Artigo seleccionado

Processos sociais e demonstrações de teoremas e programas

Richard de Millo, Richard Lipton, Alan Prelis

> *Gostaria de colocar a mesma questão que Descartes colocou. Você está a propor dar uma definição precisa de correcção lógica que deverá ser o mesmo que o meu sentimento intuitivo vago de correcção lógica. Como se propõe mostrar que são idênticos?*
>
> *... O matemático médio não deve esquecer que a intuição é a autoridade final.*
>
> *J. Barkley Rosser*

Muitas pessoas argumentaram que a programação de computadores deverá tornar-se mais como a Matemática. Talvez assim seja, mas não no modo como aparentemente o imaginavam. O objectivo da verificação de programas, uma tentativa para tornar a programação mais parecida com a Matemática, é aumentar dramaticamente a nossa confiança no funcionamento correcto de um pedaço de software, e o mecanismo que os verificadores utilizam para conseguir este objectivo é uma longa cadeia de lógica dedutiva formal. Em Matemática, o objectivo é aumentar a nossa confiança na correcção de um teorema, e é verdade que um dos mecanismos que os matemáticos *poderiam* em teoria usar para atingir este objectivo é uma longa cadeia de lógica formal. Mas de facto não o fazem. O que utilizam é uma demonstração, um animal muito diferente. Nem sequer a demonstração resolve a discussão; contrariamente ao que o nome sugere, uma demonstração é apenas um passo na direcção da confiança. Acreditamos que, no final, é um processo social que determina se os matemáticos se sentem confiantes em relação a um teorema — e acreditamos que, como não pode existir um processo social comparável entre os

verificadores de programas, a verificação de programas está determinada para falhar. Não conseguimos ver como vai ser capaz de afectar a confiança de alguém nos programas.

As pessoas de fora vêm a Matemática como um processo frio, formal, lógico, mecânico, monolítico de pura intelectualidade; propomos que na medida em que é bem sucedida, a Matemática é um processo social, informal, intuitivo, orgânico, humano, um projecto comunitário. No interior da comunidade matemática, a visão da Matemática como lógica e formal foi elaborada por Bertrand Russell e David Hilbert nos primeiros anos deste século. Eles viam a Matemática como desenvolvendo-se em princípio de axiomas ou hipóteses para teoremas por passos, cada passo facilmente justificável a partir dos seus antecedentes através de uma regra de transformação estrita, sendo as regras de transformação poucas e estando fixadas. Os *Principia Mathematica* foi o empreendimento culminante dos formalistas. Foi também o golpe de morte na perspectiva formalista. Não existe aqui nenhuma contradição: Russell conseguiu realmente mostrar que demonstrações ordinárias que funcionam podem ser reduzidas a deduções formais, simbólicas. Mas não conseguiu, em três volumes enormes e pesados, ir para além de factos elementares de aritmética. Mostrou o que pode ser feito em princípio e o que não pode ser feito na prática. Se o processo matemático fosse realmente uma progressão lógica estrita, ainda estaríamos a contar pelos dedos.

Acreditar nos teoremas e nas demonstrações

> *De facto todos os matemáticos sabem que uma demonstração não foi "compreendida" se não se fez mais do que verificar passo a passo a correcção das deduções de que é composta e se não se tentou encontrar uma compreensão clara das ideias que levaram à construção desta cadeia particular de deduções preferida a todas as outras.*
>
> *N. Bourbaki*
>
> *Concorda comigo se pareço falar verdade*
>
> *Sócrates*

Stanislaw Ulam estima que os matemáticos publicam 200.000 teoremas por ano (1976). Alguns destes são subsequentemente contraditados ou rejeitados, outros são postos em dúvida e a maioria são ignorados. Apenas uma pequena parte acaba por ser compreendida e acreditada por um qualquer grupo razoável de matemáticos.

Os teoremas ignorados ou desacreditados raramente são produto do trabalho de excêntricos ou de incompetentes. Em 1879, Kempe (1879) publicou uma demonstração da conjectura das quatro cores que se manteve durante onze anos antes de Heawood (1890) descobrir uma falha fatal no raciocínio. A primeira colaboração entre Hardy e Littlewood resultou numa comunicação apresentada no encontro de Junho de 1911 da London Mathematical Society; a comunicação nunca foi publicada porque eles subsequentemente descobriram que a sua demonstração estava errada (Bateman e Diamond, 1978). Cauchy, Lamé e Kummer pensaram num ou noutro momento que tinham provado o último teorema de Fermat (Davis, 1972). Em 1945, Rademacher pensou que tinha resolvido a Hipótese de Riemann; os seus resultados não só circularam entre o mundo matemático como foram anunciados no *Time* (Davis, 1972).

Recentemente encontrámos o seguinte grupo de notas de rodapé acrescentadas a um breve esboço histórico de alguns resultados de independência em teoria de conjuntos (Jeck, 1973):

> "(1) O resultado do Problema 11 contradiz os resultados anunciados por Levy [1963b]. Infelizmente a construção aí apresentada não pode ser completada.
> (2) A transferência para *ZF* foi também reivindicada por Marek [1966] mas o método esboçado não parece ser satisfatório e não foi publicado.
> (3) Um resultado contraditório foi anunciado e depois retirado por Truss [1970].
> (4) O exemplo no Problema 22 é um contra-exemplo para uma outra condição de Mostowski, que conjecturou a sua suficiência e destacou este exemplo como um caso para testagem.
> (5) O resultado de independência contradiz a afirmação de Felgner [1969] que o Princípio da Cofinalidade[1] implica o Axioma da Escolha. Um erro foi encontrado por Morris (ver as correcções de Felgner em [1969])" (p. 110).

O autor quer desenterrar nenhum machado de guerra, provavelmente nunca tomou conhecimento da presente controvérsia em programação e não é claramente sua intenção troçar dos seus colegas. Não existe modo de descrever a história das ideias matemáticas sem descrever os sucessivos processos sociais que funcionam nas demonstrações. A questão não é a de que os matemáticos cometem erros; isso não seria preciso dizer. A questão é que os erros dos matemáticos são corrigidos, não através da lógica simbólica formal, mas por outros matemáticos.

Aumentando apenas o número de matemáticos trabalhando num dado problema não assegura necessariamente demonstrações convincentes. Recentemente, dois grupos independentes de topologistas, um americano e outro japonês, independentemente anunciaram resultados relativos ao mesmo tipo de objecto topológico, uma coisa chamada um grupo de homotopia. Verificou-se que os resultados eram

contraditórios, e, uma vez que ambas as demonstrações envolviam cálculos simbólicos e numéricos complexos, não era de todo evidente quem se tinha enganado. Mas o que estava em jogo era suficientemente alto para justificar aprofundar o assunto, e por isso, as demonstrações japonesa e americana foram permutadas. Mas nem a demonstração japonesa nem a americana puderam ser questionadas. Subsequentemente, um terceiro grupo de investigadores obteve uma outra demonstração, deste vez apoiando o resultado americano. Estando agora o peso da evidência contra a sua demonstração, os japoneses retiraram-se para aprofundar o assunto.

Existem realmente duas morais nesta história. Em primeiro lugar, uma demonstração, por si só, não aumenta significantemente a nossa confiança na verdade provável do teorema que pretende provar. De facto, no caso do teorema sobre o grupo de homotopia, todas as demonstrações propostas eram suficientemente horríveis para sugerir que o próprio teorema deveria ser repensado. Em segundo lugar, as demonstrações consistindo inteiramente de cálculos não são necessariamente correctas.

Mesmo a simplicidade, a clareza e a facilidade não garantem que uma demonstração esteja correcta. A história das tentativas para demonstrar o Postulado das Paralelas é uma fonte particularmente rica de demonstrações encantadoras e simples que se revelam falsas. Desde Ptolomeu a Legendre (que tentou várias vezes), os maiores geómetras de cada época esforçaram continuadamente as suas cabeças contra o quinto postulado de Euclides. E, o que é pior, apesar de sabermos hoje que o postulado é indemonstrável, muitas das demonstrações imperfeitas eram tão enganadoras que no excelente comentário sobre Euclides escrito por Heath (1956, pp. 204-219), este não permite que elas valham por si próprias; Heath marca-as com itálicos, notas de rodapé, e reparos explicativos, não fosse algum jovem matemático, tropeçando no volume, ser enganado.

A ideia de que uma demonstração pode, no máximo, exprimir a verdade apenas provavelmente estabelece uma associação interessante com uma controvérsia matemática recente. Num número recente da revista *Science* , Gina Bari Kolata (1976) sugeriu que a noção aparentemente segura de demonstração matemática deva ser revista. Aqui a questão central não é "Como os teoremas se tornam acreditados?" mas "Em que acreditamos quando acreditamos num teorema?" Há duas perspectivas relevantes, que podem ser *grosso modo* classificadas em clássica e probabilista.

Os clássicos dizem que quando se acredita numa proposição matemática *A*, se acredita que *em princípio* existe uma dedução correcta, formal, válida, passo a passo, verificável sintacticamente conduzindo a *A* através de um cálculo lógico adequado tal como a teoria de conjuntos de Zermelo-Fraenkel ou a aritmética de Peano, uma

dedução de *A* segundo o estilo dos *Principia*, a dedução que formaliza completamente a verdade de *A* em binário, a noção aristotélica de verdade: "Uma proposição é verdadeira se do que é, diz que é, e do que não é diz que não é". Esta cadeia formal de raciocínios não significa, de modo algum, a mesma coisa que uma demonstração matemática comum, ordinária. A visão clássica não requer que uma demonstração ordinária seja acompanhada pela sua contrapartida formal; pelo contrário, existem razões matemáticas poderosas para deixar que os deuses formalizem os nossos argumentos. Um teórico estima, por exemplo, que uma demonstração formal de uma das conjecturas de Ramanujan supondo a teoria de conjuntos e a análise elementar ocuparia cerca de duas mil páginas; a extensão de uma dedução a partir de princípios primeiros é quase inconcebível (Manin, 1977). Mas o clássico acredita que a formalização é em princípio uma possibilidade e que a verdade que exprime é binária, ou sim ou não.

Os probabilistas argumentam que uma vez que qualquer demonstração muito extensa pode, na melhor das hipóteses, ser vista como apenas provavelmente correcta, porque não estabelecer teoremas probabilisticamente e dar demonstrações probabilísticas? A demonstração probabilística pode ter a vantagem dupla de ser tecnicamente mais fácil que a clássica, que é bivalente, e pode permitir aos matemáticos isolar as ideias críticas que dão origem à incerteza em demonstrações tradicionais, binárias. Uma ilustração da abordagem probabilista é o algoritmo de Michael Rabin para testar primaliedade provável (1976). Para inteiros muito grandes *N*, todas as técnicas clássicas para determinar se *N* é composto passam a ser impossíveis de trabalhar. Usando mesmo a programação mais inteligente, os cálculos necessários para determinar se números maiores que 10^{104} são primos requer quantidades imensas de tempo computacional. A ideia interessante de Rabbin foi que, se se está disposto a aceitar uma muito boa probabilidade de que *N* é primo (ou não primo), então esta poder-se-á obter num período de tempo razoável — e com uma probabilidade de erro cada vez mais pequena.

Dadas estas incertezas sobre o que constitui uma demonstração aceitável, que é, apesar de tudo, um elemento razoavelmente básico do processo matemático, como pode então ser que a Matemática tenha sobrevivido e seja tão bem sucedida? Se as demonstrações têm pouca semelhança com o pensamento dedutivo formal, se podem manter-se por gerações e depois cair, se podem conter falhas que desafiam a detecção, se podem exprimir apenas a probabilidade da verdade dentro de certos limites de erro — se não são de facto capazes de *demonstrar* teoremas, no sentido de garanti-los para além da probabilidade e, se necessário, para além da intuição, bem, então, como é que a Matemática funciona? Como consegue ser bem sucedida

no desenvolvimento de teoremas que são significantes e compelem a crença?

Em primeiro lugar, a demonstração de um teorema é uma mensagem. Uma demonstração não é um objecto abstracto belo com uma existência independente. Nenhum matemático compreende uma demonstração, reclina-se, e suspira alegremente ao saber que pode agora estar certo da verdade do seu teorema. Ele sai para o corredor, e procura alguém que o ouça. Irrompe pelo gabinete de um colega e pede o quadro. Esquece o tópico programado e oferece um seminário com a sua nova ideia. Arrasta os seus alunos de pós-graduação para longe das suas dissertações para que o ouçam. Põem-se ao telefone e informa os seus colegas no Texas e em Toronto. Na sua primeira incarnação, uma demonstração é uma mensagem falada, ou, no máximo, um esboço no quadro ou num guardanapo de papel.

Este estádio falado é o primeiro filtro de uma demonstração. Se não gera entusiasmo ou crença entre os seus amigos, o matemático prudente reconsidera-a. Mas se eles a acham moderadamente interessante e convincente, ele escreve-a. Depois de circular em esboço durante algum tempo, se parece plausível, o matemático faz uma versão polida e submete-a para publicação. Se os revisores também a acharem atraente e convincente, é publicada de modo a ser lida por uma audiência maior. Se um número suficiente de membros dessa audiência mais ampla a acreditarem e gostarem dela, então, depois de um período de arrefecimento adequado as publicações de revisão observam com um olhar mais sossegado para averiguar se é realmente tão agradável como parecia inicialmente e se, numa apreciação calma, realmente acreditam nela.

E o que acontece se uma demonstração é realmente acreditada? O processo mais imediato é provavelmente uma internalização do resultado. Isto é, o matemático que lê e acredita na demonstração tentará parafraseá-la, pô-la nos seus próprios termos, encaixá-la na sua visão pessoal própria do conhecimento matemático. Não é provável que dois matemáticos internalizem um conceito matemático exactamente da mesma maneira, e por isso este processo conduz usualmente a múltiplas versões do mesmo teorema, cada uma reforçando crenças, cada uma incrementando o sentimento da comunidade matemática que a afirmação original é provavelmente verdade. Gauss, por exemplo, obteve pelo menos meia dúzia de demonstrações independentes da sua "lei da reciprocidade quadrática"; até hoje mais de cinquenta demonstrações desta lei são conhecidas. Imre Lakatos dá, no seu *Proofs and refutations* (1976), discussões historicamente precisas das transformações que vários teoremas famosos sofreram desde a sua concepção original até à aceitação generalizada. Lakatos demonstra que a fórmula de Euler $V - E + F = 2$ foi reformulada por diversas vezes durante quase duzentos anos depois do seu estabelecimento

inicial, até que finalmente atingiu a sua forma actual estável. A transformação mais forte que pode ter lugar é a generalização. Se, pelo mesmo processo social que funciona com o teorema original, o teorema generalizado passa a ser acreditado, então a afirmação original ganha muito em plausibilidade.

Um teorema em que se acredita é usado. Pode aparecer como um lema em demonstrações maiores; se não conduz a contradições ficamos mais inclinados a acreditá-lo. Ou engenheiros podem usá-lo colocando nele valores físicos. Temos uma confiança bastante alta nas equações clássicas da fadiga porque vemos pontes que estão de pé; temos alguma confiança nos teoremas básicos da mecânica dos fluidos porque vemos aviões que voam.

Resultados credíveis por vezes estabelecem contacto com outras áreas de Matemática — aqueles que são importantes invariavelmente o fazem. A transferência bem sucedida de um teorema ou de uma técnica de demonstração de um ramo da Matemática para outro aumenta o nosso sentimento de confiança em relação a ele. Em 1964, por exemplo, Paul Cohen utilizou uma técnica chamada forçar[2] para demonstrar um teorema em teoria de conjuntos (Cohen, 1963, 1964); na altura, as suas noções eram tão radicais que a demonstração dificilmente foi compreendida. Mas, mais tarde, outros investigadores interpretaram a noção de forçar num contexto algébrico, associaram-na a ideias mais familiares em lógica, generalizaram os conceitos e acharam que estas generalizações eram úteis. Todas estas conexões (em conjunto com os outros processos sociais normais que conduzem à aceitação) tornaram a ideia de forçar muito mais atraente, e hoje, forçar é estudado normalmente por estudantes de pós-graduação em teoria de conjuntos.

Depois de internalização, transformação, generalização, utilização e conexões suficientes, a comunidade matemática eventualmente decide que os conceitos centrais no teorema original, agora talvez consideravelmente alterado, possuem uma estabilidade última. Se as diversas demonstrações parecem certas e se os resultados forem examinados de diferentes ângulos, então a verdade do teorema é eventualmente considerada como tendo sido estabelecida. O teorema é pensado como verdadeiro no sentido clássico — isto é, no sentido em que *poderia* ser demonstrado através da lógica dedutiva formal, apesar de, para quase todos os teoremas, tal dedução nunca ter tido, nem nunca terá, lugar.

O papel da simplicidade

Porque o que é claro e facilmente compreensível atrai; o complicado repele.

David Hilbert

Por vezes temos de dizer coisas difíceis, mas deveremos dizê-las de um modo tão simples como o soubermos fazer.

G. H. Hardy

Por norma, os problemas matemáticos mais importantes são enunciados de um modo simples e fácil. É muito mais provável que um teorema importante tome a forma A do que a forma B.

A: Todo o ——— é um ———.
B: Se ——— e ——— e ——— e ——— excepto para os casos especiais
a) ———
b) ———
c) ———,
então a não ser que
i) ——— ou
ii) ——— ou
iii) ———,
todo o ——— que satisfaz ——— é um ———.

Os problemas que mais fascinaram e atormentaram e deliciaram os matemáticos através dos séculos têm sido os mais simples de enunciar. Einstein afirmava que a maturidade de uma teoria científica podia ser julgada pela facilidade com que podia ser explicada ao homem comum. O problema das quatro cores repousa em fundações tão finas que pode ser enunciado com uma precisão completa a uma criança. Se a criança aprendeu a tabuada, pode compreender o problema da localização e distribuição dos números primos. E o fascínio profundo do problema de definir o conceito de "número" pode torná-la num matemático.

A correlação entre importância e simplicidade não é um acaso. Os teoremas simples e atractivos são os que mais provavelmente são ouvidos, lidos, internalizados e utilizados. Os matemáticos usam a simplicidade como um primeiro teste de uma demonstração. Apenas se à primeira vista parece interessante considerá-lo-ão em

detalhe. Os matemáticos não são masoquistas altruístas. Pelo contrário, a história da Matemática é uma longa busca da facilidade e do prazer e da elegância — no domínio dos símbolos, claro.

Mesmo que não o quisessem, os matemáticos teriam de usar o critério da simplicidade; trata-se de uma impossibilidade psicológica escolher de entre os 200 000 candidatos à nossa atenção outro que não seja mais simples e mais atractivo. Se existem conceitos importantes, fundamentais em Matemática que não são simples, os matemáticos provavelmente nunca os descobrirão.

Proposições matemáticas confusas e desagradáveis, que se aplicam apenas a classes de estruturas insignificantes, proposições idiossincráticas, proposições que necessitam de exorbitantes ferramentas matemáticas fora do comum, proposições que necessitam cinco quadros ou um rolo de papel para esboçar — estas dificilmente serão alguma vez assimiladas no corpo da Matemática. E, no entanto, é apenas através dessa assimilação que as demonstrações ganham credibilidade. A demonstração em si própria não é nada; apenas quando foi sujeita aos processos sociais da comunidade matemática se torna credível.

Neste texto tendemos a sublinhar a simplicidade acima de tudo porque é o primeiro filtro de qualquer demonstração. Mas não nos queremos caracterizar nem aos nossos colegas matemáticos como filistinos ou brutos. Depois de uma ideia ter satisfeito o critério da simplicidade, outros critérios ajudam a determinar o lugar de uma demonstração entre as ideias que fazem os matemáticos contemplar abstractamente a distância. Yuri Manin (1977) disse-o da melhor maneira: Uma boa demonstração é aquela que nos torna mais eruditos.

Descrendo das verificações

> *Pelo contrário, não encontro nada na logística para o descobridor senão cadeias. Não nos ajuda de modo algum na direcção da concisão, muito longe disso; e se é necessário vinte e sete equações para estabelecer que 1 é um número, quantas serão necessárias para demonstrar um teorema real?*
>
> *Henri Poincaré*

> *Um dos deveres primeiros do matemático é agir como um conselheiro dos cientistas... é desencorajá-los de esperar demasiado da Matemática.*
>
> *Norbert Weiner*

As demonstrações matemáticas aumentam a nossa confiança na verdade das

afirmações matemáticas apenas depois de terem sido sujeitos aos mecanismos sociais da comunidade matemática. Estes mesmos mecanismos condenam as chamadas demonstrações de software, as longas verificações formais que correspondem, não à demonstração matemática que funciona, mas à estrutura lógica imaginária que o matemático evoca para descrever o seu sentimento de crença. As verificações não são mensagens; uma pessoa que corresse para o átrio para comunicar a sua última verificação rapidamente se tornaria num pária social. As verificações não podem realmente ser lidas; um leitor pode esfolar-se por uma das mais curtas à força de um esforço heróico, mas isso não é ler. Não sendo legível e — literalmente — não sendo verbalizável, as verificações não podem ser internalizadas, transformadas, generalizadas, utilizadas, conectadas a outras disciplinas, e eventualmente incorporadas numa consciência comunitária. Não podem adquirir gradualmente credibilidade, como acontece com um teorema matemático; ou se acredita cegamente nelas, como um puro acto de fé, ou não se acredita em absoluto.

Neste ponto, alguns adeptos da verificação admitem que a analogia com a Matemática falha. Tendo argumentado que A — programação — se assemelha a B — matemática — e tendo subsequentemente descoberto que B não é nada como tinham imaginado, podem desejar argumentar, em vez disso, que A é como B', a sua versão mítica de B. Encontramo-nos na posição peculiar de levar a bom porto o argumento que era originalmente deles, afirmando que sim, de facto, A se assemelha a B; a nossa argumentação, no entanto, associa os termos de um modo diferente dos deles (ver as Figuras 1 e 2).

Matemática ..... *Programação*
teorema ..... programa
demonstração ..... verificação

Figura 1. A analogia original dos verificadores.

Matemática ..... *Programação*
teorema ..... especificação
demonstração ..... programa
demonstração formal imaginária ..... verificação

Figura 2. A nossa analogia.

Os verificadores que desejarem acabar com o sorriso e substituir B' devem, como

um apoio à compreensão, abandonar igualmente a linguagem de B — em particular, pode ser uma ajuda se não chamarem "demonstrações" às suas verificações. Quanto a nós, continuaremos a argumentar que a programação é idêntica à Matemática, e que os mesmos processos sociais que ocorrem nas demonstrações matemáticas condenam as verificações.

Existe uma objecção lógica fundamental à verificação, uma objecção no seu próprio campo do rigor formalista. Uma vez que o requisito para um programa é informal e que o programa é formal, deve haver uma transição, e a transição em si deve ser necessariamente informal. Ficámos desanimados ao saber que esta proposição, que nos parece auto-evidente, é controversa. Devemos, pois, sublinhar que, como antiformalistas, não iremos objectar à verificação nestes moldes; apenas questionamos como este passo inerentemente informal se enquadra na perspectiva formalista. Os adeptos da verificação perderam de vista as origens informais dos objectos formais com que trabalham? Pretendem afirmar que as suas formalizações são, de algum modo, incontroversas? Devemos confessas a nossa confusão e desilusão.

Existe depois uma outra dificuldade lógica, quase tão básica, e de modo algum tão subtil como a anterior: a demonstração formal que um programa é consistente com as suas especificações tem valor apenas se as especificações e o programa são desenvolvidos independentemente. Na atmosfera de programação-brincadeira da verificação experimental este critério é facilmente cumprido. Mas na vida real, se durante o processo de concepção um programa falha, ele é alterado, e as alterações são baseadas no conhecimento das suas especificações, ou então são alteradas as especificações, e estas alterações são baseadas no conhecimento do programa obtido através da falha. Em ambos os casos, a exigência de possuir um critério independente para comparar um com o outro deixa de se verificar. Esperamos, novamente, que ninguém sugira que os programas e as especificações sejam repetidamente modificados durante o processo de concepção. Essa seria uma posição de uma pobreza inerível — o tipo de pobreza que, tememos, costuma resultar do entusiasmo pela lógica formal.

No mundo real, os tipos de especificações de entrada e saída que acompanham o software para a produção raramente são simples. Tendem a ser longas e complexas e peculiares. Para citar um caso extremo, calcular as folhas os pagamentos para a companhia de caminhos de ferro francesa requer mais de 3 000 taxas de pagamento (uma para subir, outra para descer, e assim por diante). As especificações para qualquer compilador ou sistema operativo razoáveis enchem volumes — e ninguém pensa que são completos. Existem mesmo alguns casos de código de caixa negra,

algoritmos numéricos que se pode mostrar que funcionam no sentido em que são usados para construir aviões reais ou perfurar poços de petróleo reais, mas que funcionam por razões que ninguém conhece; as declarações de entrada para estes algoritmos não são sequer formuláveis, e muito menos formalizáveis. Tomando apenas um exemplo, durante anos soube-se que um algoritmo importante com o nome razoavelmente desprendido de Inverso Cuthill-McKee era muito melhor do que o simples Cuthill-McKee, sabido empiricamente, em testes de laboratório e experiências de campo e na produção. Só recentemente, no entanto, foi teoricamente demonstrada a sua superioridade (George, 1971), e mesmo então apenas com a demonstração matemática informal usual, não uma dedução formal. Durante todos os anos em que o Inverso Cuthill-McKee não esteve provado, apesar de ele automaticamente tornar inverificável qualquer programa no qual aparecia, os programadores obstinadamente continuaram a usá-lo.

Pode ser contraposto que, enquanto que na vida real as especificações são extensas e complicadas, não são profundas. As suas verificações são, de facto, nada mais que cadeias extremamente longas de substituições a verificar com a ajuda de identidades algébricas simples.

Tudo o que podemos dizer em resposta a isto é: precisamente. As verificações são longas e tortuosas mas pouco profundas; isso é o que está errado com elas. Mesmo a verificação de um programa insignificante pode ocupar dezenas de páginas, e não existe um momento de luz ou uma centelha de inteligência em nenhuma delas. Ninguém corre para o gabinete de um amigo com a verificação de um programa. Ninguém esboça uma verificação num guardanapo de papel. Ninguém retém um colega para ouvir uma verificação. Ninguém sequer a vai ler. Podemos sentir os olhos de cada um arregalarem-se só de pensar no assunto.

Foi sugerido que linguagens de muito alto nível, que podem lidar directamente com um largo espectro de objectos matemáticos ou linguagens funcionais, que se dizem poderem ser axiomatizadas concisamente, podem ser usadas para assegurar que a verificação se torne interessante e portanto sejam sensíveis a um processo social como o processo social da matemática.

Em teoria a ideia parece esperançosa; na prática, não funciona. Por exemplo, a condição de verificação seguinte aparece na demonstração de uma transformação rápida de Fourier escrita em MADCAP, uma linguagem de muito alto nível (Schwartz, 1973):

Se $S \in \{1,-1\}, b = \exp(2\pi i S / N), r$ é um inteiro, $N = 2^r$,

(1) $C = \{2j : 0 \le j \le N/4\}$ e

(2) $a = < a_r : a_r = b^{r \bmod (N/2)}, 0 \le r < N/2 >$ e

(3) $A = \{j : j \bmod N < N/2, 0 \le j < N\}$ e

(4) $A^* = \{j : 0 \le j < N\} - A$ e

(5) $F = < f_r : f_r = \sum_{k_1 \in R_n} k\left(b^{k_1 (r/2^{r-1}) \bmod N}\right), R_r = \{j : (j-r) \bmod (N/2) = 0\} >$ e $k \le r$

então

(1) $A \cap \left(A + 2^{r-k-1}\right) = \left\{x : x \bmod 2^{r-k} < 2^{r-k-1}, 0 \le x < N\right\}$

(2) $< \alpha_c \ \alpha_c > = < \alpha_r : \alpha_r = b^{r2^k \bmod (N/2)}, 0 \le r < N/2 >$

(3) $< (F_{A \cap \left(A+2^{r-k-1}\right)} + F_{\{j:0 \le j < N\} - A \cap \left(A+2^{r-k-1}\right)}) (< \alpha_c \ \alpha_c >^* (F_{A \cap \left(A+2^{r-k-1}\right)} + F_{\{j:0 \le j < N\} - A \cap \left(A+2^{r-k-1}\right)})) > =$

$= < f_r : f_r = \sum_{k_1 \in R_r} k_1 (\left(b^{\left(r/2^{r-k-1}\right)/\bmod N}\right), R_r = \left\{j : (j-r) \bmod 2^{r-k-1} = 0\right\} >$

(4) $< (F_{A \cap \left(A+2^{r-k-1}\right)} + F_{\{j:0 \le j < N\} - A \cap \left(A+2^{r-k-1}\right)}) (< \alpha_c \ \alpha_c >^* (F_{A \cap \left(A+2^{r-k-1}\right)} + F_{\{j:0 \le j < N\} - A \cap \left(A+2^{r-k-1}\right)})) > =$

$= < f_r : f_r = \sum_{k_1 \in R_r} k_1 (\left(b^{\left(r/2^{r-k-1}\right)/\bmod N}\right), R_r = \left\{j : (j-r) \bmod 2^{r-k-1} = 0\right\} >$

$< (F_{A \cap \left(A+2^{r-k-1}\right)} + F_{\{j:0 \le j < N\} - A \cap \left(A+2^{r-k-1}\right)}) (< \alpha_c \ \alpha_c >^* (F_{A \cap \left(A+2^{r-k-1}\right)} + F_{\{j:0 \le j < N\} - A \cap \left(A+2^{r-k-1}\right)})) > =$

$= < f_r : f_r = \sum_{k_1 \in R_r} k_1 (\left(b^{\left(r/2^{r-k-1}\right)/\bmod N}\right), R_r = \left\{j : (j-r) \bmod 2^{r-k-1} = 0\right\} >$

Isto não é o que se poderia chamar uma leitura agradável.

Alguns verificadores, embora concordando que a verificação simplesmente não funciona para a vasta maioria dos programas, argumentam que para umas poucas aplicações cruciais a agonia vale a pena. Referem-se ao controlo de tráfego aéreo, sistemas de mísseis e a exploração do espaço como áreas nas quais os riscos são tão grandes que pode ser justificado qualquer investimento de tempo e esforço.

Mesmo se isto fosse o caso, ainda insistiríamos em que a verificação abandonasse a reivindicação da sua primazia sobre todas as outras áreas da programação; ensinar estudantes de cursos introdutórios de programação a fazer verificações, por exemplo, deveria ser tão esotérico como ensinar estudantes em biologia introdutória a realizar cirurgia de coração aberto. Mas estas preocupações não afectam a nossa crença na impossibilidade básica em verificar qualquer sistema suficientemente amplo e flexível para fazer qualquer tarefa no mundo real. Por mais alto que seja o pagamento, ninguém nunca será capaz de se forçar a si próprio a ler as verificações incrivelmente longas e entediantes de sistemas da vida real, e, se não forem lidas, compreendidas e refinadas, as verificações não têm valor.

Pode, no entanto, ser argumentado que todas estas referências a legibilidade e

internalização são irrelevantes, que o objectivo da verificação é eventualmente construir um sistema de verificação automático.

Infelizmente existe uma grande evidência que aponta que sistemas de verificação completamente automáticos estão fora de hipótese. Os limites inferiores do comprimento de demonstrações formais para os teoremas matemáticos são imensas (Stockmeyer, 1974), e não existe razão para acreditar que tais demonstrações de programas seriam menores ou mais claras — muito pelo contrário. De facto, mesmo os adeptos mais fortes da verificação de programas não levam a sério a possibilidade de verificadores completamente automáticos. Ralph London, um proponente da verificação, fala de um sistema de fechado-para-almoço, que poderia ser deixado sem supervisão para triturar verificações; mas duvida que tal sistema se possa construir funcionando com uma fiabilidade razoável. Um grupo, desesperado da automação no futuro próximo, propôs que as verificações deveriam ser executadas por equipas de "matemáticos básicos", equipas matemáticas de baixo nível que verificariam as condições de verificação. A sensibilidade das pessoas que fizeram tal proposta parece estranha, mas serve claramente para indicar quão remota está a possibilidade de verificação automática.

Suponhamos, no entanto, que um verificador automático podia ser, de algum modo, construído. Suponhamos ainda que os programadores tinham, de algum modo, passado a ter fé nas suas verificações. Na ausência de qualquer base no mundo real para esta crença, seria de ser uma fé cega, mas não interessa. Suponhamos que a pedra filosofal tinha sido encontrada, que o chumbo podia ser transformado em ouro, e que os programadores estavam convencidos das vantagens de alimentar os seus programas nas mandíbulas abertas de um verificador. Parece-nos que o cenário antevisto pelos proponentes da verificação seria mais ou menos isto: o programador insere o seu pacote de 300 linhas de entrada/saída no verificador. Algumas horas depois, regressa. Encontra a sua verificação com 20 000 linhas e a mensagem "VERIFICADO".

Quando começamos a sentir que uma estrutura está logicamente, provavelmente certa, existe uma tendência para remover quaisquer redundâncias que originalmente lhe tínhamos incorporado por falta de compreensão. Levado ao extremo, esta tendência acarreta o chamado efeito de Titanic; quando a falha acontece de facto, ela é massiça e descontrolada. Por outras palavras, a severidade com a qual o sistema falha é directamente proporcional à intensidade da crença do projectista de que não podia falhar. Programas desenhados para serem claros e organizados apenas para que possam ser verificados serão particularmente vulneráveis ao efeito de Titanic. Vemos já sinais deste fenómeno. Nas suas notas sobre Euclid (Popek e outros, 1977)

uma linguagem específica para a verificação de programas, vários dos principais adeptos da verificação dizem "como esperávamos que todos os programas Euclid fossem verificados, não incorporámos provisões especiais para lidar com excepções. (...) Erros de software durante a execução não devem ocorrer em programas verificados". Erros não devem ocorrer? Sombras do barco que não se devia afundar.

Felizmente existem poucas razões para temer um tal futuro. Imaginemos o mesmo programador regressando para as tais 20 000 linhas. Que mensagem encontraria realmente, supondo que um verificador automático podia ser realmente construído? Certamente que a mensagem seria "NÃO VERIFICADO". O programador faria uma alteração, inseria de novo o programa, regressaria de novo. "NÃO VERIFICADO". De novo faria uma alteração, de novo inseriria o programa no verificador, e de novo "NÃO VERIFICADO". Um programa é um artefacto humano, um programa na vida real é um artefacto humano complexo, e qualquer artefacto humano com uma dimensão e complexidade suficientes é imperfeito. A mensagem nunca seria "VERIFICADO".

O papel da continuidade

> *Podemos, em geral, dizer, que uma ideia matemática é "significante" se poder ser associada, de um modo natural e iluminador, a um amplo complexo de outras ideias matemáticas.*
>
> *G. H. Hardy*

A única defesa realmente promissora, alguma vez proposta para a verificação é o argumento de crescimento. Aqui vai ele, no melhor que conseguimos reproduzi-lo:

(1) A verificação está agora na sua infância. Neste momento, as maiores tarefas que com que pode lidar são verificações de algoritmos como PROCURAR e modelar programas com MDC[3]. Com o tempo será capaz de atacar algoritmos cada vez mais complicados e modelar programas cada vez mais difíceis. Estas verificações são comparáveis às demonstrações matemáticas. São lidas, Geram o mesmo tipo de interesse e entusiasmo que os teoremas. Estão sujeitas aos processos sociais habituais que existem no raciocínio matemático, ou, já agora, no raciocínio em qualquer outra disciplina.

(2) Os grandes sistemas de produção são compostos de nada mais do que algoritmos e programas de modelação. Uma vez verificados, os algoritmos e os programas de modelação podem compor grandes sistemas de produção comuns, e

a verificação (assumidamente não legível) de um grande sistema será a soma de muitas verificações pequenas, atractivas, interessantes das suas componentes.

Não levantamos objecções a (1). De facto, os algoritmos foram demonstrados e as demonstrações lidas e discutidas e assimiladas muito antes da invenção de computadores — e numa notável ausência de maquinaria formal. A nossa aposta é que o estudo de algoritmos e programas de modelação se desenvolverá como qualquer outra actividade matemática, principalmente através de mecanismos informais e sociais, muito pouco, ou mesmo nada, através de mecanismos formais.

É com (2) que temos a nossa discordância fundamental. Argumentamos que não existe entre o mundo de PROCURAR ou de MDC e o mundo do software para a produção, sistemas de pagamentos que escrevem facturas reais, sistemas de horários que marcam eventos reais, sistemas de bilhetes que emitem bilhetes reais. E argumentamos que o mundo do software para a produção é, ele próprio, descontínuo.

Nenhum programador concordaria que os grandes sistemas de produção são compostos de nada mais que algoritmos e pequenos programas. Remendos, construções ad hoc, pensos rápidos e torniquetes, fogo de artifício, cola, cuspir e polir, código de assinatura, sangue suor e lágrimas, e, claro, este mundo e o outro — o calão colorido do programador prático parece dizer algo sobre a natureza das estruturas com que trabalha; talvez os teóricos o devem ouvir. Tem sido estimado que mais de metade do código em qualquer sistema de produção real consiste em interfaces com o utilizador e mensagens de erro — estruturas ad hoc, informais que não são, por definição, verificáveis. Mesmo os próprios verificadores compreendem, por vezes, a natureza inverificável da maior parte do software real. C. A. R. Hoare tem sido citado (McGowan, McHenry, 1978) afirmando "em muitas aplicações, os algoritmos não desempenham quase nenhum papel, e, certamente não apresentam quase nenhum problema". (Gostaríamos de relatar que, após isto, ele encolheu os ombros resignado e abandonou a verificação, mas não temos essa sorte).

Ou então vejamos a diferença entre o mundo do MDC e o mundo do software de produção de outro modo: As especificações para algoritmos são concisas e ordenadas, enquanto que as especificações para os sistemas no mundo real são imensas, frequentemente da mesma ordem de grandeza que os próprios sistemas. As especificações para algoritmos são altamente estáveis, estáveis durante décadas ou mesmo séculos; as especificações para sistemas reais variam em cada dia ou em cada hora (como qualquer programador pode testemunhar). As especificações para algoritmos são exportáveis, gerais; as especificações para sistemas reais são ideosincráticas e ad hoc. Estas não são diferenças de grau. São diferenças de qualidade. Tomando conta durante uma hora de uma criança que dorme não se

compara com educar uma família de dez — os problemas são essencial e fundamentalmente diferentes.

E no mundo do software para a produção também não existe continuidade. O argumento de crescimento parece baseado na noção difusa de que o mundo da programação é como o mundo da física newtoniana — feito de funções suaves, contínuas. Mas, de facto, os programas são irregulares e cheios de buracos e cavernas. Todos os programadores sabem que alterar uma linha ou por vezes mesmo um bit pode destruir completamente um programa ou mutilá-lo de formas que não compreendemos ou podemos prever. E, no entanto, noutras alturas mudanças substanciais parece não alterarem nada; contam-se muitas histórias de partidas e actos de vandalismo que frustraram os autores de tais actos ao permanecerem para sempre escondidas.

Existe uma história de ficção científica clássica sobre um viajante no tempo que regressa às selvas primitivas para observar dinossauros e depois regressa encontrando o seu próprio tempo profundamente alterado. A política, a arquitectura, a linguagem — mesmo as plantas e os animais parecem errados, distorcidos. Só depois de remover o fato de viagem no tempo ele compreende o que aconteceu. No tacão da bota, transportado do passado, e portanto incapaz de executar a sua função na evolução do mundo, encontra-se a asa esmagada de uma borboleta. Todos os programadores conhecem a sensação: uma mudança trivial, minúscula, lança o caos num sistema gigantesco. Até que saibamos mais sobre programação, faríamos melhor, para todos os efeitos práticos considerar os sistemas como sendo compostos, não de estruturas robustas como algoritmos e pequenos programas, mas de asas de borboleta.

A natureza descontínua da programação soa como o dobre de finados para a verificação. Um investigador suficientemente fanático pode estar disposto a dedicar dois ou três anos a verificar uma peça significante de software se lhe pudessem garantir que o software permaneceria estável. Mas programas na vida real necessitam de manutenção e de modificações. Não há razões para acreditar que verificar um programa modificado seja mais fácil do que verificar o original pela primeira vez. Não há razões para acreditar que uma grande verificação pode ser a soma de muitas pequenas verificações. Não há razões para acreditar que uma verificação se pode transferir para qualquer outro programa — nem sequer para um programa diferindo apenas numa linha do original.

E é esta descontinuidade que impede a possibilidade de aperfeiçoar as verificações através do tipo de processos sociais que aperfeiçoam as demonstrações matemáticas. O fanático solitário poderia construir a sua própria verificação, mas

nunca teria nenhuma razão para ler as dos outros, nem nenhum outro estaria alguma vez disposto a ler a sua. Não se poderia desenvolver nenhuma comunidade. Mesmo o mais zeloso verificador poderia ser convencido a ler uma verificação apenas se pensasse que poderia usar ou pedir emprestado ou roubar algo dela. Nada o poderia forçar a ler a verificação de outrem depois de ter compreendido que nenhuma verificação tem qualquer conexão necessária com qualquer outra verificação.

Acreditar no software

O programa em si é a única descrição completa do que o programa fará.

P. J. Davis

Uma vez que os computadores podem escrever símbolos e movê-los para um e outro lado com um gasto mínimo de energia, é tentador concluir imediatamente que tudo é possível no domínio dos símbolos. Mas a realidade não cede tão facilmente; a física não se quebra de repente. Tanto é possível construir estruturas simbólicas sem fazer uso de recursos, como é possível construir estruturas materiais sem os usar. Mesmo para as mais triviais teorias matemáticas existem proposições simples cujas demonstrações formais seriam inacreditavelmente longas. A excelente apresentação de Albert Meyer sobre a história de tal investigação (1974) conclui com uma interpretação marcante de quão difícil pode ser deduzir mesmo afirmações matemáticas bastante simples. Suponhamos que codificamos fórmulas lógicas como cadeias binárias e decidimos construir um computador que decidirá da verdade de um conjunto simples de fórmulas de comprimento, digamos, no máximo um milhar de bits. Suponhamos que nos permitimos mesmo o requinte de uma tecnologia que produz componentes electrónico do tamanho de protões ligados por fios infinitamente finos. Mesmo assim, o computador que desenharíamos deveria encher de um modo denso todo o universo observável. Esta observação sobre a extensão de deduções formais concorda precisamente com a nossa intuição sobre a dimensão do detalhe incorporado nas demonstrações matemáticas ordinárias, de todos os dias. Muitas vezes usamos "vamos supor, sem perda de generalidade..." ou "portanto, renumerando, se necessário,..." em substituição de uma enorme quantidade de detalhe formal. Insistir no detalhe formal seria um desperdício idiota de recursos. Quer as estruturas simbólicas, quer as materiais devem ser concebidas muito cautelosamente. Os recursos são limitados, o tempo é limitado, a energia é limitada. Nem mesmo o computador pode alterar a natureza finita do universo.

Supomos que estas restrições impediram os adeptos da verificação de apresentar

o que poderiam ser provas razoavelmente convincentes apoiando os seus métodos. A falta, depois deste tempo todo, de apenas uma única verificação de um sistema em funcionamento tem sido atribuída à juventude do campo. Os verificadores argumentam, por exemplo, que só agora começaram a entender os invariantes de ciclo[4]. à primeira vista, isto poderá parecer como uma outra variação do argumento de crescimento. Mas, de facto, existem numerosas classes de sistemas da vida real praticamente sem ciclos — raramente estes ocorrem em aplicações de software comerciais. E, no entanto, nunca existiu uma verificação de, por exemplo, um sistema em Cobol que imprime cheques reais. A inexistência de uma sequer torna duvidoso que possam existir muitas algures no futuro. Os recursos e o tempo e a energia são igualmente limitados para os verificadores como o são para nós outros.

Temos então de enfrentar dois problemas que ocuparam os engenheiros durante muitas gerações: primeiro, as pessoas devem mergulhar em actividades que não compreendem. Segundo, as pessoas não podem criar mecanismos perfeitos.

Como conseguem então os engenheiros criar estruturas fiáveis? Primeiro, utilizam processos sociais muito parecidos com os processos sociais dos matemáticos para conseguir sucessivas aproximações à compreensão. Segundo, possuem uma visão madura e realista do que significa "fiáveis"; em especial, uma coisa que nunca significa é "perfeito". Não existe modo de deduzir logicamente que as pontes ficam de pé, ou que os aviões voam, ou que as centrais eléctricas produzem electricidade. É verdade que nenhumas pontes cairiam, nenhuns aviões se despenhariam, nenhuns sistemas eléctricos se iriam abaixo se os engenheiros previamente demonstrassem a sua perfeição antes de os construir — seria verdade porque eles nem sequer os construiriam.

A analogia com a programação é com qualquer sistema que funcione, útil, do mundo real. Tomemos, por exemplo, um sintetizador organo-químico chamado SYNCHEM (Gelerenter, e outros, 1973). O critério de fiabilidade para este programa é especialmente imediato — se sintetiza um químico, funciona; se não, não funciona. Nenhum aumento do grau de correcção poderia alguma vez aspirar a melhorar este critério; de facto, não é de modo nenhum claro como se poderia sequer começar a formalizar um tal critério de modo que permitisse a verificação. Mas tentar aumentar o número de químicos que o programa pode sintetizar é um projecto útil e continuado.

É apenas chauvinismo dos símbolos que leva os cientistas informáticos pensar que as nossas estruturas são muito mais importantes que as estruturas materiais de modo que (a) elas são perfeitas e (b) a energia necessário para torná-las perfeitas deve ser gasta. Contrapomos que (a) elas não podem ser perfeitas e (b) a energia não deve

ser gasta na tentativa fútil de as tornar perfeitas. Não é por acaso que a visão probabilista da verdade matemática está aliada de perto com a noção de fiabilidade da engenharia. Talvez devêssemos fazer uma clara distinção entre a fiabilidade dos programas e a perfeição dos programas — e concentrar os nossos esforços na fiabilidade.

O desejo de tornar os programas correctos é construtivo e valioso. Mas a visão monolítica da verificação é cega aos benefícios que poderia resultar da aceitação de um critério de correcção como o critério de correcção para as demonstrações matemáticas reais ou o critério de correcção para as estruturas de engenharia reais. A busca da funcionalidade dentro de limites económicos, a disposição para canalizar a inovação para a reciclagem de arquitecturas bem sucedidas, a confiança no funcionamento de uma comunidade de iguais -- todos os mecanismos que tornam a engenharia e a matemática realmente funcionar são obscurecidos pela busca infrutífera da verificabilidade perfeita.

Que elementos poderiam contribuir para tornar a programação mais como a engenharia e a matemática? Um mecanismo que pode ser explorado é a criação de estruturas gerais cujos exemplos específicos se tornam mais fiáveis à medida em que aumenta a fiabilidade da estrutura geral[5]. Esta noção apareceu em várias incarnações, das quais a insistência de Knuth em criar e compreender algoritmos úteis em geral é uma das mais importantes e encorajadoras. A metodologia da programação em equipa de Baker (1972) é uma tentativa explícita de expor o software aos processos sociais. Se a reutilização se tornar um critério de uma arquitectura eficaz, uma comunidade cada vez mais ampla examinará as ferramentas de programação mais comuns.

O conceito de software verificável tem estado connosco demasiado tempo para ser facilmente desalojado. Na prática da programação, no entanto, não se deve permitir que a verificabilidade obscureça a fiabilidade. Os cientistas não devem confundir modelos matemáticos com a realidade — e a verificação não é senão um modelo de credibilidade. A verificabilidade não é e não pode ser uma preocupação dominante no desenho de software. A economia, prazos, comparações custo-benefício, estilo pessoal e de grupo, os limites do erro aceitável — todas estas têm muito mais peso no design que a verificabilidade ou a não verificabilidade.

Até agora tem havido pouca discussão filosófica sobre tornar o software fiável em vez de verificável. Se os adeptos da verificação pudessem redefinir os seus esforços e reorientar-se para este objectivo, ou se uma outra visão de software que se inspirasse nos processos sociais da matemática e nas modestas expectativas da engenharia pudesse aparecer, os interesses da programação da vida real e da

informática teórica seria ambos melhor servidos.

Mesmo se, por alguma razão que não somos capazes de compreender, se provasse que estávamos completamente errados e os verificadores completamente certos, não é este o momento de restringir a investigação em programação. Sabemos hoje demasiadamente pouco para imaginar quais as direcções mais frutuosas. Se a nossa argumentação não convence ninguém, se a verificação ainda parecer um caminho a ser explorado, que o seja; nós três apenas podemos tentar argumentar contra a verificação, não varrê-la da face da Terra. Mas imploramos aos nossos amigos e colegas para não estreitarem demasiado a sua visão a esta perspectiva, por mais promissora que ela pareça. Que ela não seja a única visão, a única via. Jacob Bronowski tem uma importante revelação sobre uma época de uma outra disciplina que é capaz de ser semelhante à nossa no desenvolvimento da computação: "uma ciência que ordena o seu conhecimento demasiado cedo fica sufocada. (...) A esperança dos alquimistas medievais de que os elementos pudessem ser alterados não era tão irrealista como outrora pensámos. Mas foi apenas danosa para uma química que ainda não tinha compreendido a composição da água e do sal comum".

Agradecimentos

Queremos agradecer especialmente àqueles que nos ofereceram fóruns públicos — a comissão de programa do 4th POPL por nos dar a nossa primeira oportunidade; Bob Taylor e Jim Morris por nos deixarem exprimir as nossas opiniões em Zerox PARC; L. Zadeh e Larry Rowe por terem feito o mesmo no Computer Science Department da University of California at Berkely; Marvin Dennicoff e Peter Wegner por permitirem dirigir-nos à DOD Conference sobre direcções de investigação na tecnologia de software.

Queremos também agradecer a Larry Landweber por nos permitir visitar a University of Wisconsin at Madison durante um verão. O ambiente e o apoio de Ben Noble e a sua equipa no Mathematics Research Center foi fundamental para nos deixar trabalhar produtivamente.

As sementes destas ideias foram formadas a partir de discussões mantidas na DOD Conference on Software Technology em 1976 em Durham, Carolina do Norte. Desejamos agradecer, em particular, a J. R. Suttle, que organizou esta conferência e tem dado um encorajamento constante para o nosso trabalho.

Também desejamos agradecer aos nossos muitos amigos que discutiram estes assuntos connosco. Incluem: Al Aho, Jon Barwise, Manuel Blum, Tim Budd, Lucio Chiaraviglio, Philip Davis, Peter Denning, Bernie Elspas, Mike Fischer, Ralph

Griswold, Leo Guibas, David Hansen, Mike Harrison, Steve Johnson, Jerome Kiesler, Kenneth Kunen, Nancy Lynch, Albert Meyer, Barkley Rosser, Fred Sayward, Tim Standish, Larry Travis, Tony Wasserman e Ann Yasuhara.

Queremos também agradecer quer a Bob Grafton, quer a Marvin Dennicoff de ONR os seus comentários e apoio.

Apenas os que viram versões anteriores deste texto conseguem apreciar a contribuição feita pela nossa editora, Mary-Claire van Leunen. Onde é costume em informática listar uma lista de crédito "conforme devido a ...", isso talvez seja uma melhor descrição do serviço que ela desempenhou.

Notas

[1] *Cofinality Principle* no original (NT).

[2] *forcing* no original (NT).

[3] *GCD* no original, isto é, Máximo Divisor Comum (NT).

[4] *loop invariants* no original (NT).

[5] Este processo foi recentemente denominado "abstracção", mas pensamos que, por uma diversidade de razões, "abstracção" é um mau termo. É facilmente confundido com a noção completamente diferente de abstracção em matemática, e muitas vezes o que foi tomado como abstracção na literatura da ciência informática é simplesmente a remoção de detalhes de implementação.

Referências

Baker, F. T. (1972). Chief programmer team management of production programing. *IBM Syst. J.*, *11*(1), 56-73.

Cohen, P. J. (1963, 1964). The independence of the continuum hypothesis. *Proc. Nat. Acad. Sci., USA, Part I, 50,* 1143-1148, *51*, 105-110.

Davis, P. J. (1972). Fidelity in mathematical discourse: Is one and one really two? *The Amer. Math. Monthly*, *79*(3), 252-263.

Bateman, P. e Diamond, H. (1978). John E. Littlewood (1885-1977): An informal obituary. *The Math. Intelligencer*, *1*(1), 28-33.

Gelerenter, H. e outros (1973). The discovery of organic synthetic roots by computer. *Topics in Current Chemistry*, *41*, 113-150.

George, J. A. (1971). *Computer implementation of the finite element method.* Tese doutoramento, Stanford University.

Heath, T. L. (1956). *The thirteen books of Euclid's Elements.* Nova Iorque: Dover.

Heawood, P. J. (1890). Map colouring theorems. *Quaterly J. Math., Oxford Series*, *24*, 322-339.

Jeck, T. J. (1973). *The axiom of choice.* Amesterdão: North Holland.

Kempe, A. B. (1879). On the geographical problem of the four colors. *Amer. J. math.*, *2*, 193-200.

Kolata, G. B. (1976). Mathematical proof: The genesis of reasonable doubt. *Science*, *192*, 989-990.

Lakatos, I. (1976). *Proofs and refutations: The logic of mathematical discovery*. Cambridge: Cambridge University Press.

Manin, Y. I. (1977). *A course in mathematical logic*. Springer-Verlag.

McGowan, C. e McHenry, R. (Eds.). (1978) *Software Management* . A publicar em *Research Directions in Software Technology*. Cambridge: MIT Press.

Meyer, A. (1974). The inherent computational complexity of theories of ordered sets: A brief survey. Int. Cong. of Mathematicians. Agosto 1974.

Popek, G. e outros (1977). Notes on the design of Euclid. *Proc. Conf. Language Design for Reliable Software, SIGPLAN Notices (ACM) 12*(3), 11-18.

Rabin, M. O. (1976). Probabilistic algorithms. Em J. F. Traub (Ed.) *Algorithms and complexity: New directions and recent results* (pp. 21-40). Nova Iorque: Academic Press.

Schwartz, J. (1973). *On programming*. Nova Iorque: New York U.

Stockmeyer, L. (1974). *The complexity of decision problems in automata theory and logic*. Tese de doutoramento. Cambridge: MIT.

Ulam, S. M. (1976). *Adventures of a mathematician*. Nova Iorque: Scribner's.

Comunicação originalmente publicada sob o título "Social processes and proofs of theorems and programs" em Communications of the ACM, *22(5), Maio 1979, 271-280, e posteriormente publicada em T. Tymoczko (Ed.), (1986),* New directions in the philosophy of mathematics, an anthology, *pp. 267-285. Boston: Birkhäuser.*

A selecção do texto contou com a colaboração do GruPo TEM, Grupo Português de Teoria de Educação Matemática..

Tradução elaborada por José Manuel Matos.