Un interviu cu Brian Kernighan

Mihai Budiu — mailto:mihaib+@cs.cmu.edu
http://www.cs.cmu.edu/~mihaib/

iulie 2000

Contents

1 Introducere

Pentru al doilea an consecutiv am avut şansa de a obţine o slujbă de vară la laboratoare Bell (Bell Labs) ale companiei Lucent. Aceste laboratoare de cercetare sunt printre cele mai prestigioase din lume, numărînd nu mai puţin de 11 premii Nobel. De Bell Labs se leagă evenimente epocale din istoria tehnicii de calcul: John Bardeen, Walter Brattain şi William Shockley au descoperit aici în 1947 efectul de tranzistor, Arthur L. Schawlow, şi Charles H. Townes au inventat în 1958 Laserul, Claude Shannon a fundamentat teoria informaţiei în 1948, Ken Thomson şi Dennis Ritchie au creat sistemul de operare Unix în 1969 pentru care Ritchie a sintetizat limbajul C, care a evoluat apoi tot aici prin munca lui Bjarne Stroustrup în 1984 în C++.

Anul trecut m-am "aventurat" să cer autorilor cărţii care descrie limbajul C, Dennis Ritchie şi Brian Kernighan, un autograf. Anul acesta mi-am luat inima în dinţi şi am solicitat un interviu domnului Brian Kernighan. Acesta a răspuns cu foarte mare amabilitate:

```
Date: Mon, 10 Jul 2000 14:52:15 -0400
To: mihaib@research.bell-labs.com
From: "Brian Kernighan" <bwk@research.bell-labs.com>
Subject: re: odd request

sure, no problem.  i'm probably pretty boring, but since
i don't read romanian, you can make things up...
```

```
come by any time; i'm mostly around.

brian
```

(sigur, nici o problemă. sunt probabil destul de plictisitor, dar pentru că nu citesc româneşte poţi inventa tot felul de lucruri. vino oricînd, sunt prin preajmă.)

Iată aici textul interviului transcris după înregistrarea făcută cu ajutorul unui calculator echipat cu un microfon. Versiunea în limba engleză a interviului va fi disponibilă din pagina mea de web, pentru a demonstra că de fapt nu am inventat mare lucru.

2 Scurtă fişă biografică

Brian Kernighan a obţinut o diplomă în ingineria fizicii de la Universitatea din Toronto în 1964 şi un doctorat în inginerie electrică de la universitatea Princeton în 1969. În acelaşi an a venit la Bell Labs, unde ocupă şi în ziua de azi acelaşi birou (care a fost vopsit o dată). La Bell Labs a făcut cercetare în algoritmi combinatoriali (algoritmul de partiţie a grafurilor Lin-Kernighan din 1970 este folosit în continuare de o mulţime de scule CAD pentru plasarea circuitelor), metode de preparare automată a documentelor (programul `eqn` creat de el împreuna cu Lorinda Cherry în 1974 este distribuit cu toate sistemele Unix standard), limbaje de programare (limbajul `awk`, care a fost conceput de Aho, Weinberger şi Kernighan este un utilitar Unix standard, pe baza căruia a fost conceput limbajul Perl) şi metodologia programării, despre care a scris mai multe cărţi, cea mai recentă fiind "Practica programării", 1999, împreuna cu Rob Pike.

Brian Kernighan este autorul a peste 80 de articole ştiinţifice şi cărţi, dintre care cea mai celebra este "Limbajul de programare C" din 1978, reeditată în 1984, care este tradusă în peste 22 de limbi (dar aparent nu şi în română).

3 Versiunea în limba română

M: *Care e modul corect în care se pronunţă numele dumneavoastră? Am auzit că nu este chiar cum ne-am astepta.*

K: Se pronunţă Kărnihăn. Litera "g" nu se aude.

M: *Aţi ales să lucraţi în informatică într-o perioadă în care aceasta nu era cea mai evidentă alegere pentru o carieră. Ce ne puteţi spune despre cum aţi făcut această alegere şi ce gîndiţi retroactiv despre această decizie?*

K: E adevărat că am început să lucrez cu calculatoarele, probabil pe la mijlocul anilor şaizeci, cînd lucrurile erau destul de la început, şi am făcut-o din accident. Cred că am văzut primul meu calculator în 1963, era un IBM 650 destul de vechi. Nu am început programarea serioasă decît prin 1964 cînd eram în student în ultimul an. Dar era foarte amuzant, şi era înainte ca informatica să fie un "domeniu" de sine-stătător. Cînd am mers la studii postuniversitare exista un program de informatică în departamentul de inginerie electrică de la universitatea Princeton; asta era destul de tipic pentru multe alte locuri: nu exista un domeniu academic separat al informaticii, ci doar o parte a unui departament care avea un calculator sau ceva oameni interesaţi de calcule, aşa că am

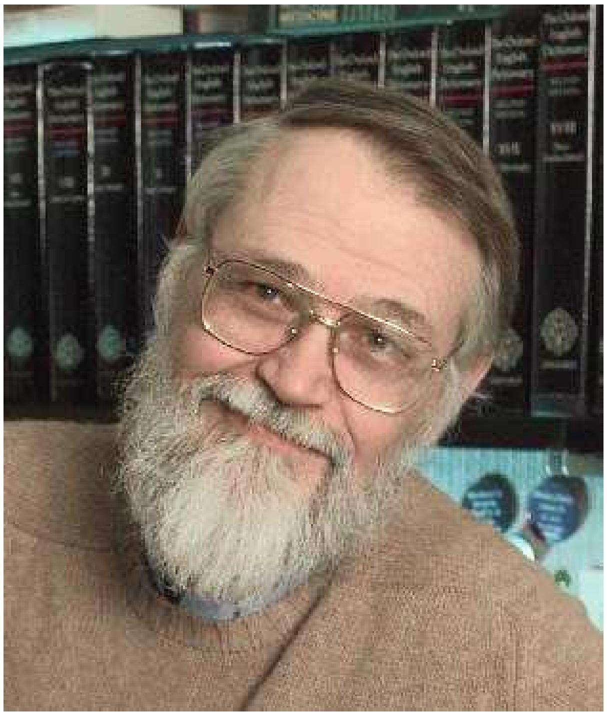

Figura 1: Brian Kernighan

intrat şi eu în domeniu, în întregime accidental. Dar a fost un accident norocos, pentru că în mod evident au fost o mulţime de întîmplări interesante în domeniu.

M: *Aţi lucrat în acest domeniu multă vreme, şi aţi fost un jucător important în evoluţia informaticii. Unele din lucrările dumneavoastră au influenţat profund domeniul. Puteţi arăta unele dintre lucrurile pe care le consideraţi avansuri fundamentale în informatică în ultimii 30 de ani; ce schimbări de paradigmă crede,ti că s-au petrecut?*

K: Cred că a fost un număr mare de schimbări, nu neapărat în informatică [ca ştiinţă], ci în calculatoare în general. Evident, faptul că hardware-ul a devenit enorm de rapid: legea lui Moore, deşi e o schimbare cantitativă, cînd este aplicată ca o creştere exponenţială timp de 30 de ani are un impact enorm; unele părţi ale acestei schimbări depind de informatică, dar nu toate. Pe de altă parte, lucrurile cu care sunt mai familiar, şi în care sunt mai interesat tehnologic, este folosirea feluritelor limbaje de programare în aşa fel încît să fim mai capabili să transmitem maşinilor ceea ce vrem să facem. Avansurile în domeniul limbajelor de programare, a tehnologiilor care ne permit să înţelegem cum să exprimăm comenzile pentru maşini, au avut de asemenea un impact enorm. Desigur, pe măsură ce maşinile deveneau mai puternice îţi puteai permite să aloci mai multe resurse pentru a folosi limbaje care nu ar fi fost eficiente cu 25 sau 30 de ani în urmă, dar care acum sunt perfect utilizabile. Alte schimbări importante sunt îmbunătăţirile algoritmice, teoria NP-completitudinii, care ne permite să gîndim despre ceea ce este greu şi ceea ce este uşor de calculat. Dar în ceea ce mă priveşte lucrurile pe care le găsesc cele mai interesante sunt dezvoltările în domeniul limbajelor de programare.

M: *De-a lungul timpului aţi lucrat în foarte multe domenii: algoritmi pentru grafuri, software engineering, scule software, tipografie digitală, dar cea mai mare parte din cercetarea dumneavoastră este în limbaje de programare. Care sunt interesele dumneavoastră curente în cercetare?*

K: [rîzînd] E interesant, ceea ce am făcut în ultimele zile este să meşteresc (hack-uiesc) la probabil cele mai vechi rădăcini ale mele în domeniul prezentării computerizate de documente, sau tipografie digitală, dacă preferi. Am luat `eqn`, care e un program pe care Lorinda Cherry şi cu mine l-am scris în 1974, şi îl convertesc pentru a produce rezultate în XML sau HTML, în aşa fel încît să putem pune mai uşor formule matematice în pagini de web. Mai multe persoane au încercat să facă acest lucru în felurite moduri, dar nici unul dintre ele nu pare să meargă prea bine, sau să fie finalizat. Noi avem nevoie de aşa ceva în companie, aşa că eu lucrez la un program C pe care l-am scris literal cu mai mult de 25 de ani în urmă. E un program C foarte mic, şi mă distrează foarte tare să încerc să-l transform. Ăsta e unul din lucrurile pe care le fac, un lucru mărunt, dar e amuzant să te întorci şi să faci ceva la care te-ai ostenit cu atîta vreme în urmă.

Un alt lucru la care lucrez acum este un limbaj numit AMPL, pe care David Gay şi cu Bob Fourer şi cu mine l-am creat. AMPL este un limbaj folosit pentru a specifica probleme de optimizare, ca de exemplu pentru a descrie probleme de programare lineară, dar nu numai. Încercăm să finisăm produsul pentru a putea fi folosit în procese mai complicate: de exemplu am învelit sistemul cu o interfaţă orientată pe obiecte ca să poată fi îngropat în alte programe sau folosit ca un obiect CORBA sau COM.

Astea sunt cele două lucruri de care mă ocup acum.

M: *Dacă tot am vorbit despre limbaje de programare, se mai compilează vechiul program* `eqn`*?*

K: Da, se mai compilează. Probabil că nu l-am mai compilat de cinci sau zece ani, şi s-a compilat fără nici o problemă. E un program C foarte mic şi simplu; cred că l-am convertit la ANSI C la sfîrşitul anilor '80. Ceea ce fac acum este mai ales să tai din el, pentru că limbajul de ieşire este mult mai simplu decît înainte.

M: *Cei mai mulţi oameni vă cunosc probabil din cauza cărţii de C, aşa că o să vă pun două întrebări legate de C. Limbajul C a avut o influenţă enormă; care credeţi că sunt trăsăturile sale cele mai valoroase?*

K: C are cel mai bun echilibru pe care l-am văzut vreodată între putere şi expresivitate. Poţi face aproape orice vrei programînd într-un fel destul de evident şi ai întotdeauna un model mental bun a ceea ce se va întîmpla pe calculator; poţi prezice destul bine cît de repede o să meargă, înţelegi ce se petrece şi ai libertate totală să faci ce vrei. C nu te constrînge în vreun fel, nu te obligă să foloseşti un anumit stil de programare; pe de altă parte nu îţi oferă o grămadă de facilităţi, nu are o bibliotecă gigantică, dar în ceea ce priveşte realizarea a ceva fără prea mult efort, nu am văzut nimic mai bun pînă în ziua de azi. Există alte limbaje reuşite pentru aplicaţii specifice, dar dacă ar fi să eşuez pe o insulă pustie cu un singur compilator, aş vrea un compilator de C.

M: *De fapt C este şi limbajul meu de programare favorit, şi pe care l-am folosit pentru majoritatea programelor mele. Dar de cînd am început să scriu compilatoare pentru C, trebuie să mărturisesc că limbajul mi se pare mai puţin plăcut: unele optimizări sunt foarte greu de făcut. Care sunt din punctul dumneavoastră de vedere trăsăturile negative ale limbajului?*

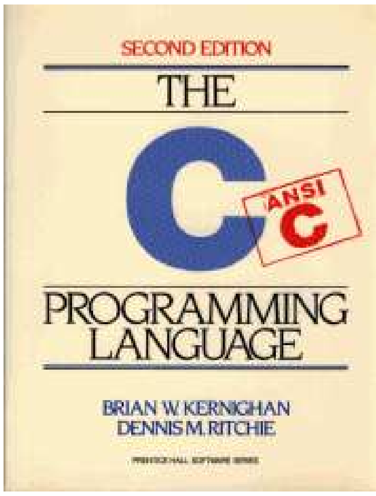

Figura 2: Cartea de C

K: Nu pot comenta despre trăsăturile negative; îţi reamintesc că C este în întregime creaţia lui Dennis Ritchie. Eu am contribuit doar la popularizarea sa, şi în particular nu pot spune ce e greu şi ce e uşor de compilat în C. Sunt cîteva lucruri trivial greşite în C: instrucţiunea `switch` putea fi mai bine proiectată, precedenţa unora dintre operatori este greşită, dar astea sunt lucruri mărunte şi lumea a învăţat să le evite. Cred că adevărata problemă a limbajului C este că nu-ţi pune la dispoziţie destule mecanisme pentru a scrie programe cu adevărat mari, pentru a crea ziduri de protecţie în interiorul programelor, care izolează feluritele bucăţi. Nu se pune problema că nu poţi face astfel de lucruri, ca nu poţi simula programare orientată pe obiecte sau alte metodologii de programare în C, poţi să faci toate astea, dar compilatorul şi limbajul nu te vor ajuta. Dar dacă luăm în considerare că C are aproape 30 de ani şi că a fost creat cînd calculatoarele erau minuscule în comparaţie cu ceea ce avem acum, cred că e o creaţie admirabilă, care a trecut de testul scurgerii timpului extrem de bine. Nu sunt multe lucruri pe care le-aş schimba în C.

Cîteodată scriu în C++ în loc de C. Cred că C++ este practic un limbaj prea mare, deşi pentru fiecare din elementele lui există un motiv de a exista. Cînd scriu un program C de orice dimensiune folosesc probabil 75, 80 sau chiar 90% din elementele limbajului. Cu alte cuvinte, cea mai mare parte a C-ului este folositoare indiferent de programul pe care îl scrii. Pe de altă parte, cînd scriu C++, folosesc probabil mai puţin de 10% din elementele sale, iar despre celelalte 90% aş putea zice că nu le înţeleg. În sensul asta pot argumenta că C++ este un limbaj prea mare, deşi C++ îţi oferă multe din mecanismele de care ai nevoie pentru a scrie programe mari: îţi permite să construieşti obiecte, şi să protejezi reprezentările lor interne cu o faţadă solidă de care nu poţi trece. C++ are o cantitate enormă de mecanisme utile pe care C nu ţi le dă.

M: *Am o întrebare despre cercetarea în limbaje de programare. E interesant de exemplu că Java este foarte mult ridicat în slăvi; comunitatea programatorilor se împarte între cei care laudă meritele şi cei care critică lipsurile limbajului. Java într-adevăr a fost înzestrat cu nişte trăsături care derivă din cercetare (cum ar fi colectorul de deşeuri — garbage collection), dar tot cercetătorii arată oricui vrea să-i asculte erori de proiectare (cum ar fi vectorii, care sunt covarianţi). O cantitate enormă de cercetare se desfăşoară în limbaje de programare, şi foarte multă cercetare interesantă în domeniul* *limbajelor de programare funcţionale, dar nu se prea văd rezultatele acestei cercetări în "lumea reală", adică în limbajele pe care le folosesc programatorii obişnuiţi în munca de lor de zi cu zi. În locul acestor limbaje academice apar din senin limbaje ad-hoc, cum ar fi Perl şi Python. Unde vedeţi buba, ce nu funcţionează cum trebuie?*

K: Asta este din păcate o întrebare foarte bună, şi există o oarecare rivalitate aici la Bell Labs între un grup extrem de puternic de limbaje funcţionale şi un grup care foloseşte limbaje ad-hoc, mai pragmatice. Sincer să fiu, nu ştiu de ce limbajele funcţionale nu au succes. De exemplu ML, care este probabil cea mai bună combinaţie, cea care ar trebui să domine: chiar dacă e un limbaj extrem de atent proiectat, la care au contribuit mult timp o mulţime de cercetători, care a dat naştere la o cantitate enormă de tehnologii de compilare, nu pare să fie adoptat pe scară largă. Simplificînd extrem, şi probabil supărînd pe unii dintre prietenii mei, aparent singurul lucru care se face în ML sunt compilatoare de ML [rîde]. În mod intenţionat exagerez, dar de fapt lucrurile cam aşa stau, şi nu ştiu exact de ce. Din punctul meu de vedere unul dintre motivele pentru care ML şi alte limbaje funcţionale nu au mai mult succes este pentru că necesită o oarecare abilitate matematică şi o gîndire mai abstractă, pe care o grămadă de lume, în care mă includ şi eu, nu o posedă. În schimb limbajele ca C sunt extrem de operaţionale şi poţi vedea foarte clar cum fiecare bucăţică dintr-un program se translatează direct într-o acţiune a calculatorului. Poate dacă aş fi fost crescut şi educat în alt fel aş fi mulţumit cu ML şi aş găsi C cam nesigur (unsafe), periculos şi nu prea expresiv. Dar impresia mea este că limbajele funcţionale au fost create de o comunitate de oameni cu puternice înclinaţii matematice şi că au nevoie de un raţionament matematic substanţial, şi ca atare nu vor fi pe placul omului de rînd.

M: *Deci sugestia dumneavoastră ar fi ca cercetătorii să coboare nivelul limbajului, pentru a-i promova calităţile pozitive?*

K: Nu am răspuns de fapt la partea a doua a întrebării tale, de ce cercetarea în limbaje nu a avut mai mult efect. Cred că de fapt cercetarea a avut un efect, în domenii ca tehnologia parserelor, generarea de cod, indiferent de limbajul în chestiune. Cercetarea a produs scule de mare succes pentru a manipula limbaje, dar a avut mai puţin efect în proiectarea limbajelor însele.

Limbajele care au succes sunt foarte pragmatice, şi sunt adesea "pătate" pentru că încearcă să rezolve probleme reale. C++ este un exemplu excelent de limbaj care are o grămadă de hibe. Una dintre ele este că încearcă foarte tare să rămînă compatibil cu C: e compatibil la nivelul fişierelor obiect, şi e aproape compatibil la nivelul surselor. Din cauza asta sunt o grămadă de lucruri urîte în limbaj, probleme sintactice ciudate, comportări semantice bizare. Într-un anume sens asta e rău şi nimeni nu ar trebui să facă aşa ceva, dar unul dintre motivele pentru care C++ a avut atîta succes este exact faptul că este compatibil cu C, şi deci putea folosi bibliotecile C, putea fi învăţat uşor de comunitatea programatorilor C, care deci puteau face tranziţia destul de uşor fără a trebui să-şi schimbe peste noapte obiceiurile. Ori asta nu e de loc adevărat în cazul ML, care a fost proiectat în acelaşi timp şi în mare parte în acelaşi loc [tot la Bell Labs], dar care a atacat problema într-un mod foarte radical. Din punct de vedere pragmatic C++ are un succes enorm, dar pentru care a plătit preţul compatibilităţii.

M: *Deci sunteţi avocatul evoluţiei incrementale.*
Văd ca sunteţi autorul a cel puţin opt cărţi, toate din ele scrise în colaborare. Deduc de aici că stilul dumneavoastră de cercetare este colaborativ?

K: Dacă ai de gînd să scrii o carte, e al naibii de uşor să mai găseşti pe altul să facă o parte din treabă [rîde]. Am fost foarte norocos să am colaboratori excelenţi la toate cărţile mele, şi în sensul ăsta a fost foarte uşor. E mult mai simplu de lucrat la o chestie mare, cum e o carte, şi care îţi ia şase luni sau un an, dacă ai pe cineva care lucrează cu tine. Asta te ţine şi cu picioarele pe pămînt: dacă o iei razna prea tare, ai pe cineva care poate să te cîrmească la loc pe direcţia cea bună.

Cred că tot ce am făcut am făcut cu altcineva: e mai amuzant să lucrezi cu alţi oameni decît să te încui în birou şi să te chinui de unul singur. Şi probabil că sunt mai bun la a găsi şi a asculta pe cineva care are o idee bună, pentru ca apoi să lucrez cu acea persoană la dezvoltarea ideii decît la a găsi ideile eu însumi.

M: *Dacă tot am vorbit de statul cu picioarele pe pămînt, lucrez la un proiect care dezvoltă nişte programe mari, unele din funcţii fiind modificate de mai mulţi autori, care schimbă fiecare stilul de aşezare în pagină după preferinţele proprii. Aţi publicat cîteva cărţi despre tehnici şi stil de programare; s-a potrivit întotdeauna stilul dumneavoastră cu al co-autorilor, sau aţi avut probleme în a reconcilia gusturile?*

K: [rîzînd] Asta e de asemenea o întrebare drăguţă. Ocazional am avut, nu "probleme", nu e cuvîntul potrivit, dar am avut discuţii despre unde se pun acoladele, unde se pun spaţiile şi cum se scriu numele de variabile. Cele mai multe din lucrurile astea au fost mărunte, în parte pentru că coautorii mei sunt prin preajmă şi am crescut în acelaşi mediu cultural. Dar, de exemplu, cînd Rob Pike şi cu mine lucram la "The Practice of Programming", cu cîţiva ani în urmă, am avut discuţii destul de intense despre lucruri minore cum ar fi "unde pui spaţiile". Tu cîte spaţii pui? Eu pun spaţii după cuvinte-cheie ca `if`, `while` şi `for`, dar Rob nu pune. Am cîştigat bătălia aia, dar sunt alte bătălii pe care le-am pierdut, nu mai ţin minte exact care. Cu siguranţă că nu am fost de acord 100%, dar am rezolvat amiabil disputele.

Cu cît mai mulţi oameni lucrează la ceva, şi cu cît e programul mai mare, cu atît mai greu o să fie, şi de la un moment dat trebuie să stabileşti standarde pe care toată lumea le respectă şi metode mecanice ca de exemplu pretty-printer-e care te obligă să joci după reguli, pentru că altfel pierzi prea mult timp şi creşte probabilitatea să faci greşeli.

M: *Aţi menţionat pretty-printer-ele; ce alte scule şi medii de programare folosiţi?*

K: Cînd am de ales programez în Unix. Folosesc editorul de texte `sam` scris de Rob Pike; nu folosesc `Emacs`. Cînd nu pot folosi `sam` folosesc `vi` din motive istorice, şi sunt chiar foarte mulţumit de `ed` [rîzînd]; ştiu, ăsta a apărut înainte ca tu să te naşti. Şi e parţial un motiv istoric: îl ştiam pe Bill Joy cînd lucra la `vi`.

Nu folosesc debugger-e sofisticate, folosesc instrucţiuni de tipărire inserate în cod; singurul lucru pentru care folosesc un debugger este să obţin o listă a procedurilor de pe stivă cînd un program moare în mod neaşteptat. Cînd scriu cod pe Windows în mod normal folosesc sculele Microsoft de dezvoltare: ele ştiu exact unde sunt fişierele header şi alte fişiere importante, deci trebuie să le folosesc chiar dacă nu se potrivesc cu stilul meu preferat de lucru. De asemenea în Windows îmi instalez pachete de scule cum ar fi `mks` care oferă utilitarele tipice Unix şi programele standard cu care sunt obişnuit pentru a căuta, compara, şi aşa mai departe.

M: *Voi schimba din nou subiectul. Cînd am venit în Statele Unite am fost extrem de surprins să descopăr că se face cercetare de foarte bună calitate, şi cercetare fundamentală — adică cercetare* *care nu e în mod necesar direcţionată spre un produs comercial sau pentru a genera profituri — nu doar în universităţi, ci şi în cîteva companii mari. Ce ne puteţi spune despre cercetarea la Lucent, o companie foarte mare, care e o parte din fostul AT&T, o companie uriaşă?*

Figura 3: Logo-ul Bell Labs şi Lucent

K: O să-ţi dau sloganul oficial, deşi cred că mult din el este în continuare adevărat. Cercetarea a fost o parte importantă a companiei ăsteia de pe cînd se chema "The Bell System", şi a rămas la AT&T şi Lucent; Bell Labs serbează 75 de ani anul acesta. Cred că cercetarea a început pentru că oamenii şi-au dat seama că sunt felurite lucruri pe care nu ştiu cum să le facă, dar pe care ar trebui să înveţe să le facă, dacă vor să îmbunătăţească produsele sau serviciile pe care le vindeau. Desigur, în timpuri ancestrale, serviciul de bază era cel telefonic; în urmă cu 30 sau 40 de ani tehnologia telefoniei a început să capete o din ce în ce mai puternică componentă informatică, şi asta a adus cercetarea în calculatoare aici. Cred că acelaşi lucru este adevărat pentru companii ca IBM, care are de asemenea laboratoare de cercetare extrem de eficiente. IBM e cu siguranţă o altă companie cu o tradiţie îndelungată de încurajare a cercetării.

Avem întrebarea interesantă "cum justifică o companie banii cheltuiţi pe cercetare?". Lucent la ora asta are cam 150 de mii de angajaţi, deci partea de cercetare, din care facem parte tu şi cu mine, e ceva mai puţin de 1%, poate 1000 sau 1500 de oameni. Veniturile anuale ale companiei sunt de 38 de miliarde de dolari în 1999, deci cheltuim cam 400 de milioane de dolari anual pe cercetare ca să ne ţinem pe tine şi pe mine în birouri comfortabile gîndind gînduri înălţătoare. Ăsta pare un fel destul de rezonabil de a investi, o investiţie cu risc mare, dar cu profituri potenţiale mari. Trebuie tot timpul să te gîndeşti "unde o să fim în cîţiva ani?", ce fel de probleme ne chinuie acum, la care încă nu cunoaştem soluţii, la care ar fi frumos să avem soluţii, chiar dacă nu ne trebuie astăzi, pentru că în viitor vom avea nevoie de ele. Din nefericire e greu să-ţi dai seama cum să faci astfel de lucruri, adesea e greu să-ţi dai seama care sunt problemele importante. Cred că cel mai bun mecanism pe care cineva l-a găsit pînă acum e să ia o mică sumă de bani, să zicem 1%, şi să angajeze o grupă de oameni brilianţi, să-i pună într-un mediu în care sunt încurajaţi să vorbească unul cu altul, să vorbească cu lumea din restul companiei, să afle ce fel de probleme are lumea din restul companiei; lumea din restul companiei este încurajată să vină la ei şi să zică "puteţi să ne ajutaţi cu problema asta?", şi să spere că acest proces aleator, şi este un proces în multe feluri absolut aleator...

M: *[întrerupînd] Dar nu numai atît: există cercetare-dezvoltare în multe alte companii; dar la Bell Labs cercetătorii sunt şi încurajaţi să publice!*

K: ...Cred că întrebarea este "cum diferă compania asta de alte companii prin faptul că publicăm?" Sunt mai multe lucruri la mijloc. Unul este că scara noastră este mult mai mare;

Lucent Bell Labs este probabil în continuare cel mai mare laborator de cercetare industrială din lume, făcînd cercetare de genul celei pe care o găseşti în universităţi în vremurile de demult, esenţialmente cercetare fundamentală, nedirecţionată pe produse. IBM este cel puţin comparabilă, şi Xerox într-o oarecare măsură, şi sunt şi alte companii de genul ăsta. Una dintre diferenţe este că aici, cel puţin în grupul de calculatoare, dar şi în toate ştiinţele fizice, cercetarea este undeva între cercetare industrială, puternic orientată pe produs, şi cercetare academică, în care fac treabă mai ales din curiozitate sau pentru că vor să analizeze problemele în mai mare adîncime. Un laborator industrial mare este întins între extremele astea: e mai focalizat pe lucrurile care sunt practice, dar în acelaşi timp are un deget în lumea universitară. Şi trebuie să facă asta, pe lîngă alte motive, din cauza mecanismului de angajări. Motivul pentru care tu eşti aici şi nu la Cisco, să zicem, este că Cisco nu face cercetare, Cisco cumpără companii. Cisco nu e un loc rău, în multe privinţe Cisco e un loc minunat, dar îşi face treaba altfel decît Lucent. Un alt lucru este că, fiind un jucător în lumea academică pe lîngă cea industrială, noi interacţionăm cu lumea din universităţi pe post de colegi egali în rang, deci putem atrage oameni ca tine, care vin aici pentru o vară şi după aia poate vin înapoi permanent. În felul ăsta avem un influx permanent de oameni buni. Dar trebuie să-i dăm înapoi ceva sistemului. Trebuie să te primim în companie, să-ţi arătăm lucruri interesante, să te lăsăm să publici articole, şi trebuie să scriem articole noi înşine, pentru că altfel nimeni nu o să ne creadă că facem lucruri interesante. Deci trebuie să fim în mare parte membri care participă la comunitatea academică, pe lîngă participarea la bunăstarea companiei. Asta e o problemă interesantă şi încă nerezolvată, cum să faci ambele lucruri fără să cazi prea tare într-o extremă sau cealaltă.

M: *L-aţi menţionat pe Rob Pike; vă este coautor la două cărţi. Aş vrea să vă întreb despre o prelegere foarte controversată pe care a ţinut-o, al cărei rezumat se găseşte la* `http://www.research.bell-labs.com/who/rob/utah2000.pdf`, *în care dînsul argumentează că cercetarea în partea de sisteme (aplicată) în informatică este moartă. Ce gîndiţi despre acest enunţ?*

K: Ca să fiu sincer, Rob are aproape întotdeauna dreptate, deşi nu aş recunoaşte asta în faţa lui [rîde]. Doar am văzut textul prezentării lui, fără să fi fost prezent la faţa locului, dar cred că în multe privinţe are dreptate. Observaţia lui este că este foarte greu de lucrat în sisteme: lucrurile se fac la scară prea mare pentru mediile academice, iar mecanismul recompenselor în mediile academice este defectuos. Ca rezultat o mare cantitate a muncii din sisteme tinde să fie incrementală, si să constea mai curînd în evaluarea performanţei decît în sinteza unor combinaţii noi şi interesante. Nu ştiu de ce lucrurile stau aşa: poate că e greu pentru universităţi să obţină suportul necesar, poate durează prea mult — observaţia lui Rob este că sistemele reale sunt construite în cinci ani sau mai mult, şi cam asta e durata unui doctorat — deci e greu să obţii ceva pe măsura carierei unui student. N-aş zice că cercetarea în sisteme este "moartă", dar cu siguranţă nu este atît de vie cît ar putea fi.

M: *Dacă tot am menţionat viaţa universitară, am văzut că aţi predat cel puţin două cursuri la Princeton. Aş vrea să vă cer opinia despre educaţia în informatică. Am auzit plîngeri din partea companiilor că studenţii din calculatoare învaţă prea multă teorie inutilă şi nu ştiu destul despre dezvoltarea de programe reale.*

K: Am predat patru cursuri la Princeton şi Harvard în ultimii patru sau cinci ani, la nivele felurite, dar asta nu e suficient pentru a face din mine un expert în educaţia în informatică. Astea sunt nişte şcoli anume şi eu am predat mai mult ciudăţenii. Nu cred că universităţile ar trebui să predea lucruri pe care trebuie să le înveţi la o şcoală profesională; nu cred că rolul universităţii este să predea utilizarea lui, să zicem, Visual C++ şi a mediului său integrat de dezvoltare. Cred că rolul universităţii este să înveţe studenţii cum să programeze într-un anumit tip de limbaje, de exemplu orientate pe obiecte, să ajute studenţii să înţeleagă meritele şi dezavantajele unei familii de limbaje ca C, C++ şi Java, şi relaţia acestora vis-a-vis de limbaje de natură diferită, cum ar fi cele funcţionale. Să înveţi studenţii tehnicile de care au nevoie ca atunci cînd intră într-un atelier Windows de software să fie capabili să scrie imediat programe COM nu e corect. Nu asta trebuie să facă universităţile, ele trebuie să predea lucruri perene, care vor ţine pentru o viaţă, dacă eşti norocos, sau măcar pentru 5, 10 sau 20 de ani, şi acestea sunt principii şi idei. În acelaşi timp, ar trebui să ilustreze principiile şi ideile cu cele mai bune exemple posibile luate din practica curentă.

La Princeton am predat un curs introductiv, o combinaţie de software engineering şi programare avansată: studenţii, mai ales cei din anii mari, erau în general foarte experimentaţi în felul de lucruri de care probabil că industria are nevoie. Ei erau familiari cu Visual C++, ştiau cum să ia componente de pe reţea şi să le asambleze, şi puteau scrie aplicaţii Java destul de sofisticate. Multe din lucrurile astea erau poate învăţate din slujbe de vară. Dacă industria vrea oameni care au mai mult decît cunoştinţe "inutile" teoretice, ce ar trebui să facă este să ia aceşti copii talentaţi din şcoli şi să le ofere slujbe de vară interesante, care complementează cunoştinţele teoretice cu detaliile particulare ale diferitelor aplicaţii. Lumea învaţă astfel de lucruri extrem de repede, şi dacă studenţii învaţă lucruri interesante la slujbele de vară vor duce astfel de cunoştinţe şi în carierele lor academice. Am fost foarte impresionat de cîte lucruri ştiau studenţii, pe care nu le învăţaseră neapărat la şcoală.

M: *Vorbind despre studenţi, ce sfaturi i-aţi da unui student care intenţionează să urmeze o carieră de cercetare? Poate unele domenii sunt mai fructuoase decît altele, sau poate unele au devenit neinteresante?*

K: Nu urma sfaturile mele despre carieră [rîde]. Din nefericire nu cred că există nici un fel de sfat bun. Cele mai interesante, pardon, nu ar trebui să zic "interesante" — domeniile cele mai dificile sunt doar două: unu, că e prea greu să scrii programe care merg, şi doi, că e prea greu să foloseşti calculatoarele. Deci dacă vrei teme asupra cărora să te apleci, astea sunt două pe care le poţi încerca. Desigur, astea sunt destul de generale, [rîde] există o mulţime de cazuri particulare cu care te poţi juca. Dacă faci progrese în orice direcţie din domeniile astea, atunci ai oportunitatea de a continua cu o carieră academică sau poţi încerca să faci avere într-o companie nou creată (start-up). La ora asta se pare că o mulţime de oameni mai curînd ar face avere într-nouă companie decît să petreacă 5 sau 6 ani pentru a obţine un doctorat. Poate eşti prost orientat [rîde]...Cred că din nefericire cel mai bun sfat pe care i-l poţi da cuiva este "fă ceea ce crezi că este interesant, fă ceva care crezi că este amuzant şi merită, pentru că altfel oricum nu o să poţi să o faci bine". Dar asta nu ajută de loc.

M: *Poate puteţi ajuta fiind mai concret: puteţi să ne recomandaţi nişte cărţi, de calculatoare sau altceva, care credeţi că au avut o influenţă mare asupra dumneavoastră?*

K: Singura carte de calculatoare pe care am recitit-o mai mult de odată, pe care de fapt o iau la fiecare cîţiva ani şi din care recitesc părţi este "The Mythical Man-Month" (Miticul om-lună), de Fred Brooks, o carte mare. În parte este foarte bine scrisă, în parte sfaturile din ea, chiar după 25 de ani, sunt foarte relevante. Sunt desigur detalii care s-au schimbat, unele lucruri le facem altfel din cauză că avem mai multă mecanizare şi mai multă putere de calcul, dar este o cantitate enormă de sfaturi utile în cartea aia, aşa că o recomand cu mare căldură. Este singura carte de calculatoare la care mă pot gîndi pe care o reciteşti pentru o combinaţie de plăcere şi viziune.

Sunt alte cărţi pe care le recitesc şi care sunt foarte relevante pentru oameni din informatică. Cărţi despre cum se scrie, în engleză în cazul meu particular, ca "Elements of Style" (elemente de stil) de Strunk şi White. Mă reîntorc şi recitesc cartea asta la fiecare cîţiva ani, pentru că cred că abilitatea de a comunica este probabil la fel de importantă ca cea de a sta la tastatură şi a scrie programe. Capacitatea de a explica ceea ce faci este foarte importantă.

Există de asemenea o carte grozavă, "Cum să minţi folosind statistica", pe care o vei găsi utilă în propria ta cercetare [rîsete].

M: *Voi schimba din nou direcţia. Unix şi C au fost create la AT&T şi la ora aceea au fost lansate sub o licenţă care era virtual o licenţă open-source, pentru că AT&T trebuia să facă asta: fiind un monopol avea o înţelegere cu guvernul american, din cîte înţeleg, să nu facă bani din calculatoare. O mulţime de oameni afirmă că licenţa liberală este motivul enormei popularităţi şi influenţe pe care atît Unix şi C le-au avut. Recent Lucent a lansat Plan 9 sub o licenţă open-source. Ce credeţi despre acest "nou" fenomen open-source[1]?*

K: Cred că în mare parte e un lucru bun. Licenţa originară Unix era, cum zici, făcută în mare parte în acest fel pentru că AT&T nu avea voie să facă orice fel de alte afaceri decît telefonie, deci nu puteau face bani pe software. Din cauza asta au luat decizia foarte înţeleaptă de a da Unix practic pe gratis universităţilor. L-au vîndut şi comercial, dar pe un preţ de nimica, dar universităţile l-au primit pe degeaba, şi ca atare o întreagă generaţie de studenţi şi profesorii lor s-a dezvoltat crezînd că Unix este felul în care se fac calculatoare. Unix era cu siguranţă mult mai productiv decît celelalte sisteme de operare comerciale ale timpului, şi pentru că codul sursă era disponibil era uşor să-ţi dai seama ce se întîmplă şi să faci îmbunătăţiri. În jurul Unix-ului s-a construit o comunitate de utilizatori, interesaţi de Unix, motivaţi de aceleaşi lucruri şi care contribuiau şi în felul acesta sporeau valoarea sistemului prin colaborare.

Cred că mişcarea curentă open-source are foarte mult din acest caracter. Multe din sculele dezvoltate open-source sunt bazate pe modele Unix. Linux este cel mai evident, pe dinafară cel puţin fiind clar bazat pe Unix; multe din programele de la Free Software Foundation sunt reimplementări ale unor scule şi limbaje standard din Unix.

Sunt desigur şi alte proiecte care apar din interese comerciale stranii, cum ar fi Mozilla, codul pentru Netscape, care e acum în domeniul public, şi la care lumea de asemenea contribuie.

Cred că fenomenul open-source este în general un lucru bun. Nu sunt sigur că va înlocui vreodată cod specializat, profesional, solid, care este vîndut pentru profit. Dar ce ar putea realiza în multe cazuri, şi cred că reuşeşte perfect în cazul compilatoarelor de C, este să ofere o implementare de referinţă şi un standard de o calitate bună, pe care toate celelalte implementări trebuie să-l atingă, pentru că altfel nu le cumpără nimeni. Cred că în sensul ăsta e un lucru bun.

Cît despre Plan 9, cred că e prea tîrziu, din păcate. Cred că Plan 9 a fost o idee măreaţă şi ar fi trebuit lansat sub o licenţă gen open-source cînd a fost făcut, cu opt ani în urmă, dar "tutorii" noştri nu ne-au lăsat. Cred că au făcut o greşeală foarte costisitoare. Licenţa actuală gen open-source merită categoric, dar nu e clar dacă Plan 9, cel puţin ca sistem de operare general,

[1] Despre open-source vedeţi şi articolul meu "Noua revoluţie în software", din PC Report din iunie 1998, disponibil şi în pagina mea de web.

va avea impact, exceptînd o mică nişă. Are multe calităţi care-l fac valoros în multe domenii, mai ales unde ai nevoie de un sistem mic şi portabil, dar va reuşi să smulgă dominaţia Linux-ului? Nu cred.

M: *Mă pregătesc să închei pe o notă mai uşoară, dar întîi voi pune încă o întrebare mai adîncă. Interpolînd din evoluţia informaticii de pînă acum, cum vedeţi evoluţia sa în viitorul apropiat, şi nu îndrăznesc să întreb, cel îndepărtat?*

K: [rîde] Dacă aş putea prezice viitorul atunci aş investi mai înţelept şi nu ar trebui să fac interviurile astea prost plătite [n.b. interviul a fost gratuit]. Ştii, de fapt sunt aşa de nepriceput la preziceri. . . . Voi ghici că într-un sens situaţia din informatică va fi aceeaşi: vom face multe progrese, vom putea face lucruri mai mari, vom putea construi lucruri care sunt mult mai interesante şi sofisticate decît cele pe care le puteam face cu zece ani în urmă. Dacă te uiţi la lucrurile care rulează pe PC-ul din faţă noastră, sunt mult mai puternice şi mai flexibile decît cu zece ani în urmă. Dar cantitatea de cod încîlcit, peticit, oribil care zace sub toate astea a crescut şi ea enorm. Într-un sens cred că vom continua să facem progrese, dar vor fi tot timpul slinoase şi pe jumătate neterminate. Pentru că oamenii mereu se apucă de mai mult decît pot face în mod rezonabil, tot timpul se întind mai mult decît plapuma, şi niciodată nu par să se întoarcă să cureţe ce au făcut înainte.

Celălalt lucru care mă cam îngrijorează este că acum calculatoarele sunt peste tot, în tot ce faci. Nu poţi să te întorci fără să ai de-a face cu ceva care depinde în mod critic de calculatoare, şi din ce în ce mai multe din lucrurile astea contează. Nu contează dacă programul de înregistrat sunet din Windows merge sau nu. . .

M: *[întrerupînd] Acum contează! [n.b. interviul acesta a fost făcut folosind un PC şi software de captare şi compresie de sunet]*

K: . . . dacă pierdem interviul, la naiba cu el. Dar, de exemplu, cînd o să zbori înapoi la Pittsburgh, tu îţi doreşti din toată inima să meargă bine calculatoarele care controlează echipamentele de pe Airbus 320, şi să nu facă nici o greşeală; ăsta este doar un exemplu dintre lucrurile care depind critic de abilitatea noastră de a scrie software care merge şi să construim hardware care-l suportă, şi cred că încă nu ştim să facem asta bine. Facem progrese, e clar mai bine, combinaţia de limbaje şi tehnici ca sculele de verificare ajută, dar avem aceeaşi problemă: pe măsură ce înţelegem mai bine cum să facem lucrurile mici ne apucăm de unele mai mari şi mai complicate şi într-un sens suntem mereu între crocodili pînă la gît.

M: *Ultima mea întrebare este despre hobby-uri.*

K: [rîzînd] Cine are timp de hobby-uri? Dacă mă uit la hobby-urile mele din clipa asta, cred că se rezumă la citit. Cînd nu mă distrez sau meşteresc la calculatoare, sau nu fac chestii legate de slujba mea, citesc. De obicei cărţi de istorie, nu ştiu de ce, e ciudat, dar îmi place să citesc cărţi de istorie. De-a lungul vieţii mele, în ultimii 20 sau 30 de ani, am trecut prin tot felul de faze lungi de 3, 4, 5 ani in care m-am cufundat adînc în cîte ceva. Am trecut printr-o fază în care am încercat să învăţ japoneza, şi pot să-ţi zic, îţi trebuie mai mult de 3 ani ca să înveţi japoneza! Deci ăsta a fost un eşec. Cînd eram student doctorand am petrecut cam cinci ani studiind karate, şi am ajuns la un nivel în care puteam aproximativ supravieţui, dar am renunţat, şi nu mai e parte din viaţa mea. Apoi am avut o fază în care eram foarte interesat de investiţii [la bursă], dar nici asta nu mi-a schimbat viaţa, deci evident nu eram prea bun. Deci lucrul pe care îl fac acum este să citesc mult.

M: *Vă mulţumesc foarte mult!*

4 Cărţile lui Brian Kernighan

Pagina de web a lui Brian Kernighan: `http://cm.bell-labs.com/who/bwk`

- R. Fourer, D. M. Gay, and B. W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, Duxbury Press, 1993, ISBN 0534509835

- B. W. Kernighan and P. J. Plauger, The Elements of Programming Style, McGraw-Hill, second edition 1988, ISBN 0070342075.

- B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, first edition 1978, ISBN 0131101633; second edition 1988, ISBN 0131103628.

- B. W. Kernighan and P. J. Plauger, Software Tools, Addison-Wesley, 1976, ISBN 020103669X.

- B. W. Kernighan and P. J. Plauger, Software Tools in Pascal, Addison-Wesley, 1981, ISBN 0201103427.

- B. W. Kernighan and R. Pike, The Unix Programming Environment, Prentice-Hall, 1984, ISBN 0139376992.

- B. W. Kernighan and R. Pike, The Practice of Programming, Addison-Wesley, 1999, ISBN 0-201-61586-X.

- A. V. Aho, B. W. Kernighan, and P. J. Weinberger, The AWK Programming Language, Addison-Wesley, 1988, ISBN 020107981X.