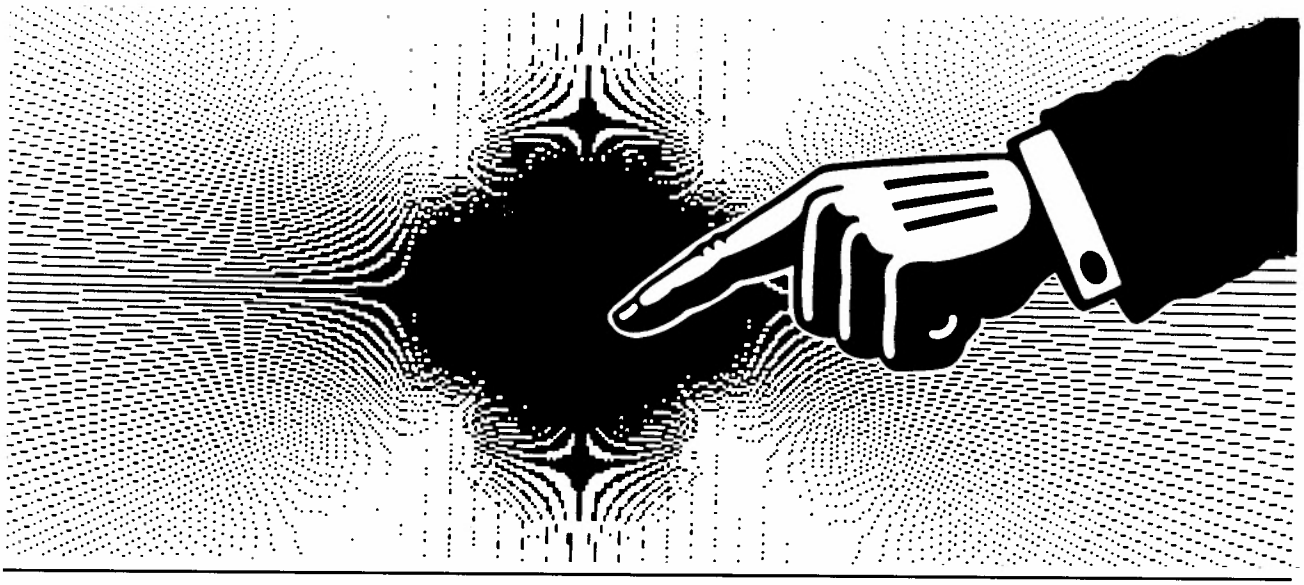
A INSEGURANÇA DO COMPUTADOR E A VULNERABILIDADE SOCIAL*



■ **Tom Forester**
Professor da Griffith University, Austrália.
■ **Perry Morrison**
Professor da University of New England, Austrália.

Tradução de **Antonio Carlos Mattos**, Professor do Departamento de Informática e Métodos Quantitativos da EAESP/FGV.

✻ **RESUMO:** *Muitos têm argumentado que as sociedades industriais estão se tornando cada vez mais dependentes da tecnologia e, conseqüentemente, mais vulneráveis às falhas tecnológicas. Não obstante a difusão da tecnologia computacional, pouco é conhecido a respeito das falhas do computador, exceto, talvez, as que são muito comuns. Este artigo analisa as fontes de insegurança do computador e revê a extensão e o custo dos computadores inconfiáveis. Ao contrário dos articulistas anteriores, os autores argumentam que os computadores digitais são inerentemente não confiáveis por duas razões: primeira, são mais propensos a falhas totais, ao invés de parciais; e segunda, sua enorme complexidade significa que jamais poderão ser completamente testados antes de serem colocados em uso. Os autores descrevem em seguida várias tentativas institucionais para melhorar a confiabilidade, bem como possíveis soluções propostas pelos cientistas da computação, mas concluem que até agora nenhuma foi adequada. Em conseqüência, recomendam que os computadores não devam ser utilizados em aplicações onde haja risco de vida.*

✻ **PALAVRAS-CHAVE:** *Confiabilidade do computador, falhas do computador, riscos sociais do computador, engenharia de software.*

✻ **ABSTRACT:** *Many have argued that industrial societies are becoming more technology-dependent and are thus more vulnerable to technology failures. Despite the pervasiveness of computer technology, little is known about computer failures, except perhaps that they are all too common. This article analyses the sources of computer unreliability and reviews the extent and cost of unreliable computers. Unlike previous writers, the authors argue that digital computers are inherently unreliable for two reasons: first, they are prone to total rather than partial failure; and second, their enormous complexity means that they can never be thoroughly tested before use. The authors then describe various institutional attempts to improve reliability and possible solutions proposed by computer scientists, but they conclude that as yet none is adequate. Accordingly, they recommend that computers should not be used in life-critical applications.*

✻ **KEY WORDS:** *Computer reliability, computer failures, computer-social risks, software engineering.*

* Texto publicado originalmente na *Futures Journal in England*, 22(5):462-74, junho de 1990, sob o título "Computer unreliability and social vulnerability".

INTRODUÇÃO

Em uma sexta-feira, 20 de novembro de 1987, um sabotador penetrou nos túneis da Telecom em Sidney, Austrália, e cuidadosamente danificou 24 cabos. Este simples ato colocou em pânico 35.000 linhas telefônicas em 40 subúrbios de Sidney, desativou centenas de computadores, bem como máquinas de consulta automática, pontos eletrônicos de venda, terminais de telex e de fax. Algumas empresas ficaram completamente fora de ação por 48 horas, enquanto os engenheiros batalhavam para restaurar o serviço telefônico, a um custo de milhares de dólares. Tivesse o sabotador a posse de um Plano que já não fosse de dois anos atrás, poderia ter havido um *blackout* em todo o sistema de telecomunicações da Austrália.

À medida que as sociedades se tornam mais e mais dependentes da tecnologia de computadores e das telecomunicações, também nos tornamos mais vulneráveis às suas falhas. Estas podem ser causadas por incêndios, inundações, terremotos e interrupções no fornecimento de energia elétrica, como também pelos ataques externos dos *hackers*, e pelas sabotagens internas de funcionários e ex-funcionários, como foi provavelmente o caso de Sidney.

Por exemplo, um grande incêndio que ocorreu na Central Telefônica de Setagaya, em Tóquio, em 16 de novembro de 1984, destruiu 3.000 circuitos de comunicação de dados e 89.000 circuitos comuns de telefonia, resultando em perdas totais, diretas e indiretas, da ordem de 13 bilhões de ienes para as empresas. Em 1987, um esquilo aventureiro derrubou todo o serviço de cotação automática da Associação Nacional de Corretores de Seguros, ao causar um curto-circuito em Trumbull, Connecticut, onde o computador central está instalado. Em julho de 1989, um operador de computador da Exxon admitiu ter inadvertidamente destruído cópias de milhares de documentos com informações potencialmente importantes sobre o famigerado derramamento de petróleo no Alasca. Em 1986, em Chicago, um funcionário da Enciclopédia Britânica, descontente por ter sido despedido, acessou o Banco de Dados da Enciclopédia e fez pequenas alterações no texto que estava sendo preparado para uma nova edição da renomada obra — como trocar as referências a Jesus Cristo por Alá, e colocar os nomes dos executivos da companhia em situações bizarras. Como disse um porta-voz da empresa, *"na era do computador, pesadelo é exatamente isto"*.

1. N.T.: A Proconsult alterou os votos dados a Brizola.

2. N.T.: Memória latch TTL de 4 *bits*.

O PROBLEMA DA NÃO-CONFIABILIDADE DO COMPUTADOR: FONTES, EXTENSÃO E CUSTO

As falhas dos sistemas de computação podem ser causadas por fatores externos, por erro humano ou por mau uso (ou pela combinação dos três), mas muitas mais são causadas pelas falhas do computador. Tipos conhecidos de falhas de computadores incluem a emissão das contas telefônicas e o chaveamento dos circuitos, controle de tráfego aéreo, consultas bancárias, transferência eletrônica de Fundos e licenciamento de veículos, muito embora os robôs industriais e os computadores da polícia também tenham dado sua contribuição à longa lista de enganos. Os erros de projeto e de programação de computadores foram os responsáveis pela falha na previsão do grande temporal inglês de outubro de 1987, e pelo caos nas eleições do Brasil em 1986.[1]

De fato, a tecnologia da computação e das telecomunicações tem figurado, de uma maneira ou de outra, em quase todas as recentes e famosas falhas de sistemas, desde os desastres de Three Mile Island, Chernobyl e do ônibus espacial Challenger, até a colisão na Antártida de aviões das Linhas Aéreas da Nova Zelândia, e a queda do vôo 007 das Linhas Aéreas Coreanas, sem mencionar o afundamento do navio britânico Sheffield na Guerra das Malvinas e o abate do *airbus* iraniano pelo USS Vincennes no Golfo Pérsico. Por "falha de sistemas" entenda-se desastre e perda de vidas humanas.

As falhas do computador podem ser geralmente divididas em falhas de *hardware* e de *software* (incluindo o projeto e a especificação do sistema). As falhas de *hardware* normalmente ocorrem quando componentes vitais do computador apresentam defeitos. Por exemplo, no dia 3 de junho de 1980, uma terça-feira, o sistema de alarme do Comando Aéreo Estratégico, perto de Omaha, Nebraska, indicou que dois mísseis balísticos lançados por um submarino estavam se dirigindo para os Estados Unidos. Logo depois, mostrou que mísseis balísticos intercontinentais (ICBMs) soviéticos também haviam sido lançados. Imediatamente, os tripulantes do bombardeiro B-52 receberam ordens para ligar os motores, os operadores dos mísseis baseados em terra foram colocados em alerta máximo, e um comando de patrulhamento aéreo foi colocado nos céus do Havai. Três minutos depois, o pessoal do Comando Aéreo Estratégico declarou que o alarme havia sido falso. A causa foi pesquisada, chegando-se ao *chip* defeituoso: um minúsculo 74175[2] de um computador Data General, usado como multiplexador de telecomunicação.

Em um outro caso de falha de *hardware*, um *chip* estabilizador, controlando o fluxo de combustível do motor dos carros Audi 5000, foi supostamente o culpado de 250 casos de aceleração súbita, alguns tendo causado acidentes fatais. Em julho de 1989, cabos partidos dentro de uma unidade de memória já antiga foram responsáveis pelas 104 falhas diárias do computador de controle do tráfego aéreo de Los Angeles, a despeito de gastos de milhões de dólares em um novo *software*.

Mas a maior parte das falhas de computador são causadas por problemas no *software*. O espaço aéreo, os militares e as viagens espaciais fornecem vários exemplos. Quando os pilotos da Força Aérea Americana estavam testando o caça F-16, a primeira coisa que fizeram, enquanto decolavam, foi dizer ao computador de bordo para levantar o trem de pouso. Um F-16 muito caro se espatifou. Em 1987, um outro F-16 mergulhou verticalmente no Golfo do México porque o computador de bordo não estava programado para voar em baixa velocidade. Em outro acidente militar, um F/A-18 tentou lançar um míssil que levava embaixo da asa. Embora tenha sido dada a ignição corretamente, o computador não conseguiu acionar as presilhas de destravamento do míssil. Assim, o infeliz piloto sentiu um empuxo adicional de 3.000 libras em uma das asas. Em outro conhecido engano de *software*, um foguete Atlas-Aegena, disparado de Cabo Canaveral em direção a Vênus, teve que ser explodido, depois de haver ficado maluco, porque faltou um simples hífen no programa de controle de vôo. A perda da sonda espacial Mariner 18 foi causada por um erro em uma das linhas do programa, enquanto que a União Soviética perdeu sua sonda marciana Phobos I de maneira semelhante.

Enganos de computadores em medicina também têm causado a perda de muitas vidas. Em 1980, um homem foi morto durante uma terapia por microondas para a artrite, devido a uma reprogramação do marca-passos; um *software* de uma bomba de insulina estava administrando a droga em doses erradas; e um sistema de monitoramento foi devolvido depois que se descobriu que estava trocando os registros e os nomes dos pacientes. O futuro da saúde do mundo também pode ter sido perturbado por erros nos programas a bordo dos satélites de observação da NASA durante os anos 70 e 80. Os programas haviam rejeitado as leituras de ozona na Antártida durante esse período, porque os valores eram baixos demais e foram interpretados como espúrios. Os desvios dos níveis normais eram tão grandes que foram assumidos como erros. Somente quando cientistas ingleses, usando instrumentos terrestres, descobriram que estava ocorrendo um declínio nos níveis de ozona, foi que os cientistas da NASA reprocessaram os dados desde 1979 e confirmaram os achados dos ingleses.

A extensão e o custo de computadores não confiáveis são imensos. Relata-se que as empresas inglesas sofrem cerca de 30 grandes desastres de computador por ano, envolvendo perdas de milhões de libras. São problemas causados por erros de máquina ou por falhas humanas e não incluem mau uso humano sob a forma de fraude ou sabotagem. Recentes relatórios da Price Waterhouse e da Logica também indicam que as falhas de *software* estão custando à indústria inglesa cerca de US$ 900 milhões por ano — e mesmo esse número inclui apenas o *software* que é desenvolvido localmente e vendido no mercado. Se for também incluído o *software* importado e o desenvolvido pelas empresas para uso próprio, os custos serão ainda maiores.

O Dr. John Collyer, um especialista em *software* do Estabelecimento Real de Radar e Sinais, do Ministério da Defesa da Inglaterra, alertou, em 1988, que *microchips* com erros iriam começar a matar pessoas em grande escala nos próximos anos, enquanto que um comitê da Sociedade Inglesa de Computação relatou, em 1989, que grande parte do *software* é tão complexo que os atuais requisitos de segurança são inadequados e, conseqüentemente, a segurança da população não pode ser garantida.

De fato, os erros no desenvolvimento e no uso de sistemas de computadores não são apenas lugar comum — são geralmente a regra. Um recente levantamento nos EUA, registrou o impressionante resultado de que 75% de todos os desenvolvimentos de sistemas levados a cabo, ou não são nunca terminados ou então não chegam nunca a ser utilizados, mesmo quando completados. Alguns articulistas têm identificado uma "crise" no desenvolvimento de sistemas, com a descoberta de que 70% dos projetos de *software* são geralmente destinados à manutenção (detecção e correção de erros), depois de o próprio sistema ter sido "verificado" e validado para uso. Outras pesquisas parecem também confirmar esse fato: um estudo de 1979, feito pelo Escritório Geral de Contabilidade (EGC) do Governo Americano, mostrou que, de nove projetos de *software* com um custo total de US$ 6,8 milhões, projetos no valor de 3,2 milhões (47% do total) foram distribuídos, mas não utilizados; US$ 1,3 milhões foram abandonados ou refeitos; e apenas US$ 200.000 chegaram a ser usados, depois de substanciais modificações. Incrivelmente, apenas um pro-

jeto valendo menos de US$ 100.000 acabou sendo utilizado da forma como foi fornecido pelo produtor.

Um relatório mais recente do EGC, sobre um antigo sistema de alerta da Força Aérea Americana (USAF) na Califórnia e no Colorado, mencionou dois enormes projetos de integração de sistemas da IBM, que estavam com atrasos de cinco anos e com gastos de centenas de milhões de dólares acima do orçamento. A estimativa inicial de US$ 600 milhões chegou aos 1,95 bilhões, antes de ser completado em 1993, afirmou o EGC.

Ao mesmo tempo, a Divisão de Sistemas Eletrônicos da USAF tem estado avaliando as potencialidades de seus fornecedores de *software*: 84% de seus 174 contratados obtiveram uma pontuação no nível mais baixo, com controles dos processos de desenvolvimento de *software* muito pobres ou mesmo caóticos; apenas 14% chegaram ao segundo nível, significando que poderiam repetir tarefas anteriores; e meros 2% atingiram o terceiro nível, com processos bem conhecidos. Nenhum contratado atingiu nem o quarto nem o quinto (e ótimo) nível.

Nesse meio tempo, um outro relatório do EGC, sobre o Escritório de Mineração de Superfície dos EUA, rotulou como "falido" um sistema computacional de US$ 15 milhões, desenvolvido para evitar que mineiros infratores obtivessem novas permissões para a exploração de minas, e o Departamento de Estado de Serviços Sociais na Virgínia abandonou um sistema de US$ 4,5 milhões, projetado para efetuar pagamentos a crianças maltratadas. O Estado da Virgínia está agora procurando um novo sistema que custará mais de US$ 10 milhões.

Exemplos mais espetaculares de enganos de computadores, ou de perdas no desenvolvimento de sistemas, incluem o Banco da América, tendo de abandonar em 1988 um sistema de US$ 20 milhões, depois de ter gasto cinco anos e um adicional de US$ 20 milhões tentando fazê-lo funcionar; e a Cia. de Seguros de Allstate vendo o custo de seu novo sistema de computação subir de US$ 8 milhões para espantosos US$ 100 milhões, com término atrasado de 1987 para 1993. O Pentágono, a cidade de Richmond, os estados de New Jersey e Oklahoma, a Cia. de Sistemas de Geofísica e a Cruz Azul de Wisconsin, todos sofreram recentemente grandes perdas. Além do mais, o problema parece estar ficando cada vez mais grave: em 1988 a Associação de Juízes Americanos trabalhou em 190 processos de computadores, a maior parte envolvendo sistemas falidos. As indenizações totalizaram US$ 200 milhões — bem acima dos US$ 31 milhões de 1984.

As falhas dos computadores e das telecomunicações têm uma história não muito elogiável. De volta a 1960, por exemplo, computadores do Sistema de Alerta Prévio de Mísseis Balísticos, em Thule, dispararam um alerta, quando interpretaram a subida da lua na linha do horizonte como um ataque nuclear. Em um outro caso, um bando de gansos foi admitido como sendo um grupo perdido de ogivas nucleares.

Mais recentemente, no Golfo Pérsico, um cruzador americano armado com mísseis, o USS Vincennes, abateu um *airbus* iraniano, causando a morte de 290 civis. Como resultado, foram levantadas críticas ao sistema de defesa aéreo Aegis, a constelação de radares, computadores e mísseis que levaram a cabo o ataque. Além do mais, o Aegis tinha sido largamente condenado por críticos, bem antes de ter sido instalado em navios americanos, a um custo de US$ 1,2 bilhões por unidade. Por exemplo, foi mencionado que o Aegis passou nos testes de capacitação, realizados em um milharal — o que não é exatamente o mesmo que alto-mar.

Aegis é um sistema de administração de batalhas, projetado para seguir objetos aéreos em um raio de 300 km e destruir até 20 alvos simultaneamente, usando uma variedade de mísseis. O sistema inclui radares avançados e um *software* complexo, que identifica alvos potenciais a partir de uma biblioteca de atacantes, e seleciona as armas correspondentes. Ainda em seu primeiro teste operacional, Aegis falhou ao não conseguir derrubar seis alvos em cada 17, mesmo quando apresentados apenas três por vez. É difícil determinar no final que parcela da tragédia do *airbus* pode ser atribuída a erros de computador e a falhas humanas, mas Aegis aparentemente falhou ao não distinguir entre um avião civil de carreira e um moderno caça, a uma distância de 17 km.

O estudo de casos militares fornece ricas fontes de informações sobre a falta de confiabilidade do computador, embora as falhas em sistemas civis possam também colocar vidas em risco e causar perturbações sociais generalizadas. Por exemplo, em 6 de agosto de 1988, um sábado, durante um dos dias mais agitados do ano, o aeroporto Heathrow de Londres deveria estar coordenando cerca de 830 pousos e decolagens, quando o sistema de controle do tráfego aéreo falhou. Não obstante o emprego de 70 sistemas especialistas rodando no computador 24 horas por dia, já tinha havido cinco falhas semelhantes nos doze meses anteriores. Heathrow utiliza um sistema de controle de tráfego conhecido como

Pacote do Espaço Aéreo Nacional, contendo em torno de um milhão de linhas de codificação e que precisou de 1.600 homens-ano para ser escrito e de 500 homens-ano para aperfeiçoamentos posteriores.

Uma falha potencialmente muito mais perigosa ocorreu em 23 de agosto de 1987, quando o Centro Oceânico dos Serviços Nacionais de Tráfego Aéreo, em Prestwick, Escócia, descobriu que seu Sistema de Processamento de Vôo — o computador que controla o grosso dos vôos transatlânticos — havia entrado em pânico. O sistema deixou de funcionar às 11:30h da manhã e, à tarde, Heathrow, Paris, Frankfurt, Zurich e os outros principais aeroportos da Europa começaram a ter que sair fora das áreas de estacionamento para as aeronaves em atraso, muitas com passageiros a bordo.

Outras falhas de sistemas, embora menos ameaçadoras para as vidas humanas, ilustram até que ponto o sangue vital de nossas economias — dinheiro e capital — tem se tornado controlado por sistemas tão complexos que beiram as fronteiras de nossa compreensão. Uma boa ilustração foi o que aconteceu na manhã de 20 de novembro de 1985, quando mais de 32.000 transações de seguro do governo estavam esperando para serem processadas no Banco de Nova York. Às 10h da manhã, os sistemas de computadores do Banco começaram a destruir as transações, ao gravar umas sobre as outras. Como conseqüência, era impossível para o Banco determinar quais os clientes que deveriam ser cobrados, os prêmios de seguro e os valores correspondentes. Nesse interim, o Banco da Reserva Federal de Nova York continuava e emitir os prêmios de seguro para o Banco de Nova York e a debitar sua conta Caixa. Ao se encerrar o expediente desse dia, o Banco estava a descoberto em US$ 32 bilhões com a Reserva Federal — e, a despeito dos frenéticos esforços para consertar os programas, o Banco estava ainda a descoberto no dia seguinte no valor de US$ 23 bilhões. Mais tarde, neste mesmo dia, o *software* foi finalmente colocado em ordem, mas o fiasco custou ao Banco nada menos que US$ 5 milhões de juros perdidos no *overnight*.

Existem muitos outros exemplos que ilustram a crescente vulnerabilidade da sociedade às falhas dos computadores — como o caso em que foi necessário fechar a Rede Ferroviária Nacional da Austrália e reverter os controles de sinalização para serem operados manualmente[3], depois de haver falhado o *software* de controle; o da enchente de Fresno na Califórnia, quando o sistema hidráulico urbano, controlado por computadores, entrou em pânico; o desastre do *airbus* A320 na França etc.

Um exemplo final da falibilidade do *software* demonstra como as prováveis aplicações futuras da Inteligência Artificial e dos Sistemas Especialistas poderão causar problemas éticos e legais ainda maiores para a sociedade. Uma moça de Nevada, a Srta. Julie Engle, foi submetida a uma cirurgia de rotina em um hospital. A operação foi realizada sem complicações. No entanto, logo depois, foi administrado à Srta. Engle um analgésico por uma máquina computerizada. Infelizmente, o sistema, por engano, instruiu as enfermeiras para injetar mais de 500 mg da droga no corpo da Srta. Engle e, 30 minutos após o término de uma operação bem-sucedida, ela foi encontrada em estado de coma. Cinco dias depois, foi declarada sua morte cerebral. A Srta. Engle, na verdade, era a secretária do Sr. Vibert Kesler, um advogado de Salt Lake City, que imediatamente processou o hospital por danos, devido ao uso incorreto e irresponsável de um Sistema Especialista. Esse caso levanta toda sorte de questões, como quem deve ser processado quando um Sistema Especialista comete um erro, mas o ponto importante aqui é que, se este tipo de tragédia pode acontecer com uma aplicação relativamente simples da Inteligência Artificial, imagine-se o que pode ocorrer com algumas aplicações mais complexas previstas para a sociedade.

POR QUE OS COMPUTADORES SÃO TÃO POUCO CONFIÁVEIS

Os problemas da não confiabilidade dos computadores e da vulnerabilidade das sociedades dependentes da tecnologia computacional já têm sido levantados. Mas a maior parte das contribuições até hoje têm se contentado em levantar a questão, a listar diferentes tipos de vulnerabilidades, ou a destacar a pouca segurança dos sistemas computacionais. Nenhuma parece oferecer uma explicação adequada da causa de serem os computadores tão inconfiáveis, e porque uma sociedade computadorizada é tão vulnerável às falhas do computador.

Por exemplo, em 1978, os suecos publicaram um relatório sobre a vulnerabilidade da sociedade computadorizada, onde foram listadas 15 fontes de vulnerabilidade, incluindo atos criminais e de guerra, mas apenas uma pequena parte foi devotada ao *hardware* e ao *software* com defeitos. A conclusão foi que a vulnerabilidade de uma sociedade computadorizada é "inaceitavelmente alta". Alguns anos mais tarde, a Comunidade Européia organizou um seminário na Espanha sobre o mesmo assunto, onde opiniões similares foram expressas, mas nenhum método analítico

3. N.T.: Isto é o que se chama nos EUA de "Sistema I.B.M.": *It's Better Manually* (é melhor fazer à mão).

novo foi oferecido. Dois relatórios posteriores, do EGC e da Comissão Européia, fizeram observações parecidas, ambos enfatizando que os sistemas computacionais estavam inadequadamente protegidos do mau uso humano sob a forma de fraude, perda, sabotagem etc.

Uma análise mais lúcida foi oferecida em 1988 por consultores da Coopers & Lybrand, que classificaram as falhas dos sistemas computadorizados em uma dúzia de tipos principais: incêndio, inundação e desastre; sabotagem e vandalismo; falhas de *hardware*, de *software* e de telecomunicações; perda de serviços (eletricidade, telefonia etc.); sobrecarga do sistema; roubo; acesso não controlado; introdução de dados incorretos; erros do sistema; falha humana; alterações incorretas; e greves. Mas, novamente, as falhas de *software* e de *hardware* não receberam atenção especial.

O problema de todas essas análises é que elas falham ao não chamar a atenção para a não-confiabilidade do computador. Nós argumentamos que os computadores são inerentemente não-confiáveis devido a duas razões principais: primeiro, são propensos a falhas catastróficas; e, segundo, sua grande complexidade garante que não possam ser exaustivamente testados antes de serem liberados para uso. Nosso argumento sobre a primeira contestação é melhor ilustrado com alguns exemplos adicionais.

Desde 1982, nada menos de 22 militares morreram em cinco colisões diferentes do sofisticado helicóptero da USAF UH-60, o Tubarão Negro. Em cada ocasião, as máquinas ou rodopiaram sem controle ou caíram batendo com o nariz no chão. Mas, foi somente em novembro de 1987, depois que essa série de colisões misteriosas foram amplamente investigadas, que os oficiais da Força Aérea finalmente admitiram que o UH-60 era inerentemente susceptível a rádio-interferências em seu sistema de controle computerizado *fly-by-wire* (FBW, vôo pelo fio), e que modificações no helicóptero seriam necessárias.

Os sistemas FBW funcionam pela eliminação das conexões mecânicas entre os controles dos pilotos e as asas e leme horizontal (ou, no caso de helicópteros, do rotor principal e traseiro). Em outras palavras, os movimentos analógicos de controle de um *joystick* (semelhante aos usados em videogames) ou de colunas controladoras são convertidos em pulsos digitais, os quais, por sua vez, acionam servomotores e outros sistemas de potência, que movimentam os *ailerons* e os lemes de direção e de profundidade. Com efeito, os pilotos desses aparelhos não mais possuem uma ligação física conectando seus controles às superfícies a serem controladas. Ao invés disso, computadores de bordo "interpretam" os movimentos de controle e enviam essas informações às unidades de acionamento. No caso do Tubarão Negro, uma blindagem inadequada em alguns dos módulos lógicos do sistema FBW possibilitou o aparecimento de "poluição eletrônica" nos circuitos, induzida através de micro-ondas e de outras transmissões de rádio. Por isso, o computador de bordo às vezes enviava sinais espúrios aos sistemas hidráulicos, ocasionando as colisões.

Em um dia do mês de março de 1986, Ray Cox visitou uma clínica em Tyler, Texas, para receber tratamento por radiação nas costas, de onde havia sido extraído um tumor canceroso. O tratamento normalmente é indolor, mas naquele dia ele recebeu a marca de um golpe. Técnicos intrigados, que estavam do lado de fora da sala de terapia, observaram que o computador que operava o equipamento fazia piscar a mensagem "Falha 54", mas não conseguiam saber do que se tratava. Na realidade, Cox havia recebido uma sobredose fatal de radiação, e cinco meses depois estava morto. Menos de um mês após Cox haver sido tratado, a Falha 54 ocorreu novamente — com a única diferença que, desta vez, uma senhora de 66 anos, Vernon Kidd, morreu em 30 dias. O equipamento em uso no Centro de Tratamento de Câncer em Tyler era o acelerador linear Therac-25, controlado por computador, e máquinas iguais ou semelhantes estão em uso em 1.100 clínicas americanas, administrando terapia por radiação a 450.000 novos pacientes por ano.

Nos antigos aceleradores eletromecânicos, forças elétricas e mecânicas eram usadas para transmitir comandos do operador para rodas, elevadores e acionadores que controlavam o feixe de radiação. Com computadores, apenas a informação é transmitida — instruções de *software* dizem à máquina o que fazer de forma análoga ao usado no sistema FBW do Tubarão Negro (e também no *airbus* A320). Antigamente, falhas eletromecânicas estavam associadas a defeitos de materiais, tais como hastes quebradas ou cabos partidos. Mas as falhas do computador existem dentro dos próprios *chips* — ou, mais comumente, são causadas por defeitos no *software*. Um erro de *software* matou Cox e Kidd.

Computadores digitais são dispositivos de estados discretos — isto é, usam representações digitais (binárias) dos dados (na memória, fita, disco ou outro meio) e instruções (i.e., programas) — de modo que um programa de computador pode efetivamente "existir" em literalmente milhões ou mesmo bilhões de estados diferentes. Assim, cada mudança em uma variável (e alguns programas têm milha-

res de variáveis com milhares de valores cada uma) efetivamente alteram o estado do sistema. Cada dado introduzido ou emitido, acesso a disco, solicitação para imprimir, conexão a modem, ou cálculo — de fato, qualquer coisa em que o sistema possa estar envolvido — altera o estado no qual o sistema existe. Multiplique todos esses estados, e também outros relevantes (hora do dia, sistema em carga, combinação de tarefas em andamento etc.), um pelo outro e você terá o número de estados possíveis nos quais o sistema poderá existir.

O primeiro problema dos computadores digitais é que cada um desses estados representa uma fonte de erro em potencial. Sistemas analógicos, ao contrário, têm um número infinito de "estados" — são na realidade contínuos — no sentido de que, digamos, um botão de controle de volume de um rádio ou uma tira bimetálica usada em um termostato pode tomar um número infinito de posições dentro de certa faixa, da mesma forma que uma régua possui um número infinito de pontos. A diferença real, entretanto, é que, enquanto o movimento contínuo do botão do rádio dificilmente terá uma falha catastrófica, nossa máquina de estados discretos pode falhar de uma maneira catastrófica, porque a execução de cada estado depende de estar o estado anterior "correto", ou seja, ter atingido o objetivo computacional que o programador esperava conseguir. Se o estado anterior não estiver correto, ou mesmo se impedir que a próxima instrução seja executada, então o programa irá falhar. Se o programa pára ou se comporta erraticamente, então isto é chamado de uma "descontinuidade". Geralmente, dispositivos analógicos tendem a ter poucas ou nenhuma descontinuidade. Em outras palavras, você dificilmente encontrará uma situação onde um dispositivo analógico esteja funcionando perfeitamente, e então atinja um ponto onde operará aberrantemente ou falhará totalmente. Um computador desempenhando a mesma função, no entanto, pode falhar de uma maneira catastrófica e em uma série surpreendentemente grande de situações.

O segundo problema dos computadores digitais é que são tão complexos que é impossível saber ou predizer todos os possíveis estados nos quais o sistema poderá se encontrar — e conseqüentemente é inviável testá-lo completamente antes de começar a ser usado. Um rápido cálculo comprova nossa afirmação. Suponha, por exemplo, que um sistema seja projetado para monitorar 100 sinais binários, de modo a determinar o desempenho de um complexo industrial, digamos, de uma usina nuclear. Esse número de monitores certamente não é excessivo. Dadas essas 100 fontes de sinais, existem então 2^{100} ou $1{,}27 \times 10^{30}$ combinações possíveis de sinais de entrada. Agora, o caminho que um tal programa segue depende da combinação dos sinais recebidos anteriormente em qualquer período de tempo (os valores dos sinais podem ser atualizados em poucos milésimos de segundos). Isto é, se uma combinação particular de sinais é registrada, então uma parte particular do programa será executada, enquanto que outras partes diferentes do programa podem ser executadas para outras configurações da entrada. Dado o grande número de subrotinas em um programa desse tamanho, poderá haver pelo menos 10.000 ou mais possíveis caminhos através dele. Isto significa que o sistema poderá existir em pelo menos $1{,}27 \times 10^{30}$ estados possíveis, cada um deles podendo fazer com que o *software* ou falhe ou retorne uma informação errada.

Agora, se alguém quisesse testar empiricamente um tal programa para identificar os estados incorretos e então corrigi-los, e se alguém pudesse fazer isto automaticamente à razão de, digamos, 100 testes por segundo (o que, incidentalmente, está além das atuais capacidades), então os testadores de *software* necessitariam de 4×10^{24} anos para exaustivamente checar o conjunto de possíveis estados que o *software* poderia ter. Infelizmente, esse número de anos é várias vezes a idade do Universo. Além do mais, alguns desses estados envolverão subrotinas de recuperação de erros — módulos que foram desenhados para detectar erros e corrigi-los — que, para serem testadas, o programa deve ser modificado no sentido de simular esses estados de erro. No entanto, essas modificações em si podem causar novas fontes de erros. De fato, Adams estimou que 15 a 20% das tentativas de remoção de erros em programas, na prática, introduzem novos erros. Para complicar ainda mais, é hoje aceito que, para programas com 100.000 a 2 milhões de linhas de codificação, as chances de introduzir um erro grave durante a correção de um erro original é tão grande que somente uma pequena fração do erro original deve ser corrigido. Em outras palavras, algumas vezes é melhor estar preparado para um erro e tentar contorná-lo, informando aos usuários as condições em que ocorrerá, do que tentar corrigi-lo e acabar por criar erros ainda piores. Programadores honestos admitem ser impossível escrever um programa complexo isento de erros.

Dada a incidência de *software* com erros e de falhas de sistemas, não será surpresa constatar que os fabricantes de *software* raramente

fornecem garantias de qualquer tipo aos seus compradores ou clientes. Enquanto que geladeiras, carros, máquinas de lavar e praticamente qualquer outro bem de consumo são vendidos com garantias de funcionamento e de fabricação, o mesmo não acontece com o *software*. Alguns pacotes de *software* contêm termos de garantia em complicada linguagem jurídica só para informar aos consumidores que seus produtos não possuem qualquer garantia.

O QUE OS CIENTISTAS DE COMPUTAÇÃO ESTÃO FAZENDO A RESPEITO?

Os cientistas da computação têm tradicionalmente estabelecido confiabilidades para o computador, fazendo estimativas da probabilidade de um *software* ou *hardware* falhar. Por exemplo, Peter Mellor define a confiabilidade de um sistema de computação como *"a probabilidade de que ele falhe durante um dado período de tempo de operação e sob dadas condições"*. Assim, as medidas de confiabilidade podem incluir o número de falhas por unidade de tempo de operação, e o período de tempo decorrido até que o sistema falhe. Como muitos cientistas da computação, ele advoga a aplicação de princípios estatísticos à qualidade do *software* de modo que, por exemplo, seja mais aceitável ter muitos erros pouco freqüentes do que poucos erros muito freqüentes.

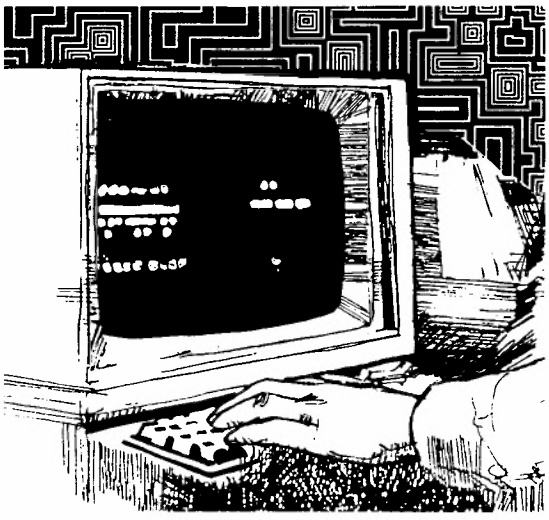
Isto simplesmente confirma a visão de que a programação de computadores é essencialmente uma "magia negra", em lugar de ser uma ciência rigorosa — um esforço altamente criativo mas que também é um pouco errático. Como Jonathan Jacky apontou, *"... o software permanece grandemente sem regulamentação ... Não há qualquer padrão educacional exigido para os programadores e muitos currículos de Ciências da Computação nem sequer mencionam o aspecto segurança. Estudos mostraram que os melhores programadores podem ser 25 vezes mais competentes do que o pior deles, e que muitos supervisores de projeto de software não estão capacitados a avaliar ou mesmo a compreender o trabalho de seus programadores"*.

Mas muitos acreditam que isto simplesmente ainda não está bom, especialmente em aplicações críticas. Nesse sentido, o Ministério da Defesa da Inglaterra publicou, em maio de 1989, o esboço de um padrão de *software* para aplicações militares, conhecido por 00-55. Esse esboço propõe a verificação formal de *softwares* críticos de segurança, isto é, o uso de métodos matemáticos complexos e de raciocínio lógico para verificar se o programa está correto. Nada parecido está previsto pelo Departamento da Defesa dos EUA, embora a Organização da Aviação Civil Internacional (uma organização internacional que regulamenta os sistemas de controle de tráfego) pretenda ir além desses assim chamados Métodos Formais. A publicação 00-55 tem levantado grandes controvérsias dentro dos círculos de desenvolvimento de *software* na Inglaterra, havendo protestos e apoios sobre a necessidade de — e sobre a possibilidade de a indústria usar — técnicas matemáticas. Por exemplo, o presidente do Comitê Especial Sobre Segurança de *Software* da Sociedade Britânica de Computação, Martyn Thomas, envolveu-se em uma grande disputa com o questionamento feito pela Empresa Central de Geração de Eletricidade, em Hinkley Point, acerca da validade de os *softwares* de Centrais Nucleares seguirem os padrões do tipo 00-55.

Até agora, a resposta da Ciência da Computação aos problemas de confiabilidade de *software* tem sido chamada de "Engenharia de *Software*". Este rótulo engloba todos os métodos, técnicas e ferramentas intelectuais utilizados no projeto e desenvolvimento de *softwares* de melhor qualidade. A maior parte desse trabalho pode ser agrupado em quatro grandes assuntos.

A primeira área é a "Programação Estruturada", derivada das críticas às codificações do tipo espaguete, que os programadores costumavam produzir usando as antigas linguagens de alto nível como FORTRAN e COBOL. Linguagens mais recentes, pela própria natureza de suas concepções, impõem estruturação e modularidade, bem como transparência de detalhes, de modo que programadores individuais de grandes projetos não necessitam se concentrar em pequenas particularidades, que dizem mais respeito ao Gerente de Projetos ou àqueles que supervisionam a produção de *software*. Esta tendência de produzir linguagens de programação mais poderosas ainda prossegue, mas alguns críticos, como David Parnas, duvidam que isso levará a melhorias de ordens de magnitude, como necessário se faz na Engenharia de *Software*, se programas confiáveis devam ser produzidos regularmente.

Uma segunda resposta parcial à questão da melhoria do *software* reside no desenvolvimento de "Ambientes de Programação". São essencialmente sistemas operacionais e conjuntos de ferramentas de *software* que ajudam os programadores, ao propiciar maneiras flexíveis e poderosas de administrar muitas das complexidades do desenvolvimento de *software*. Assim, ambientes como o UNIX propiciam poderosas facilidades para gerenciar diferentes versões de programas, atualizar arquivos afetados por modificações, so-



fisticada identificação de erros etc. Mas, novamente, embora esses ambientes se constituam em um avanço significativo no manuseio das complicações e complexidades do desenvolvimento de *software*, também não conseguem produzir as melhorias do porte que se fazem necessárias, isto se o *software* tiver que satisfazer aos padrões de confiabilidade que temos esperado em outras áreas da engenharia.

Um terceiro domínio de pesquisa reside, como já indicamos, na área de "Verificação e Derivação de Programas". Este campo diz respeito ao desenvolvimento de técnicas matemáticas para provar que o programa está correto, após ter sido escrito (verificação), ou para mostrar que o programa está certo, durante o processo de elaboração (derivação). Infelizmente, essas técnicas ainda não estão aptas a manusear os programas, mesmo os de tamanho modesto — a matemática é geralmente maior que os próprios programas! — ou os demandados pela maioria das aplicações comerciais e administrativas, sem mencionar os megaprojetos militares do tipo Iniciativa de Defesa Estratégica.[4] Além do mais, mesmo uma prova de correção apenas diz que a especificação do programa está em ordem — não informa se as especificações irão de fato satisfazer às (algumas vezes inadequadamente articuladas) necessidades do usuário. Nem, como menciona Alan Borning, informa como o sistema irá se comportar em situações nunca dantes imaginadas, quando as especificações foram definidas: *"São as circunstâncias inesperadas e as interações que ocasionam os problemas mais graves"*. Assim, "Prova de Correção" é uma designação incorreta — um termo melhor teria sido "Prova de Consistência Relativa".

Por último, os aspectos da "Administração Humana" do desenvolvimento de *software* também têm sido abordados, e uma variedade de metodologias e de práticas de administração de projetos tem se tornado moda. Os pressupostos dessas metodologias são que, através de um controle e de um gerenciamento apropriado do processo de desenvolvimento de *software*, em seus aspectos humano-organizacionais (especificação das necessidades do usuário, prototipação, documentação, consulta posterior ao cliente, teste e verificação, etc.), um produto de melhor poderá ser fornecido com maior confiabilidade.

CONCLUSÃO

Somos assim forçados a concluir que a elaboração de *Software* é um processo complexo e difícil, e que as técnicas existentes de Engenharia de *software* ainda não conseguem produzir *software* com garantia de qualidade e de confiabilidade. No caso de sistemas grandes e complexos, aos quais delegamos grandes responsabilidades sociais, e por vezes energias assustadoras, isso é extremamente lamentável. E nem a situação está com jeito de melhorar no curto prazo: a não confiabilidade do computador continuará sendo uma grande fonte de vulnerabilidade social ainda por algum tempo. Conseqüentemente, recomendamos que, por enquanto, não devam ser delegados aos computadores aplicações com risco de vida, e que isso só se faça quando a confiabilidade melhorar dramaticamente no futuro.

Dadas as evidências apresentadas aqui, vários artigos éticos também surgem para o pessoal da indústria da computação. Por exemplo, quando os programadores e engenheiros de *software* forem solicitados a construir sistemas que contenham aplicações com risco de vida, deverão eles ser mais honestos sobre os perigos e as limitações? Deverão ser os profissionais da computação responsabilizados, se um sistema falhar? Por exemplo, se um paciente morre em uma mesa de operações devido a um *software* defeituoso, será o programador culpado de homicídio não premeditado, ou de imperícia, ou de nada? Qual é o *status* ético das garantias existentes e de suas limitações declaradas? Como é que a indústria de computadores quase que sozinha é capaz de vender produtos que não podem ser garantidos contra falhas?

São esses os tipos de questões que devem ser levantadas junto ao governo, aos compradores de *software* e ao público em geral, em caráter de urgência, uma vez que os fornecedores de computadores e as indústrias de *software*, por si só, dificilmente tornarão públicas as sérias falhas de seus produtos.

4. N.T.: Projeto "Guerra nas Estrelas" de Reagan.

BIBLIOGRAFIA

■ *A CONSOLIDATED report of case study findings*. Londres, Coopers and Lybrand, 1988.

■ ADAMS, E. N. "Optimizing preventing service of *software* products". *IBM Journal of Research and Development, 28*(1):8, 1984.

■ "AI IN medical diagnosis and aviation". *Software Engineering Notes, 11*(2):5, abr. 1986.

■ "AIR Force misses target". *Computing Australia*, 21 ago. 1989.

■ BASSEN, H. et alii. "Computerized medical devices: usage trends, problems and safety technology". In: *Proceedings of the 7th Annual Conference of IEEE Engineering in Medicine and Biology Society*. Chicago, IL, 27-30 set. 1985, pp. 180-5.

■ BLACK, D. *The Independent*, 24 set. 1987.

■ BORNING, Alan. "Computer system reliability and nuclear war". *Communications of the ACM, 30*(2):113, fev. 1987.

■ BUBENKO, J. A. *Information Systems Methodologies - A Research View*, SYSLAB Report nº 40, Suécia, University of Stockholm, 1986.

■ *BUSINESS Week*, 29 mai. 1989 e 17 jul. 1989.

■ CANNING, R. G. "Getting the requirements right". *EDP Analyzer, 15*(7):1-14, 1977.

■ CHAYES, A. & WIESNER, J. *Underestimates and Overexpectations* in *ABM*. New York, Harper and Row, 1969, pp.122-3.

■ COATES, James & KILIAN, Michael. *Heavy Losses: the Dangerous Decline of American Defense*. New York, Penguin, 1986.

■ COMPUTER *News*, 22 out. 1987.

■ COOPER, B. & NEWKIRK, D. *Risks*, nov. 1987.

■ "DISPUTE over drug death". *The Australian*, 8 set. 1987.

■ ELLUL, Jacques. *The Technological Society*. New York, Vintage Books, 1964.

■ "EXXON man destroys oil spill documents". *The Australian*, 4 jul. 1989.

■ "FEDERAL Information systems remain highly vulnerable to fraudulent, wasteful, abusive and illegal practices". A Report by the US General Accounting Office (GAO), Masad 82-28, 21 abr. 1982.

■ FORESTER, Tom. "Death, injury and damage - the dire tale of systems failure". *The Australian*, 10 out. 1989.

■ FORESTER, Tom. "Programming or chaos: a litany of great and grating *software* disasters". *The Australian*, 4 abr. 1989.

■ FORESTER, Tom & MORRISON, Perry. *Computer Ethics: Cautionary Tales and Ethical Dilemmas in Computing*. Oxford, Basil Blackwell, 1990, e Cambridge, MA, MIT Press, 1990.

■ GLADDEN, G. R. "Stop the lifecycle, I want to get off". *Software Engineering Notes, 7*(2):35-9, abr. 1982.

■ HOFFMAN, Lance J. & MORAN, Lucy M. "Societal vulnerability to computer system failures". *Computers and Security, 5*:211-17, 1986.

■ HOLVAST, Jan. "Vulnerability of information society". In: CLARKE, Roger & CAMERON, Julie (orgs.) *Proceedings of SOST'89*. Sidney, Austrália, Computer Society, 1989.

■ HOPCRAFT, J. E. & KRAFT, D. B. "Toward better computer science". *IEEE Spectrum*, dez. 1987, pp.58-60.

■ INCE, Darrel. "Problems of applied maths". *The independent*, 30 jan. 1989.

■ "INFORMATION system failures - a survey and classification of the empirical literature". *Oxford Surveys in Information Technology, 4*, 1987, pp.257-309.

■ JACKY, Jonathan. "Programmed for disaster - software errors that imperil lives". *The Sciences*, set./out. 1989.

■ "LAID-off worker sabotages encyclopedia" *San Jose Mercury News*, 5 set. 1986.

■ LAMB, John. "Computer crashes and the stranded traveller - air traffic control in Britain". *New Scientist*, 8 set. 1988, p.65.

■ LIKEWISE, David Shepherd & WILSON, Greg. "Making *chips* that work". *New Scientist*, 13 mai. 1989, pp.39-42.

■ MELLOR, Peter. "Can you count on computers". *New Scientist*, 11 fev. 1989.

■ "MORE on proper British programs". *Software Engineering Notes* 14(1):9, jan. 1989.

■ MORRISON, Perry. "An absence of malice: computers and armageddon". *Prometheus*, 2(2):190-200, 1984.

■ *NEW Scientist*, 3 mar. 1988.

■ "OECD workshop stresses dependency on computers". *Transnational Data Report*, 4(5):3-4, mai. 1981.

■ PARNAS, David L. "The Parnas papers". *Computers and Society*, 14(9):27-36, 1985.

■ PERROW, Charles. *Normal Accidents: Living with High-Risk Technologies*. New York, Basic Books, 1984.

■ *RISKS*, 30 jun., 6 e 12 jul. 1989

■ ROTHFEDER, Jeffrey. "It's late, costly, incompetent - but try firing a computer system", e "Using the law to rein in computer runaways", *Business Week*, 7 nov. 1988 e 3 abr. 1989.

■ "SABOTEUR tries to blank out Oz". *The Australian*, 23 nov. 1987, p.1.

■ SIBLEY, E. "The evolution of approaches to information systems design methodologies". In: OLLE, T. W.; SOL, H. & VERRIJN-STUART, A. (orgs.) *Information Systems Design Methodologies: improving the practice*, Amsterdam, Nort-Holland, 1986, pp.1-17.

■ *SOFTWARE Engineering Notes*, *10*(2):6, abr.1985, *10*(5):10, out. 1985, *11*(1):9, jan. 1986, *11*(2), abr. 1986, *11*(3):14, jul. 1986, *12*(1), jan. 1987, *13*(3):4, jul. 1988, *13*(4):3-5, out. 1988 e *14*(2):5-6, abr. 1989.

■ "SOFTWARE of strategic defense systems". *American Scientist*, 73(5):432-40, set./out. 1985.

■ "STRAY rodent halts Nasdaq computers". *The New York Times*, 10 dez. 1987, p.33.

■ TAKNASHI, Naruko; TANAKA, Atsushi; YOSHII, Hiraoki & WADA, Yuji. "The Achilles heel of the Information Society: socioeconomic impacts of the telecommunications cable fire in the Setagaya telephone office, Tokyo". *Technological Forecasting and Social Change*, 34(1):27-52, 1988.

■ *THE AUSTRALIAN*, 2 fev. 1988 e 21 fev. 1989.

■ *THE DAILY Telegraph*, 3 nov. 1986.

■ *THE GUARDIAN*, 10 nov. 1986.

■ *THE GUARDIAN Weekly*, 17 jul. 1988.

■ *THE NEW York Times*, Science Times Section, 29 jul. 1986, p.C1.

■ *THE SYDNEY Morning Herald*, 4 ago. 1988, p.1-11

■ *THE VULNERABILITY of Computerized Society* e "Considerations and Proposals". Stockholm, Suécia, SARK, (Sarbarhetskommitten), 1978 e 1979.

■ *THE VULNERABILITY of the information conscious society - European situation*, Brussels, Commissions of the European Community, Information Task Force, 1984.

■ US Government Accounting Office Report FGMSD-80-4 (1979), apud *Software Engineering Notes*, *10*(5):6, out. 1985.

■ "USAF software contractors score poorly". *Aviation Week and Space Technology*, 14 nov. 1988, p. 103, apud *Software Engineering Notes*, 14(1):11, jan. 1989.

■ WATTS, Susan. "*Software* row dogs nuclear power plans" e "CEGB rebuffs critics of safety software". *New Scientist*, 1 abr. 1989 e 5 ago. 1989.

■ WINNER, Langdon. *Autonomous Technology*. Cambridge, MA, MIT Press, 1977.

■ WOOLNOUGH, Roger. "Britain scrutinizes *software* quality". *Electronics Engineering Times*, 13 jun. 1988, p. 19. ❑