OPINIÃO

«GUERRA DAS ESTRELAS»: INFORMÁTICA INTELIGENTE OU INTELIGÊNCIA ARTIFICIOSA?

LUÍS MONIZ PEREIRA
Departamento de Informática
Universidade Nova de Lisboa

RESUMO

Argumenta-se que o projecto de «Iniciativa de Defesa Estratégica» (IDE), ou «Guerra das Estrelas», é quimérico do ponto de vista informático, segundo múltiplas facetas, e por isso pouco inteligente. Em particular, pressupõe capacidades autónomas de Inteligência Artificial irrealistas. Por consequência o projecto não é realizável, à luz da experiência actual e dos desenvolvimentos informáticos previsíveis.
Conclui-se que o projecto é uma forma artificiosa de servir uma política de armamento com outros fins, incluindo o financiamento encapotado de empresas de armamento e outras pelos Estados, num pretenso mercado livre.
Rejeita-se a participação de Portugal e seus cientistas no projecto, com argumentos adicionais.

INTRODUÇÃO

No que segue admitimos o leitor conhecedor das linhas gerais do projecto de IDE, dos problemas de armamento nuclear em geral, e dos colocados pelo projecto. De qualquer modo, as referências indicadas no fim fornecem o contexto necessário, geral e específico, remetendo ainda para outras quanto a tópicos particulares.

Para completude, resumimos aqui as quatro fases de actuação da IDE, organizadas em etapas sucessivas: **lançamento** para o espaço, **libertação** dos veículos de reentrada (com ogivas múltiplas) e das falsas imitações, **percurso** no espaço, **término** com a reentrada dos veículos com ogivas múltiplas e sua detonação.

Na fase de lançamento, os foguetões aceleram a sua carga através e para fora da atmosfera, durante cerca de 3 minutos. Na fase de libertação, cada foguetão origina 10 veículos de reentrada apontados a diferentes alvos com ogivas múltiplas juntamente com 100 falsas imitações, durante aproximadamente 8 minutos. Na fase de percurso, que dura cerca de 15 minutos, os veículos de reentrada viajam acima da atmosfera em direcção aos respectivos alvos. Na fase terminal, nos 3 minutos seguintes, os veículos de reentrada penetram na atmosfera afectados pelo atrito desta e soltam as suas ogivas independentes.

IMPORTÂNCIA DO SOFTWARE PARA A IDE

A Organização da IDE reconhece que a tecnologia de computadores que controlaria individualmente cada arma, e coordenaria a sua operação conjunta, é substancialmente importante para o desenvolvimento das tecnologias críticas de intercepção. O sistema de comando e controlo para uma defesa abrangente contra mísseis balísticos tem que ser infalivelmente capaz de receber informação, e actuar com base nela, proveniente de milhares de lançamentos de mísseis, de dezenas de milhar de bombas neles contidas, e de centenas de milhar de falsas imitações e escombros. E tem de fazer tudo isso na

meia hora que leva a um míssil balístico intercontinental (MBIC) para viajar do local de lançamento na União Soviética até ao alvo nos EUA. Como o sistema tem que ser altamente automatizado, não há na prática ocasião para uma intervenção humana correctora de falhas inesperadas.

Um projecto destes requer um esforço enorme de software. Quais os obstáculos ao desenvolvimento desse software? Podem ser removidos?

ESTRUTURA DA ADMINISTRAÇÃO DA BATALHA

Embora os peritos ainda estejam indecisos quanto à estrutura da administração da batalha, hipoteciza-se que ela incluirá computadores e software locais, responsáveis pela administração da batalha em cada uma das fases previstas na IDE (lançamento para o espaço, libertação dos veículos de reentrada e falsas imitações, percurso no espaço, término com a reentrada dos veículos de ogivas múltiplas e sua detonação). Cada fase estará ligada às restantes através de um sistema de administração global da batalha.

O software que gere a batalha em cada fase defensiva tem que controlar os sensores e as armas destruidoras locais. Tais sensores devem localizar e seguir os alvos potenciais e distingui-los das falsas imitações. Esta parte do software criaria um ficheiro de seguimento contendo toda a informação sobre cada alvo. O software poderia então atribuir recursos defensivos na fase em questão, concertando a informação do ficheiro de seguimento com as armas disponíveis e com as regras programadas de intervenção, que especificam em que circunstâncias que alvos devem ser atacados. O sistema global de administração da batalha avaliaria a extensão e natureza do ataque em curso, especificando as regras de intervenção programadas para cada fase. Com vista a preparar os sistemas locais de administração de batalha a intervir sobre ogivas que escaparam às fases anteriores, o sistema global poderá comunicar-lhes a informação de seguimento e sensorial obtida nas fases anteriores.

Um sistema de defesa para mísseis balísticos depende pois fortemente do software que o controla; software defeituoso pode conduzir ao fracasso. Portanto o desenvolvimento de software fidedigno é um factor crítico no atingir dos objectivos propostos.

ETAPAS DO DESENVOLVIMENTO DESTE SOFTWARE E SEUS PROBLEMAS

O desenvolvimento de software é um processo intelectual divisível em etapas conceptuais que incluem o planeamento, o desenho, a implementação, e o teste e a depuração de erros. Na prática, estas etapas não são meramente sequenciais, podendo umas levar à reformulação de outras, incluindo as iniciais.

Planeamento

No planeamento, o desenvolvedor precisa determinar que funções o software deve desempenhar e antever as diferentes situações pertinentes ao seu uso. Esta especificação das instruções ou acções a tomar em cada instante e situação, torna-se mais difícil à medida que as tarefas a desempenhar crescem em complexidade. A especificação rigorosa do que um sistema de defesa para mísseis balísticos deve fazer é extremamente complexa. Por exemplo, a regra «basta abater todos os mísseis soviéticos» parece simples mas obriga a saber distinguir os mísseis soviéticos dos restantes. E que fazer se o míssil soviético se dirige à Alemanha de Leste? E se for necessário abater um míssil americano mal dirigido ou lançado enquanto se negociou uma trégua?

Estas questões tocam apenas a superfície de um problema fundamental do desenvolvimento de software, nomeadamente que é muitas vezes difícil decidir da inclusão ou não de certas características particulares num programa. Além disso, os seus desenvolvedores têm que prever com precisão todas as contingências possíveis e decidir como o software deverá responder a cada uma delas. Por exemplo, que procedimento adoptar se os computadores de uma das fases defensivas falharem? Como diferenciar um vaivém soviético de um MBIC?

Devido ao número de eventualidades não ser bem delimitado, os desenvolvedores não poderão prever todas as circunstâncias, e mesmo assim a quantidade delas que podem ser antecipadas requer plausivelmente milhares de páginas de especificações. A possibilidade de todos os potenciais cenários relevantes serem adequadamente contemplados é diminuta. Aliás, existem múltiplos exemplos de fracasso de software em situações bem menos complexas embora também cruciais, incluso em cenário de guerra (os mísseis Exocet argentinos atingiram os barcos britânicos por estes ignorarem os seus sinais de orientação; os Exocet fazem parte dos arsenal britânico e foram considerados «dos amigos»).

Desenho

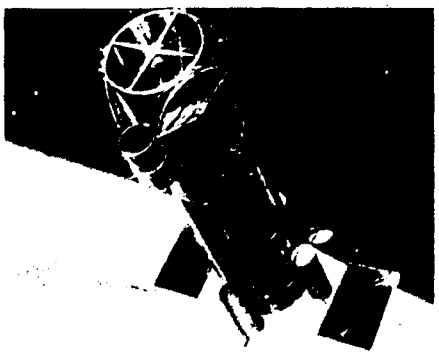
A fase de desenho do software contém também ela muitos problemas potenciais. Nesta fase os desenvolvedores perguntam-se como é que as especificações emergentes da fase de planeamento podem ser implementadas em computadores. Os erros da fase de deseho, se não são detectados durante ela, podem ser muito caros de corrigir. Como exemplo de erro temos a Gemini V a falhar o seu ponto de aterragem em 170 km por a rotação da terra em volta do sol ter sido ignorada no seu programa de manobragem. Muitos outros erros de desenho de software existem, incluindo em casos em que a segurança é crucial, como nas centrais nucleares. A lição é que nem a natureza nem a frequência dos erros de planeamento e desenho é previsível à partida. Assim, a detecção e depuração de erros é uma fase essencial de qualquer projecto de desenvolvimento de software. Como é que os erros se encontram e se avalia a confiança na operacionalidade dum sistema?

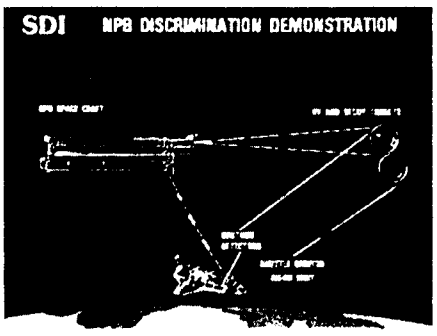
Validação: testes empíricos e simulação

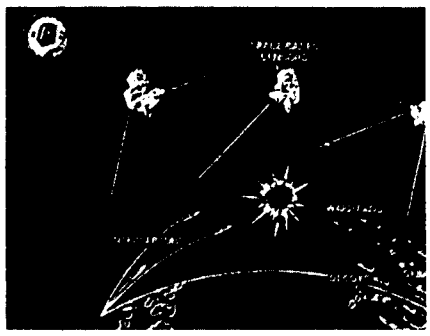
Existem duas técnicas efectivas para avaliar a confiança num sistema. Uma é analítica e requer uma **demonstração** matemática de que o sistema se comporta de acordo com as especificações. É claro que a demonstração não garante a justeza das especificações para a tarefa a desempenhar. Por outro lado as demonstrações deste tipo são pelo menos tão extensas quanto o programa a demonstrar como correcto, e a sua análise pode revelar-se mais difícil que a análise do próprio programa. Também tais técnicas aplicam-se apenas a programas relativamente pequenos, e não garantem quanto tempo um programa demonstrado correcto levará para executar os cálculos previstos (o que é fundamental saber para as aplicações a serem executadas em «tempo real», i.e. em cima do acontecimento). Finalmente, não prevêem o que acontecerá se o programa receber e processar dados não antecipados.

Uma técnica mais importante de validação de software é o **teste empírico.** Claro que uma defesa para mísseis ba-

lísticos não pode ser sujeita a testes empíricos de grande escala em condições realísticas. Portanto os desenvolvedores do sistema terão que sujeitar-se a formas mais limitadas de teste empírico: testes de pequena escala e **testes de simulação,** nos quais é usado um computador para simular acontecimentos reais de grande escala.







Testes empíricos

Embora os testes possam induzir confiança, não podem garantir o sucesso do software em condições reais de funcionamento. Aliás, os problemas resultantes da integração de vários componentes não podem ser detectados aquando do seu teste individual, e aparecem frequentemente só quando o sistema é levado aos seus limites, o que não pode ser feito nos testes de pequena escala. Existem abundantes exemplos documentados de situações de teste, em aplicações militares, que ilustram bem a fragilidade dos testes supletivos (cf. ibidem).

Testes de simulação

São conhecidas as limitações dos testes de simulação que quantas vezes simplificam excessivamente a realidade simulada para se poderem tornar simplesmente exequíveis. E quanto mais complexa essa realidade maiores as simplificações impostas (cf. DISCOVER, Setembro 1986).

Aplicada à defesa para mísseis balísticos, as técnicas de simulação vêem-se restringidas de várias maneiras. Primeiro, os simuladores poderão ser incapazes de reproduzir as características típicas (assinaturas) dos fenómenos físicos associados aos vários acontecimentos reais simultâneos eventualmente presentes nos cenários imaginados. A simultâneadade complica a detecção das assinaturas individuais e a simulação; assim, apenas um certo número de complexos de acontecimentos é considerado, através do seu processamento prévio, eliminando surpresas quanto à reacção do sistema a situações não antecipadas pelos seus criadores, e diminuindo portanto o realismo da simulação.

Segundo, embora os avanços na tecnologia de computadores permitam, pela maior velocidade obtida, simular situações mais realistas, isso de nada serve quando falta o conhecimento teórico e empírico acerca dos próprios fenómenos físicos.

Terceiro, os simuladores não podem prever satisfatoriamente situações dependentes das decisões de um oponente determinado e inteligente, o qual é quem escolhe em última análise os parâmetros determinantes da situação, havendo sempre um elemento imprevisível em relação às tácticas e em relação às armas a que o inimigo recorrerá.

Melhoria e manutenção do software

Apesar de não puderem melhorá-lo a partir da sua utilização real, como é usual em informática, os desenvolvedores do software de defesa para mísseis balísticos tentarão fazê-lo com base nos testes supletivos referidos acima, expandindo inclusive o hardware e as capacidades do software em responder à necessidades novas entrevistas. Estes esforços constituem a chamada manutenção do software, que pode rondar os 70% dos custos totais durante o ciclo completo de vigência do software desde o início do seu desenvolvimento.

Cinco questões maiores se põem a este respeito ao software da IDE:

- Primeiro, os erros significativos descobertos, depois de o sistema ter sido posto a uso operacional, têm de ser corrigidos. Ora a depuração de erros relativos a software de «tempo real» é particularmente difícil, até porque não basta usar equipamento semelhante para o fazer, pois pequenas diferenças físicas em equipamento «igual» podem ser responsáveis por erros em «tempo real» só detectados num dos equipamentos.
- Um segundo problema bem conhecido neste contexto é o de que é frequentemente difícil obter que este tipo de erros se repitam, com vista à sua detecção, pois dependem dos tempos de chegada relativos dos sinais de entrada. Assim, os erros de software nos sistemas de «tempo real» grandes só se encontram detectados e corrigidos num sentido probabilístico.
- Terceiro, mesmo que um erro seja localizado a sua correcção pode introduzir outros. A probabilidade de introduzir um novo erro ao corrigir um anterior varia entre os 10 e os 50%. Além disso, a maioria dos erros de desenho de software detectados depois do sistema estar operacional, incluindo os induzidos pela correcção de outros, só o são após extensa utilização operacional. A experiência com grandes programas de controlo (entre 100 mil e 2 milhões de linhas de código) aponta que a possibilidade de introduzir um erro grave durante a correcção de outro é tão grande que só se deve corrigir uma pequena fracção dos erros originais.

Por exemplo, o vaivém espacial americano, cujo software de «tempo real» tem cerca de 500 mil linhas de código, falhou a descolagem por um problema de sincronização entre computadores de bordo, em resultado de um erro introduzido quando outro erro havia sido corrigido dois anos atrás. O erro introduzido revelar-se-ia em média em uma em cada 67 descolagens.

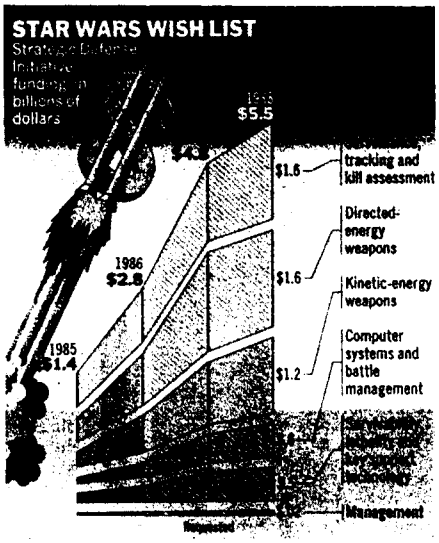
- Quarto, existe um problema de dimensão sem precedentes quanto à administração do desenvolvimento dos produtos software da IDE. Uma estimativa encomendada oficialmente refere um mínimo, no caso optimista, de 10 milhões de linhas de código de programa para todo o sistema (cf. com as 500 mil linhas do sistema de operação de um computador de grande porte). Se esta estimativa do caso optimista estiver errada por um factor de dois, o desenvolvimento do software do projecto implicará mais de 30 mil homens-ano, ou seja pelo menos 3 mil programadores e analistas de sistema trabalhando durante 10 anos. Por esta razão prevê-se uma rotação de pessoal que reduzirá a memória institucional do projecto, aumentando a possibilidade de que detalhes essenciais sejam passados em claro nas transições de pessoal. Aumenta ainda a possibilidade de infiltração de agentes com vista à sabotagem deliberada, pela introdução de erros no sistema difíceis de detectar nos testes e que só se tornariam manifestos durante a sua utilização operacional, em particular fazendo-se aparecer, por programação, só ao fim de um certo tempo. O apertar da segurança restringindo a comunicação entre os informáticos da equipa às necessidades justificadas teria a repercussão contraproducente de aumentar as possibilidades de aparecimento de erros graves involuntários.
- Quinto, as mudanças de táctica e a introdução de novos tipos de armamento por parte da União Soviética obrigará a complexas reformulações do software do sistema operacional já instalado em terra e no espaço.

INTELIGÊNCIA ARTIFICAL OU ESPERTEZA SALOIA?

Tem sido sugerido que o desenvolvimento de software para sistemas muito complexos pode ser grandemente facilitado pelo uso da Inteligência Artificial. Especificamente, menciona-se a programação automática, o diagnóstico automático de erros em programas, e os sistemas periciais («expert systems»).

Quanto às aplicações na IDE teriam relevância os sistemas periciais, bem como o reconhecimento de formas, a geração de planos, e o diagnóstico automático de avarias em artefactos. A robótica poderia introduzir capacidades de auto-reparação e manutenção.

De uma forma geral, podemos dizer que quaisquer destas técnicas se encontra ainda num estado muito experimental, com provas dadas sobretudo em pequenas aplicações de laboratório, ou reais mas cuidadosa e especialmente escolhidas para demonstração de potencialidades de princípio. Em particular, o uso da Inteligência Artificial no desenvolvimento de software mal começou.



Por outro lado, passa-se agora com a Inteligência Artificial em segundo grau aquilo que desde há anos se passa com a Informática. Quando um problema se não sabe resolver humanamente, pois há que informatizá-lo e metê-lo no computador. Desses problemas muitos há que o informático não sabe como programar. Então remete-os para a Inteligência Artificial que com certeza os há-de resolver (por via não se sabe de que homúnculo incorporado sob a forma de programa, que os resolve por sua vez remetendo-os para um homúnculo inteligente dentro de si...).

Interessa ainda referir que o funcionamento conjunto e integrado das várias técnicas de Inteligência Artificial separadamente desenvolvidas e testadas está por fazer, colocando problemas complexos cuja solução não está no horizonte.

Consideremos em mais detalhe, das várias contribuições mencionadas, aquelas que influiriam mais marcadamente o desenvolvimento genérico de software:

Programação automática

A ideia de utilizar programas para escrever outros não resolve grande parte dos problemas apontados. A função principal da programação automática é a de aliviar as dificuldades técnicas em implementar as especificações da fase de desenho e em modificar código já existente. Não é a de gerar programas *ex nihilo*. Cerca de metade dos erros de software, contudo, advêm de escolhas humanas incorrectas durante as fases de planeamento e desenho. Estas tarefas humanas não podem ser relegadas para a programação automática, que de resto se encontra num estado incipiente (ao nível das centenas de linhas de código gerado). Por outro lado, os geradores automáticos de programas são eles próprios programas susceptíveis de erro (embora possam sem dúvida aliviar a tarefa da programação).

Diagnóstico automático de erros

Trata-se de uma técnica muito recente, que é na verdade semiautomática, funcionando como um coadjuvante do programador para facilitar a identificação de erros em programas, deixando àquele a tarefa de os corrigir. Introduzirá, quando evoluir, um factor de eficiência na correcção de programas, mas não substitui o programador na tarefa de depuração nem evitará que ele substitua um erro por outro.

Sistemas periciais

A dependência em técnicas informais de raciocínio nos sistemas periciais abre a possibilidade à existência de contradições internas profundamente imbuídas no sistema que causem o seu fracasso. Até à data, a investigação em sistemas periciais tem-se concentrado em áreas bem definidas, nas quais existe um conhecimento humano detalhado ds regras de raciocínio aplicáveis, resultante da experiência. Apesar disso a tecnologia dos sistemas periciais é ainda limitada na sua aplicação a esses campos.

Os que recomendam a sua aplicação à administração de batalha na IDE ignoram a ausência de expertise humana prévia em tal situação. Por outro lado, muito nessas situações poderá nem sequer ter sido antecipado, e portanto insusceptível de colocar em regras periciais.

Estas situações referem-se tanto às estratégias de batalha a seguir, como por exemplo ao efeito dos lasers químicos e de partículas neutras sobre armas contra mísseis balísticos (ainda não construídas nem testadas), como ainda à necessidade do sistema pericial necessitar de lidar

com informação altamente distribuída e ser ele próprio distribuído por vários processadores.

Além disso, o sistema pericial teria que possuir capacidades evolutivas importantes para contemplar novas situações, tornadas agora possíveis ou entretanto detectadas como tal.

Finalmente, apesar da sua complexidade teria que ser extremamente rápido, fiável e sensato.

ESPERTEZA SALOIA

Por fim, é muito mais fácil usar meios convencionais, e até a Inteligência Artificial, para destruir os componentes espaciais da IDE, do que esta destruir os mísseis balísticos, até porque aqueles não são posicionados de uma só vez.

IDE, CIÊNCIA NACIONAL, E DEFESA NACIONAL

A Ciência nacional não se pode dar ao luxo de participar em projectos de I&D como a «Guerra das Estrelas», tecnicamente quiméricos e perigosos. Tais projectos diminuirão a segurança nacional em vez de a aumentarem.

Aliás, o até recentemente responsável científico do projecto, David Parnas, reconheceu a sua inviabilidade científica e perigosidade, demitiu-se, e faz agora campanha contra ele, em conjunto com a maioria da opinião universitária americana. Esperemos que a opinião científica portuguesa não lhes fique atrás, recusando-se a participar no projecto, o que eu próprio já fiz.

A IDE é, de facto, uma forma encapotada de subsidiar empresas, pelo Estado, num pretenso mercado lire, o qual serve de argumento à diminuição do orçamento social.

Os verdadeiros objectivos da IDE necessitariam contudo de todo um outro documento de análise. No entanto, parece-me poder avançar sem grande erro que visa «spin-offs» comerciais e militares, muito antes da sua realização total, aliás tão problemática.

Os objectivos militares seriam novas tecnologias para teatros de operações militares de pequena escala, podendo incluir o uso de armas nucleares, Aí, os porgressos da Informática, e da Inteligência Artificial e Robótica em particular, podem conduzir à robotização e gestão informatizada inteligente dessas operações.

Outra aplicação militar será a colocação no espaço de lasers capazes de iniciarem milhares de incêndios à superfície da Terra, por meio de raios de luz com 20 a 30 megawatts. Estes lasers podem ser dirigidos especificamente a silos de mísseis, fixos ou móveis, para os inutilizar.

É fácil perceber o receio soviético quanto à IDE, mesmo sendo esta tecnicamente inviável como é apresentada. A possibilidade de uma maior defesa (admitindo a sua viabilidade parcial em proteger apenas as instalações militares) e o seu potencial ofensivo, leva ao desenvolvimento de novas armas atacantes e defensivas, continuando a espiral armamentista, tal como o tem demonstrado a experiência de sucessivas inovações neste campo, em que cada arma leva a sua contra-arma. Isto numa altura em que a economia soviética pretende poupar-se aos gastos militares, não podendo além disso competir com a maior capacidade americana em aproveitar «spin-offs» comerciais civis.

Naturalmente, quaisquer dos novos produtos, militares ou comerciais, serão vendidos em quantidade lucrativa a outros Estados, incluindo aqueles que participaram na sua investigação mas que foram mantidos afastados da fase de desenvolvimento e fabrico.

Interessa reter que a C&T nacionais não são para uso de empresas estrangeiras que querem obter o nosso saber especializado, para nos venderem afinal os produtos que dele fazem uso sem incorporação de mão-de-obra nacional. Temos que evitar ainda servirmos para trabalhos menores dessa C&T, sem acesso aos resultados e patentes economicamente úteis, isto é, evitar a exploração mental. Ora é justamente isso que a IDE também pretende.

A Defesa Nacional exige que tal não aconteça, por muito que isso doa a alguns militares interessados em brinquedos caros e rapidamente obsoletos cujo objectivo é estragarem-se reciprocamente. Tais militares pretendem apenas o seu maior prestígio e a justificação da sua existência, bem como o aumento de influências junto das empresas nacionais (militares ou não) que fazem da guerra uma indústria.

Quem não se lembra dos sucessivos escândalos, veiculados pela imprensa, sobre os negócios de armas que se baseiam em, ou usam, Portugal para vender o que quer que seja muito simplesmente a quem quer que pague o preço correcto? Certo, a Defesa Nacional tem necessidades de C&T, mas necessidades nacionais, que beneficiem a indústria nacional. Não são preciso fragatas sofisticadas (como muitos militares reconhecem), e as usuais ajudas (=empréstimos), para se efectivamente proteger a Zona Económica Exclusiva dos nossos mares e realizar as necessárias pesquisas oceanográficas sobre os recursos marítimos.

Por outro lado, não colhe o argumento de que os projectos de Defesa Nacional são necessários para permitir o desenvolvimento. Como se sabe, o Japão está constitucionalmente impedido de desenvolver uma indústria militar desde 1945, e não consta que lhe falte desenvolvimento. Pelo contrário, esse desenvolvimento foi em grande parte permitido pelos menores gastos militares. A C&T não precisa de motivações militares para se desenvolver, como infelizmente tem acontecido ao longo da História (cf., mais recentemente, o aparecimento do computador e da energia atómica); os fins sociais e económicos não-militares bastam amplamente para justificar a C&T numa civilização eticamente madura.

Mas há outros riscos a acautelar. Se nos EUA existem alternativas ao financiamento militar da C&T e se por outro lado o Departamento da Defesa financia investigação em quase tudo, em Portugal corre-se o risco de se criar uma dependência mais forte dos objectivos militares mais estritos, até por inexistência de financiamento alternativo suficiente, uma vez criada a primeira ligação. Mais vale não morder o isco, nem criar tal via militar de financiamento.

Finalmente, a Defesa Nacional fazem-na os cientistas todos os dias defendendo a C&T contra a apatia do Estado.

REFERÊNCIAS UTILIZADAS

- «Star Wars at the crossroads»
tema da TIME, Junho 23, 1986, págs. 8-14.
- «Will star wars work? It isn't a question of technology?»
Thomas Powers, Discover, Dezembro 1986.
- «How much bang for the buck?»
Wayne Biddle, DISCOVER, Setembro 1986, págs. 50-63.
- «Technology for peace»
tema da SCIENCE 85, Dezembro 1985, págs. 26-51.
- «Software aspects of strategic defence systems»
David Parnas, American Scientist, vol. 73, n.º 5, Setembro-Outubro 1985.
- «The development of software for ballistic-missile defense»
Herbert Lin, Scientific American, Dezembro 1985, págs. 32-39.
- «Expert systems in the SDI environment»
Yi-Tzuu Chien & Jay Liebowitz, Computer, Julho 1986, págs. 115-122.
- «Necessidades de C&T e Informática para o desenvolvimento»
Luíz Moniz Pereira, Workshop da ACTD, Aveiro 1986.

■