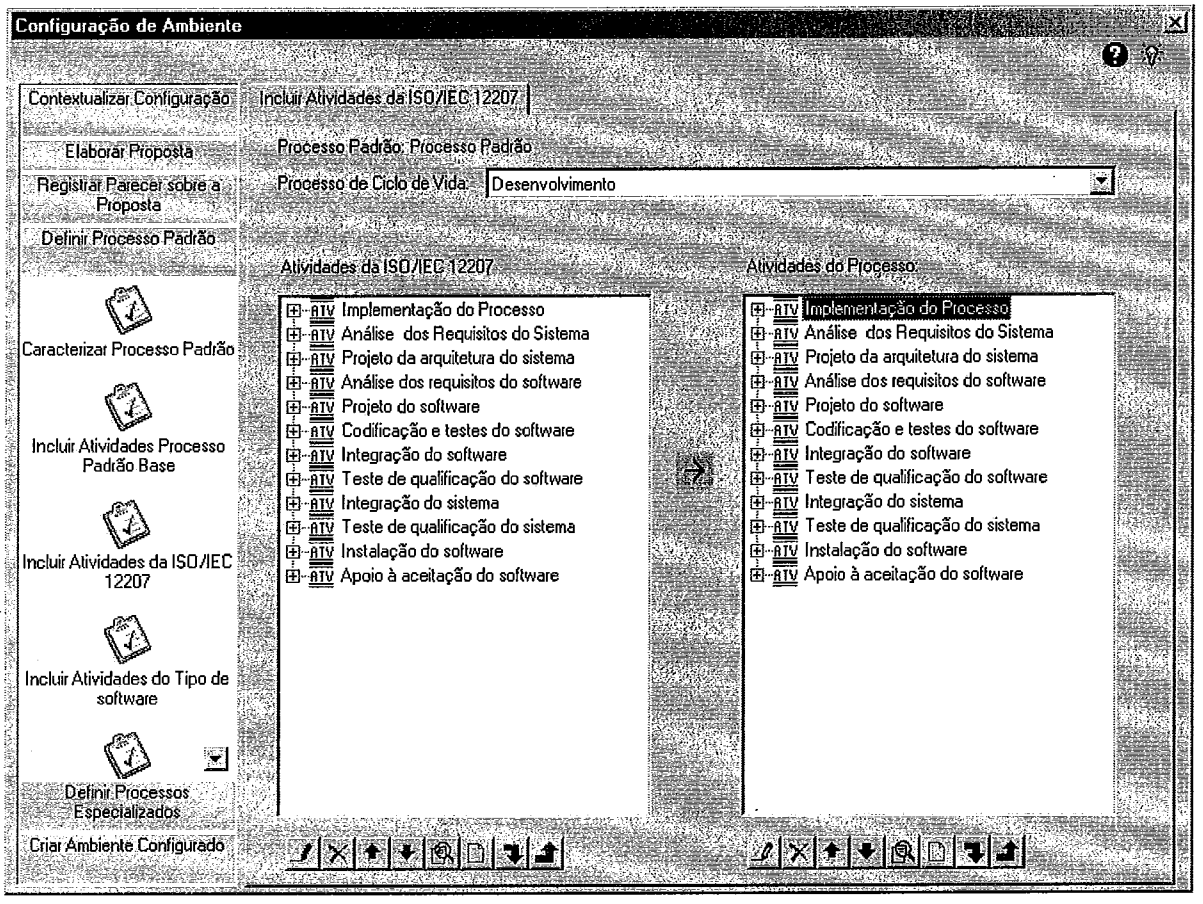

Figura 6.6 – Inclusão das Atividades da Norma ISO/IEC 12207

A figura 6.7 mostra a tela para incluir atividades do tipo de software. Nesta tela, selecionamos o processo de desenvolvimento e o tipo de "Software de Segurança Crítica". Ao selecionarmos esse tipo de software, todas as atividades do Programa de Segurança definido no capítulo anterior são listadas em "Atividades do Tipo de Software". Essas atividades foram registradas previamente na Estação TABA a partir dos resultados deste trabalho e podem, a partir de agora, apoiar engenheiros de software a configurar ambientes para organizações que desenvolvam software de segurança crítica.

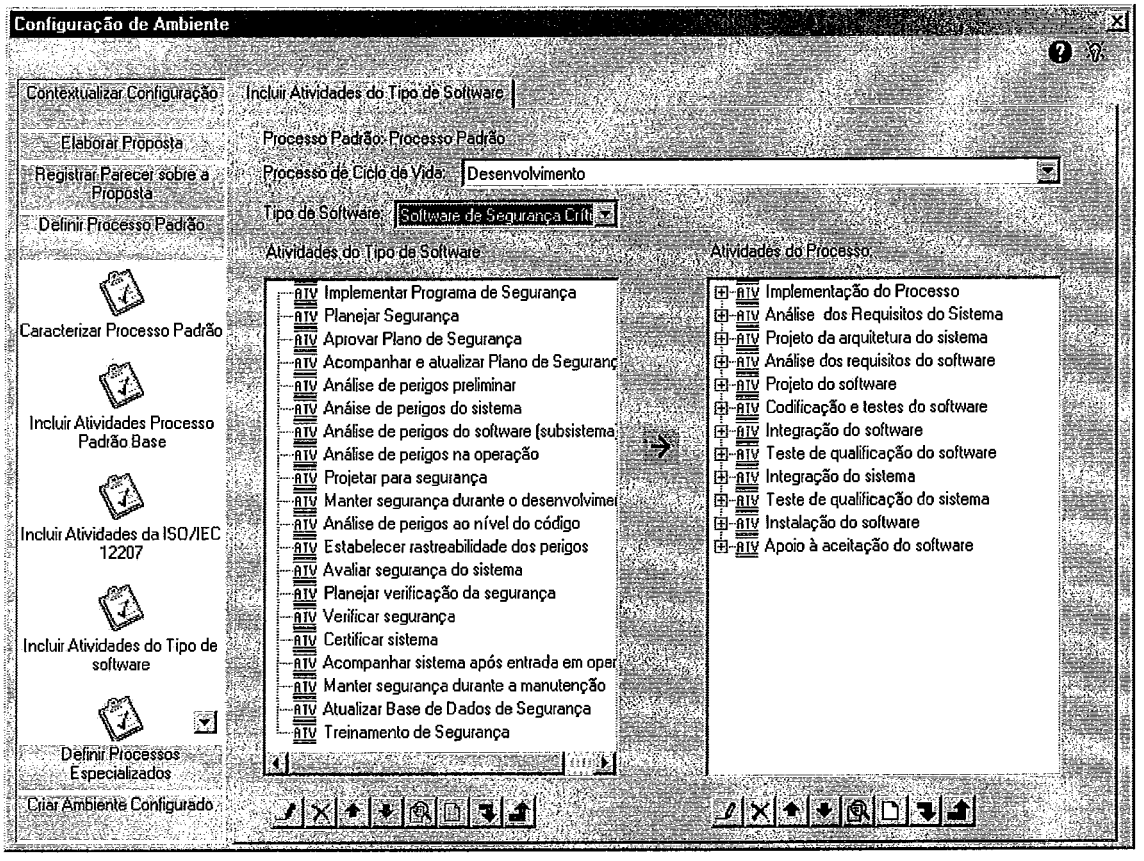

Figura 6.7 – Inclusão de Atividades do Tipo de Software

A lista de atividades necessita ainda ser mapeada para as atividades da ISO/IEC 12207. A figura 6.8 mostra as atividades do tipo de software sendo mapeadas para as atividades do processo padrão. Uma determinada atividade da lista de atividades do lado esquerdo é selecionada, o botão seta é clicado e a atividade é inserida na lista de atividades do processo sendo definido do lado direito.

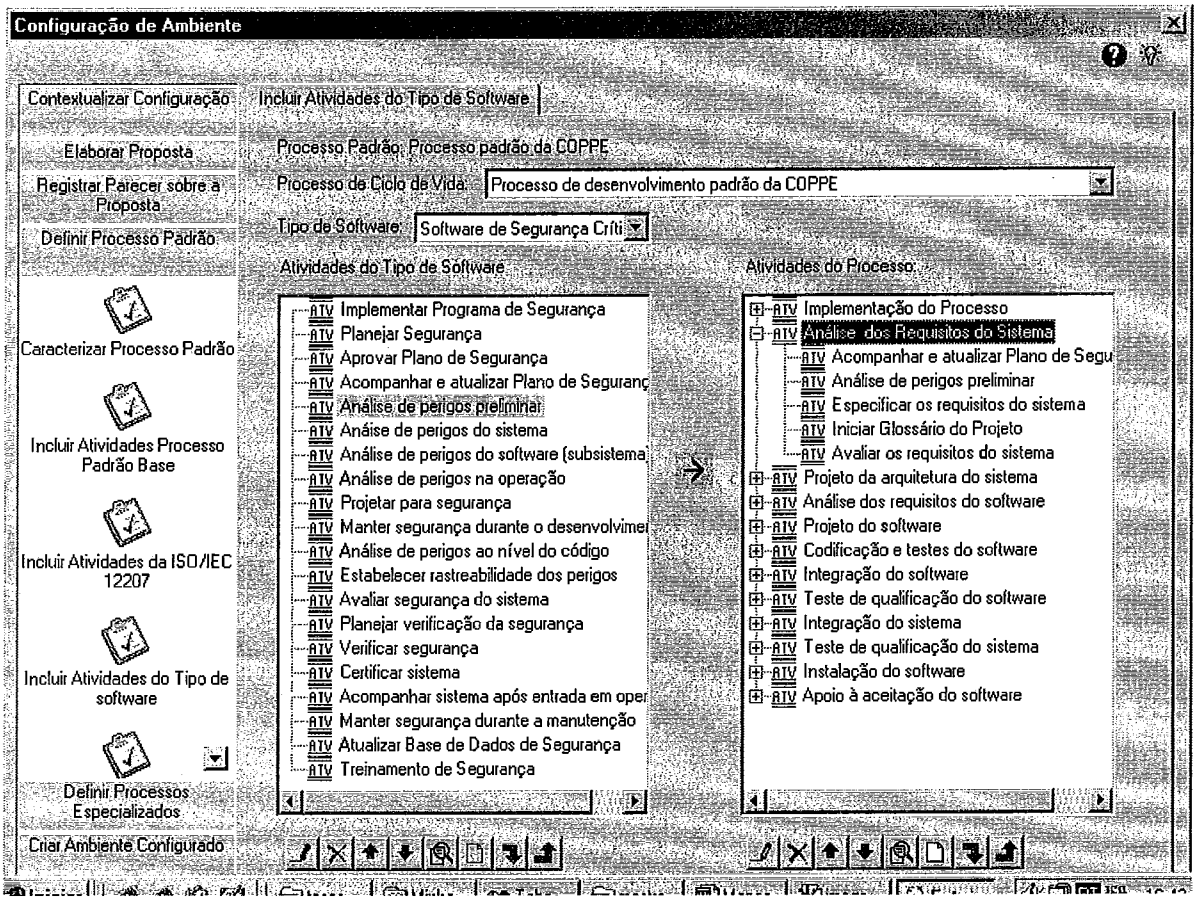

Figura 6.8 – Mapeamento de Atividades do Tipo de Software

Com esta atividade encerra-se a definição do processo padrão da organização. A próxima atividade do processo de configuração é a definição de processos especializados para os paradigmas de desenvolvimento utilizados na organização.

Definir Processo Especializado

No meta-ambiente da Estação TABA, a especialização do processo padrão é realizada automaticamente de acordo com os paradigmas de desenvolvimento adotados pela organização e já informados ao longo do processo de configuração. A figura 6.9 mostra o processo especializado que foi gerado automaticamente. Embora a especialização seja automática, o processo especializado ainda pode ser adaptado pelo engenheiro de software que está configurando o ambiente caso este considere pertinente. No caso da organização para a qual estamos configurando um ambiente, só existirá um processo especializado para desenvolvimento orientado a objetos.

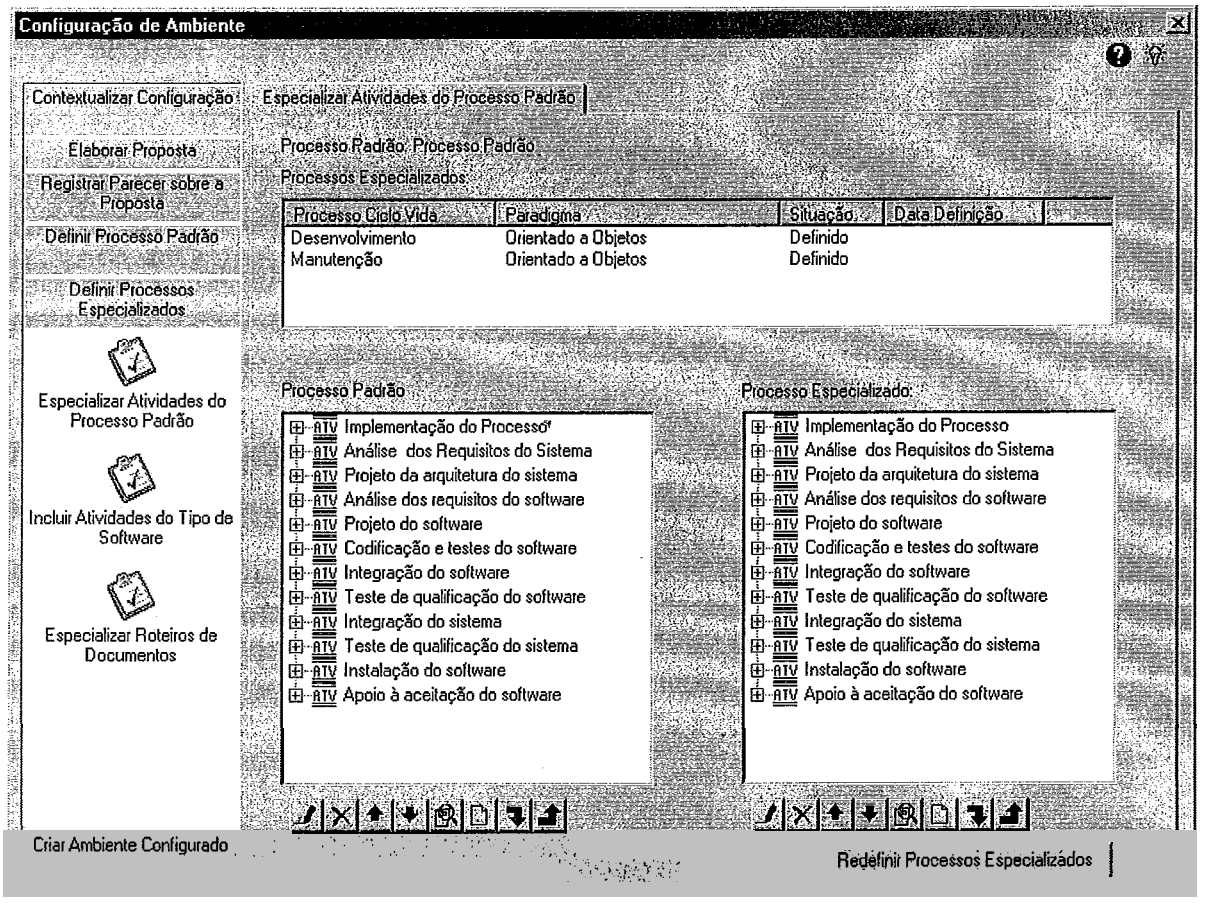

Figura 6.9 – Definição do Processo Especializado

Gerar Ambiente Configurado

Com a definição do processo especializado está concluída a definição dos processos necessária para a configuração do ambiente para a organização. O programa executável do ambiente configurado e uma série de arquivos e pastas de trabalho são gerados automaticamente.

6.3.2 Instanciação de um ADSOrg

A partir de um ambiente configurado TABA é possível instanciar um ambiente (ADSOrg) para um projeto de forma a atender às suas características específicas. A instanciação de ambientes é feita utilizando-se a ferramenta AdaptPro. Ao final do processo de instanciação tem-se a geração do ambiente instanciado e uma visualização do processo instanciado definido em HTML (figura 6.10).

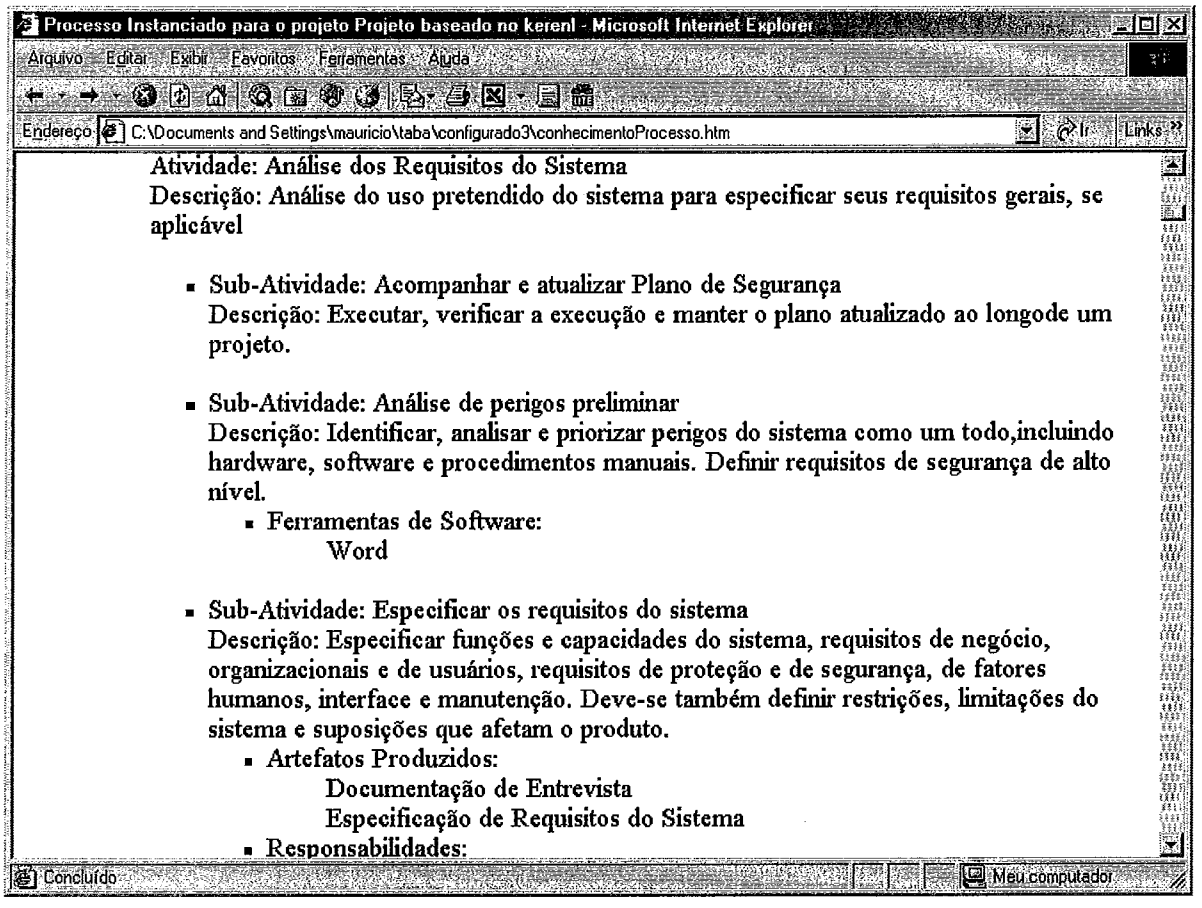

Figura 6.10 – Visualização do processo em HTML

A figura 6.11 mostra uma tela do ambiente instanciado onde podemos acompanhar as atividades do processo instanciado. Para cada atividade existe uma ou mais ferramentas disponíveis para apoiar a sua execução e os artefatos produzidos.

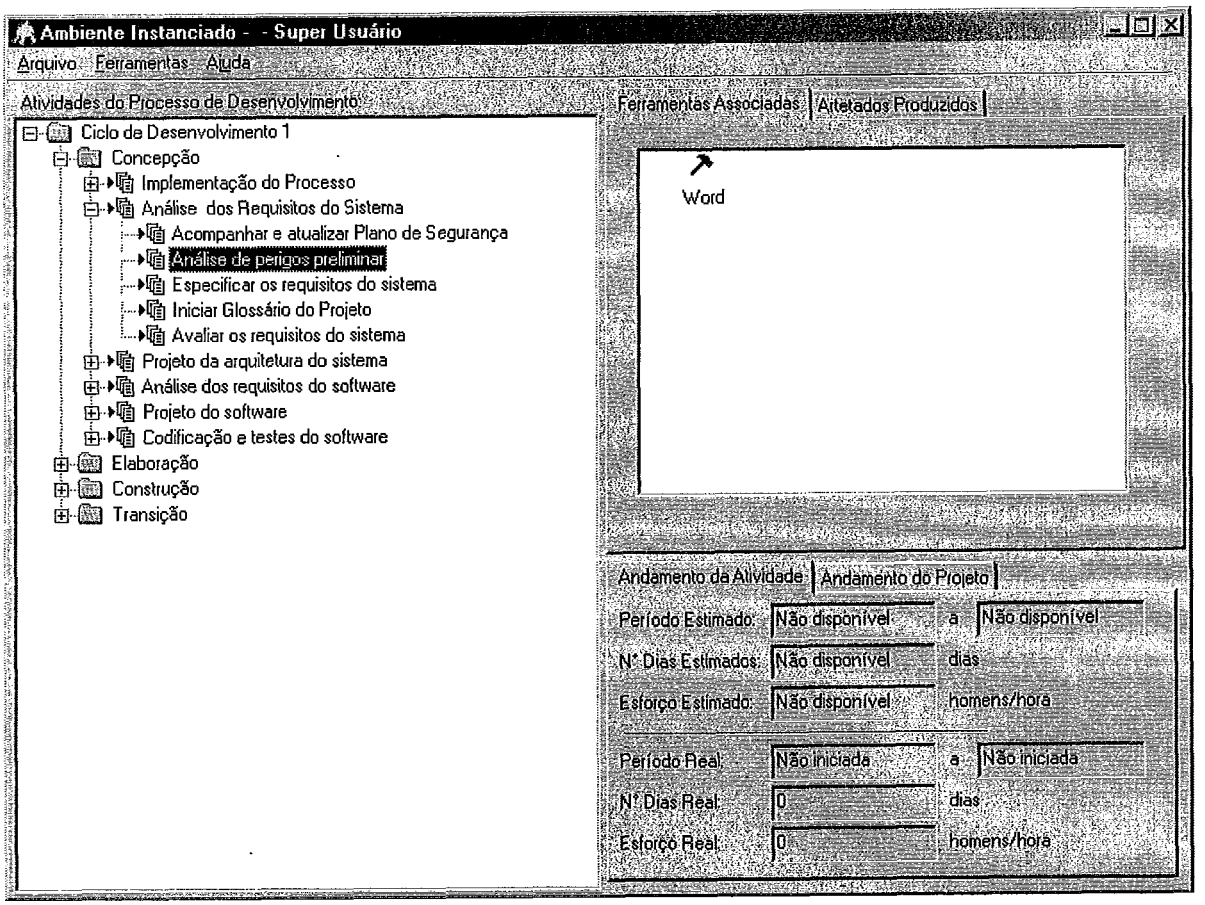

Figura 6.11 – Início da execução de uma atividade no ADSOrg

Neste exemplo, ao clicarmos no ícone da ferramenta Word associada à atividade "Análise de Perigos Preliminar", o template do documento contendo um modelo para registro dos perigos é apresentado (figura 6.12).

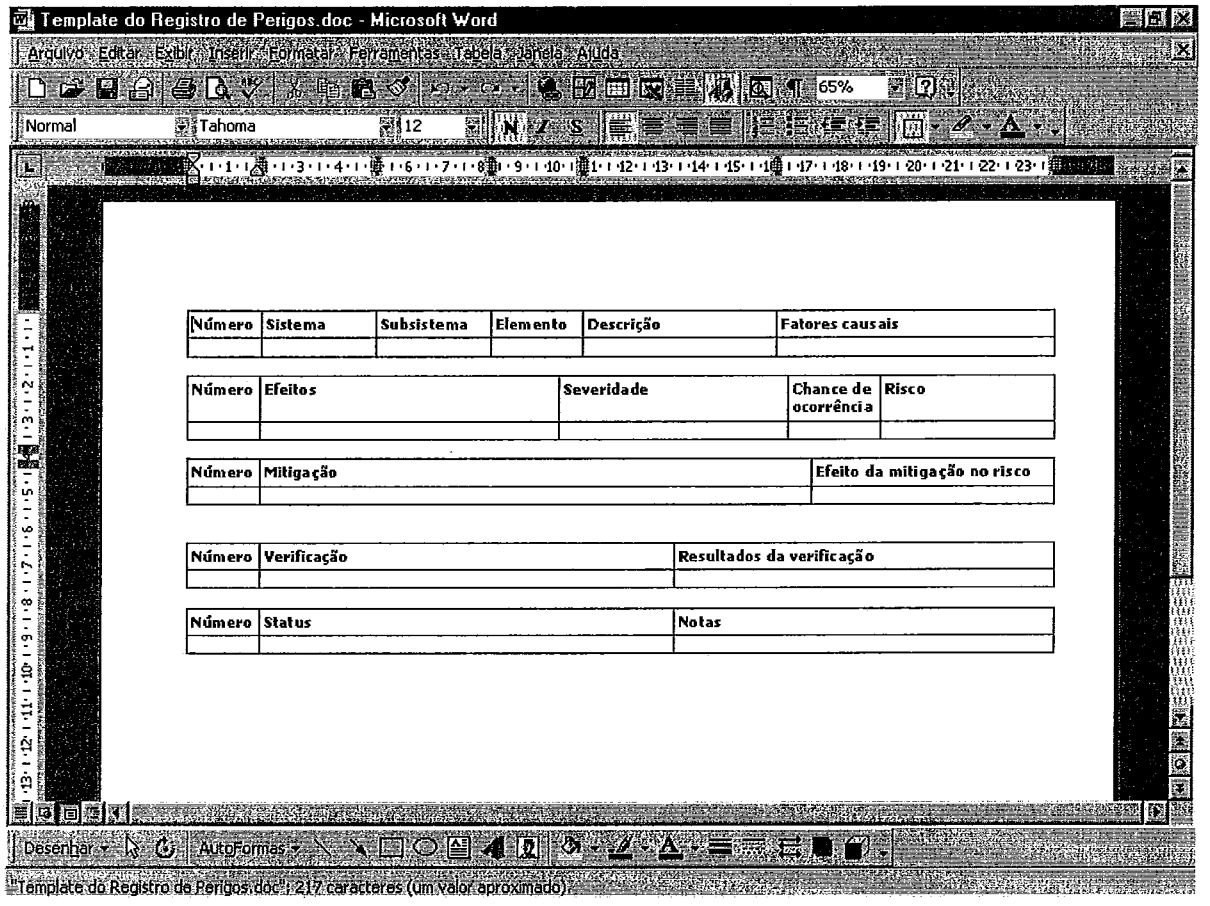

Figura 6.12 – Template do Documento Registro de Perigos

6.4 Conclusão

Este capítulo apresentou a Estação TABA em seu estágio atual e como as atividades do Programa de Segurança definido no capítulo anterior podem ser utilizadas para configuração e instanciação de um ambiente de desenvolvimento de software orientado para uma organização que desenvolva sistemas de software de segurança crítica.

CAPÍTULO VII - CONCLUSÕES E PERSPECTIVAS FUTURAS

7.1 Conclusões

Diante do crescente emprego de software em sistemas de segurança crítica e o papel cada vez mais importante que esses sistemas desempenham na sociedade, a questão de como construir sistemas de software, que funcionem conforme esperado e não causem danos, assume especial importância.

Este trabalho abordou a questão de como construir esses sistemas sob o ponto de vista do engenheiro de software e aplicando princípios conhecidos de segurança de sistemas.

Iniciamos considerando exemplos desses sistemas e alguns acidentes e incidentes já ocorridos, ilustrando a importância atual da questão escolhida pela tese, mostrando como esses sistemas vêm assumindo papéis cada vez mais importantes na sociedade, estando presentes de forma vital em quase todos os seus segmentos.

Em seguida, notamos que esses sistemas devem continuar a serem construídos a uma taxa crescente, principalmente devido à percepção, por parte dos construtores, de que software oferece amplas vantagens econômicas sobre o hardware, ou porque o software tem permitido construir sistemas inéditos.

Muitas vezes, tem ocorrido a substituição de sistemas existentes baseados em tecnologias mais antigas. Mas, o grau de inovação presente nos novos sistemas baseados em software, implica que ainda não dispomos de uma quantidade e qualidade de experiência acumulada comparável àquela de que dispomos para as tecnologias que estamos substituindo. Complementamos essa percepção com uma análise das idiossincrasias do software e as grandes dificuldades que ele oferece quando se trata de obter garantia que certos comportamentos não irão ocorrer, o que, em termos de sistemas de segurança crítica, seria a garantia, com um grau de certeza dependente da gravidade das conseqüências, de que o software não irá contribuir para a ocorrência de acidentes.

Assim, sabemos que construir esses sistemas é uma tarefa difícil e que apenas conhecer as dificuldades do software não é suficiente. Precisamos saber como construir os sistemas, que já estamos construindo, de uma forma mais segura e já, pois não há sinais de que a tendência de emprego do software irá arrefecer.

Desta forma, parte deste trabalho envolveu a busca em livros, artigos normas e na lista de discussão sobre sistemas de segurança crítica, da qual participam pesquisadores e profissionais que atuam na prática, de algo que fosse uma abordagem para construção desses sistemas, fundamentada e praticável e, na medida do possível, provada, ou seja, que estivesse associada a casos de utilização bem sucedidos. Conhecimento mais atual também foi obtido participando de um tutorial ministrado por John Knight sobre dependabilidade em sistemas embutidos e participando de um *workshop* sobre dependabilidade organizado por Rogério de Lemos, ambos no ICSE 2002 (*International Conference on Software Engineering*).

Durante nossa investigação, notamos que há ainda muita discussão sobre como construir esses sistemas, o que dificultou um pouco a pesquisa, já que não havia, e ainda não há, uma abordagem consensual.

É possível dizer que haja, hoje, pelo menos duas grandes abordagens exclusivas: a da dependabilidade e a da segurança de sistemas. Pudemos constatar que cada uma dessas abordagens é mais ou menos apropriada para um determinado tipo de sistema ou, mais especificamente, para o tipo de tecnologia empregada no sistema, e assim, optamos pela abordagem da segurança de sistemas, pois essa nos pareceu a mais apropriada para sistemas que empregam software de forma significativa, como os que vêm sendo construídos hoje.

Definida a abordagem a ser seguida, procuramos conhecer seus fundamentos teóricos – no caso da abordagem escolhida, a teoria da segurança de sistemas - e definir um conjunto de atividades de segurança, a ser praticado por quem deseje construir sistemas de segurança crítica que empregam software.

Organizamos essas atividades de segurança separadas no que comumente é chamado de Programa de Segurança, a ser mapeado em um ciclo de vida de sistemas que contenha as demais atividades usuais do desenvolvimento de sistemas de segurança

crítica. Desta forma, guardamos a possibilidade de combinação desse conjunto de atividades específicas de segurança com variadas formas de desenvolver sistemas.

Finalmente, ilustramos como seria a sobreposição das atividades de um Programa de Segurança com as de um ciclo de vida de sistemas, e cadastramos as atividades de segurança crítica na Estação TABA associando essas atividades ao tipo de software crítico.

Realizamos um exemplo de configuração de ambiente para uma organização que desenvolve sistemas de segurança crítica e instanciamos este ambiente para um projeto.

Este trabalho mostrou que, utilizando a Estação TABA, é possível disponibilizar conhecimento para engenheiros de software sobre como construir sistemas de segurança crítica, de forma tal que esse conhecimento seja facilmente incorporado nas atividades dos processos executados por esses engenheiros.

A avaliação da adequação do Programa de Segurança proposto, através de seu uso em projetos reais, excede o escopo deste trabalho. Algumas formas de avaliação sugeridas são:

- observar a taxa de acidentes ao longo do tempo;
- comparar sistemas com, e sem, o programa de segurança;
- avaliar quantos perigos foram corrigidos pelo pessoal de segurança (bem) antes que acidentes ocorressem;
- examinar os casos nos quais recomendações de segurança não foram seguidas e um acidente ocorreu.

7.2 Perspectivas futuras

Buscando melhorar e expandir a abordagem proposta, algumas perspectivas de trabalhos futuros são destacadas.

No contexto de ambientes de desenvolvimento de software, é extremamente relevante disponibilizar ferramentas para apoiar as atividades do programa de segurança proposto.

A primeira ferramenta que poderia vir a ser disponibilizada seria uma automação do acompanhamento dos perigos no Registro de Perigos. Desta forma, a ferramenta

poderia manter todos os perigos identificados do sistema e o estado da resolução desses perigos, bem como disponibilizar diversos relatórios.

Uma outra ferramenta que seria bastante útil seria um analisador de modelos, para facilitar as atividades de análises estáticas do programa de segurança proposto. Essa ferramenta iria requerer que o modelo sendo analisado estivesse representado em uma linguagem formal. Em um primeiro momento, a ferramenta poderia apenas analisar códigos em uma linguagem de programação formal. Posteriormente, artefatos de mais alto nível poderiam vir a ser analisados, como por exemplo, especificações de requisitos ou casos de uso. Quanto mais cedo a segurança é abordada no desenvolvimento, mais barato e melhores compromissos podem ser estabelecidos.

Outros tipos de conhecimento importantes no desenvolvimento de sistemas de segurança crítica, além do conhecimento de que atividades realizar, poderiam ser disponibilizados na Estação TABA, como por exemplo, lições aprendidas ou roteiros para execução das atividades do programa de segurança proposto. A base de dados de acidentes e incidentes poderia ser implementada.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] LEVESON N., DAMON J., 1999, *Sample TCAS Intent Specification*, In: M.I.T System Safety Course readings

[2] LADKIN P., 1998, *ACAS and South German Midair,* In: Technical Note RVS-Occ-02-02, University of Bielefeld, Faculty of Technology, Germany

[3] JOHNSON C., 2003, *Failure in safety critical systems: A Handbook of Incident and Accident Reporting*, http://www.dcs.gla.ac.uk/~johnson/book/, consultado em 20/02/2004

[4] HERMANN D., 1998, Software *Safety and Reliability*, IEEE Computer Society

[5] STOREY N., 1996, *Safety Critical Computer Systems*, Addison-Wesley

[6] Anônimo, Flight Deck Automation Issues, http://www.flightdeckautomation.com, consultado em 20/02/2004

[7] REASON J., 2001, *Managing the risks of organizational accidents*, Ashgate Publishing Limited

[8] Health and Safety Executive - HSE, 1998, *The use of computers in safety critical applications*

[9] NASA, 2003, *F-8 digital fly-by-wire*

[10] LADKIN P., 1996, *Report on the accident to AIRBUS A320-211 Aircraft in Warsaw*

[11] LADKIN P., HOHL M. 1997, *Analysing the 1993 Warsaw Accident with a WB-Graph*

[12] LEVESON N., 2003, *System Safety Course*, M.I.T Courseware

[13] LADKIN P., *Computer related incidents with commercial aircraf,* http://www.rvs.uni-bielefeld.de/publications/Incidents/, consultado em 20/02/2004

[14] Anônimo, *Trains a Grand Vitesse,* http://www.tgv.com, consultado em 20/02/2004

[15] LEVESON N., 1995, *Safeware: System Safety and Computers*, Addison-Wesley

[16] SCRIVNER J., 2003, *Japanese bullet trains still don't have dead man switches*, RISKS 22-60

[17] SANDBLOM E., *ICE accident*, Erik's rail news, http://www.artech.se/~sandblom/archive/eschede.html, consultado em 20/02/2004

[18] MAIN A., 2003, *NASA cultural failures on STS-107*, RISKS 22-54

[19] CNN, 24/02/2003, *NASA to give future astronauts a fighting chance*

[20] MISRA, 1994, *Development guidelines for vehicle based software,* Motor Industry Software Reliability Association, http://www.misra.org.uk/index.htm, consultado em 20/02/2004

[21] NEUMANN P., 1998, *Car computer directs couple into river*, RISKS 20-14

[22] BERGER R., 2003, *MS Windows crash traps Thai politician in car*, RISKS 22-73

[23] PERROW C., 1999, *Normal Accidents*, Princeton University Press

[24] KNIGHT J., 2002, *Intl Conference on* Software *Engineering - ICSE, Tutorial 4 – Dependability of Embedded System*

[25] MICHAELSON G., 2003, *Digital mobile phones can phreak pacemakers,* RISKS 22-77

[26] LEVESON N., 1996, *The THERAC-25 Accidents*

[27] LEVESON N., 1992, *High-Pressure Steam Engine and Computer* Software, Keynote Talk ICSE 1992

[28] MACKAY W., 1999, *Is paper safer? The role of paper flight strips in Air Traffic Control,* ACM Trans. On Computer Human Interaction, Vol. 6, No. 4, December 1999

[29] RONCO C. *et al.*, 1999, *The role of technology in hemodialisys,* Journal of Nephrology, 12 (suppl 2): S68-S81

[30] BAINBRIDGE L., 1998, *Ironies of automation,* In: New Technology and Human Error, ed. Jens Rasmussen, Keith Duncan e Jacques Leplat, John Wiley and Sons

[31] Anônimo, Site oficial do APACHE, http://www.APACHE-msi.com, consultado em 20/02/2004

[32] KLETZ T., 1999, *Hazop and Hazan,* 4 ed. Hemisphere Pub

[33] Health and Safety Executive - HSE, 2000, *Industrial use of safety-related expert systems,* Division of Informatics, University of Edinburgh and the Advanced Computation Laboratory, Imperial Cancer Research Fund for Health and Safety Executive, Dr. David Robertson, Prof. John Fox

[34] GERHART S. , CRAIGEN D., RALSTON T., 1994, *Case Study: Darlington Nuclear Power Station*, IEEE Software, Vol. 11 No. 1

[35] FLOWERS S., 1996, *Software failure: management failure*, John Wiley & Sons

[36] LEVESON N., *Software Safety in Embedded Computer Systems*, Comm. ACM, 1991

[37] *Computer crashes threaten hospital operations*, RISKS 22-62

[38] Troca de mensagens discutindo o problema no Beth, http://slashdot.org/comments.pl?sid=46238&cid=4768662, consultado em 20/02/2004

[39] Ministry of Defense – Defense Std 00-55, http://www.dstan.mod.uk, consultado em 20/02/2004

[40] WICHMANN B., 2003, *Open versus closed systems*, http://www.cs.york.ac.uk/hise/safety-critical-archive/2003/0639.html

[41] LEVESON N., 1996, *Re: Software modelling*, http://www.cs.york.ac.uk/hise/safety-critical-archive/1996/0125.html, consultado em 20/02/2004

[42] LITTLEWOOD B., STRIGINI L., 2000, *Software Reliability and Dependability: a roadmap*, Future of Software Engineering, 2000

[43] LUTZ R., 2000, *Software Engineering for Safety: a roadmap*, Future of Software Engineering

[44] DUNN W., 2002, *Practical Design of Safety-Critical Computer Systems*, Reliability Press

[45] GOTH G., 2002, NIST *Report takes a step toward better testing*, Greg Goth, IEEE Software, Vol. 19 No. 6

[46] CHILLAREGE R., 2002, *The marriage of business dynamics and software engineering*, Chillarege Inc. IEEE Software Nov./Dec. 2002

[47] LEVESON N., *A New Accident Model for Engineering Safer Systems*, Nancy Leveson, SERL, MIT

[48] SCHNEIER B., 2003, *We are all security consumers*, IEEE Security and Privacy, Vol.1 No. 1

[49] Information Week Brasil, *SPB-2*, 5 jun 2003, Ano 5, nº 95

[50] Basel Committee on Banking Supervison, www.bis.org/bcbs/cp3ov.pdf, consultado em 20/02/2004

[51] MERCURI R., 2002, *A Better Ballot Box ?,* Rebecca Mercuri, IEEE Spectrum, Oct. 2002

[52] PARNAS, D. L., SCHOUWEN J., KWAN P. S., 1990, *Evaluation of Safety-Critical* Software, Communications of ACM, Vol. 33, Number 6

[53] Department of Defence: *The procurement of computer-based safety critical systems*, Australian Defence Std Def (Aust) 5679

[54] PETROSKI, H. 1992, *To engineer is human – the role of failure in successful design*, Vintage Books

[55] LEVESON N., 1997, *Searching the limits of complexity*, Comm. Of the ACM, Vol. 40, No. 2

[56] DIJKSTRA E., 2001, *The end of computing science ?*, Comm. of the ACM, Vol. 44 No. 3

[57] Sea King *Helicopter crash – fire control system deployment failure*, 4 jun 2003, RISKS 22-76

[58] FINELLI G., BUTLER R., 1991, *The Infeasibility of Experimental Quantification of Life-Critical Software Reliability*, Ricky W. Butler, George B. Finelli, NASA Langley Research Center

[59] LITTLEWOOD B., *SIL Claims*, http://www.cs.york.ac.uk/hise/safety-critical-archive/2003/0096.html, consultado em 20/02/2004

[60] LADKIN P., 2000, *Notes on the Foundations of System Safety and Risk*

[61] FAA, 2000, System Safety Handbook

[62] WORDSWORTH J., 1992, *Software development with Z*, Addison-Wesley

[63] JONES C. B., 2003, *The early search for tractable ways of reasoning about programs*, IEEE Annals of the History of Computing, Apr-June 2003

[64] DE MILLO R. A., LIPTON R., PERLIS A., 1979, *Social Processes and Proofs of theorems,* Comm of ACM, Vol 22 No 5

[65] FETZER J. H., 1988, *Program verification: the very idea*, Communications of ACM, Vol 37 No 9

[66] HAREL D., 2000, *Computers Ltd.*, David Harel, Oxford University Press

[67] LEVESON N., *Intent Specifications : An Approach to Building Human-Centered Specifications*, Aeronautics and Astronautics, MIT

[68] LEMOS R., *Confiança no Funcionamento: Terminologia em Português*, http://www.cs.ukc.ac.uk/people/staff/rdl/CoF/, consultado em 20/02/2004

[69] KNIGHT J., 2002, *Software challenges in aviation systems*, University of Virginia, US, SAFECOMP 2002

[70] RUSHBY J., 1994, *Critical System Properties : Survey and Taxonomy*

[71] LAPRIE J. C., *Guide de la sûreté de fonctionnement*, Cépaduès Éditions

[72] LAMPORT L., OWICKI S., 1982, *Proving liveness properties of concurrent programs*, Comm of ACM, Vol 4 No 3

[73] LEVESON N., *The role of Software in Recent Aerospace Accidents*, MIT

[74] LEVESON N., *A giant step forward ?*, 1999, 0092, http://www.cs.york.ac.uk/hise/safety-critical-archive/1999/0092.html, consultado em 20/02/2004

[75] ALLEN R., 1997, *A Formal Approach to Software Architecture*, CMU-CS-97-144

[76] PFLEEGER S., 2001, *Software Engineering: Theory and Practice*, Prentice Hall, Second Edition, 2001

[77] LEVESON N., HEIMDAHL M. P. E., DAMON J., *Requirements Specification for Process-Control Systems*

[78] KJELLÉN U., 2000, *Prevention of accidents through experience feedback*, Urban Kjellén, Taylor & Francis

[79] PFLEEGER S. L., HATTON L., HOWELL C., 2001, *Solid Software*, 1 ed. Prentice Hall

[80] COLLINS R. *et al*, 1994, *How good is good enough*, Comm ACM, Vol. 37 No. 1

[81] REDMILL F., CHUDLEIGH M., CATMUR J., 1999, *System safety: HAZOP and Software HAZOP*, John Wiley & Sons

[82] LEVESON N., 2001, *Evaluating accident models using recent aerospace accidents*, MIT

[83] NIETZCHE F., 1886, *Beyond Good and Evil*, trad. Walter Kaufmann

[84] NIETZCHE F., 1882, *Gay Science*, trad. Walter Kaufmann

[85] POPPER K., 1989, *Conjectures and Refutations*, Rotledge & Kegan Paul plc

[86] POPPER K., 2000, *The Logic of Scientifica Discovery*, Rotledge

[87] CNN, 21/05/2003, *Shuttle mystery 'blessing in disguise'*

[88] RASMUSSEN J., 1988, *Reasons, Causes and Human Error*, New Technology and Human Error, ed. Jens Rasmussen, Keith Duncan e Jacques Leplat, John Wiley and Sons

[89] BAHR N., 1997, *System Safety and Risk Assessment: A Practical Approach*, Taylor & Francis

[90] REASON J., 1990, *Human Error*, Dept. of Psychology, Univ. Manchester, Cambridge University Press

[91] REASON J., 1997, *Managing the risks of organizational accidents*, Ashgate Publishing limited

[92] REASON J., MAURINO D., JOHNSTON N. LEE R., 1995, *Beyond Aviation Human Factors*, Ashgate Publishing Limited, 1995

[93] LEVESON N., KNIGHT J., DEWALL M., ELLIOT L. *et al*, 2001, *On Licensing Software Engineers Working On Safety-Critical Software*, Final Report of an ACM Task Force

[94] FENTON N., NEIL M., 1998, *A Strategy for Improving Safety Related Software Engineering Standards*, IEEE Transactions on Software Engineering, Vol, 24 No. 11

[95] ACM, *Software Engineering Code of Ethics and Professional Practice*, http://www.acm.org/serving/se/code.htm, consultado em 20/02/2004

[96] ROCHA, A. R. C., AGUIAR, T. C., SOUZA, J. M., 1990, *TABA: A Heuristic Workstation for Software development*, In: *Proceedings of COMPEURO 90*, Tel Aviv, Israel, Maio.

[97] MOURA, L. M. V., ROCHA, A. R. C., 1992, *Ambientes de Desenvolvimento de Software*, Publicações Técnicas COPPE/UFRJ, ES-271/92, Rio de Janeiro, Brasil

[98] TRAVASSOS, G. H., 1994, *O Modelo de Integração de Ferramentas da Estação TABA*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

[99] GARG, P., JAZAYERI, M., 1995, Introduction. In: Garg, P., Jazayeri, M. (eds), *Process-Centered Software Engineering Environments*, chapter 1, IEEE Computer Society Press

[100] FISCHER, G., 1996, *Seeding, Evolutionary and Reseeding: capturing and evolving knowledge in domain-oriented design environments*, In: Sutcliffe, A., Benyon, B. van Assche (eds) - IFIP 8. 1/13. *Joint Working Conference – Domain-Knowledge for Interactive System Design*, pp 1-16, Geneva, Switzerland, May.

[101] OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H., 1999, *A Domain-Oriented Software Development Environment for Cardiology*, In: *Proceedings of America Medical Informatics Association conference – AMIA*, Washington, D.C., Nov.

[102] OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.*, 1999, *Using Domain-Knowledge in Software Development Environments*, In: *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering*, pp. 180-187, Kaiserlauther, Alemanha, Jun.

[103] OLIVEIRA, K. M., ROCHA, A. R., TRAVASSOS, G. H. *et al.*, 1999, *O uso da Teoria do Domínio no Processo de Desenvolvimento de Software*, In: *Anais da X Conferencia Internacional de Tecnologia de Software*, pp 223-235, Curitiba, Brasil, Maio.

[104] FISCHER, G, 1994; *Domain-Oriented Design Environments*, Automated Software Engineering - The International Journal of Automated Reasoning and Artificial Intelligence in Software Engineering, Vol 1, N 2 (Jun), pp 177-203.

[105] OLIVEIRA, K. M., GALOTTA, C., ROCHA, A. R., et al., 2000, *Construção de Ambientes de Desenvolviemento de Software Orientados a Domínio*, In: Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes de Software IDEAS 2000 – Cancún, México, Abr.

[106] GALOTTA, C., 2000, *Netuno: Um Ambiente de Desenvolvimento de Software Orientado ao Domínio de Acústica Submarinha*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

[107] FOURO, A. M., 2002, *Apoio à Construção de Base de Dados de Pesquisa em Ambientes de Desenvolvimento de Software Orientados a Domínio*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

[108] VILLELA K., 2004, *Definição e Construção de Ambientes de Desenvolvimento de Software Orientados à Organização*, Tese de D Sc.

[109] MACHADO, L.F.C., 2000, Modelo para Definição, *Especialização e Instanciação de Processos de Software na Estação TABA*, Tese de M. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

[110] OLIVEIRA, K. M., 1999, *Modelo para Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio*, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

[111] IEC, IEC 61174, 2001, *Maritime navigation and radio communication equipment and systems - Electronic chart display and information system (ECDIS) - Operational and performance requirements, methods of testing and required test results*

[112] LEVESON N., 1995, *A new approach to system safety engineering*, M.I.T

[113] KAUFMANN W., BAIRD F., 2000, *From Plato to Derrida*, David Hume, Prentice-Hall

[114] C. W. JOHNSON, 2003, *Failure in Safety Critical Systems: A Handbook of Accident and Incident Reporting*, University of Glasgow Press, Glasgow, Scotland

[115] Patrícia, M. B., "Instanciação de Processos de Software em Ambientes Configurados na Estação Taba", Tese de M Sc, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, julho 2003

ANEXO I – PROCESSO MÍNIMO PARA DESENVOLVIMENTO DE SISTEMAS DE SEGURANÇA CRÍTICA

Atividade: Implementação do Processo

Sub-atividade: Implementar Programa de Segurança

Sub-atividade: Desenvolver Plano do Projeto

Tarefas:

- Planejar Acompanhamento e Controle
- Estabelecer cronograma
- Planejar Recursos Humanos
- Planejar Treinamento
- Planejar Recursos em Geral
- Planejar Custos
- Planejar Gerência de Configuração
- Planejar Riscos
- Planejar Segurança

Sub-atividade: Avaliar Plano do Projeto

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Análise dos Requisitos do Sistema

Sub-atividade: Entender o Domínio da Aplicação e Sistemas Similares

Sub-atividade: Identificar os Usuários do Sistema

Sub-atividade: Realizar Análise de Perigos Preliminar

Sub-atividade: Especificar os Requisitos do Sistema

Sub-atividade: Criar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar Requisitos do Sistema

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Projeto da Arquitetura do Sistema

Sub-atividade: Realizar Análise de Perigos do Sistema

Sub-atividade: Especificar uma Arquitetura de Alto Nível do Sistema

Sub-atividade: Atualizar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar Projeto do Sistema

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Planejar a Verificação da Segurança

Sub-atividade: Realizar Treinamento de Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Análise dos Requisitos do Software

Sub-atividade: Definir Fronteira do Software

Sub-atividade: Entender Domínio da Aplicação e Produtos de Software Similares

Sub-atividade: Análise de Perigos do Software

Sub-atividade: Especificar Requisitos Funcionais

Sub-atividade: Especificar os Requisitos de Qualidade e Outros Requisitos Não-Funcionais

Sub-atividade: Planejar Controle da Qualidade

Sub-atividade: Atualizar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar Requisitos de Software Especificados

Sub-atividade: Gerar Modelo de Análise

Sub-atividade: Atualizar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar Modelo de Análise

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Planejar a Verificação da Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Projeto do Software

Sub-atividade: Desenvolver o Projeto de Alto Nível, projetando para segurança

Sub-atividade: Atualizar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar Projeto de Alto Nível

Sub-atividade: Desenvolver o Projeto Detalhado, projetando para segurança

Sub-atividade: Atualizar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar Projeto Detalhado

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Codificação e Testes do Software

Sub-atividade: Planejar Testes de Unidades

Sub-atividade: Codificar Unidades

Sub-atividade: Realizar Análise de Perigos ao Nível do Código

Sub-atividade: Testar Unidades

Sub-atividade: Atualizar Matriz de Rastreabilidade de Requisitos

Sub-atividade: Avaliar o Código de Unidades

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Planejar a Verificação da Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Integração do Software

Sub-atividade: Planejar Teste de Integração do Software

Sub-atividade: Integrar Unidades de Software

Sub-atividade: Realizar Análise de Perigos ao Nível do Código

Sub-atividade: Planejar Testes de Qualificação do Software

Sub-atividade: Avaliar o Plano do Teste de Qualificação do Software e os Testes de Integração

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Teste da Qualificação do Software

Sub-atividade: Realizar Testes de Qualificação do Software

Sub-atividade: Verificar a Segurança

Sub-atividade: Atualizar Documentação do Usuário

Sub-atividade: Avaliar o Software

Sub-atividade: Realizar Auditorias

Sub-atividade: Preparar o Produto de Software a ser Entregue

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Integração do Sistema

Sub-atividade: Integrar Sistema

Sub-atividade: Realizar Análise de Perigos do Sistema

Sub-atividade: Planejar Testes de Qualificação do Sistema

Sub-atividade: Avaliar o Plano de Testes de Qualificação do Sistema e o Sistema Integrado

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Teste de Qualificação do Sistema

Sub-atividade: Avaliar Segurança

Sub-atividade: Realizar Homologação Interna

Sub-atividade: Realizar Homologação Externa

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Instalação do Sistema

Sub-atividade: Planejar Instalação

Sub-atividade: Completar a Documentação do Sistema

Sub-atividade: Elaborar a Documentação do Usuário

Sub-atividade: Avaliar a Segurança

Sub-atividade: Instalar o SIstema

Sub-atividade: Acompanhar e Atualizar Plano de Segurança

Sub-atividade: Refinar Planejamento para Próxima Atividade

Atividade: Apoio à Aceitação do Software

Sub-atividade: Avaliar a Segurança

Sub-atividade: Realizar Treinamento de Segurança

Sub-atividade: Apoiar a Avaliação do Usuário

Sub-atividade: Consolidar Rastreabilidade de Perigos

Sub-atividade: Certificar Sistema

Sub-atividade: Acompanhar e Atualizar Plano de Segurança