OŚRODEK BADAWCZO-ROZWOJOWY INFORMATYKI

POSTĘP W TELEKOMUNIKACJI PRZENOŚNOŚĆ I WYMIENNOŚĆ CZĘŚĆ 2

Europejski Program Badawczy Diebolda

84

Warszawa 1977

OŚRODEK BADAWCZO-ROZWOJOWY INFORMATYKI

POSTĘP W TELEKOMUNIKACJI PRZENOŚNOŚĆ I WYMIENNOŚĆ CZĘŚĆ 2

Europejski Program Badawczy Diebolda

Wyłącznie do użytku na terenie PRL

84

Warszawa 1977

Tytuł oryginału: Advances in Telecommunication's Portability and Compatability.
Conference Proceedings Meeting XXXVI
Munich, 9-11 March 1976
Document No. EC 36

Tłumaczenie: Aleksander Małecki
Redakcja: Ryszard Terebus

Komitet Redakcyjny

Andrzej Idźkiewicz, Janina Jerzykowska /sekretarz/, Stanisław Nelken, Wacław Pankiewicz, Witold Staniszkis, Ryszard Terebus /przewodniczący/

Wydawca

OBRI - Dział Wydawnictw, 02-021 Warszawa, ul. Grójecka 17

Warszawa 1977. Nakład: 900+100 egz. Objętość: ark. wyd.16,5; ark. druk. 40. Format A4. Papier offsetowy kl.III,80g, 61x86

zam.27/77 DN.444-13/73 Cena zł 92,-

SPIS TREŚCI

SESJA

F TELEKOMUNIKACJA W PRZETWARZANIU ROZPROSZONYM
J.F.David, IBM, Francja 5

G SYTUACJA W DZIEDZINIE SYSTEMÓW PRZENOŚNYCH
Alan Duncan, Barclays Bank, Londyn 40

H KRYTERIA WYMIENNOŚCI SYSTEMÓW PRZEZNACZONYCH DLA WIELU UŻYTKOWNIKÓW
M.F.Wolters, Siemens AG, RFN 63

I PRAKTYCZNE ZAGADNIENIA NORMALIZACJI JĘZYKÓW
M.V.Vickers, Krajowe Biuro Normalizacyjne, Stany Zjednoczone 87

K KLASYFIKACJA TECHNIK PROGRAMOWANIA
Zespół dyskutantów koordynowany przez przedstawiciela Diebolda w RFN 112

Ł ADAPTACJA UNIWERSALNYCH SYSTEMÓW DO KONKRETNYCH ZASTOSOWAŃ
J.Peers, J.Peers Inc., Stany Zjednoczone 144

SESJA F

TELEKOMUNIKACJA W PRZETWARZANIU ROZPROSZONYM

J.F.David, IBM, Francja

O dziedzinie tej mówiono już bardzo wiele na poprzednich konferencjach. Na przeźroczu F1 wyszczególniono pięć tytułów. Będę mówił raczej o technice, ale nie tyle w odniesieniu do samej telekomunikacji, ile w odniesieniu do problemów związanych z komputerami znajdującymi się na końcach linii, to znaczy o szybkości wyszukiwania informacji w porównaniu z potrzebami telekomunikacji, gdyż problemy przechowywania i wyszukiwania są bardzo często czynnikami limitującymi. Mówiąc o przetwarzaniu rozproszonym, musimy też mieć na uwadze zagadnienia ekonomiczne; odnosi się to także do producentów. Ceny w dziedzinie telekomunikacji spadają i jeśli chcemy dotrzymać kroku tej tendencji, to samo powinno nastąpić w dziedzinie przetwarzania danych.

Dziś jest tak wiele wąskich gardeł, że gdy chcemy zgranego lub harmonijnego rozwoju, musimy coś zrobić, by znaleźć nowe typy architektury systemów.

Mówiono też o oprogramowaniu. Nie wiem, czy powiedziano coś o IBM-owskim systemie operacyjnym sieci lub co można zrobić w sprawie złożoności, ale wiążą się z tym pewne istotne problemy. Oczywiście jeszcze większe są problemy w dziedzinie zdalnego przetwarzania. Musimy unikać tych problemów złożoności i starać się upraszczać sprawy. Oszczędności wynikające z wielkiej skali możemy uzyskać tylko pod warunkiem wyeliminowania złożoności.

I wreszcie, musimy pamiętać, że w "pętli" przetwarzania danych uczestniczą też ludzie, a więc i ich trzeba wziąć pod uwagę.

Wiemy, że gdy komputer ulega awarii, musimy znaleźć jakiś sposób utrzymania operacji w ruchu. Musimy wówczas uzyskać dostęp do urządzeń, które przedtem nie były nam dostępne, tzn. przyłączyć się gdzie indziej; ale jest to równocześnie sprawa bezpieczeństwa. Oto więc jeszcze jeden punkt, o którym powinienem wspomnieć.

Na początku pomówmy o technice.Mogę to zrobić bardzo krótko, poruszając sprawę pojemności pamięci i czasów przełączania. Są w tej dziedzinie pewne granice, których nie da się przekroczyć nawet w długoterminowej perspektywie. Jeśli chodzi o przepustowość łącznic,możliwa jest przepustowość setki razy większa od dzisiejszej. W komputerze zwykłego typu czas przełączania wynosi 0,1 nanosekundy. Pięć pikosekund stanowi może ostateczną granicę, ponieważ jest to kwestia efektu Wilsona oraz pewnych wielkości i zjawisk fizycznych. Nie wiem więc, czy potrafimy kiedykolwiek uzyskać lepsze rezultaty w dziedzinie czasów przełączania.

Ale nieco inaczej przedstawia się sprawa, jeśli chodzi o czasy wyszukiwania, choć czasy przełączania narzucają im pewne granice. Jest to także kwestia miniaturyzacji - miniaturyzacji w stosunku do tego, co jest dziś wykonalne. Taka miniaturyzacja wymaga jednak bardzo specjalnych technik,będących obecnie przedmiotem badań w laboratoriach wszystkich wielkich producentów. Uzyskanie 200-krotnej poprawy w zakresie pojemności pamięci on-line pozwoliłoby wkroczyć w dziedzinę informacji niekodowanej, tzn. transmisji tekstów lub obrazów. Dziś granicę stanowi pojemność 400 miliardów bajtów on-line,ale żeby przejść do przechowywania informacji w postaci obrazów, musimy mieć pamięci o pojemności około 100 razy większej. Niektóre techniki są obiecujące, ale wciąż jeszcze napotykamy na wąskie gardła, mimo bardzo szybkiego postępu techniki, szczególnie jeśli chodzi o urządzenia końcowe.

Przed trzema lub czterema laty podstawowym urządzeniem końcowym było nieinteligentne urządzenie końcowe z ekranem, nie mające żadnej mocy obliczeniowej ani zdolności do przechowywania danych.Dziś dość powszechnie spotyka się urządzenia końcowe

z pamięcią o pojemności 4 K lub 8 K i zaczynają się pojawiać generacje urządzeń końcowych z pamięcią 100 K lub nawet więcej. Bardzo rychło będziemy mieli urządzenia końcowe z jeszcze większą pojemnością pamięci. A zatem pod tym względem postęp jest bardzo szybki.

Ale jest jeszcze jeden problem, wobec którego stają teraz użytkownicy, a mianowicie problem zachowania ich systemów użytkowych. Chodzi o to, że niejeden użytkownik opracował szeroki wachlarz systemów użytkowych, ale być może zrobił to w czasie, gdy urządzenia końcowe nie były inteligentne. Dziś mamy urządzenia końcowe z pamięcią 4 K, jutro może to będzie 100 K. Co - w sensie urządzeń końcowych i nakładu pracy ludzkiej - pozostanie z tego wszystkiego co poprzednio zaprojektowano? Oczywiście urządzenie końcowe będzie można bardzo szybko przenieść gdzie indziej. Ale jak sprawić, by sam system użytkowy został zachowany? Nie jest to prosty problem.

Problem ten jest już dziś aktualny w przypadku systemów centralnych obejmujących zbiory danych. Oznacza to, że oprogramowanie idzie szybkimi krokami naprzód, tak samo jak sprzęt. Jedno i drugie szybko się rozwija. Jedno pociąga za sobą drugie; jest to szeroko rozpowszechnione zjawisko. Zmiany w sprzęcie bardzo często pociągają za sobą też ilościowe i jakościowe zmiany w oprogramowaniu. Ale w systemach centralnych istnieje tendencja do stosowania systemów baz danych i transmisji danych /data base and data communications - DBDC/, co być może da tym systemom możliwość przetrwania zmian technicznych w odniesieniu do przechowywania danych.

Technika DBDC pozwala systemom użytkowym wywoływać dane niezależnie od tego, czy mamy okrągły dysk, kwadratowy dysk, pamięć rdzeniową czy też jakąkolwiek inną. Ale problem teleprzetwarzania jest znacznie szerszy, toteż jeśli chcemy, by system użytkowy przetrwał, musimy uwzględnić fakt, że będzie on w przyszłości częściowo zdecentralizowany.

Inną dobrze znaną okolicznością, która doprowadziła do wdrożenia systemu baz danych w centrum, było to, że około roku 1965 można było opracować bardzo piękny system użytkowy A1, po-

tem system A2, a potem A3 i można było złożyć z nich bardzo piękny zintegrowany system. Szczególnie zwracam na to uwagę, ponieważ wchodzi to w grę w przypadku terytorialnie rozłożonych systemów użytkowych. Jak wiecie, z chwilą gdy opracowaliśmy A2, zmianie uległy interface'y A1, z chwilą zaś,gdy opracowałem A3, A1 stało się zupełnie inne,a A2 jest niewiele użytkowane. Jeśli nie opracujemy od razu na początku organizacji do aktualizowania interface'ów, będziemy musieli za każdym razem gdy tworzymy nową funkcję lub opracowujemy nowy system użytkowy, poświęcać masę czasu na sprzężenie zwrotne w stosunku do dawniejszych systemów użytkowych.A po jakimś czasie zahamowany zostanie wszelki rozwój.

Poruszyłem ten problem, bo taki sam typ problemu spotykamy w sensie terytorialnym lub geograficznym. Gdy macie zakłady rozrzucone w różnych miejscach, powstaje problem organizacyjny, czy dać każdemu z nich możliwość rozwijania się w swoim małym kąciku przy zachowaniu jedynie pewnych reguł komunikowania się na właściwie zorganizowanych zasadach, czy też znaleźć system, który będzie automatycznie utrzymywać te reguły. Od rozwiązania tego problemu będzie zależał wybór typu minikomputerów lub urządzeń końcowych.

Ale powróćmy do tematu. Co się dzieje w przypadku systemów centralnych, jednostek centralnych? Nie możemy stale aktualizować interface'ów, bo jest to zbyt kosztowne,wobec tego tworzymy zbiór spójnych danych, a systemy użytkowe będą, że tak powiem, obracać się wokół niego. Przyjrzyjmy się, co się dzieje, gdy mamy do czynienia z terytorialnie rozrzuconą organizacją.

Mamy sprzęt w centralnym systemie liczącym, mamy oprogramowanie, mamy interface na szczeblu bazy danych oraz mamy nasze systemy transmisji danych. Ale wszystko to jeszcze nie wystarczy, bo zdajemy sobie sprawę, że wszystko to stanowi tylko narzędzie do wytwarzania półproduktu. Co ono produkuje? Daje nam tabulogramy i materiały wyjściowe na ekranie monitora,ale użytkownikowi potrzebne jest narzędzie decyzyjne, niezależnie od tego czy będzie to decyzja o podjęciu części zamiennej z magazynu, gdy wszystkie dane wskazują, czy mam ją podjąć z magazynu

czy nie; czy też będzie to decyzja o zmianie polityki marketingowej przedsiębiorstwa. Obecnie pojawia się na rynku szereg dodatkowych systemów, które wychodząc od tych półproduktów dają produkt bardziej skończony. Wspomniałem o IBM-ie, ale jestem pewien, że jest szereg innych systemów, które spełniają zadanie operowania danymi wyjściowymi wspomnianego poprzednio systemu informacyjnego. Mówię o tym dlatego, że fakt ten musimy uwzględniać przy opracowywaniu urządzeń końcowych, to znaczy, powinniśmy dążyć do zaopatrzenia użytkownika w narzędzia tego właśnie typu.

Powracając do sprawy rozproszenia terytorialnego, o jakim przed chwilą wspomniałem, mamy w tych różnych punktach problem: jak stwierdzić, które rozwiązanie byłoby najlepsze. Pierwszym rozwiązaniem byłoby w ogóle zrezygnować z łączy, to znaczy powiedzieć sobie: "trzeba unikać komplikacji. Potrzebny mi jest pewien system. Ale gdybym miał zająć się tym wszystkim, byłby on zbyt skomplikowany. Idealnym rozwiązaniem będzie więc system bez wszelkich łączy, system mogący rozwijać się bez łączności." Przy takim rozwiązaniu wyłaniają się dwa główne problemy, a mianowicie: w izolowanym punkcie nie będzie można uzyskać dostępu do wszystkich danych firmowych, w chwili gdy będą one temu punktowi potrzebne do pracy w danym dniu. Dlatego, gdy w punkcie tym trzeba będzie podjąć decyzję, będzie ona prawidłowa, powiedzmy, w 80 procentach, ponieważ dana osoba ma do swej dyspozycji tylko 80% danych; ale decyzja mogłaby być lepsza, gdyby ta osoba miała dostęp do pozostałych 20% i tym samym mogła uwzględnić wszystkie rozporządzalne w przedsiębiorstwie dane.

To jest pierwszy aspekt, ale sądzę, że jest on mniej ważny. Głównym problemem jest spójność danych. Jest to ten sam problem, z którym spotkaliśmy się już wcześniej i który skłonił nas do utworzenia baz danych dla systemów scentralizowanych. W istocie rzeczy jest to problem stopniowego narastania niespójności danych oraz trudności aktualizowania interfaceów i reguł komunikacji w zorganizowany sposób.

Załóżmy, że mamy komputer z jego kompilatorem. W pierwszym roku wszystko jest w porządku, w drugim roku pojawi się pewna

ilość danych lokalnych, które rozwiną się w kierunku innym niż pozostałe. Gdy przyjdzie czas notyfikowania tych danych z jednego punktu do drugiego, nagle stwierdzimy, że reguły w punkcie nadawczym i w punkcie odbiorczym nie są już jednakowe. Dlatego punkt odbiorczy mówi: "nie, to nie to samo.Nie mamy tych samych mierników". Jeden człowiek może np. powiedzieć: "potrzeba mi 54 bajtów", a na to drugi: "nie- według mnie, potrzeba ci 68 bajtów". A stwierdziliśmy poprzednio, że koszt organizacyjny aktualizacji interface'ów jest niemały.

Co robi się więc? Coś co nazywa się teoretycznie "rozproszoną bazą danych". Co to jest rozproszona baza danych? Jest to techniczna możliwość automatycznego doskonalenia systemu.Lepiej może będzie, jeśli podam przykład,bo to najlepszy sposób objaśnienia. Na przykład mówię, że w centralnym punkcie potrzebny nam jest pewien typ danych w pewnej porze dnia. Zostaje to ustalone na początku. W innym punkcie potrzebny mi jest inny typ informacji lub danych i muszą one być rozporządzalne co do minuty lub sekundy. Zadaniem systemu będzie automatyczna wymiana danych z okresowością niezbędną, by koncepcja bazy danych mogła być jednolicie zastosowana w całym systemie.

W tym celu zakładamy,że na naszych liniach komunikacyjnych, niezależnie od ich typu, wymieniamy istniejące tam dane dla zapewnienia ich spójności, nawet gdy to nie jest bezpośrednio związane z ich użytkowaniem.Musimy aktualizować zbiory centralne i lokalne. Obecnie nie opłaca się przesuwać tych danych tylko po to, by zapewnić ich spójność.

Wiecie, że IBM zajmuje się m.in. telekomunikacją. Trzecie, pośrednie, rozwiązanie polega na tym, że powiadamy: "skoro nie możemy utrzymywać spójności metodami automatycznymi, zachowajmy scentralizowaną kontrolę nad spójnością danych".Nie przeszkodzi to nam w razie potrzeby rozproszyć mocy obliczeniowej, personelu i pojemności pamięci. Metodę tę można też bardzo dobrze pogodzić z dzisiejszą techniką, tzn. z dzisiejszą architekturą sieci.

Chcąc dokonać wyboru między danymi scentralizowanymi i niescentralizowanymi, trzeba pamiętać, że nawet gdy wybierzemy

rozwiązanie scentralizowane, nacisk w przedsiębiorstwie będzie tak silny,że już po dwóch czy trzech latach ludzie będą domagać się stosunkowo niezależnych narzędzi i urządzeń dla wykonywania swej pracy. Jest to też kwestia czasu wykonania operacji. Nawet w bardzo szybkich systemach zawsze znajdzie się ktoś,kto uważa, że nie dość szybko otrzymuje odpowiedź; czas odpowiedzi będzie dla niego za długi i będzie wolał mieć swoje własne specjalizowane urządzenia.

Oczywiście pierwszym pytaniem,jakie trzeba sobie postawić, to czy zadowalająca jest struktura manipulowania danymi i struktura informacji. Jeśli chodzi o informację, trzeba się zastanowić, czy informacja,jaką chcemy rozproszyć, ma charakter tymczasowy czy trwały. Jaki jest okres jej użyteczności? Czy zniknie po tygodniu, czy też będzie zachowana? Jeśli ma być zachowana, idealnym rozwiązaniem byłoby jej przechowanie gdzieś w terenie. Czy otrzymamy ją bezpośrednio w wyniku zbierania danych,czy też będzie ona rezultatem pewnego uprzedniego przetwarzania danych? Czy jest niezależna,czy powiązana z czymś innym? Czy jest to informacja "sama przez się" czy też wymaga licznej innej informacji, by nabrać jakiegoś znaczenia? Jeśli tak, będziemy musieli wdrożyć albo lokalnie albo centralnie jakiś system typu bazy danych, bo inaczej nie będziemy w stanie posługiwać się tą informacją.

To samo można powiedzieć o mocy obliczeniowej. Czy programy, które zamierzamy napisać, będą specjalne, czy też przeznaczone do szerokiego użytkowania? Jak długo dany program będzie zdolny do życia? I tak dalej. A potem, jaki będzie wpływ głównych tendencji rozwojowych informatyki? Tutaj /ilustr.F2/ widzicie Państwo przykład scentralizowanego sposobu przetwarzania danych. Te strzałki wskazują, że wszystko wchodzi do bazy danych,a nic z niej nie wychodzi.Powinienem był narysować strzałki w obu kierunkach. W każdej z komórek,na jakie podzielone jest przedsiębiorstwo,dolna część oznacza operacje produkcyjne; środkowa - kierownictwo operacyjne; górna - kierownictwo funkcjonalne. Mamy różne komórki. Można sobie bardzo łatwo wyobrazić,że komórki te mogłyby być kierowane za pomocą przetwarzania

danych,przy użyciu systemów użytkowych "zakotwiczonych" w bazie danych, aby zapobiec kolizjom między poszczególnymi systemami i umożliwić ich spójny rozwój; pozwoliłoby to też uniknąć tego rodzaju sprzężenia zwrotnego, o jakim przed chwilą wspomniałem.

Rozwiązanie takie ma oczywiście pewne dobrze znane, nieodłączne od niego wady, chociażby problemy niezawodności, np. w przypadku strajku lub awarii centralnego systemu liczącego.Poza tym istnieje zagadnienie czasu odpowiedzi, a także poważniejszy problem, że w lokalnej jednostce można wykonywać tylko bardzo podstawowe zadania, 2 + 2, np. sprawdzanie czy chodzi o obszar cyfrowy czy alfabetyczny.Inne zadania muszą powracać do systemu centralnego.

Zalety tego rozwiązania są bardzo wyraźne.Przede wszystkim rozwój jest spójny. Niektóre przedsiębiorstwa bardzo zdecydowanie weszły na drogę tego typu przetwarzania danych i dziś są one wysoce zintegrowane przy scentralizowanym podejściu do przetwarzania danych.

Tutaj natomiast /ilustr. F3/ poszliśmy w odwrotnym kierunku, to znaczy stosujemy informatykę oddzielnie w poszczególnych komórkach przy użyciu pewnego rodzaju systemu, który będzie załatwiał wszystkie problemy danej komórki.Oczywiście jednak znowu powracamy tu do problemu spójności w większym jeszcze nawet stopniu niż poprzednio; innymi słowy, gdy tylko mamy w jakimś przedsiębiorstwie wysokie nasilenie komunikacji - bo mamy tam przecież nie tylko produkcję, ale i komunikację - wyłaniają się problemy interface'ów i bardzo wysoki jest koszt organizacji. Oczywiście możliwy jest pewien rozwój; ustanawiamy pewne reguły i normy; nie można już postępować jak kto chce,trzeba natomiast przestrzegać pewnych norm, ale wówczas zatracają się zalety lokalnego przetwarzania danych.

Oczywiście korzyści są wyraźne. Każdy manipuluje swoimi własnymi danymi, a urządzenia i systemy lepiej mu odpowiadają, bo może je przystosować do własnych potrzeb. Komputer staje się czymś bliskim dla użytkujących go ludzi.

Architektura sieci przedstawiona na ilustracji F4 jest syntezą wszystkich tych różnych podejść. Zobaczmy, w jakim stopniu

jest to naprawdę synteza. Koncepcja polega na tym, by umożliwić równoczesne przyjęcie obu rozwiązań, to znaczy mieć stale te bardzo silne powiązania, które przedstawiłem liniami ciągłymi, a także powiązania potencjalne, przedstawione liniami kreskowymi. Linie takie istnieją, ale są użytkowane tylko od czasu do czasu. Może to być np. praca typu zdalnego wprowadzania zadań w godzinach porannych lub wieczornych; możliwe są też przypadki, gdy w ogóle nie ma połączeń. A zatem teoretycznie możemy osiągnąć syntezę.

Dlaczego więc dotąd nie zrobiono tego? Z tej prostej przyczyny, że są trudności z architekturą systemów. Architektura systemu, która umożliwiłaby jego działanie, bynajmniej nie jest prosta w tym przypadku. Powrócę do tego nieco później.

Jakie kryteria można by przyjąć dla porównywania tych różnych architektur? Może to być stopień uporządkowania informacji. A co z redundancją? W pierwszym przypadku redundancja będzie bardzo niewielka; będzie istniała, ale tylko w małym zakresie. Natomiast w drugim przypadku mamy wysoką redundancję,ale z punktu widzenia kosztu nie jest to bardzo istotne, choć oczywiście wszelka redundancja zajmuje pewną ilość miejsca pamięci. Różnie z tym bywa w zależności od dziedziny. Będą też trudności przy komunikowaniu się.

Jak się przedstawia odpowiedzialność za informację na poszczególnych szczeblach scentralizowanego systemu? Na szczeblu indywidualnych urządzeń końcowych mamy urządzenia zależne, niewiele więc jest indywidualnej odpowiedzialności za informację. Jeśli chodzi o kontrolę dostępu do danych, sprawa jest dość prosta. Trzeba mieć system bazy danych.Nie ma problemu kontroli dostępu, bo każda z komórek organizacyjnych otrzymywać będzie tylko informację, do której otrzymywania jest uprawniona. Toteż nie ma rzeczywistej potrzeby kontrolowania tego dostępu.

Zacząłem mówić o przetwarzaniu rozproszonym, w której to dziedzinie IBM prowadzi rozległe prace badawcze.

Przypatrzmy się kolejno każdej z tych architektur. Czy w tym pierwszym przypadku pewne rodzaje przetwarzania danych,które będą miały miejsce stosunkowo rzadko, będą się powtarzać w

każdej z tych komórek,czy też będą wykonywane w jednym centralnym punkcie? Czy każdy może w każdej chwili otrzymać informację wszelkiego rodzaju? Teoretycznie byłoby to możliwe, gdyż przy tego typu architekturze można by utworzyć supersieć. Ale wynikłyby trudności z architekturą,gdybyśmy zechcieli stworzyć tego rodzaju sieć z takimi możliwościami.

Czy mamy tu możliwość podziału obciążeń roboczych? Przy tym konkretnym rozwiązaniu cała manipulacja danymi odbywa się w ośrodku centralnym i nie ma sposobu przerzucania obciążeń na obszary peryferyjne.

Jak dalece atrakcyjny byłby ów pośredni typ systemu informatycznego? Pozwala on nam mieć proste systemy użytkowe na szczeblu lokalnym, wyszukiwać dane u źródła, a także działać niezależnie w przypadku awarii. Pozwala odciążyć jednostkę centralną od pewnych prac i uzyskać oszczędność na transmisji, szczególnie na liniach przenoszących wyniki przetwarzania danych.W jednostce centralnej skomplikowane systemy będą przeznaczone do zadań wysokiego rzędu. Równocześnie są pewne zadania, do których wykonywania nie jest potrzebny system centralny. Na przykład programy liniowe można wykonywać na małym "rozproszonym" komputerze. Są jednak pewne rzeczy, które wymagają wykonywania w centralnym ośrodku: przechowywanie zbiorów historycznych i tekstów, przy których mamy do czynienia z dużą ilością dokumentacji. Nie można tego zrobić w sposób rozproszony, toteż jednostka centralna powinna być wyspecjalizowana do tego rodzaju funkcji.

Jakie jest praktyczne znaczenie tego wszystkiego dziś, z punktu widzenia krótkoterminowej problematyki? Załóżmy, że ze względu na ceny komputerów IBM-u lub urządzeń końcowych nie możemy sobie pozwolić na stworzenie systemu tego ostatniego typu i musimy wybrać rozwiązanie typu drugiego lub rozwiązanie scentralizowane oparte na elementarnych urządzeniach końcowych. Trzeba przy tym dokonać pewnych wyborów. Po pierwsze należy pamiętać, że potrzebna będzie częściowa decentralizacja i niezbędny będzie jakiś sposób takiego utrzymywania systemów użytkowych,by pewne dane i pewne typy przetwarzania były rozrzucone

na określonym terytorium; bo żeby osiągnąć pewien poziom wydajności, będziemy zmuszeni do decentralizacji w ramach pewnego terenu.

Chciałbym teraz podać Państwu kilka definicji. Scentralizowanym przetwarzaniem danych nazywać będziemy zgrupowanie w pewnym punkcie terytorialnym mocy obliczeniowych należących do przedsiębiorstwa, do których dostęp można uzyskiwać z elementarnych urządzeń końcowych /ilustr.F5/. Jest to dobrze znany typ pracy.

Przetwarzanie lokalne - znowu może wydać się Państwu, że IBM preferuje przetwarzanie rozproszone - jest rozwiązaniem pośrednim /por.ilustr.F6/. Co prawda nie jest to problem trudny, ale musimy pamiętać, że zawsze są pewne sprawy, którym należy dawać pierwszeństwo.Przy przetwarzaniu lokalnym są nimi problemy produkcji.Pierwszeństwo przysługuje tu rozwiązywaniu problemów lokalnych. Małe komputery, rozmieszczone wszędzie dla załatwiania tych problemów, mogą też komunikować się między sobą, ale nie jest to ich głównym zadaniem.

Przetwarzanie rozproszone nie wymaga, naszym zdaniem, rozproszenia mocy obliczeniowej aż do najdalej położonych punktów lokalnych. Inteligentne urządzenia końcowe umieszcza się w węzłach systemu informacyjnego, to znaczy w miejscach, w których strumień danych jest najsilniejszy; w tym przypadku priorytetowym problemem jest transmisja danych. Takie urządzenie końcowe może oczywiście załatwiać także problemy lokalne, ale priorytet mają dla niego sprawy transmisji danych /ilustr. F7/.

Są trzy tryby przetwarzania danych, jakie można wziąć pod uwagę przy projektowaniu systemu informatycznego. Zanim więc zajmiemy się architekturą sieci, popatrzmy jakie mamy dziś możliwości w tym względzie. Mówić będę o architekturze zarówno układów wielokomputerowych jak i sieci urządzeń końcowych. Przede wszystkim pomówmy o transakcjach.Oto typ architektury zapewniający komunikację tylko od elementarnych urządzeń końcowych do zbiorów danych. Ale nawet i to stwarza pewne trudności. Jeśli bowiem chodzi o zbiory, musimy oczywiście mieć zbiory danych, ale obok nich także zbiory dotyczące formy wyjścia. Potrzebny

nam jest format i przechowujemy go w takim zbiorze. Mamy też zbiory służące do kontroli lub weryfikowania informacji itd. Potrzebne nam są więc pewne zbiory w centrali, których nie będziemy mieli w innych miejscach.

Weźmy dla przykładu zbiór dyskowy. Nie twierdzę, że jest to idealne urządzenie końcowe, ale interesuje mnie jego architektura, co jest ono w stanie robić i jaka jest jego wydajność. Dla tego typu dysków mamy specjalną filozofię. Użytkownik w tym przypadku komunikuje się z lokalnymi urządzeniami końcowymi, które mogą w każdej chwili albo komunikować, czyli notyfikować dane lokalne albo komunikować się z centralnymi systemami użytkowymi, które z kolei mogą komunikować się z centralnymi zbiorami danych. Może to być atrakcyjne pod warunkiem, że będzie wykonywane w sposób dynamiczny, tzn. jeśli w każdym momencie urządzenie końcowe lub użytkownik może bardzo szybko przejść od pracy z danymi lokalnymi do pracy z danymi centralnymi, potem znów przestać i pracować przez resztę dnia ze swoimi danymi lokalnymi, a później w dowolnym momencie przejść do pracy na bieżąco lub w trybie podziału czasu, posługując się konwencjonalnymi językami wysokiego rzędu lub jakimkolwiek innym rodzajem programu. Zagadnienie polega na tym,jak opracować architekturę, która by rozwiązywała postawiony przeze mnie problem, bo oczywiste jest, że samo założenie linii między dwiema jednostkami inteligentnymi nie wystarczy do jego rozwiązania.Potrzebna jest pewna architektura.

Nie twierdzę, że musi to być koniecznie architektura typu SNA /standard network architecture/, mówię jedynie, że potrzebujemy jakiejś architektury, czy to będzie SNA czy jakaś inna.

A teraz parę słów o obecnej sytuacji w dziedzinie jednostek tego rodzaju. Koncepcję rozproszonego przetwarzania przy użyciu urządzenia 3790 przedstawiono na ilustracji F8. Potrzebny nam jest drugi szczebel hierarchiczny,to znaczy w pewnej odległości od tego konkretnego miejsca musimy mieć drugi monitor ekranowy. Może to być stacja regionalna z monitorami ekranowymi w biurach, użytkowanymi w dokładnie taki sam sposób,jak gdyby były rozmieszczone na terenie regionu.Potrzebna nam jest więc pewna hie-

rarchia, obejmująca centralę, regiony, a potem pewne jeszcze mniejsze obszary.

To jest pierwsze stadium architektury sieci, bo gdy nie możemy mieć układu sieciowego, w którym każdy punkt jest bezpośrednio połączony z wszystkimi pozostałymi /grid network/, musimy mieć sieć zorganizowaną hierarchicznie.To byłaby więc sieć typu hierarchicznego. Potem zobaczymy sieć,w której każdy punkt jest powiązany z wszystkimi pozostałymi.

Oczywiście nie jest moim zadaniem tutaj reklamowanie wszystkich możliwości urządzenia 3790; jest wielu innych pracowników IBM-u, którzy Wam mogą to powiedzieć.Chciałbym tylko zaznaczyć, że staramy się, by wszystkie te operacje były dla użytkownika "niewidzialne". Co to znaczy "niewidzialne dla użytkownika" /"transparent for the user"/? Jest to oczywiście IBM-owska terminologia. Gdy rozmawiacie z ludźmi z IBM-u, słyszycie ciągle tylko słowa "wirtualny", "niewidzialność" itd. SNA jest m.in. próbą uczynienia sieci wirtualnych "niewidzialnymi" dla użytkownika.

Gdy zastosujecie architekturę SNA, zobaczycie sami, ile z tego jest prawdą, a ile niezupełnie prawdą. Oto tutaj inny wariant. Użytkownika nic nie obchodzi,czy komunikuje się z danymi lokalnymi czy też pracuje z bazą danych w centrali. Nie dostrzega żadnej różnicy; po prostu ma dostęp do danych - i to wszystko /ilustr. F9/. Problem polega jedynie na tym,gdzie dane są zlokalizowane. Ale to jest problem architektury, który rozstrzyga projektant systemu.

Innym klasycznym problemem w systemach teleinformatycznych jest to, że bardzo często program użytkowy obejmuje pewne elementy, które na dobrą sprawę nie mają z nim nic wspólnego. Na przykład większość programów użytkowych w istniejących systemach operuje meldunkami lub dostępem do danych, ale również w pewnej mierze zarządza siecią, np. żeby wiedzieć,czy ekran monitora jest kwadratowy, albo jaki rodzaj meldunku dane urządzenie końcowe może odebrać lub nadać. Jest to o tyle niebezpieczne,że gdy zmienia się ekran lub technika, system użytkowy zostaje częściowo pozbawiony wartości i trzeba dokonać dość rozległej jego aktualizacji.

Co robimy więc? Zasadą jest, że program użytkowy powinien zajmować się tylko systemem jako takim, natomiast odpowiedzialność za wszystko inne należy złożyć na system operacyjny w centrali, na jednostki sterujące i na stacje urządzeń końcowych. Oczywiście sposób, w jaki przedstawimy te koncepcje użytkownikowi będzie następujący. Powiemy mu, że może mieć za pośrednictwem małych inteligentnych urządzeń końcowych zarówno dostęp do przetwarzania rozproszonego jak i do wielkich systemów centralnych i tak dalej. Musi być przy tym stworzona infrastruktura systemu niezbędna do uzyskiwania tego dostępu. Powinna ona objąć całe przedsiębiorstwo. Są to bardzo skomplikowane zagadnienia.

Gdy użytkownikowi systemu operacyjnego powiemy, że musi podjąć jeszcze 10-krotnie większy wysiłek dla stworzenia architektury sieci, powie on: "nie, to trochę za wiele". Przyjrzyjmy się więc nieco bliżej architekturze sieci,takiej jaką spotykamy w obecnych rzeczywistych warunkach.

Zanim przejdę do szczegółów SNA,chciałbym powiedzieć kilka słów o sieci jako takiej. SNA oznacza "Standard Network Architecture" /standardowa architektura sieci/. "Sieć" to może w tym przypadku zbyt wielkie słowo, mimo to jednak SNA może być punktem wyjścia dla koncepcji sieci. Widzicie Państwo tutaj sieć, która pozwala każdej jednostce komunikować się z każdą inną jednostką w sposób stały /ilustr. F10/. Szczebel 1 przedstawia mniej więcej to, co można zrobić w systemie transmisji danych przy dzisiejszej architekturze.Szczebel 2: w tym przypadku istnieje możliwość wysyłania transakcji przez urządzenie końcowe do komputera, ale ten komputer ma możliwość przerzucić ten program użytkowy do innego komputera,lepiej nadającego się do jego wykonania, a za pośrednictwem tych dwóch połączonych komputerów można uzyskać dostęp od urządzenia końcowego 1 do urządzenia końcowego 2. Jest to dziś możliwe w wielu standardowych systemach dostępnych na rynku. Ze względu na problemy protokółu między dwoma komputerami jest to trudne do zrealizowania, ale zupełnie wykonalne. Szczebel 3 jest nieco bardziej wymyślny - linia kreskowa przedstawia żądanie urządzenia końcowego 1 skomu-

nikowania się z urządzeniem końcowym 2. Można by zupełnie dobrze wyobrazić sobie, że żądaniem tym zajmie się komputer 1 i przetransmituje je do komputera 2; od tej chwili komputer 1 zostaje wyłączony z obwodu, a urządzenie końcowe 1 łączy się z komputerem 2 i przez niego z urządzeniem końcowym 2. Daje to nam krótszą pętlę w tej łączności między urządzeniami końcowymi.

W ten sposób dochodzimy do szczebla 4. Układy tego typu są już wdrożone. W przypadku szczebla 2 i 3 zachodzi wiele komplikacji, jeśli chcemy spójności. Natomiast w układzie szczebla 4 nie mamy do czynienia z linią czasową,ale jest to system, który automatycznie zapewnia realizację wszystkich tych żądań, to znaczy, że gdy urządzenie końcowe 1 chce nawiązać łączność z urządzeniem końcowym 2 lub z komputerem, albo gdy komputer chce nawiązać łączność z urządzeniem końcowym, wszystko to może być zrealizowane, a równocześnie mogą być rozstrzygane wszelkie ewentualne konflikty o linię. Tego rodzaju możliwości są wbudowane w SNA.

Głównie chcę mówić o szczeblu 1, to znaczy o SNA, z którym jesteście już dziś obeznani. To co widzimy na ilustracji, to mniej więcej to co widzi użytkownik; to znaczy użytkownik powinien być w stanie orientować się w systemie transmisji danych; użytkownik chce właśnie tego co tu pokazano - chce komunikować się z innym użytkownikiem. Gdy mówię "użytkownik", muszę zdefiniować ten termin, wyjaśniając, że rozumiem przez to zarówno ludzi jak i programy. Użytkownikiem może więc być albo program albo człowiek.Jest to zatem problem komunikacji programu z człowiekiem, człowieka z człowiekiem lub wreszcie programu z programem.

Pierwsza rzecz, jakiej potrzebujemy, to bramy prowadzące do systemu transmisji danych.W IBM-owskiej koncepcji są to jednostki logiczne pełniące funkcje jak gdyby bram wejściowych do systemu. Po drugie, musimy być w stanie ustanawiać dynamiczne łącze między dwiema bramami, to znaczy, gdy mamy dwie bramy, musimy stworzyć rodzaj abstrakcyjnego obwodu między tymi dwiema bramami wejściowymi do systemu /ilustr. F11/. Takie są ogólne koncepcje w zakresie telekomunikacji. Ta koncepcja obwodów wir-

tualnych, np. w systemie TRANSPAC polega na dynamicznym tworzeniu - czyli tworzeniu w określonym momencie i na określony czas - łącza między dwoma punktami. W naszym przypadku problem jest może nieco trudniejszy, bo chodzi o łącze logiczne.

Co mam na myśli mówiąc "łącze logiczne"? Oznacza to, że nie tylko pragnę ustanowić łącze, ale potrzebny mi jest także protokół transmisji danych, to znaczy reguły pozwalające mi wprowadzać zmiany gdy tylko zechcę. Ten protokół transmisji jest bardzo interesujący. Jest to właśnie SNA. Każdy z przedstawionych na ilustracji elementów jest bezwzględnie niezbędny. Na szczycie widzimy protokół SSCP, który zna wszystkie bramy wejściowe do systemu. Ten element jest absolutnie konieczny. Gdy nie ma nikogo, kto zna wszystkie bramy, nie można zapoczątkować "sesji" między dwoma punktami. Ktoś lub coś musi powiedzieć: "otwieram sesję" lub "zamykam sesję" między tym a tamtym punktem. SSCP jest więc tym, który zna wszystkich w sieci.

Istnieje problem wyboru między sieciami hierarchicznymi, a sieciami, w których każdy punkt jest bezpośrednio powiązany z każdym innym /grid networks - sieci "siatkowe"/. Gdy mamy tylko jeden taki punkt sterujący, to znaczy gdy SSCP jest scentralizowany, sieć musi być typu hierarchicznego. Jest wówczas jeden uprzywilejowany punkt, który zna adresy wszystkich innych punktów sieci. Gdy rozłożę ten punkt na większą liczbę różnych punktów, mogę stworzyć sieć typu "siatkowego", ale przy obecnej wersji SNA architektura sieci przewiduje tylko jeden punkt z SSCP, a zatem chwilowo jest to hierarchiczny typ sieci.

Ponieważ SSCP zna adresy różnych punktów sieci, może dynamicznie zestawiać połączenie między jednym punktem a drugim.

Skąd się wzięły te różne współśrodkowe koła uwidocznione na ilustracji? Są one niezbędne i chciałbym krótko omówić ich znaczenie. Najbardziej wewnętrzne koło, które nazywamy "siecią wspólną" /common network/ spełnia fizyczne funkcje transmisji. Przy tej koncepcji podziału funkcji problem polega na tym, że funkcje te są bardzo ściśle powiązane wzajemnie. Kto wysyła meldunek? Kto zajmuje się sterowaniem? Kto robi to? Kto robi owo? Skorzystaliśmy z potrzeby stworzenia sieci typu "siatkowego", by

sformalizować te pojęcia,a zatem szczebel wewnętrzny będzie się zajmował przede wszystkim formatyzowaniem meldunków w zależności od użytkowanych łącz. Na przykład gdy mam urządzenie końcowe połączone z komputerem przez kanał, sformatyzuję meldunek nadając mu format kanału. Jeśli natomiast mamy transmisję typu SDLC /synchronous data link control - synchroniczne sterowanie łączami transmisji danych/, format będzie odpowiedni do tego. Tak zatem przedstawia się formatyzowanie.

Dalszą funkcją jest marszrutowanie. Sieć ma często złożoną strukturę, co stwarza wiele możliwości marszrutowania, to znaczy, że meldunek może przechodzić przez różne punkty, czyli po różnej marszrucie. Marszrutowaniem zajmuje się owa "sieś wspólna". W zasadzie nie ma niewymienności między tą siecią wspólną a publicznymi sieciami PTT.

Natomiast krąg o szczebel wyższy dotyczy systemu transmisji. Jaką rolę odgrywa? Ma zapewniać kontrolę sesji, w tym także dynamiczne ustanawianie sesji. Problem polega na tym, że sesje te powinny być kontrolowane na tyle,by można było wiedzieć, czy wszystko przebiega prawidłowo;czy meldunki wyszły z jednego punktu i dotarły do innego. Meldunki są przesyłane przez sieć, a gdy docierają do miejsca przeznaczenia,system powinien sprawdzić, czy numeracja jest kolejna, a gdy brak któregoś numeru, trzeba wrócić do punktu nadania i zawiadomić go,że jednego meldunku brak; np. że zabrakło meldunku nr 36 z będącej w toku sesji. Umożliwia to płynną transmisję i zapobiega kłopotom. Jest to coś w rodzaju rozmowy: "wysłałem meldunek; czy otrzymałeś go?" "tak, otrzymałem" lub "nie, nie otrzymałem". Jest to operacja przebiegająca "tam i z powrotem" i wskazująca, czy jakieś meldunki nie zaginęły po drodze.

Ten rodzaj funkcji wykonywany jest przez "system transmisji" i jak widać, jest to funkcja o zasadniczym znaczeniu. A jakiego rodzaju funkcje wykonuje system najwyższego szczebla, wyrażony kołem zewnętrznym? Ten system najwyższego szczebla świadczy usługi użytkownikowi,spełnia funkcję usług użytkowych, to znaczy decyduje jak zainicjować i zamknąć sesję, na przykład, gdy użytkownik przychodzi i powiada: "chciałbym rozmawiać z sy-

stemem centralnym" lub "teraz chcę pracować z systemami lokalnymi". Taki to typ funkcji będzie wykonywany na tym szczeblu.

A teraz, jak te różne warstwy są rozłożone w całym systemie? W rzeczywistości przy obecnym "stanie sztuki" i na raczej wczesnym etapie rozwoju architektury, mamy komputer, mamy jednostki sterujące i ewentualnie inne jednostki do zdalnego sterowania. Widzimy tu urządzenia końcowe, możemy też mieć jeszcze inne urządzenia końcowe, także i takie, które będą jeszcze bardziej oddalone. Gdzie będą rozmieszczone wszystkie funkcje, to znaczy jednostka logiczna,jednostka fizyczna i tak dalej? Jedno jest niezaprzeczalne, a mianowicie to, że funkcja jednostki fizycznej będzie umieszczona na tym szczeblu. SSCP systemu będzie musiała znajdować się mniej więcej tutaj i musi wiedzieć o istnieniu tych jednostek fizycznych by móc dokonywać marszrutowania meldunków.

W tym samym kręgu spotykamy kilka jednostek logicznych, bo stanowią one bramy prowadzące do systemu. Oznaczyłem to linią kreskową, ponieważ chodzi tu o pewną możliwość,to znaczy, jeśli zechcemy zastosować w swoim systemie komunikacyjnym komputery czołowe, to właśnie będzie miejsce, w którym umieszczone zostanie inteligentne urządzenie końcowe. Gdzie zlokalizowane będą funkcje typu "sieci wspólnej"? Na przykład formatyzowanie i deformatyzowanie? Niektóre będą tutaj, niektóre tam, część tam u góry, część tu u doły.Na każdym z łączy znajdziemy mały obszar zajmujący się tym co nazywamy "sterowaniem łączem transmisji danych" /data link control/, co oznacza zarządzanie linią lub łączem po to, by meldunek skierowany został właściwą marszrutą.

Tutaj zlokalizowane będzie SDLC. Będzie to jednostka sterująca kanałem i odpowiadająca jej jednostka sterowania łączem transmisji danych /DLC/. Ten typ architektury ma swoje zalety. Załóżmy bowiem, że nastąpi jakiś przewrót techniczny i że nagle zniknie DLC i zastąpione zostanie przez jakąś inną metodę - X, Y czy Z.Co trzeba będzie wówczas zmienić w omówionej koncepcji? Widzimy tutaj to koło odpowiadające modułowi. Teraz jest to moduł DLC. Staraliśmy się zastosować układ modułowy, który może przetrwać,dostosowując się do bardzo szybko następujących zmian technicznych.

Załóżmy,że chcemy zastosować tę koncepcję bramy wejściowej do punktów sterowania, tak aby wszystkie meldunki wchodziły tędy. Meldunek może być po zakończeniu odbioru przesłany do innej jednostki sterującej tego rodzaju, która wysyła go do innego komputera. Może to wymagać bramki logicznej. Zupełnie łatwo można by tutaj, w tej jednostce sterującej, stworzyć funkcję jednostki logicznej;ale taka konstrukcja z małych cegiełek nastręcza oczywiście trudności producentowi. Nie bardzo będzie rad zmianom w architekturze. Natomiast użytkownik chciałby móc dotrzymywać kroku postępowi technicznemu i stosować nowości, gdy tylko się pojawią.

Dzięki tej funkcji tutaj na przykład, moglibyśmy ewentualnie, gdyby zmieniła się liczba urządzeń pamięci,przyłączyć więcej urządzeń końcowych niż dawniej. Z punktu widzenia architektury byłoby to zupełnie możliwe. Ale gdy założymy, że jednostka ta została przewidziana wyłącznie dla jednego urządzenia końcowego, nie moglibyśmy dodać do niej więcej urządzeń końcowych, a gdybyśmy w przyszłości zechcieli przyłączyć 50 urządzeń końcowych, trzeba by zapewnić 50 bram wejściowych,co byłoby zupełnie nie do pogodzenia ze strukturą systemu jako całością.

Inna rzecz, na jaką pozwoliłby ten typ architektury, wypływa z samej koncepcji "sesji",tego nowego wynalezionego przez nas pojęcia. Myśl polega na tym, by w każdym czasie każdy użytkownik mógł otworzyć sesję, albo lokalną - zwracam uwagę na to pojęcie sesji lokalnej polegającej na komunikowaniu się z danymi lokalnymi - a mianowicie przez jedną z bram, albo sesję centralną - poprzez inną bramę. Oznacza to, że użytkownik może w każdej chwili dokonywać transakcji; przez pewien czas może pracować w trybie wielodostępu, a w chwilę później dokonywać operacji typu zdalnego wprowadzania zadań, a wszystko to przez różne bramy.

Chcę przez to powiedzieć, że samo urządzenie końcowe jako takie nie jest specjalizowane, może wykonywać szereg funkcji, ale koncepcja sesji oznacza dynamiczne ustanawianie łącza, co umożliwia urządzeniu końcowemu wykonywanie różnych konkretnych funkcji. Możemy dokonywać transakcji w systemie IMS lub jakim-

kolwiek innym typie systemu, obsługiwanym przez jedną bramę. Inna brama służy do wielodostępu, a jeszcze inna zapewnia nam owo automatyczne marszrutowanie ku temu czy innemu systemowi użytkowemu.

Czego brak w tego typu architekturze? /Obawiam się,że brak wielu rzeczy. Jeśli mówię, że czegoś brak, mam na myśli coś, o czym się już myśli i co można by w jakimś niezbyt odległym czasie zrealizować/. Po pierwsze, sądzę, że potrzebne nam są w tym tutaj punkcie,jeśli zechcemy to potraktować jako duży komputer, funkcje manipulowania danymi lokalnymi. Bez nich nie będziemy mogli stosować dowolnie wszelkich potrzebnych nam programów. Cała ta koncepcja architektoniczna wynikła stąd, że systemy użytkowe są scentralizowane po to, by uniknąć niespójności.Nawet gdybyśmy chcieli oprzeć się na rozproszonych bazach danych z dynamicznym dostępem natychmiastowym i tak dalej, nadal musielibyśmy utrzymać kontrolę nad danymi, a równocześnie użytkownik chciałby sam operować i manipulować swoimi danymi. Ale tego nam w tej chwili brak. Jeśli użytkownik chce rzeczywiście sam manipulować swoimi danymi lub modyfikować je, nie może postępować dowolnie. Często użytkownik chce manipulować danymi, które są scentralizowane. Dziś nie ma on możliwości dokonywania tego za pomocą języków spontanicznych lub rozszerzalnych.Może natomiast zdefiniować swoje własne podzbiory,ale musi to zrobić na szczeblu centralnym a nie lokalnym.

Innym problemem tego samego rodzaju jest spontaniczny dostęp lub wyszukiwanie tekstów i fotografii.

Inny problem pokazano na ilustracji F11. Uwidoczniono tam różne możliwe szczeble interface'u. Możemy zadawać pewne pytania. Większość systemów może zadawać pytania albo lokalnie albo na szczeblu centralnym. Ale już wykrywanie błędów, czyli anomalii stwarza trudności na szczeblu lokalnym. Trudności sprawia diagnostyka,ponieważ operujemy dużą ilością danych i na dłuższą metę staje się to na szczeblu lokalnym szczególnie trudnym problemem. Traktowanie wyjątków: zależy to od projektu. Trzeba to traktować jako element procesu decyzyjnego w ramach całego przedsiębiorstwa, toteż jest bardzo nieprawdopodobne, by był to

typ funkcji nadający się do delegowania na szczeble lokalne. Nie sądzę więc, by dokonano dużego postępu w kierunku delegowania tych funkcji w ramach struktury systemu.

Chciałbym zaproponować pewien system klasyfikacji. Prawdopodobnie nie zgodzicie się z nim, bo jest bardzo schematyczny, ale przynajmniej da on nam podstawę do dyskusji. Pragnąłbym sprecyzować różnicę między urządzeniem końcowym a komputerem. Gdy mówię "komputer",mam w tym przypadku na myśli minikomputery lub przynajmniej małe komputery, w odróżnieniu od urządzeń końcowych. Pierwszym różniącym je punktem są priorytety; priorytetowym problemem dla urządzeń końcowych jest transmisja danych, natomiast dla minikomputerów jest nim produkcja w postaci danych nie tylko cyfrowych ale i nienumerycznych. Dziś operowanie danymi nienumerycznymi możliwe jest jedynie w centrali; np. wyszukiwanie dokumentów za pomocą słów kluczowych itp. Omówiony typ architektury i urządzeń końcowych ma umożliwiać manipulowanie danymi, o których stosunkowo mało wiadomo, oraz modyfikowanie ich.

Dotychczas mówiłem o danych kodowanych, ale słyszeliśmy dziś przed południem, jak ogromne możliwości wiążą się z danymi niekodowanymi. Pierwszym przykładem jest tekst - manipulowanie tekstem pisanym. To jeden z przykładów danych niekodowanych. Operowanie takimi danymi wymaga ogromnych urządzeń pamięci i niewiele jest nadziei,by można je było w tym stopniu rozbudować na szczeblu lokalnym. Toteż pierwsze zastosowania niekodowane będą musiały być ujęte w zbiorach w centrali, w jednostce centralnej, bo jeśli mamy mieć dostęp do biblioteki on-line, potrzebne nam będą duże zbiory; na szczeblu rozproszonego przetwarzania nie ma takiej możliwości.

Dziś czas wyszukiwania dokumentów jest dość długi. Musimy się z tym pogodzić, bo wiecie,że problem wyszukiwania jest bardzo obszerny. Gdyby można było umieszczać te systemy informacji niekodowanej na szczeblu lokalnym, potrzebny byłby nam sprzęt znacznie szybszy niż ten, jakim dysponujemy obecnie.

Rozróżnienie priorytetów bynajmniej nie znaczy, że urządzenie końcowe nie może wykonywać pracy produkcyjnej, ale jest

przeznaczone w pierwszym rzędzie do komunikacji.Urządzenie końcowe zawsze działa w ramach pewnej architektury. Nie można wyobrazić sobie urządzenia końcowego poza jakąś architekturą, podczas gdy w przypadku minikomputerów problemy architektury nie odgrywają żadnej roli, a jedynie problemy efektywności ekonomicznej i wydajności, ponieważ minikomputery nie są związane z żadną architekturą. Przy tej samej cenie i tej samej funkcji inteligentne urządzenie końcowe często kalkuluje się drożej niż minikomputer, ponieważ do ceny urządzenia końcowego trzeba doliczyć jeszcze koszt architektury, w której ramach działa.

Dalszą różnicą jest, że na dłuższą metę z minikomputerem wiążą się problemy aktualizacji, o których już mówiłem, a to z tej przyczyny, że systemy użytkowe z czasem zmieniają się. Są zmienne i trzeba je zmieniać tak,by każdy mógł się komunikować, zaś typ architektury z urządzeniami końcowymi daje teoretycznie lepsze możliwości rozwijania systemów użytkowych.

Tutaj jednak czyha na nas pułapka. Powszechnie uważa się, że "otwarty" typ architektury daje możliwości wyboru. Gdy mamy określoną architekturę,wiemy, że w ramach tej architektury może odbywać się określony rozwój; ale w tym przypadku nie mamy określonej architektury. Gdy później zachodzą zmiany w organizacji,niełatwo jest przyjąć inne rozwiązanie w miejsce rozproszonej mocy obliczeniowej. Natomiast tam, gdzie mamy pewną podstawową architekturę - w naszym przypadku SNA - mamy w jej ramach większe możliwości wyboru, np. między centralizacją a decentralizacją, pozostając mimo to w ramach tej architektury i nie rezygnując z niej. Mamy więc większy stopień elastyczności.

I wreszcie, uważam, że gdy mamy w ramach rozproszonego przetwarzania monitory ekranowe wprost u użytkowników, potrzeba nam większej liczby specjalistów APD. Nie mamy tylu ludzi ilu potrzeba do tego, bo przy każdym urządzeniu końcowym powinniśmy mieć kogoś,kto potrafi użytkować komputer i wie choć w przybliżeniu, co to jest kompilator; natomiast przy architekturze typu SNA nie są nam potrzebni informatycy w punktach lokalnych, bo wszystko wykonywane jest w centrali.

Myślę, że na tym powinienem skończyć. Sądzę, że dałem Państwu materiał do przemyślenia i do naszej dyskusji. Dziękuję Państwu.

NIEZIDENTYFIKOWANY MÓWCA: Oparł Pan całe swoje podejście do problemu na kilku bardzo zasadniczych założeniach, wyszczególnionych na pierwszym przeźroczu. Dwa z tych założeń to - po pierwsze, że przyszłość należy do systemów bardziej złożonych i - po drugie, że zachodzi potrzeba zintegrowania mechanizmów sterujących. Zanim wszedłem w ten tzw. "świat informatyki",miałem pewne doświadczenie jako kierownik fabryki i mam podstawy wątpić w oba te założenia. Jeśli zaś założenia te nie są słuszne, można zakwestionować wartość SNA. Jeżeli np., jak napisał profesor Dijkstra, nazywając siebie "skromnym programistą", w przyszłości wymagana będzie raczej prostota niż skomplikowanie, to co nazywa on "racjonalnie łatwymi do sterowania systemami" i skoro sterowanie ma opierać się, zgodnie z przewidywaniami McGregora, na ludzkim samosterowaniu, a samosterowanie jest zasadniczo proste, wówczas SNA może stać się przysłowiową "armatą na muchy".

Jeśli poza tym jest prawdą, że narzędzie wpływa na ludzi, którzy posługują się nim, SNA może skłonić ludzi do szukania efektownych, skomplikowanych i zintegrowanych rozwiązań w przypadkach, gdy to wcale nie jest potrzebne, a nawet gdy jest zupełnie błędne i przeszkodzić tworzeniu prostych systemów,takich o jakich mówi profesor Dijkstra. Twierdzę, że przyszłość należy w ogromnej większości do tych bardzo prostych, podstawowych systemów; ludzie używać będą notatników kieszonkowych, takich jak ten tutaj, tyle tylko że elektronicznie. Wątpię bardzo, czy ta armata, choć bardzo piękna, będzie nam potrzebna poza wyjątkowymi przypadkami.

DAVID: Zgadzam się z Panem zupełnie,że kiedyś komputer stanie się czymś w rodzaju telefonu, to znaczy, że interface człowiek/ /maszyna stanie się czymś bardzo prostym.Ale mocno obawiam się, że w ciągu najbliższych 10 lat, to znaczy do roku 1985, nie potrafimy - może ze względu na wąskie gardła telekomunikacji - bu-

dować w tak bardzo prosty sposób komputerów i rozproszonej inteligencji. Oczywiście nie twierdzę, że SNA jest cudownym lekarstwem na wszystkie niedomagania, ani że będzie rozwiązaniem na najbliższe pół wieku. Powiedziałem jedynie, że zaznacza się nasilająca się tendencja ku bardzo daleko idącemu rozproszeniu mocy obliczeniowej i że tą rozproszoną mocą obliczeniową można operować w sposób bezładny lub z pewnym stopniem spójności. Uważam, że infrastruktura SNA niekoniecznie musi być potwornie skomplikowana, żeby móc tym wszystkim manipulować. W obecnym stanie rzeczy, to znaczy na pierwszym etapie rozwoju SNA, nie ma w niej niczego monstrualnego; co prawda przypuszczam,że wraz z rozwojem sieci typu "siatkowego",struktura ta stanie się bardziej skomplikowana, ale niekoniecznie aż potwornie skomplikowana.

My jako producenci mamy problem, o którym wspomniałem na początku, a mianowicie spadek cen i rentowności produkcji, szczególnie jednostek centralnych. Jedynym sposobem zrekompensowania tego i utrzymania poziomu zysku jest rozwój systemów użytkowych. Na szczęście uważamy, że systemy użytkowe stanowią ogromną dziedzinę. Dotknęliśmy na razie tylko samej powierzchni dziedziny systemów niekodowanych;pozostaje jeszcze ogromne pole dla rozwoju ich zastosowań.Szybkość,z jaką rozwijane są systemy użytkowe, musi dotrzymywać kroku spadkowi cen lub nawet wyprzedzać go. Aby to uczynić,musimy uwolnić się od istniejących dziś wąskich gardeł. Główne wąskie gardło to sprzężenie zwrotne, ściślej mówiąc organizacyjne sprzężenie zwrotne, z jakim spotyka się użytkownik za każdym razem, gdy chce wprowadzić jakąś zmianę lub pragnie nowych usług, albo też gdy my chcemy zaoferować mu nowy typ usług.Odpowiedź ośrodka obliczeniowego zwykle brzmi: "tak, to bardzo interesujące, ale my musimy chodzić po ziemi i robić to co jest wykonalne". A zatem zupełne zwolnienie hamulców nie byłoby może rozsądne.

SNA jest w każdym razie jednym z istotnych elementów w tym obrazie. Wiem,że nie odpowiedziałem w pełni na Pańskie pytanie, ale IBM znany jest z tego, że nie zawsze chce udzielać pełnej odpowiedzi.

DUVERGER: Opisana przez p.Davida architektura jest bardzo atrakcyjna. Łatwo mi wyobrazić sobie, że duże przedsiębiorstwa zainteresują się tym systemem. Ale jest jedna rzecz, co do której mam pewne wątpliwości: sprawa aktualizacji systemów użytkowych. Uważam, że grupom regionalnym należy dać pewien zakres samodzialności, aby umożliwić im koegzystencję z tym systemem przetwarzania danych. Oczywiście, gdy przyznamy pewien zakres samodzielności, będziemy mieli pewne stałe jądro,a wokół tego jądra zachodzić będzie pewien rozwój niezupełnie zgodny z tym co zaplanowano w centrali. Toteż przy transmitowaniu informacji od jednego regionu przetwarzania danych do drugiego natkniemy się na problem interface'u. Tutaj właśnie służba aktualizacji systemów użytkowych ma swoją rolę do odegrania. Chciałbym zapytać referenta, czy ma on jakąś koncepcję, jak powinna działać taka służba aktualizacyjna i na jakim szczeblu powinna odbywać się aktualizacja. Ja mam oczywiście swoje zdanie na ten temat, chciałbym jednak usłyszeć co myśli referent.

DAVID: Pytanie nie należy do łatwych. Architektura sieci jest rzeczą dobrze znaną. IBM ma dwa działy, z których jeden zajmuje się niespójnym przetwarzaniem danych, ponieważ sprzedaje małe systemy. Dziedzina ta jest szeroko otwarta dla wszelkich rodzajów rozwiązań. SNA reprezentuje jedynie to, co mogłoby odpowiadać wielkim przedsiębiorstwom.

Gdzie jest miejsce służby aktualizacyjnej? Niełatwo na to odpowiedzieć. Jeśli za aktualizację odpowiedzialne są różne ośrodki, napotykamy na wieczny problem komunikowania się między użytkownikami a usługodawcami. Gdyby natomiast pracownicy tej służby, mający pisać programy, byli rozmieszczeni w węzłach komunikacyjnych, byłoby to idealne rozwiązanie. Za aktualizację powinny odpowiadać regiony.Gdy przedsiębiorstwo obejmuje szereg regionów, sądzę, że w każdym takim regionie powinien znajdować się ośrodek aktualizacji. Taka służba znajdowałaby się bliżej użytkownika i byłaby do jego dyspozycji dla opracowywania takich programów, jakie mu są rzeczywiście potrzebne.

Ale problem ten jest bardzo rozległy i nienowy. Istniał on nawet jeszcze zanim pojawiło się przetwarzanie rozproszone.Pro-

blem, gdzie ma się znajdować interface między przetwarzaniem danych a użytkownikiem, to stary i bardzo trudny problem. Czy są jakieś pytania dotyczące przyszłego rozwoju sieci?

NIEZIDENTYFIKOWANY MÓWCA: Chciałbym nieco inaczej sformułować pytanie. Chciałbym powiedzieć coś o złożoności. Czy SNA nie jest wynikiem tego, że IBM nie uczynił może wszystkiego co mógł, by ułatwić zestawienie różnych systemów? IBM stworzył tryb konwersacyjny, APL i wiele innych rzeczy. Ciągle ukazują się nowe systemy; za każdym razem oznacza to, że musimy mieć X różnych sieci. Czy nie sądzi Pan, że SNA dowodzi, iż rozwiązując szczególny problem bram wejścia w centrali, mamy - jeśli chodzi o całą resztę - nadal do czynienia z tym samym tradycyjnym, starym problemem, to znaczy z potrzebą nowych sieci? W końcu powstaje prawdziwe monstrum. Czy IBM nie mógłby raz na zawsze rozwiązać problemu dostępu do komputera?

DAVID: Oto odpowiedź. Przetwarzanie na bieżąco, wielodostęp i zdalne wprowadzanie zadań to trzy systemy odpowiadające trzem różnym grupom problemów. Użytkują one linie w różny sposób, obciążają linie w różny sposób, odpowiadają trzem różnym potrzebom. Te rozwiązania już istnieją; zostały utworzone dla zaspokajania określonych potrzeb.

Bez odpowiedniej architektury nie możemy pozwolić określonemu punktowi na danej linii robić coś innego niż robi inny punkt na tej samej linii. Gdy jeden punkt dokonuje zdalnego wprowadzania zadań, nie możemy mieć na tej samej linii innego punktu, który by równocześnie pracował w trybie wielodostępu. Z fizycznego punktu widzenia można oczywiście transmitować dane, ale trudno jest powiedzieć: "A teraz ja rozpoczynam swoją sesję zdalnego wprowadzania zadań", wówczas gdy na tej samej linii ktoś inny posługuje się nią dla użytkowania komputera w trybie konwersacyjnym. Jest to problem architektury i nie bardzo widzę, jak można by przezwyciężyć ten problem nie posługując się koncepcją sesji, czyli koncepcją łączenia każdego z każdym innym.

Oczywiście zgadzam się z Panem, że dzisiejsze trudności wynikają być może z przyczyn historycznych. Mimo to niektórzy

ludzie w IBM-ie uważają,że architektura SNA jest czymś ważniejszym od architektury komputerów serii 360; innymi słowy, że SNA otwiera znacznie szersze możliwości, niż te jakie daje sam komputer 360. Oczywiście komputer 360 został tymczasem zastąpiony przez model 370,choć ten ostatni nie stanowi może tak wielkiego postępu. Ale SNA rzeczywiście bardzo rozszerza zakres naszych możliwości.

Na krótką metę musimy patrzeć na nasze problemy z dużą dozą realizmu. Potrzebny nam jest SNA czy nie? Są ludzie, którym już dziś potrzebny jest SNA; innym nie jest potrzebny. Ale SNA to w gruncie rzeczy szczyt góry lodowej, wskazuje na narodziny czegoś, co będzie rosło i znacznie rozwijało się w przyszłości. Kiedy to wszystko nastąpi, to zupełnie inna sprawa. Ale na zakończenie pragnę zaznaczyć, że częściowo zgadzam się z tym, co Pan powiedział.

NIEZIDENTYFIKOWANY MÓWCA: Moje pytanie jest znacznie prostsze. Wskazał Pan bardzo wyraźnie na różnicę między inteligentnym urządzeniem końcowym a minikomputerem. Wyposażenie urządzeń końcowych w pamięć nie jest niczym nowym. Znane są lampy oscyloskopowe i pamięci buforowe. Mówi Pan,że pojemności pamięci końcowych wzrosną do 100 K. Dobrze. Ale czy to znaczy, że takie urządzenie końcowe jest naprawdę inteligentne? Chciałbym usłyszeć, jakie są zamiary IBM-u.Czy można by wyposażać te urządzenia końcowe w jakiś rodzaj inteligencji, która ułatwiłaby życie użytkownikowi i programiście? Jaki rodzaj inteligencji zamierza IBM przenieść z komputera centralnego do tak zwanego inteligentnego urządzenia końcowego? Czy znaczy to, że zamierzacie opracować systemy operacyjne dla tych inteligentnych urządzeń końcowych? Czy zechciałby Pan powiedzieć nam coś o poglądach IBM-u?

DAVID: Nie sądzę, by IBM w ogóle myślał o tworzeniu systemów operacyjnych dla urządzeń końcowych. Gdyby nawet miały powstać takie systemy operacyjne,byłyby mikroprogramowane.Powie mi Pan, że to także nic nowego. Będą one, innymi słowy, integralną częścią sprzętu. Wspomniany przeze mnie 3790 ma 16 przyłączonych urządzeń końcowych i dlatego w grę wchodzi wieloprogramowość,

ale ta wieloprogramowość nie stanowi jeszcze systemu operacyjnego. W zasadzie ma Pan zupełną rację: dla ciężkich urządzeń końcowych będziemy może musieli stworzyć coś równoważnego z tym, co mamy w niektórych komputerach. Niewątpliwie istnieje jednak pewien pułap, ponad który nie zechcemy pójść. Trzeba być zawsze zupełnie pewnym, że pisane dla tych urządzeń końcowych programy są rzeczywiście mikroprogramami. Gdy mówimy, że wszystko co ma przebiegać na urządzeniu końcowym, powinno być mikroprogramem, stwarzamy sobie tę automatyczną samocenzurę. Mikroprogram musi oczywiście spełniać pewne kryteria niezawodności. Gdy zachodzi w nim błąd, jest podobny raczej do wady sprzętu. Myślę, że mikroprogramy będą jednym z czynników dynamiki systemu.

Traktujemy nasze urządzenia końcowe, jak klocki gry Lego[x/]; są to uniwersalne jednostki, które mogą być użytkowane przez wszystkie części systemu, a ostateczna funkcja, jaką będzie miało pełnić konkretne urządzenie końcowe będzie wyznaczona przez mikroprogram. Oznaczać to będzie, że uzyskamy znacznie większą elastyczność. Tak właśnie rozumiemy inteligencję, którą zamierzamy nadać urządzeniu końcowemu. Wie Pan, że występujemy na rzecz zasady rozproszonej inteligencji; inteligencję najłatwiej rozproszyć. Bardzo trudno mi jest powiedzieć coś więcej. Jeśli chodzi o to, jak daleko pójdziemy w rozpraszaniu funkcji, to nie zdradzając żadnych tajemnic firmowych mogę Panu powiedzieć, że bardzo rychło wynalezione zostaną środki operowania danymi na szczeblu urządzenia końcowego, APL i podobne. Ale nie sądzę, by kiedykolwiek ktoś posunął się tak daleko, by tworzyć masę skomplikowanego oprogramowania dla urządzenia końcowego poza przeniesieniem pewnych funkcji z komputera głównego do urządzenia końcowego. Na przykład IBM posiada system ATMS, który można stosować do wprowadzania dokumentów. Jest to operacja dość złożona, polegająca na wprowadzaniu pewnych ustępów i skreślaniu innych w punkcie wejścia. Jest to właściwie funkcja zbierania i wprowadzania danych, jakkolwiek w nieco wymyślny sposób; funkcja ta zostaje przeniesiona z komputera głównego do urządzenia końcowego.

---

x/ Lego - gra polegająca na układaniu różnych konstrukcji z klocków specjalnego kształtu.

AKOS /Politechnika Delft/: Pragnąłbym dorzucić coś do dyskusji między p.Duvergerem a pierwszym dyskutantem. Mam wrażenie, że IBM obawia się dokonania zdecydowanego kroku. Myślę, że IBM wciąż jeszcze upiera się przy wielkich centralnych komputerach i obawia się umieszczenia większej inteligencji w urządzeniach lokalnych, takich jak urządzenia końcowe i minikomputery. Dlatego IBM znowu powiększa swoje systemy liczące. Myślę, że SNA powinnaby zapewniać możliwość komunikacji przynajmniej między komputerami IBM-u. Jako użytkownik IBM-u chciałbym otrzymać od IBM-u prawdziwy system operacyjny sieci, umożliwiający prawdziwe przetwarzanie rozproszone; taka jest moja opinia.

DAVID: Powtórzę jeszcze raz, że IBM robi małe komputery, czyli minikomputery dla ludzi, którzy ich chcą; zawsze warto o tym pamiętać, nawet jeśli robimy głównie duże maszyny. Komunikacja między komputerami. No cóż, są techniki komunikacji międzykomputerowej bez bardzo wysoko rozwiniętej architektury, Gdy potrzebna nam jest bardzo szybka transmisja od jednego komputera do drugiego, może się to łatwiej odbywać poza architekturą. W ramach architektury jest to obecnie bardziej skomplikowane i oczywiście wymaga posiadania przez nas jakichś rozproszonych baz danych, rozmieszczonych w różnych komputerach, a to jest coś, co nie bardzo potrafimy dziś zrobić.

Ale wracając do pierwszej części Pańskiej wypowiedzi, a mianowicie, że boimy się rozproszyć niektóre funkcje ze względu na to, że są związane z bardzo wielkimi systemami liczącymi. Wcale nie pragniemy,by nasze komputery przekroczyły pewną wielkość, a to ze względu na skomplikowanie. Nie wiem, czy zdaje Pan sobie sprawę, co to znaczy utrzymywać w aktualnym stanie systemy operacyjne takie, jakimi są one obecnie, ale nawet w IBM-ie sprawia to wewnętrzne problemy. Oprogramowanie, które przekracza pewien rozmiar, jest nie zawsze niezawodne.Jest z nim jak z tym uczniem czarnoksiężnika: wymyka się spod kontroli. Myślę więc, że przy rozwijaniu informatyki trzeba zawsze pamiętać, że niepożądane są monstrualne rozmiary,a architektura SNA jest po prostu wynikiem pragnienia pewnej regulacji i kontroli w systemie, bo nie można bez końca powiększać rozmiarów.

Jestem wielkim zwolennikiem regulacji wewnętrznej.Nie sądzę, by można było wyjść poza określony stopień złożoności, bo wymyka się on spod naszej kontroli i trudno sobie z nim radzić. Użytkownicy również nie życzą sobie rzeczy nadmiernie skomplikowanych. Rozpowszechnianie się minikomputerów stwarza wszędzie bardzo skomplikowany typ struktury systemów,toteż gdy wkraczamy w nowe dziedziny, takie jak np. transmisja obrazów,nie będziemy mogli kontynuować tego procesu rozpraszania; będziemy musieli zatrzymać się i powiedzieć: "dobrze, a teraz resztę będziemy transmitować". Stadium małych zbiorów danych i małych elementów operacyjnych to może podejście bardziej nowoczesne,ale przynosi z sobą problemy, z którymi nie umiemy sobie radzić. Komplikuje to sprawy i dlatego potrzebna nam jest architektura sieciowa; przynajmniej ja tak sądzę.

HERSCHBERG /Unilever/: Nie zgadzam się z Panem, że mikroprogramowanie jest narzędziem gwarantującym prawidłowość. Skoro uważamy, że prawidłowość jest w pewnym sensie funkcją złożoności, wówczas nie sądzę, by mikroprogramowanie miało większe szanse prawidłowości niż cokolwiek innego. Jako dowód tego twierdzenia wymienię dwie rzeczy, z którymi mamy kłopoty, a które leżą na przeciwległych krańcach skali. Jest taki dość znany kalkulator kieszonkowy, który ma wadę oprogramowania i nie potrafi prawidłowo obliczyć arccosinusa zera. Jest to wada na poziomie pastylki integracji wielkoskalowej. Z drugiej strony, jak wiadomo - i sądzę, że będzie Pan pierwszym, który z tym się zgodzi - złożone systemy operacyjne, takie jak OS/VS, mają wysoką i niemal niezmienną liczbę błędów. Toteż mikroprogramowanie, stojące gdzieś pośrodku między nimi, zarówno pod względem złożoności jak i wykonania technicznego, nie wydaje się bardziej zabezpieczone przed tego rodzaju błędami niż te dwa przykłady, stojące na przeciwległych krańcach skali. Dziękuję Panu.

DAVID: Odpowiadam na to pytanie, które bardzo mnie interesuje ze względu na ów problem niezawodności i dokładności w podanym przez Pana przykładzie awarii. Mamy zawsze awarie i oczywiście mamy problemy. Wiemy jak bywa z oprogramowaniem. Zdarzają się błędy w rozkazach; niezależnie od rodzaju przeprowadzonych tes-

tów, zawsze gdzieś trafi się błąd. Mikroprogramowanie stoi gdzieś pośrodku między tymi dwoma. Jaką ma szansę większej niezawodności niż wyżej wspomniany sprzęt? Szansa taka istnieje, bo mamy w tym przypadku do czynienia z zupełnie prostym oprzewodowaniem. Funkcje tworzone są przez mikroprogram,a zatem mamy bardzo prosty system. Jest więc szansa podniesienia niezawodności w zakresie owych wspomnianych przeze wąskich gardeł.

Jest to rodzaj pseudooprogramowania. Przypuśćmy, że mamy błąd, błąd w konstrukcji. Jest to błąd produkcyjny; taki błąd może zaistnieć równie dobrze w sprzęcie stojącym na dolnym jak i na górnym krańcu skali. Są pewne techniki prac rozwojowych, techniki hierarchii i kodowania, dzięki którym, gdy zdarza się wadliwy element w konstrukcji,to znaczy element nie spełniający swej funkcji lub dający nieścisłe wyniki, jego funkcje może przejąć jakiś inny element lub któraś inna część i dzięki pomysłowej metodzie kontynuować pracę, w sposób co prawda pogorszony, ale tym niemniej kontynuować pracę i zapewnić diagnozę.

Dzisiejsza architektura systemów z urządzeniami końcowymi, również architektura typu SNA, daje takie możliwości. Mamy tego rodzaju mikroprogramy i gdy zachodzi błąd w funkcji, jest to oczywiście niepożądane, jednak nie tracimy tej funkcji; mamy ją nadal, jakkolwiek pogorszonej jakości. Są pewne typy oprogramowania działające w ten sposób i nie jest to nic nowego.Oprogramowanie opracowane w latach 1960-1965 zapewniało zerowanie oraz odpowiednią do tego architekturę. Przy mikroprogramowaniu jest to niezbędne, bo podczas gdy błąd obwodu jest łatwy do naprawy, bo można obwód szybko wymienić, błąd konstrukcyjny w mikroprogramowaniu jest sprawą bardziej poważną, bo trzeba wówczas albo ściągnąć na miejsce konstruktora albo skontaktować się z laboratorium.I rzeczywiście w niektórych przypadkach tak się dzieje.

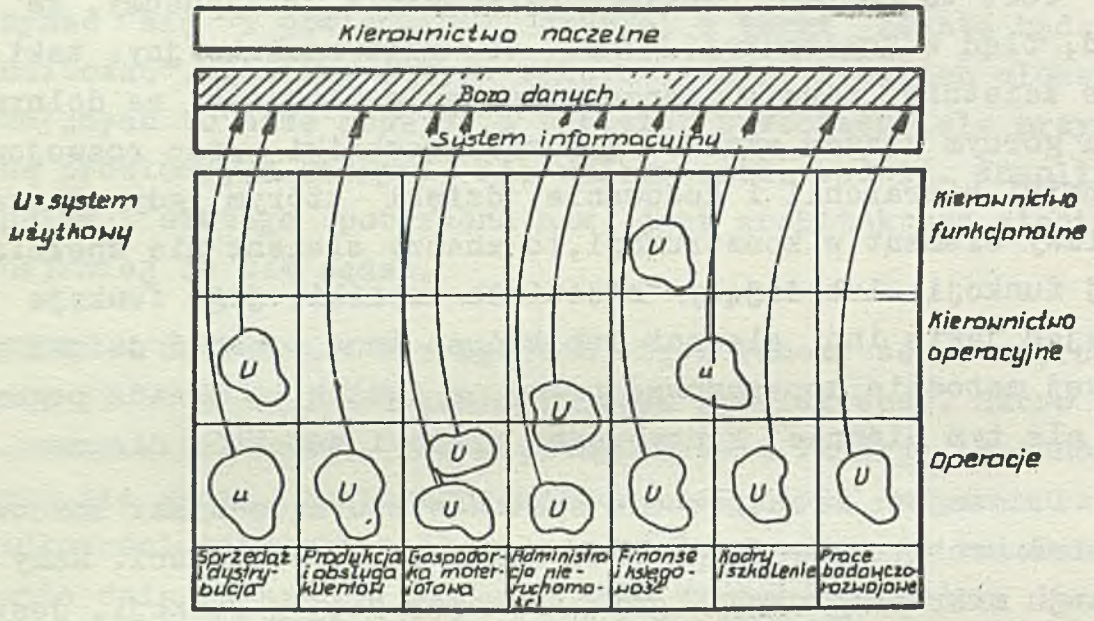
Wiem, że nie jest to pełna odpowiedź na Pana pytanie, ale mikroprogramy opracowuje się dziś metodami, które są właściwe programowaniu wielkoskalowemu. Te metody pozwalają nam cofnąć się oraz wyśledzić i rozpoznać błąd.

Ilustracja F 1

Obecny stan

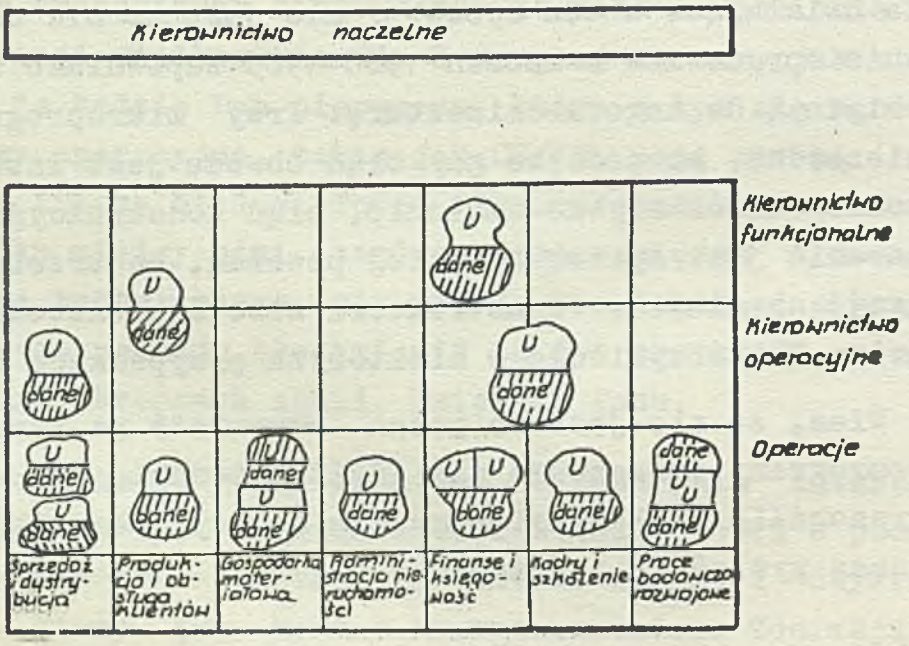
- rozwój techniki w tempie wykładniczym
- ekonomika
- aspekty społeczne
- złożoność
- udział człowieka w „pętli"

Ilustracja F 2



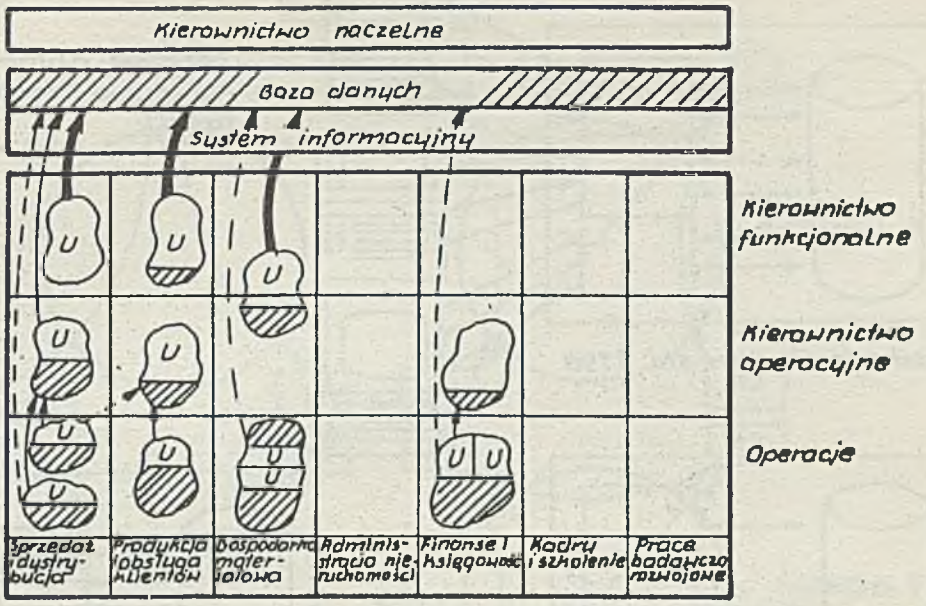
Zintegrowane systemy użytkowe i scentralizowane przetwarzanie danych

Ilustracja F 3



Oddzielne systemy użytkowe i przetwarzanie lokalne

Ilustracja F 4



Zintegrowane lub oddzielne systemy użytkowe z przetwarzaniem rozproszonym

Ilustracja F 5

PRZETWARZANIE SCENTRALIZOWANE

Zgrupowanie głównych środków informatycznych w jednym miejscu pozwala na scentralizowane zarządzanie przedsiębiorstwem. W razie potrzeby może, dzięki urządzeniom końcowym, zapewnić środek dostępu dla odległych użytkowników

Ilustracja F 6

PRZETWARZANIE LOKALNE

Zbiór "elementów przetwarzania informacji" rozproszony w różnych miejscach w celu zapewnienia priorytetu zaspokajaniu lokalnych potrzeb przedsiębiorstwa. Elementy te mogą, w razie potrzeby, wymieniać między sobą dane zbiorcze.

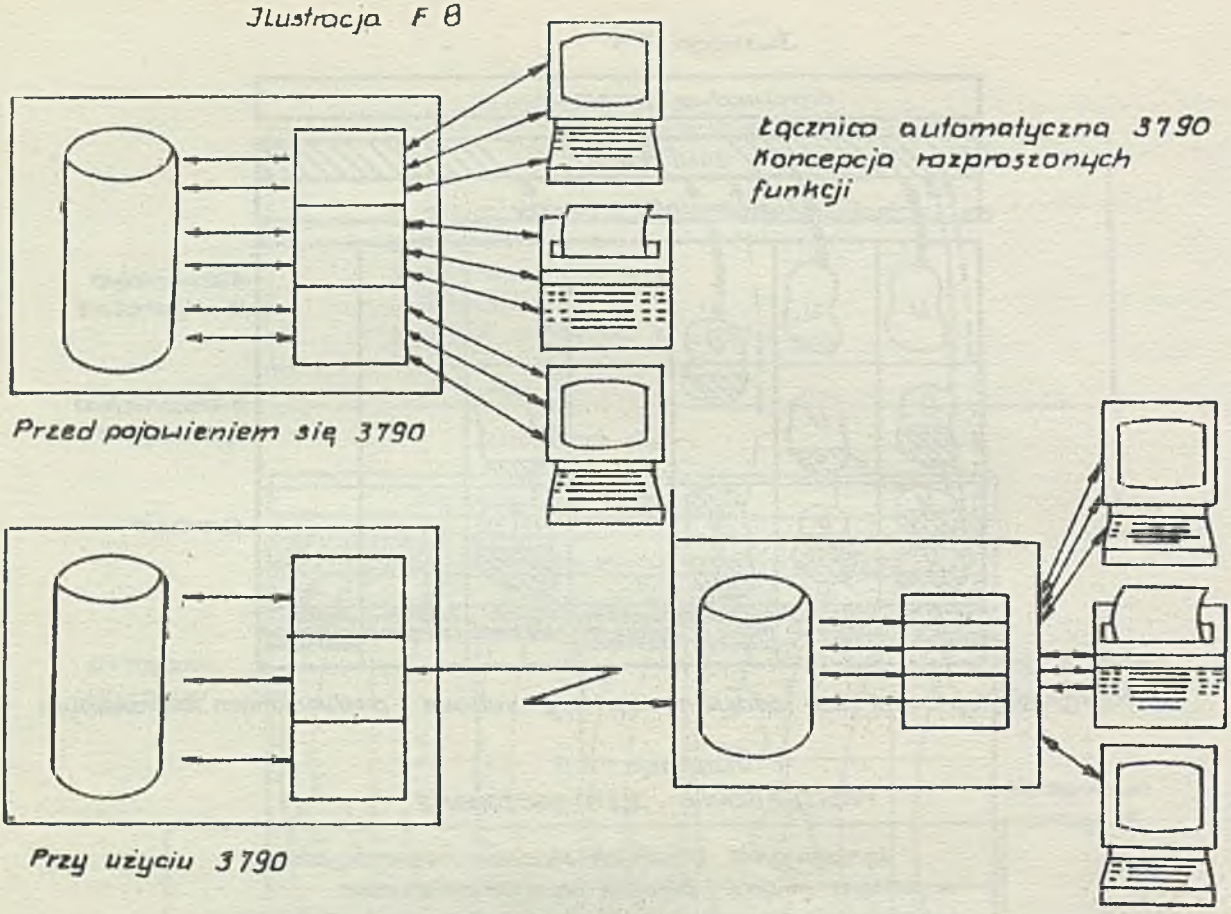
Ilustracja F 7

PRZETWARZANIE ROZPROSZONE

Zbiór, ewentualnie hierarchiczny, "elementów przetwarzania informacji" rozproszonych w różnych węzłowych punktach systemu informacyjnego przedsiębiorstwa. Elementy komunikują się wzajemnie za pomocą ujednoliconej architektury w celu osiągnięcia globalnego i spójnego zarządzania.

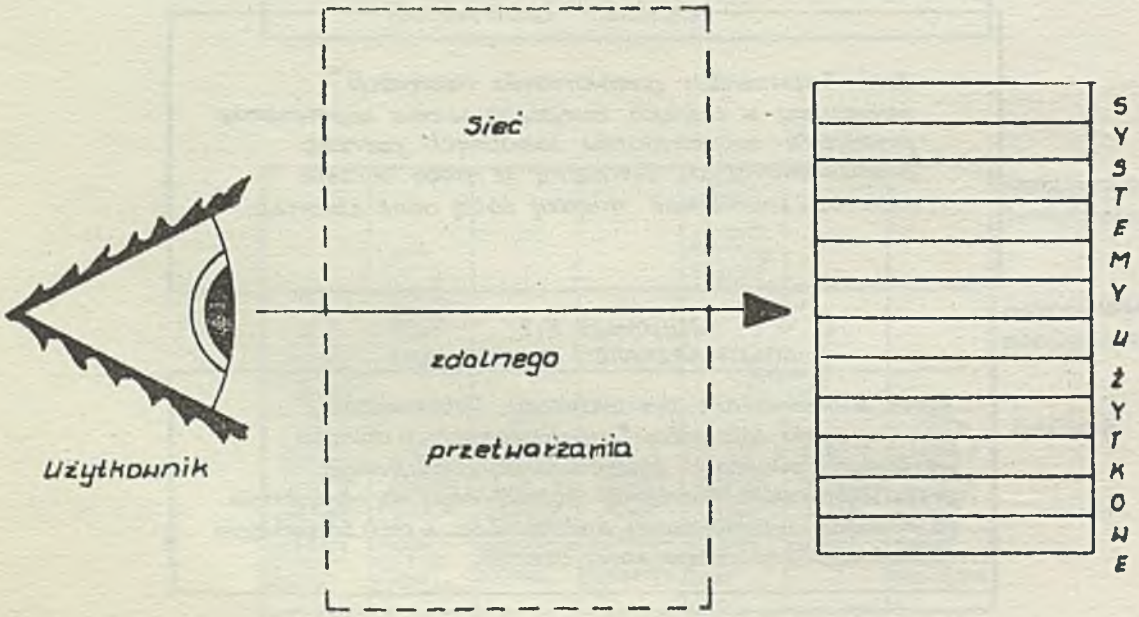
Ilustracja F 8

Łącznica automatyczna 3790
Koncepcja rozproszonych funkcji

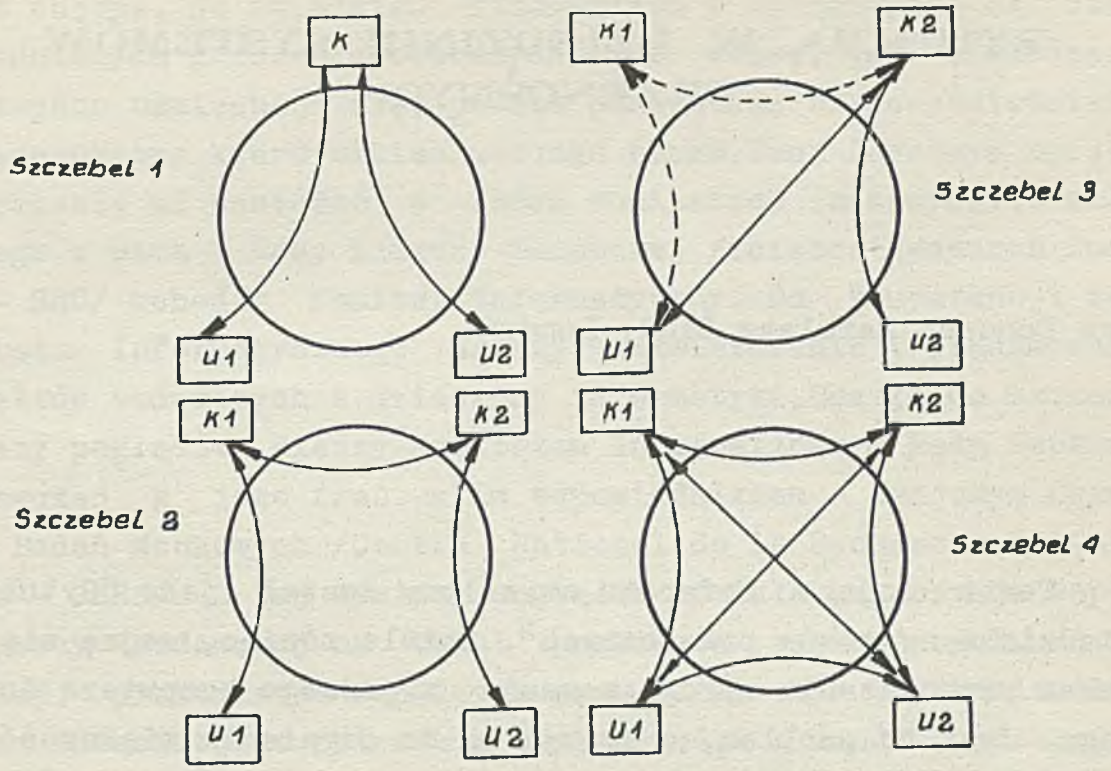


Ilustracja F 9

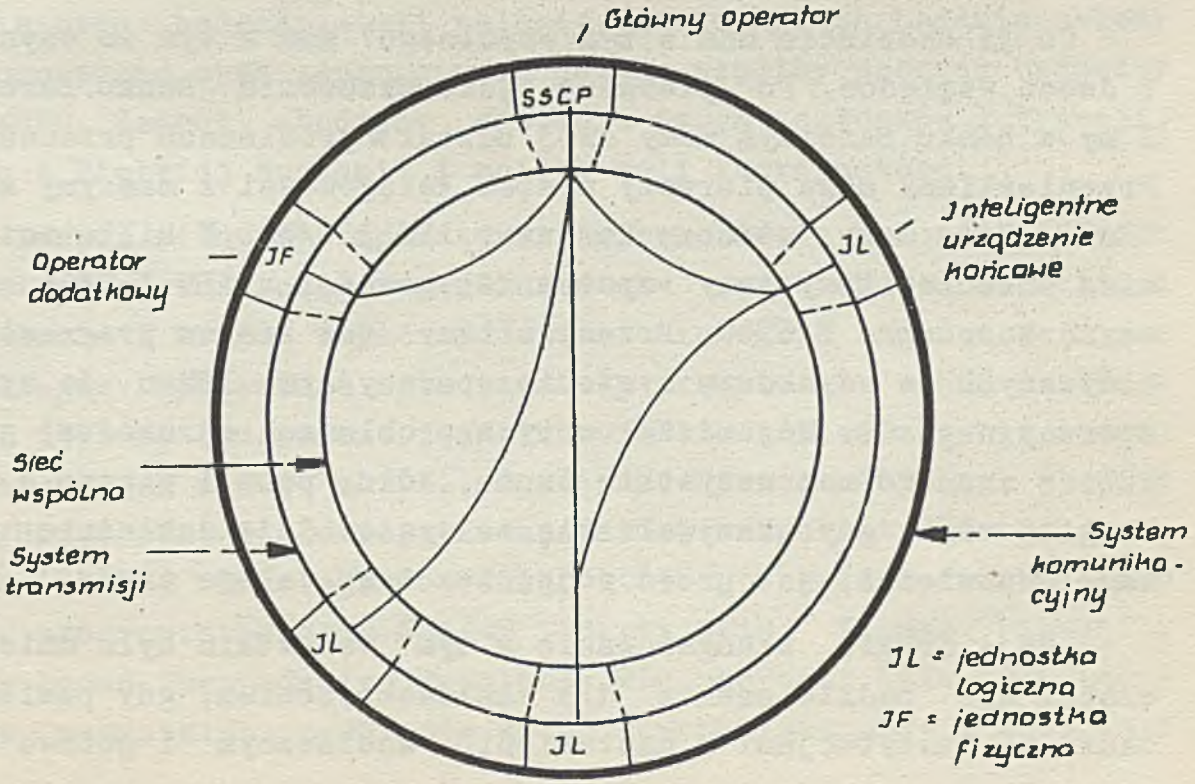
" Niewidzialność " dla użytkownika



Ilustracja F 10



Ilustracja F 11
Sieć jednostek adresowalnych



SESJA G

SYTUACJA W DZIEDZINIE SYSTEMÓW PRZENOŚNYCH

Alan Duncan, Barclays Bank, Londyn

Temat mojego referatu określony został jako "Sytuacja w dziedzinie systemów przenośnych". Ściśle mówiąc, zajmę się problemem przenoszenia oprogramowania z jednego komputera do drugiego. Jest to problem, z którym ma do czynienia większość użytkowników, toteż gdy przemnożymy ponoszone nakłady przez liczbę wszystkich użytkowników w krajach europejskich, stwierdzimy, że stanowi to duży problem ekonomiczny. Powszechnie nazywa się ten problem "przenośnością". A ściślej mówiąc problemem jest nie tyle przenośność ile jej brak.

Co ja osobiście mam z tym wspólnego? Mam z tym do czynienia z dwóch względów. Po pierwsze jako pracownik Banku Barclays. I my w Banku Barclays mamy swój udział w problemach przenośności. Przenieśliśmy nasz pierwszy system księgowości z maszyny zwanej "Emidec" na nowy wówczas komputer IBM 360/30. W kilka lat później przenieśliśmy inny system księgowości z IBM 360/65 na maszynę Borroughs B 6700. Przenieśliśmy też zestaw programów wykonywanych w dyskowym systemie operacyjnym IBM-u do systemu operacyjnego OS. Mój udział w tych problemach sprowadzał się do dobrze znanego nam wszystkim trudu, bólu, potu i płaczu,a potem błogiej ulgi, gdy okazywało się,że system daje dokładnie takie same odpowiedzi, jak przed podjęciem tego całego trudu.

Moje drugie zaangażowanie w tym wszystkim było mniej męczące. Nie będzie pewnie dla Was zaskoczeniem, gdy powiem, że banki są instytucjami o nastawieniu społecznym i gotowe są, a

nawet chętne, do świadczeń finansowych i usługowych na rzecz uzasadnionych potrzeb społecznych. Już widzę, jak niektórzy z Was kwaśno uśmiechają się, pewnie porównując to co powiedziałem ze sknerowatym kierownikiem swojego banku. Bank Barclays uprzejmie pozwala mi zasiadać w dwóch komitetach rządowych. W skład jednego z nich - Rady Naukowo-Badawczej /Science Research Council - SRC/ wchodzi Komitet Informatyczny. Do kompetencji tego Komitetu Informatycznego należy zatwierdzanie i finansowanie projektów badawczych z dziedziny informatyki. Doszło do szczerej wymiany poglądów między Komitetem Informatycznym Rady Naukowo-Badawczej a jego francuskim odpowiednikiem - Krajowym Ośrodkiem Badań Naukowych /Centre National de la Recherche Scientifique - CNRS/. Z tej wymiany poglądów wynikła propozycja wspólnej pracy badawczej na temat problemów przenośności. Mówiąc ściślej, przedmiotem wspólnych prac miało być uzyskanie ilościowej informacji o potrzebie przenośności w handlu, przemyśle, szkolnictwie wyższym, przemyśle komputerowym i w biurach programistycznych, przybliżona ocena ekonomicznej wartości ułatwienia przenośności oraz rozeznanie narzędzi i metod już dostępnych lub będących w toku opracowania, a także zbadanie i ocena obecnego praktycznego zastosowania tych narzędzi i metod. Na zakończenie prac badacze mieli zalecić, jakie dalsze badania należy przeprowadzić oraz zaproponować dalszy wspólny program badawczy CNRS/SRC. Zespół studyjny obejmuje dwóch członków z Francji, dwóch z Wielkiej Brytanii i mnie w roli koordynatora.

Zespół studyjny sporządził sprawozdanie złożone z dwóch części. Pierwsza część wspólna w obu językach - francuskim i angielskim i druga część podająca szczegóły badań w każdym z obu krajów. Sprawozdania te zostały przedłożone CNRS i SRC w marcu 1975 r. Zamierzone jest ich opublikowanie, ale nie sądzę, by były już w tej chwili dostępne. Równocześnie ze studium CNRS/SRC, również Komisja Europejskiej Wspólnoty Gospodarczej zorganizowała równoległe badania, które powierzono francuskiemu biuru programistycznemu Sogeti i brytyjskiej firmie Dataskil. Celem badań Europejskiej Wspólnoty Gospodarczej była inwentaryzacja wszystkich problemów krępujących wymienność i zalecenie

wszelkiego rodzaju działań ułatwiających przenoszenie systemów użytkowych z jednego sprzętu na drugi lub ich równoczesne wykonywanie na kilku różnych typach sprzętu. Sprawozdanie z tego studium przedstawione zostało w lipcu 1975 r. Praca ta przyczyniła się do sformułowania polityki Wspólnoty w dziedzinie informatyki, wyszczególnionej w dokumencie COM /75/ 467 z września 1975 r.

Ankieta EWG przeprowadzona została za pomocą kwestionariusza i "wywiadów otwartych" /według określenia grupy roboczej/. Nie bardzo wiem, co to jest "wywiad otwarty"; przypuszczam, że znaczy to, iż ankieter nie został zaproszony do wnętrza zakładu i musiał wypełnić formularz na schodach frontowych. Poszczególne kraje EWG zostały uwzględnione w sposób "ważony". I tu również nie bardzo wiem, co znaczy!

Studium CNRS/SRC dotyczyło tylko Wielkiej Brytanii i Francji, ale było bardziej osobiste, mniej komercjalne. Ankieterów zapraszano do środka i na ogół nawet częstowano winem.

Wielkość próbki była w obu studiach mniej więcej taka sama - około 20 firm w każdym kraju.

Obie grupy nie porozumiewały się między sobą i podobno, gdy na pewnym etapie prac zdarzyło się, że przebywały w tym samym hotelu, piły na przeciwległych końcach baru. To ja miałem przyjemność powiązania ich prac.

Nie mam zamiaru wchodzić we wszystkie szczegóły, zakładam bowiem że sprawozdania albo już są albo wkrótce będą dostępne i będziecie Państwo mogli je sami przeczytać. Postaram się jedynie wyciągnąć z nich główny sens praktyczny. Co najmniej czterech na każdych pięciu z Was miało już kiedyś do czynienia z poważną zmianą sprzętu, systemu operacyjnego lub języka, wiem zatem, że stoję przed audytorium pełnym zrozumienia.

Przede wszystkim zdajmy sobie sprawę z istniejącej sytuacji i pozwólcie, że wprowadzę Was w koncpecję "szczebli niewymienności".

Między różnymi systemami użytkowymi, między sprzętem i oprogramowaniem dostarczanym przez różnych producentów, a także

i między sprzętem jednego producenta istnieją niewymienności. Skutki tych niewymienności mogą przejawiać się w trzy sposoby: przeniesione zadanie w ogóle nie może przebiegać w nowych warunkach - tak bywa najczęściej; czasem - bardzo rzadko - może przebiegać, choć ze zmniejszoną szybkością; wreszcie może przebiegać, ale wydaje inne dane wyjściowe.

Jak do tego doszło? Po prostu dlatego, że komputery były opracowywane równolegle w wielu miejscach i to w warunkach konkurencji.

Jak doniosły jest problem przenośności? Program badawczy CNRS/SRC spróbował uzyskać jakąś ocenę kosztu przeniesienia systemu na nowy sprzęt, ale bez większego powodzenia. Dwie są tego przyczyny:

. niełatwo jest obliczyć koszty przeniesienia,
. ludzie niechętnie mówią o nakładach tego typu.

Ale nie ulega wątpliwości, że traci się na to wiele wysiłku. Pożyteczną ocenę otrzymano od pewnej firmy amerykańskiej. Ocenia ona, że w Stanach Zjednoczonych wydaje się obecnie 3 miliardy dolarów na przeniesienie systemów w nowe środowisko.

W Wielkiej Brytanii oceniono, że dla klientów ICL-u koszty konwersji do "nowej serii" ICL-u wyniosą w przybliżeniu 800 milionów funtów. Sprawozdanie EWG podaje te liczby w "jednostkach obliczeniowych"; jest to waluta, której nie wypłaci Wam żaden bank i wartość jej zmienia się wraz z ciśnieniem barometrycznym. Studium EWG przytacza ocenę, że suma wydatkowana na przenoszenie systemów wynosi około 10% pierwotnej inwestycji, ale zaraz potem zastrzega się, że nie bierze odpowiedzialności za tę cyfrę.

Dr Brown z Uniwersytetu Kent, członek zespołu studyjnego SRC, zauważył, że gdybyśmy potrafili zaoszczędzić choćby mały procent od tej wielkiej kwoty, to i tak opłaciłoby się to. Oba zespoły studyjne poczyniły oczywiście zalecenia jak zaoszczędzić parę centów tu i parę centów tam. Ale kto odczuje efekt tych oszczędności? W skali ogólnospołecznej dadzą one pewien efekt, ale dla indywidualnego przedsiębiorstwa te groszowe osz-

czędności w milionowym budżecie na konwersję są bez znaczenia i mieszczą się w granicach błędu w kosztorysie. Nie chcę wdawać się w dyskusję filozoficzną, ale rządy ze swej strony tak przyzwyczaiły mnie do miliardowych budżetów,że już wcale nie odczuwam entuzjazmu na myśl o kilkuset milionach w tę lub ową stronę dla Wspólnoty. Interesuję się tylko oszczędnościami, mającymi wpływ na moje najbliższe otoczenie.

Są dwie informacje, które trzeba wziąć pod uwagę przy rozpatrywaniu przenośności. Pierwszą z nich jest,że udział procentowy prac programistycznych skierowanych na nowe opracowania zmniejsza się.

Drugą informacją jest fakt, że nakłady na oprogramowanie wzrastają w porównaniu z nakładami na sprzęt, co czyni użytkowników mniej skłonnymi do przenoszenia systemów. Sprawozdanie CNRS/SRC podało klasyfikację problemów przenośności,co nie znalazło echa ze strony zespołu studyjnego EWG,prawdopodobnie dlatego, że zespół CNRS/SRC miał nastawienie bardziej akademickie. Dostrzegł on różnicę między tym, co można by z grubsza zaklasyfikować jako zadania informatyczne typu handlowego, a zadaniami typu naukowego. Kilka przykładów zadań typu naukowego dopomoże w objaśnieniu tego. Przykłady te pochodzą z firm ankietowanych przez zespół studyjny CNRS/SRC.

Jest w Wielkiej Brytanii grupa stanowiąca obecnie samodzielne przedsiębiorstwo pod nazwą NAG. Muszę szybko wyjaśnić, co oznaczają te inicjały, zanim nasi świetni i szybcy tłumacze podadzą,że słowo to w angielskim znaczy "kucyk" lub "kobieta lekkich obyczajów". NAG oznacza Numerical Algorithms Group.

Celem NAG jest udostępnianie biblioteki algorytmów numerycznych wyższym szkołom i innym organizacjom. Firma NAG jest szczególnie interesująca,ponieważ przenośność jest główną racją jej istnienia. Algorytmy są dostarczane dobrowolnie i bezpłatnie przez licznych dawców.Personel NAG ma siedzibę w centralnym biurze w Oksfordzie, gdzie przechowywana jest biblioteka algorytmów.Biblioteka ta jest wykonywana na różnych seriach maszyn, szczególnie tych, które użytkowane są w brytyjskich szkołach wyższych. Dla każdej serii maszyn istnieje oddzielny zespół

wdrażający - obecnie jest takich zespołów dziewięć. NAG dostarcza pracowników do koordynowania pracy tych zespołów, ale znaczną część pracy wdrożeniowej wykonują same szkoły wyższe. Zadaniem NAG jest więc stwarzanie kontaktu między dawcami algorytmów a tymi, którzy tych algorytmów potrzebują. NAG zatrudnia około tuzina pełnoetatowych wykwalifikowanych pracowników i w dużej mierze opiera się na współpracy ze specjalistami spoza organizacji na zasadzie prac zleconych.

Biblioteka NAG jest dość duża /około 300 algorytmów plus programy testowe, łącznie blisko 100 000 wierszy kodu/.Gdy mowa o przenośności biblioteki NAG, należy pamiętać, że algorytmów nie można uważać za samodzielne pakiety programowe; algorytmy NAG przeznaczone są do włączenia do programów użytkownika. Dlatego NAG nie może swobodnie wybierać języka programowania w jakim mają być wydawane algorytmy. Jest natomiast zmuszony do dostarczania biblioteki w językach stosowanych obecnie przy pracy numerycznej. Toteż postanowiono wydawać algorytmy w FORTRANie IV i ALGOLu 60.

Stosowanie standardowego FORTRANu IV i ALGOLu 60 nie pozwala na uzyskanie pełnej przenośności. Zwykle trzeba trzech roboczolat na wdrożenie biblioteki po raz pierwszy na nowej serii maszyn. Trudności wynikają z różnic sprzętu /szczególnie różnej dokładności wykonania/ i różnic między kompilatorami. W pierwszym okresie istnienia biblioteki różnice między sprzętem stanowiły duży problem, ale obecnie zostały one w znacznej mierze przezwyciężone dzięki odpowiedniej parametryzacji algorytmów. Teraz głównym problemem są różnice oprogramowania.

Od NAG grupa studyjna otrzymała kilka ciekawych cyfr na temat przenośności programów naukowych. Dla pięciu serii maszyn /dla czterech tylko w przypadku ALGOLu/ stwierdzono, że 85% instrukcji FORTRANu i 80% instrukcji ALGOLu jest wspólne dla wszystkich serii.Cyfry te podlegają oczywiście całemu szeregowi "jeśli" i "ale", mimo wszystko jednak dowodzą, że FORTRAN i ALGOL są co najmniej znośnymi narzędziami do przenoszenia programów naukowych.

Drugim przykładem, na który chciałbym zwrócić Waszą uwagę, jest brytyjska firma pod nazwą Genesys Ltd., której nie należy mylić z kalifornijską firmą o tej samej nazwie.

Genesys Ltd. jest firmą,która utrzymuje i sprzedaje system GENESYS, będący zbiorem pakietów programowych dla inżynierów budowlanych. System GENESYS biegnie obecnie na około 60 różnych komputerach, obejmujących maszyny siedmiu różnych producentów i 16 różnych systemów operacyjnych.Firma Genesys rozpoczęła działalność jako organizacja finansowana przez rząd w celu zapobiegania dublowaniu pakietów programowych dla budownictwa.Niedawno stała się niezależnym przedsiębiorstwem. Ma siedzibę w Loughborough i zatrudnia około 30 pracowników. Na pierwszy rzut oka Genesys Ltd. jest podobna do NAG, ale zachodzą między nimi 4 istotne różnice:

1. Ponieważ Genesys jest handlowym przedsiębiorstwem działającym w dziedzinie budownictwa, kontrola nad dawcami i użytkownikami jest dużo ściślejsza. Społeczność NAG jest znacznie mniej zdyscyplinowana.
2. NAG zajmuje się algorytmami, które zwykle są włączane do cudzych programów.
3. Programy GENESYS o podobnym charakterze łączone są w komplety. /Jest 20 takich kompletów i niektóre z nich obejmują blisko 50 programów./ Dlatego komunikacja między programami jest w tym przypadku ważną sprawą.
4. Programy GENESYS obejmują zwykle duże ilości danych.

Użytkownik kompletu programów GENESYS zwykle wywołuje kolejno szereg programów, przy czym każdy program przekazuje dane następnemu. Normalnie każdy program wymaga danych w postaci tablic. Dla osiągnięcia niezależności od maszyny Genesys Ltd. zaprojektowała swój własny uniwersalny język do wywoływania programów; język ten jest powiązany z "językiem opisu danych" służącym do opisu wejścia i wyjścia. Wszystkie programy GENESYS wykonywane są w specjalnym systemie operacyjnym GENESYS obsługującym ich wymagania. W szczególności ma on swój własny system rejestracji i własny program ładujący, który zajmuje się powiązaniami międzyprogramowymi.

Przenośność systemu GENESYS opiera się na FORTRANie ANSI. Jednakże FORTRAN sam przez się nie zaspokaja w pełni potrzeb systemu. Nie tylko istnieje oczywisty problem różnic między kompilatorami,ale FORTRAN nie uwzględnia pamięci wirtualnej ani pól dynamicznych i jego funkcje służące komunikacji międzyprogramowej są nieporęczne. Toteż Genesys Ltd. zaprojektowała swój własny język dla realizacji systemu. Nazywa się on GENTRAN i jest w gruncie rzeczy superzbiorem FORTRANu. Typowy program w GENTRANie składa się z 75% FORTRANu i 25% uzupełnień.

Przykłady te ilustrują fakt, że przenośność programów naukowych jest wykonalna bez dużych nakładów.

W pierwszym przykładzie algorytmy NAG są w istocie rzeczy tym, co dawniej nazywano podprogramami standardowymi /subroutines/. Gdy napisane są w FORTRANie lub ALGOLu, są wysoce przenośne.

Z kompletami inżynierskimi GENESYS sprawa nie jest tak prostą. W porównaniu z handlowym przetwarzaniem danych te programy i zbiory są stosunkowo małe. Ale żeby uczynić te komplety przenośnymi, Genesys Ltd. musiała stworzyć specjalne oprogramowanie zarządzające. Mimo tych trudności, okazało się to rzeczą zupełnie wykonalną.

Terminem często stosowanym w dziedzinie przenośności jest "przewidywanie", to znaczy pisanie programów z uwzględnieniem ich późniejszej przenośności, w przeciwieństwie do "korekcji". Obecnie w większości przypadków przeniesienie wiąże się z korekcją.

Zupełnie inaczej ma się rzecz w dziedzinie handlowego przetwarzania danych.

Skoro godzimy się z rzeczywistością, że zachodzą różnice między sprzętem i oprogramowaniem różnych producentów, a nawet tego samego producenta,to skąd bierze się problem przenośności? Skąd się bierze pragnienie przenoszenia systemów z jednego środowiska do drugiego? A oto najpospolitsze przyczyny.

Pierwsza: postanawiamy zmienić dostawcę komputera, bo nie jesteśmy zadowoleni z dotychczasowego dostawcy lub dlatego, że

nowy ma coś specjalnego do zaoferowania. W toku mych badań spotkałem przelotnie człowieka, który powiedział mi, że skłonność do zmiany dostawców pozostaje w danym kraju w ścisłej proporcji do częstotliwości rozwodów w tym kraju. Ciekawa uwaga. Głęboko zamyśliłem się nad tą perłą mądrości, ale gdy chciałem poprosić o bliższe wyjaśnienia, człowiek ten już poszedł sobie.

Druga przyczyna: zmuszeni jesteśmy przez producenta do zmiany systemu operacyjnego ze względu na przestarzałość dotychczasowego systemu albo ze względu na nasze pragnienie zastosowania nowego urządzenia peryferyjnego, które nie może działać w starym systemie operacyjnym. Jest to nieuczciwy chwyt stosowany przez niektórych producentów.Albo też chcemy wykonywać programy na sprzęcie rezerwowym, który normalnie działa w innym systemie operacyjnym.

Trzecia przyczyna: chcemy zastosować cudzy pakiet użytkowy, który robi niezupełnie to czego sobie życzymy i biegnie na niezupełnie takim samym sprzęcie.

Takie są trzy główne przyczyny i wyglądają bardzo różnie, w zależności od tego, czy jesteście wielką firmą komputerową, biurem programistycznym czy użytkownikiem /który zawsze w końcu płaci za wszystko/. Bo z jakiejkolwiek przyczyny chcielibyśmy zmienić warunki działania, jedno jest zawsze prawdziwe, a mianowicie to, że liczone w roboczolatach lub pieniądzach nakłady inwestycyjne na system, który wymaga przeniesienia, są duże w porównaniu z wielkością naszych operacji. Pewne jest także, że jeśli nawet przeszliśmy już kiedyś przez coś podobnego, nigdy nie jesteśmy w tym ekspertami. Nie ma wygodnych, dokładnych podręczników,jak to robić; trzeba zaryzykować i trzymać się utartej praktyki.

Rozważmy przypadek, gdy przechodzimy od jednej maszyny do drugiej albo tego samego albo innego producenta. Są dwa możliwe sposoby przejścia - konwersja lub emulacja. Konwersja oznacza przekształcenie, czyli przepisanie zbiorów dla umożliwienia ich przebiegu w nowym środowisku. W handlowym przetwarzaniu danych konwersja danych jest na ogół najbardziej pracochłonną częścią zadania, natomiast w dziedzinie naukowej stanowi ona niemal

drobnostkę. Konwersja opiera się w dużej mierze na istniejącej dokumentacji, ale daje okazję do zmiany i aktualizacji pierwotnych programów.

Emulacja oznaczała kiedyś "czarną skrzynkę" sprzętową,która sprawiała,że dla programów nowa maszyna "wygląda jak" dawna, a symulacja oznaczała konwersję programatyczną podczas przebiegu. Ale obecnie emulacja w potocznym tego słowa znaczeniu oznacza przebieg taki jak starego systemu, niezależnie od tego czy osiąga się to za pomocą sprzętu czy oprogramowania, czy też za pomocą tego co Honeywell nazywa "twardym oprogramowaniem technicznym" /hard firmware/.

Wszyscy główni producenci komputerów dostarczają pewnych pomocy, niektórzy nawet wielu pomocy. Przed przygotowaniem tego referatu napisałem do kilku z nich i poprosiłem o wykaz dostarczanych przez nich pomocy. Byli trochę zakłopotani, ale w końcu wszyscy odpowiedzieli mi. Dwaj poprosili, żeby nie powoływać się na nich, a jeden poprosił, by treści jego odpowiedzi nie ujawniać nikomu. Na jednym końcu skali mamy IBM. Byłem zaskoczony i zdumiony, jak rozległe są pomoce oferowane przez IBM. Obejmują one emulatory, np. funkcję emulowania komputera 1401 na komputerze 370/158, pakiety konwersji danych i konwertery z jednego języka do drugiego, np. z FORTRANu do PL/1. IBM ma też bibliotekę programów "użyczonych" /śmieszne, ale użyczonych w większości przez pracowników IBM-u/, umożliwiających użytkownikowi porzucenie jego dotychczasowej maszyny i przejście do maszyny IBM-owskiej. Biblioteka zawiera programy głównie symulujące i odpowiadające im programy konwertujące dane, np. program komputera IBM System/360 symulujący komputer GE 225.

Dbając bardzo o poprawność wobec konkurentów, IBM nie mówi o emulowaniu cudzego sprzętu, a programy symulujące są "użyczone" bibliotece. Niektóre inne firmy nie wykazują tak daleko posuniętej dbałości o zachowanie form.

I tak np. UNIVAC i Borroughs mają programy konwertujące, np. z IBM-owskiego RPG do ich własnego COBOLu. Mówią też wprost o emulowaniu maszyn IBM 1401 i IBM 360/20.

Ale w tym miejscu natrafiamy na bardzo ciekawe ograniczenie. Borroughs B 1700 był jedną z pierwszych "miękkich" maszyn, to znaczy nie miał stałego repertuaru rozkazów;widoczne rozkazy są w nim zmikroprogramowane. Może działać jako maszyna o 32-bitowej długości słowa i może być zmikroprogramowany tak,by symulować inną maszynę, powiedzmy IBM 360/30 lub 370/175.Można więc bez trudu emulować jakiś inny komputer, ale nie znaczy to, że można zupełnie swobodnie wziąć komplet programów wykonywanych na systemie liczącym jednego producenta i wykonywać je na innym z tymi samymi komunikatami pulpitowymi itd. Jeśli chodzi o poziom niewymienności na szczeblu sprzętu, wiąże się on głównie z urządzeniami peryferyjnymi, które nie są identycznie identyczne lub - bardzo rzadko - są uwarunkowane czasowo /time-dependent/, tak że nawet przy emulacji może być potrzebna pewna doza modyfikacji albo żeby umożliwić przebieg zadań, albo żeby podnieść wydajność.Należy podkreślić, że ograniczenia w stosunku do emulacji na szeroką skalę na "miękkich" maszynach nie są natury technicznej, ale prawnej! Istnieje między producentami rodzaj "gentelman's agreement", że nie należy emulować tych maszyn, których oprogramowanie systemowe nie jest jeszcze "powszechnie dostępne", co po prostu oznacza, że dozwolone jest emulowanie tylko przestarzałych systemów liczących.

Studium CNRS/SRC dostrzegło duże możliwości tkwiące w emulacji i maszynach "miękkich" i zauważyło, że możliwości te są często limitowane ograniczeniami prawnymi i sztucznymi przeszkodami technicznymi.

W studium EWG ten aspekt tematu pominięto. Nie twierdzę, że problemy prawne wynikające, gdy oprogramowanie jest użytkowane w drodze emulacji na maszynie innego producenta, są łatwe do rozwiązania, ale postęp techniczny wymaga, by uregulować te sprawy. Wchodzące w grę zasady są już próbnie stosowane przez producentów mini- i mikroprocesorów. Wiem na przykład, że Intersil produkuje i sprzedaje IM 6100,który "potrafi rozpoznawać repertuar rozkazów komputera DEC PDP 8/e";tak powiada producent i dodaje - cytuję - "dużą część oprogramowania napisanego już dla komputera DEC można bezpośrednio ładować do systemu licz-

cego IM 6100 i wykonywać na nim".Podjęcie tej problematyki leży w oczywistym interesie EWG. Powiedziałem poprzednio, że nikt z nas nie jest ekspertem, gdy zachodzi potrzeba przejścia od jednego środowiska do drugiego. I jest tak nawet, jeśli już kiedyś robiliśmy coś takiego. Producent sprzętu ma nieraz pewne narzędzia, którymi możemy się posłużyć, ale na ogół jesteśmy słabo wyposażeni do dokonywania transformacji systemów. Także przeciętne biuro programistyczne ma mało doświadczenia; dokonało kilku przepisań programów dla swoich klientów, ale nie ma dość zleceń tego rodzaju, by mogło opracować wszechstronny zestaw pomocy konwersyjnych lub standardową technikę konwersji. Istnieje tylko jeden znany mi wyjątek - firma Brandon Applied Systems Inc. Wyspecjalizowała się w konwersji, czy też - jak to nazywa - transformacji. Ma biura w Ameryce i w Wielkiej Brytanii. Jestem pełen uznania dla Dicka Brandona, który poznał się na handlowym znaczeniu przenośności i zrobił coś w tej sprawie. Firma Brandon podejmuje się już od sześciu lat zadań transformacyjnych w odniesieniu do wszelkiego typu systemów oprogramowania i maszyn.Jej usługi transformacyjne składają się z trzech części.

Pierwsza z nich to tzw. META-język.Jest to bardzo pomysłowy chwyt. Jest to język pośredni, który przerzuca most między tym od czego wychodzimy,a tym do czego chcemy dojść /ilustr. G1/.Zaletą META-języka jest to, że dla każdej nowej maszyny lub dla każdego nowego języka maszynowego wystarczy tylko napisać translator z tego nowego języka na META-język, a natychmiast można korzystać z całej biblioteki możliwości przejścia od META-języka do różnych innych języków, np. do IBM-owskiego PL/1. Tak samo, gdyby jakiś producent wystąpił z nowym językiem wyższego rzędu, potrzebny byłby tylko jeden translator z META-języka do tego języka wyższego rzędu, a natychmiast otworzyłby możliwość przejścia od dużej liczby języków do tego nowego języka.

Drugą częścią metodyki Brandona jest szerokie stosowanie narzędzi i pomocy programowych dla ułatwienia zadania transformacji. Firma posiada bibliotekę 150 takich narzędzi i pomocy. Te dwie pierwsze części metodyki są oczywiście niewidzialne dla klienta.

Trzecią i może najważniejszą częścią jest ustrukturowane i udokumentowane podejście do całego procesu. Jest to płaszczyzna styku z klientem i jego problemem.Firma Brandon traktuje usługi transformacyjne jako linię produkcyjną. Mam tu przeźrocze,które pokazuje w postaci macierzy zadania transformacyjne, jakich podjęła się ona.

Dotychczas mówiłem o emulacji i konwersji tak, jak gdyby można je było postawić na tej samej płaszczyźnie. Emulacja jest stosunkowo łatwa,natomiast konwersja jest niezwykle kłopotliwa. Udało mi się wypożyczyć od Brandona kilka przeźróczy ilustrujących złożoność konwersji. Pierwsze przeźrocze dotyczy ogólnej problematyki i wskazuje, że przede wszystkim wyłania się potrzeba pokierowania wielce złożonym projektem. Pragnienie przejścia od jednego systemu do drugiego zawsze pojawia się podczas dokonywania normalnych poprawek i aktualizacji, ale trzeba zdawać sobie sprawę,że dla dokonania konwersji potrzebne są dodatkowe środki, zarówno w postaci pracowników jak i czasu komputera. Ponieważ przedsiębiorstwom brak rzeczywistego doświadczenia w zakresie konwersji, mają skłonność nie doceniać kosztu, a przekonanie, że zmierzają ku lepszej sytuacji, często przesłania im rzeczywisty koszt tego przedsięwzięcia /ilustr. G2/.

Mówiąc bardziej szczegółowo, konwersja oznacza potrzebę wytestowania wielkiej liczby programów, tabulogramów itd. i można przewidywać, że dla każdego konwertowanego programu potrzebna będzie duża liczba testowych przebiegów na komputerze.Zajdą też zmiany w programach na tej zasadzie, że skoro już dokonuje się przemiany, chciałoby się dodać coś niecoś. A przy tym, gdy dokonuje się konwersji czynnego programu, dział użytkowniczy najczęściej nie chce nic o tym wiedzieć i nie potrafi zrozumieć, dlaczego miałby w tym uczestniczyć.Problemem jest potrzeba dodatkowego personelu dla takiego przedsięwzięcia. Teoretycznie będzie on potrzebny tylko jednorazowo, a równocześnie byłoby pożądane,by miał pewne doświadczenie w zakresie konwersji. Podobnie ma się rzecz z obciążeniem komputera, bo mało jest prawdopodobne, by istniało dość wolnego czasu maszyny do wykonania zwiększonej liczby testów. W przypadku stosunkowo skom-

plikowanych systemów użytkowych zawsze trudno jest zdecydować, ile testów przeprowadzić.Na przykład w bankowym systemie użytkowym można by w charakterze próbki testowej dokonać powtórnego przebiegu jednomiesięcznej pracy. Na pierwszy rzut oka wygląda to na zupełnie wystarczającą próbkę. Ale taka próbka nie uwzględni, być może, 5% standardowych podprogramów, a mianowicie tych dodatkowych podprogramów, które wprowadzane są tylko na końcu każdego półrocza. Firma Brandon uznała za celowe włączyć do testów "nieprzewidziane przeszkody", takie jak np. odejście pracowników, choroby, opóźnienia w dostawie sprzętu itd. Ważne jest też określenie momentu zakończenia, to znaczy zakończenia każdego etapu konwersji, a w szczególności określenie punktu, w którym nieodwracalnie przechodzi się od starych warunków do nowych.Nieuniknione są błędy we wszystkich programach i potrzebne jest w dużej mierze równoległe wykonywanie już po przeniesieniu. Jak już powiedziałem, firma Brandon w dużym stopniu zautomatyzowała cały proces transformacji. Posiada narzędzia programowe do kontroli prac nad projektem jak również do pomocy w konwersji programów. Na dalszych przeźroczach podano pewne szczegóły prac nad projektem.Pierwszym etapem jest "planowanie" i obejmuje ono:

. inwentaryzację programów,
. analizę różnicy źródeł i celów,
. określenie priorytetów,
. wyszczególnienie kolejności i podziału na partie,
. analizę porównawczą metod,
. dobór zestawu programów transformacyjnych,
. określenie potrzebnych środków,
. ustalenie procedur dla poszczególnych zadań,
. sporządzenie harmonogramu projektu.

Drugim etapem jest "przygotowanie", które obejmuje:

. uzyskanie narzędzi programowych,
. zgrupowanie programów w partie,
. ustalenie testowych podzbiorów zbiorów wejściowych,
. wygenerowanie testowych zbiorów wyjściowych,
. zorganizowanie materiałów.

Następnym etapem jest "ustawienie":

- ustanowienie kontroli nad projektem,
- skatalogowanie wszystkich materiałów,
- weryfikacja materiałów,
- zakończenie opisów programów,
- zdublowanie programów i danych testowych,
- przygotowanie sterowania zadaniami dla każdego kroku.

Następnym etapem jest "konwersja":

- przetłumaczenie kodu,
- rozwiązanie problemów zniekształceń funkcjonalnych,
- zmodyfikowanie kodu w miarę potrzeby,
- oczyszczenie kompilatora,
- ręczne sprawdzenie i aktualizacja,
- skonwertowanie danych testowych.

Dalszym etapem jest "test jednostkowy":

- sprawdzenie materiału testowego i procedury sterowania zadaniami,
- wykonanie przebiegów testowych,
- powtórzenie przebiegu dla 100-procentowego porównania zbiorów,
- identyfikacja kodu odniesienia i operowanie nim,
- powtórzenie przebiegu dla dostatecznie reprezentatywnej części wykonania,
- ręczne sprawdzenie wszystkich niewykonanych kodów,
- powtórny test w razie potrzeby,
- rozładowanie transformowanych programów na taśmę,
- ponowne załadowanie i testowanie z taśmy.

Następnym etapem jest "testowanie systemów":

- zbudowanie sterowania zadaniami,
- utworzenie zbiorów testowych w miarę potrzeby,
- weryfikacja testów jednostkowych w miarę potrzeby,
- wykonanie strumienia zadań z danymi testowymi,
- powtórzenie przebiegu dla 100-procentowego porównania zbiorów.

Dalszym etapem jest "test równoległej produkcji":

- wdrożenie procedur produkcyjnych,
- modyfikacja sterowania zadaniami,
- określenie i zastosowanie zmian w aktualizacji,

. konwersja zbiorów produkcyjnych,
. wykonanie przebiegu równoległego,
. ponowny przebieg dla 100-procentowego porównania zbiorów,
. powrót do testu jednostkowego w miarę potrzeby,
. powtórzenie przebiegu dwóch cykli.

Następnym etapem jest "zatwierdzenie":
. weryfikacja porównania zbiorów,
. weryfikacja procentu wykonania,
. weryfikacja sprawozdań wyjściowych,
. weryfikacja sterowania zadaniami,
. kontrola czasu wykonania,
. aprobata ze strony wydziału operacyjnego APD,
. aprobata ze strony użytkowników.

Ostatnim etapem jest "wykończenie":
. ustalenie procedur rezerwowych,
. koordynacja operacji APD i działań użytkowników,
. aktualizacja zbiorów testowych.

Oczywiście,gdy konwertowane programy są modułowe i ustrukturowane, konwersja jest łatwiejsza i tam gdzie system użytkowy obejmuje kilka oddzielnych stadiów i gdzie interface'y danych mogą być zachowane, mamy też jako alternatywę możliwość wprowadzenia udoskonaleń do istniejącego systemu. Podoba mi się termin "okienka udoskonaleniowe".

Mam nadzieję, że udało mi się przekonać Was, że konwersja jest sprawą niezmiernie złożoną. Mimo to można ją z powodzeniem przeprowadzić za pomocą wypróbowanej metodyki i odpowiednich narzędzi programowych. Firma Brandon zawiera kontrakty na warunkach stałej ceny, ustalonego terminarza i gwarantowanej jakości programów.

Co zatem ustaliliśmy z punktu widzenia praktycznego użytkownika? Po pierwsze, że producenci dostarczają pomocy do konwersji i jeśli ktoś ma do czynienia ze stosunkowo łatwą konwersją, na przykład z COBOLu do COBOLu lub z COBOLu do PL/1, albo z czymś podobnym,wówczas narzędzia dostarczane przez producenta będą prawdopodobnie wystarczające. Gdy mamy jednak do czynienia

z konwersją od jednej maszyny w jednym języku do innej maszyny w innym języku, stoimy wobec gigantycznego zadania; ale jest przynajmniej jedna firma posiadająca odpowiednie doświadczenie i umiejętności praktyczne i zdolna do podjęcia się takiej pracy.

Trzecia rzecz, jaką ustaliliśmy to, że emulacja jest w gruncie rzeczy bardzo łatwa i w ramach ograniczeń wynikających z wielkości maszyny, czasu przebiegu i legalnej dostępności oprogramowania, jest to najprostszy sposób przejścia od jednego środowiska do drugiego.

Przyjrzyjmy się teraz zaleceniom obu grup studyjnych i popatrzmy w jakiej mierze odpowiadają one nam, użytkownikom.

Oba opracowania - zarówno EWG jak i CNRS/SRC zawierają szereg wspólnych zaleceń. Oba podkreślają potrzebę standardowych języków i przenośnych kompilatorów. Tak samo oba mówią o narzędziach programowania, ale bardzo różnie rozłożony jest akcent w obu sprawozdaniach. Podczas gdy studium CNRS/SRC ma podejście bardziej naukowe, studium EWG było nastawione na sprokurowanie pracy europejskim przedsiębiorstwom programistycznym. Różnica ta przejawia się w szczegółowych zaleceniach.

Mam tu przeźrocze, które jest niezbyt czytelne - projektowanie i opracowywanie przenośnych kompilatorów.Liczby po prawej stronie są w tych "europejskich jednostkach obliczeniowych", które z grubsza odpowiadają dolarom amerykańskim. Zaleca się więc wydatkowanie w ciągu najbliższych kilku lat 2 783 000 jednostek obliczeniowych - a zatem masę pieniędzy - na zaprojektowanie i opracowanie języka pisania programów.Na zaprojektowanie i opracowanie narzędzi konwersji ma się przeznaczyć 2 160 000 jednostek obliczeniowych; to również masa pieniędzy. Łącznie ma się udzielić przedsiębiorstwom programistycznym dotacji 9 milionów jednostek obliczeniowych na prace nad ułatwieniem przenośności.

Na przykład sprawozdanie ze studium EWG zaleca prace nad COBOLem uniwersalnym, FORTRANem uniwersalnym, superjęzykami, funkcjami systemów operacyjnych, uniwersalnymi programami nadzorczymi, podsystemami pierwszego szczebla,przetwarzaniem tran-

sakcji, bazami danych oraz pewne ograniczone prace nad oprogramowaniem systemowym i narzędziami programowania. Natomiast studium CNRS/SRC zaleca systemy oprogramowania do testowania kompilatorów, sugeruje też zbadanie cech języków w celu ustalenia tych cech, które utrudniają przenośność oraz zaleca szukanie sposobów poprawy struktury języków.Innym cennym punktem w sprawozdaniu tym jest propozycja, by przeprowadzić pewne prace w celu ułatwienia użytkownikom ustanowienia i egzekwowania swoich własnych norm. Umożliwiłoby to użytkownikowi usuwanie z języka pewnych funkcji, gdyż nie chce, by jego programiści posługiwali się nimi. Sprawozdanie CNRS/SRC podkreśla również doniosłość prac nad odgórnie ustrukturowanym programowaniem i nad metodyką programowania. Jak już uprzednio wspomniałem, określa ono też prawne aspekty emulacji i wyraża przekonanie, że warte wsparcia są dalsze prace nad sprzętem, prowadzące do rozwoju "miękkich maszyn" i emulacji.

Moją reakcją na oba te studia było,że każde z nich na swój sposób ma sporo racji.Ale jakie to będzie miało skutki dla nas, użytkowników? Odpowiedź brzmi, że prawdopodobnie skutki ujawnią się nie prędzej niż za 10 lat. Może to ocena trochę za pesymistyczna.Dyrektor Generalny do Spraw Przemysłowych i Technicznych Komisji Wspólnoty Europejskiej C.Layton, mówiąc o Wspólnocie Europejskiej i celach współpracy w dziedzinie gospodarczych zastosowań informatyki, wymienił konkretnie problem przenośności, ale czego brak z punktu widzenia użytkownika, to bodźca, który by skłaniał każdego producenta sprzętu, każde biuro programistyczne,a nawet każdego użytkownika, do działania w tym kierunku. Załóżmy, że zostanie opracowany uniwersalny COBOL z przenośnym kompilatorem.Czy naprawdę doczekamy się jego wdrożenia na skalę europejską? Twierdzę,że zalecenia dotyczące zmian języka i norm są po prostu niewykonalne i dlatego musimy zwrócić się ku "miękkiej maszynie" i emulacji, jako rozwiązaniom wielu problemów przenośności.

PARKINSON: Dziękuję Ci bardzo, Alan. Otwieram dyskusję i proszę o pytania.

WOLTERS /Siemens/: Sądzę, że trzeba jasno powiedzieć, iż przenośność nie jest koncepcją jednoznaczną, ale że istnieją różne rodzaje przenośności. Problemy przenośności pojawiają się nie tylko wówczas, gdy zamierzamy zerwać z jednym producentem komputerów i związać się z innym; uważam, że mamy takie problemy nawet we własnym ośrodku, gdy przechodzimy od jednego systemu operacyjnego do drugiego, bardziej zaawansowanego systemu operacyjnego, albo gdy mamy zbiór programów napisanych w czasie, gdy operowaliśmy taśmami i dyskami, a teraz korzystamy z wielodostępu i stąd mamy kłopoty.

A dalej, istnieje problem przenośności, gdy trzeba napisać program w taki sposób, żeby można go było wykonywać na komputerach dostarczonych przez różnych producentów.

I wreszcie,jest problem przenośności uwarunkowany czasowo, który pojawia się wówczas, gdy program napisany dla konkretnego komputera konkretnej generacji ma przebiegać na komputerze następnej generacji, dostarczonym przez tego samego producenta i chcielibyśmy być w stanie symulować tę następną generację.

Słyszeliśmy wiele o trudnościach przenośności i zupełnie zgadzam się w tej sprawie z referentem, zwłaszcza gdy mamy istniejący program i nagle chcemy go użytkować w innym środowisku. Może p.Duncan zechciałby nam powiedzieć, co trzeba robić pisząc nowe programy, żeby zapewnić, że te nowe programy będą w przyszłości możliwie najbardziej przenośne, a także powiedzieć coś o wspomnianych przeze mnie czterech rodzajach przenośności.

DUNCAN: Ile mam czasu? Około dwóch godzin? To rzeczywiście bardzo trudna sprawa. Sądzę,że jedyną znaną mi praktyczną wskazówkę podałem już, gdy powiedziałem, że należy stosować odgórnie ustrukturowane programowanie i starannie dokumentować zadania; jest to maksimum tego, co można na tym etapie zrobić.

DUVERGER /Diebold, Francja/: Byłem zdziwiony słysząc, że wspomina Pan o emulatorach. Emulatory to przecież bardzo stara rzecz - emulatory 1401 i tak dalej. O ile wiem, są dwaj producenci,którzy opracowali i produkują nowoczesne emulatory: Decca i Honeywell Bull. Wiem też, że Siemens miał od roku 1965 w po-

staci swojej serii 4004 sprzęt całkowicie wymienny z IBM-em, a jego seria 77 jest dzięki nowym emulatorom również wymienna z IBM-em. Sądzę,że przenośność zależy też w dużym stopniu od sposobu, w jaki wybieramy naszego pierwszego producenta.

DUNCAN: Chciałbym tylko coś dodać do odpowiedzi na poprzednie pytanie. Pragnąłbym podkreślić uczynioną dziś przed południem uwagę, że można bardzo ułatwić sobie życie w przyszłości,przyjmując proste podejście.

Jeśli chodzi o ostatnie pytanie,choć może było to nie tyle pytanie co stwierdzenie faktu, zgadzam się z nim; emulatory rzeczywiście istnieją od długiego czasu, ale myślę, że dopiero w ostatnich latach ludzie nagle uświadomili sobie, że za oprogramowanie płaci się oddzielnie i że przedstawia ono wartość pieniężną. Choć moglibyśmy nieraz wykonywać swoje zadanie za pomocą emulatora, użytkowanie w tym celu cudzego oprogramowania stwarza problem prawny; ludzie często nie biorą pod uwagę tego problemu, a czasem też producentowi lepiej odpowiada, żebyśmy nie pamiętali o tym.

NIEZIDENTYFIKOWANY MÓWCA: Panie Duncan, we Wstępnej Informacji zamieścił Pan pewną ilustrację,której nie użył Pan w referacie; pokazywała ona, że sprzęt tanieje w stosunku do oprogramowania. Powiedział Pan wprawdzie coś o tym, ale można to objaśnić dwojako. Załóżmy, że sprzęt rzeczywiście tanieje i że korzyści z jego potanienia nie są już niweczone przez skomplikowane oprogramowanie. Przyjmijmy, że zaczynamy ten tani sprzęt użytkować, to znaczy użytkować bardziej specjalizowane maszyny. Wówczas bylibyśmy w stanie robić dokładnie to samo, co od dawna robią technicy-automatycy - wstawiają przyrząd i pozostawiają go bez zmiany dopóty dopóki zadowalająco spełnia swoje zadanie; nie potrzeba żadnych konwersji; przyrząd umiera naturalną śmiercią i zastępuje go się nowym. Bardzo być może - i dziwię się, że te komitety nie wystąpiły z taką radą - że za kilka lat to będzie właściwy sposób. Ugrzęźliśmy w tych systemach zintegrowanych i innych, które w imię ciągłości trzeba konwertować itp., ale to jest tylko nasza obecna sytuacja. Będziemy może w stanie wydos-

tać się z tej sytuacji w ciągu, powiedzmy, pięciu lat.

DUNCAN: Zgadzam się ze wszystkim, co powiedział przedmówca. Jest zupełnie do pomyślenia, że za kilka lat sprzęt będzie tak tani, że będziemy w stanie kupować oddzielny sprzęt dla jednego zadania - w bankowości, powiedzmy, dla systemu kontroli kart kredytowych, a potem nie zajmować się nim więcej. Gdy system kart kredytowych będzie wykonywany w jakimś konkretnym systemie operacyjnym, będziemy mogli wydzielić do tego pewien sprzęt i pozwolić na wykonywanie systemu przez szereg lat; w ogóle nie będzie wchodzić w rachubę przeniesienie go na nową maszynę. Zgadzam się więc zupełnie z przedmówcą.

WOLTERS: Przykro mi, że znowu zabieram głos, ale chciałbym zapytać, czy rzeczywiście uważa Pan, że oprogramowanie powinno być związane z wielkością komputera? Ja sądzę, że jeśli chodzi o oprogramowanie, nie ma dla niego żadnego znaczenia, czy komputer kosztuje 5 milionów marek i ma objętość 12 metrów sześciennych, czy też kosztuje 1 markę i jest wielkości główki od szpilki. Uważam, że w obu przypadkach mamy ten sam rodzaj problemu i prawdopodobnie takie samo oprogramowanie. Producenci na ogół wolą operować tylko jedną rodziną programów, zaś w przypadku mikroprocesorów trzeba nam będzie o wiele więcej oprogramowania niż obecnie i będzie ono o wiele bardziej zróżnicowane. Ale nie bardzo zrozumiałem sens Pańskiej wypowiedzi.

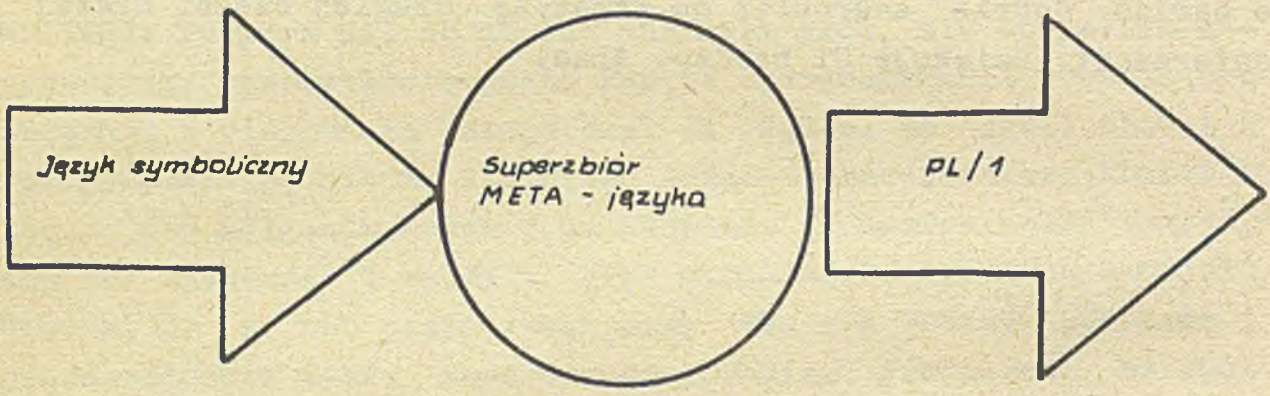
DUNCAN: Spróbuję wyrazić to nieco jaśniej. Chodziło mi o to, że obecnie znajdujemy się często w sytuacji, gdy musimy przenosić systemy z jednej maszyny do drugiej, dlatego że producenci zmieniają coś w maszynie, albo dlatego że musimy zmienić nasz system, bo stosowany przez nas dotąd system operacyjny uznany został za przestarzały i musimy wdrożyć nowy system operacyjny ze względu na inne zadania, np. ze względu na zadania wymagające zastosowania telekomunikacji. Musimy więc przejść do bardziej zaawansowanego systemu operacyjnego, a to oznacza, że musimy skonwertować wiele konkretnych systemów użytkowych.

Sądzę, że obu nam chodziło o to, że gdy koszt sprzętu spadnie jeszcze bardziej, znajdziemy się być może w sytuacji, że gdy

stworzymy system użytkowy z jego oprogramowaniem będziemy mogli wydzielić go na osobnym sprzęcie i pozostawić nienaruszonym przez długi czas, ponieważ potrzeba przejścia do bardziej zaawansowanego systemu operacyjnego w celu ustawienia innych zadań i innych systemów użytkowych na pożądanym poziomie, nie będzie wpływać na ten wydzielony system użytkowy. Będzie to więc możliwość izolowania zadań na oddzielnym sprzęcie i pozostawiania ich na dłuższy czas bez zmian. Mam nadzieję,że odpowiedziałem Panu na pytanie.

PARKINSON: Zanim zamkniemy sesję, pragnę podziękować Alanowi Duncanowi w imieniu Was wszystkich za ciekawy referat. Uważam, że bardzo dobrze zakreślił on tematykę drugiej części naszej konferencji. Dziękuję Ci bardzo, Alan.

Ilustracja G 1



Ilustracja G 2
Ogólne trudności konwersji

- Złożoność zarządzania pracami
- Potrzeba dodatkowych środków
- Zaniżone kosztorysy
- Ukryte koszty

SESJA H

KRYTERIA WYMIENNOŚCI SYSTEMÓW PRZEZNACZONYCH DLA WIELU UŻYTKOWNIKÓW

M.F.Wolters, Siemens A.G., RFN

Panie i Panowie, na wstępie muszę uczynić dwie uwagi. Po pierwsze pójdę w ślad przedstawiciela Barclaysa. Państwo pewnie nie znacie mego herbu: oto on. Jednorożec jest symbolem prawdomówności. Moją drugą uwagą wstępną jest, że znalazłem się w przykrej sytuacji. Pozwoliłem sobie przerwać panu Martinowi i skrytykować go, choć bardzo go lubię; wielka szkoda,że nie może on zrewanżować mi się. Pana Martina nie ma dziś i dlatego mam nadzieję, że Państwo zastąpicie go i nie zawahacie się wyrazić swej dezaprobaty. Gdybym zaczął Was nudzić, możecie gwizdać lub wykrzykiwać.

Po trzecie, tło tego o czym mam mówić, jest następujące. Siemens i ja zawarliśmy umowę: firma płaci mi pensję i ja nie uskarżam się na nią; ja produkuję nowe pomysły dla Siemensa - i firma nie skarży się. Nie znaczy to jednak, że wszyscy pracownicy Siemensa na całym świecie co do joty zgadzają się z koncepcjami, które omówię dziś z Wami, bo wiele z tych koncepcji jest zbyt nowatorskich.

Pan David zwrócił dziś przed południem uwagę na stale rosnącą złożoność techniki. Ja chciałbym skupić uwagę na oprogramowaniu. W związku z tym pragnąłbym Państwu opowiedzieć historyjkę, którą może słyszeliście już. Pewnego pięknego wiosennego ranka siedziało trzech starych panów w Ogrodzie Luksemburskim i przeszła koło nich bardzo piękna dziewczyna. Pierwszy powie-

dział: "ach, przed 60 laty spacerowaliśmy sobie pod rękę z takimi dziewczynami!" Drugi powiedział: "o tak, nie tylko spacerowaliśmy pod rękę, ale nawet całowaliśmy się!" A na to trzeci: "o tak, ale po całowaniu się, bywało jeszcze coś!". I właśnie to "bywało jeszcze coś innego" przypomniało mi się, gdy słuchałem dotychczasowych referatów.

Przysłuchiwałem się referentom i wydawało mi się, że czegoś w nich brak. Opisano nam te wszystkie piękne systemy; to co usłyszeliśmy wczoraj było bardzo futurologiczne i oczywiście opisał to nam ktoś z kraju, gdzie praktycznie wszystko jest możliwe. Po tej futurologicznej wycieczce wróciliśmy jak gdyby na ziemię. Ale we wszystkie te systemy trzeba tchnąć życie; trzeba sprawić, żeby działały. Aby to umożliwić, setki i tysiące ludzi, każdy w swoim kąciku, pracują nad programami - są to programiści produkujący oprogramowanie.

Nie pomylę się chyba twierdząc, że producenci oprogramowania - a dowiodły tego liczne konferencje począwszy od Konferencji w Monterey w 1973 r., na których mówiono o bardzo wysokich kosztach oprogramowania - i wiele zainteresowanych organizacji dochodzi do zgodnego przekonania, że koszt aktualizacji istniejącego oprogramowania zaczyna przekraczać koszt uzyskania nowego oprogramowania. Zakładam, że reprezentowane tu wielkie organizacje gospodarcze również zatrudniają pewną liczbę programistów i mam nadzieję, że nie mylę się sądząc, że chcielibyście wiedzieć, co - jeśli w ogóle - robi się, aby można było w sposób bardziej ekonomiczny tworzyć lepsze i bardziej przenośne oprogramowanie.

Powracam więc do problemu złożoności, rosnącej złożoności rozwoju technicznego przedsiębiorstw. Jest to nieuniknione. Nie tak dawno zaczęliśmy od wyłącznych użytkowników, którzy mieli całość komputera dla siebie; dlatego rozumieli swój system, który był dla nich zupełnie przejrzysty, a odpowiednie oprogramowanie można było uzyskać w dość prosty sposób. Wieloprogramowość i przetwarzanie wielomaszynowe skomplikowały sprawę w czasie, gdy pracowano w dyskowym lub taśmowym systemie operacyjnym. Już nawet na tym etapie niektórzy ludzie z rezygnacją rozkładali

ręce. Nie wiem, ale gdybym zadał Państwu pytanie "z iloma programami jesteście obeznani, które rzeczywiście wykonują większość przetwarzania wieloprogramowego i wielomaszynowego",prawdopodobnie powiedzielibyście, że z niezbyt wieloma.

Teraz znajdujemy się na początku ery wielodostępu i tutaj sprawy jeszcze bardziej komplikują się. Sądzę, że jeśli chodzi o programowanie, trzeba należycie uwzględnić konfigurację systemu wielodostępnego. Trzeba bardzo starannie rozważyć, gdzie inny użytkownik może zacząć wykonywanie swego konkretnego programu i jak najlepiej to wszystko powiązać.

Jest jeszcze druga rzecz. Człowiek,choć nazywa się go homo sapiens, nie jest w stanie i prawdopodobnie nigdy nie będzie w stanie śledzić abstrakcyjnie w swym umyśle więcej niż jedno zjawisko na raz, więcej niż jedną procedurę na raz. Ponadto,jak powiedział p.Duncan w swoim referacie,nie tylko musimy pogodzić się z trudnościami wynikającymi z wielodostępu, ale musimy też być świadomi, że na drugim planie symulujemy pewną liczbę starszych systemów operacyjnych. Wiemy, że wkrótce pojawią się dwuprocesory, co oczywiście doprowadzi do dalszych komplikacji. Ale wcale nie jestem pewien, czy wszyscy są świadomi,że dwuprocesor oznacza nie tylko redundancję mocy; że - innymi słowy - dotychczas drugi procesor normalnie zaczynał pracować dopiero gdy pierwszy ulegał awarii; ale gdy mamy dupleksowy układ dwuprocesorów, użytkujemy oba w sposób zrównoważony i dopełniają się one wzajemnie. System dupleksowy, czyli dwuprocesor szybko zostanie zastąpiony przez system wieloprocesorowy, a jeszcze dalej posuniemy się na drodze złożoności gdy dostępne nam będą omówione tu wczoraj międzynarodowe sieci teleprzetwarzania i teletransmisji. Na pewno nieprędko pojawią się standardowe dane, toteż przez dość długi okres będziemy musieli mieć do czynienia z danymi transmitowanymi przez linie razem ze wskazówkami,jak z nimi postępować. Będziemy musieli też zdać sobie sprawę, że trzeba będzie symulować na własnym komputerze programy napisane dla innych komputerów. A przy tym każdy z tych programów może być programem wieloprocesorowym i każdy z tych symulatorów będzie mógł symulować inne systemy operacyjne i być sto-

sowany przez inną grupę użytkowników.Śledzenie tego wszystkiego przekracza możliwości ludzkiego umysłu, bo - powtarzam - człowiek może śledzić w swej głowie równocześnie tylko jedno abstrakcyjne zdarzenie, a nie więcej. Oznacza to,że potrzeba nam coraz więcej programów systemowych i pomocy systemowych, programów, które będą wspierać programistę w toku programowania. Nie jest to łatwe. Dotychczasowe doświadczenie pomaga nam tylko do tego punktu i nie obejmuje nawet przetwarzania wielomaszynowego.

Celem tych programów systemowych jest próba przezwyciężenia tej złożoności i zmniejszenia jej stopnia, tak aby użytkownik nie stracił orientacji. Innymi słowy, trzeba stosować systemy, aby umożliwić użytkownikowi łatwiejsze użytkowanie innych systemów. Dziś w samochodzie nie musicie już pociągać wyłącznika zasysania przed uruchomieniem silnika,macie teraz w samochodzie małe urządzenie,które robi to za Was. Macie zatem w samochodzie większą złożoność, ale z Waszego punktu widzenia komplikacje uległy zmniejszeniu.

Pojawią się z konieczności i inne pomoce. W szczególności chciałbym podkreślić,że będziemy musieli zacząć myśleć nie tylko o przenośności programów,ale także o przenośności umiejętności praktycznych,czyli "know-how". Czy moglibyśmy spodziewać się lepszych i bardziej przenośnych programów, gdybyśmy nie umieli uczyć się,jak robić takie programy? Obecna metoda przekazywania umiejętności praktycznych - a odnosi się to także i do umiejętności wskazywania komputerowi co ma robić - jest na pewno bardzo niedoskonała.

Chciałbym wyodrębnić w moim referacie podrozdział o proponowanej metodyce rozwiązywania tych problemów.Na jakie problemy napotykamy w praktyce, gdy tworzymy oprogramowanie? Na tym przeźroczu niebieska krzywa jest w istocie rzeczy karykaturą tego co dzieje się obecnie z programem lub zbiorem programów. U dołu mamy oś czasu. W pewnym punkcie czasowym wszystko zaczyna się; to znaczy, od tego momentu pracujemy nad oprogramowaniem. Jak? No cóż,często działalność ta przypomina raczej działanie kolumny niezależnych artystów, brak bowiem dostatecznej

koordynacji. Cała rzecz kosztuje masę pieniędzy;w końcu program jest gotów, oprogramowanie jest gotowe i oddajemy je do praktycznego użytkowania.

Podczas fazy rzeczywistego użytkowania natychmiast popadamy w koszty utrzymania programów. Jeśli chodzi o utrzymanie programów, rozumiemy przez to trzy rzeczy. Po pierwsze, uruchomienie, bo jest to już pewien aspekt utrzymania. Po drugie,adaptację kodu; było to niejednokrotnie przyczyną nieporozumień.Trzecia rzecz objęta pojęciem utrzymania to dalszy rozwój produktu. I wreszcie, w pewnym punkcie czasowym kończy się użyteczność produktu i zostaje on zastąpiony innym - miejmy nadzieję - lepszym. Wciąż jeszcze mówimy o oprogramowaniu.

Problem jest następujący. Gdy chcemy uzyskać zamiast niebieskiej krzywej krzywą optymalną przedstawioną na przeźroczu linią czerwoną, musimy zwrócić szczególną uwagę na następujące sprawy. Przede wszystkim trzeba by opracować i udostępnić programistom lepszy zestaw metod planowania. Uważamy,że gdyby programiści mieli lepszy i wszechstronniejszy plan,byliby w stanie znacznie dokładniej przewidzieć przyszły rozwój. A zatem pomoc systemowa dla fazy planowania oraz kierownictwo i kontrola samych prac rozwojowych.

Drugim problemem jest potrzeba pomocy systemowych, które obniżą koszt programowania. Koszty te rzeczywiście wymagają zupełnie radykalnego zmniejszenia. Na przeźroczu sugeruję zmniejszenie ich o 50%; w praktyce można by spróbować osiągnąć nawet lepsze wyniki. Potrzebne nam są pomoce systemowe, które pozwolą nam uwolnić się od błędów w gotowym produkcie i to zanim program zostanie oddany do praktycznego użytku. Chciałbym powiedzieć Państwu, co my u Siemensa zrobiliśmy w tym kierunku.

W miarę możności i jeśli pozwoli czas,powiem coś o naszych konkretnych osiągnięciach. W rzeczywistości powinien nastąpić bardzo ostry skok. Pomoce systemowe mają przede wszystkim zapewnić - zanim jeszcze zaczniecie pisać rozkazy - że program Wasz oparty jest na solidnej podstawie teoretycznej. Po drugie pomoce systemowe są nam potrzebne po to, by przyśpieszyć same prace programistyczne i tym samym przybliżyć datę wdrożenia. Po

trzecie, tworzymy pomoce systemowe dla ułatwienia konserwacji i zapewnienia wysokiego stopnia przenośności. Konserwacji, tak jak ją tu rozumiem i jak to w dalszym ciągu referatu przedstawię, nadano nowe znaczenie. Ponadto potrzebne nam są pomoce systemowe, które pozwolą optymalizować istniejące programy. Wiemy że dziś, gdy program raz już zacznie działać, jesteśmy tak szczęśliwi, że zostawiamy go w spokoju i nie zastanawiamy się, czy nie mógłby przebiegać szybciej. I wreszcie pomoce systemowe powinny przedłużać użyteczność produktu; im dłuższa jest użyteczność produktu, tym prawdopodobniejsze jest, że osiągniemy jakiś zysk z inwestycji. Oto pięć funkcji pomocy systemowych.

Z drugiej strony mamy szereg problemów, do których przywykliśmy od dłuższego czasu. Ogółem biorąc, istniejącą procedurę zawsze nazywano programowaniem, a uczestniczyli w niej specjaliści-informatycy. Zatem specjalistą, który omawia te sprawy z przyszłym użytkownikiem, jest w zasadzie informatyk.Wie on zwykle dość dużo o sytuacji w jakiej system będzie użytkowany. Zna on też zwykle dobrze język,tak że może rozmawiać z potencjalnym użytkownikiem. Ale tutaj natrafiamy już na pierwszy problem. Czy specjalista od EPD rzeczywiście rozumie, czego chce użytkownik, czy też sprawia mu to trudność?

Istnieje pewna podstawowa trudność, której nie wolno nam przeoczyć. Jest to problem interpretacji. A mówi coś, a B rozumie coś; a ponieważ mówią tylko kolejno, nie w sposób analogiczny i nie równolegle i nawet gdy wyrażają wszystko na piśmie, czynią to kolejno, zupełnie nie są w stanie dostrzec całokształtu problemu,który kryje w sobie wiele aspektów dynamicznych. Gdy słuchacz B opuści choćby tylko jeden drobny szczegół,sądzi, że zrozumiał, co powiedział mu A; i nie ma sposobu,żeby stwierdzić, czy rzeczywiście zrozumiał on to co miał na myśli A, czy też nie. Powstaje tu znaczny obszar niejasności i sytuacja może być bardzo zła.

Pierwszym problemem jest więc problem dokładnego opisu i wyjaśnienia życzeń użytkownika. Drugim jest problem analizy.Czy jesteście zupełnie pewni,że wzięliście pod uwagę wszystkie możliwe okoliczności? Przenośność programu możliwa jest, jak po-

wiedział poprzedni referent, tylko wówczas, gdy program jest prawidłowo ustrukturowany. Jest to jeden z warunków, który musi być spełniony. Doszliśmy do przekonania, że jest 30 parametrów, które wpływają na przenośność programów i mam nadzieję, że w dalszym ciągu referatu będę w stanie wymienić większość z nich.

Od sposobu, w jaki użytkownik przedstawia swój problem, w dużej mierze zależy, czy analiza będzie prawidłowa czy nie. Gdy przedstawia się problem, trzeba to uczynić w sposób ustrukturowany, bo użytkownik na ogół wie, gdzie może się coś zmienić w jego wydziale i dobrze by było wspomnieć o tym programiście, aby mógł napisać specjalny moduł pokrywający obecną sytuację, który to moduł można będzie potem, gdy sytuacja zmieni się, zastąpić czymś innym.Ponadto użytkownik ma zwykle pewną koncepcję co do typu konfiguracji sprzętu, jakiej życzyłby sobie. Jeśli nie powie tego informatykowi, oczywiście wynikną kłopoty.

Czy analiza jest prawidłowa? Czy jest spójna? I czy jest jakiś sposób sprawdzenia, czy wszystko zostało właściwie zrobione? Taka możliwość istnieje i powiem o niej później.

Potem wchodzimy w fazę projektowania systemu. Następny problem pojawia się, gdy zaczynamy mówić o strategii programowania, przy czym trzeba ją odróżnić od taktyki programowania, zaś wielu programistów uważa, że taktyka jest synonimem improwizacji.

Problemem występującym przy opracowywaniu strategii programowania, któremu dotąd poświęcano mało uwagi, jest optymalizacja. Jak zoptymalizować dany program? Pod jakim względem należałoby go zoptymalizować? Można oczywiście pisać program w taki sposób, by jak najszybciej zakończyć programowanie;w takim przypadku optymalizujemy czas programowania. Albo można napisać program, który będzie bardzo oszczędny w sensie wykorzystania pamięci. Albo też można napisać program, który jest wykonywany w optymalnym czasie, albo wreszcie można programować tak, by zapewnić optimum przenośności. Optymalizacja jest pojęciem wieloznacznym, a ponieważ nie można zoptymalizować wszystkiego na raz, trzeba zdecydować, jakiego rodzaju lub jakich rodzajów optymalizacji chcemy. Jest to w istocie rzeczy sprawa analizy po-

równawczej i trzeba się zgodzić na pewien kompromis; ale jest to coś, o czym zbyt często zupełnie zapomina się.

Z kolei taktyka programowania. Zbyt mało jest pomocy do testowania i programowanie wciąż jeszcze odbywa się niemal w taki sam sposób, jak odbywało się na początku ery informatyki. Wszystko musi być ustalone przed wdrożeniem systemu. Należałoby zrobić to raz na zawsze, a potem już system powinien wszystko znać. Można by przechować wszystkie te informacje w listach i wywoływać od czasu do czasu. Programista powinien by tylko raz podać swoje nazwisko i swój numer identyfikacyjny, a system wyszukiwałby to ponownie.

Następnie istnieje problem szkolenia i przenoszenia "know-how", a także problem dokumentacji. Prágnąłbym zwrócić Waszą uwagę na fakt, że program jest znacznie bardziej przenośny, gdy dokumentacja jest staranna. Oznacza to - a aż nazbyt często zapomina się o tym - że w dokumentacji trzeba nie tylko powiedzieć, co program jest zdolny robić, ale także wskazać, jakie zasady rozwiązania zastosowano.

Weźmy konkretny przykład z dziedziny medycyny. Przed laty byliśmy bardzo dumni, że udało się nam stworzyć program, który analizował automatycznie krzywe EKG; szczególnie ważne było trzepotanie lewej komory ze względu na wpływ na drugi skurcz u chorych, którym podano digitalinę. Ale w istocie rzeczy była to tylko analiza krzywej; zastosowano analizę czynnikową i regresyjną i system porównywał krzywą normalną z krzywą, która odchyliła się od normy. Te metody analizy były znane już od długiego czasu, ale żadnemu lekarzowi nigdy przedtem nie wpadło do głowy, żeby zajrzeć do podręczników ergonometrii, aby zobaczyć, czy nie dałoby się wykorzystać którejś z tych technik.

Następną sprawą jest organizacja i wdrożenie ośrodka obliczeniowego. Tutaj znowu problemem jest przekazywanie "know-how" oraz dokumentacji; a dalej istnieje problem konserwacji pojmowanej tak jak to poprzednio opisałem.

Jak można to wszystko usprawnić? Uznaliśmy, że w tym celu wszystko, co dotąd robili informatycy, należałoby rozbić na trzy

części, wykonywane przez trzy różne grupy specjalistów, a równocześnie zapewnić optymalne wykorzystanie wszystkich istniejących urządzeń.

Użytkownik w ogóle nie powinien mieć kontaktu z informatykami, ale ze specjalistami, którzy są czynni w tej samej dziedzinie co użytkownik i wiedzą o niej nieraz więcej niż sam użytkownik. Powtarzam, że nie będą to informatycy. Ale powinni znać któryś z języków modelowania - LISP lub PASCAL: LISP ze względu na jego elastyczność i ponieważ można przy jego użyciu budować modele programów dziesięciokrotnie szybciej niż za pomocą innych języków; natomiast PASCAL jest wysoce użyteczny, ponieważ umożliwia opisywanie struktur danych. Innymi słowy, potrzebny nam jest problemowo nastawiony specjalista, który powinien znać język modelowania taki jak LISP i znać się nieco na pomocach systemowych;dzięki temu potrafiłby - może dość z grubsza - zbudować model,który uwzględniałby wszystko czego potrzebuje użytkownik i byłby zupełnie zdolny do funkcjonowania. Na tym etapie zupełnie pominęliśmy optymalizację, bo też na razie wcale nie chodziło nam o optymalizację. Jedyną rzeczą, jaką chcieliśmy tu wykazać, jest to, że całość można zautomatyzować; innymi słowy chodziło nam o wykonalność.Ta metoda organizacyjna ma na celu uniknięcie nieporozumień i ułatwienie wzajemnego zrozumienia; usprawnienie dotyczy więc komunikowania się.

Specjalista, o którym mowa,jest zdolny do porozumienia się z użytkownikiem, a model zostanie wykonany w jednej czwartej czasu, jaki byłby wymagany w innym przypadku. Model uwidoczni wszystko, co ma robić program. Użytkownik może temu modelowi przyjrzeć się; można ten model objaśnić użytkownikowi. Może on więc powiedzieć: "nie, nie zrozumiał mnie Pan. Ja nie tego chciałem". I jeśli musicie wskutek tego zmienić model, często możecie to przy pomocy LISPu zrobić w ciągu 15 minut i natychmiast przedstawić użytkownikowi nową wersję modelu.Jest jeszcze inny ważny aspekt, który poruszę za chwilę.

Następnie modelem zajmuje się drugi specjalista. Tym razem będzie to "specjalista od komputerów". Nie wiem, czy jest to prawidłowe określenie. Może należałoby nazwać go "informaty-

kiem". Ten specjalista jest w stanie przeanalizować model i sprawdzić, czy odpowiada on rzeczywiście potrzebom użytkownika i zaspokaja je. Może zastosować do tego modelu pewne techniki optymalizacyjne. Oczywiście będzie on rozumieć model; zresztą model będzie już wówczas na komputerze i będzie można przetwarzać go za pomocą technik matematycznych. Oczywiście tych technik matematycznych nie można do niego zastosować, zanim nie znajdzie się on na komputerze. Słyszy się wiele o modelach matematycznych, ale trzeba zawsze być bardzo ostrożnym. Nie wiem, czy kiedykolwiek widzieliście lub czy potraficie sobie wyobrazić model matematyczny roweru lub łodzi. Co byście z takim modelem robili? Ale jeśli macie model całkowicie obiektywny,możecie zastosować do niego techniki matematyczne.

Trzecią grupą specjalistów będą specjaliści od programowania. Chciałbym ich uważać za "specjalistów od metod". Tę grupę nazywamy w RFN "inżynierami software´owymi".Nazywamy ich tak, bo zawód programisty nie cieszy się jeszcze wysokim prestiżem i często się zdarza,że ludzie, którzy byli doskonałymi programistami nagle decydują się porzucić ten zawód,chcąc zostać projektantami systemów lub kierownikami; oczywiście bardzo często nie są tak dobrymi projektantami systemów ani zbyt dobrymi kierownikami, natomiast byli doskonałymi programistami. Zadaniem specjalistów od programowania będzie optymalizacja produktu.

"Specjaliści problemowi", którzy tworzyć będą model, będą doskonalić swoją specjalistyczną wiedzę, to znaczy będą kontynuować swoje kształcenie, ale tylko w swoim przedmiocie. Także specjaliści od komputerów będą doskonalić swoją wiedzę w swojej konkretnej dziedzinie, a programiści będą się uczyć nowych technik programowania i niczego więcej.

W przeszłości wadą informatyków było, że dużo wiedzieli o wielu rzeczach, ale nie dość o czymkolwiek szczegółowym.

Gdy już te trzy grupy specjalistów kolejno przepracowały model, przechodzimy do fazy projektowania systemów.Ponieważ zaś od początku prac do tego momentu upłynęło niewiele miesięcy, można ewentualnie omówić teraz z użytkownikiem konfigurację i zasugerować mu, że inny typ konfiguracji byłby może bardziej celowy.

Następnie dochodzimy do fazy, którą nazwałem "fazą pragmatycznie ustrukturowanego programowania". U podłoża wszystkiego leży system wyszukiwania informacji, który musi odznaczać się pewnymi cechami, jakich nie posiada dzisiejsza generacja systemów wyszukiwania informacji, a które omówię za kilka minut.

W tym przyszłym systemie wyszukiwania informacji znajdziemy m.in. opis produkowanych programów, które zostały skonwertowane i ich części składowych. Przy takim systemie wyszukiwania ujawnia się, że posiadamy niewiarygodne bogactwo, którego nie jesteśmy nawet świadomi. Jestem pewien, że bardzo trudno by Wam było powiedzieć, ile macie w swoich programach modułów znajdujących wielokrotne zastosowanie w różnych programach. Słyszeliśmy, że Genesys ma około 300 algorytmów. Narzędzia, o których wspomniałem, są również uniwersalne, a 300 algorytmów to niezbyt dużo. Spodziewamy się, że w stosunkowo krótkim czasie, przez przetwarzanie istniejących systemów wyszukiwania informacji, dla których również piszemy oprogramowanie, można będzie analizować istniejące oprogramowanie, aby rozpoznać wszystkie algorytmy lub moduły, które pojawiają się wielokrotnie. Wszystkie one powinny być przekształcone w "jednostki pragmatyczne".

Co to jest "jednostka pragmatyczna"? Koncepcja jednostki pragmatycznej jest czymś co otwiera przed Wami cały świat. Na przykład, gdy mówię "sortować" nie muszę podawać Wam długiego objaśnienia, bo wiecie dokładnie co znaczy "sortować" albo "łączyć" albo "odwrócone przetwarzanie tablic". Wszystkie tego rodzaju pojęcia nazywamy jednostkami pragmatycznymi. W niektórych bardzo dużych programach są pewne składniki będące takimi jednostkami pragmatycznymi i reprezentujące bardzo rozległe funkcje. Przy mikroprogramowaniu jednostki te pozwalają zwiększyć ilość rozkazów wykonywanych w jednostce czasu. Można je przechować w banku metod, w banku procedur itd. I można je z tych banków wyszukiwać.

Chcielibyśmy znaleźć sposoby i środki rozpoznawania podobnych programów, programów, które niezupełnie odpowiadają Waszemu konkretnemu problemowi, ale które po małej modyfikacji mogą stać się zupełnie użyteczne. Napisaliśmy bardzo podobny do emulatora

program, który uniezależnia nas od systemu operacyjnego, dla którego konkretny program został napisany.Na przykład programy, które zostały napisane dla konfiguracji taśmowo-dyskowej, można dzięki temu emulatorowi wykonywać w bardziej nowoczesnym systemie operacyjnym, przy czym współczynnik straty sprawności wynosi tylko 1,2, podczas gdy w dawniejszych emulatorach z ery 1401 współczynnik ten był znacznie wyższy. W ten sposób można symulować obce systemy operacyjne.

Jest to prawdziwa symulacja,a nie konwersja. Ale w okresie przejściowym może przyczynić się do większej przenośności.

Następne z kolei jest więc pragmatycznie ustrukturowane programowanie oznaczające, że będziemy posługiwać się systemem informacji o programach,który będzie wskazywać pewne istniejące moduły, nadające się do wielorakiego zastosowania. Opisujemy nasz problem najpierw za pomocą programu zwanego CONDOR;pozwala on komunikować się w języku naturalnym z dialogowymi systemami wyszukiwania. Jest to program lingwistyczny. Analizuje on język naturalny, na razie tylko niemiecki, ale za kilka lat będzie w stanie analizować także język angielski i rosyjski. Opisujemy więc nasze problemy w języku naturalnym za pomocą pewnej liczby słów kluczowych,a CONDOR analizuje ten opis i zwraca naszą uwagę na już opublikowane programy, które zajmowały się podobnymi problemami i wskazuje, które z zawartych tam funkcji moglibyśmy wykorzystać; jakie zawarte tam moduły - ewentualnie pochodzące z innych systemów operacyjnych - moglibyśmy zużytkować.Wskazuje też bardzo wyraźnie, co pozostaje nam jeszcze do zaprogramowania we własnym zakresie.

Możemy więc obejść się bez strategii i taktyki, ponieważ moduły te zostają wybrane pod kontrolą systemu w taki sposób, że zgodne są z planami strategicznymi obowiązującymi w naszym przedsiębiorstwie. Nic więcej nie jest zawarte w bazie modułów. Jest to bardzo ważne, bo normalnie nie zawsze jesteśmy w stanie nieustannie wyjaśniać, jaka jest ogólna strategia w konkretnej dziedzinie.

Pominęliśmy konserwację i szkolenie, ale za chwilę zobaczymy dlaczego. Mamy tu powtarzającą się procedurę i jeśli pro-

gramy zostały zadawalająco ustrukturowane, programista jest stale kontrolowany i gdy nie przestrzega wszystkich proponowanych tu reguł, zostaje mu zwrócona uwaga. Robi to inny program zwany COLUMBUS. W ten sposób wszyscy programiści przestrzegają wszystkich konwencji i nie zdarza się naruszanie norm.

Odkryliśmy, że w 90% przypadków normy są naruszane wskutek ich nieznajomości. Toteż musimy zapewnić, by sam system dbał o to, by normy nie były naruszane. Gdy ktoś świadomie zechce je naruszyć, będzie mógł to zrobić, ale będzie musiał podać, dlaczego zdecydował się na to. Zostanie to zapisane w odpowiednim protokóle i być może stanie się podstawą do wprowadzenia poprawki do istniejącej normy. Ma to więc skutek sprzężenia zwrotnego. Dalsze opracowywanie programu będzie polegało na wymianie pewnych modułów lub jednostek strukturalnych lub na dodaniu takich podobnych modułów.

Jestem pewien, że wszyscy zetknęliście się już kiedyś z sytuacją, gdy mamy ogromny program, rozpełzający się na wszystkie strony jak ameba. Gdy chcemy coś w nim zmienić, musimy zatrudnić bardzo doświadczonego programistę. Niewątpliwie popełni on przy tym pewne błędy, ponieważ całość programu jest mało przejrzysta. Zapewne zgodzi się raz poprawić program, ale kiedy zechcecie, żeby to zrobił znowu, prawdopodobnie odejdzie, bo woli wykonywać jakąś bardziej twórczą pracę.

Jak powiedziałem,nie wspomnieliśmy tu w ogóle o szkoleniu. Nazywamy je "orientacyjno-informacyjną składową systemu". Nasz sposób rozumowania jest następujący. Gdy ktoś popełni błąd obsługując program, urządzenie końcowe wydrukowuje "Błąd nr 136". Skoro już przechowujemy w pamięci dane diagnostyczne, dlaczego nie mielibyśmy przechować w niej równocześnie wskazówki, co należy zrobić, żeby błąd poprawić? Jest to tylko sprawa tekstu jaki przechowujemy w pamięci. Gdy się pomyśli ile podręczników trzeba napisać i jak obszerne muszą być te podręczniki, aby wytłumaczyć, jak posługiwać się dużym systemem oprogramowania, człowiek zawsze jest pełen obawy "czy to co mam, to ostatnia wersja podręcznika, czy też nie jest to ostatnia wersja?". Ilekroć chcecie zmienić program, musicie napisać nową broszurę.

Nawet gdy użytkownik posiada najnowszą wersję podręcznika, zachodzi poważna obawa, że nie otworzy go na właściwej stronie. A więc po co wskazywać tylko błąd? Dlaczego nie wskazać od razu, jak naprawić ten błąd, tak by system podpowiedział użytkownikowi, co ma być zrobione? W takich warunkach szkolenie stanie się sprawą znacznie łatwiejszą.

Przy konwencjonalnym sposobie pisania programu, gdy chodzi o ambitny system, upływa kilka lat zanim program zostaje po raz pierwszy zastosowany w praktyce. Użytkownik powie zapewne: "ale to przecież zupełnie nie to, czego chciałem. Myślałem, że tu wprowadzicie odgałęzienie w tym kierunku, a tutaj w tamtym kierunku. Nie miałem pojęcia, że to tak wypadnie." Albo też może powiedzieć: "no cóż, przykro mi ogromnie, ale cała sytuacja u mnie zmieniła się w ciągu tych dwóch lat.To czego potrzebowałem przed dwoma laty, nie jest mi już więcej potrzebne; potrzebuję czegoś innego". I co możecie wtedy zrobić?

Być może, przez ten czas zaszły zmiany w sprzęcie i oprogramowaniu i stwierdzacie, że zupełnie nie jesteście w stanie przejść od dawnej sytuacji do nowej. Natomiast przy naszej udoskonalonej metodzie już po kilku miesiącach będziecie mieć swój model i prawdopodobnie niektóre programy, dlatego też będziecie mogli natychmiast skontaktować się z użytkownikiem i zobaczyć, czy jest to rzeczywiście to, czego chciał. A gdy użytkownik wyrazi aprobatę,możecie natychmiast wejść w produkcyjną fazę faktycznego pisania programu, a przy wszystkich pomocach systemowych, o których wspomniałem, będziecie w stanie ponownie wykorzystać części innych programów,co oznacza, że będziecie znacznie prędzej posuwać się naprzód.

Są też inne pomoce, o których należałoby może wspomnieć tutaj; o niektórych z nich już mówiłem. Weźmy np. całą fazę przygotowawczą, przygotowującą Wasz ośrodek do fazy testowej. Znacie może książkę Bakera i Millsa na temat głównego programisty i sekretarza programu. W Europie nie ma funkcji sekretarza programu, ponieważ człowiek o potrzebnych do tego bardzo wysokich kwalifikacjach i bardzo dużym doświadczeniu nie zgodziłby się być sekretarzem programu. Jedyny sposób, w jaki moglibyście

sobie poradzić w tej sytuacji, to powiedzieć: "jeśli nie podoba się Panu ta propozycja, zwalniam Pana z pracy; jest dziesięciu czekających na Pana stanowisko", ale w Europie czegoś takiego nie można zrobić.

Funkcję tego sekretarza programu można w dużej mierze włączyć do systemu: zarządzanie zbiorami danych i zaplanowanie sieci może być znacznie ułatwione dzięki pomocom systemowym. Stworzyliśmy program,który spełnia te wszystkie funkcje. Nazywa się on STARK. Oferuje pomoc już na wstępie,już na samym początku prac programistycznych. Oznacza to, że programista będzie znacznie szybciej posuwał się naprzód. Oznacza to też, że dane gromadzone w toku pisania programu bardzo szybko wejdą do bazy danych i w ten sposób unikniecie dwukrotnego robienia tego samego.

Dublowanie pracy jest bardzo często wynikiem tego, że nie zebrano dość danych na etapie planowania. Gdyby zebrano te dane podczas etapu planowania, przechowano je, a potem wyszukiwano, uniknięto by wiele dublowania. Można by powiedzieć: "patrzcie, to co zamierzacie zrobić, albo coś bardzo podobnego,zostało już zrobione przy innej okazji".

Podczas fazy kontrolnej można stwierdzić, czy projekt logiczny, opisany już za pomocą schematu działania, rzeczywiście uwzględnia wszystkie problemy, czy rzeczywiście będzie działał bez martwych punktów i czy jest wystarczająco dynamiczny.

Profesor Petri, który pracuje w Bonn w GMD, stworzył tzw. grafy Petriego. Mogę tu opisać tylko podstawową zasadę, która jest bardzo prosta. W praktyce cała rzecz jest nieco bardziej skomplikowana.

Podstawowym założeniem jest,że pewne zdarzenie może nastąpić tylko, jeśli w Waszym systemie automatycznym zaistnieją pewne stany. Zdarzeniem może być na przykład konwersja, a gdy to zdarzenie nastąpiło, pojawia się nowy stan. Spróbuję pokazać Państwu na tej ilustracji, jak taki system może zaprowadzić nas do martwego punktu. Do tego potrzeba mi paru drobnych monet; jedna marka to za dużo, a karta kredytowa nie zda się na nic.

Oto graf odpowiadający pewnej sytuacji. Widzicie, że podobieństwo między nim a schematem działania jest zupełnie oczywiste. Koła z małym znaczkiem odpowiadają pewnemu stanowi początkowemu i teraz przykryję je monetami. Jakie zdarzenie może teraz nastąpić? Nie to, ponieważ nie położyłem monet na tamtych dwóch stanach. To zdarzenie również nie może nastąpić,bo tamten stan nie jest przykryty. To zaś nie może zajść, bo ten i tamten stan nie są przykryte, natomiast tamte zdarzenie może nastąpić, bo te dwa stany są przykryte. Pojawiają się te nowe stany. Zdarzenie to daje więc w wyniku stan równowagi.

Co jeszcze może się zdarzyć? Nic;doszliśmy już do martwego punktu. To nie może się zdarzyć, bo tamte stany nie są zakryte. To zaś nie może się zdarzyć, bo nie jest zakryte. Nic więc nie może się ruszyć.Odnosi się to zarówno do sprzętu jak i oprogramowania. W związku z tym powiecie zapewne: "nie rozumiemy,czegoś tu brak".

A teraz druga sytuacja. System nie jest niezawodny.I znowu czegoś gdzieś brak, a czego brak - to decyzji. Mamy tę samą sieć, ale sytuacja początkowa jest różna. Oto ten stan początkowy, czyli sytuacja początkowa. Co może teraz zdarzyć się? Tylko to jedno zdarzenie może nastąpić. Czy rzeczywiście tak? Sądzę, że tak. Tylko to jedno zdarzenie może nastąpić. A więc - faktyczna równowaga. A co teraz?

Następne co może się zdarzyć,to tylko to. Najgorszą jednak rzeczą jest,że bardzo wiele rzeczy może się zdarzyć w tym samym czasie; to może się zdarzyć,dlatego że te dwa stany są zakryte; a to może się zdarzyć, bo tamte dwa są zakryte. I dlatego brak decyzji: "co powinno teraz zdarzyć się?" Program stoi wobec dwóch możliwości i nie bardzo pewne jest, jaka ma być jego następna czynność.

Trzecia możliwość, jeśli kontynuujemy grę do końca, polega na tym, że może się zdarzyć albo to albo tamto.Nastąpić może to zdarzenie; może też to tutaj; i to; i to;i w ten sposób wróciliśmy do naszej początkowej sytuacji.

Taka jest podstawowa zasada tego systemu. Oczywiście, w rzeczywistości sprawy są nieco bardziej skomplikowane; nie bę-

dziemy mieli takiego prostego schematu i trzeba będzie całość wykonywać na komputerze.

Metoda ta dowiodła swojej wartości w odniesieniu do sprzętu i oprogramowania. Pierwotnie opracowano ją dla sprzętu, potem spróbowaliśmy zastosować ją do oprogramowania i zrobiliśmy to z powodzeniem szczególnie w dziedzinie telekomunikacji, gdzie praktycznie nie ma innej metody, pozwalającej na kontrolę zjawisk dynamicznych. Wszelkie inne reprezentacje tego co dzieje się w sieciach telefonicznych, wszelkie opisy techniczne, są statyczne i nie ma w ogóle sposobu, w jaki można przedstawiać zjawiska dynamiczne.

To byłaby więc metoda, która przy użyciu komputera umożliwiłaby nam bardzo szybkie i łatwe ustalenie, czy rozwiązanie o jakim myślimy, jest rzeczywiście tym, którego chcemy, czy jest ono kompletne i dynamiczne.

Wróćmy teraz do zagadnienia przenośności i do omówionego poprzednio schematu, w którym mieliśmy model. Od teraz cała sytuacja nie jest już związana z określonym zastosowaniem. Oprogramowanie jest zawsze to samo. Jeden program jest bardzo podobny do drugiego. Nie ma znaczenia,czy program jest pisany dla stacji sztucznego unasieniania czy dla sterowania reaktorem atomowym. Oprogramowanie jest oprogramowaniem.

Model napisany w języku modelowania jest absolutnie przenośny. Można go zastosować do innych rozwiązań i do innych komputerów. Wyobraźcie sobie co to znaczy! Dokonaliśmy obiektywizacji modelu, dowiedliśmy jej wykonalności,a samo programowanie po utworzeniu tego modelu jest znacznie ułatwione. Niepotrzebna jest cała faza analizy, udało się nam uwolnić od niej.

Przejdźmy na chwilę do innego tematu, ale chodzi o sprawę dość ważną dotyczącą przenośności. Wspomniałem poprzednio, że zależy nam na uniknięciu dublowania pracy. Wspomniałem też, że należy bardzo dokładnie przeanalizować problem. Im dokładniej zdefiniowaliście Wasz problem,tym lepiej będziecie w stanie pokierować pracami programistycznymi i tym prawdopodobniej uda się Wam uzyskać przenośność. Chciałbym omówić macierz problemo-

wą, nazywaną czasem "krajobrazem problemu". Znana jest ona od dłuższego czasu pod tą niezbyt ścisłą nazwą.Nowe natomiast jest jej szczegółowe zastosowanie. Uważam,że w ogóle nie dość mówimy o szczegółach. Na spotkaniach takich jak obecne zwykle referat urywa się właśnie, gdy przychodzi kolej na omówienie szczegółów technicznych. Prawdopodobnie dlatego m.in., że bardzo trudno jest jasno wytłumaczyć takie rzeczy. Ale spróbujmy.

Miałem zaszczyt być Kierownikiem Produkcji przy wdrażaniu programów na komputerze zastosowanym tutaj w Monachium w czasie Olimpiady. Wiecie, że na Olimpiadzie w Meksyku nie było komputera, była jedynie dalekopisowa sieć dystrybucyjna,a przy manipulowaniu tymi wszystkimi danymi zatrudnionych było 28 000 ludzi. Musieliśmy więc w Monachium zinterpretować dla komputera reguły 200 różnych dyscyplin sportowych, a także reguły stosowane przez zespoły sędziowskie przy wydawaniu orzeczeń; i jak widać, udało się to nam. Ale Igrzyska Olimpijskie nie przedstawiają w gruncie rzeczy poważnego problemu intelektualnego. Całość problematyki Igrzysk można sprowadzić do czterech problemów.

Pierwszy problem: przebiegnij jak najszybciej z miejsca A do miejsca B i zmierz upływ czasu. Drugi problem: rzuć przedmiot - lub siebie samego - możliwie najdalej od punktu wyjścia i zmierz odległość. Trzeci problem: wykonaj ruch zgodnie z zadaną procedurą i zmierz wszelkie odchylenia.Po czwarte - wyniki powinny być rozprowadzone po wydrukowaniu na 6 000 list. I to wszystko; cała reszta to tylko sprawa organizacji.

Zawiesiliśmy na ścianie 200 schematów działania odpowiadających 200 różnym typom gier, aby zobaczyć, jakie zachodzą między nimi podobieństwa i czy istnieją ewentualne wspólne moduły. Zatrudniliśmy wówczas przy tej pracy 150 programistów i w rezultacie otrzymaliśmy 600 000 wierszy kodu; ale w zasadzie była to sprawa organizacji.

Stwierdziliśmy, że gdy mamy dany problem, tzn. gdy postawiono przed nami określone zadanie, można je rozbić na części składowe i otrzymać macierz. Można też stworzyć strukturę hierarchiczną obejmującą cztery szczeble i tak normalnie postępu-

jemy. Potem tworzymy zespoły, które opracowują te różne zadania częściowe albo bierzemy zespoły, które już istnieją w przedsiębiorstwie, zwołujemy je i zapytujemy: "co możecie wnieść do części problemu oznaczonej numerem 112? Jaki jest Wasz wkład?" Wtedy oni wypowiadają się i piszą to na kartach.

Następnie zbieramy te karty. Na pierwszy rzut oka może się wydać, że zachodzi tu wiele dublowania. Istnieją przy tym trzy możliwości. Pierwsza możliwość to, że jest to rzeczywisty przypadek dublowania i wówczas będziemy musieli zdecydować, który zespół ma się tym zająć, a który nie. Druga możliwość to, że pewne rzeczy potrzebne są bardzo szybko, natomiast inne potrzebne będą znacznie później. W takim przypadku możemy oczywiście pozwolić, by odbywało się to równolegle. Trzecia możliwość: zrozumieli oni temat w różny sposób; nie mają na myśli tego samego. W takim przypadku musimy uściślić postawiony temat i opisać wszystkie czynności w sposób daleko bardziej jednoznaczny.

A dalej jest coś, co jest ważne i czego dotąd nie dostrzegano może dość wyraźnie, a mianowicie to, że gdy wykrywamy moduł nadający się do wielokrotnego zastosowania, zawsze jest ktoś, komu będzie on potrzebny raz na sześć miesięcy i ktoś inny, kto może go potrzebować co cztery sekundy. Dlatego optymalizacja w odniesieniu do tego modułu będzie różna, w zależności od tego, czy macie go używać co cztery sekundy czy raz na sześć miesięcy. Informacja ta powinna być włączona do opisu programu. Będzie ona Wam potrzebna, gdy zechcecie skonwertować ten program do innego systemu; w przeciwnym razie nastąpi w wyniku konwersji utrata sprawności. Tego rodzaju rzeczy nie można tak łatwo symulować. Dlatego informację tę należy nie tylko umieścić w opisie programu, ale także wprowadzić ją do Waszej bazy danych funkcjonalnych.

Moje dotychczasowe doświadczenie, przynajmniej w moim przedsiębiorstwie, wykazało, że powyższa metoda pozwala rzeczywiście uniknąć dublowania prac. Potrzeba na to trochę czasu, ale prawdopodobnie wystarczą cztery pełne tygodnie; a gdy już raz zrobicie to, zobaczycie, że bardzo szybko zacznie się opłacać.

Zapomniałem przedtem powiedzieć, jak planuje się taką pracę - od początku aż do zakończenia. Nie będę się nad tym rozwodził, bo opublikowaliśmy na ten temat książkę. Napisałem książkę, ale sam jej jeszcze nie czytałem; nosi tytuł "Planowanie i wdrażanie projektów EPD". Jest to 700-stronicowa publikacja, która ukazała się przed czterema tygodniami. Kosztuje tylko 390 marek i jest już całkowicie wyczerpana; mam nadzieję, że dodrukują jeszcze dodatkowy nakład.

Jest jeszcze jedna sprawa, którą chciałbym tutaj poruszyć. Chodzi o przenośność wiedzy, czyli "know-how". Wspomniałem już o problemie podręczników, do których musicie sięgać za każdym razem, gdy ukazują się nowe wersje programów. Posunąłbym się nawet do twierdzenia, że program źle udokumentowany w ogóle nie jest przenośny.

Istnieją pewne środki lingwistyczne, które chętnie omówiłbym tutaj, gdybym miał czas. Są już techniki lingwistyczne,które pozwalają stosować komputer do automatycznego wytwarzania streszczeń. Można przekształcić język w taki sposób, by uczynić go bardziej ogólnym. Nazywamy to "językiem strategicznym" i pozwala on systemowi rozwiązać jeden poważny problem,który istnieje w niemieckim, a nie istnieje w angielskim. W angielskim wszystko jest łatwiejsze; w angielskim macie o 40% więcej słów niż w niemieckim. W niemieckim istnieje problem synonimów, homonimów itd. i jest to duży kłopot. Sądzę, że także nasi francuscy przyjaciele mają pewne specjalne problemy lingwistyczne. Podobno wydano niedawno we Francji nową ustawę, która zabrania użytkowania słów niefrancuskich. Mimo to jakoś nie umarłem z głodu któregoś dnia, kiedy zamówiłem we Francji po angielsku stek, bo jakoś zrozumieli mnie. Nie można dekretować, że od danej chwili nie będzie się używać danego terminu; to bezskuteczne. Języki sztuczne nigdy do nikąd nie zaprowadziły.

Nie wiem,czy te liczne produkowane obecnie języki, które dają pewne ograniczone korzyści, ale także pociągają za sobą wiele ujemnych następstw, okażą się w ostatecznym rachunku zupełnym fiaskiem. Wynajdują je informatycy w swoich wieżach z kości słoniowej, a potem wmuszają je ludziom. Uważamy, że jako

środek komunikacji, a szczególnie komunikacji z bazami danych, należałoby stosować normalne, potoczne języki - niemiecki, angielski, francuski, rosyjski - i mam nadzieję, że stanie się to możliwe w niezbyt odległej przyszłości. Nawiasem mówiąc, sugerowałbym, by na jednej z najbliższych konferencji Diebolda dać nam okazję do pokazania, co zrobiliśmy dotąd w dziedzinie rozpoznawania i analizy języka.Otworzyło się już wiele nowych możliwości w tej dziedzinie.

Zamierzamy sporządzić statystykę,by ustalić, które terminy najczęściej powtarzają się przy opisywaniu systemu. Te najczęściej powtarzające się terminy stałyby się następnie deskryptorami i byłyby zapisywane w systemie. Upewniliśmy się,że w razie wybrania takich deskryptorów użytkownicy chętnie zaczną posługiwać się nimi.

Powróćmy do przenośności umiejętności praktycznych, czyli tzw. "know-how". Zrobiliśmy coś, co teraz przyjęło się nie tylko w naszej firmie, ale też w innych krajach i na wielu uczelniach. Dokonujemy mianowicie wyraźnego rozróżnienia między tym co nazywamy "katalogowym know-how" a "bieżącym know-how". Gdy studiowałem na uniwersytecie, zawsze irytowało mnie, gdy dawano mi duży stos książek i mówiono: "masz to wszystko przeczytać", ale nikt nie mówił mi, ile z tego rzeczywiście mam przechować w umyśle, a co do reszty wystarczy żebym wiedział, że znajduje się to w takiej a takiej książce i że będę mógł później w każdej chwili zajrzeć do niej.

Usłyszeliśmy tu, że postęp techniczny jest bardzo szybki i że człowiek stale musi się uczyć czegoś nowego. Nawet kierownik średniego szczebla musi korzystać z pomocy pewnej liczby specjalistów,bo nie można być specjalistą w swojej własnej konkretnej dziedzinie, a równocześnie matematykiem, psychologiem, socjologiem i Bóg wie czym jeszcze. Musicie więc mieć możność zasięgnięcia porady. Oczywiście musicie sami umieć dość,by potrafić ocenić wartość takiej porady.

Napisaliśmy inną książkę zatytułowaną "Klucz do komputera". W tej książce byliśmy w stanie dokazać, że całej obecnej wiedzy dotyczącej komputerów można się nauczyć w ciągu 30 godzin i że

w gruncie rzeczy wystarczy nauczyć się jednej trzeciej tej całej wiedzy. Na początku trzeba dokonać dużego wysiłku,żeby zrozumieć podstawowe zasady danego przedmiotu. Musicie więc postarać się tak określić podstawową koncepcję programu, żeby można było łatwo nauczyć się, jaki jest cel programu, do czego służy on oraz co takiego pozwala użytkownikowi czynić,czego nie mógłby uczynić bez tego programu, a także w jaki sposób program ten współdziała z urządzeniami peryferyjnymi.

Gdy to zrobicie, sam system zapewni, że po naciśnięciu odpowiedniego klawisza dla wprowadzenia nazwy programu,na ekranie ukaże się instrukcja, w jaki sposób użytkować ten program. Potem, gdy użyjecie pióra świetlnego i dotkniecie nim np. numeru 5,na ekranie ukaże się znowu coś innego. Nie będzie Wam już więcej potrzebny żaden podręcznik; sam system zastąpi go; zniknie wysiłek dydaktyczny i nie będziecie już musieli mozolnie nabywać wiedzy. Jest to oczywiście duże ułatwienie.

Powracając jeszcze do przenośności "know-how", chciałbym podkreślić, że bez możliwości analizy języka znaleźliśmy się w następującej sytuacji. Normalne środki językowe pozwalają opisać i zdokumentować program, ale nie przyczynia się to do przenośności, ponieważ czytelnik nie zawsze będzie w stanie dokładnie zrozumieć co miał na myśli autor opisu i nie będzie w stanie objąć całości za pomocą opisu; w szczególności,gdy nie jest obeznany ze specjalnymi problemami, dla których program został napisany.

Weźmy przykład z dziedziny maszyn.Do niedawna środki opisu kształtów i sposobów obróbki materiału były zupełnie takie jak w średniowieczu i dopiero znacznie później pojawił się bardziej techniczny język do opisu bardziej rozwiniętej techniki. Ten język był jednak wciąż jeszcze oparty na języku naturalnym. Ale lepszą przenośność osiągnęliśmy dopiero wtedy, gdy doszliśmy do jednostek pragmatycznych, przy których zamiast posługiwać się językiem, nawet specjalizowanym, zaczynamy stosować zrozumiałe na całym świecie symbole. Każdy wie, że np. taki symbol oznacza wał. To tutaj oznacza zgrubną obróbkę powierzchni, a taki znowu symbol - dokładną obróbkę powierzchni. Każdy kto spojrzy na to, może dzięki symbolom zrozumieć o co chodzi.

Weźmy inny przykład: matematykę. Ktoś kto nie zna matematyki, może oczywiście powiedzieć,że są to jakieś zbiory i cyfry itd., bo żeby stosować język matematyczny, trzeba znać matematykę i dopiero wtedy można zacząć stosować symbole. Jest to sprawa analizy lingwistycznej. Trzeba jej dokonać, a dopiero potem można pójść dalej, tzn. tworzyć jednostki pragmatyczne, po czym osiąga się rzeczywistą przenośność.

Pozostało mi jeszcze pięć minut, wobec czego powiem może kilka słów o nowej generacji systemów operacyjnych. Profesor Went z Hamburga wynalazł nową metodę opisu sprzętu. Nie wiem, czy widzieliście Państwo podręcznik Intelu o mikrokomputerze Intel 8080. Liczy on 300 stron. Profesor Went, który - nawiasem mówiąc - jest jeszcze młodym człowiekiem, gdy wykładał w Hamburgu, dał ten podręcznik swoim doktorantom. Mieli pracować na 8080. Wiadomo, że przed samymi egzaminami i zaraz po nich człowiek ma masę wiadomości świeżo w głowie; mimo to trzeba im było około 3 miesięcy, żeby pojąć i zapamiętać zawartość tych 300 stron.

Potem zastosowaliśmy inną technikę do opisu tego mikrokomputera. Podstawową zasadą tej nowej procedury jest to,że wychodzimy od jednostek pragmatycznych, aby podać jak są one zrealizowane na pastylce mikro, a nie na odwrót. Wychodzimy więc od znanych faktów i na nich budujemy opis. W ten sposób byliśmy w stanie opisać ten sam mikrokomputer tylko na 16 stronach zamiast na 300. Dzięki temu studenci byli w stanie przyswoić sobie w ciągu 4 dni to, czego inni z trudem nauczyli się w ciągu 3 miesięcy.

Teraz próbujemy zastosować to samo do oprogramowania, aby móc łatwiej opisywać istniejący program. Celem jest, by czytelnicy, którzy mają później pracować z tym programem, mogli szybciej zrozumieć, o co w tym programie chodzi, łatwiej zachować tę wiedzę w pamięci i łatwiej ją użytkować. Jeśli uda się nam - jeszcze nie jesteśmy pewni, a wiedzieć będziemy z końcem roku - stworzyć taką metodę opisową dla oprogramowania, wówczas metodę tę zastosujemy też do niej samej, aby uczynić tę dydaktyczno-opisową metodę prawdziwie przenośną; i aby przyswoić ją ludziom,

którzy później będą musieli definiować, analizować i dokumentować zadania. Oni zaś z kolei będą w stanie stosować prostsze i lepsze klasyfikacje, które będą łatwiej zrozumiałe dla specjalistów od komputerów i dla programistów. Być może uda się nam dojść do sytuacji, w której metody takie pozwolą nam udzielać komputerowi poleceń w języku naturalnym, tak że programowanie będzie można oprzeć na zastosowaniu normalnego języka ludzkiego.

Jest dokładnie piąta. Dano mi czas do piątej i nikt mi nie przerywał, nikt nie gwizdał. Przydałoby mi się jeszcze pięć minut, aby powiedzieć coś Państwu o drugiej generacji systemów operacyjnych,ale pozostawię to może do spotkania z referentami. Dziękuję Państwu bardzo.

SESJA I

PRAKTYCZNE ZAGADNIENIA NORMALIZACJI JĘZYKÓW

M.V.Vickers, Krajowe Biuro Normalizacyjne, Stany Zjednoczone

Wyrażano wiele obaw w związku z normalizacją języków i wpływem takich norm dla języka programowania. Wszystko co dotyczy COBOLu jest śledzone z dużym zainteresowaniem,zarówno przez użytkowników jak i przez realizatorów oprogramowania, ponieważ jest to pierwsza norma języka programowania, która spotkała się z czynnym poparciem i znalazła zastosowanie w społeczności informatycznej.W Stanach Zjednoczonych COBOL jest pierwszym i jak dotąd jedynym językiem programowania, który został znormalizowany do użytku federalnych instytucji rządowych.

Problematyka normalizacji języków programowania ma wiele aspektów. W rzeczywistości przyjęcie normy dla COBOLu zamyka jedną i otwiera inną erę, być może trudniejszą i pełną rozczarowań - erę zarządzania normą. Działalność na przestrzeni ostatnich 15 lat nauczyła nas przede wszystkim,jak skutecznie opracowywać i normalizować techniczne wymagania języka i jak realizować te wymagania w kompilatorach. Ale dopiero teraz zaczynamy rozeznawać problemy zarządzania użytkowym stosowaniem normy języka programowania i nabierać doświadczenia w tym zakresie.

Podejmowano szereg wysiłków mających na celu zarządzanie stosowaniem normy COBOLu, ale aż do niedawna wysiłki te ograniczały się głównie do organizacji,które mogły w sposób samodzielny podejmować decyzje i sprawować kontrolę.Te doświadczenia są nieocenionym wkładem do rozważań nad realną polityką i praktyką zarządzania wdrażaniem normy COBOLu w zróżnicowanym środowisku,

takim jakie stanowią federalne instytucje rządowe w Stanach Zjednoczonych, gdzie muszą być należycie zaspokajane rozmaite potrzeby różnych organizacji. Jednak na podstawie obecnej znajomości metod zarządzania stosowaniem norm języka wydaje się nieuniknione, że znajdą się tacy,których te normy pozbawią najlepszego technicznie lub ekonomicznie rozwiązania do wykonywania swoich zadań. Ważne jest zatem - i przewidujemy w naszym programie działalności - aby integralną częścią wszelkich planów wdrażania normy językowej był mechanizm badania i rozstrzygania uzasadnionych próśb o zgodę na odstępstwo od normy i od związanej z nią polityki wdrażania i użytkowania.

Podjęto już pewne pionierskie wysiłki dla znalezienia sposobów zarządzania normami języka programowania i ich skutecznego stosowania. Pod warunkiem, że nie uszczuplą one tradycyjnej pomysłowości i tradycyjnego nowatorstwa, normy języków programowania przyniosą radykalną oszczędność kosztów tworzenia, aktualizacji i przenoszenia programów.

Normalizacją COBOLu zajmuje się rozległy i skomplikowany zespół międzynarodowych, krajowych i federalnych grup i komitetów,współpracujących między sobą dla ustanowienia normy COBOLu.

Na ilustracji I1 przedstawiono schematycznie powiązania różnych komitetów i różnych działalności w ramach krajowego środowiska,w którym pracujemy; schemat ten w dużym stopniu tłumaczy niektóre aspekty naszej polityki i sposoby w jakie opracowaliśmy nasz mechanizm wdrożeniowy.

Opiszę krótko niektóre spośród tych działalności i rolę poszczególnych komitetów. Jestem pewna,że niektórym z Was znany jest Komitet Języka Programowania CODASYL, który od 1959 r. uczestniczy w opracowywaniu wymagań technicznych języka. Wyniki jego prac publikowane są w "Journal of Development", zawierającym wymagania techniczne języka. Te wyniki prac CODASYLu są następnie wykorzystywane przez Amerykański Krajowy Komitet Normalizacyjny COBOLu dla ewentualnego włączenia do normy krajowej[x/].

---

x/ W Stanach Zjednoczonych rozróżnia się Normy Krajowe /national standards/ jako ogólnie obowiązujące od Norm Federalnych /federal standards/ obowiązujących dla instytucji rządu Federalnego USA /przyp.tłum./

Komitet ANSI zajął się językiem źródłowym COBOL jako jedynym punktem wyjścia dla prac nad normami krajowymi. Oba te komitety nawiązały łączność z zagranicznymi grupami roboczymi, w szczególności z Japońską Organizacją Normalizacyjną i z ECMA TC6. Grupa CODASYLu otrzymuje ponadto materiały z Australii i Południowej Afryki.

My w rządzie federalnym USA bardzo aktywnie uczestniczymy w opracowywaniu norm krajowych. Mamy we wspomnianym komitecie ANSI szereg przedstawicieli rządu i wykorzystujemy dokumenty komitetu ANSI w naszej działalności w zakresie norm federalnych.

Na ilustracji I1 przedstawiono graficznie - poczynając od szczebla ogólnokrajowego - główne elementy składowe programu normalizacji COBOLu oraz ich wzajemne powiązania. Z tych wzajemnych powiązań widoczne są cztery podstawowe kierunki zainteresowania:

- wymagania techniczne języka i jego zastosowanie,
- kompilatory standardowe,
- ocena kompilatorów,
- intepretacja języka.

Przedmiotem dotychczasowej działalności rządu USA w zakresie normalizacji COBOLu było głównie definiowanie języka standardowego. Ale ogólny zakres prac, a ostatnio także działalność w ramach programu federalnego, są bardziej wszechstronne i mają za podstawowy cel zapewnienie szerokich instrukcji i kryteriów dla skutecznego stosowania ustanowionej normy. Poza problemem wymagań technicznych języka, inne zasadnicze problemy objęte federalnym programem normalizacji to:

- wypracowanie niezbędnej polityki i niezbędnych procedur w celu zapewnienia efektywnego stosowania normy językowej w procesie opracowywania programów użytkowych,
- opracowanie i ustanowienie odpowiednich mechanizmów dla wsparcia normy przez testowanie kompilatorów pod względem ich zgodności z normą,
- opracowanie polityki i procedur zapewniających rozwiązywanie zagadnień interpretacji normy językowej i wyników testów,

. opracowanie niezbędnej polityki i niezbędnych procedur udzielania użytkownikom federalnym pomocy w uzyskiwaniu znormalizowanych kompilatorów.

W marcu 1972 r. rząd federalny USA ogłosił przyjęcie Amerykańskiej Krajowej Normy COBOLu z 1968 r. jako Normy Federalnej. Normę Federalną zaktualizowano w grudniu 1975 r., zastępując normę z 1968 r. Amerykańską Normą Krajową z 1974 r.

Norma Federalna, przyjmując Amerykańską Normę Krajową, dopuszcza tylko cztery szczególne połączenia modułów przewidzianych w Normie Krajowej. To ograniczenie zestawu dopuszczalnych połączeń modułów było konieczne, by sprowadzić do minimum nieuniknione trudności dopasowania możliwości kompilatorów do funkcji języka rzeczywiście użytych w programie źródłowym. Każdy szczebel normy federalnej stanowi odpowiednio osadzony podzbiór następnego wyższego szczebla normy /ilustr. I2/.

Wraz z ustanowieniem Normy Federalnej stosowanie COBOLu stało się obowiązkowe we wszystkich programach użytkowych,które cechuje rozległa manipulacja znakami, zapisami i zbiorami oraz rozległe operacje wejścia/wyjścia, w przeciwieństwie do programów przeznaczonych głównie do rozwiązywania problemów obliczeniowych. Obok wprowadzenia obowiązku użytkowania COBOLu,Norma Federalna określa też politykę wdrażania tego obowiązku.

Przy wdrażaniu norm dla języków programowania możliwe są cztery podejścia,które zostały określone i były stosowane jeszcze przed zaistnieniem formalnie ogłoszonej normy.

1. Zastosowanie przedkompilatora do tłumaczenia nieznormalizowanego kodu w programach źródłowych na kod znormalizowany.
2. Zezwolenie, by kompilatory zawierały nieznormalizowane uzupełnienia i zezwolenie użytkownikom na stosowanie tych nieznormalizowanych uzupełnień, ale z równoczesnym wywarciem pewnego nacisku administracyjnego dla wyeliminowania tych niestandardowych cech przy pisaniu nowych lub modyfikowaniu starych programów.
3. Zezwolenie, by kompilatory zawierały niestandardowe uzupełnienia i zezwolenie programistom na stosowanie uzupełnień

opartych na potrzebach bieżącego programowania, bez względu na skutki, jakie te doraźne decyzje mieć będą na szersze cele normalizacji.

4. Stosowanie wyłącznie kompilatorów, które realizują tylko normę i tym samym egzekwowanie programowania wyłącznie w języku znormalizowanym.

Toczy się ożywiona debata nad zaletami podejścia drugiego w porównaniu z czwartym. Podejście drugie wypływa z zasady, że kierownikom ośrodków należy zezwolić na odgrywanie bardzo istotnej i czynnej roli w zakresie decyzji dotyczących właściwego stosowania norm, natomiast podejście czwarte wypływa z przekonania, że zarządzanie stosowaniem normy może się odbywać jedynie drogą ustawodawczą. Przy wdrażaniu Normy Federalnej przyjęliśmy w zasadzie ostrożniejszą metodę, to znaczy podejście drugie.

Gdy mowa o podejściu stosowanym przy realizacji Normy Federalnej, należy wziąć pod uwagę dwa ważne momenty:

- zagadnienia polityki w odniesieniu do użytkowego stosowania języka,
- zagadnienia funkcji kompilatora.

Przyjęta polityka wdrażania federalnej normy COBOLu stwarza obowiązek stosowania COBOLu we wszystkich programach uprzednio określonego typu, które są opracowywane lub nabywane do użytku instytucji rządowych, gdy:

- program jest centralnie projektowany lub programowany dla zdecentralizowanego systemu posługującego się komputerami różnych marek, modeli i konfiguracji,
- zachodzi możliwość, że program będzie wykonywany na sprzęcie innym niż ten, dla którego został pierwotnie napisany,
- przewiduje się, że okres użytkowania programu będzie dłuższy niż obecnie zainstalowanego sprzętu,
- program jest stale aktualizowany lub może ulegać częstym zmianom,
- korzyści z zastosowania języka wysokiego rzędu mogą mieć lokalnie duże znaczenie, niezależnie od problemu wymienności.

Dopuszczalne są tylko cztery wyjątki od obowiązku stosowania COBOLu:

1. Wolno używać języków generowania sprawozdań, zarządzania bazą danych i przetwarzania testów, gdy gwarantuje to efektywniejsze i ekonomiczniejsze zaspokajanie potrzeb programowania.

2. Wolno używać języków innych niż COBOL,gdy w istniejącym systemie liczącym,na którym mają przebiegać programy, COBOL nie jest dostępny.

3. Gdy ośrodek obliczeniowy jest nastawiony na wykonywanie naukowych i technicznych systemów użytkowych, zaprogramowanychw językach bardziej odpowiednich dla rozwiązywania tego rodzaju problemów programowania,systemy użytkowe, które normalnie powinnyby być zaprogramowane w COBOLu, mogą być zaprogramowane w lokalnie stosowanych językach.

4. Wolno stosować inne języki, gdy programy mają być wykonywane na sprzęcie,dla którego normalnie nie opracowuje się COBOLu.

Wszelkie inne odstępstwo od obowiązku stosowania COBOLu musi być uzasadnione analizą porównawczą, dowodzącą, że nad korzyściami wynikającymi z użycia standardowego COBOLu wyraźnie górują korzyści osiągalne przez zastosowanie innego języka. Zezwolenia w takim przypadku udziela szef danej instytucji, a nie lokalne kierownictwo ośrodka obliczeniowego.

Ponieważ przy takim podejściu wdrożeniowym kompilatory mogą zawierać cechy językowe wychodzące poza ramy wymagań normy, zagadnieniem była kontrola nad tymi niestandardowymi elementami języka. Wyrażano obawy, że bez jakiejś metody kierowania użytkowaniem niestandardowych cech w programach źródłowych, nastąpi poważne osłabienie normy. Do tych obaw w gruncie rzeczy sprowadzają się argumenty wysuwane na rzecz ustawodawczej metody wdrażania normy.

Po dłuższej ożywionej dyskusji i po głębszym zbadaniu sprawy postanowiono,że Norma Federalna nie powinna ustawowo wymagać stosowania wyłącznie standardowych elementów języka w programach, ale powinna stworzyć narzędzie,za pomocą którego zarówno kierownictwo jak i programiści mogą oznaczać niestandardowe kody w programach. Tak więc norma wymaga,by wszystkie kompilatory nabywane przez instytucje rządowe były zdolne do komentowania

list źródłowych w celu oznaczenia niestandardowych kodów użytych w programach. Tę zdolność kompilatora nazywamy "znakowaniem" /flagging/.

Po blisko trzech latach obowiązywania normy COBOLu uznano za celowe dokonać oceny rzeczywistego wpływu normy na bieżącą praktykę programowania w instytucjach federalnych. W związku z tym przeprowadzono na początku roku 1975 ankietę wśród starannie dobranej grupy kierowników ośrodków obliczeniowych.Kwestionariusz ankiety zredagowany był tak, by ustalić co wiedzą kierownicy ośrodków na temat federalnej normy COBOLu, w jakim zakresie ją stosują i jaka jest ich opinia o niej.

Wyniki ankiety dowodzą, że COBOL jest szeroko stosowany - 94% objętych ankietą ośrodków obliczeniowych w instytucjach federalnych, które mają dostęp do tego języka, rzeczywiście stosuje COBOL w takim czy innym zakresie. Podczas gdy prawie 60% ankietowanych oświadczyło, że ich programy w COBOLu posługują się wyłącznie COBOLem standardowym, mniej niż 20% respondentów podało, że w ich ośrodkach formalnie zabronione jest stosowanie niestandardowych uzupełnień w kompilatorach.Z tych ustaleń można wyciągnąć wniosek,że ze strony kierowników ośrodków istnieje dostateczne zainteresowanie użytkowaniem standardowego COBOLu, by zapewnić dobrowolne wdrożenie normy.

Drugi wniosek z ankiety wskazuje, że narzędzia stosowane przez programistów dla wspierania stosowania standardowych elementów języka w miejsce niestandardowych i dla oznaczania tych ostatnich, mają jak się wydaje, ograniczoną wartość. Z trzech najszerzej stosowanych narzędzi, dostarczone przez realizatora normy komentowane podręczniki są użytkowane przez 52% ankietowanych, natomiast standardowa publikacja ANSI jest użytkowana przez 37%, a nieformalna ustna komunikacja - przez 34% ankietowanych.

Choć Norma Federalna wymaga, by kompilatory obejmujące elementy językowe wykraczające poza ramy normy były zdolne do oznaczania i komentowania list programu źródłowego,gdy są w nim elementy niestandardowe, tylko 11% respondentów podało, że stosuje "oznakowanie" jako środek odróżniający użytkowanie języka

standardowego od niestandardowego. Okazało się,że niski stopień posługiwania się "znakowaniem" wynika z powszechnego braku odpowiednich kompilatorów na rynku, bo gdy zapytano ankietowanych kierowników, jakie narzędzia najbardziej chcieliby mieć do dyspozycji dla odróżnienia stosowania języka standardowego i niestandardowego, 70% z nich podało, że pragną znakowania list źródłowych przez kompilator. Ujawnione powszechne niestosowanie się dostawców kompilatorów do wymaganego przez normę znakowania, należy przypisać temu, że realizatorzy kompilatorów pragną w pierwszym rzędzie rozwinąć istniejące kompilatory, dostosowując je do wymagań językowych normy. Kompilatory ze zdolnością do znakowania zaczynają dopiero teraz pojawiać się na rynku.

Federalny program normalizacji zapewnia dwa narzędzia do oceny kompilatorów COBOLu i do pomocy ich użytkownikom.O jednym z tych narzędzi,znakowaniu uzupełnień językowych w listach programu źródłowego, już wspomniano.Drugim narzędziem jest dostępna wszystkim użytkownikom "Federalna Służba Testowania Kompilatorów COBOLu", która ma za zadanie badać stopień, w jakim kompilatory spełniają wymagania Normy Federalnej.

Potrzeba testowania kompilatorów COBOLu została uznana od samego początku prac nad obu normami - krajową i federalną. Zadanie opracowania środka do pomiaru zgodności z Federalną Normą COBOLu wykonano w 1973 r. w drodze porozumienia między Departamentem Obrony a Krajowym Biurem Normalizacyjnym /NBS/. Na podstawie tego porozumienia Departament Obrony przejął odpowiedzialność za wdrożenie, aktualizację i eksploatację Systemu Weryfikacji Kompilatorów COBOLu. Do wykonywania tych obowiązków Departament Obrony z kolei utworzył w ramach Departamentu Marynarki Wojennej Federalną Służbę Testowania Kompilatorów COBOLu. Służba Testowania jest już czynna i opublikowała dotychczas wyniki testów 16 kompilatorów COBOLu, przy czym wszystkie te testy z wyjątkiem ostatniego przeprowadzono na podstawie normy COBOLu z 1968 r.; ostatni test przeprowadzono na podstawie normy COBOLu z 1974 r.

Warunki porozumienia, na mocy którego działa Służba Testowania, przewidują,że ocena kompilatora COBOLu dokonywana będzie

na żądanie albo zainteresowanej instytucji rządowej albo dostawcy kompilatora, przy użyciu szeregu programów kontrolnych nazywanych zbiorczo "Systemem Weryfikacji Kompilatorów COBOLu". Każdy program kontrolny jest programem w COBOLu, obejmującym wiele testów pozwalających określić, czy pewna konkretna część Normy Federalnej została prawidłowo wdrożona. Wszystkie te programy kontrolne łącznie sprawdzają całość wymagań Normy Federalnej COBOLu. Te programy testowe zaczęto opracowywać już w roku 1967. Porozumienie między Departamentem Obrony a Krajowym Biurem Normalizacyjnym z 1973 r.,ustanawiające Federalną Służbę Testowania Kompilatorów COBOLu stanowiło kulminację kilku lat starań Krajowego Biura Normalizacyjnego, Departamentu Marynarki Wojennej i Departamentu Lotnictwa, by dać instytucjom rządu federalnego możliwość testowania kompilatorów COBOLu.

Departament Lotnictwa faktycznie zapoczątkował testowanie kompilatorów zanim jeszcze zajęła się tym Marynarka Wojenna. Uczynił to w postaci zlecenia udzielonego przemysłowi prywatnemu na opracowanie programów testujących.System testowania,który powstał w wyniku tego zlecenia, stał się podstawą, na której później Marynarka Wojenna oparła swój system testowania.

Wyniki oceny kompilatora zostają przeanalizowane, podsumowane i podane w dokumencie, które nazywamy "zbiorczym sprawozdaniem z weryfikacji". Każde takie sprawozdanie zbiorcze przedstawia niezależną analizę konkretnej wersji lub konkretnego wydania kompilatora działającego na konkretnym sprzęcie i przy konkretnym systemie operacyjnym. Sprawozdania zbiorcze nie są specjalnie przeznaczone do porównywania różnych kompilatorów, ani nie zawierają ostatecznych stwierdzeń co do dopuszczalności kompilatorów dla pewnych zamierzonych zastosowań. Sprawozdanie zbiorcze relacjonuje jedynie wyniki oceny kompilatora, określając - o tyle o ile sprawdzono to za pomocą programów kontrolnych - stopień, w jakim badany kompilator odpowiada normie COBOLu.

Jako kryteria zgodności kompilatora z normą, Norma Federalna wymienia dwa wymagania:

. wdrożenie musi obejmować wszystkie elementy co najmniej jednego szczebla Federalnej Normy COBOLu,
. wdrożenie musi spełniać wszystkie wymagania Amerykańskiej Krajowej Normy COBOLu.

W ramach oceny działalności testowania kompilatorów dokonano przeglądu dziesięciu sprawozdań zbiorczych. Na ilustracji I3 przedstawiono podsumowanie odchyleń od zgodności z normą odnotowanych w objętych przeglądem sprawozdaniach zbiorczych. W tabeli wyszczególniono procentowy udział odchyleń składniowych i wykonawczych z podziałem na szczeble i moduły Federalnej Normy COBOLu. Sumy wierszy i szpalt wskazują procent kompilatorów z odchyleniami składniowymi i wykonawczymi w przekroju poszczególnych szczebli i modułów.Peryferyjne pozycje wierszy i szpalt podają jaki procent objętych przeglądem kompilatorów wykazywał odchylenia wszelkiego typu. Podczas gdy tylko 40% sprawozdań zbiorczych donosiło o odchyleniach na wyższym pośrednim szczeblu normy, wszystkie kompilatory wykazywały odchylenia na jej najniższym szczeblu. Moduł "jądrowy" był pod tym względem najgorszy, gdyż 100% kompilatorów wykazywało odchylenia w tym module, natomiast najlepszy był moduł "segmentowania",gdzie tylko 20% kompilatorów wykazywało odchylenia.

Na podstawie dalszego przeglądu wyników testowania kompilatorów przeprowadzono analizę dla ustalenia tendencji, jakie zaznaczają się jeśli chodzi o zgodność kompilatorów z Normą Federalną.Każde z dziesięciu sprawozdań zbiorczych zostało przejrzane i obliczono, jaki procent programów testujących wykazał odchylenia od normy. Na ilustracji I4 wykreślono obliczone procenty jako funkcję czasu, tzn. jako funkcję daty pojawienia się kompilatora na rynku. Jeśli w danym roku pojawił się więcej niż jeden kompilator, podano przeciętną.

Na stopniowe ustanawianie mechanizmu testowania kompilatorów realizatorzy kompilatorów COBOLu zareagowali, jak się wydaje, opracowaniem kompilatorów bliższych znormalizowanym wymaganiom języka. W kompilatorach wprowadzonych po roku 1969 widać znaczną poprawę,a kompilatory wprowadzone w późniejszych latach poprawiły się jeszcze bardziej.

Od marca 1972 r. Norma Federalna COBOLu wymaga, by wszystkie kompilatory wprowadzane do inwentarza federalnego były zgodne z jednym ze szczebli normy COBOLu, Ale do niedawna korzystanie z usług Służby Testowania w celu sprawdzania kompilatorów nabywanych do użytku instytucji rządowych było dobrowolne i najczęściej oparte na indywidualnej polityce poszczególnych resortów federalnych, jeśli w ogóle można było o takiej polityce mówić.

W grudniu 1975 r. rząd federalny określił swoją politykę w tej sprawie, wymagając by wszystkie kompilatory wprowadzane do inwentarza federalnego były poddawane testowi. A dalej polityka ta wymaga, by w razie wykrycia przez test odchyleń od normy, odchylenia te zostały skorygowane w przeciągu 12 miesięcy od daty zawarcia kontraktu na dostawę kompilatora. Jeśli dostawca kompilatora nie skoryguje wykrytych odchyleń w przepisanym czasie, płaci karę umowną. W charakterze dodatkowej kary rząd federalny może ewentualnie uznać,że nieskorygowany kompilator nie może być przedmiotem jakichkolwiek późniejszych zakupów federalnych.

Wymagania wynikające z programu normalizacyjnego rządu federalnego USA odnoszą się do użytkowników federalnych, a nie do dostawców komputerów. Znaczy to, że dostawcy komputerów mają pełną swobodę decyzji, czy dostosować swoje produkty do Norm Federalnych, natomiast użytkownik jest obowiązany przestrzegać normy. Ale poprzez proces zakupów zaopatrzeniowych stwarza się kontakt między federalnym programem normalizacyjnym a dostawcami komputerów.

Rząd zakupuje swoje komputery poprzez proces selekcji przetargowej. Resort ogłasza zaproszenie do składania ofert, w którym precyzuje wymagania, jakim powinien odpowiadać system, aby mógł być wzięty pod uwagę przy selekcji. Między innymi wymaganiami zaproszenie ofertowe podaje normy jakim musi odpowiadać oferowany system. Potem prosi się dostawców o złożenie rządowi ofert przetargowych na pożądany system.Ten właśnie proces przetargowy stwarza bodziec do dobrowolnej realizacji kompilatorów zgodnych z normą. W tym to punkcie styku krytycznego znaczenia

nabiera istnienie skutecznej polityki i skutecznych procedur ustanowionych dla egzekwowania norm. Norma Federalna,ustanowiwszy obowiązek i kryteria zgodności i testowania kompilatorów wprowadzanych do inwentarza federalnego, stanowi ponadto, że polityka zgodności i testowania kompilatorów stosuje się do wszystkich kompilatorów COBOLu,niezależnie od tego, czy zostały opracowane we własnym zakresie, czy też nabyte w ramach zakupu całego systemu APD lub oddzielnie.

W celu rozeznania, gdzie polityka zakupu kompilatorów mogłaby wywrzeć najskuteczniejszy wpływ,postawiono we wspomnianej poprzednio ankiecie pytanie dotyczące pochodzenia kompilatorów użytkowanych obecnie przez instytucje federalne. Wyniki ankiety wskazują, że 51% federalnych ośrodków obliczeniowych użytkujących więcej niż jeden kompilator COBOLu nabywa te kompilatory dla komputerów różnych producentów. Ponadto ustalono, że federalne ośrodki obliczeniowe, posługujące się COBOLem posiadają przeciętnie po 1,6 kompilatora. Spośród kompilatorów użytkowanych przez instytucje federalne 73% nabyto w ramach zakupu całkowitego systemu liczącego, 16% nabyto od głównego dostawcy sprzętu, ale cena kompilatora była wydzielona z całości zakupu. Tylko 1% kompilatorów COBOLu użytkowanych przez instytucje federalne nabyto od niezależnych dostawców kompilatorów.

Język programowania jest to starannie zaprojektowany zbiór powszechnie uznanych metod, którymi programiści mogą się posługiwać do opisu konkretnych procesów komputerowych. Kompilator jest w zasadzie programem, który pośredniczy między tym "językiem" a kodem maszynowym komputera. Ten program może być realizowany tylko w bardzo ściśle określonych granicach, jeśli chodzi o przetwarzane dane i użytkowany sprzęt. Te własności kompilatorów nie pozwalają badać ich zgodności z normą w sposób taki, jak gdyby w grę wchodziły łatwe do określenia normy techniczne typu sprzętowego. Charakter kompilatora nie pozwala na zaprojektowanie go z absolutną doskonałością.Pewne utajone dwuznaczności w standardowych specyfikacjach języka ujawniają się dopiero podczas użytkowania języka, czy to przy realizacji kompilatora, czy przy opracowywaniu testów dla kompilatora i jego

rzeczywistym testowaniu, czy nawet przy pisaniu programów źródłowych.

Zasadniczym warunkiem prawidłowego zarządzania normą jest stworzenie mechanizmu zapewniającego rozwiązywanie wszelkich wynikających z normy zagadnień i jednolite stosowanie tych rozwiązań w stosunku do całej społeczności obsługiwanej przez normę. W tym celu zorganizowano Federalny Komitet Interpretacji COBOLu i obarczono go zadaniem pomocy w badaniu i rozwiązywaniu wszelkich problemów interpretacji Normy Federalnej COBOLu. Wszystkie takie interpretacje są uzgadnianie z Komitetem ANSI X3J4 i Komitetem CODASYLu ds. Języków Programowania,dzięki czemu osiąga się jednolite rozumienie języka nie tylko przez instytucje federalne,ale także przez tych, którzy czynnie uczestniczą w opracowywaniu języka.

Dotychczas wpłynęło do Federalnego Komitetu Interpretacji COBOLu tylko jedno zapytanie. Zapytanie to dotyczy przewidywalności wyników oceny wyrażenia arytmetycznego. Rozwiązanie tego problemu za pomocą specyfikacji języka pozwala,jak się okazało, realizatorowi przewidzieć wyniki oceny wyrażeń arytmetycznych; ale w toku dyskusji nad specyfikacją języka Komitet CODASYLu ds. Języków Programowania zmodyfikował język,aby zapewnić ściślejszą semantykę języka, dającą wymienne i przewidywalne wyniki wyrażeń arytmetycznych. Długoterminowym pozytywnym skutkiem zwracania się do organu interpretacyjnego będzie rozpoznanie i skorygowanie pewnych niedociągnięć języka. Skutkiem natychmiastowym jest zwrócenie uwagi użytkownika na obszar języka mogący ewentualnie dawać wyniki niewymienne między różnymi kompilatorami.

Poza istnieniem federalnej procedury interpretacyjnej,także Komitet ANSI X3J4 ma za zadanie udzielać interpretacji Normy Krajowej. Komitet ten udzielił już ponad 20 interpretacji normy COBOLu z 1974 r. Procedury interpretacyjne zarówno rządu federalnego jak i Komitetu ANSI X3J4 są stosunkowo nowymi pionierskimi przedsięwzięciami, toteż kompetencje i zasady postępowania w tej dziedzinie nie są jeszcze wyraźnie określone ani uzgodnione. Działalność obu tych komitetów w jej obecnej postaci

częściowo zachodzi na siebie. Jednak obie organizacje działają i współpracują z dużym oddaniem, wobec czego te niedociągnięcia w mechaniźmie interpretacji można będzie szybko i w sposób racjonalny rozwiązać, po czym obie procedury będą lepiej służyć programowi normalizacyjnemu, raczej dopełniając się niż konkurując między sobą.

Ustanowienie normy i opracowanie instrukcji wdrożeniowych dla tej normy nie zamyka samo przez się wszelkich problemów związanych z użytkowaniem systemów liczących. Norma COBOLu dotyczy małej, ale ważnej części zadania uczynienia komputerów sprawnymi i stosunkowo prostymi w użyciu - dostarcza mianowicie interface'u człowiek/maszyna. Ale język raz opracowany nie pozostaje statyczny. W toku użytkowania języka przez ludzi ujawniają się jego wady i niedociągnięcia, wskutek czego trzeba podejmować akcje w kierunku poprawy, udoskonalenia lub rozszerzenia definicji języka. Aby język był żywy, trwać musi ten ciąg cyklicznych zdarzeń tworzenia, wdrażania, użytkowania i doskonalenia. Istnieje skomplikowany mechanizm służący ciągłemu procesowi rozwoju COBOLu, aby język ten stale reagował na potrzeby użytkowników. W ramach rządu federalnego podjęto kilka akcji dla podniesienia użyteczności COBOLu jako narzędzia rozwiązywania problemów przetwarzania danych.

Ponieważ COBOL jest językiem "żywym", społeczność informatyczna ma duży stopień gwarancji, że także w przyszłości, gdy wprowadzone zostaną systemy liczące nowej generacji, COBOL będzie nadal zaspokajał jej potrzeby w zakresie przetwarzania danych. Ale obok korzyści posiadania coraz bardziej pomysłowych i coraz lepszych kompilatorów COBOLu, pojawia się też pewna krótkoterminowa niedogodność. W miarę dodawania nowych urządzeń, wpływają one w większości przypadków na już znormalizowane wymagania języka i ten wpływ prowadzi czasem do zmian w działaniu bieżąco użytkowanych funkcji języka. W niektórych innych przypadkach zdefiniowane przez realizatora zadania zostają usunięte na rzecz działania "standardowego", wyspecyfikowanego przez sam język.

Przeciętny programista piszący programy w COBOLu nie jest świadomy zmian zachodzących z dnia na dzień w wymaganiach języka. Uznając potrzebę poinformowania użytkowników COBOLu o różnicach w stosunku do pierwotnych wymagań Normy Federalnej COBOLu, opublikowano jako dokument FIPS PUB 43 przejrzysty podręcznik wszystkich zmian dokonanych w Normie Federalnej COBOLu. Dokument dostarcza dwóch rodzajów informacji pomocnej przy konwersji programów:

- szczegółową opisową listę uzupełnień, skreśleń i zmian języka w odniesieniu do poszczególnych modułów normy,
- złożony szkielet języka stanowiący porównawcze zestawienie składni języka według obu norm.

Ten materiał informacyjny ma stanowić pomoc przy konwersji programów dotychczas użytkowanych z kompilatorami zgodnymi z normą COBOLu z 1968r. do ich użytkowania z kompilatorami zgodnymi z normą COBOLu z 1974 r.

Wysokiej jakości dokumentacja programowa jest konieczna, ponieważ jest to główny środek komunikowania się i ponieważ stanowi wymierny wskaźnik postępu prac. W dążeniu do uzyskania spójnej i dobrej jakości dokumentacji i do usprawnienia komunikowania się, opracowano w ciągu szeregu lat wiele różnych norm dokumentacji, m.in. formularze do kodowania programów.

W ramach dalszych działań na rzecz realizacji Normy Federalnej COBOLu podjęto badania nad celowością opracowania standardowego formularza do pisania programów w COBOLu. Poza oczywistymi korzyściami wynikającymi z istnienia normy, którą można stosować przy programowaniu i przy szkoleniu programistów, wprowadzenie jednolitego formularza do użytku szerokiego kręgu zainteresowanych umożliwiłoby znaczne dodatkowe efekty ekonomiczne.

W ramach tych badań stwierdzono istnienie ponad 20 różnych formularzy do kodowania w COBOLu użytkowanych przez instytucje federalne. Dokonano analizy zawartości tych różnych formularzy i przeprowadzono wywiady z programistami, żeby ustalić ich opinie i poglądy co do pożądanej zawartości standardowego formularza do kodowania. Następnie ustalono kryteria, jakie należy sto-

sować przy włączaniu poszczególnych pozycji do formularza. Obok oczywistego wymogu zgodności z formatem COBOLu głównymi kryteriami były:

- łatwość kodowania - rozpatrywano wielkość poszczególnych pól do pisania kodu źródłowego COBOLu, rozmieszczenie numerów szpalt i wierszy, metody poziomego i pionowego oddzielania pól, nagłówki szpalt oraz liczbę wierszy i kolumn dostępnych dla rzeczywistego kodowania,
- informacja nagłówkowa - rozpatrywano informację służącą do kontroli prac rozwojowych, informację służącą rozliczeniom oraz specjalną informację unikalną odnoszącą się do konkretnego zadania lub konkretnej maszyny,
- wielkość formularza - rozpatrywano łatwość produkcji, wygodę oprawiania oraz powierzchnię zajmowaną przez nagłówki w stosunku do powierzchni rozporządzalnej dla kodowania,
- kolor - brano pod uwagę względy widoczności i odtwarzalności,
- upakowanie - brano pod uwagę druk formularza na obu stronach arkusza,możliwość oprawiania formularza w zszywki oraz zaopatrzenie go w dziurki dla oprawiania w skoroszytach.

Opracowany w wyniku tych prac standardowy formularz do kodowania znajduje się obecnie w końcowym stadium zatwierdzania jako Norma Federalna przewidziana do stosowania przez instytucje federalne.

Innym aspektem działalności rozwijanej w ramach federalnego programu normalizacji COBOLu są sympozja, spotkania dyskusyjne i instruktaże na tematy interesujące użytkowników COBOLu w celu poinformowania użytkowników o nowościach w zakresie języka oraz dla wymiany doświadczeń i omówienia zagadnień mających krytyczne znaczenie dla stosowania i dla treści normy COBOLu. Takie imprezy odbyły się na temat zarządzania bazą danych,przetwarzania asynchronicznego, tablic decyzyjnych w COBOLu, COBOLu w warunkach wielodostępu oraz - co najważniejsze - odbyła się publiczna dyskusja nad celowością restrykcyjnej polityki w odniesieniu do dopuszczalnej treści kompilatora spełniającego warunki Normy Federalnej.

W konkluzji można powiedzieć, że dzięki tej energicznej i

czasem pionierskiej działalności w ramach programu normalizacji COBOLu, założono cele normy, a mianowicie:

- podniesienie ekonomicznych efektów przetwarzania danych dzięki uproszczonym metodom i procedurom użytkowania urządzeń informatycznych,
- zwiększenie swobody wyboru nabywanych kompilatorów i tym samym nasilenie współzawodnictwa między dostawcami,
- zapewnienie większej elastyczności w użytkowaniu programów i danych na komputerach dostarczanych przez wszystkich dostawców,

zaczynają być realizowane lepiej niż kiedykolwiek dotąd. Jest przy tym w tym programie normalizacji szereg niedociągnięć: nie dość skuteczna polityka, niewłaściwe procedury i niewłaściwa treść techniczna oraz luki w dziedzinach,które jeszcze nie stanęły na porządku dziennym programu.Najbardziej odczuwalne braki w programie normalizacji dotyczą:

- opracowania właściwych mechanizmów, które by sprzyjały wdrażaniu i w razie potrzeby egzekwowaniu ustalonej realnej polityki i realnych procedur,
- podjęcia energiczniejszych wysiłków na rzecz przeszkolenia programistów w wydajnym i sprawnym posługiwaniu się standardowym COBOLem i związanymi z nim produktami.

Podsumowując wszystkie istotne elementy praktyki normalizacji COBOLu, należy stwierdzić, że rząd federalny nie sprawuje kontroli nad rozwojem ani normalizacją języka; procesy te odbywają się raczej przez dobrowolny wysiłek przemysłu. Rola rządu federalnego ogranicza się do popierania rozwoju i testowania dobrowolnych norm przemysłowych. Udział rządu federalnego sprowadza się do roli inicjatora i współuczestnika na równej stopie z użytkownikami przemysłowymi i realizatorami kompilatorów. Żywotnie ważna dla powodzenia normalizacji programu jest pewność, że przy wszelkich decyzjach należycie uwzględnione zostaną potrzeby i wpływy zarówno przemysłu jak i rządu federalnego.

Jak już powiedziałam na początku tego referatu, przy wdrażaniu normy wyłania się wiele problemów;nie do uniknięcia jest, że ktoś zostanie pozbawiony najdogodniejszego technicznie i e-

konomicznie rozwiązania dla wykonania swoich zadań. Mimo to, ze względu na globalne korzyści, jakie może dać stosowanie normy, trzeba iść naprzód, dążąc do osiągnięcia celów programu normalizacji, jednakże z pełną świadomością skutków różnych dostępnych alternatyw. My w rządzie federalnym USA obraliśmy szczególną drogę, która może nie będzie właściwa dla wszystkich organizacji - mogą być inne metody równie, jeśli nie bardziej, skuteczne niż podejście regulacyjne przez nas obrane.Przy obecnym poziomie naszej wiedzy i naszych umiejętności wdrażanie norm języków programowania niewątpliwie stwarza wiele trudnych problemów do rozwiązania przez nowatorskie umysły w naszej branży.

BUTLER: Dziękuję Pani bardzo,Pani Vickers. Gdy p.Vickers mówiła o producentach i wskazywała, że mogą swobodnie zdecydować, czy zastosują się do normy czy nie, przypomniały mi się słowa pani dr Davis na temat reprezentowanej przez nią siły nabywczej. Istnieje doskonałe wyrażenie amerykańskie na określenie siły nabywczej: nazywają ją "clout", co znaczy mniej więcej "siła uderzenia". Mam nadzieję, że p.Vickers wybaczy mi, gdy powiem, że kiedy wspomniała o swobodzie producentów niestosowania się do normy, przypomniało mi się powiedzenie Anatola France´a, że "prawo z majestatyczną bezstronnością zabrania zarówno bogaczom jak i biedakom sypiać pod mostem".

Może mógłbym jako pierwszy zadać pytanie. Czy uważa Pani, że wobec coraz większego udziału przetwarzania na bieżąco i w trybie dialogowym, utrzyma się akcent kładziony przez Was obecnie na COBOL?

VICKERS: Tak, z pewnością. Uważam,że warunki przetwarzania dialogowego i na bieżąco, choć na pewno są celowym i przyszłościowym trybem pracy dla użytkowników federalnych,nie wyprą całkowicie przetwarzania partiowego ani nie wyeliminują potrzeby tego, co teraz znamy jako nasze tradycyjne gospodarcze zastosowania informatyki.Toteż nadal potrzebny będzie język programowania.

BUTLER: Czy przewiduje Pani przyjęcie oddzielnej normy dla języka przetwarzania na bieżąco?

VICKERS: Teraz pracujemy nad normą dla BASICu, który jest niewątpliwie językiem przetwarzania na bieżąco. Rozważamy też możliwość opracowania normy dla PL/1. Jestem pewna,że potem przyjdzie kolej na inne normy, które zaspokoją potrzeby tych użytkowników, którzy działają w warunkach przetwarzania na bieżąco i przetwarzania dialogowego.

TESTER /British Oxygen/: Czy próbował ktoś obliczyć koszt programu normalizacji COBOLu?

VICKERS: Nie podjęliśmy takiej próby, ale jestem pewna, że jest to program bardzo kosztowny, sądząc choćby po liczbie uczestniczących w nim ludzi i ilości czasu, jaki poświęciliśmy dotąd, by dojść do punktu, w którym znajdujemy się obecnie.

TESTER /British Oxygen/: Czy potrzebne będą takie same nakłady dla każdego języka, gdy zechcemy stworzyć z niego język standardowy?

VICKERS: Obawiam się, że już tak jest w przypadku BASICu, PL/1 i FORTRANu. Jest to przykre, ale sądzę, że nie mamy lepszego sposobu niż arbitralna metoda komitetowa dla zapewnienia, że należycie chronione i reprezentowane będą interesy wszystkich.

BUTLER: Pragnąłbym pójść nieco dalej w sprawie poruszonej przez p.Testera. Nasunęło mi się pytanie, w jakiej mierze jest rzeczywiście pożądane przenoszenie systemów z jednych warunków w inne. Dla przykładu podam, że jednym z pożytecznych przymusów, który w wielu ośrodkach powoduje przeprojektowanie systemu,jest fakt, że czasem po prostu nie da się przenieść systemów użytkowych z jednej generacji sprzętu do innej. Niektóre z naszych systemów są dość stare i ich parametry projektowe wydają się przestarzałe nawet bez tej trudności. Gdybyśmy działali w prawdziwie standardowym języku, czy nie przenosilibyśmy tych dość marnych systemów z jednej generacji do drugiej, nigdy nie odczuwając potrzeby ich przeprojektowania?

VICKERS: Przenosząc programy kierujemy się kilkoma motywami. Oczywiście jednym z nich jest unowocześnienie sprzętu. Pośpiech przy zmianie systemów liczących czasem nie pozwala nam na natychmiastowe przeprojektowanie programów, wobec czego programy

muszą być przeniesione do nowego systemu w postaci takiej, w jakiej istnieją.

W Stanach Zjednoczonych mamy pewną liczbę dość dużych scentralizowanych zespołów programistycznych opracowujących programy, które później rozdaje się podległym ośrodkom obliczeniowym do wykonywania. Te podległe ośrodki nie posiadają lub nie mają do swej dyspozycji mocy programistycznych. Istnieje więc także takiego rodzaju wymiana.

We wspomnianej przeze mnie ankiecie zadawaliśmy konkretne pytanie o przenoszenie programów i okazało się, że 60-70% respondentów przenosi programy.

LEWI /Jansen Pharma/: Czy będzie norma dla APL?

VICKERS: My w rządzie federalnym nie pracujemy w tej chwili nad taką normą. Nie jestem zupełnie pewna,co dzieje się w odniesieniu do APL na płaszczyźnie normalizacji krajowej.

CHEVALIER /A.M.Berliet/: Kiedy użytkownicy odczują realne korzyści z nowych wersji COBOLu?

VICKERS: Normę z 1974 r. wdraża już szereg producentów. Weryfikacji, o której wspomniałam poprzednio, dokonano na Interdata 716, co jest moim zdaniem o tyle ciekawe, że jest to minikomputer. Wiem o szeregu kompilatorów, które w najbliższej przyszłości ukażą się na rynku i które będą zgodne z normą z 1974 r.

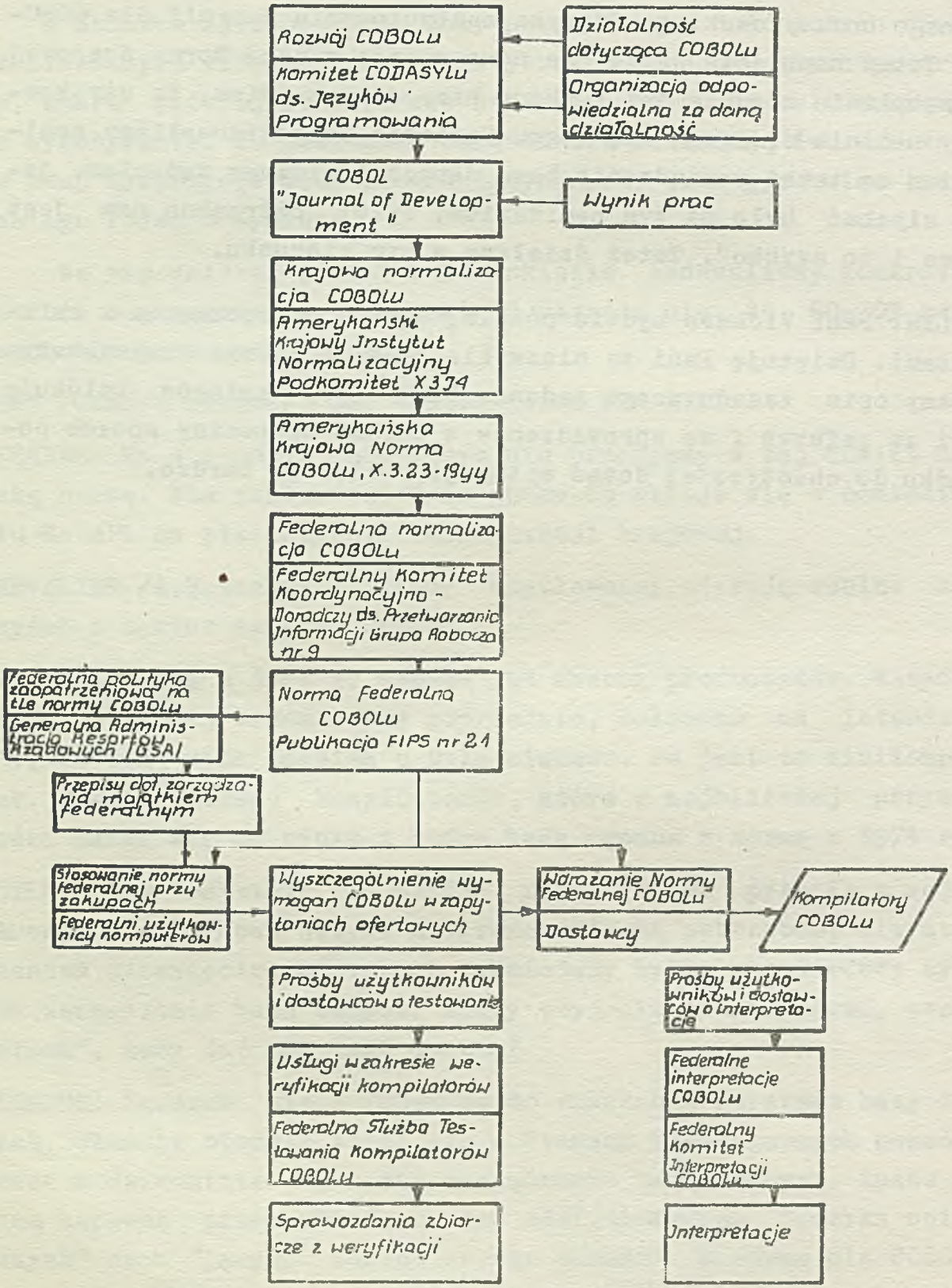
BUTLER: Oto ostatnie i bardzo podchwytliwe pytanie z mojej strony. Standardowe języki programowania są potrzebne, ale moim zdaniem dziesięciokrotnie potrzebniejszy byłby standardowy system zarządzania bazą danych. Kiedy użyjecie swego wpływu, owego "clout", żeby dać nam taki system?

VICKERS: Zgadzam się z Panem co do znaczenia systemu bazy danych. Właśnie obecnie toczą się w Stanach Zjednoczonych poważne prace w dziedzinie systemów zarządzania bazą danych. Znane są Panu zapewne prace CODASYLu nad zdefiniowaniem "języka opisu danych" oraz "języka manipulowania danymi" zarówno dla COBOLu jak i dla FORTRANu do komunikowania się z bazą danych opracowaną w "języku opisu danych".

Komitet Zarządzający organizacji ANSI zlecił ostatnio Komitetowi ds. Normalizacji COBOLu rozważenie niedawno zatwierdzonego codasylowskiego "języka manipulowania danymi" dla COBOLu. Toteż osobiście sądzę, że nowa wersja naszej Normy Krajowej niewątpliwie obejmie też funkcję bazy danych. Wiem, że użytkownicy usilnie domagają się tego. Ostatnio zorganizowaliśmy seminarium na temat zarządzania bazą danych i głównym żądaniem, jakie słychać było na tym seminarium, było: "potrzebna nam jest norma i to szybko". Toteż działamy w tym kierunku.

BUTLER: Pani Vickers będzie później obecna na spotkaniu z referentami. Dziękuję Pani za niezwykle jasny i dobrze usystematyzowany opis zasadniczego zadania Pani i jej kolegów. Dziękuję Pani za referat i za wprowadzenie w bardzo skuteczny sposób porządku do chaotycznej dotąd sytuacji. Dziękuję bardzo.

Ilustracja I 1



Wzajemne powiązania działań dotyczących COBOLu

Ilustracja I2. Norma federalna

Podzbiory języka

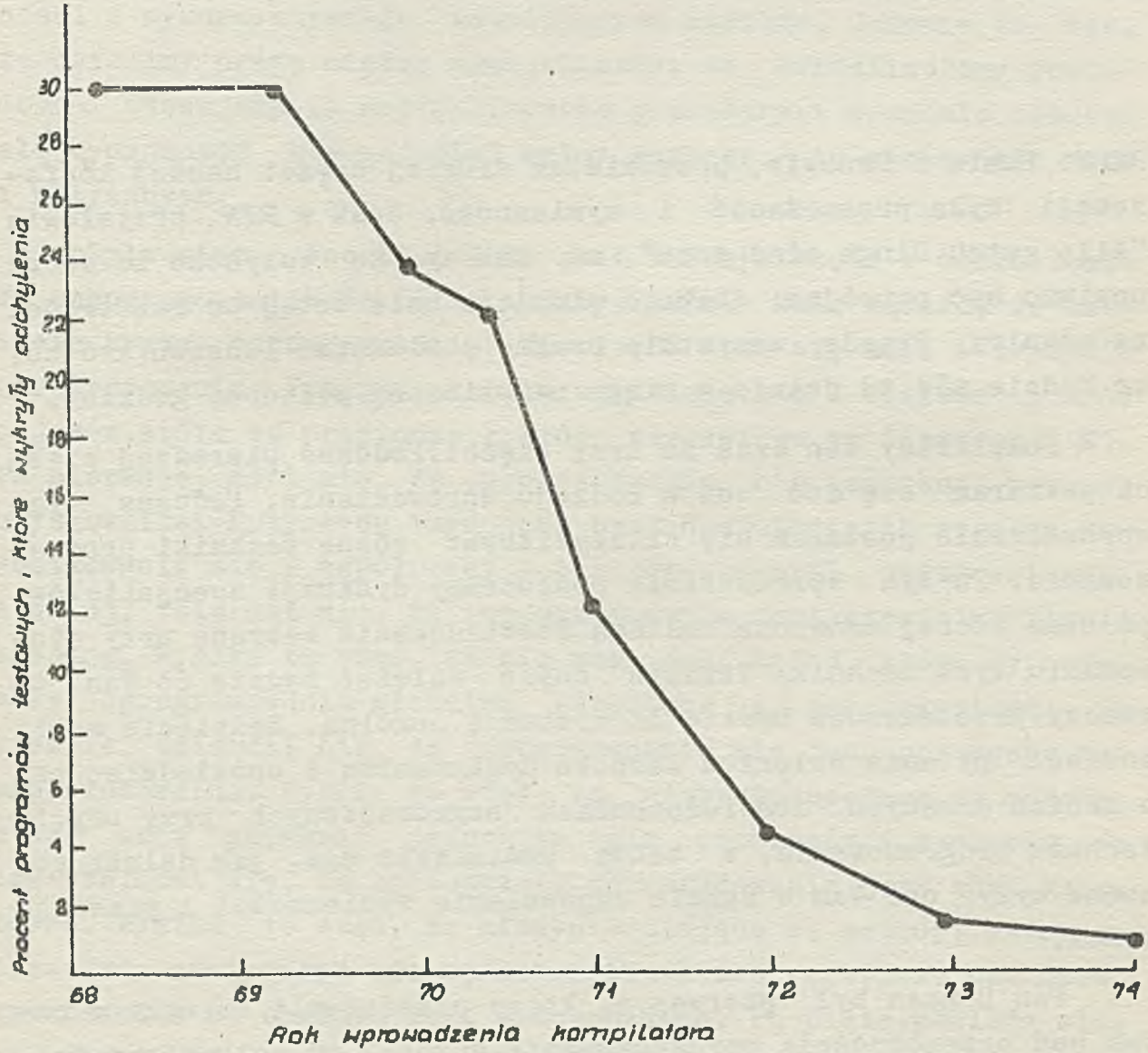
Moduł	Szczebel 1	Szczebel 2	Szczebel 3	Szczebel 4
Jądro	podzbiór	podzbiór	pełny	pełny
Manipulacja tablicami	podzbiór	podzbiór	pełny	pełny
Sekwencyjne we/wy	podzbiór	podzbiór	pełny	pełny
Względne we/wy	-	podzbiór	pełny	pełny
Indeksowe we/wy	-	-	-	pełny
Sortowanie - łączenie	-	-	podzbiór	pełny
Segmentacja	-	podzbiór	podzbiór	pełny
Biblioteka	-	podzbiór	podzbiór	pełny
Uruchamianie	-	podzbiór	pełny	pełny
Komunikacja między programami	-	podzbiór	pełny	pełny
Transmisja danych	-	-	pełny	pełny

Ilustracja I3. Procent zbiorczych sprawozdań z weryfikacji donoszących o odchyleniach od normy /w przekroju szczebli i modułów/

Moduł	% zbiorczych sprawozdań z weryfikacji donoszących o odchyleniach /składniowych lub wykonawczych/							Łącznie szczebel	% zbiorczych sprawozdań donoszących o odchyleniach wszelkiego typu
Szczebel	J	MT	Sekw.	Wzgl.	S.-Ł.	Segm	B		
Najniższy	80/90	20/10	20/40					80/90	100
Niższy średni	70/61	10/0	60/70	50/50		0/20	30/0	80/70	80
Wyższy średni					30/20			30/20	40
Najwyższy		10/30			20/40	0/0	50/0	50/50	60
Moduł łącznie	90/90	30/40	70/70	50/50	30/50	0/20	80/0	90/90	100
% sprawozdań donoszących o odchyleniach wszelkiego rodzaju	100	50	80	50	60	20	80	100	100

Ilustracja I 4

Odchylenia od normy w przekroju roku wprowadzenia kompilatora



SESJA K

KLASYFIKACJA TECHNIK PROGRAMOWANIA

Zespół dyskutantów koordynowany przez przedstawiciela Diebolda w RFN

HELM: Panie i Panowie, przedmiotem drugiej części naszej konferencji była przenośność i wymienność. Jest w RFN przysłowie "Alle guten Dinge sind drei", co znaczy, że wszystko co dobre powinno być potrójne; dlatego widzicie mnie tutaj po raz trzeci na mównicy. Przede wszystkim pragnę powiedzieć Państwu krótko, co będzie się tu działo w ciągu najbliższej półtorej godziny.

Podzielimy ten czas na trzy części. Podczas pierwszej części postaram się dać coś w rodzaju wprowadzenia. Podczas tego wprowadzenia postaram się sklasyfikować różne techniki programowania. Po tym wprowadzeniu posłuchamy dyskusji specjalistów, podczas której omówione zostaną doświadczenia zebrane przy stosowaniu tych technik. Trzecia część należeć będzie do Was, to znaczy przeznaczona będzie na dyskusję ogólną. Będziecie mogli zadawać pytania członkom zespołu dyskutantów i opowiedzieć nam o swoich własnych doświadczeniach nagromadzonych przy użyciu technik programowania, a także powiedzieć nam, jak dalece pomocne były one Wam w sensie zapewnienia wymienności i przenośności.

Pan Duncan był pierwszym, który przedstawił wnioski z badań nad przenośnością oprogramowania. Wczoraj po południu p. Wolters wspomniał o kilku kryteriach, które można zastosować w celu określenia, czy systemy są wymienne z punktu widzenia użytkownika, a pani Vickers omówiła znormalizowany język. Referenci ci omówili różne metody i różne zastosowania. Wszystkie referaty, jakie słyszeliśmy tu, wykazały bardzo dobitnie, że przenośność jest problemem nader złożonym i wielostronnym.

Moim zdaniem, można powiedzieć, że sednem każdego referatu było stwierdzenie - powtarzające się we wszystkich referatach - że oprogramowanie powinno być modułowe i znormalizowane.Wszyscy byli zgodni, że od tego zależy przenośność. Wykazano, że jest wiele sposobów rozwiązania problemu. Mówi się często, że stosujemy techniki inżynieryjne do przetwarzania danych, ale co to właściwie znaczy? Co to jest technika inżynieryjna? Oznacza to działanie w sposób uporządkowany, oparte na planowaniu działalności i wykorzystywaniu istniejących modułów. Oznacza to też, że dzielimy pracę między specjalistów; że normalizujemy procedury i stosujemy te znormalizowane procedury;i wreszcie ostatni ale bynajmniej nie najmniej ważny warunek - to stosowanie norm i wytycznych.

Tyle więc o teorii.A jaka jest rzeczywistość w odniesieniu do oprogramowania? Jakie są objawy, które każą sądzić, że istnieje kryzys oprogramowania? Wiecie,że dużo się mówi o kryzysie oprogramowania. Przypuszczam,że zgodzicie się z wieloma spośród objawów,które tu przytoczę i które zestawiłem na ilustracji K1. Po pierwsze, mówi się, że oprogramowanie nie zaspokaja potrzeb użytkownika. Przyczyną tego jest brak dostatecznych środków komunikowania się i współpracy z zainteresowanymi użytkownikami. A dalej, twierdzi się, że oprogramowanie jest często wadliwe i zawodne. Wynika to stąd, że nie mamy dość metod, które by nadawały oprogramowaniu właściwą strukturę i przejrzystość. Po trzecie, twierdzi się, że oprogramowanie nie jest przenośne ani przystosowalne. Wynika to stąd, że niewystarczające są metody, które mają zapewnić jednolity opis problemów i systemów. Po czwarte,mówi się, że opracowanie oprogramowania jest zbyt kosztowne. Wynika to stąd, że niewystarczające są metody, które by pozwalały zmniejszyć liczbę personelu.Prace rozwojowe nad oprogramowaniem są obecnie zbyt pracochłonne. Po piąte,powiada się, że często nie są dotrzymywane daty dostawy. Wynika to stąd, że niewystarczające są metody pozwalające planować prace i kontrolować ich postępy i koszty.

Podsumowując możemy powiedzieć, że kryzys oprogramowania wynika w gruncie rzeczy z niedostatku metod technicznych umoż-

liwiających strukturowanie, planowanie i kontrolę oprogramowania. Dlaczego tak jest? Głównie dlatego, że ludzie nie dość wiedzą o istniejących metodach. Inną przyczyną jest, że to co jest oferowane, nie jest łatwo zrozumiałe: oferowane metody są niedostatecznie przejrzyste. Wynika z tego, że konieczna jest klasyfikacja technicznych metod programowania.

W 1974 r. przeprowadzono w RFN badania nad oprogramowaniem w celu sprawdzenia sposobu wykorzystania personelu EPD, a także w celu zbadania sposobu wykorzystania istniejących metod. Uwidocznione tu wyniki /ilustr. K2/ wskazują, że metody te zostały nie dość szeroko rozpowszechnione. Stwierdzono, że co najwyżej 50% zbadanych przez nas użytkowników posiadało te metody. Oczywiście ta cyfra 50% wcale nie mówi nam, w jakim stopniu metody te są właściwie rozumiane i należycie użytkowane. W rzeczywistości więc procent wykorzystania metod jest zapewne znacznie niższy.

Jakie są tego przyczyny? Dlaczego metody i techniki programowania są niedostatecznie znane? Dlaczego są one nie dość szeroko stosowane? Są dwie grupy przyczyn /por. ilustr.K3/. Po pierwsze, użytkownicy nie zawsze rozumieją, jaka jest wartość użytkowa tych metod i jaki jest ich cel. A po drugie, istnieje pewien opór przeciw przyjęciu tych metod. Inny aspekt, o którym rzadko mówi się głośno, to niedoskonałość metodologiczna techniki. Ludzie są skłonni mówić: "w naszym przypadku problemy są tak specyficzne, że prawdopodobnie nie bylibyśmy w stanie rozwiązać ich za pomocą takich standardowych metod".

Na obecnej sesji postaramy się więc sklasyfikować te techniki programowania. Słyszeliśmy już jakie są skutki braku odpowiedniej metodyki. Widzieliśmy też, że gdy nawet istnieją takie metody, nie są dostatecznie znane; nie są też właściwie stosowane. I wreszcie widzieliśmy,że bardzo często wprowadzanie tych metod napotyka na opory.

Zanim spróbujemy sklasyfikować te techniki programowania, pragnąłbym zastanowić się wspólnie z Wami, jakim wymaganiom powinny odpowiadać te metody. Aby to uczynić,zastosujmy procedurę niewątpliwie dobrze Wam znaną, a mianowicie analizę efektywnoś-

ci ekonomicznej. Jak wiadomo, analizę efektywności ekonomicznej stosuje się w celu przekształcenia subiektywnych kryteriów oceny w kryteria obiektywne. Na pewno użyteczność i stopień akceptacji metod będą różne u różnych użytkowników. Wartość analizy efektywności ekonomicznej polega na tym, że zmusza ona Was do określenia pewnej liczby celów; że potem zestawiacie materiał i ujmujecie go w uporządkowany sposób; i wreszcie przystępujecie do oceny jako takiej. Wykonamy tutaj dwa pierwsze etapy, a mianowicie określenie celów i zestawienie przeznaczonego do oceny materiału, to znaczy zestawienie technik programowania. Zestawienie ich w sposób uporządkowany - oto co zamierzam teraz zrobić. Jeśli chodzi o ocenę, zrobicie to Wy.

Po pierwsze sporządzimy układ celów. Do czego służy taki układ celów? Układ celów jest potrzebny po to, by móc sobie odpowiedzieć na pytania takie jak: "kto odniesie korzyść z tych metod?", "która grupa użytkowników odniesie korzyść?" A następnie: "dla jakich funkcji metody te okażą się pomocne?" Po trzecie: "jakie decyzje trzeba będzie podjąć?" Odpowiedź na to pytanie wywrze oczywiście wpływ na łatwość przyjęcia metody. A dalej: "jakiego typu oprogramowania potrzebujemy?", "jakiej techniki?" i wreszcie: "jaki problem pomagają nam te metody rozwiązać?" Takie mogą być niektóre z celów jakie sobie stawiamy i w ten sposób tworzymy drzewo naszego układu celów /ilustr. K4/.

Możemy na przykład zadać sobie pytanie: "jakie będą korzyści z zastosowania takich technik,gdy zechcemy określić potrzeby w zakresie kadr lub w zakresie definicji problemu?" w przypadku, gdy zintegrowane oprogramowanie ma być częścią ogólnego systemu w dużej organizacji gospodarczej. W ten sposób można określić 34 cele pierwotne dla czterech typów oprogramowania przeznaczonych dla trzech typów użytkowników.

Ten układ celów rozbijamy na następujące grupy użytkowników: dostawcy oprogramowania, użytkownicy w wielkich organizacjach zatrudniających ponad 15 ludzi przy pracach rozwojowych nad oprogramowaniem oraz użytkownicy w małych i średnich organizacjach /ilustr. K5/. W naszym układzie celów podział ten wiązał się z pierwszym pytaniem o typ użytkownika,czyli grupę użyt-

kowników.Dalszy podział dotyczy typu oprogramowania użytkowego, dla którego te metody lub techniki mogą być przydatne i stwierdzamy, że można rozróżnić cztery typy oprogramowania. Po pierwsze, oprogramowanie systemowe; po drugie, zintegrowane oprogramowanie składające się na całokształt systemu; po trzecie, systemy specjalizowane,MVT itp.; i wreszcie, programy jednorazowe, bardzo często spotykane w systemach naukowo-technicznych /ilustr. K6/.

Można też rozróżnić te metody pod względem ich przeznaczenia w sensie funkcjonalnym. I tak np.metody te mogą mieć skutki w zakresie: produkcji oprogramowania, konserwacji oprogramowania, użytkowania oprogramowania oraz zarządzania oprogramowaniem.Nie będę oczywiście wchodził szczegółowo we wszystkie różne cele, których w sumie może być 400.Ale na pewno zainteresuje Was aspekt zarządzania i oto szereg celów, które bezpośrednio dotyczą zarządzania.Techniki programowania mogą dopomóc kierownictwu w ściślejszym kontrolowaniu operacji w jego sektorze, zapewnią, że kierownictwo będzie wiedziało,w jakim stopniu produkty lub procedury są odtwarzalne. Wszystkie te cele objęte układem celów pomogą do nakreślenia profilu wymagań, które powinny być spełnione przez metodę. Pośrednio pozwolą one odpowiedzieć na jeszcze inne pytanie, a mianowicie: "komu najbardziej potrzebne są te techniki? W jakich dziedzinach techniki te przyniosą największe korzyści?"

Przejdźmy teraz do innego aspektu,a mianowicie do problemu gotowości do przyjęcia metod.Tutaj trzeba będzie należycie uwzględnić psychologię, życzenia i cele personelu EPD. Nie przypadkiem wybrałem ten element jako pierwszy z tych,które chciałbym tu omówić. Zajmuję się szkoleniem i dlatego wiem, że tzw. "twórcza" lub "artystyczna" postawa personelu EPD jest przeszkodą w zadawalającej pracy.

Można być pewnym, że metody będą chętniej przyjęte, gdy będą należycie zrozumiane i jeśli wyraźnie widoczne będą korzyści i rezultaty z posługiwania się tymi metodami i technikami /ilustr. K7/. Istnieją ponadto inne subiektywne czynniki odgrywające pewną rolę,takie jak np. autorytet osób proponujących

te nowe techniki, intensywność szkolenia ułatwiającego zrozumienie tych technik, stopień, w jakim te metody i techniki były już uprzednio znane. Poza tym istnieje oczywiście problem kosztów wdrożenia. Tutaj trzeba należycie uwzględnić rzeczywisty koszt dzierżawy, koszt adaptacji oprogramowania, a także koszt szkolenia.

Gotowość do przyjęcia metody należy więc rozpatrywać z dwóch punktów widzenia: po pierwsze gotowość do jej wprowadzenia, a po drugie - gotowość do jej stosowania w charakterze pomocy decyzyjnej. W procesie decyzyjnym prowadzącym do przyjęcia metody uczestniczą liczne czynniki, z których każdy odgrywa określoną rolę /por.ilustr.K8/. Po pierwsze - kierownictwo naczelne, które określa cele. Po drugie - wydziały funkcjonalne, które mają swoje własne problemy. Po trzecie - specjaliści wytwarzający konkretne produkty. Po czwarte - ci, którzy mają wykonywać te decyzje; a mają oni swoje własne koncepcje, jak to należy robić. Poza tym jest sprzęt. Sprzęt wywiera oczywiście wpływ na zakres w jakim metody te mogą być zastosowane. I wreszcie, są konsultanci z zewnątrz. Wiecie, że jeśli zwrócicie się do konsultanta, możecie być absolutnie pewni, że wystąpi z zupełnie nową metodą, o jakiej nigdy przedtem nie słyszeliście.

Co omówiliśmy do tej pory? Stwierdziliśmy, że istniejąca sytuacja jest niezbyt zadawalająca. Staraliśmy się ustalić, jakie są przyczyny kryzysu oprogramowania. Staraliśmy się też wskazać, jak zbudować układ celów. Widzieliśmy też, z jakimi problemami wiąże się przyjęcie tych technik. A zatem położyliśmy podwaliny pod ocenę technik programowania.

Czym są te techniki? Czy rzeczywiście można je sklasyfikować? Czy można je zestawić w uporządkowany sposób? Znaczenie metody wynika z jej roli jaką odgrywa, gdy rozwiązujemy jakiś problem. Na samym początku mamy problem, czyli zadanie. Określone przez nas cele i rozporządzalne środki i narzędzia warunkują zakres, w jakim problem daje się rozwiązać, zaś metoda określa procedurę, jakiej trzeba przestrzegać, żeby dojść do rozwiązania.

Na tej ilustracji K9 spróbuję objaśnić koncepcję metody. Metoda jest to zaplanowany i uporządkowany sposób postępowania

ukierunkowany ku określonemu celowi i obejmujący użycie niezbędnych środków. Jest wiele różnych metod. Jest technika tablic decyzyjnych, są normy opisu, wytyczne do zarządzania pracami rozwojowymi, algorytmy, ustrukturowane programowanie, wytyczne dokumentacyjne itd. Czy są to rzeczy porównywalne? Czy można w ogóle mieć nadzieję na ich sklasyfikowanie?

Spróbujmy podejść do problemu od innej strony. Załóżmy, że mamy do rozwiązania pewien problem zastosowania EPD. Jakim narzędziem rozporządzamy? "Czarną skrzynką" w postaci komputera. U podłoża każdego problemu mamy jego model. Na przykład każdy magazyn będzie prawdopodobnie działał zgodnie z pewnym modelem sterowania gospodarką materiałową. U podłoża każdego systemu księgowości spotkamy prawdopodobnie pewien rodzaj modelu ekonomiki przedsiębiorstwa.

Są modele także i dla maszyn. I tak np. są komputery operujące bajtami i komputery operujące słowami, a oprogramowanie umożliwia nam wprowadzenie reprezentacji problemu do maszyny. Przy tym sam program będzie również oparty na pewnym modelu.

Istnieje kilka typów modeli. Przede wszystkim są modele problemów; są to abstrakcyjne reprezentacje pewnych dziedzin problemowych i różnią się między sobą stopniem szczegółowości. Następnie są modele produktów;są to środki opisu możliwego rozwiązania ujęte w absolutnie neutralnych wyrażeniach. I wreszcie - są modele maszyn.

Jak zdefiniowaliśmy koncepcję metody? Powiedzieliśmy, że metoda jest uporządkowanym i zaplanowanym sposobem postępowania odniesionym do konkretnego celu i obejmującym użytkowanie pewnych narzędzi.Wszystkie wspomniane przeze mnie modele są odniesione do określonego celu. I tak np. możemy utworzyć model produkcji. Jest to zupełnie neutralny rodzaj modelu,który wskazuje nam, jakie działania należy wykonać równolegle i kolejno, aby wyprodukować oprogramowanie. Przede wszystkim musimy więc zupełnie wyraźnie rozróżnić rozmaite typy modeli. Gdy to zrobimy, również metody ukierunkowane na te rozmaite modele można będzie podzielić na wyraźnie zróżnicowane typy.

Klasyfikacja modeli oznacza więc, że będziemy w stanie sklasyfikować również poszczególne metody odpowiednio do modeli,do których się odnoszą. Modele są oczywiście wysoce abstrakcyjne. Znaczy to, że i metody mogą być równie abstrakcyjne. Można je zatem zgrupować w typy metod, które z kolei będą odniesione do odpowiednich typów modeli.

Dobrze znane Wam pojęcia, takie jak systemy operacyjne, kompilatory, programy tłumaczące, generatory i bazy danych, są pojęciami, które odnosimy do modeli.Ale same przez się nie stanowią metod; tylko uporządkowane użytkowanie takich pojęć staje się metodą. Ponadto mamy inne pojęcia, takie jak programowanie konwersacyjne, przetwarzanie partiowe itd. Te muszą być odniesione do planowego działania, czyli do procedur. I one również nie stanowią jeszcze metody. Dopiero ich użytkowanie odniesione do modelu staje się metodą; na przykład etapowa dokumentacja oprogramowania. Można więc powiedzieć, że model plus planowe działanie równa się metodzie.

Zdefiniowaliśmy następujące cztery kategorie modeli: modele problemów, modele produktów, modele maszyn,a także modele produkcji - i to nie jeden, lecz kilka. Pierwsze trzy kategoriemodeli odnoszą się do konkretnych obiektów.Są one typowe dla EPD. Ale ostatnia kategoria, modele produkcji, ma uniwersalną zastosowalność.I tutaj właśnie potrzebne jest podejście inżynieryjne.

Przypatrzmy się teraz poszczególnym typom metod. Przede wszystkim istnieją metody ukierunkowane problemowo /ilustr. K10/. Tutaj należą zbiory i katalogi algorytmów oraz metody opracowywania programów za pomocą list pytań kontrolnych i kwestionariuszy. Cały ten typ metod można by nazwać "metody tworzenia programów"; zmienia się jedynie forma, jaką to tworzenie przybiera. Do tego właśnie typu należą konkretne metody takie jak algorytmy matematyczne lub znormalizowane programowanie, np. norma DIN 66/220 lub APT, a także generowanie programów za pomocą list pytań kontrolnych w IBM-owskim systemie programowania 32.

Jeśli chodzi o metody odnoszące się do modeli produktów /ilustr. K12/ to w tym przypadku spotykamy następujące typy me-

tod: po pierwsze - normy neutralnego opisu;słyszeliście Państwo może o metodach takich jak schematy HIPO.Są to produkty dostępne obecnie na rynku w RFN. Po drugie - metody ograniczające swobodę w odniesieniu do kodowania. Do tej grupy zaliczamy metody takie jak kodowanie wolne od skoków, kodowanie ustrukturowane itd. Po trzecie - makrostrukturowanie programów; do tej grupy należą konkretne metody, takie jak tworzenie maszyn abstrakcyjnych, metoda etapowego uściślania, programowanie modułowe itd. Po czwarte - formalne języki do wytwarzania struktur. Oraz po piąte - zintegrowane strategie testowania i wdrażania.

Podam tylko kilka słów kluczowych dla wskazania metod, które zaklasyfikowaliśmy do tej ostatniej grupy: wdrażanie odgórne z wzorcami próbnymi /dummies/ lub wdrażanie oddolne.

Jeśli chodzi o metody, które odnieśliśmy do modeli maszyn, wymienię tylko metody służące do pomiaru wydajności i do optymalizacji; wszystkie inne wykraczałyby oczywiście poza nasz temat, którym są techniki odnoszące się do oprogramowania.

Ale pamiętacie, że mamy jeszcze modele produkcji. Metody zaliczane do tego typu /ilustr. K11/ są procesowo ukierunkowane. Te typy metod opierają się na normach zarządzania pracami rozwojowymi i można je sklasyfikować na podstawie tych norm. Mamy tu bardzo wiele norm, które są specyficzne dla różnych organizacji. Praktycznie biorąc, każde przedsiębiorstwo i każdy koncern opracowały swoje własne normy. Ponadto jest na rynku kilka bardziej uniwersalnych metod, takich jak AUGWARE, BISAD, METHODUS itd. A dalej idą metody statystyczne i projektowe, takie jak PAC 1, PROMPT, PKS, PCS, Plan Control i inne. Podałem tylko kilka przykładów. Na pewno nie stanowi to wyczerpującej listy tych metod. Próbuję tylko pokazać, o co mi chodzi.

Następnie mamy metody dokumentacji programów: metody takie jak AUTOFLOW, QUICKDRAW, biblioteki dokumentacyjne producentów itd., obejmujące też wskazówki o modułach oprogramowania. Niektóre z tych metod wspomnieliśmy już wcześniej, a istnieją też i inne.

A dalej - skład zespołu programistów ukierunkowany na

strukturę oprogramowania. Jest to metoda znana skądinąd pod nazwą zespołu głównego programisty.

Wszystkie te przykłady wskazują, że istnieje szereg metod, bardzo różnych pod względem swego znaczenia. Możemy rozróżnić rozmaite koncepcje techniki i rozmaite narzędzia ustrukturowanego kodowania.

Jedna metoda może się opierać na kilku modelach, to znaczy że może ale nie musi być oparta na jednym modelu. Im większa jest liczba modeli leżących u podstawy danej metody, tym użyteczniejsza jest ta metoda.

Podsumowując, możemy powiedzieć, że konkretne metody można usystematyzować, dzieląc je na pewne typy metod. Typy metod mogą być odniesione do określonych typów modeli. Modele są różne pod względem stopnia konkretności i pod względem sposobu, w jaki są wyrażone. Modele mogą być ustrukturowane. Ocena poszczególnych znanych nam metod powinna odbywać się zawsze na podstawie określonego modelu.

W sumie, Panie i Panowie, staraliśmy się dokonać analizy efektywności ekonomicznej metod i w trakcie tej analizy otrzymaliśmy układ celów, a następnie sklasyfikowaliśmy metody w sposób uporządkowany.

Trzecią częścią analizy ma być ocena wartości tych metod. Wymieniliśmy kilka kryteriów, które mogą nam dopomóc w ocenie. Kryteria te mogą być ważone i ujęte ilościowo, ale istnieje poza tym szereg kryteriów jakościowych, takich jak np. łatwość użytkowania, stopień konkretności metody itd. Trudno to ująć ilościowo. Ważne jest natomiast, że wartość metody na pewno będzie zależała od liczby celów, jakie pozwala osiągnąć. Im większa liczba osiąganych celów, tym lepsza jest metoda. Mogą to być np. cele dotyczące produkcji oprogramowania, jego konwersji itd. A zatem jednym z kryteriów wartości metody jest zasięg jej oddziaływania.

Panie i Panowie, starałem się omówić z Wami szereg bardzo oderwanych pojęć. Starałem się wskazać, jak można sklasyfikować bardzo liczne i bardzo różne metody i techniki programowania,

tworząc coś w rodzaju systemu metodycznego. Uczyniwszy to,stworzyliśmy pewną podstawę i opierając się na niej możemy teraz przejść do drugiej części naszej sesji, a mianowicie do dyskusji. Dziękuję Państwu za cierpliwość i uwagę, z jakimi wysłuchaliście tego wprowadzenia. Dziękuję.

Panie i Panowie, pozwólcie mi przedstawić członków zespołu dyskusyjnego. Po mojej prawej stronie siedzi p.dr Griere obecny tu w charakterze przedstawiciela użytkowników w dziedzinie naukowo-technicznej. Jest on obecnie szefem bardzo ciekawego projektu medyczno-informatycznego realizowanego w ośrodku obliczeniowym Szpitala Diagnostycznego w Wiesbaden i jestem pewien, że będzie miał coś do powiedzenia nam. Po mojej lewej stronie siedzi p.Gewald. Jest on tu m.in. w charakterze przedstawiciela producenta, a mianowicie Siemensa. Jest odpowiedzialny za przedinstalacyjne oprogramowanie systemów informatycznych do celów gospodarczych. Trzecim dyskutantem jest p.dr Franssen z Bawarskiej Kancelarii Państwa w Monachium. Jego zadaniem jest planowanie i wdrażanie systemów informatycznych w instytucjach państwowych. Powie nam coś o związanych z tym potrzebach i wymaganiach.

Pierwsze pytanie chciałbym zadać Panu, P. Franssen.Jest Pan bawarskim funkcjonariuszem państwowym. Jakie są w związku z tym Pańskie specjalne wymagania,które powinny być zaspokojone przez omawiane przez nas metody programowania, mając na uwadze, że ogólnym tematem naszej dyskusji jest przenośność?

FRANSSEN: Dla użytkowników państwowych charakterystyczne jest, że mają bardzo wiele różnych komputerów. Użytkownik w postaci resortu rządowego to w rzeczywistości wiele różnych biur lub wydziałów, które mają wiele różnych komputerów. Są lokalne ośrodki z małym komputerem, są też duże ośrodki obliczeniowe należące do poszczególnych krajów związkowych lub do resortów rządu federalnego;i w zależności od charakteru resortu,problemy będą bardzo różne. Gdy się ma wiele komputerów, trzeba zdecydować, czy trzymać się wyłącznie jednego producenta, czy też wybierać najlepsze produkty, jakie różni producenci są w stanie dostarczyć. Są tacy, którzy nie chcą być uzależnieni od jednego

producenta, a skoro użytkują komputery pochodzące od różnych producentów, bardzo wcześnie napotykają na trudności pod względem przenośności i wymienności.

Państwo bawarskie wybrało to drugie rozwiązanie, a mianowicie nie trzymać się jednego producenta i dlatego bardzo wcześnie natknęło się na te problemy wymienności. Oprogramowanie przeznaczone do użytkowania w sposób zdecentralizowany na bardzo różnych komputerach trzeba było opracować w sposób zapewniający jego przenośność. Zawsze więc dbaliśmy o przenośność. A to oznaczało, że musieliśmy opracować normy programowania.

Toteż zarządziliśmy, że jedynymi dopuszczonymi do użytkowania językami są te, dla których istnieją normy. Innym wymaganiem było stosowanie wymiennych interfaceów z bazami danych. Wiecie Państwo może, że są w tej chwili w toku badania na ten temat finansowane przez Federalne Ministerstwo Badań Naukowych i Techniki. Wszystkie systemy bazy danych dostarczone przez różnych producentów będą uzupełnione takimi neutralnymi interfaceami.

Użytkownicy, którzy sądzili, że potrafili raz na zawsze uniknąć problemu wymienności, nabywając wszystkie swoje maszyny od tego samego producenta, nagle odkryli, że bynajmniej nie uniknęli tych problemów, ponieważ okazało się, że o ile na początku opracowywano proste systemy użytkowe, o tyle teraz nowo projektowane systemy użytkowe są znacznie bardziej skomplikowane, tak skomplikowane, że pojedynczy użytkownik nie może sobie już pozwolić na samodzielne opracowanie takiego złożonego rozwiązania. Dlatego coraz więcej użytkowników spośród władz publicznych łączy się w celu wspólnego opracowania skomplikowanych systemów użytkowych.

Pozwolę sobie przypomnieć np. opracowanie i wdrożenie systemu informacyjnego dla policji, systemu ksiąg wieczystych itd. W tym przypadku problemem jest nie tyle przenośność, ile przystosowalność, wspomniana wczoraj przez p.Helma. Procedury muszą być opracowane wspólnie i część z nich musi być wspólnie zaprogramowana, ale równocześnie te różne procedury muszą oczywiście zaspokajać potrzeby różnych użytkowników.

Ponadto władze publiczne utworzyły szereg grup roboczych, które po systematycznym zbadaniu rozwoju oprogramowania doszły do wniosku, że bardzo liczne zadania w różnych sektorach publicznych są bardzo podobne.Przedtem nigdy nie zdawano sobie sprawy z tego. Leżące u podłoża systemów problemy, takie jak np. model gospodarczy leżący u podłoża systemu księgowości, powinny być zebrane razem w celu zbudowania wspólnego abstrakcyjnego modelu. Na przykład wszystkie systemy gospodarki zapasami i kontroli zapasów można by sprowadzić do jednego szczególnego modelu, np. modelu programowania liniowego. Takie dane można też znormalizować, można by też znormalizować przenoszenie tych danych, ustalając np. że powinny być łączone w bloki.

Ważną rzeczą jest wykryć zawczasu, kiedy potrzebna będzie normalizacja,a także podjąć wcześniej wszelkie niezbędne wysiłki, by osiągnąć tę normalizację. Trzeba też określić i zastosować programy, które ułatwią to. Oznacza to, że za każdym razem, gdy staje przed nami nowe zadanie, musimy przede wszystkim starać się sprawdzić, do której abstrakcyjnej grupy modeli i metod można to zadanie odnieść.

HELM: A teraz inne pytanie, które omówiłem wcześniej prywatnie z kimś innym. Zastanawiam się, czy gdy władze publiczne wprowadzają wytyczne zmierzające do osiągnięcia większej wymienności i przystosowalności i gdy starają się narzucić te wytyczne, przychodzi im to łatwiej niż prywatnej organizacji? Czy możecie zadekretować: "od jutra mają wszyscy postępować w taki a taki sposób", czy też to nie jest takie proste?

FRANSSEN: No cóż,władze publiczne nie mogą narzucić swojej woli łatwiej niż ktokolwiek inny. Nie można narzucić swojej woli wielu różnym ciałom, które choć są częścią administracji państwowej, mają jednak pewien zakres niezależności. Nie jest to bynajmniej łatwiejsze niż w dużej organizacji gospodarczej. Ale muszę powiedzieć, że w wielu przypadkach opory, z jakimi spotykamy się, wynikają stąd, że podjęte poprzednio decyzje co do sprzętu, oprogramowania itd. bardzo utrudniają użytkownikom przyjęcie ogólnie obowiązujących norm. Dlatego staramy się pozostawić im maksimum swobody;ale normalizacją wszystkich funk-

cji, które takiej normalizacji wymagają, zajmuje się w Bawarii jeden centralny ośrodek.

Wymiana danych między różnymi systemami jest teraz możliwa i przebiega zadawalająco. Sytuacja pod tym względem jest więc bardzo dobra. Ale trzeba zawsze zdawać sobie sprawę, że użytkownik nie lubi czuć się zbyt skrępowany normami. Mimo to musi on również zdawać sobie sprawę, że zastosowanie takich norm umożliwia jemu samemu pewne rzeczy, które inaczej byłyby wręcz niemożliwe.

HELM: Panie Gewald, pracuje Pan w firmie, która m.in. produkuje oprogramowanie dla siebie samej i dla innych; jesteście oczywiście także użytkownikami oprogramowania. Co może dziś zrobić producent oprogramowania, aby ułatwić przenośność?

GEWALD: Wydaje mi się, że łatwość przenoszenia programów i danych między komputerami jest czymś, co leży w interesie nie tylko użytkownika, ale i producenta. Producent zrobi wszystko co w jego mocy, aby zapewnić łatwą przenośność. Między innymi czyni to podczas wytwarzania własnych produktów. Innymi słowy producent będzie zawsze starał się upewnić, że programy i dane nadają się do łatwego przeniesienia z wyprodukowanych przez niego starych komputerów na nowe komputery jego produkcji. Stara się to zapewnić w toku prac rozwojowych nad nowym komputerem. Opracowując nowe oprogramowanie zwraca w miarę możności należytą uwagę na problemy przenośności.

Nie jest to łatwa sprawa. Będą konflikty. Z jednej strony mamy postęp techniczny i nie chcemy go hamować, z drugiej strony są te problemy wymienności. Jedno, co można w tych warunkach zrobić, to nigdy nie tracić z oczu tej potrzeby wymienności w czasie prac projektowych i rozwojowych, które przynieść mają w efekcie nową rodzinę komputerów i tym samym nowe systemy operacyjne.

Tyle dla wprowadzenia. Istnieje także inny problem, a mianowicie problem przenośności programów i danych z komputera jednego producenta na komputer innego producenta. To zupełnie inny problem. Także i w tym przypadku producent będzie się starał udostępnić pomoce do konwersji; pomoce, które ułatwią doko-

nywanie takiego przeniesienia przy możliwie najmniejszych nakładach. I to również leży w interesie samego producenta, ponieważ zależy mu na tym,by zastępowano inne komputery jego komputerami.

Jeśli chodzi o te pomoce, nie tylko są programy dopomagające w konwersji, ale istnieje też inny bardzo ważny aspekt - przenoszenie umiejętności praktycznych, to znaczy wskazywanie ludziom, jak mają się zabierać do konwertowania swoich systemów użytkowych. I wreszcie jest oczywiście i trzeci punkt.Producent musi zawsze brać bardzo czynny udział we wszelkich pracach normalizacyjnych odbywających się na płaszczyźnie krajowej i międzynarodowej. Dr Franssen wspomniał już o niektórych formach takiej działalności normalizacyjnej.

Jest też i coś innego, o czym chciałbym wspomnieć. Ma to duże znaczenie dla przenośności. Istnieje grupa robocza EWG opracowująca sposoby i środki pozwalające osiągać wymienne bazy danych, kompilatory, systemy i transakcje, a zatem osiągnąć w ostatecznym efekcie prawdziwe wymienne oprogramowanie użytkowe.

I wreszcie, wydaje mi się,że producent komputerów powinien udostępnić swoim klientom i wszystkim użytkownikom pomoce, metody i "know-how", ażeby użytkownicy, gdy sami zaczynają pisać własne programy, byli również w stanie poświęcić dostateczną uwagę późniejszej przenośności tych programów. Ale o tym wiele już powiedziano, nie będę więc wchodził w szczegóły.

HELM: Dr Franssen wskazał nam,jakie są wymagania władz publicznych pod względem przenośności. Widzieliśmy, że władze publiczne nie różnią się pod tym względem od innych użytkowników. Chciałbym zwrócić się do dra Griere,który reprezentuje inny typ użytkownika. Pan, doktorze Griere, jest typowym użytkownikiem. Jakie znaczenie ma dla Pana przenośność? Czy uważa Pan, że ustrukturowane programowanie może być szczególnie pożyteczne?

GRIERE: Przede wszystkim pragnę podziękować za zaproszenie mnie do udziału w tym zespole dyskusyjnym. Niezupełnie dobrze czuję się, zasiadając w nim. Jestem z zawodu lekarzem i nie bardzo czuję się na siłach dyskutować o komputerach ze specjalistami

od komputerów. Ale chodziłem do szkoły z Jarem Franssenem i zajmuję się zastosowaniem informatyki w medycynie od tak dawna, jak on zajmuje się komputerami w swojej dziedzinie.

Mieliśmy następujący problem w dziedzinie medycznego zastosowania informatyki. Istniały dwa systemy programów, które trzeba było przeprogramować, a potem miały one być wdrożone i wykonywane w sposób zdecentralizowany na trzech minikomputerach dostarczonych przez różnych producentów. Innymi słowy mieliśmy dwa rodzaje problemów przenośności. Przede wszystkim trzeba było przejść od wielkiego centralnego komputera do małych komputerów; a przy tym te komputery nie pochodziły wszystkie od jednego producenta.

Po pierwsze, musieliśmy bardzo dokładnie wytestować pierwotny program. Zrobiliśmy to w taki sposób, że połączyliśmy z komputerem centralnym dalekopisy i udostępniliśmy je lekarzom użytkującym system, dzięki czemu uzyskali oni dostęp do komputera centralnego. Stosowane to było przez blisko dwa lata. Dopiero potem zajęliśmy się problemem przenośności jako takim. Zdefiniowaliśmy na nowo zbieranie danych, system ich przechowywania oraz drugi system programów - system rozpowszechniania informacji - używając do tego "znormalizowanego formularza Backusa". Dlaczego? Dlatego, że już wówczas zdaliśmy sobie sprawę, że istniejące kompilatory FORTRANu dla minikomputerów nie odpowiadają naszym potrzebom. Równocześnie byliśmy świadomi faktu, że w Stanach Zjednoczonych toczą się prace nad normalizacją MUMPSa. Niektórzy z Was pewnie słyszeli o tym. Ten ostatni może być wykonywany na minikomputerach ze zintegrowaną pamięcią.

Ale stwierdziliśmy, że nie jesteśmy w stanie posługiwać się nim, bo nie był łatwy w użyciu, toteż aby otrzymać system w wyznaczonym nam terminie, musieliśmy napisać swoje programy w FORTRANie. Musieliśmy bowiem być gotowi w określonym terminie, na koniec ostatniego roku danego programu rozwojowego. Był to warunek, który musieliśmy przyjąć i pogodzić się z nim.

Potem rozpoczęliśmy od trybu dialogowego z użytkownikiem, tworząc system od góry ku dołowi, z jednym tylko wyjątkiem - programu zarządzania dyskami. W tym przypadku musieliśmy wyjść

od najmniejszego z trzech rozporządzalnych komputerów i zaprojektowaliśmy program odpowiedni do jego wymagań. I wreszcie, zastosowaliśmy pomoce do programowania,po czym skonwertowaliśmy moduły strukturalne na programy w FORTRANie,wydawszy poprzednio dla ludzi, którzy mieli je napisać, zbiór reguł, których mieli przestrzegać.

Potem wypróbowaliśmy system na naszym komputerze centralnym, którym był komputer Siemensa.Teraz jesteśmy w toku procesu wdrażania. Pierwszy etap wdrażania mamy już za sobą i niektóre moduły zakodowane kodem bezskokowym /GOTO-free code/ zostały obecnie zastąpione programami w języku symbolicznym. To było stosunkowo łatwe.

Przekonaliśmy się, że jesteśmy w zasadzie w stanie dotrzymać terminu, jaki sobie wyznaczyliśmy. /Zamiast spodziewanych ośmiu miesięcy potrzebowaliśmy dziewięciu miesięcy./ Byliśmy w stanie bardzo ściśle określić zapotrzebowanie na siłę roboczą.

Potrzebowaliśmy kilku ludzi do kierowania pracami - jeden matematyk, jeden student matematyki i ja sam, a poza tym personel pomocniczy do pisania kodów; niepotrzebni byli nam w ogóle konwencjonalni programiści.

Stwierdziliśmy, że istnieje łatwość komunikowania się między matematykiem a lekarzem. Myśleliśmy, że będzie to stanowiło problem, ale przekonaliśmy się,że byli oni w stanie dość dobrze rozumieć się wzajemnie. System, który ostatecznie otrzymaliśmy, był prawdopodobnie bardziej wolny od błędów, niż gdybyśmy zastosowali bardziej konwencjonalną metodę. Dużym osiągnięciem było, że udało się nam dokazać,iż takie systemy mogą być przenośne. Przenośność wydaje się nam bardzo ważna, bo jesteśmy absolutnie pewni, że jeśli chodzi o minikomputery, rozwój będzie nadal bardzo szybki, a także dlatego, że nasz system dowiódł, iż systemy oprogramowania mają znacznie dłuższy okres użytkowania niż sprzęt.

HELM: Myślę, że wszystkie te odpowiedzi zawierają wiele bardzo cennych informacji, ale chciałbym zwrócić się ponownie do p. Gewalda i zadać mu jeszcze jedno pytanie.Produkujecie,sprawdza-

cie i uruchamiacie oprogramowanie, które użytkujecie sami, ale także i oprogramowanie, które udostępniacie klientom. Panie Gewald, czy napotkaliście na wiele trudności, jeśli chodzi o gotowość przyjęcia metod i pomocy? Czy w ogóle napotkaliście już na takie trudności? Chodzi o ową barierę psychologiczną,o której tak często się mówi. Czy mógłby Pan krótko odpowiedzieć na to pytanie, po czym zadamy je panu Franssenowi?

GEWALD: Jeśli chodzi o gotowość przyjęcia nowych metod i pomocy, które przyczynią się do większej przenośności, sądzę, że dwie przyczyny utrudniają to przyjęcie. Po pierwsze, większa przenośność oznacza, że oprogramowanie opracowane z uwzględnieniem przenośności jest kosztowniejsze, zaś korzyści ujawnią się dopiero znacznie później. Dlatego dość trudno jest nakłonić użytkowników, by przełknęli to droższe oprogramowanie. Gdy produkuje się oprogramowanie, ma się masę trudności. Często brak czasu, a czasem dostatecznej siły roboczej, aby wyprodukować to czego się chce.I wreszcie wychodzimy na rynek z gotowym produktem, a wówczas okazuje się on za drogi i trudno jest nakłonić ludzi do jego przyjęcia.

Poza tym jest jeszcze coś,a mianowicie postawa kadr, które zatrudniamy przy tym. Programista przyzwyczaił się do pewnego sposobu pisania programów. Nagle przychodzimy do niego i powiadamy: "nie, teraz będziesz stosował inne techniki. Musisz sporządzać dokładniejszą dokumentację i to w inny sposób." Ludzie nie lubią zmieniać przyzwyczajeń. Uważamy, że takie są dwie główne przyczyny trudności.

FRANSSEN: No cóż, zna Pan rodzaj reakcji, gdy przychodzi Pan z nową metodą. Ludzie powiadają: "hm, ta metoda jest może nawet bardzo dobra, ale trzeba ją dostosować do naszego specyficznego typu organizacji. Trzeba uwzględnić nasz własny typ organizacji." A wtedy Pan zapytuje: "jakie to są osobliwości waszego typu organizacji, do których należałoby dostosować metodę?" Na to pytanie ludzie nie potrafią Panu odpowiedzieć.

Myślę, że będziecie mogli łatwiej uplasować Wasze metody, jeśli te nowe metody będą opisane w sposób wskazujący, na czym polegają jej zalety. Następnie musicie wskazać użytkownikowi

bardzo wyraźnie, że metody, które przyczyniają się do przenośności, to - jak powiedziano wczoraj - na ogół te same metody, które ułatwiają aktualizację i przystosowywanie do nowo pojawiających się potrzeb. Jeśli powiecie użytkownikowi, że aby otrzymać bardziej racjonalny produkt,wystarczy po prostu przestrzegać pewnych reguł, które tylko z początku mogą mu się wydać trudne, zastanowi się on i powie: "no tak, skoro da to w efekcie racjonalniejszy produkt, to dlaczego nie?"

Dlatego nie uważam, by ta bariera psychologiczna była nie do pokonania.

HELM: Dziękuję Panu bardzo,panie doktorze Franssen. Przypomniał nam Pan o typowym problemie, z jakim spotykamy się, ilekroć staramy się wprowadzić konkretną metodę;ludzie powiadają: "tak, ale nasze problemy są zupełnie inne".

GRIERE: Gotowość przyjęcia.W naszym szczególnym przypadku byłem w stanie ocenić to tylko w ramach bardzo małego projektu badawczego;ale zainteresowany zespół pracował już od wielu lat razem i wystarczały mu tylko ogólne wytyczne dokumentacyjne.Sądziłem, że będę miał trudności z przyjęciem nowej metody. Dlatego uważałem, że powinienem sam dać przykład. Zacząłem sam badać metody,a potem opracowałem na ich podstawie metody testowe. Później zatrudniłem studenta matematyki nie znającego dotychczasowych metod, żeby przeszkolił mój zespół.

Przede wszystkim dałem mu całą literaturę na temat ustrukturowanego programowania, zespołu głównego programisty, programowania odgórnego itd. I powiedziałem: "niech Pan to przestudiuje, a potem napisze Pan program". Okazało się to bardzo pożyteczne. Zrobiłem to dlatego, że kiedyś poprzednio, gdy próbowałem nauczyć pomocniczy personel lekarski języka programowania, przekonałem się, że lekarz jest bardzo złym nauczycielem dla swojego personelu pomocniczego, podobnie jak główny projektant systemów też jest często niezbyt dobrym nauczycielem.

Potem zastosowaliśmy wytyczne FORTRANu, aby skonwertować nasze strukturogramy na programy w FORTRANie.Matematykowi oczywiście FORTRAN bardzo przypadł do gustu;przyjął go i poczuł się

w nim jak ryba w wodzie. Ale inni, praktycy EPD, również nie mieli większych trudności, m.in. dlatego, że metody te mogły być tak łatwo stosowane na maszynie do pisania. Co prawda jeden z naszych programistów wypowiedział pracę. Był to jeden z naszych najlepszych analityków i programistów, przyzwyczajony do samodzielnego pisania swoich programów i śledzenia ich aż do wnętrza komputera. Mieliśmy wśród projektantów systemów i programistów dwóch takich gwiazdorów i mieliśmy z nimi sporo trudności.

Trudności nie wynikały stąd, że nie podobała im się metoda zastosowana do przedstawienia problemów, ale podejście logiczne - podejście odgórne było czymś, z czym nie mogli się pogodzić.Rozbicie problemu na poszczególne elementy i modularyzacja były czymś, co nie bardzo im się podobało. Temu właśnie sprzeciwiali się.

Jakie wnioski można z tego wszystkiego wyciągnąć? Możemy, jak sądzę, powiedzieć, że nie napotkamy na żadne trudności z przyjęciem nowych metod,gdy będziemy mieli do czynienia z ludźmi chodzącymi po ziemi, z ludźmi, którzy nie uważają, że są wielkimi ekspertami i że wszystko już wiedzą. Nie będziemy mieli wielkich kłopotów z ludźmi, którzy zdają sobie sprawę, że istnieją ogromne trudności, które trzeba pokonać. Ale napotkamy na poważne trudności, gdy będziemy mieli do czynienia z niektórymi wielce wrażliwymi primadonnami, będącymi czasem autorami bardzo wielkich systemów i uważającymi, że już niczego więcej nie mogą się nauczyć.

HELM: A teraz Panie i Panowie, zanim zwrócę się do Was z pytaniem o Wasze własne doświadczenia w tej konkretnej dziedzinie, p. Franssen prosi jeszcze raz o głos.

FRANSSEN: Tak, jest pewna rzecz, z której nie chciałbym robić tajemnicy. Przyjęcie pewnych grup metod - systemów bazy danych i transmisji danych /DB/DC systems/- stało się łatwiejsze, gdy zaczęliśmy stosować pewne generatory i inne dostępne na rynku produkty. Przekazaliśmy je do zbadania grupom funkcjonalnym. Te grupy funkcjonalne mają za zadanie śledzić rynek. Mają zbierać wszelkie nowo pojawiające się metody,wypróbowywać je, a w razie

decyzji o przyjęciu i zastosowaniu tych metod, te właśnie grupy mają później wspomagać użytkowników i służyć im radą. W wyniku tego powstała organizacja typu macierzowego. Rozporządzamy pewnym potencjałem kadrowym,który można będzie zaczerpnąć z takich grup funkcjonalnych, gdy toczyć się będą prace nad konkretnym produktem. Innymi słowy,normalny programista nie musi zaprzątać sobie głowy wszystkimi nowymi metodami, które być może znajdą tylko wąskie zastosowanie. Do tego mamy specjalne grupy funkcjonalne, i stwierdziliśmy, że mamy dzięki temu mniej trudności w uzyskiwaniu przyjęcia nowych metod.

GEWALD: Doktorze Franssen, i ja również chciałbym powrócić do tej sprawy. Stwierdziliśmy coś bardzo podobnego, a mianowicie to samo zdarzyło się w naszym przedsiębiorstwie, u Siemensa, jako użytkownika komputerów. Stwierdziliśmy dokładnie to samo. Odkąd powołaliśmy doradców, którzy mają wskazywać, jak powinny być stosowane pomoce do programowania, odkąd ustanowiliśmy takie nowe funkcje i my również stwierdziliśmy, że nowe metody są przyjmowane znacznie szybciej.

HELM: Spróbowałem sklasyfikować różne techniki programowania, aby wprowadzić trochę porządku do tej całej dziedziny. Mam nadzieję, że okaże się to w pewnej mierze pożyteczne. Wymieniłem też wiele poszczególnych metod i starałem się wykazać,że stoimy w obliczu prawdziwej mozaiki tych metod. Mamy jeden kamyczek w górnym rogu, jeden w środku, a ponadto jeszcze jeden kamyczek dodany do pierwszego kamyczka, a całość jest budowana w sposób stopniowy i nie zawsze bardzo dobrze zorganizowany.

Jedną z trudności, na jakie napotykamy, jest to, że istnieje wiele niewymienności między tymi metodami. Jestem pewien, że każdy z Was napotkał kiedyś na takie trudności, gdy musiał przystosować te metody do swego sposobu działania.Byłbym bardzo wdzięczny, gdybyście Państwo zabrali głos i powiedzieli nam coś o swoich własnych doświadczeniach.

NIEZIDENTYFIKOWANY MÓWCA: Jeśli dobrze oceniam sytuację, przenośność jest dziś tematem dnia. Ale dotychczas, jak mi się wydaje, zajęliśmy się tylko dwoma aspektami przenośności. Po pierwsze, zajęliśmy się przenośnością z punktu widzenia ośrodka

obliczeniowego w przedsiębiorstwie użytkowniczym; po drugie - przenośnością o tyle, o ile wpływa ona na stosunki między użytkownikiem EPD a producentem. Oczywiście należałoby pomóc obu tym grupom. Ośrodkowi EPD w przedsiębiorstwie należy pomóc w bardziej efektywnym użytkowaniu jego komputerów, a producentowi trzeba dopomóc, by mógł sprzedać więcej i większych komputerów swoim klientom. Ale jest jeszcze jeden aspekt,który, jak mi się wydaje, pominięto, a mianowicie użytkownik końcowy - klient ośrodka obliczeniowego w przedsiębiorstwie, a zatem wydział użytkowniczy.

Mamy problemy przenośności w stosunkach między wydziałami użytkowniczymi a ośrodkiem EPD. Stosujemy metodę HIPO,aby przerzucić most między informatykami a wydziałami użytkowniczymi. Sądzę, że jest to jedyna metoda pozwalająca omawiać oprogramowanie z użytkownikiem w sposób znaczący.Ale jest jeden problem, którego nie udało się nam rozwiązać i jest to pytanie, które chciałbym zadać Panu lub zebranym tu. Czy znacie jakąś metodę lub może stosujecie jakąś metodę, która sprawia,że dokumentację sporządzoną zgodnie z techniką HIPO można przekształcić w programowanie ustrukturowane? Czy jest jakiś sposób zrobienia tego? Czy może mi ktoś odpowiedzieć?

GEWALD: Przede wszystkim trzeba by zobaczyć, jaki rodzaj problemów może Pan rozwiązywać za pomocą HIPO, a jakie problemy rozwiązuje Pan za pomocą ustrukturowanego programowania. HIPO umożliwia Panu scharakteryzowanie problemów bez uwzględnienia jakichkolwiek warunków ograniczających. Na przykład mówi się czasem, że trzeba stosować środki opisu o charakterze wyłącznie technicznym. HIPO zupełnie świadomie wprowadza ograniczenia. Problem musi być przedstawiony w jasny i ustrukturowany sposób; i w końcu, gdy dochodzi Pan do podfunkcji, czyli do końcowego etapu analizy, musi Pan - wciąż jeszcze nie wchodząc w sprawy techniczne - wskazać, jaka ma być wewnętrzna logika. Tu właśnie można by użyć ustrukturowanej formy opisu kodowego. Ale HIPO pozwala zrobić coś więcej, coś czego nie można by zrobić za pomocą samego ustrukturowanego programowania,a mianowicie funkcje logiczne Pańskiego systemu użytkowego zostają przeanalizowane

i sformalizowane; sformalizowane zostają też potrzebne do opisu parametry. Potem oczywiście będzie Pan musiał zająć się technicznym aspektem przetwarzania danych i tutaj natrafi Pan na techniczne ograniczenia. Jest to etap późniejszy. Musi Pan więc wyraźnie odróżnić z jednej strony funkcje, które system ma spełniać, a z drugiej strony - realizację techniczną tych funkcji. Ta ostatnia będzie oczywiście zależeć w bardzo dużym stopniu od już posiadanego sprzętu. Nie może wynikać bezpośrednio z Pańskiego opisu logicznego.

W Stanach Zjednoczonych podjęto pewne próby, ale były nie bardzo udane.Nie ma jak dotąd sposobu,który by pozwalał przejść od opisu logicznego bezpośrednio do realizacji technicznej przez proste dodanie do opisu pewnej liczby warunków EPD. Myślę więc, że są to dwa kroki, które wciąż jeszcze muszą być dokonywane oddzielnie.

HELM: Panie Kober, czy to wyjaśnienie zadowala Pana? Czy jest ono pomocne?

KOBER: Nie, nie sądzę, by można było dać na to pytanie tak prostą odpowiedź. Zgadzam się oczywiście, z tym co Pan powiedział, ale jeśli chcemy być realni, wydaje mi się, że już przy projektowaniu wykresów HIPO musimy brać pod uwagę ograniczenia techniczne wynikające z posiadanego sprzętu.Jeśli przyjęliśmy pewną procedurę, która będzie się stosować do całego problemu, musimy z góry uwzględnić sprzęt, który albo już mamy u siebie albo spodziewamy się mieć. Istnieje więc ów aspekt ekonomiczny, którego nie wolno pominąć. Zadając swoje pytanie, chciałem dowiedzieć się, czy jest jakaś metoda, która by nam umożliwiała skopiowanie, że tak powiem, procedury HIPO w odniesieniu do struktury programu. Ale odnoszę wrażenie, że nikt z Panów nie wie o istnieniu takiej metody.

GEWALD: Starałem się wyjaśnić, że to czego Pan chce jest niemożliwe. Ważne jest, żeby opisać problem jako taki, nie troszcząc się o ograniczenia techniczne. Na przykład potrzebne mi są pewne dane i te dane muszą być gdzieś przechowane, ale w czasie gdy opisuję to w kategoriach logicznych, nie troszcząc się jak będę te dane przechowywać.

FRANSSEN: Ja również myślę, że HIPO jest normą opisu problemów, tak samo jak "znormalizowany formularz Backusa" i być może jeszcze inne podobne metody, które pojawią się w przyszłości. Wiem, że w ramach programu badawczo-rozwojowego finansowanego przez Ministerstwo Nauki i Techniki RFN istnieje pewien system zmierzający do rozwiązania postawionego przez Pana problemu - mogę dostarczyć Panu informacji bibliograficznych.

Jest to opracowanie oparte na następującej koncepcji. Mamy język organizujący, w którym różne przebiegi są reprezentowane w notacji podobnej do notacji "znormalizowanego formularza Backusa"; umożliwia to przedstawienie połączeń i powiązań między różnymi danymi i w końcu dochodzimy do czegoś w rodzaju komputera procesowego sterowanego danymi. Jest to pierwsza próba tego rodzaju. Nie wiem jeszcze, czy okaże się to wielce użyteczne, ale włożono w to wiele pracy i wypróbowano przez długi okres w pewnym bardzo dużym koncernie przemysłowym. Może rzecz tego rodzaju doprowadzi do rozwiązania poruszonego przez Pana problemu.

GRIERE: Chciałbym powrócić do problemu, o którym wspomniał Pan na początku, a mianowicie do problemu użytkownika końcowego - nie działu EPD w przedsiębiorstwie, ale wydziału użytkowniczego. To co przedtem omówiłem dotyczyło systemu, który byłby użytkowany przez samego użytkownika końcowego. Użytkownik końcowy może użytkować go posługując się bardzo prostym językiem. Stoimy w obliczu gigantycznego problemu, a mianowicie tego, że "normalny" lekarz nie ma pojęcia o tym, co może robić komputer. A jeśli nawet ma jakieś pojęcie, jest ono z gruntu fałszywe. On jest naszym użytkownikiem końcowym. Dlatego projektując system musimy dostosować go do poziomu wiedzy tego użytkownika i musimy bardzo starannie prowadzić tego użytkownika końcowego za rękę i pomagać mu, ażeby żądania jakie będzie wysuwał wobec komputera były sensowne.

Zbudowaliśmy system z funkcjami w zasadzie neutralnymi i funkcje te mogą być za pomocą dość prostego języka interpretacyjnego przystosowane do konkretnych systemów użytkowych. Innymi słowy, na tym szczeblu działamy w trybie dialogowym. Wolimy napisać pięć lub dziesięć programów, które są wszystkie praktycz-

nie jednokrokowe, niż liczyć na to, że wszyscy lekarze potrafią w sposób liniowy definiować systemy, definiować wymagania itd., coś czego absolutnie nie są oni w stanie zrobić w racjonalny sposób.

Sądzę, że tak właśnie będzie Pan musiał postąpić, ilekroć Pańskim użytkownikiem końcowym będzie osoba, którą nigdy przedtem nie miała do czynienia z komputerami. Myślę, że będzie Pan musiał to zrobić, żeby uniknąć poważnych kłopotów, w przeciwnym bowiem razie użytkownik będzie raz po raz przychodził do Pana mówiąc: "chciałbym robić to, jak również owo", a wynikiem tego będzie monstrualny system, zupełnie niepodobny do tego, co miał Pan na myśli na początku.

GEWALD: Panie Kober, chciałbym jeszcze coś dodać w odpowiedzi Panu. My również uważamy, że HIPO jest bardzo dobrym środkiem komunikacji. Ułatwia komunikację między działem EPD a wydziałem użytkowniczym. Niewątpliwie istnieje poważny problem, a mianowicie jak przekształcić wykresy HIPO na architekturę oprogramowania. Niestety muszę przyznać, że nie znam żadnej metody,która umożliwiałaby sformalizowanie tej konwersji. Prawdopodobnie nie pozostaje nam nic innego jak tylko tworzyć architekturę oprogramowania w charakterze oddzielnego kroku,po otrzymaniu wykresów HIPO. Ale oczywiście tę konwersję wykresów HIPO na oprogramowanie we właściwym znaczeniu tego słowa trzeba udokumentować tak, aby podtrzymać ów dialog między ośrodkiem obliczeniowym a działem użytkowniczym podczas całego okresu użyteczności oprogramowania.

My u Siemensa,aby rozwiązać ten problem komunikacji między działem EPD a wydziałami użytkowniczymi, stosujemy pewną koncepcję organizacyjną. Wydział użytkowniczy wciągnięty zostaje do uczestnictwa bardzo wcześnie,już podczas projektowania oprogramowania. Staramy się, by uczestniczył we wszystkich pracach. Uważamy, że nawet w czasie wstępnego projektowania oraz przez cały czas realizacji programowania i przebiegu próbnego, użytkownika należy wciągać do udziału we wszystkich decyzjach,które mają być podjęte. Staramy się, by czuł on rzeczywiście, że może przyjąć produkt końcowy, bo przez cały czas uczestniczył w jego opracowywaniu.

Chciałbym uczynić jeszcze jedną uwagę. Wiem, że podjęto jeszcze inną próbę przejścia od opisu problemu wprost do komputera. Jest to opis problemu za pomocą wykresów przejściowych. Opisujemy problem semantycznie i składniowo w języku wykresów przejściowych. Jest to język, który pozwala na bezpośrednią realizację w charakterze maszyny wirtualnej zarówno na dużych komputerach jak i na minikomputerach. Koncepcja ta została wynaleziona przez Wassermanna z San Francisco. Drugi, wspomniany poprzednio przeze mnie system został opracowany przez Martina.

CHEVALIER /A.M.Berliet/: Wszyscy mówcy z RFN wydają się bardzo dobrze obeznani z metodą HIPO,ale sądzę, uczestnicy konferencji innych krajów w ogóle nie znają tej metody. Chcielibyśmy wiedzieć dokładnie, z czego się ona składa i jakie są dostępne dokumenty informacyjne opisujące tę metodę HIPO.

HELM: Dziękuję Panu bardzo, Panie Chevalier. Spróbujemy może bardzo zwięźle opisać tę metodę. Oczywiście nie mamy zbyt wiele czasu. Możemy Panu powiedzieć, kto ją wynalazł, kto ją oferuje na rynku, ale nie sądzę, byśmy mogli powiedzieć Panu, ile ona kosztuje. Kto chciałby spróbować? Pan Kober. Proszę bardzo.

KOBER: Metoda HIPO. Pierwszy raz spotkałem się z nią w 1974 r. Oferował ją IBM. Nazywała się "Metody do rozwiązywania problemów". Problem zostaje najpierw ustrukturowany w postaci hierarchicznej. Zostaje rozbity na dużą liczbę bardzo małych jednostek; te zaś zostają później ustrukturowane hierarchicznie. Jest to tzw. struktura pionowa, ale istnieje też poziomy typ strukturyzacji.W tym ostatnim przypadku przyjmuje się za punkt wyjścia pożądane rezultaty, czyli pożądany materiał wyjściowy. Wychodząc od niego zmierzamy ku pożądanemu materiałowi wejściowemu i wymaganemu przetwarzaniu. Doprowadziło to do powstania metody HIPO jako takiej - hierarchicznej procedury wejścia/wyjścia /hierarchical input/output procedure/.

A zatem, Panie Chevalier, metodę tę produkuje IBM. Dlaczego więc nie zwróci się Pan do p.Davida? Jestem pewien, że będzie w stanie dopomóc Panu, choć temat, który wczoraj omawiał on, był nieco inny. SNA jest oczywiście zupełnie inną metodą, mimo to jednak ... Nawiasem mówiąc, zapytanie nic nie kosztuje.

HELM: Oto dlaczego powiedziałem, że nie będziemy wchodzić w sprawę kosztów, skoro wiemy, kto jest dostawcą. HIPO jest oferowane przez IBM, a drugą metodą jest ustrukturowane programowanie, czyli ustrukturowane kodowanie. Obie razem można objąć wspólną nazwą IPT /improved program techniques - udoskonalone techniki programowania/. Proszę Państwa, zapytywałem Was, czy macie nam coś do powiedzenia o Waszych trudnościach w związku z tym. Czy ktoś chce zabrać głos? Widzę, że wszyscy już marzycie o kawie. Temat, który omawialiśmy, jest oczywiście bardzo suchy i niestety musieliśmy go omawiać w bardzo abstrakcyjnych kategoriach. Chciałbym przypomnieć,że o piątej po południu odbędzie się sesja poświęcona spotkaniu z referentami i może wówczas zechcecie Państwo powrócić do jednego lub kilku punktów obecnej sesji. Dziękuję Państwu bardzo.

Ilustracja K 1

Objawy kryzysu oprogramowania

- oprogramowanie nie zaspakaja potrzeb użytkownika
- oprogramowanie jest często wadliwe
- oprogramowanie jest nierozprowadzalne /nieprzystosowalne i nieprzenośne/
- opracowanie oprogramowania jest zbyt kosztowne
- terminy dostawy często nie są dotrzymywane

Ilustracja. K. 2

Wdrożenie technicznych metod dotyczących oprogramowania

	1974	
	W	WK
Zarządzanie pracami rozwojowymi	17	7
Znormalizowany opis problemów	23	8
Tablice decyzyjne	31	22
Znormalizowane programowanie	42	28

W = wdrożenie narzędzi metodycznych
WK = wdrożenie wspomagane komputerem
} w % użytkowników APD

Początek 1976r

- Maksymalnie 50% użytkowników posługuje się technicznymi metodami do opracowywania oprogramowania
- Maksymalnie 30% użytkowników posługuje się narzędziami wspomaganymi komputerem

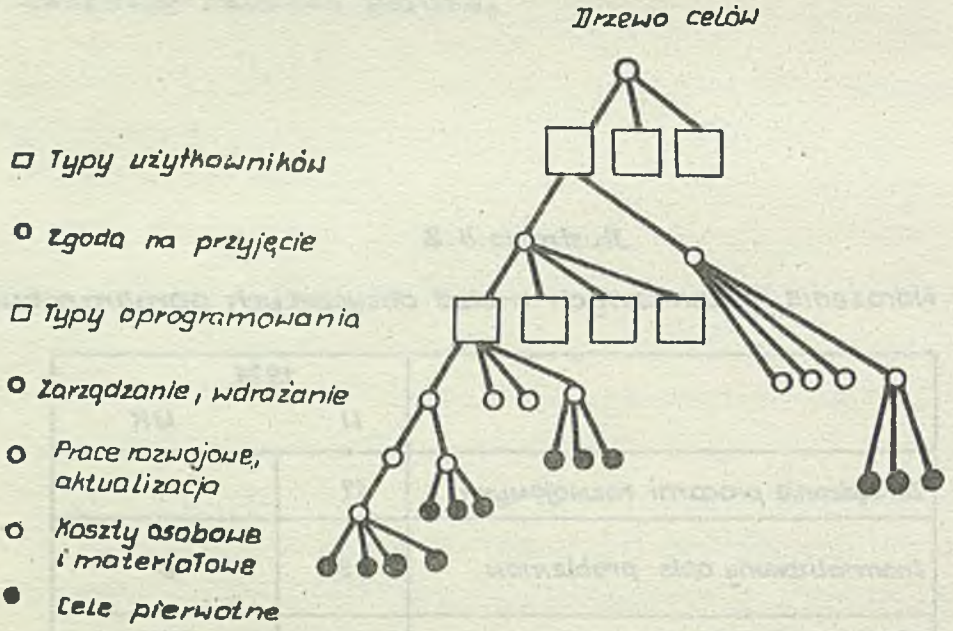
Ilustracja K 3

Przyczyny stosunkowo małego rozpowszechnienia technicznych metod programowania

(1) ● Niedostateczna przejrzystość rynku dla użytkownika
(2) ● Wysokie koszty metod
(3) ● Mała przejrzystość korzyści wynikających z metod
(4) ● Brak pracowników zdolnych do wyboru metod
(5) ● Niedostateczny opis metod
(6) ● Mała wymienność metod
(7) ● Niedostateczne uwzględnienie psychologicznych celów użytkowników metod

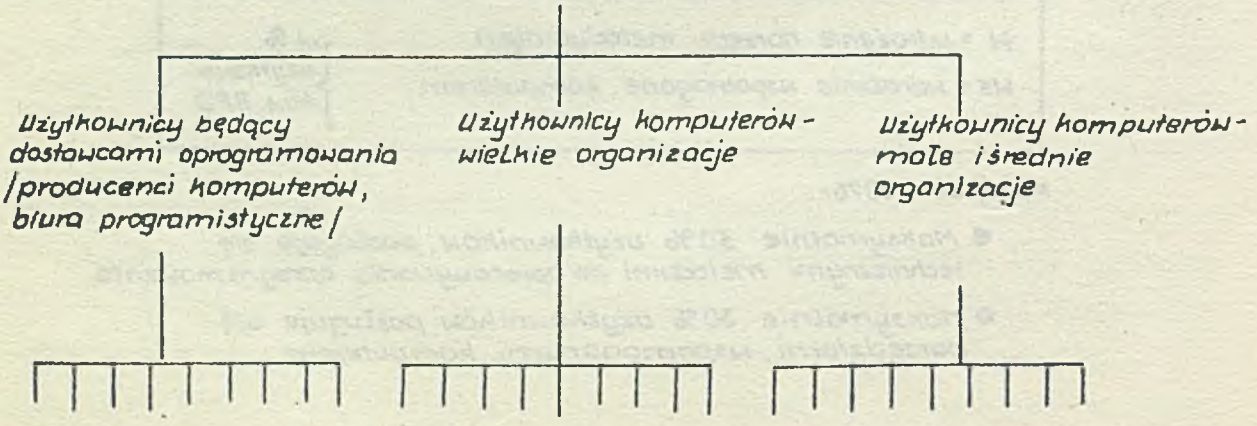
Ilustracja K 4

Przegląd struktury układu celów



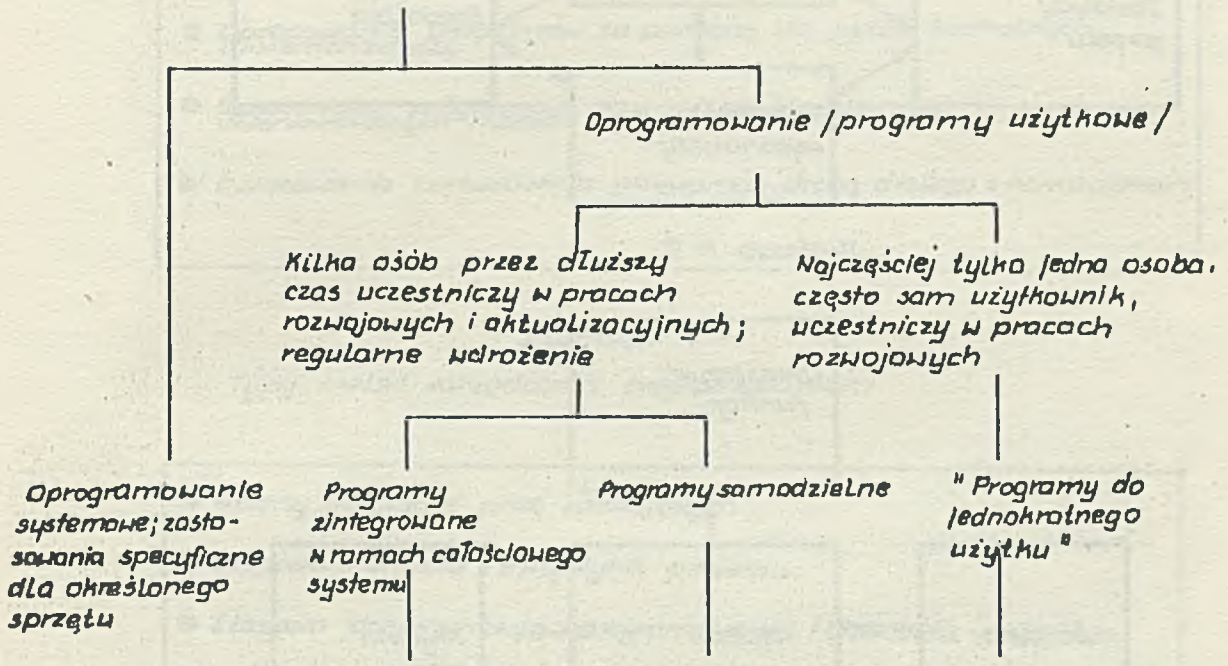
Ilustracja K 5

Rozszczepienie układu celów na typy użytkowników



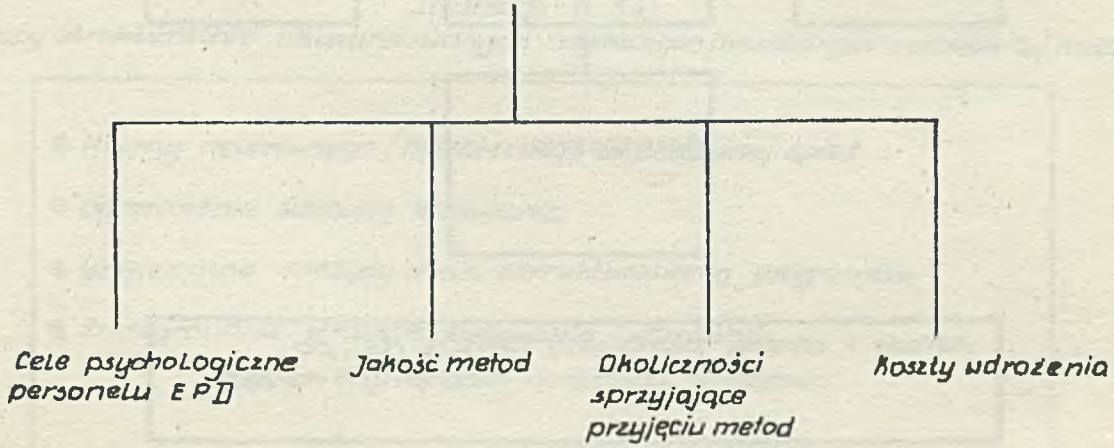
Ilustracja K 6

Typy oprogramowania



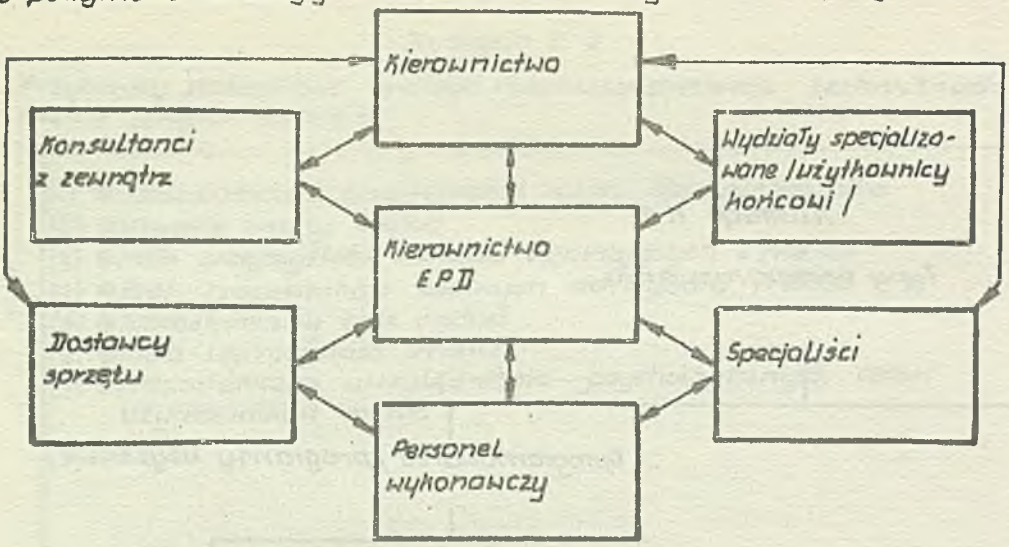
Ilustracja K 9

Czynniki warunkujące gotowość do przyjęcia metod

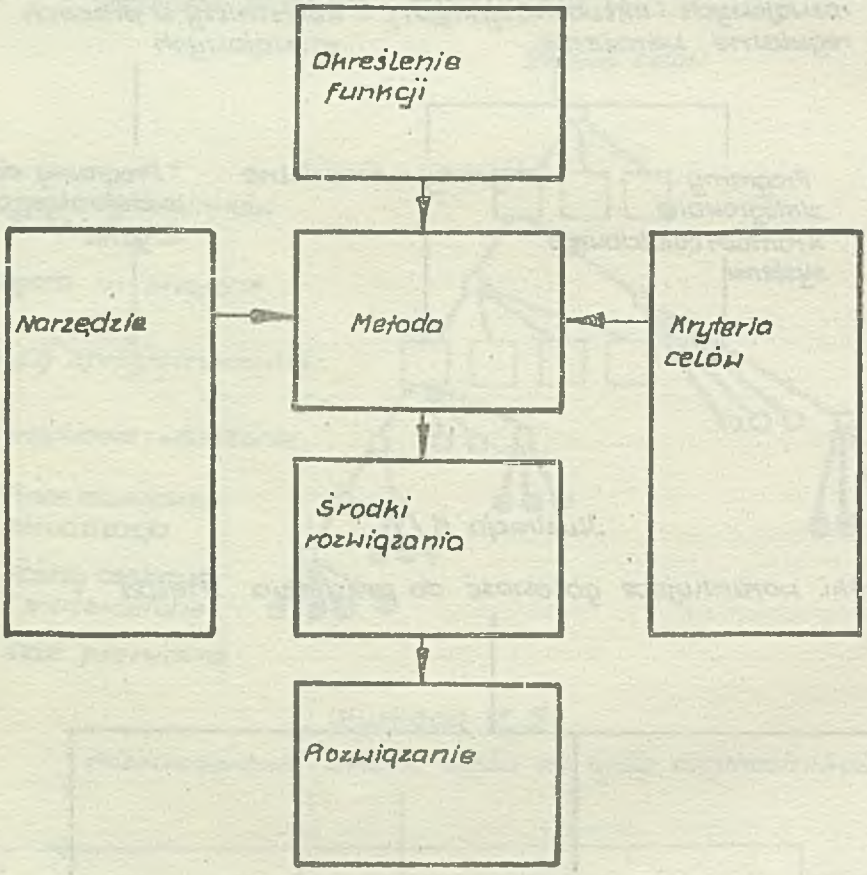


Ilustracja K 8

Proces podejmowania decyzji o wdrożeniu technicznych metod programowania



Ilustracja K 9



Metoda = określa planowaną funkcję i cel, po uprzednim wdrożeniu niezbędnych narzędzi

- Koncepcja
- niezautomatyzowana /nie wspomagana komputerem/ technika operacyjna
- technika operacyjna zautomatyzowana przy użyciu narzędzia software'owego

Jlustracja K 10

Typy metod problemowo ukierunkowanych

- Zbiór i katalog algorytmów
- Opracowanie programów przy użyciu specjalizowanych języków problemowych
- Opracowanie programów za pomocą list pytań kontrolnych i kwestionariuszy
- Opracowanie prpgramów przy użyciu problemowo ukierunkowanych modeli
- Opracowanie i aktualizacja programów drogą dialogu z komputerem

Jlustracja K 11

Typy metod zarządzania programowaniem

- Normy dotyczące prac rozwojowych
- Etapowa kontrola i statystyka projektu
- Etapowa dokumentacja oprogramowania i biblioteka projektu
- Instrukcje zastosowania modułów oprogramowania i informacji
- Organizacja zespołu ukierunkowana na strukturę oprogramowania
- Pomiar wydajności i optymalizacja

Jlustracja K 12

Typy strukturalnie ukierunkowanych neutralnych/niezależnych o problemu/ metod

- Normy neutralnego / niezależnego od problemu/ opisu
- Ograniczenie swobody kodowania
- Uniwersalne metody makrostrukturowania programów
- Zintegrowane strategie testowania i wdrażania

SESJA L

ADAPTACJA UNIWERSALNYCH SYSTEMÓW DO KONKRETNYCH ZASTOSOWAŃ

J.Peers, J.Peers Inc., Stany Zjednoczone

BUTLER: Panie i Panowie, witam Was na ostatniej sesji konferencji, zatytułowanej "Adaptacja uniwersalnych systemów do konkretnych zastosowań". Usłyszymy dziś o metodzie, która wnosi, być może, nową jakość do projektowania systemów - jakość wysokiego stopnia wynalazczości.

Nasz gość i referent tej sesji pracował w kilku przedsiębiorstwach znanych nam wszystkim,m.in. w Unidata.Ale to wszystko podano w "Informacjach wstępnych" i nie będę tego powtarzał. Ostatnio p.Peers udał się jako Europejczyk do Stanów Zjednoczonych, by założyć tam przedsiębiorstwo, rzucając dzięki swej wynalazczości wyzwanie technice amerykańskiej na jej własnym rynku krajowym. Mówiono mi, że wywołał wprost sensację, gdy wystąpił na ostatniej konferencji Amerykańskiego Programu Badawczego Diebolda; i sądzę, zamierza osiągnąć to samo dzisiaj. Oddaję głos Johnowi Peersowi.

PEERS: Dzień dobry. Przede wszystkim pragnę powiedzieć,że sprawiłem duży kłopot Dieboldowi. Zapraszając mnie tutaj, przede wszystkim zapytali: "o czym będzie Pan mówić?", a ja odpowiedziałem: "na ogół nie wiem tego, aż do ranka dnia, w którym mam wystąpić".A na to oni: "nie bardzo to się nadaje do zamieszczenia w "Informacjach Wstępnych" .Toteż nie muszę mówić, że treść mego dzisiejszego referatu będzie znacznie ogólniejsza, niż to podano w "Informacjach Wstępnych"; a winę za to ponoszę ja, nie Diebold.

Chciałbym bardzo, żeby to była prawdziwa dyskusja, a nie dyskusja intelektualna. To co powiem, będzie,jak mi się wydaje, umiarkowanie kontrowersyjne. Aby osiągnąć ten nastrój kontrowersyjności, muszę widzieć, że słuchacze śledzą to co mówię. Dlatego poproszę Was o dawanie wyrazu swojej reakcji przez niezwykłe zachowanie się, jak np. podskakiwanie, zdejmowanie marynarek itp.

Jest coś szczególnego w funkcji ostatniego referenta na konferencji. Musi być dostatecznie interesujący, z pewną dozą oryginalnego zachowania się, by słuchacze zechcieli pozostać na tej ostatniej sesji; z drugiej strony, nie powinien być kimś zbyt ważnym,aby ci, którzy musieli wcześniej wyjechać,nie czuli się pokrzywdzeni.

Pomyślałem sobie, że jest to coś na podobieństwo butelki dobrego wina.Kupiliście butelkę Bordeaux 1961 i wypiliście wino prawie do końca; pozostało jeszcze jakieś pół cala na dnie i niestety ma trochę osadu. I teraz macie dylemat - wypić ten ostatni łyk, bo przecież zapłaciliście za wino; ale może on nie być tak dobry jak pierwszy. Albo zostawić ten ostatni łyk i zapamiętać smak pierwszego. W takiej właśnie sytuacji tego ostatniego łyku ja jestem dzisiaj.

Jak powiedziałem, sądzę, że dyskusja stanie się kontrowersyjna. Przyczyna jest bardzo prosta: moje audytorium zapewne okaże się nierozsądne. Muszę jednak zdefiniować termin "nierozsądny". Nierozsądny jest dla mnie każdy, kto natychmiast nie zgadza się ze mną. Toteż jeśli ostro zareagujecie na to co powiem i poczujecie, że popadacie w stan katatoniczny, na korytarzu jest lekarz; już go uprzedziłem.

A teraz, mówiąc poważnie, nie występuję tu jako ekspert. Nie występuję tu nawet jako zawodowiec, a już na pewno nie jako konsultant. A propos konsultanta,przypomniała mi się definicja, którą zapewne wszyscy znacie. Konsultant to człowiek, który przychodzi do Was i pożycza Wasz zegarek, żeby powiedzieć Wam, która godzina.Pójdę w tej definicji jeszcze dalej i powiem Wam, że nie wiem nic o wielkich problemach naukowych. Nie wiem nic o symulacji ekonomiki. W ogóle lista spraw, o których nic nie

wiem, jest tak przerażająco długa, że naprawdę nie wiem, po co tu jestem.

Ale jest jedna dziedzina,na której znam się coś niecoś, a mianowicie świat interesów i problemy związane z tym, żeby maszyna, którą kupiliście, robiła to po co ją kupiliście. Wygląda to na problem bardzo prosty, ale bynajmniej takim nie jest. Natomiast rzeczą, o której nie zamierzam mówić,jest produkt, który stworzyłem i który moja firma wytwarza i sprzedaje - chyba że mnie do tego zmusicie. O, nie zmuszajcie mnie!

Istnieje w naszej branży podstawowy problem.I tak doniosły, że dopóki go nie rozpoznamy, grozi nam poważna choroba. W tej chwili cała nasza branża cierpi ponieważ to co chcemy robić, kosztuje za dużo, trwa za długo i wymaga zbyt wielu ludzi.

Pomyślcie, że komputery w swej dzisiejszej postaci istnieją mniej więcej od roku 1946, to znaczy od zbudowania maszyn takich jak ENIAC i ILLIAC. Ale metoda, czyli podejście, jakie przyjęliśmy, by spowodować działanie tych maszyn, nie zmieniła się. Wciąż jeszcze uważamy,że trzeba najpierw określić problem, dla którego chcemy znaleźć rozwiązanie; stwarzamy takie rozwiązanie i - mój Boże! - jakoś udaje się nam.Nie potrzebuję mówić, że zwykle potem nie działa to tak, jak byśmy chcieli.

Słyszałem kiedyś historię opowiedzianą mi przez kogoś, kto miał większy dar opowiadania niż ja. Była to historia o pewnym przedsiębiorcy, który po raz pierwszy zobaczył komputer. Wyobraźcie sobie, że jesteście przedsiębiorcą i zobaczyliście ów supernaukowy komputer zbudowany w 1946 r. dla obliczeń balistycznych. Postanawiacie: "ależ tak, to świetnie się nada do robienia moich faktur i list płac". Potem jednak przyjrzeliście się stronie ekonomicznej i powiadacie: "ale maszyna jest za droga. Za dużo kosztuje. Nie opłaci się do wykonywania jednego zadania." Co można było wówczas zrobić? Powiedziano: "dlaczego nie mielibyśmy umieścić na maszynie dwóch zadań? Maszyna jest tak szybka, że na pewno będzie w stanie je rozwiązać." Wtedy producent powiedział: "tak, ale ja nie zaprojektowałem jej do wykonywania dwóch zadań na raz, będziecie więc musieli kupić nieco szybszą drukarkę i zastosować nieco inną formę wejścia i

wyjścia"; ale wtedy znowu rosną koszty. A więc umieszczacie na maszynie trzy zadania.Teraz potrzebujecie znowu trochę szybszej drukarki, trochę wydajniejszego wejścia i trochę większej pamięci; koszt znowu wzrasta.Umieszczacie więc na maszynie cztery zadania. Innymi słowy, rzecz bardzo szybko wymyka się spod kontroli.

A zatem w ciągu 30 lat istnienia komputerów nie było zasadniczo świeżego podejścia do problemu; i o tym właśnie chcę dzisiaj mówić. Trzeba przyznać, że nasza branża ma bardzo złą opinię. Mamy opinię, że dostarczamy nie to co klient zamówił, że bierzemy od niego więcej niż uzgodniono; a gdy produkt zostaje dostarczony,nie spełnia tego,do czego został zakupiony i klient nie otrzymuje go wtedy, kiedy go potrzebuje. Wyobraźcie sobie, że w zwykłym handlu zamówiliście pralkę z dostawą za tydzień; miała Was kosztować 250 dolarów i miała prać bieliznę.Natomiast gdy nadeszła za sześć miesięcy, kosztowała 600 dolarów i zmywała naczynia.

Telefonujecie do dostawcy i mówicie: "przykro mi, ale mam kłopot",a on na to: "och, jaki kłopot?".Powiadacie: "otrzymałem maszynę, która zmywa naczynia". Na to on: "proszę poczekać,zajrzę do kartoteki". Zagląda do kartoteki, a potem mówi: "nie, nie, Pan musiał się pomylić; w kartotece pisze "maszyna do prania".

Co zdziałaliśmy w ciągu tych 30 lat istnienia branży? Sądzę, że można śmiało powiedzieć, iż opracowaliśmy dobry, niezawodny i szybki sprzęt - maszyny i ich wyposażenie. Uważam, że rozwój wyraźnie zmierza ku mniejszemu, szybszemu i tańszemu sprzętowi. Powinienem w tym miejscu zdefiniować moje pojmowanie "niezawodności". Urządzenie mechaniczne jest niezawodne tylko wówczas, gdy zupełnie nie ma ruchomych części!

To wszystko prawda. Szczególnie w dziedzinie półprzewodników dochodzimy dziś do sytuacji, że sprzęt kosztuje mniej niż oprogramowanie. A to jest absurd.

Skąd się to bierze? Jaka jest tego przyczyna? Czy to dlatego, że programiści są całkowicie nieudolni? Czy dlatego, że

nie zostali należycie przeszkoleni? Czy dlatego, że robimy nie to co trzeba? Nie sądzę,by chodziło o którąkolwiek z tych przyczyn. Moim zdaniem przyczyną jest, że uwierzyliśmy w podejście, jakie obraliśmy w początkach branży i nigdy nie zrewidowaliśmy go. Nie spojrzeliśmy za siebie i nie powiedzieliśmy sobie: "popatrzmy świeżym okiem na problem, jak powodować, żeby maszyna działała".

Przed mniej więcej dwoma tygodniami przemawiałem do grupy doświadczonych zawodowych programistów w Kalifornii. Na końcu konferencji jeden z obecnych podszedł do mnie i podał mi wydruk, który mam tutaj ze sobą. Jest to wykaz różnych "praw", takich jak np. "prawo Murphy". Przeczytam Wam kilka z nich. Pierwszym jest "prawo siły Anthony'ego"; brzmi ono: "nie próbuj siłą, weź większy młot". "Prawo Brooke'a": "dodanie pracowników do opóźnionego programu czyni go jeszcze bardziej opóźnionym". Jest i inne prawo - "pewnik Camma": "gdy wszystko inne zawodzi, przeczytaj instrukcje". Jest wreszcie "prawo Colesa": "kapustę trzeba bardzo cienko szatkować".

Z góry przepraszam obecne tu panie, bo to co teraz odczytam, będzie trochę ordynarne. Dalej na liście mam bowiem trzy największe kłamstwa.Pierwsze: "pieniądze to nie wszystko"; drugie: "wspaniale jest być Murzynem"; i trzecie: "wsadzę tylko kawałeczek". Dalsze prawo to "prawo Gamage'a": "poziom wiedzy specjalistycznej zmienia się w stosunku odwrotnym do liczby twierdzeń zrozumiałych dla ogółu". "Prawo wielkich programów Hoare'a": "wewnątrz każdego wielkiego programu siedzi mały program, starający się przebić na zewnątrz". "Prawo przewrotności natury" odkryte przez panią Murphy: "nie da się z góry ustalić, którą stronę chleba posmarować masłem".

Jedno z praw dotyczy programowania komputerów: "każdy program rozrośnie się tak, że wypełni całą rozporządzalną pamięć". Mam nadzieję, że zrozumieliście o co chodzi. Niektóre z tych praw są doskonałe. A przyczyną, dla której wręczono mi ten dokument - możecie mi wierzyć lub nie - było to, że jest w nim także "prawo Peersa", co brzmi imponująco, choć wcale takim nie jest. Ale wprowadza mnie ono do następnego tematu mojego refe-

ratu. To "prawo Peersa" brzmi: "rozwiązanie problemu zmienia charakter tego problemu".

Podam kilka przykładów. Odczuwane przez Was problemy,które skłaniają Was do zakupu samochodu, to już nie te problemy, które macie po zakupieniu samochodu. Czy to prawda? Tak. Problemy, które spowodowały, że ożeniliście się, nie są problemami, jakie macie, gdy już jesteście żonaci.I w tym tkwi sedno zagadnienia. Problemy jakie mieliśmy przed nabyciem komputera nie są problemami, jakie mamy po jego otrzymaniu. Zgoda? Dlaczego więc programujemy wymagania, zanim go otrzymaliśmy? To absurd. Innymi słowy,formułuję coś w rodzaju makrowersji zasady nieoznaczoności Heisenberga: "żadnego problemu nie można z góry określić na tyle szczegółowo, by można go było zaprogramować".

Co więc w takim razie my wszyscy mamy zrobić? No cóż, jest inny sposób podejścia. Podam małą analogię. Weźmy pod uwagę koleje i samochody - choć obecny tu przedstawiciel Kolei Austriackich i niektórzy inni pewnie bardzo się oburzą na to co powiem. Powszechnie uważa się,że pociąg i samochód są bardzo różne. Ale czy naprawdę? Nie, wcale nie są różne, są tym samym. Bo zastanówcie się: oba mają koła, przekładnie, hamulce,miejsca do siedzenia, muszą mieć napęd itd. Jaka jest zatem różnica? Różnica polega na tym, że ktoś kiedyś przyjrzał się zużyciu kół i powiedział: "potrzebny nam jest trudno zużywalny materiał, którego nie będzie trzeba zbyt często wymieniać; idealnym materiałem będzie stal." Był to bardzo logiczny wniosek. Kłopot w tym, że stal nie może biec po ziemi,bo zaryje się w nią. Odpowiedzią na to jest zatem ułożenie arkusza stali wszędzie tam, gdzie zamierzamy puścić pociąg. Tego jednak nie można zrobić, bo byłoby zbyt kosztowne, prawda? Robimy więc oszczędność, kładąc tylko dwa oddzielne pasma stali. Teraz kłopot polega na tym, że między tymi pasmami trzeba utrzymać stałą odległość. Potrzeba więc progów drewnianych, by utrzymać odstęp między stalą. Teraz znów kłopot polega na tym, że musimy kupić teren, na którym układamy te progi i stal, bo ludzie, czyją własnością jest teren nie pozwolą nam położyć ich, gdyż całe to urządzenie jest trwałe. A teraz, kiedy już nabyliśmy teren,kupiliśmy stal i drewno i ułożyliśmy je,nie opłaca się nam puszczać pociągu dla jednej osoby.

Jeśli zaś mamy go puszczać dla 100 osób,dobrze byłoby zbudować parę stacji, żeby ludzie nie czekali w każdym miejscu na trasie. Kłopot w tym, że stacje te są kosztowne,opłaca się więc eksploatować pociąg dopiero dla 1 000 osób i tym samym trzeba zbudować więcej stacji. W końcu rzecz wymyka się zupełnie spod kontroli, tylko dlatego, że ktoś - zresztą bardzo logicznie - doszedł do przekonania, że idealnym materiałem na obręcze kół jest stal, bo jest trwała. To prawda.

Wróćmy jednak do branży, w której wszyscy uważamy się za specjalistów. Przyjrzyjmy się cechom, jakimi powinien odznaczać się programista. Myślę, że zgodzicie się ze mną, że im bardziej logicznie myśli programista, tym lepszym jest programistą. Zgadzacie się z tym? Tak? To dobrze. Im cierpliwiej radzi sobie z powtarzającymi się, nużącymi szczegółami, tym lepszym jest programistą. Zgoda? Dobrze. Bądźcie cierpliwi, nie oponujcie, bo mnie to denerwuje.Im lepiej potrafi radzić sobie z nieustannymi szczegółami, tym więcej pracy wykonuje na dobrej,spójnej bazie.

Gdy się zastanowicie, zobaczycie, że cechy, które przed chwilą opisałem, tzn. logiczność, cierpliwość, powtarzalność, dokładność i wytrwałość, jako charakteryzujące czynność programowania, charakteryzują również komputer. Cechy,jakich wymagamy od człowieka, któremu zlecamy napisanie programu, to właśnie cechy, w jakie już wyposażyliśmy komputer. A więc, gdy spojrzycie na to logicznie, dojdziecie do przekonania, że idealnym narzędziem do pisania programu nie jest istota ludzka, lecz komputer. Twierdzę zatem, że koncepcja pisania programów w języku jest założeniem fałszywym.

Pójdę dalej. Powiem,że program lub to co ewentualnie stworzymy dla zmiany lub zastąpienia programu, to w rzeczywistości zupełnie to samo co dane w komputerze. Nie ma sposobu oddzielenia ich. Koncepcja maszyny przetwarzającej dane jest taka, że to co potocznie nazywamy "programem" i to co potocznie nazywamy "danymi", są tak wzajemnie splecione,że nie można zmienić jednego bez zupełnego przewrócenia do góry nogami drugiego.

Moja propozycja zmierza więc do rewizji całego zagadnienia pisania programu. Wynalazłem sposób spowodowania, by system

kierował sobą samym, a tworzenie programów wewnątrz systemu, które powodują, że system robi to co chcecie, żeby robił, dokonywane jest przez maszynę, a nie przez człowieka. Oczywiście nadal trzeba maszynie powiedzieć, co ma robić, ale nie trzeba jej programować w dzisiejszym znaczeniu tego słowa.

Wynikają z tego pewne bardzo ciekawe wnioski: jeśli macie cierpliwość wysłuchać ich. Wychodzi na to, że choć nie zamierzałem tego, zastąpiłem koncepcję programu koncepcją słownika. Zastępujemy koncepcję stałego programu, który funkcjonuje w oparciu o szereg powiązanych, wzajemnie dla siebie znaczących słów, słownikiem i sekwencją, która ma pewne znaczenie dla maszyny, ale człowiek nie ma możliwości stwierdzić, jaka to jest sekwencja; człowiek posługuje się tylko systemem jako takim.

Skoro zaś tak jest, wynika z tego, że komputerowi nie są już potrzebne standardowe języki takie jak COBOL czy FORTRAN, bo maszyna radzi sobie ze słowami - zwykłymi, naturalnymi słowami; słowami angielskimi, niemieckimi, portugalskimi - jakimikolwiek chcecie. Logika powodująca funkcjonowanie maszyny jest niezależna od sekwencji znaków, jaką stosujecie, aby nadać znaczenie temu, co chcecie, żeby maszyna robiła.

W ten sposób dochodzę do następnej sprawy. Od lat informatyka zachowuje się jakby była Panem Bogiem. Przychodzimy do przedsiębiorstwa, mającego pewien problem; rozmawiamy tam z człowiekiem,który zajmował się tym problemem być może od 12 lat. I mówimy mu: "przyślemy tu jednego z naszych bystrych chłopców na pięciodniowy przegląd systemu i rozwiążemy wszystkie problemy!" Może nie zawsze wygląda to dokładnie tak, ale chcę przez to powiedzieć,że jedyną osobą, która może szczegółowo wiedzieć, co powinno być zrobione, jest sam klient. Proponuję,byśmy zaprzestali uprzedniego programowania wszelkich wytwarzanych i dostarczanych przez nas systemów; byśmy dostarczali sprzęt,który jest tak prosty i tak łatwy do zaprogramowania,by użytkownik mógł to sam zrobić.

Jak doszedłem do tego? Na tę myśl wpadłem przed mniej więcej 2 1/2 laty,a surową próbą wszelkiej słusznej koncepcji jest gotowość zainwestowania w nią własnych pieniędzy. Niestety, mu-

szę Państwu powiedzieć, że zrobiłem to. Założyliśmy w Kalifornii przedsiębiorstwo, które nazywa się Logical Machine Corporation; pierwotnie nazywało się John Peers i Spółka. Produkujemy małe systemy liczące, w których zastosowaliśmy tę koncepcję.

Nie twierdzę, że rozwiązuje to wszelkie istniejące na świecie problemy programowania; tak nie jest. Ale dowodzi to, że istnieją alternatywne metody podejścia do problemu powodowania, by maszyny pracowały. Nasza pierwsza maszyna ma szereg poważnych ograniczeń, z których jednym jest, moim zdaniem, ograniczenie o charakterze metodycznym. Ograniczenie to polega na tym, że dla każdego elementu w systemie, czyli dla każdej jednostki obliczającej potrzebny jest jeden procesor.

Muszę koniecznie powiedzieć Państwu jeszcze o jednej koncepcji. Uważam, że słowo "komputer" do tego stopnia obrosło złym znaczeniem, że należałoby je zastąpić słowem w nowym stylu. Termin, który proponuję dziś jest skrótem słów "logical machine" /maszyna logiczna/. Proponuję więc nazwę LOMAC. O naszej maszynie ADAM nie mówi się jako o komputerze; nazywamy ją LOMAC.

W porównaniu ze skalą produkcji wielkich producentów, wykonaliśmy dotąd tylko drobną ilość sprzętu. Od kiedy wprowadziliśmy maszynę na rynek w kwietniu ubiegłego roku, wyprodukowaliśmy i dostarczyliśmy 35 systemów liczących. Ale, rzecz niewiarygodna, ani jednej maszyny nie wdrożyliśmy. Wszyscy klienci sami wdrożyli swoje maszyny i sami nauczyli je czego od nich chcą oraz sposobu, w jaki chcą, żeby to było robione; a żaden z nich nie był programistą.

Dokąd to nas prowadzi? No cóż, konsekwencje tego wszystkiego są wręcz zastraszające. Siedziba naszej firmy leży zaraz na północ od tak nazywanego w Ameryce Silicon Gulch. Nie jest to jakieś miasto na Dzikim Zachodzie, gdzie wszyscy obsypują się wzajemnie kulami z sześciostrzałowych rewolwerów, ale jest to pas, w którym mieści się największa w świecie produkcja półprzewodników. Znam paru tamtejszych ludzi i jestem przekonany, że to co się tam robi, wpłynie w sposób radykalny na to co będziemy w stanie uczynić w ciągu najbliższego dziesięciolecia.

Wyrazem elektroniki, jaką dziś posługujemy się, jest procesor 16-bitowy o szybkości 200 nanosekund, mieszczący się na jednej płycie 8 x 8 cali. A widziałem już prototypy 16-bitowych mikroprocesorów mieszczących się na jednej pastylce krzemowej i działających z szybkościami cyklu ponad 100 mikrosekund. Przewidywana data masowej produkcji i cena sprzedaży to mniej więcej 2 lata i 30 dolarów.

Skoro tak jest i skoro tak samo dzieje się w dziedzinie pamięci i w dziedzinie różnych metod robienia tego co mamy do zrobienia, nie możemy iść nadal obecną drogą i być nadal branżą o tak wysokim poziomie zatrudnienia.Musimy znaleźć nową metodę. Jeszcze raz powtarzam: bynajmniej nie twierdzę, że nasz sposób jest jedyną drogą do celu; mówię tylko, że jest to przykład świeżego podejścia i chcę jedynie sprowokować Was do zastanowienia się nad możliwością znalezienia świeżego podejścia.

W naszym konkretnym przypadku następny etap rozwoju dotyczyć będzie tego, co w dzisiejszej terminologii nazywa się "przetwarzaniem rozproszonym". Ale każdy element w systemie będzie zawierał swój własny procesor.Toteż wszelkie przeniesienie sekwencji informacji między jakimikolwiek dwoma elementami systemu odbywa się z tą samą łatwością rozumienia tego słownika; a uczenie się potrzebne nowemu elementowi może się odbyć wtedy, gdy jest wymagane.

Nasza branża, branża przetwarzania danych, miała dotąd tendencję do izolowania się od wielu innych branż, ale jestem przekonany, że w najbliższych kilku latach będziemy świadkami gruntownej zmiany. Będziemy świadkami łączenia się i stapiania obecnego przetwarzania danych z przetwarzaniem tekstów, transmisją faksymili, telekomunikacją i przetwarzaniem rozproszonym. Wszystko to zleje się w jedną dyscyplinę, a podejście do niej nie będzie mogło polegać na mówieniu o potrzebie lepszych sposobów kontroli norm tego czy innego języka programowania.

Nasze możliwości są ograniczone; jesteśmy tylko małym przedsiębiorstwem i nie zrobiliśmy wszystkiego, co chcielibyśmy zrobić. Przypominacie sobie "prawo Peersa"; otóż i nam zdarzyło się, że gdy rozwiązaliśmy pierwszy problem,natrafiliśmy na dwa-

dzieścia innych; ale zaczynamy przezwyciężać i te problemy.Wierzymy, że w ciągu bardzo krótkiego czasu uda się nam wyprodukować konwersacyjny,nie programowany w konwencjonalnym tego słowa znaczeniu, system liczący o całkowicie swobodnej formie;system, który w procesie uczenia się tego, czego od niego wymagacie, po prostu prowadzić będzie z Wami dialog dla ustalenia znaczenia tego, czego pragniecie, dokładnie w taki sam sposób, w jaki dzisiejszy system rezerwacji pyta Was o dane. Maszyna będzie zapamiętywać znaczenie słów, których zechcecie użyć, by spowodować jej działanie.

Jaki będzie następny krok po tym? Wiecie chyba wszyscy, że mniej więcej przed dwoma miesięcami nadeszła z Massachusetts wspaniała wiadomość o człowieku, który w ramach programu badawczego dotowanego przez Instytut dla Niewidomych wynalazł maszynę, która czyta. Wkładacie do góry nogami książkę do maszyny; ta przeszukuje strony książki i czyta ją Wam, wiersz po wierszu z szybkością około 200 słów na minutę. Gdy natrafia na słowo, którego nie zna, przeliterowuje je. Słyszeliście też prawdopodobnie,że w Kalifornii, również w ramach programu finansowanego przez Instytut dla Niewidomych, ktoś inny wynalazł mówiący kalkulator. Zamiast wydawania cyfr świetlnych, maszyna mówi Wam: "wynik brzmi ..." Skoro coś takiego istnieje dzisiaj,a istnieje już, musimy oczywiście bardzo starannie przemyśleć, dokąd zmierza nasza branża. Jestem przekonany, że następnym etapem będzie obliczanie w pełni audio-dialogowe, to znaczy, że będziemy mówić do systemu liczącego, a on będzie odpowiadał słyszalnymi wyrażeniami. Jesteśmy tego znacznie bliżsi niż się komukolwiek wydaje.

My - nasza uboga, mała firma w Kalifornii - opracowaliśmy system, który jest zdolny do posługiwania się koncepcją słownikową. Mamy już 35 dostarczonych i czynnych systemów liczących oraz portfel zamówień na około 400 dalszych. Nie sądzimy, by krzyżowe zdefiniowanie słów w słowniku akustycznym przedstawiało większe trudności niż zdefiniowanie ich w naszym słowniku cyfrowym.

Pomyślcie tylko, jakie to będzie miało skutki. Na początek zaoszczędzimy miliony drzew. Dlaczego to mówię? No bo pomyślcie

choćby o książce telefonicznej.Telefoniczny spis abonentów jest chyba - w sensie procentu wykorzystania - najbardziej nieefektywną rzeczą w historii ludzkości, jeśli porównać to co otrzymujemy z tym, czego rzeczywiście potrzebujemy. Dają nam książkę z około dwoma milionami numerów, a używamy tylko jedenastu. Tak jest.

Dlaczego musimy produkować książkę telefoniczną? Dlatego, że nie mamy maszyny, która rzeczywiście potrafi zrozumieć słowo mówione. Gdybyśmy taką maszynę mieli, moglibyście podnieść słuchawkę i powiedzieć: "chcę rozmawiać z panem Schmidtem, zamieszkałym przy ulicy..." i zostaniecie z nim połączeni, nawet jeśli nie będzie pod swoim stałym adresem, jeśli tylko podał maszynie, gdzie będzie. A zatem żadnych książek telefonicznych; oszczędzone zostaną te wszystkie drzewa, o których wspomniałem. To jest coś dla obecnych tu zwolenników ochrony przyrody.

Spójrzmy w przyszłość. Dokąd zmierzamy? Zakładając, że to co powiedziałem jest prawdziwe - a każdego w moim położeniu powinniście zawsze uważać za zawodowego kłamcę - przyszłość należy zdecydowanie do komputerów domowych. Mogę przewidzieć czas, może już za 6 lub 7 lat, gdy każde gospodarstwo domowe na świecie będzie mogło mieć swój własny komputer; będzie mogło zapisywać programy telewizyjne; lub - powracając do referatu Jima Martina z pierwszego dnia naszej konferencji - mieć własną małą antenę na dachu, aby otrzymywać informację, której potrzebuje, wtedy gdy jej potrzebuje i w takiej postaci, w jakiej potrzebuje. A "niewidzialność" systemów polegać będzie na tym, że wystarczy po prostu powiedzieć, czego się chce.Będzie to mieć dla ludzkości równie dalekosiężne skutki co wynalazek druku.

Można będzie zatelefonować z biura do swego komputera domowego i powiedzieć mu: "nie zdążę do domu na czas. Proszę mi zapisać audycję telewizyjną o ósmej." Albo przemawia polityk i powiada: "to straszne, że 17% ludzi robi to a to", a Wy mówicie swojemu komputerowi: "nie wierzę w to". Wasz komputer sprawdza przez chwilę,po czym łączy się ponownie z Wami i mówi: "on plecie bzdury; to tylko 1,4%".

Przykłady, które wybrałem, są może błahe. A oto inny taki przykład. Macie swój domowy komputer połączony z wagą w łazience; rano stajecie na wadze, a komputer powiada Wam: "nie powinieneś był jeść tego makaronu wczoraj na kolację". Skutki są niebywałe.

Wszystko co powiedziałem, można podsumować następująco. Po pierwsze, uważam, że podejście polegające na uprzednim określeniu problemu i napisaniu oprogramowania,nie jest jedynym słusznym podejściem. Myślę, że czas potrzebny na dokonanię modyfikacji związanych z poprzednią zmianą, staje się już dłuższy niż czas jaki upływa między tą zmianą a następną. Już dziś można powiedzieć, że nastąpi dziesięciokrotny, a może i stokrotny wzrost liczby użytkowanych na świecie komputerów.A nie ma - mam nadzieję! - sposobu,żeby stokrotnie zwiększyć liczbę programów.

Sądzę, że jedynym sposobem rozwiązania problemu, w którego obliczu wszyscy stoimy - wysoki koszt, opóźnienia, fakt, że otrzymujemy nie to czego chcemy i że nie dajemy użytkownikowi tego czego sobie życzy - jest oddanie problemu z powrotem w ręce użytkownika, to znaczy dostarczenie mu komputera, którym sam będzie kierować. Musimy - jeśli wolno mi posłużyć się analogią - zaprzestać dostarczania całych systemów transportowych, a sprzedawać samochody. Ile samochodów mogliby sprzedawać wielcy producenci, gdyby do każdego sprzedanego samochodu musieli mieć mechanika, dwóch kierowców i człowieka z czerwoną chorągiewką, który by biegł przed samochodem? Chyba niewiele.Dziękuję bardzo za uwagę.

ZIJLKER /AKZO/: Jak mógł Pan oczekiwać,nie potrafiłem powstrzymać się, żeby nie podskoczyć podczas tego niezwykle prowokującego do myślenia referatu. Moją pierwszą reakcją było: "to fantazja";drugą moją reakcją było: "to zupełna fantazja!" Wyraził Pan przekonanie, że jedynym sposobem ostatecznego rozwiązania problemu jest oddanie komputera w ręce użytkownika, by mógł on samodzielnie użytkować go. W innym miejscu powiedział Pan, że żaden problem nie da się z góry określić tak, aby można było stworzyć program, który go rozwiąże. Jeśli jednak oddamy komputer w ręce użytkownika,będzie on musiał zawierać wszystkie ele-

menty niezbędne do rozwiązania problemu; ale ta ilość elementów jest tak fantastycznie duża, że nie sądzę, by dało się to kiedykolwiek zrobić. Choć myślę, że są pewne przypadki, w których ten typ urządzenia logicznego, czyli LOMAC będzie mógł znaleźć zastosowanie, sądzę, że nie rozwiązuje on całkowicie problemu. Korci mnie, żeby zapytać, ile Pan ma lat, bo jeśli czytał Pan Wittgensteina, wie Pan, że zanim doszedł do 40 lat, wierzył, że każde słowo jest odzwierciedleniem jakiegoś elementu rzeczywistości. Była to koncepcja statyczna. Myślę,że gra tu dużą rolę rozwój mentalności z wiekiem, że ludzie w wieku od 20 do 40 lat wierzą w możliwość rozwiązania problemów. Po czterdziestce to się zmienia, może mi Pan wierzyć! Wittgenstein także przekonał się o tym,bo wtedy napisał swoją drugą słynną książkę i stwierdził, że używanie słów odbywa się w ramach gry językowej, że znaczenie słów zależy od ich zastosowania i że to znaczenie zmienia się także z czasem. Mamy więc zestawy słów,które są zupełnie przydatne np. w tenisie lub czymś podobnym. Być może zestaw słów dla tenisa dałby się zautomatyzować w sposób, o jakim Pan mówił, ale to niepotrzebne; znacznie przyjemniej jest widzieć sędziego siedzącego na wysokiej drabince. Gdyby rzeczywiście miał Pan w tym swoim komputerze tyle elementów, by uczynić wszelką konwersację zupełnie niepotrzebną,bo rozporządzałby Pan wszystkimi elementami, wtedy to co Pan mówi mogłoby być prawdziwe. Ja jednak sądzę, że jest to z zasady nieosiągalne, choćby nie wiedzieć jak pięknie brzmiało.

PEERS: Moją pierwszą odpowiedzią jest, że musi Pan przyjechać do Kalifornii i zobaczyć. Dłuższej odpowiedzi wymaga Pańskie twierdzenie,że aby zapewnić rozwiązanie problemu,potrzebny jest system zawierający liczne elementy. Nie uważam tego twierdzenia za słuszne.

Pozwolę sobie powtórzyć swoją tezę. Jedyny,który wie, czego chce użytkownik, jest on sam. Ale z "prawa Peersa" wynika, że w chwili, gdy zaczyna uczyć się jak rozwiązać to co chce zrobić, zmienia się to czego w swoim przekonaniu chciał. Jedynym sposobem dotrzymania kroku tym zmianom, jest dać użytkownikowi narzędzie, za pomocą którego sam będzie znajdował rozwią-

zanie.My nie potrafimy tego zrobić za niego;nie możemy go przez cały czas prowadzić za rękę.

Gdy mamy system, który można nauczyć słownika,a także zmodyfikować to czego go wyuczono lub oduczyć go tego równie szybko jak go nauczono, wówczas - przynajmniej tymczasowo, dopóki frenetyczne tempo życia znowu nie przyśpieszy się - rozwiązaliśmy częściowo problem. Nie twierdzę, że jesteśmy przez to w stanie spełnić wszystkie wymagania wszystkich; coś takiego w ogóle nie jest możliwe. Musimy założyć, że klient jest trochę inteligentny.

Odpowiadając więc na Pańskie pytanie, powiem, że wcale nie trzeba wielu elementów, bo potrzeba tylko i wyłącznie tego,czego dana osoba chce. Nie ma pakietu; nie ma niczego zbędnego; tylko to, czego użytkownik chce.

BUTLER: Czy byłoby trafną analogią, jeśliby powiedzieć,że przecież Szekspir wcale niemało zrobił za pomocą tylko 26 liter?

CHEVALIER /A.M.Berliet/: Chciałbym nawiązać do wcześniej postawionego pytania. Powracając do analogii z telefonami, musimy pamiętać, że miliony adresów w spisie abonentów telefonicznych, czyli adresy milionów ludzi,z którymi mogą ewentualnie zechcieć rozmawiać abonenci telefonu, te miliony adresów i numerów muszą być gdzieś zapisane. Musi przy tym istnieć jakiś rodzaj formatyzowania - posługując się terminologią EPD - aby wszystko było przedstawione komputerowi czy też maszynie logicznej w ujednolicony sposób dla celów wyszukiwania. Wymaga to bardzo dużej pojemności pamięci. Podobnie jak p.Peers zawsze jestem przerażony, gdy widzę te ogromne wydawane słowniki, w których rzadko szuka się czegoś. Gdyby ich jednak nie publikowano, potrzebne byłyby bardzo duże pamięci,a także jakieś reguły tworzenia tych pamięci. Reguł tych musiałby przestrzegać użytkownik i nie wolno byłoby mu ich naruszać. Musiałby się pogodzić z pewną dozą dyscypliny.

PEERS: Wychodzi Pan z klasycznego założenia,którego ja nie uważam za słuszne. Nawiasem mówiąc padło tu jedno pytanie, którego nie chciałbym pominąć. Mam poniżej 40 lat. Ale wracając do Pań-

skiego pytania: czy lepiej jest mieć jedną pamięć zawierającą miliard numerów telefonicznych niż miliard książek telefonicznych, z których każda zawiera ich milion? Mówimy dziś o przetwarzaniu rozproszonym w ramach przedsiębiorstwa, ale to samo odnosi się do systemu telefonicznego.Możliwe jest proste marszrutowanie przez maszynę, ale rzeczą kluczową, o której może nie dość jasno powiedziałem, jest to, że niewątpliwie będą pewne reguły,według których maszyna będzie pracować; powinny one jednak być organizowane przez samą maszynę, tak aby forma naszego posługiwania się nią była zupełnie swobodna. Za każdym razem, gdy wykroczy Pan poza to, co maszyna potrafi zrozumieć, powinna ona przerwać dialog z Panem i wyjaśnić to nieporozumienie, dokładnie tak samo jak dzisiejsza baza danych wyszukuje informację, a gdy zachodzi jakakolwiek wątpliwość, powiada: "czy to ma być to, czy to, czy to?" Moją zasadniczą tezą jest, że komputer potrafi manipulować sekwencją równie łatwo jak danymi,a manipulacja sekwencją to właśnie to, co obecnie nazywamy programowaniem. Weźmy analogię z telefonem, Sądzę, że jest rzeczą zupełnie prostą mieć w centrali telefonicznej system utrzymujący w pamięci nazwiska i adresy abonentów przyłączonych do tej centrali. Używając swego telefonu w tym systemie akustycznym, możecie po prostu powiedzieć - tak jak to robicie teraz, z tym tylko, że nie będzie telefonistki w centrali - "chciałbym połączyć się z panem Smithem zamieszkałym w Nowym Jorku przy ulicy...." Nie będzie różnicy między sortowaniem tej sekwencji przez maszynę dla ustalenia dokąd ma nastąpić marszrutowanie, a dzisiejszym sortowaniem jej przez człowieka, z tym tylko, że sortowanie maszynowe będzie bardziej niezawodne,tańsze i szybsze.

BUTLER: Dziękuję Ci John. Może mógłbym krótko nawiązać do czegoś, co zdarzyło się przed kilku laty i potwierdza to o czym tu mówiono. Mój przyjaciel, którego zapewne niektórzy z obecnych tu znają, Kit Grindley, opracował język nazwany SEMATICS, pomyślany jako język projektantów systemów do definiowania problemów. Przeprowadziliśmy na Uniwersytecie Manchesterskim w Wielkiej Brytanii ciekawą próbę zdefiniowania wszystkich powiązań w słowniku danych w sensie określenia,co jest pochodną cze-

go i co z tego wychodzi. Zbudowaliśmy słownik danych, a potem bez jakiegokolwiek elementu proceduralnego wprowadziliśmy całość do maszyny, aby zbadać, czy potrafi ona to po prostu wysortować w pewną logiczną proceduralną sekwencję. I o dziwo, potrafiła.

PEERS: Odchodzimy nieco od tematu, ale wiemy już w ogólnych zarysach, że można wprowadzić do systemu wyniki i format obliczenia,a ten potrafi określić na podstawie przedstawionych mu powtarzalnych przykładów tego samego obliczenia, jakie to było obliczenie i jakie jest powiązanie między tymi ciągami liczb. Wszystko mi jedno, jak to nazwiecie - uczeniem się, programowaniem,podprogramami standardowymi,czy też logiką pochodną - chcę tylko powiedzieć Wam,że jeśli nie będziemy dynamiczni w podejściu do problemu, jak spowodować, by maszyna logiczna działała - a mam na myśli nie tylko komputery, ale wszelkie urządzenia logiczne - prześcigną nas inni, jakaś inna branża, ktoś nowy kto się pojawi na rynku.

Klasycznym przykładem są ci wszyscy, którzy produkowali maszyny do powielania na mokro. Gdy mówiono im,że można przejść bezpośrednio od zwykłego papieru do suchej odbitki, odpowiadali: "bzdura! A jeśli nawet nie bzdura, będzie to zbyt drogie. I w ogóle, komu to potrzebne?" Myślę, że wolałbym mieć dziś akcje Xeroxa niż akcje którejkolwiek z fabryk maszyn do powielania na mokro, bo to co Xerox sprzedaje, to właśnie to o czym dziś Wam mówię. Nie chodzi o to, że Xerox sprzedaje suche kopiarki, ale o to, że zaoferował gospodarce lepsze rozwiązanie rzeczywiście istniejącego problemu i umożliwił wykonywanie tej pracy przez mniej wykwalifikowanych ludzi. Zaoferujcie coś takiego gospodarce, a uchwyci się tego oburącz. Twierdzę, że coś takiego jest bliskie w naszym przypadku. Zrobiliśmy jeden malutki krok, ale ten malutki krok dowiódł, że istnieją nowe drogi. Po prostu chcę Wam to uświadomić. Mamy nadzieję, że nasza firma będzie miała powodzenie,ale nie to jest w tej chwili najważniejsze; chodzi mi o to, że można sobie wielce uczenie rozważać, jaki nadać format normom COBOLu, jak zorganizować powiązania programów i jak zapewnić przenośność. Można pracować nad

tym aż do siódmych potów, ale gdy pojawi się na rynku maszyna, która sama to zrobi,nikt nie będzie Was słuchał. Tak to już jest.

Ludzki umysł jest śmieszny. Ilekroć uczymy go czegoś nowego, czegoś lepszego, czegoś co jest dla niego łatwiejsze w użyciu, przestaje wierzyć, że metoda, którą stosował dotąd, jest tak dobra, jak przedtem sądził; nowa metoda staje się normą, a dawna staje się przestarzała.

WALIGÓRSKI /KBI/: Należy Pan do rosnącej grupy ludzi, którzy powiadają: "nie bądźcie tak zapatrzeni w sprzęt, komputery lub programowanie, jak to ma miejsce obecnie". Ale zupełnie jasne jest, że słownik to jeszcze nie wszystko. Ci, którzy próbowali rozwiązać ten problem w sposób podobny do tego, jaki Pan przedstawił,zmuszeni byli - wskutek semantyki nieuchronnie związanej z procesem - żeglować gdzieś między zupełnym brakiem precyzji języka, a bardzo trudnym przetwarzaniem danych. Muszę powiedzieć, że ci, którym nie powiodło się, powrócili do programowania w FORTRANie lub - co gorsza - do systemów, które prowadzą użytkownika za rękę w każdym szczególe. Jakie jest Pana zdaniem dobre rozwiązanie tego problemu?

PEERS: Wiąże się to z tak wieloma tematami, że zupełnie nie wiem, od czego zacząć. Poruszę jeden.Zakłada Pan w swoim wystąpieniu, że ważna jest wydajność maszyny. Wiem,że nie powiedział Pan tego wprost, ale należy to tak rozumieć. Od kiedy pojawił się szybki minikomputer, nie jest to już istotne. Może nastąpić znaczne pogorszenie wydajności maszyny, zanim w ogóle zauważy Pan to w ludzkiej skali,Jeśli ma Pan oddzielną maszynę dla każdego elementu w systemie, dla każdego użytkownika w systemie, może Pan w konwencjonalnym sensie, znieść silne pogorszenie sprawności systemu bez jakiejkolwiek widocznej dla człowieka straty wydajności.

Jeśli chodzi o użytkowanie systemu,mogę jedynie powiedzieć, że gdy przekazujemy problem w ręce człowieka, którego ten problem dotyczy, wówczas do niego należy wybór, jak daleko idące i jak skomplikowane rozwiązanie zaproponuje dla siebie. Wszyscy

wiemy, co się dzieje, gdy sprzedajemy kompletny pakiet programowy za 20 000 dolarów. Kupujący chce wszystko, bo przecież zapłacił 20 000 dolarów. Nawet jeśli sporządza tylko trzy faktury eksportowe na miesiąc, chce żeby i to było zaprogramowane. Oddajcie problem w jego ręce, a sam będzie decydować,co chce zrobić i kiedy ma to zrobić.

Obecna technika użytkowania maszyny... Zupełnie nie widzę, dlaczego wprowadzenie użytkownika w szczegóły potrzebne, żeby spowodować funkcjonowanie maszyny, ma być czymś gorszym niż wprowadzenie programisty, który nie zna przedsiębiorstwa, w szczegóły tego, co użytkownik chce, żeby było zrobione.Ponieważ z natury rzeczy, ilekroć wprowadzamy w łańcuch kogoś nowego, dajemy więcej okazji do nieporozumień, fakt, że użytkownik robi wszystko sam, powinien dać szybszy efekt, nawet jeśli nie pracuje on szybciej. Mam nadzieję, że odpowiedziałem na Pańskie pytanie.

AKOS /Politechnika w Delft/: Pańskim punktem wyjścia jest,panie Peers, że użytkownik jest rozumnym człowiekiem. Zgadzam się z Panem, ale nasze doświadczenie jest nieco inne. Gdy mówię "nasze", włączam w to p.Phillipa Dorna,który wystąpił na Konferencji Wyższego Kierownictwa we Frankfurcie. Z jego wieloletniego doświadczenia wynika, że użytkownik nie wie czego rzeczywiście chce. Według mojego doświadczenia, problem polega na tym, że nie wie dokładnie czego chce.Chciałbym przypomnieć Panu Pańskie własne "prawo", że rozwiązanie problemu zmienia problem. Nawet gdy użytkownik jest bardzo rozumnym człowiekiem, zgodnie z Pańskim "prawem" rozwiązanie zmieni problem. Widzimy to każdego dnia.

Poza tym obawiam się, że nie docenia Pan znaczenia dialogu i współpracy między użytkownikiem a informatykiem.Z mojego doświadczenia wynika, że rozwiązanie problemu jest procesem iteracyjnym w czasie. Moim zdaniem, problemy są czymś żywym,co zmienia się każdego dnia. Nie sądzę więc, by najlepszy i najszybszy komputer na świecie był w stanie rozwiązać problem tak jak Pan tego chce.

PEERS: Widocznie nie dość dobrze użyłem niektórych słów. Pozwólcie mi sprecyzować. Przede wszystkim mogę chyba powiedzieć, że skoro mówi Pan, że klient nie wie, co chce zrobić, to gdy sprzedajemy takiemu człowiekowi komputer, sami ściągamy sobie kłopot na głowę. Skoro nie wie on, co chce zrobić,to przy obecnej metodzie musi powstać z tego kompletny chaos.

Nie twierdziłem, że dialogowy proces uczenia się jest jednoznaczny z prostą maszyną klienta. Natomiast stwierdzam, że prawie każdy programista chce rzucić ten zawód i być projektantem systemów lub kierownikiem. Doświadczenie, jakiego nabyli ci ludzie przy organizowaniu rozwiązań problemów, jest wyraźnie wielce poszukiwane na całym świecie, bo problemów jest więcej niż rozwiązań, a umiejętności, szczególnie umiejętności poparte doświadczeniem, są towarem wielce deficytowym. Twierdzę więc, że problemy, z jakimi mamy do czynienia - kompilacja, tłumaczenie, uruchamianie, poprawki, modyfikacje i wszystko inne,z czym musimy się zmagać - hamują postęp, którego chce zarówno użytkownik jak i my jako branża.

Powiadam więc, że będzie istniał dialog, ale dialog ten będzie wolny od mechaniki wpływającej na rozwiązanie, nie będzie natomiast wolny od procesu uczenia się. Ale ujemną stroną procesu uczenia się przy dzisiejszej technice jest to, że jeśli nie zaczniecie go już od samego początku i nie obejmiecie nim od razu wszystkiego, co może się zdarzyć, to gdy zdarzy się coś takiego, będzie Was drogo kosztować.Jeśli uwolnicie się od tego podstawowego problemu, jeśli nigdy nie tworzycie z góry programu, stwarzacie po prostu dynamiczny strumień, znaczącą sekwencję, która sprawia pojawienie się bieżącego rozwiązania; sekwencję tę można w każdej chwili zmienić i równocześnie wzrasta zdolność uczenia się, wzrasta chęć uczenia się, a spada koszt i skraca się czas. I to uważam za główne cele.

AKOS /Politechnika w Delft/: Chcę powiedzieć, że zgadzam się z Panem, gdy mówi Pan, że te przyszłe maszyny będą dobrą pomocą dla przyśpieszenia i ułatwienia tego iteracyjnego procesu.Jeśli o to Panu chodziło, zgadzam się z Panem.

PEERS: Tak.

BUTLER: Oczywiście dyskusję tę można by jeszcze kontynuować i niewątpliwie będzie ona kontynuowana dla tych,którzy będą mogli pozostać na sesji poświęconej spotkaniu z referentami. Dziękuję Johnowi Peersowi za jego podniecające, kontrowersyjne i prowokujące wystąpienie. Gdy przybył tu, przyrzekł mi, że choć może nie spodoba się Wam,to co powie, na pewno nieprędko o tym zapomnicie. Myślę, że dotrzymał przyrzeczenia.Dziękuję Ci bardzo, John.

WOSI "Wspólna Sprawa" zleo.155/77, nakł.1002egz.

Cena zł 92.-

WOSI »Wspólna Sprawa«III, zlec. 478 /76, nakł. 10000