ZARZĄD XIV SZTABU GENERALNEGO WP
WOJSKOWY INSTYTUT INFORMATYKI

XXV-LECIE INFORMATYKI WOJSKOWEJ STAN I PERSPEKTYWY ROZWOJU

(MATERIAŁY Z KONFERENCJI)

RYNIA

LISTOPAD 1986

ZARZĄD XIV SZTABU GENERALNEGO WP
WOJSKOWY INSTYTUT INFORMATYKI

XXV-LECIE INFORMATYKI WOJSKOWEJ STAN I PERSPEKTYWY ROZWOJU

(MATERIAŁY Z KONFERENCJI)

RYNIA

LISTOPAD 1986

CZ. I REFERATY WYGŁASZANE

prof.dr hab. Władysław M.TURSKI

Tendencje rozwoju metod i środków informatyki użytkowej

Wielu z nas zastanawia się, jak też będzie wyglądać informatyka w końcu bieżącego stulecia.

Niektórzy, ulegając pokusie lub namowom, formułują prognozy. Mocno ryzykowna to zabawa, zwłaszcza wtedy, gdy przedmiotem prognozy są dane, mające obrazować liczebność zainstalowanych komputerów, ich parametry techniczne, czy też owo sławetne 'zapotrzebowanie na kadry'. Aby przekonać się o tym, jak bardzo ryzykowna, wystarczy cofnąć się myślami o 15 lat. Kto mógł wówczas przewidzieć wystąpienie zjawiska masowej popkultury komputerowej, pojawienie się dziesiątków milionów komputerów osobistych, jednokostkowych równoważników procesorów maszyn IBM 370, megabajtowych pamięci SRAM na półcalowej kostce, pięćdziesięciomegabajtowych dysków mieszczących się - razem z napędem - w pudełku o gabarytach mniejszych od warszawskiej książki telefonicznej? Kto mógł przewidzieć, że w Polsce 1986 roku więcej komputerów będzie w rękach prywatnych niż w instytucjach i przedsiębiorstwach, że przy malejącym zatrudnieniu w dziale 'informatyka', będzie rosnąć - sięgając setek tysięcy! - liczba osób (głównie w wieku przedprodukcyjnym) łapczywie korzystających z każdej nadarzającej się szansy

zdobycia umiejętności obsługi komputera? Kto zdobędzie się na odwagę wypowiedzenia przypuszczenia, że następne piętnaście lat nie przyniesie równie zaskakujących zmian, że nie pojawią się równie rewolucyjnie nowe rozwiązania techniczne, że nie nastąpią równie głębokie zmiany technologiczne, że w podobnej skali nie zmienią się parametry społeczno-ekonomiczne?

W obliczu zmian morfogenetycznych jakie występują w całej sprzętowej sferze informatyki, jakiekolwiek prognozowanie na okresy dłuższe niż czas życia kolejnej generacji układów jest bezsensowne - jeśli za przedmiot zainteresowania bierze komputery. Dokładnie tak samo, jak w 1970 roku nie można było przewidzieć dzisiejszej sytuacji sprzętowej, tak samo dziś nie można przewidzieć, jak będzie wyglądać obraz sprzętu komputerowego w roku 2000. *Jedyne, co można powiedzieć, to to, że będzie on zupełnie inny - prawie na pewno i prawie pod każdym względem - niż ten, który widzimy dziś.*

Gdybym uważał, że informatyka zajmuje się wykorzystywaniem komputerów, powinienem by zakończyć mój referat w tym miejscu. Ale, przyjmując za twórcami pojęcia, że informatyka zajmuje się przetwarzaniem informacji przy użyciu środków technicznych, mogę pokusić się o sformułowanie poglądu co do tego, w jakim kierunku pójdzie jej rozwój.

Wyjaśnijmy sobie najpierw, na czym polega różnica między tymi dwoma poglądami na istotę informatyki.

Pogląd pierwszy przyjmuje za pierwotny *proces tworzenia* nowych urządzeń technicznych. Kwestia zastosowań tych urządzeń

jest uważana za wtórną. Dla prawdziwego wynalazcy jest to zapewne jedyny możliwy do przyjęcia punkt widzenia: im oryginalniejsze tworzy urządzenie, tym mniej wiadomo o jego rzeczywistych walorach użytkowych. Koncentrując się na osiągnięciu głównych zamierzeń, wynalazca musi pomijać ogromnie wiele cech z przyjętego punktu widzenia drugorzędnych i często niemożliwych do ustalenia zanim urządzenie nie zostanie zbudowane i skonfrontowane z realiami otoczenia, w którym ma funkcjonować i z którym ma współdziałać. Nadmierne przejmowanie się takimi cechami w trakcie tworzenia nowego urządzenia stwarzałoby zbyt wiele hamulców, opóźniało, lub zgoła uniemożliwiało dokonanie aktu twórczego.

Jeśli jednak pogląd taki wyznaje producent urządzenia, sytuacja komplikuje się niepomiernie. Albo bowiem producent, inwestując w podjęcie produkcji urządzenia o niesprawdzonych walorach użytkowych, ponosi ogromne ryzyko ekonomiczne, albo też - jeśli chce się uchronić przed takim ryzykiem - musi wybrać jedną z dwu dróg 'bezpiecznych'.

Pierwsza z nich polega na uruchamianiu produkcji małymi krokami: najpierw modele, potem krótkie serie próbne i wreszcie rozwinięcie produkcji w pełnej skali. Ta droga najeżona jest jednak wieloma trudnościami technicznymi, wypływającymi z konieczności adaptacji linii produkcyjnych do zmiennej skali wytwarzania, a ponadto prawie nigdy nie pozwala w pełni zrealizować potencjalnego zysku, jaki może przynieść szybkie uruchomienie produkcji udanego wyrobu.

Druga droga 'bezpieczna' polega na zapewnieniu sobie pozycji monopolisty: nie dopuszczając do tego by potencjalni użytkownicy produkowanych urządzeń mieli możliwość wyboru między konkurencyjnymi rozwiązaniami, zmusza ich do dokonania wyboru między używaniem jedynego dostępnego urządzenia danej klasy a rezygnacją z ich stosowania. Eliminując ryzyko producenta i zapewniając mu optymalne warunki ekonomiczne, droga monopolistyczna przekształca wszystkie niedoskonałości procesu produkcji urządzeń w dodatkowe niewygody (a czasem straty) użytkownika. Ponadto, jeszcze bardziej niż druga stopniowego uruchamiania produkcji, droga monopolistyczna hamuje postęp, gdyż producent-monopolista nie doznaje ekonomicznego przymusu innowacji.

Pogląd drugi przyjmuje za pierwotne *użytkowanie urządzeń*, zakładając, że potencjalne zyski z używania urządzeń stwarzają dostateczne społeczno-ekonomiczne bodźce uruchomienia produkcji urządzeń najlepiej odpowiadających potrzebom użytkowników. Jest to punkt widzenia użytkownika.

Przyjmując ten punkt widzenia zakłada się, że ekonomiczny potencjał użytkowników przeważa potencjał producentów, a dokładniej, że użytkownicy są w stanie uruchomić dostatecznie silny nacisk ekonomiczny na producentów. Oczywiście, zakłada się też znaczny stopień swobody działania użytkowników, a przede wszystkim - możliwość wydatkowania środków zgodnie ze swą wolą.

Pogląd o prymacie użytkowania przed konstruowaniem nie jest oczywiście sprzeczny z interesem producenta nadążającego za

wymaganiami użytkowników i w najmniejszym stopniu nie umniejsza swobody konstruktorów. Zmusza jedynie producentów do głębszego rozpoznania potrzeb potencjalnych użytkowników i do starannej selekcji dostępnych konstrukcji. Innymi słowy, dążący do osiągnięcia sukcesu ekonomicznego producent powinien kierować się potrzebami użytkowników bardziej niż pomysłowością konstrukcji czy wygodą (konserwatyzmem) zespołu produkcyjnego.

Powyższe rozważania są dość ogólne, nie do samej tylko informatyki mają zastosowanie. Jednakże z nich właśnie i z analizy sytuacji ekonomicznej, panującej w krajach intensywnego rozwoju zastosowań informatyki, wynika moje przekonanie o mającym miejsce ekonomicznym prymacie użytkowania przed konstruowaniem urządzeń informatyki.

Istotnie, wystarczy zwrócić uwagę na dwa przykłady, z dwu przeciwległych krańców skali.

Jak powszechnie wiadomo, koncern IBM nie należy do firm oferujących ostatnie nowinki techniczne. Pod względem konstrukcji, komputery tego producenta są z reguły o kilka lat opóźnione w stosunku do wielu znacznie mniejszych firm. A jednak to właśnie IBM od lat niezmiennie zwiększa swe zyski, utrzymuje dominującą - choć na szczęście nie monopolistyczną! pozycję wśród producentów. Rozbudowane laboratoria koncernu i szczodrze finansowane współpracujące z nim ośrodki uczelniane generują wiele wynalazków i nowych technologii. Na pierwszy rzut oka: same sprzeczności. Koncern dysponuje nowinkami, niezbyt żwawo wdraża je do produkcji, a przy tym znakomicie

prosperuje na rynku, na którym - jak twierdzę - panują interesy użytkowników. W rzeczywistości jednak nie ma żadnej sprzeczności.

Użytkownik, ten który ma dostateczny potencjał ekonomiczny by narzucać swą wolę producentom komputerów, wcale nie jest zainteresowany nowinkami technicznymi jako takimi. Ten użytkownik jest zainteresowany wyłącznie rzeczywistym usprawnieniem działalności, która wykonuje. Tego użytkownika wcale nie obchodzi, jak jest zbudowany komputer, ile ma pamięci, i jakie w nim są kostki. Tego użytkownika obchodzi ile zaoszczędzi lub o ile zwiększy swój zysk instalując system przetwarzania informacji. Dodatkowo, interesuje go jaką ma gwarancję, że system będzie funkcjonować tyle godzin na dobę i tyle dób w roku, ile ma funkcjonować. I jeszcze to, czy dostawca umie szkolić osoby, które mają z systemem bezpośrednio współpracować, czy dostarcza właściwą dokumentację itd. No i czy jeśli za kilka miesięcy coś się zepsuje, a za kilka lat będzie się chciało coś zmienić, to czy dostawca będzie natychmiast gotów i zdolny pomóc.

Otóż IBM właśnie to wszystko sprzedaje. A żeby móc sprzedawać nie tylko system doskonale dopasowany do potrzeb użytkownika, ale także gwarancję bezpieczeństwa i komfortu, nie może sobie pozwolić ani na zbyt raptowane wdrażanie nowości niezintegrowanych z całą dotychczasową historią, ani na żadne próby zbyt radykalnego naginania sposobu użytkowania systemów do wymagań sprzętu. Aby zrozumieć istotę sukcesu koncernu IBM, trzeba

pojąć, że sprzedaje on godne zaufania usługi informatyczne, a komputery - tylko jako element tych usług.

Przykład drugi dotyczy tanich komputerów masowych. Znana firma, założona przez wybitnego wynalazcę, Clive'a Sinclaira, po okresie błyskotliwego rozwoju, finansowanego rozbudzonym przez tę firmę popytem na komputerowe zabawki, upadła i została wykupiona przez firmę Amstrad, kierowaną przez handlowca, Alana Sugara. Amstrad odniósł sukces na rynku komputerowym nie dzięki wynalazkom, a dzięki doskonałemu rozpoznaniu potrzeb i możliwości potencjalnych klientów, oferując niezawodne, bardzo przydatne w prawie każdym domu komputerki do wygodnego pisania tekstów. W komputerkach Amstrada nie ma żadnych fajerwerków. Przeciwnie, są składane z najtańszych w danej chwili elementów. Ale zawsze są sprzedawane w komplecie wystarczającym do sprawnego i absolutnie niezawodnego wykonywania tych czynności, do których zostały pomyślane. No i kosztują znacznie taniej niż bardziej uniwersalne komputery, też używane przeważnie tylko do pisania tekstów.

Na masowym rynku pełno świetnych pomysłów, wiele tanich komputerów, które mogą robić i to, i tamto, i jeszcze owo. Tylko nie bardzo widać klientów indywidualnie zainteresowanych taką rozmaitością czynności. W życiu codziennym, rozmaitością wykonywanych czynności charakteryzują się nie poszczególne osoby, lecz zespoły. A komputer osobisty jest do użytku jednej osoby. Między uniwersalnością komputera jako typu, a specjalizacją urządzenia informatycznego pomocnego w pracy danej osoby jest gdzieś punkt równowagi. Znalezienie tego punktu wymaga wiedzy

wcale nie informatycznej, lecz handlowej: rozpoznania potrzeb rynku. I dlatego wynalazca Sinclair zrobił klapę, a handlowiec Sugar - fortunę. A poważnie, należy to rozumieć tak, że postawiona w sytuacji zupełnie swobodnego wyboru ekonomicznego społeczność użytkowników wybrała komputer pełniący wyraźnie określoną funkcję użytkową. I w tym przykładzie, empirycznie potwierdza się pogląd, że urządzenie jest wtórne względem funkcji.

Przedstawiwszy argumenty ogólne i zilustrowawszy je dwoma - mam nadzieję: ciekawymi - przykładami, czuję się w zupełności usprawiedliwiony przyjmując w dalszym ciągu pogląd, że wyznacznikami tendencji rozwojowych informatyki stosowanej będą nie tyle urządzenia, ile umiejętności ich użytkowania i możliwości zastosowań.

Nie oznacza to, że lekceważę uwarunkowania sprzętowe. Oczywiście, *brak urządzeń informatycznych wyklucza jakiekolwiek zastosowania informatyki*. Podobnie, w przypadku uniezależnienia producentów urządzeń informatycznych od woli potencjalnych użytkowników wystąpi patologia rozwoju zastosowań, o trudnych do przewidzenia skutkach i objawach (jak dotąd, rozwój informatyki przebiega właściwie wyłącznie w warunkach ekonomicznego uzależnienia producentów od użytkowników; inne drogi rozwoju, jeśli nawet możliwe, musimy na razie traktować czysto spekulatywnie). W obydwu jednak przypadkach, zarówno niedostępności urządzeń, jak i dyktatu producentów urządzeń, mamy do czynienia z sytuacją tak anormalną, że wiedza i doświadczenia informatyka stają się bezużyteczne, a nawet bezprzedmiotowe. Obydwa

przypadki dotyczyć mogą bowiem tylko takich stosunków gospodarczo-społecznych, w których informacja nie ma realnej wartości ekonomicznej, albo takich, w których wartości ekonomiczne nie decydują o postępowaniu podmiotów działalności gospodarczej. Innymi słowy, zarówno brak sprzętu, jak i jego chroniczne niedostosowanie do wmagań aplikacyjnych, świadczą o tak głębokim rozstroju gospodarczym, że dole czy niedole zastosowań informatyki nie mają większego znaczenia.

Wydaje się, że dalszy postęp zastosowań informatyki będą wyznaczać sukcesy (lub niepowodzenia) w rozwiązywaniu trzech wielkich problemów:

(1) Opanowanie metod dobrego oprogramowania.
(2) Opanowanie metod specyfikowania zadań i systemów.
(3) Opanowanie zagadnień krotności.

Każdy z tych problemów (tu sformułowanych hasłowo) mógłby stać się przedmiotem sporej monografii. Nie są też one wcale tak ostro rozgraniczone, jak by się mogło wydawać z powyższego wyliczenia. Resztę niniejszego referatu poświęcimy omówieniu poszczególnych problemów 'wielkiej trójki'.

Najlepiej udokumentowany jest problem dobrego oprogramowania. Ponieważ poświęcono mu wiele artykułów a nawet książek, nie musimy tu przedstawiać jego różnorodnych aspektów. Przełomem w sposobie traktowania pojęcia 'dobry program' było upowszechnienie przekonania, że *dobry program to przede wszystkim program poprawny*, czyli spełniający specyfikację, przy czym przez 'spełnienie specyfikacji' rozumie się spełnienie dobrze określonej relacji matematycznej. Przyjęcie tego kluczowego dla

naszych dalszych rozważań postulatu metodologicznego pozwoliło rozwinąć metody dowodzenia poprawności programów, a także - co znacznie ważniejsze z praktycznego punktu widzenia - metody programowania gwarantujące poprawność programów na mocy konstrukcji.

Okazało się przy tym, że zalecane od wielu lat przez czołowych metodologów techniki poprawnego programowania, takie jak używanie instrukcji strukturalnych, unikanie skoków, konsekwentna modularyzacja itd., znajdują pełne uzasadnienie w świetle analitycznej teorii poprawności programów. W ten sposób, badania teoretyczne nie tylko stworzyły jednolitą, ścisłą bazę koncepcyjną wielu pojęć używanych uprzednio w sposób intuicyjny, ale także wyjaśniły, dlaczego pewne techniki postępowania programistycznego prowadzą do lepszych niż inne programów. Kosztem pewnej matematyzacji pracy programistów osiągnięto więc znaczny postęp zarówno pojęciowy, jak i metodologiczny.

Osiągnięcia w metodologii programowania nie pozostały bez konsekwencji dla konstrukcji narzędzi pracy programistów. Pojawiły się nowe języki programowania mocno osadzone w nowych realiach współczesnej wiedzy o procesie programowania, zaczęto atestować translatory (dając ich użytkownikom nie *duże prawdopodobieństwo*, lecz *pewność* poprawnej pracy), zestaw narzędzi ekranowych wzbogacono o edytory strukturalne i systemy wspomagające, a nawet automatyzujące dowodzenie poprawności programów.

Jeśli cofniemy się myślą wstecz, do lat dominacji programowania w kodzie maszynowym, bez trudu zobaczymy, jak wielkiego

dokonano postępu. Jednocześnie, nie unikniemy spostrzeżenia, że wszystkie osiągnięcia w tej materii, od budowy translatorów do ścisłej analizy semantyki programów, zawdzięczamy matematyzacji, albo jeśli kto woli - formalizacji obowiązującego podejścia.

Jednakże, daleko nie wszystko zostało już zrobione. O ile umiemy sobie radzić z niewielkimi programami (powiedzmy z 'wnętrzami' modułów), o tyle układanie wielkich programów nadal nastręcza wiele trudności. Podobnie, względnie dobrze opanowane zostało programowanie sekwencyjne, podczas gdy programowanie systemów współbieżnych ciągle nastręcza trudności dość podstawowego charakteru. Nie trudno dostrzec, że aczkolwiek wymieniamy te trudności pod nagłówkiem problemu (1), wiążą się one bardzo ściśle z pozostałymi dwoma problemami.

Istotnie, fundamentalne trudności pisania wielkich programów wynikają z dwu źródeł: ich specyfikacje są na ogół znacznie mniej dokładne niż specyfikacje małych programów, a ponadto wielkie programy prawie zawsze wiążą się z krotnościami dwojakiego rodzaju: w systemach, dla których się je tworzy z reguły mamy do czynienia z tą czy inną postacią nieusuwalnej współbieżności zjawisk, a ponadto programy tej klasy są zazwyczaj pisane przez wiele osób naraz.

Zagadnieniem specyfikacji zajmiemy się nieco dalej. Teraz zasygnalizujemy tylko, że w bardzo wielu praktycznie ważnych dziedzinach specyfikacje są nie tylko nieprecyzyjne, ale także płynne w tym sensie, że ulegają zmianie w trakcie pracy nad programem, albo tuż po jej ukończeniu, często wprost na skutek

rozpoczęcia eksploatacji systemu zgodnego z poprzednią wersją specyfikacji.

Współdziałanie wielu osób przy realizacji jednego wielkiego programu kryje w sobie niezmierzone wręcz możliwości nieporozumień, niezgodności i opuszczeń. Trudności tych nie daje się uniknąć bez narzucenia rygorystycznej dyscypliny, tak drobiazgowej, że jej przestrzeganie przez uczestników kolektywnego przedsięwzięcia jest praktycznie niemożliwe, a w każdym razie niezwykle uciążliwe.

Połączenie tych dwu aspektów pracy nad stworzeniem wielkiego programu: płynności specyfikacji i wzajemnych uwarunkowań pracy zespołowej prowadzi do apokaliptycznego skomplikowania pracy. W trakcie trwającej nieraz kilka lat pracy powstają dziesiątki wariantów rozwiązań poszczególnych modułów, oddzielne moduły pisane są w różnych językach, często przy różnych założeniach co do zasad współpracy z innymi modułami, a nawet z systemem zarządzania. Bardzo często zdarza się, że w trakcie opracowywania takiego programu pojawiają się nowe elementy sprzętowe systemu, które z tych czy innych względów należy uwzględnić. Wszystko to stwarza niezwykle skomplikowaną sieć powiązań logicznych i uwarunkowań typu 'jeśli moduł X będzie w wariancie N, to moduły Y i Z muszą być w wariancie B, zaś moduł M należy zastąpić połączeniem modułów K i L w wariantach C i D'.

Nie trudno zgadnąć, że jedyną drogą umożliwiającą skuteczne prowadzenie prac programistycznych w takich warunkach jest stworzenie wydatnie zautomatyzowanego środowiska, które

przejmuje na siebie ciężar zapewnienia zgodności różnych wersji i wariantów, pilnowania, żeby rozdane zadania cząstkowe były realizowane i weryfikowane zgodnie z ustalonymi dla nich kryteriami poprawności itp. Tego rodzaju środowiska - zasady ich tworzenia i eksperymentalne realizacje - znajdują się obecnie w centrum uwagi przodujących firm softwarowych.

Nie są to produkty małe ani tanie. Eksperymentalne środowisko programistyczne ISTAR opracowane przez brytyjską firmę Imperial Software Technology liczy ok. 10 mln instrukcji i wymaga co najmniej 100 MB pamięci dyskowej. Zależnie od zakresu zakontraktowanych przez nabywcę usług kosztuje od 300 tys. do 3 mln. funtów. Klienci czekają w kolejce, trwają prace nad nowymi wersjami ISTARu, m.in. nad przygotowaniem specjalnych stanowisk pracy dla programistów: kompletów, składających się ze specjalnego komputera z ogromnym, kolorowym ekranem graficznym, szybkim procesorem i bardzo pojemnym dyskiem typu winchester. Mimo szacowanego na 50 - 100 tys. funtów kosztu sprzętowego wyposażenia takiego stanowiska pracy, gros kosztów środowiska stanowić będzie jego oprogramowanie: rozproszona wersja ISTARa.

Mimo tak wielkich kosztów, *środowiska programistyczne stanowią jeden z najważniejszych kierunków rozwoju informatyki stosowanej*. Prace programistyczne nie mogą na dłuższą metę rozwijać się w technologiach ekstensywnych, pracochłonnych. Tak jak wszystkie działy przemysłu muszą przejść na technologie intensywne, kapitałochłonne. Inaczej czeka je blokada rozwoju, wynikająca z coraz ostrzejszego deficytu kwalifikowanych pro-

gramistów i ugrzeźnięcia w chaosie nieskoordynowanych zadań cząstkowych.

Co do konieczności stosowania środowisk programistycznych nikt z autorytetów inżynierii oprogramowania nie ma większych wątpliwości, tak samo jak np. nikt nie wątpi, że wydajna praca programisty nie jest możliwa bez dostępu do dobrego translatora. Ale też nikt ze specjalistów nie głosi, że środowiska programistyczne rozwiąża wszystkie problemy. W szczególności, udana konstrukcja samych środowisk zakłada prawidłowe rozwiązanie problemów specyfikacji i krotności, a także odkrycie istotnych praw rządzących procesem programowania.

Co do problemu specyfikacji, to nie da się go omówić nie poświęciwszy nieco uwagi filozofii tego zagadnienia.

Już w początkowych latach stosowania komputerów zauważono istotną różnicę w jakości oprogramowania służącego do rozwiązywania zadań z nauk ścisłych i techniki z jednej strony i zadań dotyczących innych typów działalności. O ile np. dla fizyki czy budowy mostów dość szybko powstały w miarę powszechnie stosowane biblioteki procedur obliczeniowych i prawie wszyscy użytkownicy maszyn liczących posługiwali się tymi samymi algorytmami funkcji elementarnych, o tyle prawie każde przedsiębiorstwo stosowało odmienne programy sporządzania listy płac i tworzyło własne systemy gospodarki materiałowej.

W miarę upływu czasu i rozwoju zastosowań, narastało ogólne niezadowolenie z jakości oprogramowania; coraz bardziej narzekano na nieterminowość zespołów programujących, coraz częściej

klienci byli niezadowoleni z tego, co im programiści dostarczali. Bliższa analiza tego zjawiska, ochrzczonego mianem *kryzysu oprogramowania*, wykazuje jednak, że wśród niezadowolonych stosunkowo niewielu było klientów reprezentujących nauki ścisłe.

Z drugiej strony, publikowane w literaturze przykłady konstrukcji pięknych programów dotyczyły prawie wyłącznie zadań bardzo precyzyjnie wyrażonych w ściśle matematycznych kategoriach. Krytycy naukowych, precyzyjnych metod programowania, korzystając z faktu, że zwolennicy takich metod używają nazwy *metody formalne*, uznali za stosowne stwierdzić, że metody formalne nadają się tylko do zadań małych. Rozpowszechniło się więc przekonanie, że małe zadania można rozwiązywać metodami formalnymi, a dla wielkich potrzebne są inne, nieformalne metody.

Natura problemu polega jednak na czymś zupełnie innym.

Program komputerowy jest z istoty rzeczy *tworem formalnym*, tj. obiektem nie posiadającym żadnych innych własności, poza tymi, które przy pomocy jednoznacznych reguł można wyprowadzić z wyraźnie sformułowanych zasad. Nawet sam wynik wykonania programu dla konkretnych danych jest taką właśnie własnością. Jeśli sformułowanie problemu też jest tworem formalnym (w wyżej podanym sensie), wówczas przekształcenie sformułowania problemu w program jest przekształceniem jednego formalnego obiektu w inny. I choć nie zawsze jest to łatwe, a czasem nawet zgoła niemożliwe, to zawsze jednak takie *wyprowadzenie programu ze specyfikacji* można przedstawić jako ciąg dobrze

określonych przekształceń. (Zadania nierozwiązalne wymagają nieskończonego ciągu przekształceń.)

W ramach tego schematu można przedstawić wszystkie znane metody wyprowadzania programów (programowanie strukturalne, metodę kolejnych uściśleń, programowanie zstępujące, metodę Jacksona itp.) Jednocześnie, na wyprowadzenie programu można nałożyć różne ograniczenia, np. żądając, aby każdą własność, przysługującą specyfikacji, przysługiwała też programowi (oczywiście po zastosowaniu tych samych przekształceń, które przetwarzają samą specyfikację w program), uzyskujemy schemat wyprowadzeń programów poprawnych itd.

Okazuje się więc, że powodzenie uznanych metod programowania opiera się na traktowaniu specyfikacji (sformułowania problemu) jako obiektu formalnego. Dla zadań matematycznych jest to oczywiste. Dla zadań dotyczących wielu innych interesujących dziedzin zastosowań - o wiele trudniejsze do przyjęcia.

Nauki ścisłe, takie jak je dziś znamy, są wynikiem wielowiekowego procesu tworzenia teorii przystosowanych do formalnego postępowania. Dlatego właśnie formalnie poprawne formułowanie zadań nie sprawia większych trudności w fizyce czy chemii. W konsekwencji, dla tych nauk potrafimy budować dobre oprogramowanie.

Inaczej, niestety, wygląda sprawa wtedy, gdy przystępujemy do specyfikowania zadań w dziedzinach, które nie wykształciły jeszcze teorii pozwalających na formalne postępowanie. Proces budowy specyfikacji staje się de facto procesem budowy

częściowej przynajmniej *teorii* takiej dziedziny. I o ile droga od formalnej specyfikacji do programu - przynajmniej w zasadzie - podlega obliczalnym regułom postępowania, o tyle proces tworzenia teorii, czyli proces pisania specyfikacji zadań, takich reguł nie zna i znać nie może.

Istotnie, dwa formalne obiekty można powiązać relacją (np. dowodem poprawności), która - przynajmniej w praktycznie interesujących przypadkach - jest obliczalna. Stąd, m.in., wynika całkowity bezsens stosowania empirycznych środków sprawdzania poprawności programów (równie 'uzasadnionych', co np. doświadczalne sprawdzanie twierdzenia Pitagorasa!). Natomiast poprawności teorii względem dziedziny świata rzeczywistego, którą ma opisywać, nie da się, niestety, udowodnić w matematycznym sensie tego słowa. Weryfikacja teorii, pominąwszy pewne sytuacje specjalne, kiedy to badamy jej zgodność z innymi teoriami, chwilowo uznawanymi za 'prawdziwe', *musi obejmować postępowanie eksperymentalne*. Doświadczenie, którego wynik jest niezgodny z teorią, obala ją, albo przynajmniej poważnie nadwerężа jej wiarogodność. Najdłuższy nawet program badań doświadczalnych, których wyniki są zgodne z teorią, nie dowodzi jeszcze jej poprawności. Dziedziny świata rzeczywistego, w przeciwieństwie do dziedzin matematycznych, *nie są bowiem obiektami formalnymi* i wcale nie wszystkie ich właściwości wypływają na mocy przyjętych reguł z przyjętych zasad.

Tak więc, problem konstruowania dobrych specyfikacji jawi się nam w dwoistej postaci: muszą one być na tyle formalne, by pozwalały na wyprowadzanie programów (tj. by można zastosować

metodologicznie uzasadnione techniki programowania) i na tyle zgodne z dziedziną zastosowania, by można na nich polegać.

Uświadomiwszy sobie tę podwójną trudność problemu specyfikacji, możemy zapytać, jaki ma to związek z rozwojem zastosowań informatyki? Odpowiedź składa się z kilku części.

Po pierwsze, ogromna większość zastosowań informatyki dotyczy dziedzin, które nie mają dobrych teorii formalnych. Jeśli chcemy dla takich zastosowań tworzyć dobre oprogramowanie, musimy dla nich budować specyfikacje, tworząc de facto formalne teorie tych dziedzin. Innej drogi po prostu nie ma. W tych więc dziedzinach, problem specyfikacji stoi przed nami w całej rozciągłości, ułomności istniejącego oprogramowania i trudności jego budowy prawie zawsze zaczynają się w momencie zlekceważenia jednego (lub obu!) aspektów problemu specyfikacji.

Po drugie, należy się liczyć z intensywnym rozwojem środków językowych ułatwiających wyrażanie formalnych specyfikacji. Pojawiają się liczne języki pisania specyfikacji, takie jak np. Clear, Iota, Z, Larch. Pojawiają się też bardziej rozbudowane systemy, jak np. VDM, pozwalające nie tylko specyfikować, lecz także przekształcać specyfikacje formalne. Obejmując zakresem swych zastosowań znaczną część drogi od abstrakcyjnej specyfikacji do programu, systemy takie w istotny sposób upraszczają stosowanie metodologicznie poprawnych technik postępowania programistycznego. W wielu ośrodkach tworzy się obecnie systemy oprogramowania warsztatowego, ułatwiające pisanie i operowanie na formalnych specyfikacjach. Wielkie programy badawcze w zakresie informatyki (takie jak program

Esprit krajów EWG, czy program Alveya w Wielkiej Brytanii) wyraźnie faworyzują ten typ badań i konstrukcji. Ich uzupełnieniem są programy badawcze tworzenia systemów przekształcania proghramów z zachowaniem ich treści, np. system CIP Uniwersytetu Technicznego w Monachium.

Należy zauważyć, że korzystanie ze środków ułatwiających i wspomagających pisanie formalnych specyfikacji stawia przed programistami nowe wymagania: muszą oni w pewnym przynajmniej stopniu opanować matematyczne podstawy nowoczsnej teorii programowania. Wynikają z tego określone zadania dla szkolnictwa, a także konieczność intensywnego dokształcenia programistów już pracujących. Zaniedbanie tego uniemożliwi korzystanie ze środków, których stosowanie wydatnie ułatwia pisanie specyfikacji - coraz bardziej centralny aspekt wdrażania informatyki.

Po trzecie, przed praktyka informatyki staje problem treściowej weryfikacji specyfikacji, tj. wiarogodnych środków sprawdzania ich zgodności z wyobrażeniami o dziedzinie. Jeszcze inaczej powiedziawszy, musimy nauczyć się metod *doświadczalnego sprawdzania specyfkacji*.

Dwa nader interesujące kierunki badań przynoszą już wyniki w tym zakresie. Znaczne zainteresowanie tzw. *programowaniem w logice* (Prolog i podobne języki) wynika właśnie stąd, że traktuje ono program jako opis problemu, albo, dualnie, że opis problemu, wyrażony w pewnym formalizmie, pozwala ono traktować jako program i - choć niezbyt sprawnie - wykonać. Można w ten sposób weryfikować skutki opisu, a więc doświadczalnie sprawdzać jego wierność względem wyobrażeń o opisanej

dziedzinie. Podobny charakter mają badania nad możliwościami tzw. *szybkiego makietowania* (rapid prototyping), tj. szkicowego przekształcania specyfikacji w 'chodzący' program. Budując taką makietę stosuje się postępowanie zachowujące zgodność programu z głównymi cechami specyfikacji, pomijając troskę o sprawność, niezawodność itp. cechy programu. Badając działanie programu-makiety weryfikuje się poprawność specyfikacji względem dziedziny zastosowań. Obydwa te kierunki badań są już przekształcane w użyteczne systemy wspomagania programowania, ich zastosowanie ogromnie skraca czas weryfikacji specyfikacji a przez to pozwala na znacznie wszechstronniejsze badanie ich jakości. Ponieważ badanie jakości specyfikacji poprzedza kosztowny proces wyprowadzenia ostatecznej postaci programów, znacznie zmniejsza się ryzyko poniesienia strat wynikających z uruchomienia systemów niezgodnych z oczekiwaniami, choć całkowicie zgodnych z (niezweryfikowaną) specyfikacją.

Po czwarte wreszcie, daje się zaobserwować bardzo intensywne poszukiwanie takich fragmentów dziedzin zastosowań, dla których można zbudować dostatecznie dobre parametryczne teorie-specyfikacje. Chodzi tu np. o wyodrębnienie z rozległej dziedziny księgowości tych działań, które w mało różniącej się postaci występują we wszystkich lub prawie wszystkich systemach finansowych. Dla tak wyizolowanych poddziedzin buduje się programy, które po prostym wprowadzeniu kilku czy kilkunastu parametrów będą przydatne dla prawie każdego użytkownika zainteresowanego daną dziedziną. W ten właśnie sposób powstają pakiety software'owe przeznaczone na rynek masowy. Powodzenie takich pakietów jest ogromne. Niektóre zostały sprzedane w

setkach tysięcy egzemplarzy, co pozwoliło wydatnie obniżyć cenę, a przez to wyraźnie rozszerzyć krąg użytkowników. Należy przypuszczać, że poszukiwanie takich uniwersalnych poddziedzin, ich specyfikowanie i budowa odpowiedniego oprogramowania będą i nadal odgrywać wielką rolę, zwłaszcza dla użytkowników mikrokomputerów.

Należy podkreślić, że sukces takiego podejścia do budowy oprogramowania (w tym przypadku całkowicie równoważny z sukcesem zastosowania) zależy od nienagannego wprost wyspecyfikowania problemu. Masowo sprzedawane oprogramowanie nie może być poprawiane ani uzupełniane ex post, chociażby ze względu na anonimowość nabywcy. Specyfikacja, której nietrafność wykaże dopiero plajta na rynku, to klęska, na którą trudno sobie pozwolić wtedy, gdy sprzedawany pakiet został nisko wyceniony w nadziei sprzedania pół miliona kopii!

Przechodząc do problemu krotności, chciałbym zwrócić uwagę na pewną metodologiczną trudność w jego ujmowaniu. Pół żartobliwie, należałoby wręcz zapytać, czy *krotność* to problem, czy rozwiązanie problemu? Chodzi o to, czy wszelkiego rodzaju krotności, z jakimi mamy do czynienia we współczesnych systemach liczących (wielość procesorów, wielość współbieżnie wykonywanych programów, wielość komputerów połączonych siecią itd.), stanowią środki rozwiązywania zadań lub choćby środki usprawniające tę czynność, czy też owe krotności stwarzają trudności, które należy dopiero przezwyciężyć przy rozwiązywaniu zadań, albo same w sobie stanowią problem do rozwiązania.

Sprawa nie jest tak prosta, jak mogło by się wydawać na pierwszy rzut oka. Zilustrujmy to przykładem.

Wiadomo, że nie istnieje algorytm podziału dowolnego programu na części, które można wykonywać niezależnie. Nie można więc żywić nadziei, że kiedykolwiek będziemy w stanie całkowicie wyeliminować konieczność świadomego aranżowania rozwiązania w postaci współbieżnych procesów obliczeniowych.

Oczywiście, zastrzeżenie to nie dotyczy specjalnych klas problemów, np. niektóre problemy numeryczne (takie, jak większość zadań macierzowych) charakteryzują się budową wewnętrzną pozwalającą na algorytmiczną aranżację w postaci równoległych procesów obliczeniowych i to praktycznie dowolnej krotności. Można więc budować niesłychanie sprawne *układy systoliczne* do rozwiązywania takich zadań, a także stosować nieco bardziej uniwersalne, choć - per saldo - mniej efektywne *procesory macierzowe* czy *wektorowe* i zawierające je superkomputery typu Cray.

Podobnie, formułując problemy w postaci uproszczonych formuł rachunku logicznego (tzw. klauzul Horna) - co stanowi istotę najbardziej rozpowszechnionej wersji programowania w logice (Prolog), uzyskujemy możliwość swobodnego korzystania z dostępnej równoległości torów obliczeniowych sprzętu. To właśnie spostrzeżenie leży u podstaw architektury tzw. *komputerów piątej generacji*, pomyślanych jako rozwiązanie trapiącego japoński przemysł elektroniczny problemu nadprodukcji układów liczących. W tym samym kierunku idą pomysły proponentów tzw. *programowania funkcjonalnego*.

Pozostając jednak przy tradycyjnym poglądzie na uniwersalność sprzętu liczącego, tj. obstając przy tym, że przechodząc do kolejnego zadania, nie mamy zamiaru zmieniać komputera, nie możemy lekceważyć faktu, iż uwzględnienie potencjalnych możliwości krotnego sprzętu *wymaga dodatkowego wysiłku przy programowaniu*. Tak więc, przynajmniej wtedy, gdy chcemy zachować uniwersalność komputera, korzystanie z jego krotności jest okupione dodatkową pracą człowieka. Tego rodzaju zależność zawsze dotąd w informatyce świadczyła o jakimś zasadniczym błędzie w sztuce, jak bowiem pamiętamy, komputery są po to by ułatwiały pracę, nie zaś po to, by wkładać dodatkową pracę w celu ich 'wykorzystania'.

Znaczenie dylematu krotności zaczyna docierać do producentów sprzętu komputerowego. Firma Inmos, twórca i producent wielce sprawnego układu do budowy silnie równoległych komputerów, transputera, lansuje język Occam, pomyślany jako język programowania współbieżnego. Podstawowe koncepcje Occama wywodzą się z Hoare'owskiej koncepcji komunikujących się procesów sekwencyjnych. Współbieżność programów pisanych w Occamie jest więc całkowicie zależna od ręcznej aranżacji. Jedyne narzędzia specjalne, jakich programiście dostarcza ten język, to mechanizmy synchronizacji, polegające na nadzorowanej i bezpiecznej transmisji komunikatów między strumieniami obliczeń. Jest to koncepcja bezpieczna, ale i w niej krotność jawi się jako problem, lub co najwyżej - surowy 'materiał', z którego programista ma dopiero zbudować użyteczne rozwiązanie. Tym niemniej, jest to przykład ogromnego kroku naprzód, jaki dokonał się w poglądach producentów sprzętu: rozumieją już oni, że

powodzenie sprzetu zależy od jego akceptacji przez programistów, że nie mogą już nikogo zmusić do używania tego czy innego sprzętu. Mariaż transputera z Occamem świadczy o uznaniu konieczności prezentowania sprzetu *łącznie z atrakcyjnymi środkami jego użytkowania*.

Problem, czy można liczyć na silniejsze niż wymiana komunikatów środki synchronizacji w językach programowania, jest nadal otwarty. Niektóre języki, tworzone specjalnie dla użytkowników wyraźnie zainteresowanych rozbudowanymi systemami współbieżnymi, takie jak Chill i Ada, mają tych mechanizmów bardzo wiele, jednakże przyjęcie ich przez użytkowników jest dość oziębłe. Być może język Modula-2, leżący gdzieś w połowie drogi między spartańską zwięzłością Occama a barokowym rozgadaniem Ady, zyska taką sympatię programistów modularnych systemów krotnych, jaką wśród programistów procesów sekwencyjnych cieszy się Pascal.

Jedną z niewielu rzeczy, które można dość bezpiecznie przewidzieć, jest dalszy i coraz intensywniejszy rozwój sieci komputerowych. Co więcej, wygląda na to, że sieci te będą nie tylko zupełnie heterogeniczne (zarówno pod względem typów partycypujących komputerów, jak i ich klas), lecz także iż będą obejmować, obok komputerów, wiele typów innych urządzeń przechowywania i transmisji informacji. W zasadzie już dziś można powiedzieć, że nie ma wyraźnego rozgraniczenia między usługami, jakie świadczy sieć łączności i sieć komputerowa. W niedalekiej zapewne przyszłości dołącza do nich sieci rozpowszechniania informacji tekstowej, graficznej, dźwiękowej i,

zapowne, wizualnej (telewizji). W każdym razie, poważne koncerny telekomunikacyjne i komputerowe z wielkim rozmachem podejmują prace nad oprogramowaniem tego rodzaju sieci informacyjnych. Wtórują im banki i inne instytucje finansowe: obieg pieniądza, udzielanie kredytu i płatność należności coraz bardziej staja się kwestia wprowadzenia odpowiedniej informacji do sieci. Jak mało która dziedzina życia cywilnego, łączność, usługi informacyjne i finanse zmieniaja swoje oblicze pod wpływem zastosowań informatyki.

Ogrom środków finansowych, którymi dysponuja te dziedziny gospodarki, zatrudniające w krajach do końca uprzemysłowionych gros pracowników, pozwala im na szczodre finansowanie badań i odważne wprowadzanie eksperymentalnych systemów. W wielu przypadkach eksperymenty okazują się niezwykle udane. Dla przykładu można wymienić komputeryzacje pracy redakcji i zecerni gazet, pozwalająca praktycznie wyeliminować potrzebę zatrudniania niemerytorycznych pracowników wydawnictwa, tj. zmniejszyć globalne zatrudnienie o ponad połowę, sytemy elektronicznego przekazywania pieniędzy (EFT), pozwalające obciążać konto nabywcy i zwiększać stan konta sprzedawcy w momencie dokonywania transakcji (np. regulowania rachunku w kasie sklepowej) i tym samym wyzwalające znaczne sumy dotychczas nie pracujące dla gospodarki w czasie tranzytu itp.

Fuzja informatyki z łącznością stanowi niezwykle silny czynnik kształtowania nowej cywilizacji. Generalnie powoduje ona wydatne polepszenie usług przy jednoczesnym wyraźnym zmniejszeniu zużycia zasobów wyczerpywalnych - materiałów i energii.

Wynika stąd absolutna nieodwracalność tego kierunku ewolucji cywilizacyjnej. Sieci informatyczne odegrają taką samą rolę jak niegdyś koleje żelazne i druk razem wzięte. Dla informatyki oznacza to stałe i niezwykle bogate źródło finansowania jej rozwoju, ale i przyczynę stałych trudności.

Na zakończenie omawiania problemów krotności, chciałbym wspomnieć o jednym jeszcze aspekcie, niezwykle ważnym, a często lekceważonym w początkowych, entuzjastycznych etapach tworzenia sieci. Chodzi o fundamentalną różnicę między indywidualnym używaniem osobistego komputera, a indywidualnym charakterem realizowanych przy jego pomocy funkcji.

Istnieją, oczywiście, pewne, niezbyt co prawda liczne, przypadki, w których obydwa zakresy indywidualizacji pokrywają się. Nawet jeśli taki zupełnie indywidualny komputer jest przyłączony do sieci, nie koniecznie musi to pociągać upublicznienie realizowanych przy jego pomocy funkcji. W szczególności, bez trudu można sobie wyobrazić całkowicie idywidualne użytkowanie osobistego komputera, przy którym sięga się do publicznych zasobów informacji, w niczym nie zmieniając ogólnych warunków pracy pozostałych abonentów sieci, ani treści dostępnej im informacji. W ogromnej jednak większości przypadków, mimo indywidualizacji form korzystania z przyłączonego do sieci komputera osobistego, realizowane przy jego użyciu funkcje wywierają wpływ na szersze otoczenie. Czasem jest to zamierzonym skutkiem działania, często jednak - skutkiem zupełnie nieprzewidzianym, a nawet przeciwnym intencjom użytkownika.

Problem ten jest szczególnie poważny wtedy, gdy sieć komputerowa łączy, albo obejmuje, pewną liczbę komputerów osobistych, używanych przez osoby zatrudnione w jednej instytucji, siłą rzeczy korzystające - świadomie lub nie - ze wspólnych zasobów informacji. Indywidualizacja form korzystania z komputera osobistego wykształca u użytkowników poczucie fałszywego bezpieczeństwa informacyjnego, wyrażające się przeświadczeniem, że ma się pełną kontrolę nad wszystkimi zasobami komputera, zarówno sprzętowymi, jak i informacyjnymi. Tymczasem - nawet jeśli pracuje się tylko na 'własnych' kopiach zbiorów informacyjnych - można przyczynić wielu kłopotów całemu zespołowi wtedy, gdy nie dbając o aktualność pobranych kopii generuje się informacje wyjściowe (wyniki) *rzekomo poprawne*, a w rzeczywistości niekoniecznie zgodne z tymi, jakie uzyskałby inny użytkownik, stosujący te same algorytmy do *swoich* kopii zbiorów informacyjnych. Jeszcze groźniejsze mogą być skutki aktywnego korzystania z ogólnie dostępnych zbiorów informacji.

To, że współbieżna krotność korzystania z zasobów informacyjnych kryje w sobie poważne niebezpieczeństwo, było od dawna plagą wielodostępnych baz danych. Rozszerzenie zasady współbieżności na sieciowe konfiguracje komputerów osobistych, ogromnie zwiększając stopień indywidualizacji sposobu korzystania ze wspólnych zasobów, bardzo poważnie komplikuje ten problem, nie tylko w płaszczyźnie technicznej, lecz także - w psychologicznej. Ten ostatni aspekt nabiera szczególnej ostrości wtedy, gdy sieć powstaje przez połączenie uprzednio rozłącznych komputerów osobistych, których użytkownicy nie są przyzwyczajeni do respektowania nadrzędnej dyscypliny.

Jak sie wydaje, rozwiazaniem tego problemu bedzie odejście od koncepcji czysto łacznościowej funkcji sieci, w każdym razie w kontekście informatyki profesjonalnej, na rzecz rozproszonych systemów przetwarzania informacji, albo sieci z wyraźnie zaznaczoną funkcją koordynacyjno-kontrolną.