2-3
1982

informatyka

WYDAWNICTWO NOT SIGMA

Giełda informacji

Dla ośrodków informatycznych w Polsce nadszedł czas poważnej próby. Ograniczenia wynikające z obecnego stanu gospodarki oraz trudności zaopatrzeniowe — utrudniają im rozwój na miarę sił i aspiracji. Muszą podołać sytuacji, wbrew wszystkim kumulującym się przeciwnościom.

Chcemy — choćby w minimalnym stopniu — pomóc przedsiębiorstwom informatycznym w ich dzisiejszej walce z trudnościami. A możemy to zrobić tylko w jeden sposób: przyspieszając przepływ informacji — ułatwić wykorzystanie dotychczasowego potencjału kadry, sprzętu, gotowych systemów. Otwieramy zatem giełdę bieżących informacji (publikowanych w skróconym, czterotygodniowym cyklu). Proponujemy ośrodkom informatycznym, by wybrały swojego rzecznika prasowego, który będzie nas stale informował o nie wykorzystanych możliwościach ośrodka oraz o takich jego niedomaganiach, które dadzą się, z pomocą innych ośrodków, wyeliminować. Informacje te powinny być rzeczowe i precyzyjne: jakie mamy nowe, nieznane szerzej systemy, kto i na jakich warunkach może z nich korzystać; jaki mamy sprzęt, do czego mógłby być wykorzystany, na jakich zasadach; co może zaoferować nasza kadra, za jaką cenę; czego nam brakuje i co moglibyśmy dać w zamian. Również indywidualni informatycy (poszukujący — na przykład — odpowiedniej pracy) mogą odtąd zamieszczać na naszych łamach swoje propozycje i oferty.

Jeśli już nie możemy liczyć na informatyczny rozwój, pomagajmy sobie przynajmniej w maksymalnym wykorzystaniu istniejących jeszcze wszędzie rezerw. Brońmy się, starając się usprawnić własne pole działania.

REDAKCJA

NACZELNA ORGANIZACJA TECHNICZNA
WYDAWNICTWO
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
ul. Świętokrzyska 14a
00-950 Warszawa
skrytka pocztowa 1004

Redaktor naczelny: prof. dr hab. Leon ŁUKASZEWICZ

prof. dr hab. inż. Konrad FIAŁKOWSKI (zastępca redaktora naczelnego), mgr inż. Zbigniew GLUZA, dr Janusz GWIAZDA, Władysław KLEPACZ (zastępca redaktora naczelnego), dr inż. Tomasz PAWLAK, dr inż. Janusz ZALEWSKI
Sekretarz redakcji: mgr Teresa JABŁOŃSKA

RADA PROGRAMOWA

Prof. dr hab. Tadeusz PECHE (przewodniczący), mgr inż. Tomasz BAŃKOWSKI (sekretarz), mgr inż. Antoni BOSSOWSKI, mgr inż. Roman BURNO, prof. dr hab. Andrzej JANICKI, mgr inż. Jan KRAMARCZUK, prof. dr hab. inż. Juliusz KULIKOWSKI, prof. dr hab. Leon ŁUKASZEWICZ, prof. dr hab. Antoni MAZURKIEWICZ, gen. dr inż. Marian PASTERNAK, mgr inż. Bronisław PIWOWAR, mgr Zbigniew SUBSTYK, doc. dr hab. Tadeusz WALCZAK

Materiałów nie zamówionych Redakcja nie zwraca.

Redakcja: 00-011 Warszawa, ul. Jasna 14/16, pok. 131 i 133, tel. 27-71-40, dyżury redakcji 10.00—12.00

Zakł. Graf. „Tamka". Zam. 82. Obj. 5 ark. druk. Nakład 5000 egz. Z-47.

Cena egzemplarza zł 50.— | INDEKS 36124 | Prenumerata roczna zł 600.—

Zalewski J.: ADA — nowy język programowania (3). Typy i inne konstrukcje językowe

INFORMATYKA 1982, nr 2—3, s. 4

Trzecia część charakterystyki języka ADA. Omówiono pojęcia wyrażeń i nazw (identyfikatorów), różnych rodzajów typów oraz jednostek generycznych, ilustrując je licznymi przykładami. W zakończeniu podano podstawowe pozycje bibliograficzne pozwalające pogłębić znajomość tego języka.

Залевски Я.: АДА — новый язык программирования (3). Типы и другие языковые конструкции

ИНФОРМАТИКА 1982, № 2—3, стр. 4

Третья часть характеристики языка АДА. Оговариваются понятия выражений и названий (идентификаторов), различные виды типов и генерических единиц, иллюстрируемые многими примерами. В конце даются основные библиографические позиции, позволяющие углубить знания этого языка.

Stokalski A.: Projektowanie i budowa systemów informatycznych. Organizacja cyklu rozwojowego

INFORMATYKA 1982, nr 2—3, s. 8

Zasady prawidłowej organizacji projektowania systemu informatycznego. Wskazano na znaczenie i wpływ popełnianych błędów, scharakteryzowano cykl rozwojowy projektowania i poszczególne jego fazy z podkreśleniem znaczenia kontroli międzyfazowej. Przedstawiono koncepcję tzw. rozwoju generacyjnego oraz sformułowano podstawowe zasady postępowania podczas prowadzenia prac projektowych.

Стокальски А.: Проектирование и строительство информационных систем. Организация цикла развития

ИНФОРМАТИКА 1982, № 2—3, стр. 8

Принципы соответствующей организации проектирования информационной системы. Указывается значение и влияние допускаемых ошибок, охарактеризован цикл развития проектирования и отдельные его фазы, подчеркивается значение контроля между фазами. Представлены концепции так называемого генерационного развития, а также сформулированы основные принципы поведения во время проведения проектных работ.

Huzar A.: Weryfikacja programów — podstawowe pojęcia

INFORMATYKA 1982, nr 2—3, s. 13

Przegląd podstawowych pojęć oraz metod sprawdzania poprawności programów. Sformułowano wnioski dotyczące skuteczności tych metod.

Хузар А.: Верификация программ — основные понятия

ИНФОРМАТИКА 1982, № 2—3, стр. 13

Просмотр основных понятий и методов проверки правильности программ. Сформулированы предложения, касающиеся эффективности этих методов.

Fladrowska E.: Czas systemów powtarzalnych

INFORMATYKA 1982, nr 2—3, s. 15

Analiza niektorych istotnych przyczyn małej efektywności zastosowań informatyki w zarządzaniu przedsiębiorstwem, takich jak zła jakość i nieprawidłowy dobór powtarzalnych systemów informatycznych. W oparciu o wieloletnie doświadczenia podano propozycję właściwej metody wyboru powtarzalnego systemu informatycznego.

Флядровска К.: Время повторяющихся систем

ИНФОРМАТИКА 1982, № 2—3, стр. 15

Анализ некоторых существенных причин малой эффективности применения вычислительной техники в управлении предприятием, таких как плохое качество и неправильный подбор повторяющихся систем вычислительной техники. На основе многолетнего опыта предлагается соответствующий метод выбора повторяющейся системы вычислительной техники.

Ryznar Z.: S&DL — język specyfikacyjny do projektowania strukturalnego (zarys propozycji)

INFORMATYKA 1982, nr 2—3, s. 18

Ogólna charakterystyka języka specyfikacyjnego opracowanego przez autora na potrzeby projektowania strukturalnego. Podano notację i składnię języka, zasady opisu specyfikacyjnego oraz przykłady jego zastosowania dla wybranych typów obiektów.

Рызнар З.: S&DL — спецификационный язык для структурального проектирования (зарисовка предложений)

ИНФОРМАТИКА 1982, № 2—3, стр. 18

Общая характеристика спецификационного языка, разработанного автором для потребностей структурального проектирования. Указывается обозначение условными знаками и синтаксис языка, принципы спецификационного описания, а также примеры его использования для избранных типов объектов.

Dawidowski J., Owczarczak P.: Zastosowanie pakietu PSL/PSA do prowadzenia słownika-skorowidza bazy danych

INFORMATYKA 1982, nr 2—3, s. 21

Zwięzła charakterystyka pakietu programowego PSL/PSA oraz sposobu jego wykorzystania przy projektowaniu systemów informatycznych zarządzania. Wskazano na możliwość zastosowania tego pakietu do prowadzenia słownika-skorowidza bazy danych oraz podano zakres jego praktycznego stosowania w Akademii Ekonomicznej w Poznaniu.

Давидовски Я., Овчарчак П.: Применение пакета PSL/PSA для ведения словаря-указателя базы данных

ИНФОРМАТИКА 1982, № 2—3, стр. 21

Сжатая характеристика программового пакета PSL/PSA, а также способы его применения при проектировании информационных систем руководства. Оговаривается возможность применения этого пакета при ведении словаря-указателя базы данных, а также сфера его практического употребления в Экономической академии в Познани.

Sondej H., Wiśniewski J.: MOLATO — system wspomagający nauczanie

INFORMATYKA 1982, nr 2—3, s. 24

Ogólna charakterystyka zrealizowanego na Uniwersytecie Toruńskim komputerowego systemu nauczania z zastosowaniem komputera R-32. Omówiono budowę systemu, sposób jego wykorzystywania, potrzebną konfigurację sprzętu i podstawowego oprogramowania, a także doświadczenia z dotychczasowego stosowania i eksploatacji MOLATO.

Сондей Х., Вишневски Я.: МОЛЯТО — система, помогающая обучению

ИНФОРМАТИКА 1982, № 2—3, стр. 24

Общая характеристика осуществленной в Торуньском университете компьютерной системы обучения с применением компьютера Р-32. Оговорена постройка системы, способ ее использования необходимая конфигурация оборудования и основного программного обеспечения, а также некоторые замечания в адрес существовавшего по сегодняшний день применения и эксплуатации МОЛЯТО.

Zalewski J.: ADA — a new programming language (3). Types and other language structures

INFORMATYKA 1982, No 2—3, p. 4

Third part of the ADA language presentation. Conceptions of expressions and names (identifyiers), different type kinds and generative units, illustrated with many examples, are discussed. In the termination, basic bibliography items, which help to achieve better knowledge of the language, are presented.

Zalewski J.: ADA — eine neue Programmiersprache (3). Die Organisation des Entwicklungzyklus

INFORMATYKA 1982, Nr. 2—3, S. 4

Dritter Teil der ADA-Sprachecharakteristik. Es wurden die Begriffe von Ausdrücken und Namen (Identifikatoren), veschiedenen Typenarten und generativen Einheiten mit vielen Beispielen besprochen. Am Ende des Artikels wurden die wichtigsten Literaturangaben zur vertieften Kenntnis dieser Sprache angegeben.

Stokalski A.: Designing and building of data processing systems. Organization of the development cycle

INFORMATYKA 1982, No 2—3, p. 8

Principles for proper organization of data processing designing. Significance and influence of commited errors are pointed out, the designing development cycle and its individual phases, with emphasis on interphase check, are characterized. The idea of so called „generative development" is presented and basic principles of dealing during designing work are formulated.

Stokalski A.: Projektierung und Bau der EDV-Systeme. Organisation des Entwicklungzyklus

INFORMATYKA 1982, Nr. 2—3, S. 8

Grundlagen der ordnungsmässigen Organisation bei der EDV-Systemprojektierung. Es wurde Bedeutung und Einfluss der gemachten Fehler gezeigt, sowie Projektierungentwicklungzyklus und seine einzelne Phasen mit Betonung der Zwischenphasenkontrolle charakterisiert. Es wurde das Konzept sog. generativen Entwicklung vorgestellt und die wichtigsten Handlungsgrundsätze während der Projektierungarbeit formuliert.

Huzar A.: Program verification — basic notions

INFORMATYKA 1982, No 2—3, p. 13

Survey of basic notions and veryfying methods of program correctness. Conclusions referring to efficiency of these methods are formulated.

Huzar A.: Programmprüfung — die Grundbegriffe

INFORMATYKA 1982, Nr. 2—3, S. 13

Eine Übersicht der Grundbegriffe und Methoden zur Prüfung der Programmordnungsmässigkeit. Es wurden Folgerungen zur Thema der Wirksamkeit dieser Methoden formuliert.

Fladrowska E.: Times of repeatable systems

INFORMATYKA 1982, No 2-3, p. 15

Analysis of some essential reasons for insufficiently effective EDP application in business management, like bad quality and incornectly choice of repelable data processing systems, is discussed. On the base of many years experience a proposal of proper method for repeatable DP systems choice is presented.

Fladrowska E.: Die Zeit der wiederholbaren Systeme

INFORMATYKA 1982, nr 2—3, S. 15

Eine Analyse mancher Ursachen der kleinen Effektivität von Datenverarbeitunganwendungen im Bereich der Untermehmenverwaltung, wie z.B. schlechte Qualität und falsche Vorwahl des wiederholbaren EDV-Systems. Es wurde ein auf Grund der vieljährigen Erfahrungen erarbeiteter Vorschlag der Vorwahlmethode für die wiederholbaren EDV-Systeme angegeben.

Ryznar Z.: S & DL — specification language for structural designing (a draft proposal)

INFORMATYKA 1982, No 2-3, p. 18

General characteristics of the specification language, elaborated by the author for structured design are discussed. Language notation and syntax, rules for describing specification, as well as examples of application for selected types of objects are presented.

Ryznar Z.: S & DL — eine Spezifikationssprache für Strukturelle Projektierung (eine Vorschlagsskizze)

INFORMATYKA 1982, nr 2—3, S. 18

Eine allgemeine Charakteristik der Spezifikationssprache, die vom Autor für Bedürfnisse der strukturellen Projektierung erarbeitet wurde. Es wurden Schreibweise und Syntax der Sprache, die Grundsätze der Spezifikationsbeschreibung, sowie Beispiele ihrer Anwendung für ausgewählte Objektentypen angegeben.

Dawidowski J., Owczarczak P.: The PSL/PSA package application for carrying on the data base vocabulary-index

INFORMATYKA 1982, No 2—3, p. 21

Concise characteristics of the PSL/PSA program package and the way of its use for designing of data processing systems for management purposes. Possibility of the package application for carrying on the data base vocabulary-index is pointed out, as well as its practical application in the Economic Academy in Poznań is presented.

Dawidowski J., Owczarczak P.: Anwendung des Programmpakets PSL/PSA zur Führung eines Datenbasiswörterbuch-Verzeichnisses

INFORMATYKA 1982, Nr. 2—3, S. 21

Eine kurzgefasste Charakteristik des Programmpakets PSL/PSA und seiner Anwendung bei der Projektierung der EDV-Systeme für Managementzwecke. Es wurde die Anwendungsmöglichkeit dieses Pakets zur Führung eines Datenbasiswörterbuch-Verzeichnisses gezeigt und der Bereich seiner praktischen Verwendung an der Ökonomischen Akademie in Poznań angegeben.

Sondej H., Wiśniewski J.: MOLATO — a computer assisted teaching system

INFORMATYKA 1982, No 2-3, p. 24

General characteristics of the computer assisted teaching system, realized in the Toruń University on the R-32 computer, is presented. System's structure, way of its application, necessary hardware and basic software configuration, as well as experience of the MOLATO application and operation is discussed.

Sondej H., Wiśniewski J.: MOLATO — ein lernunterstützendes System

INFORMATYKA 1982, nr 2—3, S. 24

Allgemeine Charakteristik des rechnerunterstützten Lernsystems, das auf dem R-32 Rechner an der Toruń Universität realisiert wurde. Es wurden die Bau-und Ausnutzungweise des Systems, die nötige Hardware — und Basissoftwarekonfiguration, sowie die ersten Erfahrungen über die bisherige Anwendung und den Betrieb von MOLATO besprochen.

Informatyka

zastosowania w gospodarce, technice i nauce

Nr 2-3
MIESIĘCZNIK
1982
ROK XVII
Maj — Czerwiec

ORGAN KOMITETU INFORMATYKI, MINISTERSTWA NAUKI, SZKOLNICTWA WYŻSZEGO I TECHNIKI ORAZ KOMITETU NAUKOWO-TECHNICZNEGO NOT DS. INFORMATYKI

W NUMERZE: Strona

ADA — nowy język programowania (3). Typy i inne konstrukcje językowe
Janusz Zalewski 4

Projektowanie i budowa systemów informatycznych. Organizacja cyklu rozwojowego
Andrzej Stokalski 8

Weryfikacja programów — podstawowe pojęcia
Andrzej Huzar 13

Czas systemów powtarzalnych
Emilia Fladrowska 15

S&DL — język specyfikacyjny do projektowania strukturalnego (zarys propozycji)
Zygmunt Ryznar 18

Zastosowanie pakietu PSL/PSA do prowadzenia słownika-skorowidza bazy danych
Jan Dawidowski, Piotr Owczarczak 21

MOLATO — system wspomagający nauczanie
Halina Sondej, Janusz Wiśniewski 24

ALGORYTMY

Procedura generująca drzewa etykietowane
Andrzej Szałas, Zbigniew Świrski 26

Z KRAJU

Sytuacja kadry w ośrodkach informatyki
Grażyna Klajn-Zienkiewicz 28

POLSKIE TOWARZYSTWO INFORMATYCZNE
Z prac PTI *(B.O.)* 30

ZJEDNOCZENIE INFORMATYKI 31

Badania ankietowe użytkowników systemów informatycznych
Kazimierz Dudek, Józef Dziedzicki

ZE ŚWIATA

Przemysł komputerowy USA w roku 1980 34
Oprac. *Jan Ryżko*

Dysk optyczno-numeryczny zamiast mikrofilmowania? 36
Oprac. *P. A. Milewski*

RECENZJE

Sterowanie komunikacją w sieciach komputerowych 37
Mieczysław Bazewicz

TERMINOLOGIA

Terminologia języka ADA (cd.) 39
Janusz Zalewski

LISTY

Kilka uwag o „ALGORYTMACH" 40
Jacek Żebrowski

Trudne chwile informatyki?
Bartłomiej Kruszelnicki III okł.

JANUSZ ZALEWSKI
Instytut Badań Jądrowych
Warszawa

ADA – nowy język programowania (3)

Typy i inne konstrukcje językowe

Zanim omówimy pojęcie typu i rodzaje typów w języku ADA, poświęcimy kilka słów podstawowemu tworzywu omówionych już elementów języka (tj. instrukcji i jednostek programowych), a więc — wyrażeniom i nazwom (identyfikatorom). Szczegółowe ich omówienie zajęłoby zbyt wiele miejsca, przyjmiemy zatem, że są to pojęcia intuicyjnie znane i zrozumiałe.

W języku ADA nazwa, która jest ciągiem liter i cyfr zaczynającym się od litery i mogącym zawierać łącznik (symbolizowany przez podkreślenie) — nazywa się identyfikatorem. Każdy identyfikator musi być zadeklarowany przed użyciem. Nazwy mogą określać wiele elementów języka, przede wszystkim — stałe i zmienne, ale także: składowe indeksowane, np. X (1), wycinki (ang. slices), np. X (2..20) oraz — składowe zamknięte i atrybuty.

Składowych różnych elementów języka mających charakter zamknięty (np. bloku) można używać na zewnątrz przy użyciu zapisu „z kropką", tj. NAZWA IDENTYFIKATOR, gdzie NAZWA jest nazwą odpowiedniego elementu a IDENTYFIKATOR określa jego składową. Atrybuty są to określone właściwości niektórych elementów języka, wprowadzone przy definicji tych elementów i oznaczone przy użyciu zapisu „z apostrofem" NAZWA' IDENTYFIKATOR, gdzie NAZWA jest nazwą odpowiedniego elementu, a IDENTYFIKATOR określa jego właściwość. Przykłady składowych zamkniętych i atrybutów podaliśmy już omawiając pakiety i zadania, tj.

NAZWA.ZMIENNA -- użycie nazwy poza pakietem

NAME'COUNT -- liczba zadań w kolejce do wejścia NAME.

Wyrażenia opisują sposób obliczania wartości, a więc mogą się składać z nazw i operatorów. Zbiór zdefiniowanych operatorów obejmuje wszystkie powszechnie używane operatory arytmetyczne i logiczne, a także: konkatenację (&), operację modulo (MOD) oraz resztę (REM). Rzadko spotykanymi operatorami logicznymi są **and then** oraz **or else.** Wartością wyrażeń A1 △ A2 jest koniunkcja (alternatywa) argumentów liczone od strony lewej, przy czym liczenie jest przerywane przy pierwszej wartości FALSE (i odpowiednio: TRUE). Warto także wiedzieć, że potęgowanie (**) jest zdefiniowane tylko dla całkowitych wykładników, a operator nierówności zapisuje się dość oryginalnie jako /=.

TYPY

Różnicowanie nazw wraz z operatorami uzyskuje się przez pojęcie typu. Typ określa zbiór wartości i odpowiadających im operacji. Znaczy to, że dla każdego typu zdefiniowany jest zbiór operacji. Jest to pojęcie bardzo ważne, gdyż dzięki silnej typizacji danych uzyskuje się możliwość sprawdzenia już w fazie kompilacji poprawności programu w wielu aspektach, np. łatwo ochronić się przed przypadkową zamianą typów.

W języku ADA wyróżnia się typy skalarne, złożone, typ dostępowy i typy prywatne. Wszystkie zmienne tego samego typu muszą być zadeklarowane jako takie — nie wystarcza, że przybierają wartości z tego samego zbioru.
Wartości typu skalarnego charakteryzują się tym, że nie mają składowych. Do typów skalarnych należą typy liczbowe i typ wyliczeniowy (ang. enumeration).

Typ wyliczeniowy określa ciąg wartości w sposób jawny, np.

type MOTOR_CONTROL is (177, 100, 011, 000);

Istnieją trzy zdefiniowane systemowo typy wyliczeniowe: boolowski BOOLEAN, znakowy CHARACTER i łańcuchowy STRING, np.

type BOOLEAN **is** (FALSE, TRUE);

type CHARACTER is (NUL, SOH, ..., 'A', 'B', ..., DEL);
Zwróćmy uwagę, że znaki sterujące typu CHARACTER są oznaczone identyfikatorami, natomiast znaki alfanumeryczne są podane literalnie w pojedynczym cudzysłowie. Warto pamiętać, że typ wyliczeniowy może być wskaźnikiem iteracji i indeksem tablicy.

Do typów liczbowych należy typ całkowitoliczbowy INTEGER oraz — typy rzeczywiste FLOATING_POINT i FIXED_POINT, które stanowią przybliżenie liczb rzeczywistych. Typ zmiennoprzecinkowy określa przybliżenie z błędem względnym, np.

type REAL **is digits** 10;
a typ stałoprzecinkowy — z błędem bezwzględnym, np.

type FRACTION **is delta 0.001 range** 0.1..1.0;

Do atrybutów dowolnego typu TYP należą TYP'FIRST i TYP'LAST, które oznaczają najmniejszą i największą wartość typu TYP. Atrybutami typów dyskretnych D są:

D'POS (X) — położenie wartości X typu D

D'SUCC (X) i D'PRED (X) — następnik i poprzednik wartości X typu D

D'VAL (N) — wartość pozycji N typu D

Należy pamiętać, że dwa typy zadeklarowane oddzielnie są różne, nawet jeśli zawierają wartości z tego samego zbioru, tzn. przy deklaracjach

X, A: INTEGER;

B: NEW_INTEGER;
instrukcja

X:= A+B;
jest niepoprawna, natomiast poprawna jest instrukcja

X:= A+INTEGER (B); -- konwersja typów
Znaczy to, że w określonym wyrażeniu mogą wystąpić tylko obiekty tego samego typu. W tym celu należy dokonać jawnej konwersji (jeżeli zachodzi potrzeba) biorąc w nawiasy odpowiedni argument i poprzedzając go nazwą właściwego typu.

W tym miejscu warto wprowadzić pojęcie typu pochodnego (ang. derived type)

type POCHODNY **is new** TYP;
który jest użyteczny przy przenoszeniu oprogramowania i umożliwia uniknięcie używania typów zdefiniowanych wstępnie.

Przykładowo, jeżeli program zawiera deklarację

type REAL **is new** FLOAT;

to przenosząc go na komputer o krótszym słowie zachowujemy poprzednią dokładność obliczeń zastępując powyższą deklarację przez nową

type REAL **is new** LONG_FLOAT;

lub, na przykład

```
type REAL is digits 10;
```

W tym drugim przypadku wybór między FLOAT a LONG_FLOAT zostanie wykonany automatycznie.

Typu pochodnego nie można oczywiście mieszać z podstawowym.

Do typów złożonych zaliczamy obiekty o składowych jednakowego typu, tj. tablice (ang. arrays) lub — różnych typów, tj. rekordy (ang. records).

Deklarując typ tablicowy należy ustalić wymiarowość, typy indeksów i ich pary graniczne, które mogą pozostać nieustalone, oraz — typ elementów, np.

```
type MATRIX is ARRAY (INTEGER range <>) of BOOLEAN;
```

Wymiary tablicy mogą być statyczne, nieokreślone (jak powyżej) i dynamiczne, np.

```
type MATRIX1 is ARRAY (1..24) of INTEGER;
type VECTOR is ARRAY (1..N) of INTEGER;
```

Jednak indeksy typu tablicowego o nieokreślonych parach granicznych muszą być określone przy deklaracji obiektu, np.

```
M : MATRIX (1..10, 1..100);
```

Wartości indeksów mogą być dowolnego typu dyskretnego, np.

```
type MATRIX3 is ARRAY (DAY) of BOOLEAN;
```

gdzie

```
type DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);
```

Warto dodać, że w języku ADA można operować częścią tablicy, tzw. wycinkiem (ang. slice), np.

```
DATA (1..8) := MATRIX1 (17..24);
X := FUN (A(I..J));
```

Wartości typów złożonych nazywają się agregatami. Agregat można przedstawić w zapisie pozycyjnym, np.

```
TABLE := (4,4,4,2,0,0);
```

lub bezpośrednim (niepozycyjnym), np.

```
TABLE := (4 => 2,1/2/3 => 4, others = 0);
```

Typ rekordowy ma postać

```
type NAZWA is
  record
  -- deklaracje
  end record;
```

na przykład

```
type DATE is
  record
    JOUR : INTEGER range 1..31;
    MONS : MONTH_NAME;
    AN : NATURAL;
  end record;
```

gdzie

```
type MONTH_NAME is (JAN, FEB, ..., DEC);
type NATURAL is range 1.. INTEGER'LAST;
```

Deklaracja zmiennej typu DATE może mieć postać

```
TODAY : DATE := (30, SEPT, 1981);
```

Składowe rekordu mogą mieć nieokreślone rozmiary lub typy. Taki rekord zawiera stałą zwaną dyskryminatorem, od której zależą jego właściwości, np.

```
type BUFFER is
  record
    LENGTH : INTEGER; -- dyskryminant określający
                      -- długość bloku BLOCK
    BLOCK : ARRAY (1..LENGTH) of REAL;
  end record;
```

Dyskryminant można uważać za parametr rekordu i stosować zapis z nawiasami, np.

```
MESSAGE : BUFFER (200);
BRIEF : BUFFER (LENGTH => 50);
```

Typ rekordowy jest jedynym typem, który może mieć dyskryminanty.

Użycie dyskryminanta typu zilustrujemy przykładem rekordu z wariantami.

```
type SWITCHES is (R, W, D);
type FILE_HEADER (ATTRIBUTE : SWITCHES) is
  record
    NAME : STRING;
    OPEN : DATE; -- DATE jest typu rekordowego
  case
  when R => CHARACTERS : NATURAL;
  when W => CHARACTERS : NATURAL;
  when D => PROTECTED : BOOLEAN;
  end case;
end record;
```

Deklaracja zmiennej typu FILE_HEADER może mieć postać

```
FH0 : FILE_HEADER := (R, FILE0, (2,AUG,1981), 187));
FH1 : FILE_HEADER := (D, FILE1, (24, JUL, 1981), FALSE);
FH2 : FILE_HEADER (ATTRIBUTE => W);
```

Agregat rekordu z dyskryminantem musi mieć dyskryminant na pierwszym miejscu.

Dostęp do elementów typu rekordowego jest możliwy przy użyciu zapisu z kropką, np.

```
L := BRIEF.LENGTH;
OPEN := FH0.OPEN;
```

Po przedstawieniu podstawowych typów warto wspomnieć o stałych. Deklaracja stałej polega na użyciu słowa kluczowego **constant** oraz — podaniu jej typu, np.

```
X : constant REAL := 0;
N : constant INTEGER := A'LAST;
```

Wartość stałej nie musi być określona podczas deklaracji lecz później. Jest to tzw. stała przesunięta (ang. deferred constant), np.

```
PASSWORD : constant STRING;
```

Należy zaznaczyć, że deklaracja postaci

```
PI : constant := 3.14;
```

jest w rozumieniu podręcznika ADY deklaracją liczby, tzn. nadaniem nazwy liczbie. Można powiedzieć, że stała jest nazwą obiektu, który otrzymuje wartość tylko raz.

Typ dostępowy (ang. access type) określa wskaźnik do niezadeklarowanego obiektu innego typu, np.

```
type STACK_POINTER is access STACK;
```

Obiekty takie mogą być utworzone w programie przy użyciu tzw. alokatorów (ang. allocators). Przykładowo, alokator tworzy obiekt typu STACK

```
new STACK
```

i przypisuje wartość dostępu wskaźnikowi (tj. zmiennej typu dostępowego) MY_STACK

```
MY_STACK := new STACK;
```

który stanowi odsyłacz do tego obiektu. Inaczej mówiąc, zmienna typu dostępowego, MY_STACK, ma nadaną wartość dostępu **new** STACK i wskazuje ten nowy obiekt.

Typ dostępowy najczęściej odnosi się do rekordu (STACK jest typu rekordowego), ale odnosić się nie musi, np.

```
type REFERENCE is access INTEGER;
N : REFERENCE := new INTEGER (13);
```

Należy jednak pamiętać, że obiekty, do których odnosi się N, muszą być utworzone dynamicznie przez alokator, a nie mogą być zadeklarowane, tzn. podstawienie

```
N1 : REFERENCE := I;
```

po deklaracji

```
N1 : INTEGER;
```

jest błędne.

Domniemana wartość początkowa typu **access** jest równa **null**. Użycie typu dostępowego ilustruje poniższy przykład

```
type REF is access LINK;
type LINK is
  record
    LINK_DATA : STRING;
    POINTER : REF;
  end record;
HEAD : REF := new LINK (..., null);
```

Jeżeli konieczne jest użycie całego obiektu, a nie tylko wskaźnika, to należy wykorzystać atrybut **all**, np.

```
X := HEAD.all;
```

Typ prywatny umożliwia zasłonięcie sposobu reprezentacji typu, jeżeli reprezentacja ta jest nieistotna w używaniu pakietu lub podprogramu, a istnieją powody dla jej ukrycia (np. dla zabezpieczenia przed zmianą wartości, tj. w celu ochrony danych). Właściwości zewnętrzne typu zostają zachowane przy każdej zmianie reprezentacji, np.

```
private
type COMPLEX is
  record
  -- współrzędne prostokątne
  end record;
```

lub

```
private
type COMPLEX is
  record
  -- współrzędne biegunowe
  end record;
```

Często zachodzi potrzeba ograniczenia zakresu wartości typu bez zmiany zbioru operacji, co jest możliwe przez utworzenie podtypu, np.

```
subtype NATURAL is INTEGER range 1.. INTEGER'LAST;
```

Ograniczenie za pomocą podtypu może odnosić się nie tylko do typów prostych lecz i złożonych, tzn. można ograniczać indeksy tablic i warianty rekordów. Jednakże podtyp nie jest nowym typem.

W celu zwiększenia przenośności programu można posługiwać się typami pochodnymi (ang. derived types)

```
type NOWY is new STARY;
```

Niekiedy nie trzeba wprowadzać podtypu ani typu pochodnego, bo wystarcza ograniczony typ INTEGER, np.

```
type NOWY is range 1..200;
```

lub typ rzeczywisty, np.

```
type NOWY_FLOAT is digits 10 range LEWY..PRAWY;

type NOWY_FIXED is delta 0.001 range LEWY..PRAWY;
```

Jest interesujące, że pojedyncze zadanie można uważać za rozwinięcie pewnego typu, tj. typu zadaniowego (ang. task type).
Jest to o tyle ważne, że deklarując w programie określony typ zadaniowy można w dalszej części posługiwać się różnymi zadaniami tego samego typu, podobnie jak to się robi w odniesieniu do zmiennych. Deklarując przykładowo

```
task type SEMAPHORE is
  entry P;
  entry V;
end;
task body SEMAPHORE is
...
end SEMAPHORE;
```

można użyć w programie wiele semaforów, np.

```
S, FLAG : SEMAPHORE;
```

Możliwa jest deklaracja dostępu do typu **task type**, np.

```
task type DRIVER is
...
end;
type KEYBOARD is access DRIVER;
...
TERMINAL : KEYBOARD := new DRIVER;
```

Obiekt typu zadaniowego może być składową tablicy lub rekordu, a także — parametrem wejściowym podprogramu. Typy zadaniowe są bardzo użyteczne przy tworzeniu oprogramowania dla różnych klas niewiele różniących się obiektów, np. statków, samolotów, radarów (w zastosowaniach wojskowych).

PRZECIĄŻENIE NAZW

Typizacja języka, a więc istnienie i możliwość tworzenia różnych lecz zbliżonych typów, tak jak w przypadku ADY stwarza konieczność różnej interpretacji tych samych nazw. Właściwość ta dotyczy nazw i polega na nadawaniu im wielu znaczeń zależnych od kontekstu, co proponuję określić jako przeciążenie (ang. overloading).

Przykładowo, pojedyncza operacja może być wykonywana na orgumentach wielu typów. Choć nazwa operacji (symbole operatorów są w języku ADA zaliczane do nazw) może być we wszystkich przypadkach jednakowa (np. * dla mnożenia), to oczywiście jej wartość semantyczna będzie zupełnie inna w przypadku zmiennych typu REAL, **array** lub zmiennych rozmytych. Słowem, na podstawie typu argumentów kompilator powinien wnioskować, jakie jest znaczenie użytego operatora. Właściwość ta dotyczy wszystkich języków wysokiego poziomu, jak FORTRAN, ALGOL i in. — lecz w dużo mniejszym stopniu. Nietrudno domyślić się, że znak * w wyrażeniu języka FORTRAN

```
A * B
```

znaczy co innego, gdy A i B są typu REAL, a co innego, gdy są typu INTEGER.

Zatem, ten sam operator może odnosić się do wielu różnych typów. W praktyce powoduje to konieczność uważnego definiowania operatorów jako funkcji, na przykład, po deklaracjach

```
function „*" (X, Y : in COMPLEX) return COMPLEX;
function „+" (X, Y : in COMPLEX) return COMPLEX;
```

i po zdefiniowaniu tych funkcji oraz — typu COMPLEX, dodawanie i mnożenie są interpretowane dwojako, np.

```
XC, XM, XA, X1, X2: COMPLEX;
YR, YM, YA, Y1, Y2: REAL;
  XC := X1*X2*+XA -- wynik typu COMPLEX
  YR := Y1*Y2+YA -- wynik typu REAL
```

ADA umożliwia przeciążanie operatorów nawet dla jednakowych typów argumentów lecz różnych typów wyników, np.

```
A : REAL;
Y : RATIONAL;
A := 4/5; -- wynik typu REAL
Y := 4/5; -- wynik typu RATIONAL
```

W podobny sposób możliwe jest przeciążenie nazw procedur. Spośród kilku procedur mających tę samą nazwę, przy wywołaniu wybierana jest ta, której argumenty odpowiadają wywoływanym. Należy jednak pamiętać, że wywołanie takiej procedury musi jednoznacznie określać, o którą realizację chodzi. Przykładowo, po deklaracjach

```
procedure NAZWA (V : INTEGER)...
procedure NAZWA (V : NATURAL)...
```

wywołanie

```
NAZWA (7);
```

nie jest poprawne. Natomiast wywołania

```
NAZWA (X => 7);

NAZWA (Y => 7);
```

są poprawne, jeśli zadeklarowano, np.

```
X : INTEGER;
Y : NATURAL;
```

Przeciążanie nazw procedur może dotyczyć bardzo różnych argumentów i typów, np.

```
PUT (I);              -- drukowanie zmiennej I
PUT (I,6);            -- drukowanie 6 cyfr zmiennej I
PUT(„MESSAGE")        -- drukowanie komunikatu
```

Warto zaznaczyć, że przeciążanie nie dotyczy operatorów =i/=.

JEDNOSTKI GENERYCZNE

Parametrami zwykłego podprogramu lub pakietu nie mogą być inne podprogramy ani typy. Język ADA ma jednakże mechanizm umożliwiający parametryzację za pomocą tzw. jednostek generycznych, z zastrzeżeniem, że nie można ich wykonywać bezpośrednio.

W definicji jednostki generycznej należy podać parametry formalne, tzn.

```
generic
  PARAMETER_FORMALNY_1
  PARAMETR_FORMALNY_2
  ...

package NAZWA is -- to samo dotyczy podprogramów
  ..
end;
```

Parametrem generycznym nie może być parametr wyjściowy, **out**, a parametryzowaną jednostką może być oprócz pakietu procedura lub funkcja.

Postać wykonawczą jednostki generycznej stanowi tzw. rozwinięcie (ang. instantiation), w którym podaje się parametry aktualne, np.

```
declare
package NOWA_NAZWA is new NAZWA (PAR_AKT
-1, PAR_AKT_2)';

use NOWA_NAZWA; -- użycie rozwinięcia
```

W tym sensie moduły generyczne są analogiczne do makrorozkazów.

Poniżej podano przykłady jednostek generycznych, których parametrami są podprogramy i typy.

```
generic
  type ITEM is private; -- parametr 1
  type VECTOR is array (INTEGER range <>) of
ITEM; -- parametr 2
  with function SUM (X, Y : ITEM) return ITEM;
  -- parametr 3
package A is
  -- deklaracje

end;
```

Zmiana podprogramu SUM w rozwinięciu, np.

```
with SUM1;
package A1 is new A (REAL, REAL, SUM);
```

umożliwia użycie podprogramu odpowiadającego aktualnym typom.

Rozwinięcie procedury generycznej

```
generic
type TYP
procedure SWAP (X, Y : in out TYP) is

  TEMP : = U;
  U : = V;
  V : = TEMP;
end SWAP;
```

dla zmiennej typu VECTOR ma postać

```
procedure SWAP_VECT is new SWAP (VECTOR);
```

a użycie —

```
SWAP (A, B);
```

Przedstawione przykłady ilustrują cel stosowania jednostek generycznych, tj. stworzenie możliwości definiowania podprogramów dla różnych typów. Polega to na zdefiniowaniu podprogramu bibliotecznego dla nieokreślonych typów oraz przesunięciu wiązania typów na czas translacji programu użytkowego. Dzięki temu mechanizmowi dla różnych typów można używać funkcjonalnie tych samych jednostek programowych.

Kończąc omawianie konstrukcji językowych należy wspomnieć o innych ważnych cechach języka, które umożliwiają oddzielną kompilację modułów, realizację zależną od konfiguracji sprzętu, obsługę wejścia-wyjścia oraz sterowanie pracą kompilatora.

Klauzula **with** powoduje dołączenie wymienionych po niej, oddzielnie skomplikowanych programów bibliotecznych, co umożliwia używanie ich w danej jednostce programowej. Wtedy można zadeklarować rozwinięcia jednostek generycznych, a niegeneryczne podprogramy lub pakiety mogą być, odpowiednio, wywołane lub zadeklarowane przy użyciu klauzuli **use**, np.

```
with MATHLIB;
procedure NAZWA (...) is
  use MATHLIB;
begin
..
end;
```

Klauzula **use** (deklaracja) powoduje uzyskanie widoczności pewnych szczegółów pakietu bibliotecznego.

Specyfikacja i ciało pakietu stanowią także oddzielne jednostki kompilacji, a używające danego pakietu jednostki programowe są zależne nie od ciała a tylko od specyfikacji. Zgodność sprzęgów jednostek programowych, tzn. właściwą specyfikację powinien zapewniać kompilator.

Należy wspomnieć także o właściwościach języka umożliwiających realizację zależną od sprzętu, np. specyfikację rekordów i tablic (ogólnie — typów), dostęp do pojedynczych komórek pamięci, a także dołączanie programów napisanych w innym języku, np. w języku symbolicznym.

Trzy spośród modułów bibliotecznych zdefiniowanych systemowo dotyczą różnych rodzajów operacji wejścia-wyjścia. Są to: pakiet generyczny INPUT_OUTPUT oraz pakiety TEXT_IO i LOW_LEVEL_IO. Pakiet TEXT_IO jest używany do znakowych operacji wejścia-wyjścia łącznie z formatowaniem danych, LOW_LEVEL_IO — do komunikacji z urządzeniami specjalnymi, a INPUT_OUTPUT — do operowania plikami. Pakiety te nie są włączone do definicji języka.

Dyrektywy w języku ADA określone są terminem pragma. Istnieje 12 dyrektyw, które mogą sterować procesem kompilacji lecz ich użycie nie jest wskazane i może prowadzić do nieporozumień, np. w przypadku dyrektywy SUPPRESS anulującej obsługę wypadków.

* * *

Przedstawiając krótki opis podstawowych konstrukcji języka ADA nie zamierzaliśmy opisywać go w sposób wyczerpujący. Dlatego, niech Czytelnicy zechcą wybaczyć lakoniczność i niedoskonałość tego opisu. Podstawowym celem było zachęcenie informatyków do zainteresowania się ADĄ i — sięgnięcia do innych źródeł.

Obecnie istnieje ok. 450 pozycji bibliograficznych, z których można polecić kilka, na poziomie podstawowym [2, 3, 4] lub — średnim [1,6]. Istnieje również jedno, łatwo dostępne opracowanie w języku polskim [5]. Aktualną bibliografię języka ADA można otrzymać od autora artykułu (IBJ, ul. Dorodna 16, 03-195 Warszawa).

W następnym numerze INFORMATYKI spróbujemy przedstawić przykładowy program stanowiący oprogramowanie urządzenia sterowanego przy użyciu bloków zestawu CAMAC, a w jednym z następnych — krótki test stanowiący nie tyle rodzaj sprawdzianu ze znajomości języka, ile dalszą ilustrację jego właściwości.

Za ewentualne nieścisłości i błędy w przedstawionym opisie ponosi winę sam autor, który będzie wdzięczny Czytelnikom, jeśli zwrócą mu na nie uwagę.

LITERATURA

[1] Barnes J.G.P.: An Overview of Ada, Software — Practice and Experience, Vol. 10, p. 851, 1980

[2] Booch G.: Ada Promotes Software Reliability with PASCAL-like Simplicity, EDN, Vol. 26, No. 1, p. 171, 7 January 1981

[3] Johnson R.C.: Special Report. Ada, the Ultimate Language?, Electronics, Vol. 54, No. 3, p. 127, 10 February 1981

[4] Loveman D.: Subprograms and Types Boost Ada Versatility, Electronic Design, Vol. 28, No. 22, p. 152, 1980

[5] Müldner T.: Pewne uwagi o nowych językach programowania wysokiego poziomu: LOGLAN i Ada, Biuletyn Techniczny MERA, nr 11, str. 23, nr 12, str. 22, 1980

[6] Wegner P.: Programming with Ada: An Introduction by Means of Graduated Examples, SIGPLAN Notices, Vol, 14, No 2, p. 1, 1979.

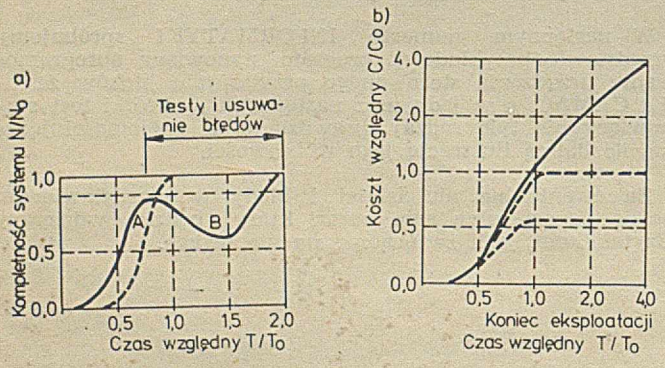
ANDRZEJ STOKALSKI
Wojskowa Akademia Techniczna
Warszawa

PROJEKTOWANIE I BUDOWA SYSTEMÓW INFORMATYCZNYCH

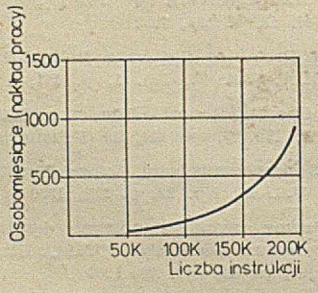
Organizacja cyklu rozwojowego

Znane są problemy związane z wprowadzaniem do eksploatacji dużych i unikalnych systemów informatycznych. Terminy zakończenia prac projektowo-produkcyjnych są wielokrotnie odkładane, koszty produkcji wielokrotnie przekraczają planowany budżet, a jakość systemu po wdrożeniu okazuje się nieoczekiwanie niska na skutek małej wartości użytkowej, nadmiernych kosztów eksploatacji i dużej zawodności. Wymiana kierownika projektu, projektantów czy programistów nie jest w stanie rozwiązaś problemu — może najwyżej zmienić nieco jego skalę.

Główna przyczyna wymienionych zjawisk leży poza sferą personalną; ich źródłem jest nieadekwatna struktura cyklu rozwojowego oraz brak efektywnych metod i technik projektowania. Sytuację tę dobrze charakteryzują profile kompletności i kosztów projektu, pokazane na rys. 1. Za miarę kompletności przyjmuje się tu tę część oprogramowania, którą zgodnie z przyjętym kryterium akceptacji można uznać za uruchomioną i poprawną. Z danych statystycznych [2, 5, 10] wynika, że z wielkością projektu maleje wydajność prac projektowo-produkcyjnych (rys. 2) i odpowiednio rosną jednostkowe koszty produkcji oprogramowania w przeliczeniu na jeden rozkaz uruchomionego i wytestowanego programu. Ponadto nadmierne wydłużenie cyklu rozwojowego (nawet do 10 lat) grozi dezaktualizacją systemu informatycznego.



Rys. 1. Typowe dane systemów informatycznych, profile kompletności (a) i kosztów (b) [10]



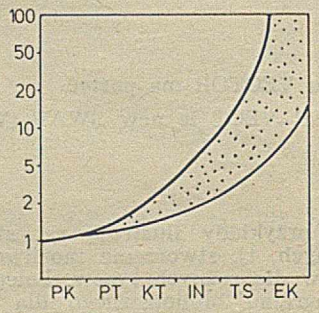
Rys. 2. Nakłady pracy rosną wykładniczo z wielkością projektu (projekty Europejskiej Agencji ds. Badań Kosmosu — ESA) [5]

Przebieg krzywej nakładów pracy (rys. 2) uzasadnia tezę, iż przy określonej technologii dla każdego zespołu projektowo-produkcyjnego istnieje pewna krytyczna wielkość projektów, którym można zagwarantować cykl rozwojowy o sensownej długości (np. 4—6 lat). Projekty przekraczające wielkość krytyczną są praktycznie niemożliwe do realizacji.

BŁĘDY PROJEKTOWE

Omówione zjawiska powstają głównie wskutek błędów popełnianych w różnych fazach cyklu rozwojowego, braku nadzoru kierownika nad przebiegiem prac projektowo-produkcyjnych oraz mało skutecznej kontroli jakości produktu programowego. Szczególnie szkodliwe jest rozprzestrzenianie się błędów na kolejne fazy cyklu rozwojowego (rys. 3). Toteż współczesna inżynieria systemów informatycznych opiera się na trzech podstawowych zasadach:

- technologia projektowania i produkcji oprogramowania powinna zapobiegać powstawaniu błędów
- popełnione błędy nie powinny przenikać do kolejnych faz cyklu życia systemu
- kompletność projektu i koszty powinny być kontrolowane we wszystkich fazach cyklu jego życia.



Rys. 3. Względny koszt wykrywania, lokalizacji i usuwania błędów w różnych fazach cyklu życia systemu [2]

Błędy w projektach systemów informatycznych można podzielić na charakterystyczne grupy. Niektóre firmy informatyczne kontrolują ponad 400 typów błędów występujących tylko w fazach projektowania programów i kodowania. Szczególnym rodzajem błędów są tzw. b ł ę d y s t r u k t u r a l n e (tabela), nie związane jawnie ze sposobem implementacji projektu. Błędy strukturalne w większości przypadków mogą być wykrywane przy użyciu analitycznych przekształceń sformalizowanego modelu projektowanego oprogramowania [3, 11].

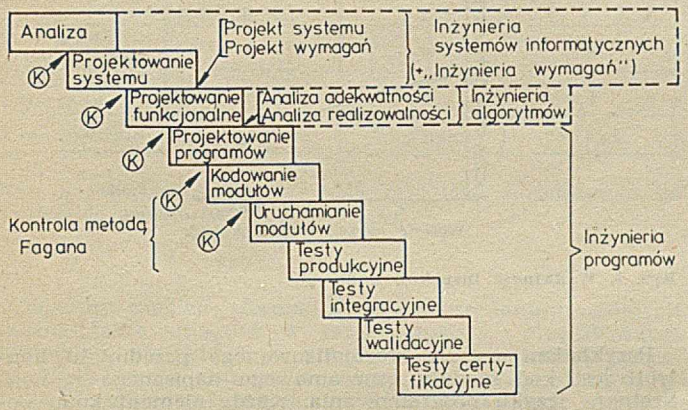
CYKL ROZWOJOWY

Wprowadzenie niezbędnej dyscypliny projektowej wymaga odpowiednich przedsięwzięć na dwóch poziomach: ogólnokoncepcyjnym i warsztatowym. Poziom ogólnokoncepcyjny zawiera podział cyklu rozwojowego na fazy oraz określenie specjalizacji inżynierskich wymaganych w tym cyklu i postaci produktów międzyfazowych. Poziom warsztatowy dotyczy „wnętrza": organizacji zespołów, metod, technik, komletnych technologii i narzędzi. Podział cyklu rozwojowego na fazy powinien umożliwiać

specjalizację typu technologicznego, związaną z jednoznacznie określonym „warsztatem" inżynierskim [6], oraz zapewniać wytworzenie w każdej fazie kompletnie udokumentowanego produktu, w pełni określającego zadania dla fazy następnej.

Warunkiem zakończenia fazy jest zaakceptowanie końcowej dokumentacji zgodnie z procedurą międzyfazowej kontroli jakości. Błędy w ocenie potrzeb użytkownika i nieadekwatność proponowanych rozwiązań są praktycznie nie do wykrycia w fazach produkcyjnych. Dlatego dokumentacja powstająca w fazie analizy i dokumentacja projektowa systemu powinny być czytelne dla użytkownika we wszystkich fazach poprzedzających projektowanie i produkcję programów — w takim stopniu, aby umożliwić jego współuczestnictwo przynajmniej w kontrolach międzyfazowych.

Strukturę cyklu rozwojowego odpowiadającą tym postulatom pokazano na rys. 4.



Rys. 4. Struktura cyklu rozwojowego (K — kontrole międzyfazowe)

Wyodrębniono projektowanie funkcjonalne, którego celem jest wytworzenie kompletnych specyfikacji dla programów wymagających zastosowania nietrywialnych modeli matematycznych i algorytmów oraz weryfikacja ich adekwatności i technologiczności. Projektowanie techniczne (szczegółowe) rozdzielono na fazy: projektowania programów, kodowania, uruchomiania, testów produkcyjnych, integracji, testów walidacyjnych programów i testów certyfikacyjnych systemu.

Faza projektowania funkcjonalnego jest istotna jedynie w przypadku, gdy problemy związane z algorytmizacją procesów przetwarzania wykraczają poza ramy rutynowych, profesjonalnych umiejętności projektanta systemu lub programisty wiodącego i zachodzi konieczność udziału w projekcie odpowiedniego specjalisty, np. z dziedziny badań operacyjnych czy automatyzacji procesów technologicznych. Zauważmy, że nawet elementarny, „klasyczny" algorytm odwracania macierzy traci dla programisty wartość użytkową w przypadku, gdy macierz nie mieści się w pamięci operacyjnej.

Wydzielenie fazy projektowania funkcjonalnego ułatwia profesjonalizację takich czynności, jak: konstruowanie matematycznych modeli i złożonych algorytmów, kontrola ich adekwatności i zdolności do realizacji komputerowej oraz przekształcenie modeli i algorytmów z „czystej" postaci matematycznej w postać technologicznych specyfikacji programów (specyfikacje te definiują zadania dla programistów).

W efekcie możliwe jest skuteczne zapobieganie dość licznej klasie błędów projektowych związanych z wymaganiami nieadekwatnymi lub niemożliwymi do spełnienia, które prowadzą do kosztownych modyfikacji projektu w końcówce cyklu rozwojowego, ewentualnie do dramatycznej rozbudowy zestawu komputerowego, na którym system będzie eksploatowany.

Dodatkową korzyścią z wydzielenia omawianej fazy jest precyzyjne określenie miejsca i roli specjalistów zajmujących się modelowaniem, co zapobiega zdominowaniu projektu przez tę problematykę. Nie umniejszając w niczym roli aparatury matematycznej w budowaniu współczesnych systemów informatycznych, zwłaszcza służących potrzebom kierowania, należy uwzględniać następujące zasady:

- Podjęcie implementacji modeli (algorytmów) niemożliwych do zrealizowania przy obowiązujących ograniczeniach techniczno-technologicznych prowadzi w efekcie albo do ich nieadekwatności na skutek wprowadzania modyfikacji upraszczających, albo do katastrofalnego wydłużenia terminów realizacji i wzrostu kosztów projektu.
- Specyfikacje zawierające wielowejściowe, tasiemcowe schematy logiczne, wielopiętrowe indeksy we wzorach matematycznych, itp. „barok", z zapałem uprawiany w niektórych środowiskach informatycznych — praktycznie uniemożliwiają wytworzenie bezbłędnego oprogramowania.
- Czym istotniejsza jest rola aparatury matematycznej w projekcie (modele, algorytmy), tym silniejsza jest tendencja do ignorowania warstwy technologicznej projektu, która jednak ma decydujący wpływ na końcowy sukces lub upadek całego przedsięwzięcia.

Wyodrębnienie projektowania programów ma na celu stworzenie warunków do sformalizowanej i całościowej kontroli poprawności projektu oraz oceny jakości i technologiczności przyszłego produktu przed rozpoczęciem produkcji. Można dzięki temu zapobiegać marnowaniu nakładów pracy i kosztów ponoszonych w wyniku rozpoczynania produkcji według projektów niekompletnych lub zawierających błędy. W efekcie łatwiejsze jest też utrzymywanie dyscypliny projektowej, wyrażającej się przez: rygorystyczne przestrzeganie standardów konstrukcyjnych i technologicznych, ilościowe uzasadnianie decyzji projektowych (np. na drodze oszacowań podstawowych parametrów przebiegu programów) i kontrolę wydajności pracy programistów.

Traktowanie projektowania programów jako autonomicznej fazy cyklu rozwojowego jest naturalnym zabiegiem organizacyjnym w przypadku projektowania modularnego, szczególnie w zespole programisty wiodącego (ang. Chief Programmer Team). Projekt pojedynczego programu powinien być działem jednego, a co najwyżej dwóch ludzi (programista wiodący i jego asystent). W fazach produkcyjnych (kodowanie, uruchomienia, testy) kodyści pracują pod ciągłym nadzorem merytorycznym autora projektu; jeden projektant nie zawsze może zagwarantować ciągłość nadzoru (choroby, narady, rozmowy z użytkownikiem, konferencje itp.). Ponadto w dwuosobowym zespole łatwiej jest konfrontować alternatywne rozwiązania i zapewnić ewentualną współbieżną realizację pewnych fragmentów projektu.

Produkcję programu zaprojektowanego w sposób modularny można realizować w zespole wieloosobowym. Współbieżne kodowanie, uruchomianie i testowanie wielu modułów skraca czas wyprodukowania całego programu.

Oddzielenia fazy projektowania modularnego od faz produkcyjnych ułatwia minimalizację kosztów osobowych przez wykorzystywanie specjalistów o kwalifikacjach adekwatnych do zadań. Wysokie kwalifikacje zawodowe muszą w zasadzie mieć jedynie projektanci programu (programista wiodący i jego asystent). Czym precyzyjniej specyfikacje definiują zadania dla kodystów (moduły), tym niższe mogą być kwalifikacje tej kategorii pracowników.

Rozdzielenie faz kodowania i uruchomiania umożliwa wprowadzenie sformalizowanej kontroli poprawności kodu modułów przed ich skierowaniem do kompilacji. Do przebiegów kompilacyjnych powinny przechodzić jedynie moduły nie zawierające już błędów gramatycznych. Jest to w praktyce możliwe. Na przykład w technologii opisanej [12] uzyskano średnio 1,2 przebiegów kompilacyjnych na moduł. Skuteczna kontrola poprawności kodów pozwala zminimalizować obciążenie komputera produkcyjnego i zredukować związane z tym koszty produkcji.

Testy produkcyjne powinny zagwarantować poprawność modułów przed przystąpieniem do składania (integracji) programu. Jest zasadnicza różnica między testowaniem indywidualnym modułów, a testowaniem programu złożonego z modułów niewytestowanych (chociaż uruchomionych); koszt przebiegów testowych może być w pierwszym przypadku znacznie niższy. Tylko w przypadku konsekwentnego przestrzegania zasady indywidualnego testowania modułów można na podstawie modułów wyprodukо-

wanych (kompletnych) oceniać stopień kompletności projektu oraz prognozować nakłady pracy i koszty potrzebne na dokończenie realizacji projektu.

Wydzielając fazy testów uzyskuje się ponadto możliwość dodatkowej specjalizacji technologicznej w ramach zespołu, ukierunkowanej na planowanie i realizację testów produkcyjnych modułów, testów integracyjnych, sprawdzających poprawność współdziałania modułów oraz testów walidacyjnych, sprawdzających zgodność programu z wymaganiami. W przedsiębiorstwach pracujących tradycyjnie testowanie z reguły nie jest objęte standaryzacją ani co do stopnia wytestowania, ani w zakresie technologicznym (jak testować). Stopień wytestowania programu wychodzącego z produkcji jest sprawą etyki zawodowej projektanta, a nie normą techniczną.

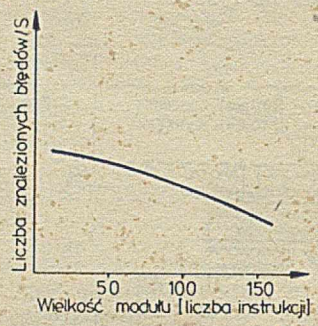
Plan integracji programu z modułów technologicznych i związanych z tym testów ma istotny wpływ na koszty produkcji. Ściśle też wiąże się z harmonogramem produkcji modułów, ponieważ obie te fazy mogą się do pewnego stopnia nakładać (jeżeli nie dotyczą tych samych modułów). Jest to więc ukryta rezerwa, umożliwiająca niekiedy skrócenie cyklu rozwojowego. Dlatego plan integracji należy opracować w fazie projektowania programów. Powyższe uwagi dotyczą także integracji systemu z poszczególnych programów; jej koncepcja powinna być opracowana w fazie projektowania systemowego.

Pozornie drugorzędnym, ale praktycznie bardzo istotnym skutkiem dobrego rozbicia cyklu rozwojowego na fazy jest możliwość precyzyjnego zdefiniowania specjalizacji inżynierskich, pokrywających spójnie cały cykl życia (ang. *life cycle*) projektu. Profile i kryteria kwalifikacyjne, adekwatne do faktycznych potrzeb technologicznych procesu projektowo-produkcyjnego, realizowanego w ramach wyodrębnionych specjalizacji (inżynierii), mogą się stać podstawą organizacji i wyposażenia inżynierskiego warsztatu każdej fazy oraz podstawą kształtowania właściwych programów kształcenia w uczelniach odpowiednich specjalistów.

KONTROLE MIĘDZYFAZOWE

Efektywność podziału cyklu rozwojowego na fazy w znacznym stopniu zależy od skuteczności przeciwdziałania rozprzestrzenianiu się błędów na kolejne fazy. Obok technologii wewnątrzfazowych, które powinny być [6] ukierunkowane na przeciwdziałanie błędnym decyzjom projektowym i błędom ściśle dokumentacyjnym, (takim jak: jednoznaczność, niekompletność itp.), zasadniczą rolę odgrywają międzyfazowe kontrole projektu.

Zależnie od zawartości informacyjnej dokumentacji i sposobu jej prowadzenia, kontrolę międzyfazową można automatyzować w mniejszym lub większym stopniu, ale prawdopodobnie przez długi jeszcze czas zasadniczą rolę będą odgrywać metody oglądowe. Warunkami skuteczności metod oglądowych są: bardzo dobra znajomość typów błędów specyficznych dla fazy, której produkt podlega kontroli; modularność projektu (przedmiot kontroli nie może przekraczać pewnego progu złożoności — por. rys. 5) oraz podporządkowanie przebiegu odpowiednio ukierunkowanym scenariuszom, opartym na statystycznych informacjach o skatalogowanych typowych błędach. Zarówno katalogowanie błędów, jak i scenariusze ich wyszukiwania w projekcie mają sens jedynie pod warunkiem daleko posuniętej formalizacji dokumentacji projektowej i ujęcia jej w ścisłe ramy standardów.



Rys. 5. Wydajność inspekcji kodu

Przykładem wysoce sformalizowanego przedmiotu kontroli jest kod modułu programowego napisanego w konkretnym języku programowania. Każdy element kodu posiada z jednej strony znaczenie w sensie algorytmu opisanego kodem, z drugiej natomiast — ściśle określony sens syntaktyczny jako element języka. Można więc bez trudu opracować wygodną systematykę błędów, np. w układzie: syntaktyczne, algorytmiczne, konstrukcyjne — skatalogować błędy typowe i opracować scenariusz sesji kontrolnej. Przykładowy fragment katalogu błędów tego typu pokazano w [12].

Sam scenariusz sesji przyjmuje formę poleceń i pytań, a ich układ zależy od przedmiotu kontroli. Może być — na przykład — inny dla modułów leżących w rdzeniu i

Błędy strukturalne w projektach oprogramowania systemów informatycznych

Typ błędu	Komentarz
Niekompletność struktury	Niezdefiniowane lub zdefiniowane w nadmiarze: — funkcje przetwarzania — dane używane lub wytwarzane podczas przebiegów
Niespójność struktury	Dane wytwarzane w pewnym kroku procesu przetwarzania wykraczają poza dziedziną określonych funkcji realizowanej na nich w kolejnym kroku
Niewłaściwe użycie obszarów danych	Argumenty funkcji pobierane z niewłaściwych obszarów; wyniki funkcji umieszczane w niewłaściwych obszarach
Niespójność przebiegów	Zatrzymanie przebiegu następuje przed osiągnięciem specyfikacyjnego punktu końcowego
Niekompletność przebiegów	Pewne kroki procesu przetwarzania zostają podczas przebiegu pominięte
Niewłaściwa segmentacja	Nieodpowiednie rozbicie programu i danych na segmenty powoduje nadmierne obciążenie procesora funkcjami wymiany segmentów i wydłużenie czasu przebiegów
Niewłaściwa modularyzacja	Nieodpowiednie rozbicie programu na moduły powoduje nadmierne obciążenie procesora funkcjami komunikacji międzymodułowej lub nadmierne koszty produkcji
Naruszenie integralności procesów	Na skutek przebiegu jednego procesu dochodzi do niedopuszczalnych zmian w obszarach innych procesów
Blokady procesów	Każdy ze współbieżnie przebiegających procesów (zablokowanych) oczekuje na zdarzenie, które mogłoby zajść jedynie w przypadku, gdyby któryś z pozostałych procesów mógł kontynuować przebieg

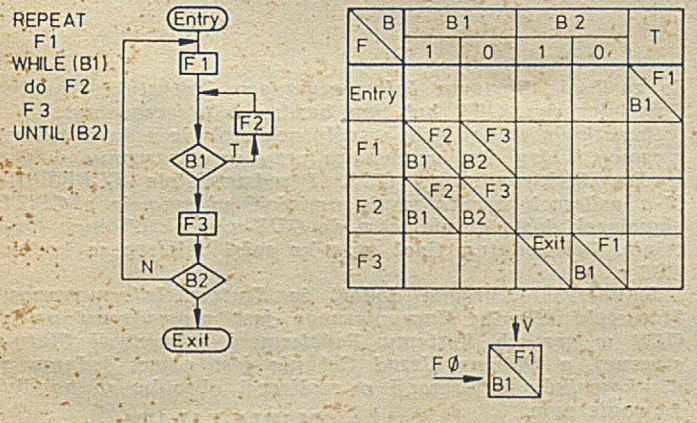
modułów poza rdzeniem przebiegu. W pierwszym przypadku sprawdzenia wymaga także efektywność przebiegu kodu, która w drugim przypadku jest bez znaczenia. Dla modułów rdzeniowych sekwencja przechowywania zawartości akumulatorów wg wersji C z rys. 6 może być uznana za dobrą, a wg wersji A za błędną, podczas gdy w przypadku modułów leżących poza rdzeniem — odwrotnie (jakkolwiek najlepsza jest wersja B). Dysponując dobrymi katalogami błędów i dobrymi scenariuszami sesji kontrolnych można osiągnąć w przypadku oglądowej kontroli kodów skuteczność 80% (pozostaje ok. 20% błędów).

Funkcja / Parametr	Alternatywne konstrukcje		
	A	B	C
Przechowywanie zawartości akumulatorów	STO Ø X STO 1 X+1 LDN Ø 2 LDX 1 'X+2' MOVE Ø 6	STO Ø1 X STO 23 X+2 STO 45 X+4 STO 57 X+8	STO Ø X STO 1 X+1 : : STO 7 X+7
Liczba instrukcji źródłowych	5	4	8
Liczba instrukcji maszynowych	5	8	8
Czas wykonania	~43 μs	~28 μs	~28 μs

Rys. 6. Sekwencje przechowywania zawartości akumulatorów po wejściu do modułu (ODRA 1305)

Znaczny stopień sformalizowania przedmiotu kontroli można także osiągnąć w fazach analizy projektowania systemu i projektowania programów. Formalizację narzuca się wówczas poprzez wprowadzenie odpowiedniego języka. Przykładami takich języków mogą być: notacja Jacksona [9], notacja RSL [1] lub system formularzy PSL [7]. Jeżeli stosowane jest projektowanie modularne, wówczas przedmiotem kontroli projektu są specyfikacje, odpowiednio — modułów lub programów. Skuteczność kontroli będzie zależna zarówno od pomocniczej dokumentacji technologicznej (scenariusze, katalogi błędów), jak i od zawartości dokumentacji projektowej. Przykładowo, formularz definiowania procesu typu PSL/PSA pozwala wygenerować strukturę operacyjną [8, 11] procesu, a na jej podstawie, częściowo przy użyciu metod analitycznych, dokonać kontroli poprawności strukturalnej, obejmującej większość błędów z tabeli odnoszących się do fazy projektowania systemu.

Czynnikiem wpływającym istotnie na efektywność kontroli międzyfazowej jest adekwatność informacji zawartych w dokumentacji projektowej w stosunku do technik kontroli. Na przykład, zapis logiki w języku PASCAL jest bardzo poręczny zarówno przy manualnym, jak i skomputeryzowanym prowadzeniu dokumentacji. Jednakże manualna analiza i chronometraż ścieżek przebiegowych są znacznie wygodniejsze przy użyciu schematów logicznych, natomiast komputerowe generowanie ścieżek przebiegów programu, schematów logicznych czy nawet kodów źródłowych jest najbardziej wydajne przy użyciu tablicowych reprezentacji logiki — tablic decyzyjnych i tablic sterowania przebiegami (rys. 7).



Rys. 7. Różne formy opisu logiki przebiegów

Katalogowanie błędów może wśród programistów napotykać na opory, argumentowane ogromną liczbą tych błędów. Argument ten jest niewątpliwie słuszny, szczególnie w odniesieniu do błędów występujących w fazach objętych inżynierią programów, ale też nie wszystkie błędy możliwe, nawet te najbardziej nieprawdopodobne, wymagają uwzględnienia w scenariuszach kontroli. W praktyce częstość występowania błędów projektowych i kodowych jest bardzo nierównomierna. Można wyróżnić grupę kilkudziesięciu błędów, których wystąpienia obejmują 80—90% wszystkich przypadków (błędy podstawowe). Kontrola oglądowa może się więc w znacznej części sprowadzać do identyfikowania w projekcie lub kodzie modułu konstrukcji odpowiadających katalogowym błędom podstawowym.

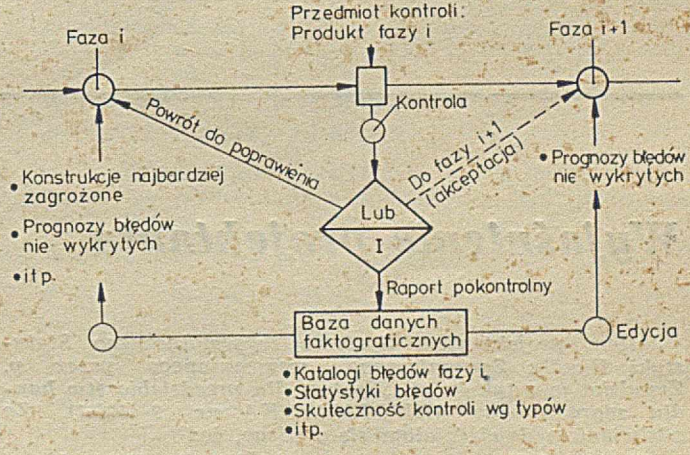
Wykrycie wszystkich błędów podstawowych nie gwarantuje wprawdzie bezwzględnej poprawności, ale niebezpieczeństwo wystąpienia błędu niezauważonego ogranicza się do bardzo niskiego poziomu. Znając skuteczność kontroli oglądowej w odniesieniu do błędów różnego typu, można na błędy, co do których jest mało skuteczna, ukierunkować inne formy kontroli (np. testy).

Dobrze zaprojektowany system kontroli międzyfazowych ma ponadto do spełnienia dwie bardzo istotne funkcje: poznawczą i dydaktyczną.

Funkcja poznawcza polega na systematycznym gromadzeniu danych o sytuacjach projektowych, w których głównie powstają błędy, o rozkładach błędów wg typów, o ich wadze ze względu na obowiązujące kryterium jakości itp. Na podstawie tych informacji można ukierunkować prace badawcze i rozwojowe w dziedzinie technologii projektowo-produkcyjnych oraz w dziedzinie zawodowego szkolenia kadry.

Funkcję dydaktyczną kontroli realizuje się poprzez udział w niej średnio i mniej wykwalifikowanego personelu. Jak wynika z badań opisanych w [12], efekt jest obiecujący.

Z dotychczasowych badań wynika [4, 12], że oglądowej kontroli poprawności projektów można nadać kompleksowy charakter, którego sens ilustruje rys. 7. Kontrole takie winny być prowadzone przez osobę posiadającą specjalne przygotowanie metodyczne. W pewnych przypadkach może się okazać uzasadnione stosowanie dla niektórych członków zespołu kontrolnego odrębnych scenariuszy, ukierunkowanych na szczególne cele, jakie kontrola spełnia w ich przypadku.

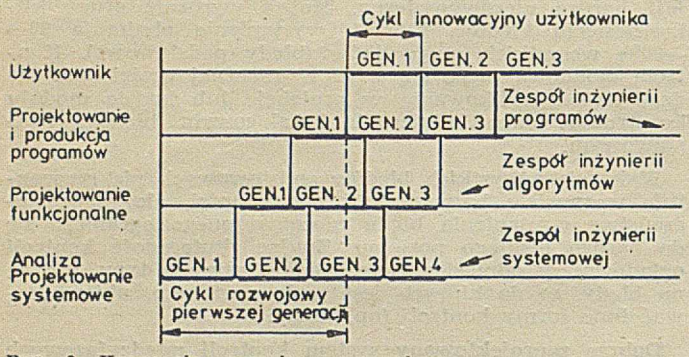


Rys. 8. Obieg i wykorzystanie informacji gromadzonych podczas kontroli międzyfazowych

KONCEPCJA ROZWOJU GENERACYJNEGO

Ideę generacyjnego rozwoju systemu ilustruje rys. 9. W tym przypadku w opracowywaniu znajdują się jednocześnie trzy kolejne generacje systemu, wzajemnie przesunięte o jedną fazę. W każdej fazie działa wyspecjali-

zowany zespół projektowy. Można w ten sposób skrócić długość cyklu innowacyjnego użytkownika do 3—4 lat przy 8—10 letnim cyklu rozwojowym. Rozwój generacyjny wymaga, aby dokumentacja międzyfazowa jednoznacznie i wyczerpująco definiowała nie tylko produkt fazy bieżącej, ale i zadania fazy następnej, w sposób czytelny dla każdego specjalisty spełniającego kryteria kwalifikacyjne odpowiedniej inżynierii. Tak rozumiana autonomiczność dokumentacji międzyfazowej odgrywa dodatkowo bardzo istotną rolę z punktu widzenia swobody decyzji personalnych kierownictwa projektu.



Rys. 9. Koncepcja rozwoju generacyjnego w rezerwowym cyklu rozwojowym

W „klasycznych" technologiach dokumentacja projektowa systemów informatycznych jest z reguły na tyle ogólnikowa, niejednoznaczna i niekompletna, że jedynym nosicielem wiedzy o projekcie jest jego autor. Poza nim nikt nie zna celu ani motywacji większości decyzji projektowych; bez niego nikt nie jest w stanie projektu kontynuować ani tym bardziej modyfikować go w fazie eksploatacji. Przy dużych projektach powstaje w ten sposób mechanizm kreowania „ludzi niezastąpionych", którzy ze swej strony, w miarę upływu czasu, skutecznie wiklają projekt w dodatkowe koszty i wtórne błędy na skutek zapominania drobiazgów związanych z jego funkcjonowaniem (i ewentualnie poprzednimi modyfikacjami), nie ujętych w żadnej oficjalnej dokumentacji. Odpowiednie standardy dokumentacyjne projektów mogą problem ten wyeliminować całkowicie.

* * *

Reasumując, warto zwrócić uwagę na cztery zasady:

- Rozbicie cyklu rozwojowego na fazy służy nie tylko kontroli przebiegu i kosztów prac projektowo-produkcyjnych, ale także ma zapobiec rozprzestrzenianiu błędów i umożliwić efektywną specjalizację technologiczną w ramach faz.
- Znajomość bieżącej kompletności projektu i struktury nakładów ma służyć przede wszystkim kroczącemu planowaniu i doskonaleniu technologii, a nie „optymalizacji" sprawozdań.
- Przerwanie projektu jest zawsze decyzją dramatyczną, która jednak musi być podjęta, jeżeli pewność sukcesu przedsięwzięcia spada poniżej pewnego racjonalnego minimum. W takiej sytuacji czym wcześniej to nastąpi tym lepiej.
- Nawet najlepiej przemyślany system dokumentacji i kontroli międzyfazowych będzie szybko odczuwany jako nadmierna biurokratyzacja, jeżeli nie zostanie „podparty" rzetelnym warsztatem inżynierskim, umożliwiającym efektywną specjalizację technologiczną i zapobiegającym w ten sposób nadmiernemu rozpraszaniu wiedzy, umiejętności i uwagi.

LITERATURA

[1] Alford M. W.: A. Requirements Engineering Methodology for Real-Time Processing Requirements. IEEE Trans. on. Soft. Eng., SE-3, No 1, 1977

[2] Boehm B. W.: Software Reliability: Measurment and Management. Proc. Int. Soft. Mngmnt Conf., London, 1977

[3] Davis C. G., Vick C. R.: The Software Development. System. IEEE Trans. on Soft. Eng., SE-3, January 1977

[4] Fagan M.: Design and Code Inspection. IBM TR 21.572

[5] Fornica G.: Experience of the European Space Agency in Procuring Software. Proc. Int. Soft. Mngmnt Conf., London, 1977

[6] Głąb R., Stokalski A.: Projektowanie unikalnych systemów informatycznych. Materiały INFOGRYF'80

[7] Hershey E. A., Teichroew D.: PSL/PSA — A Computer — Aided Technique for Structured Documentation and Analysis of Information Processing Systems. IEEE Trans. on Soft. Eng., SE-3, 1/1977. Także: Informations System Description Analysis — An Introduction to PSL, ISDOS Working Paper No 86, Dep. of Ind and Oper. Eng., Univ. of Michigan. Ann Arbor, 48104/313/763—3469

[8] Jackson K.: United Kingdom Approaches to Software Procurement. Proc. Int. Soft. Mngmnt Conf., London, 1977

[9] Jackson M.: Principles of Program Desing. APIC Studies in Data Processing, No 12. London-New York Francisco, 1975

[10] de Roze B. C.: The United States Defence Systems Software Menegement program. Proc. Int. Soft. Mngmnt. Conf., London, 1977

[11] Stokalski A.: Zastosowanie teorii struktur operacyjnych w ogólnej inżynierii procesów. Postępy cybernetyki, 3/1981

[12] Stokalski A., Suski Z., Zieliński Z.: Technologie projektowo-produkcyjne zespołu programisty wiodącego. Materiały INFOGRYF'80. Również: MOD — eksperymentalny warsztat inżynierii programów. Myśl wojskowa 4/81 (w druku).

Wyjaśnienie projektanta

Jestem projektantem Systemu Informatycznego Statystyki Handlu Zagranicznego, wspomnianego przez p. Żmudzką w artykule pt. „System dla przedsiębiorstw handlu zagranicznego" (INFORMATYKA nr 9—10/81). Ze zdziwieniem dowiedziałam się z niego, że:

— System Informatyczny Statystyki Handlu Zagranicznego utrzymuje Główny Urząd Statystyczny dla Głównego Urzędu Ceł

— System otrzymuje z Centrum Projektowania i Zastosowań Informatyki informacje przeniesione z faktur FE i FI na taśmach magnetycznych.

Myślę, że sprawa ta powinna zostać wyjaśniona. Czas bowiem najwyższy, aby zamiary nie były przedstawiane jako fakty dokonane. Rzeczywiście, CPiZI prowadzi system dla phz ELEKTRIM i miały miejsce rozmowy z przedstawicielami GUS na temat przystosowania systemu do zasilania Systemu Informatycznego Statystyki Handlu Zagranicznego. GUS był tym bardzo zainteresowany.

Przeszkodą w podjęciu współpracy stał się jednak reżim czasowy, który jest wymagany przy przetwarzaniu w Ośrodku Elektronicznym GUS. Przedstawiciele ELEKTRIM-u usiłowali bezskutecznie umieścić w umowie z CPiZI obowiązek dostarczania taśm do GUS-u w określonym terminie. Ten smutny finał rozmów miał miejsce w końcu 1980 r.

Jednocześnie pragnę dodać, że System Informatyczny Statystyki Handlu Zagranicznego jest prowadzony przez GUS w ramach Systemu Państwowej Informacji Statystycznej, a Główny Urząd Ceł nie jest nawet jednym z użytkowników informacji o realizacji obrotów w handlu zagranicznym.

MARIA SALATA
Ośrodek Elektroniczny GUS

Warszawa, 11.12.1981 r.

ANDRZEJ HUZAR
Centrum Obliczeniowe
Politechnika Wrocławska

Weryfikacja programów — podstawowe pojęcia

Pojęcia związane z formalną poprawnością programów oraz metody weryfikacji programów są mało spopularyzowane w środowisku informatyków. Na ogół tworzy się program według pewnej koncepcji, a potem (po uruchomieniu) następuje jego testowanie. Ujawnia ono zwykle część błędów. Gdy wszystkie testy wypadają zgodnie z oczekiwaniami, program uważa się za poprawny. Jednakże ten sposób tworzenia programów nie zapewnia ich poprawności. Testując programy można bowiem jedynie wykazać istnienie błędów [1]. Zresztą, powszechnie znane są przypadki, że program — pracujący przez dłuższy czas poprawnie — generuje dla pewnego zestawu danych błędne wyniki. Czy to znaczy, że program „zepsuł się"? Nie, program od początku nie był poprawny. Czy można zatem uzyskać poprawny program? Co należy rozumieć pod pojęciem jego poprawności?

Przede wszystkim trzeba zdać sobie sprawę z różnych sposobów interpretacji programu. Stara interpretacja, związana z cechami pierwszych maszyn cyfrowych, to: *program jest ciągiem instrukcji sterujących pracą maszyny.* Interpretacja nowsza, najbardziej zresztą popularna, to: *program to ciąg instrukcji języka programowania.* Semantykę tych instrukcji podaje się w sposób operacyjny, czyli — jakiego rodzaju obliczenia są wykonywane przy realizacji poszczególnych instrukcji i jaka jest kolejność ich wykonywania. Jeśli przy wykorzystaniu jednego z tych sposobów spojrzy się na program, to jedyną metodą sprawdzenia jego poprawności jest testowanie.

Dla przykładu — rozpatrzmy algorytm Euklidesa obliczania największego wspólnego podzielnika dwóch liczb naturalnych *a0, b0.*

```
begin
  input (a0, b0);
  a: = a0; b: = b0;
  while a ≠ b do if a > b then a: = a—b
                          else b: = b—a;
  output (a)
end.
```

Niewątpliwie taki program można łatwo przetestować. Ale okazuje się, że poprawność tego programu można wykazać nie uruchomiając go nawet, czyli — nie potrzeba do tego maszyny. Bowiem program jest poprawny bądź nie, niezależnie od tego czy będzie wykonywany. A oto wpółformalny dowód przedstawionego programu.

Mgr inż. ANDRZEJ HUZAR w 1979 r. ukończył Wydział Podstawowych Problemów Techniki Politechniki Wrocławskiej. Pracuje w Centrum Obliczeniowym Politechniki Wrocławskiej. Zajmuje się metodami weryfikacji programów i możliwością automatyzacji tego procesu. Interesuje się metodologią programowania, programowaniem strukturalnym i nowymi językami programowania: PASCAL, SIMULA, PROLOG.

Trzeba zacząć od przytoczenia własności funkcji *nwp* (x, y), (*nwp* — największy wspólny podzielnik), gdzie x, y — naturalne.

(1) nwp (x, y) = nwp (y, x)
(2) nwp (x, y) = nwp (x—y, y), gdy x > y
(3) nwp (x, x) = x

Dalej dowód przebiega następująco:

Załóżmy, że $a > 0$ i $b > 0$. Jeśli a = b, to pętla while nie jest wykonywana i rzeczywiście wtedy *nwp* $(a0, b0) = a$. Jeśli $a \neq b$, to wykonywana jest jedna z instrukcji: $a: = a—b$ lub $b: = b—a$. Po wykonaniu dowolnej z tych instrukcji, z uwagi na własność (2), wartość funkcji *nwp* (a, b) nie ulega zmianie. Podczas wykonywania pętli prawdziwe są więc następujące wartości: (+) nwp (a0, b0) = nwp (a, b)
(++) a > 0 i b > 0.

Pętla while kończy się, gdy $a = b$. Taka sytuacja na pewno nastąpi; rozpatrzmy bowiem funkcję

f (a, b) = a+b.

Otóż, ze względu na własność (++), zachodzi podczas wykonywania pętli (+++) f (a, b) > 0.
Ale przy wykonywaniu jednej z instrukcji podstawienia w pętli while, $f(a, b)$ zmniejsza swą wartość przynajmniej o 1. Stąd wartość $f(a, b)$ może zmniejszyć się skończoną liczbę razy, czyli iteracja jest zawsze skończona. Po zakończeniu wykonywania instrukcji while, zachodzi $a = b$, co z uwagi na (3) implikuje równość

a = nwp (a, b) = nwp (a0, b0).

Niewątpliwie łatwo zauważyć, że dowodzenie podobnych programów byłoby przejrzyste, gdyby można ten proces ująć w bardziej standardowe ramy. Dlatego też potrzebne są odpowiednie definicje.

DEFINICJE

Specyfikacją wejściową programu $\Phi(\bar{x}_o)$ nazywamy predykat określony na początkowym wektorze stanu programu (ciąg wszystkich zmiennych programu to wektor stanu).

W przedstawionym programie

Φ (a0, b0, a, b) = (a0 > 0 i b0 > 0).

Specyfikacją wyjściową programu $\Psi(\bar{x}_o, \bar{x}_k)$ nazywamy predykat na początkowym i końcowym wektorze programu.

W przykładzie

Ψ (a0, b0, a, b) = a = nwp (a0, b0).

Dopiero mając definicje tych pojęć można mówić o poprawności programu.

Program jest częściowo poprawny względem swych specyfikacji wejścia $\Phi(\bar{x}_o)$ i wyjścia $\Psi(\bar{x}_o, \bar{x}_k)$, jeśli zainicjowanie obliczeń dla dowolnego $\bar{x}_o$, spełniającego $\Phi(\bar{x}_o)$, i zakończenie obliczeń — implikują spełnienie specyfikacji końcowej $\Psi(\bar{x}_o, \bar{x}_k)$.

Program jest skończony względem $\Phi(\bar{x}_o)$, jeśli zainicjowanie obliczeń dla $\bar{x}$ spełniającego $\Phi(\bar{x}_o)$ implikuje zakończenie wykonywania programu.

Program jest całkowicie poprawny względem swych specyfikacji $\Phi(\bar{x}_o)$ i $\Psi(\bar{x}_o, \bar{x}_k)$, jeśli jest częściowo poprawny i skończony względem tych specyfikacji.

Zatem nie można mówić o poprawności konkretnego programu, gdy nie zna się jego specyfikacji. Można zauważyć, że proces dowodzenia polega na tym, aby mając specyfikację wejścia, czyli pewien predykat, stwierdzić jak będzie się on zmieniał w trakcie wykonywania kolejnych instrukcji programu. Program będzie poprawny, jeśli predykat będzie implikował specyfikę wyjścia.

Taka właśnie metoda została po raz pierwszy zaproponowana pod koniec lat sześćdziesiątych przez Floyda [2] i Hoare'a [2]. Została nazwana metodą niezmienników. Podobny proces można prowadzić od specyfikacji wyjścia do specyfikacji wejścia. Taką metodę, z kolei, zaprezentowali po raz pierwszy Manna [2] i Morris z Wegbreitem [2]. Nosi ona nazwę celów pośrednich.

Powróćmy teraz do odpowiedzi na pytanie: czym jest program, a ściślej: jak zdefiniować to pojęcie, aby można było mówić o programie formalnie poprawnym. W świetle podanych definicji można uważać program za funkcję przekształcającą predykat w inny predykat. Aby jednak móc tak mówić o programie, trzeba inaczej określić semantykę poszczególnych instrukcji; trzeba stwierdzić, w jaki sposób każda instrukcja języka programowania przekształca dowolny predykat. Takie właśnie podejście prezentują Dijkstra [1] i Wirth [3].

Istotną przeszkodą w takiej definicji semantyki są niektóre znane powszechnie instrukcje, np: instrukcja skoku bezwarunkowego goto. Omijając tę przeszkodę Dijkstra tworzy język posiadający takie instrukcje, których semantyka umożliwia tę definicję. Dla każdej instrukcji języka określa on sposób wyliczania najsłabszego warunku wstępnego, o ile znany jest warunek końcowy.

Dla przykładu — najsłabszym warunkiem wstępnym dla instrukcji przypisania $x := x+1$ i warunku końcowego $x > 0$ jest $x > -1$.
Oznacza to, że jeśli przed wykonaniem tej instrukcji spełniony był warunek $x > -1$, to po jej wykonaniu spełniony jest warunek $x > 0$ oraz że każdy słabszy warunek od $x > -1$ nie zapewnia (po wykonaniu instrukcji) spełnienia warunku $x > 0$.

W podobny sposób można zdefiniować najsilniejszy warunek końcowy, niemniej pojawiają się pewne problemy rachunkowe. Język Dijkstry ma tę własność, że wykonanie programu polega na sekwencyjnym wykonaniu kolejnych instrukcji. Dowodzenie programu polega więc na wyliczaniu najsłabszych warunków wstępnych dla kolejnych instrukcji od końca do początku programu.

Jedyny i poważny problem stanowią pętle. Dowodzenie poprawności pętli — zgodnie z ideami Dijkstry i Wirtha — przebiega w dwóch krokach:

- trzeba znaleźć predykaty niezmiennicze w pętli (niezmiennik pętli); w omawianym przykładzie niezmiennikiem jest $nwp(a0, b0) = nwp(a, b)$ oraz $a > 0$ i $b > 0$
- trzeba znaleźć taką funkcję wektora stanu $f(\overline{x})$, aby $f(\overline{x}) > 0$ oraz wykonanie jednokrotne instrukcji w pętli spowodowało zmniejszenie $f(\overline{x})$ przynajmniej o 1.

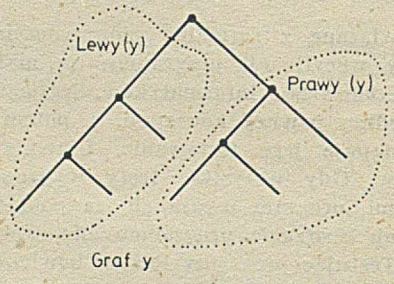
Jeśli funkcja o tych własnościach istnieje — pętla jest skończona.

Po takich krokach wnioskiem z pętli jest koniunkcja zaprzeczenia warunku wykonywania pętli i niezmiennika pętli. Przedstawiona metoda dowodzenia poprawności iteracji jest jedyną ogólną metodą. Nie oznacza to, że niekiedy nie można stosować metody specjalnie skonstruowanej. Tę sytuację ilustruje następny program, obliczający liczbę wierzchołków spójnego drzewa binarnego.

```
  begin
  input (t_o);
L_o: S: = (t_o);
  c: = 0;
L_1 : while S ≠ ( ) do
  begin
  y: = wierzchołek (S);
  S: = pozostałość (S);
  if y jest końcem then c: = c+1
                   else S: = lewy (y) o prawy (y) oS
  end;
L_2: output (c)
  end
```

Występująca w programie zmienna S ma strukturę stosu. Pozostałe oznaczenia w programie:

( ) — stos pusty
c — liczba naturalna, po zakończeniu obliczeń — liczba końców grafu
wierzchołek (S) — element na wierzchu stosu
pozostałość (S) — stos bez wierzchołka
y jest końcem — oznacza, że y jest grafem składającym się z jednego punktu
lewy (y), prawy (y) — wyjaśnia rysunek
o — znak konkatenacji na stosie.



Aby udowodnić, że program jest poprawny, udowodnimy następujące twierdzenie.

Twierdzenie

Jeśli obliczenia znajdują się w L_o, to obliczenia zakończą się i $c =$ *liczbie wierzchołków* t_o w L_2. Liczba wierzchołków t_o będzie oznaczana przez licz (t_o).

Lemat

Jeśli $c = c_o$ i $S = t_o s$ w L_1, to po skończonej liczbie obliczeń $c = c_o + licz\ (t)$ i $S = s$ w L_1.

Dowód twierdzenia (na podstawie lematu)

Załóżmy, że obliczenia znajdują się w L_o. Z tego wynika, że $c = 0$ i $S = t_o = t\,o\,(\ )$ w L_1.
Na podstawie lematu po skończonej liczbie obliczeń
$c = 0 + licz\ (t_o) = licz\ (t_o)$ i $S = (\ )$ w L_1.
Na pdstawie tekstu programu wnioskujemy, że w L_2
$c = licz\ (t_o)$

c.n.d.

Dowód lematu

Dowód będzie indukcyjny ze względu na liczbę końców grafu.

Jeśli $t' =$ *wierzchołek* (S), to t' jest poddrzewem t. Wynika to z przechodniości relacji „być poddrzewem" oraz z tego, że w skład stosu S wchodzą tylko elementy postaci *lewy* (t), *prawy* (t), które są poddrzewami t.

Niech
$c = c_o$ i $S = t_o s$ w L_1
a) Pierwszy krok indukcyjny: sprawdzenie, że lemat jest prawdziwy, gdy t ma jeden wierzchołek, czyli jest końcem.
Po obliczeniach
$c = c_o + 1$ i $S = s$ w L_1.
Oznacza to, że
$c = c_o + licz\ (t_o)$ i $S = s$ w L_1
b) Drugi krok indukcyjny: założenie indukcyjne mówi, że lemat jest prawdziwy dla każdego poddrzewa t (każde właściwe poddrzewo t ma mniej wierzchołków niż t).

Z założenia indukcyjnego wynika, że w następstwie obliczeń $c = c_o + licz\ (lewy\ (t))$ i $S =$ prawy $(t)\ _o s$ w L_1,
a w wyniku dalszych obliczeń
$c = c_o + licz\ (lewy(t)) + licz\ (prawy\ t))$ i $S = s$ w L_1

c.n.d.

Dowód z programem, opisany w [2], jest udaną próbą jednoczesnego dowodzenia skończoności i częściowej poprawności. Aktualnie znane metody weryfikacji programów nie są na tyle efektywne, aby gotowy program był wystarczającym materiałem do dowodu swej poprawności. Można jednak wysnuć następujące wnioski.

- Do dowodzenia poprawności programów napisanych dawniej (często w złych językach programowania) należy

— dla potrzeb weryfikacji — używać dowolnej z metod. Trzeba się jednak liczyć z trudnościami w formułowaniu potrzebnych predykatów. Trudności te mogą uniemożliwić nawet proces dowodzenia.

• Przystępując do budowy nowego programu należy konstruować go w taki sposób, aby możliwy był stosunkowo prosty dowód jego poprawności. Właściwie należy równolegle tworzyć program i jego dowód.

Aktualnie prowadzone są prace, głównie w USA, nad budową systemów automatycznej weryfikacji programów. Dotychczas zbudowano kilka takich systemów, ale mają one głównie charakter doświadczalny.

Temat ten jednak wykracza poza ramy niniejszego artykułu.

LITERATURA

[1] Dijkstra E. W.: Umiejętność programowania. WNT, Warszawa, 1979
[2] Manna Z.: Six lectures on the logic of computer programing. NSP Regional Conference Rensselaer Polytechnic Institute, Troy, NY, 29 May — 2 June 1978
[3] Wirth N.: Wstęp do programowania systematycznego. WNT, Warszawa, 1979.

EMILIA FLADROWSKA
Zakłady Teleelektroniczne
TELKOM TELFA
Bydgoszcz

Czas systemów powtarzalnych

Wykorzystanie systemów informatycznych w zarządzaniu przedsiębiorstwem w największym stopniu określa efektywność pracy komputerów. A jednocześnie — jak wskazują: literatura przedmiotu, konferencje informatyczne i badania prowadzone w przedsiębiorstwach — podstawowa słabość dotychczasowego rozwoju informatyki leży w sferze jej zastosowań.

Bariera praktycznych zastosowań jest głównym problemem informatyki w naszym kraju [1]. Fascynacja techniką komputerową była w początkowym okresie stymulatorem wprowadzania informatyki do zarządzania. Aktualne rozczarowania (często nieuzasadnione) są hamulcem rozwoju zastosowań [5]. Stan taki daje się zaobserwować przede wszystkim w gospodarce. Komputerowe wspomaganie operacyjnego zarządzania produkcją jest bowiem dopiero w stadium eksperymentów [2, 3, 7].

Aktualnie, w okresie konieczności głębokiego zreformowania naszej gospodarki narodowej, sytuacja informatyki jest bardzo niepewna. Pojawiają się głosy, że przedsiębiorstwa informatyczne są *kosztownym dodatkiem wprowadzonym decyzjami technokratycznych woluntarystów po to, aby stanowić pokazowy dowód nowoczesności metod zarządzania. Łatwo można przewidzieć, że obecne głosy zmienią się w ataki. Pytanie: czy informatyka jest potrzebna gospodarce? — może znaleźć odpowiedź, która doprowadzi do zaprzepaszczenia istniejącego już dorobku i uniemożliwi rozwój polskiej informatyki. Spowoduje to fatalne skutki dla ludzi zatrudnionych w informatyce, ale może się okazać także brzemienne dla gospodarki narodowej* [6].

Celem niniejszego artykułu jest częściowe wskazanie przyczyn nieskutecznych zastosowań informatyki w zarządzaniu (co jest powodem ataków na informatykę) oraz metod działania, które potrafią obronić informatykę — dziedzinę praktyki gospodarczej, narzędzie sprawnego i racjonalnego zarządzania. Będzie tu dominował punkt widzenia Zakładów Teleelektronicznych Telkom-Telfa w Bydgoszczy, które od 1969 r. współpracują z ZETO, Zakładów, które skutecznie wdrażają i eksploatują wielodziedzinowy powtarzalny system informatyczny — autorstwa ZETO Wrocław (Świdnica).

W pracy [4] udowodniono pięć poniższych hipotez odnoszących się do skuteczności zastosowań informatyki w zarządzaniu.

• Skuteczne stosowanie sytemu informatycznego w zarządzaniu przedsiębiorstwem zależne jest od metody wyboru powtarzalnego systemu informatycznego, doprowadzającej do takich zastosowań komputera, bez których osiągnięcie założonych celów byłoby bardzo utrudnione bądź niemożliwe.

• Wyniki informatyzacji przedsiębiorstwa zależne są od wypracowania i konsekwentnego przestrzegania podstawowych zasad skutecznego wdrażania i stosowania systemu informatycznego w zarządzaniu.

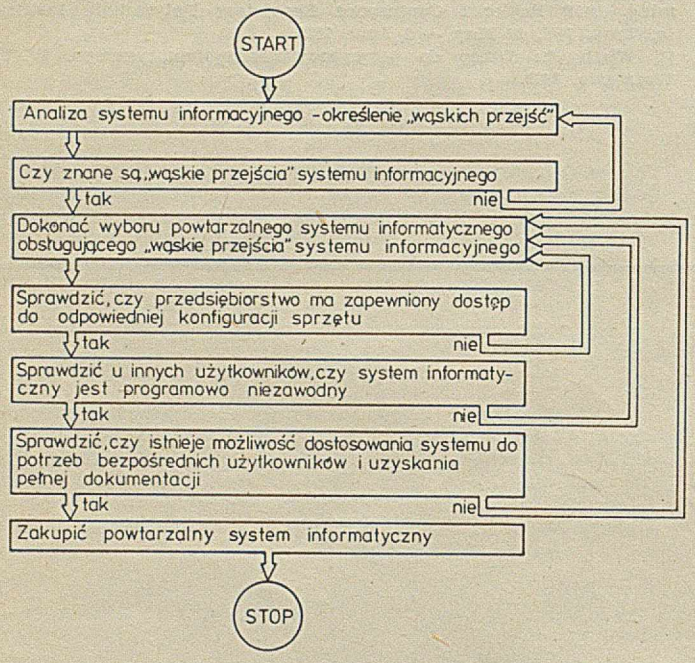
• Warunkiem koniecznym skutecznego stosowania systemu informatycznego w zarządzaniu przedsiębiorstwem jest kompletność i aktualność informacji zawartych w bazie danych systemu (szczególnie dotyczy to podsystemu TPP — Techniczne Przygotowanie Produkcji). Zapewnienie kompletności i aktualności tych informacji wymaga (obok typowych metod kontroli) takiego doprowadzenia wyników do użytkownika, które w efekcie (na zasadzie sprzężenia zwrotnego) „wymusi" aktualność informacji w bazie.

• Skuteczność wdrażania i stosowania systemu informatycznego w zarządzaniu przedsiębiorstwem zależy od metod szkolenia — intensyfikowanych stosownie do potrzeb poszczególnych komórek organizacyjnych, a nawet poszczególnych osób (w szczególności głównych decydentów). Mnogość wariantów szkoleń decyduje o skuteczności systemu informatycznego.

• Skuteczność wdrażania i stosowania systemu stymuluje służba informatyczna przedsiębiorstwa, a zwłaszcza jej kierownictwo. Organizację i funkcje służby informatycznej należy traktować jako czynnik nadrzędny i wiodący w stosunku do grup czynników przedstawionych w czterech poprzednich punktach.

Szerzej zostanie tu przedstawiona jedynie problematyka odnosząca się do pierwszej tezy. W tym zakresie zostaną przedstawione wyniki badań ankietowych, prze-

prowadzonych w 1980 r. w ramach prac Komisji Informatyki Oddziału Wojewódzkiego PTE w Bydgoszczy. Badaniem objęto 35 przedsiębiorstw z całego kraju, współpracujących z Komisją (70% tych przedsiębiorstw to przedsiębiorstwa przemysłu maszynowego, elektrotechnicznego i teletechnicznego).



Rys. 1. Algorytm wyboru powtarzalnego systemu informatycznego (zasady ogólne)

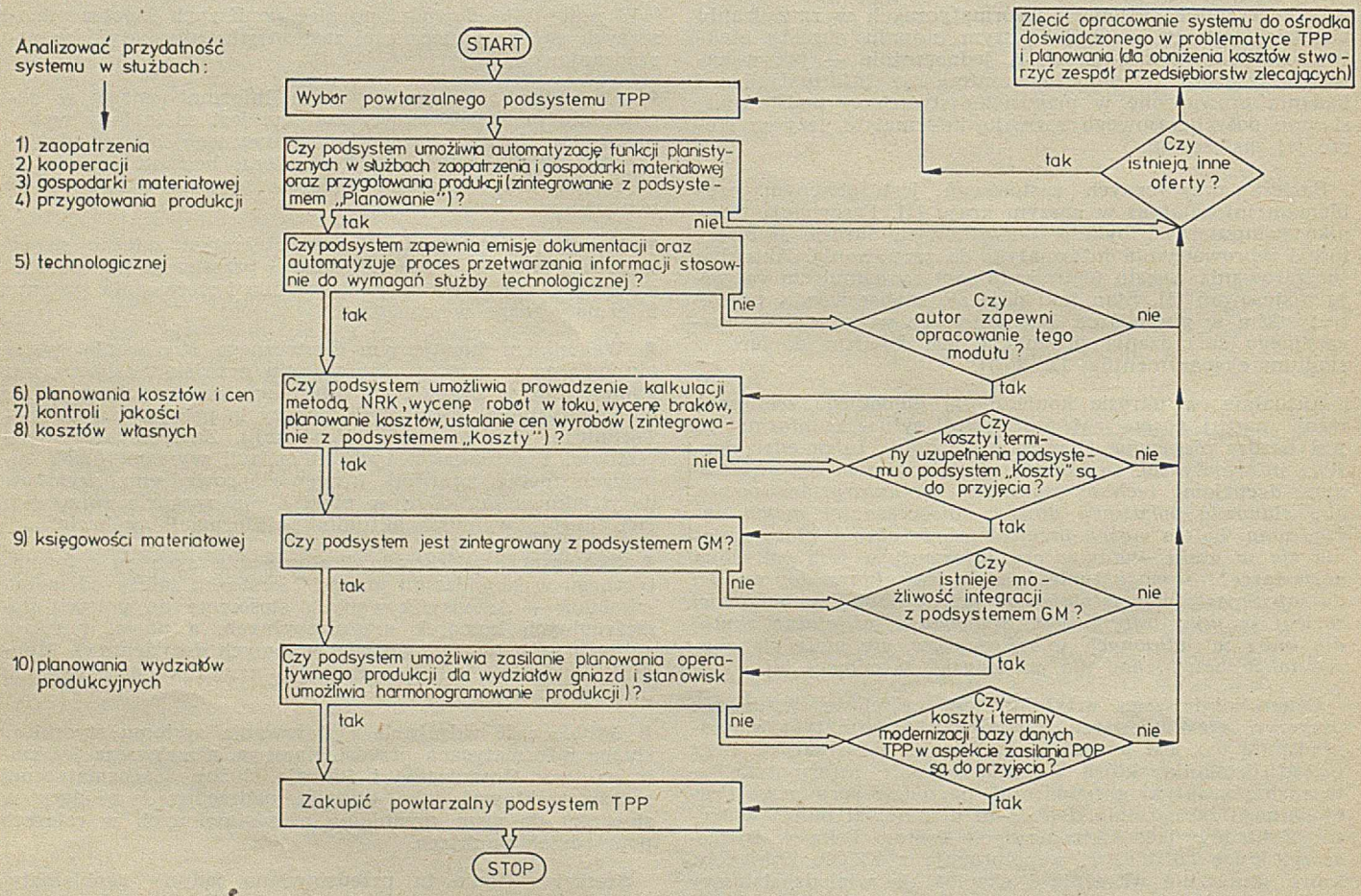
Z przeprowadzonych badań wynika, że dominującymi wdrożonymi podsystemami są:
— podsystem GM (Gospodarka Materiałowa) — 75% przedsiębiorstw
— podsystem „Środki Trwałe" — 55% przedsiębiorstw
— podsystem F-K (Finansowo-Księgowy) i GM-PNU (Gospodarka Materiałowa — Przedmioty Nietrwałe w Użytkowaniu) — 38% przedsiębiorstw
— podsystem „Kadry" — 30% przedsiębiorstw
— podsystem GWG (Gospodarka Wyrobami Gotowymi) — 28% przedsiębiorstw.

Z ogólnych charakterystyk niektórych przedsiębiorstw nie wynika konieczność automatyzowania ewidencji środków trwałych, kadr oraz wyrobów gotowych. Dla przykładu:

- w siedmiu przypadkach komputeryzowano ewidencję środków trwałych przy liczbie kartotek: 400—1000
- w trzech przypadkach komputeryzowano ewidencję kadrową przy stanie zatrudnienia: 400—1000 osób
- w czterech przypadkach kompteryzowano ewidencję wyrobów gotowych przy liczbie kartotek: 120—600, przy czym ewidencja ta służy głównie księgowości.

Przyczyną podejmowania wdrożeń omawianych podsystemów były polecenia odgórne i „aktywne oferowanie" tych systemów przez ZETO. Średnio dwukrotnie przedsiębiorstwa zmieniały wersje wdrażanych podsystemów — wskutek poleceń odgórnych oraz błędów programowych.

Podsystem TPP oraz wynikowe podsystemy zasilające produkcję eksploatuje tylko jedno przedsiębiorstwo. Trzy przedsiębiorstwa (jedno wyłączono z analizy jako nietypowe) realizują od sześciu lat prace wdrożeniowe. Przyczyną tak długotrwałych prac jest w jednym przypadku polecenie władz zwierzchnich zmieniające wersję wdrażanego podsystemu TPP, natomiast w dwóch przypadkach — trudności wewnętrzne. W jednym przypadku, trudności te wynikają z całkowicie samodzielnego realizowania przez



Rys. 2. Schemat metody wyboru powtarzalnego podsystemu TPP (przy zachowaniu zasad przedstawionych na rys. 1)

służbę informatyczną prac projektowo-programowych TPP i Planowania, podczas gdy zatrudnienie tej służby wynosi tylko 11 osób. W drugim przypadku, służba informatyczna wydaje się być zaabsorbowana problemami własnej instalacji komputerowej, co odbija się ujemnie na procesie wdrażania bazy danych podsystemu TPP oraz systemu planowania produkcji.

Generalnie biorąc — w zakresie systemów ewidencyjno-księgowych przeważająca część przedsiębiorstw przestrzegała zasady wdrażania sprawdzonych, niezawodnych opracowań powtarzalnych. Jednakże decyzje o wdrażaniu podsystemów nie zawsze uwzględniały zasadę wdrażania systemu do kluczowych dziedzin działalności, nie wynikały z konieczności usprawnienia systemu przetwarzania informacji w „wąskich przejściach" systemu informacyjnego.

W okresie głębokiej reformy naszej gospodarki, w okresie „walki o samodzielność i samorządność" [6] należy zdecydowanie krytycznie ustosunkować się do dotychczasowej działalności informatycznej, zarówno w zakresie realizowanym przez sieć ZETO, jak i przez użytkowników systemu.

Podstawowy problem: o wyborze systemu [1]) informatycznego decydowały często nie autentyczne potrzeby usprawnienia systemu zarządzania przedsiębiorstwem (obiektem), lecz przypadkowość i wpływ „otoczenia". Stąd nagminne były przykłady wdrażania systemów automatyzujących ewidencję środków trwałych w przedsiębiorstwach, w których tę ewidencję z powodzeniem prowadziła (tradycyjnym systemem) jedna osoba lub wdrażania komputerowej ewidencji pracowników przy stanie załogi nie przekraczającej 1000 osób. Wdrożenia takie absorbowały skromne zespoły informatyków, a znikome efekty doprowadziły do negatywnej atmosfery „informatycznej" w przedsiębiorstwie. Zamiast rozwoju, dochodziło do stagnacji komórek informatycznych.

Podkreślić również należy wadliwość oprogramowania oferowanego przez jednostki autorskie. Doświadczoną przez Dział Informatyki Telfy „plagą" oprogramowania powtarzalnego była jego zawodność. Eliminowanie błędów programowych przysparzało Działowi Informatyki ZT Telkom-Telfa znacznie więcej kłopotów, niż przygotowanie organizacyjne przedsiębiorstwa do wdrażania systemu informatycznego.

Dziesięcioletnie doświadczenia ZT Telkom-Telfa we wdrażaniu i eksploatacji powtarzalnego systemu informatycznego, badania prowadzone w ramach Komisji Informatyki OW PTE w Bydgoszczy oraz współpraca z szeregiem przedsiębiorstw z całego kraju pozwalają na wskazanie pięciu ogólnych, dotychczasowych problemów:

- konieczność wdrażania systemu narzuconego przez jednostki zwierzchnie
- wdrażanie systemu z wadliwym oprogramowaniem
- wdrażanie systemu powtarzalnego bez możliwości bieżącej modyfikacji wyjść i wejść, docelowej modernizacji, rozwoju i wewnętrznej integracji (głównie wskutek nieudostępniania przez ZETO pełnej dokumentacji systemu oraz nieprzygotowywania użytkowników do operowania narzędziami programowymi typu FIND-2 czy JAZ)
- wdrażanie systemu mało efektywnego, nie automatyzującego procesów przetwarzania danych w „wąskich przejściach" systemów informacyjnych przedsiębiorstwa
- znikomy postęp (w ostatnim pięcioleciu) w unowocześnianiu konfiguracji komputerów.

•

Propozycja metody wyboru powtarzalnego systemu informatycznego przedstawiona jest na rysunkach 1 i 2.

• • •

Informatykę należy traktować jako niezbędne narzędzie sprawnego i racjonalnego zarządzania, skutecznie wspierające proces głębokiego zreformowania naszej gospodarki narodowej. *Podstawowym ważnym celem stawianym przed komputerem powinna być pomoc w osiągnięciu określonych efektów ekonomicznych w takich przypadkach, kiedy bez komputera byłoby to nieosiągalne lub nieekonomiczne* [5]. Bezwzględnie należy zrezygnować z wdrażania systemów informatycznych nieefektywnych (mało efektywnych), stanowiących *pokazowy dowód nowoczesności metod zarządzania* [6].

Szczególnie ważnym problemem warunkującym efektywność zastosowań komputerów w zarządzaniu jest rozwój jednolitej linii komputerowej (kompatybilność EMC) — wraz ze sprawnymi narzędziami programowymi — w kierunku przekroczenia „zaklętej" bariery sprzętowej, narosłej w ostatnich latach, uniemożliwiającej realizację sprawnych systemów VIDEO w czasie rzeczywistym.

[1]) Zakłada się wdrażanie systemów powtarzalnych (gotowych opracowań); zarówno teoria, jak i praktyka przesądziła o efektywności tego rodzaju podejścia do wdrażania systemów informatycznych.

LITERATURA

[1] Bachner T., Straszak A.: Obiektowe systemy informatyczne. Kursokonferencja naukowa AMPIG-76, TNOiK, 1976

[2] Drelichowski L.: Systemy planowania operatywnego produkcji w przedsiębiorstwach przemysłowych. Materiały pokonferencyjne, PTE, Bydgoszcz, 1978

[3] Drelichowski L., Kurkiewicz K.: Systemy planowania operacyjnego produkcji w przedsiębiorstwach przemysłowych. Materiały pokonferencyjne, PTE, Bydgoszcz, 1980

[4] Fladrowska E.: Analiza czynników determinujących skuteczność wdrażania i stosowania systemu informatycznego w zarządzaniu przedsiębiorstwem przemysłowym (na przykładzie Zakładów Teleelektronicznych Telkom-Telfa w Bydgoszczy). Rozprawa doktorska. Wydział Zarządzania UW, Warszawa, 1980

[5] Grindley K., Humble J.: Skuteczność wykorzystania komputera, PWE, Warszawa, 1976

[6] Młynarski M.: Walka o samodzielność i samorządność. INFORMATYKA nr 4, 1981

[7] Trybulski J.: Potrzeba realnej oceny możliwości. INFORMATYKA nr 3, 1978

Publikujemy artykuł Zygmunta Ryznara w przekonaniu o celowości prac nad językiem specyfikacyjnym do projektowania strukturalnego. Mamy jednak wątpliwości, czy przedstawiona poniżej propozycja języka S&DL daje szanse praktycznej realizacji. Wraz z Autorem czekamy na listy. (Red.)

ZYGMUNT RYZNAR
ZETO Kraków

S&DL – język specyfikacyjny do projektowania strukturalnego (zarys propozycji)

Aktualnie istnieje kilka tysięcy języków programowania. Tylko kilkadziesiąt z nich znajduje szersze zastosowanie, niemniej stanowią materiał trudny do opanowania ze względu na brak wspólnych (jednolitych) podstaw teoretycznych, niejednolity aparat notacyjny i pojęciowy oraz różnorodność zrealizowanych wersji. Z tych względów nowe języki powinny być wprowadzane tylko wówczas, gdy rozwiązują zadania niewykonywalne przy użyciu języków dotychczasowych.

Wydaje się, że do tej kategorii zaliczyć można klasę języków specyfikacyjnych, ułatwiających użytkownikowi posługiwanie się komputerem. Nie ma bowiem dostatecznie sformalizowanej (a równocześnie — praktycznej) metody określania potrzeb informacyjnych użytkownika, która prowadziłaby możliwie bezpośrednio do użytecznego produktu informatycznego. Gdyby taka metoda istniała [1]), można byłoby szybko i tanio weryfikować te potrzeby oraz modyfikować już funkcjonujące systemy informatyczne. Cóż z tego, że komputery stają się coraz bardziej wydajne, skoro projektowanie i programowanie ma charakter rękodzielniczy, wskutek czego trwa kilka razy dłużej niż życzy sobie tego użytkownik, a system jest tak zaprojektowany, że każda niemal zmiana wymaga długiego czasu i znacznych nakładów finansowych.

Język specyfikacyjny służący projektowaniu strukturalnemu [2]) powinien więc posiadać:

- aparat formalny zdolny do odwzorowania różnorodnych struktur (w tym — hierarchicznych, hierarchizowanych i swobodnych sieciowych) oraz list niestrukturalnych,
- względnie prostą (niematematyczną) notację, pozwalającą na zakładanie komputerowych zbiorów dokumentacyjnych,
- możliwości generowania dokumentacji projektowej, programowej i eksploatacyjnej oraz samych programów,
- komunikatywność dla różnych grup zawodowych uczestniczących w budowie i eksploatacji systemu (użytkowników, analityków, projektantów, programistów i technologów).

Ogólnie znane języki algorytmiczne i symulacyjne nie spełniają powyższych wymagań w stopniu zadowalającym. W związku z powyższym opracowana została propozycja języka S&DL.

NOTACJA I SKŁADNIA JĘZYKA S&DL

Spełniając wymaganie prostoty — niezwykle ważne dla użytkowników i analityków, nie posiadających dostatecznego przygotowania matematycznego — zastosowano minimalną liczbę znaków specjalnych oraz ograniczoną liczbę konstrukcji składniowych, mających charakter ogólny, niezależny od rodzaju opisywanego obiektu. Oznacza to, że można użyć tej samej konwencji do opisu problemu, procesu lub programu, pomimo różnej treści zwrotów.

Wśród znaków specjalnych występują:
: : = wyróżnik zwrotów występujących w definicji składni języka
<> ograniczniki nazw zmiennych i list w definicji składni języka
: : wyróżnik listy strukturalnej
: wyróżnik listy niestrukturalnej (nie wyrażającej relacji uporządkowania, dekompozycji i kompletności)
= wyróżnik wartości
/ „lub" (rozłączne występowanie elementu syntaktycznego)
& „i" (łączne występowanie elementu syntaktycznego)
[] opcja

Typowe konstrukcje zwrotów przedstawione zostały w tabeli 1. Język S&DL składa się z dwóch podzbiorów: języka specyfikacyjnego &SL (Specification Language) i języka konstrukcyjnego DL (Design Language). Język SL służy do opisu obiektów rzeczywistych i abstrakcyjnych (patrz tabela 2: specyfikacja typów obiektów). Charakterystyczna dla obiektu jest możliwość jego identyfikacji (a więc dostępu doń) bez wchodzenia w jego powiązania z innymi obiektami.

Język konstrukcyjny &DL przeznaczony jest do generowania dokumentacji i programów oraz realizacji różnorodnych operacji związanych z przeszukiwaniem katalogów, ich modyfikacją, selektywnym wydrukiem, itp. Ze względu na objętość niniejszej publikacji i pewne analogie do znanych języków manipulacyjnych typu DML (Data Manipulation Language), język ten nie zostanie tutaj omówiony.

Język specyfikacyjny SL może definiować:

- samego siebie, czyli pełni funkcję metajęzyka (tab. 1).
- metastruktur, obowiązujących na poziomie danej implementacji [3]) (np. funktor STRUCTURE określić może następującą strukturę hierarchiczną opisu typu obiektu PROBLEM:

```
DEF PROBLEM
  STRUCTURE : : (PROBLEM(ACTIVITY(PROCESS(ACTION
  (EVENT)))))
  OBLIGATORY-LIST : (lista obowiązkowych zwrotów specyfikacyjnych i atrybutów)
ENDDEF)
```

- systemu, a mianowicie do szczegółowego opisu obiektów, ich powiązań dokumentacyjnych, logicznych i technologicznych.

[1]) Pewien krok w tym kierunku stanowi projekt ISDOS z językiem PSL/PSA

[2]) Istotę projektowania strukturalnego przedstawiłem w poprzednim artykule (INFORMATYKA nr 1/81)

[3]) Dopuszcza się równoczesne stosowanie kilku implementacji (np. każde zastosowanie może mieć własne metastruktury).

ZASADY OPISU SPECYFIKACYJNEGO

Opis specyfikacyjny obiektu odbywa się głównie w bloku definicji (zwrot 3.1.1. w tab. 1) obejmującym nadanie nazwy, identyfikatorów, atrybutów oraz opis struktury. Nazwa pozwala na wyodrębnienie danego obiektu w ramach jego typu, np. ACTION (AKCEPT-ZAM), ACTION (EMISJA-ZLEC). Obiekt może być opatrzony atrybutami standardowymi, wchodzącymi do z góry określonej listy atrybutów języka SL, i atrybutami użytkownika, występującymi za zwrotem ATTRIBUTE. Suma tych atrybutów składa się na opis dokumentacyjny obiektu.

Atrybuty standardowe stosowane są w zależności od typu obiektu. Przykładowo, w stosunku do PROCEDURE używa się atrybutów: ALTERNATIVE/ITERATIVE, RELOCATABLE/FIXED, CONTROL/DRIVER/SUBORDINATED/ /STUB, REENTERABLE, itp. W charakterze atrybutów standardowych wystąpić mogą ponadto dowolne typy obiektów (tab. 2).

Nadanie nazwy odbywa się poprzez umieszczenie jej w nawiasach bezpośrednio po typie obiektu. Wyróżnia się następujące rodzaje nazw:

- prosta (xxxxxx)
- kwalifikowana (xxxx.xxxxx.xxxx.xx), w której ostatni człon jest nazwą obiektu elementarnego, zaś poprzednie dotyczą obiektów hierarchicznie wyższych
- stowarzyszona z synonimem (xxxxxx/xxxxxx), w której na drugim miejscu występuje synonim, przy czym synonim nadawany jest tylko raz (potem można używać i nazwy pierwotnej, i synonimu)
- przemianowana (xxxxxx,xxxxxx), w której wymieniona na pierwszym miejscu pierwotna nazwa przestaje obowiązywać.

Typ opisywanego obiektu określany jest bezpośrednio za starterem definicji DEF. Listę typów obiektów podaje tabela 2. Niektóre typy obiektów wystąpić mogą również w opisie wewnętrznym (wewnątrz bloku definicji), jeśli niezbędne są do charakterystyki obiektu głównego lub niezbędne jest ich wykorzystanie przez procesor języka. Wśród takich typów obiektu może wystąpić — na przykład — starter BEGIN (nazwa wsadu) i wówczas definicja tego obiektu określa strukturę etykiety nadanej przez użytkownika (UHL — User Header Label), która zawiera m.in. hasła, numery uprawnionych terminali lub użytkowników.

Repertuar typów obiektu obejmuje również funktory, służące określeniu struktury, np. PART (nazwa), jeśli występują do nich wielokrotne odwołania w różnych obiektach lub wymagają one dodatkowego opisu (atrybutowego i strukturalnego) przeznaczonego do skatalogowania.

Na uwagę zasługują również tzw. funktory wynikowe, np. STATE, FLOW, INTERSECTION, LIFE-HISTORY, które otrzymywane są w wyniku przetwarzania komend języka & DL i mogą podlegać katalogowaniu. W bloku specyfikacyjnym obiektu występować może ciało obiektu lub odsyłacz do niego. Przykładowo, dla typu obiektu PROCEDURE ciałem jest tekst źródłowy procedury, zaś dla typu SCHEMA ciałem jest schemat bazy danych. Właśnie do ciała obiektu odnoszą się modyfikatory (typ obiektu MODIFIER), za pomocą których można tworzyć wersje tego samego obiektu — zwane inkarnacjami.

Określenie struktury obiektu odbywa się za pomocą:

- wyrażenia PARTS : : (lista), gdzie PARTS jest funktorem wewnętrznym (umieszczanym wewnątrz definicji obiektu)
- wyrażenie PART/PARTS OF (nazwa obiektu) : : (lista), gdzie funktor PART lub PARTS umieszczany jest poza definicją danego obiektu jako deklaracja zewnętrzna

wyrażenie typ — obiektu > OF < typ-obiektu > < (nazwa obiektu) > : : (lista), które jest uszczegółowieniem metastruktury (określonej przez funktor STRUCTURE) na jednym poziomie strukturalnym, np.

EVENTS OF ACTION (nazwa-akcji) : : (lista zdarzeń)

- zagnieżdżenie definicji:

```
DEF1 <typ-obiektu> <(nazwa obiektu)>
 ...
 DEF2 [<typ-obiektu>] <(nazwa)>
 ...
 ENDDEF2
 ...
ENDDEF1
```

Tabela 1. Definicja składni języka S&DL ze szczególnym uwzględnieniem podzbioru SL (poziom metajęzyka)

Symbol zwrotu	Konstrukcja zwrotu
1	<wsad> : : = <wsad-specyfikacyjny-SL>/<wsad-komend-DL>
2	<wsad> : : = <starter> <nazwa-wsadu> <lista zdań> <ogranicznik>
3	<wsad-specyfikacyjny> : : = BEGIN <nazwa-wsadu> <lista-zdań-specyfikacyjnych> END
4	<wsad-komend> : : = START <nazwa-sesji> <lista-komend> STOP
3.1.	<lista-zdań-specyfikacyjnych> : : = <blok-definicji>/ &<deklaracje-zewnętrzne>/&<[ciało obiektu]>
3.1.1.	<blok-definicji> : : = DEF <typ-obiektu> <nazwa-obiektu)> <lista-zwrotów-definiujących> ENDDEF
3.1.1.1.	<lista-zwrotów-definiujących> : : = [<identyfikator-obiektu>] <atrybuty> <opis-struktury>
3.1.1.1.1.	<identyfikator-obiektu> : : = <klucze-dostępu> <[nazwa-pełna]>
3.1.1.1.2.	<opis-struktury> : : = <wyrażenie-funktora-PART>/& <zagnieżdżenie-definicji> /& <zdanie-RELATED> /& <zdanie-INVOLVED [IN]> /& <zdanie-CONTAINED IN> /& <wyrażenie-funktora-OF> <strukturotwórczy-typ-obiektu> <(nazwa)> : : <(lista)>
3.1.1.1.2.1.	<wyrażenie-funktora-PART> : : = PART/PARTS : : <(lista-składników)>
3.1.1.2.2.	<wyrażenie-funktora-PART OF> : : = PART/PARTS OF <(nazwa-obiektu)> : : <(lista-składników)>
3.1.1.1.2.3.	<zagnieżdżenie-definicji> : : = DEF1 <typ-obiektu> <(nazwa-obiektu)> <zwroty> DEF2 <typ-obiektu> <(nazwa-obiektu)> <zwroty> ENDDEF2 DEF3 .... ENDDEF3 ..... ENDDEF1
3.1.1.1.2.4.	<zdanie-RELATED> : : = <(typ-obiektu> RELATED TO <typ-obiektu> <wyrażenie-ON>/ <(typ-obiektu> <(nazwa-obiektu)> RELATED TO <typ-obiektu> <(nazwa-obiektu)> <wyrażenie-ON>/ <(nazwa-obiektu)> RELATED TO <(nazwa obiektu)> <wyrażenie-ON>/ <(nazwa-obiektu)> <klucze-dostępu> = <wartość> RELATED TO <(nazwa-obiektu)> <wyrażenie-ON>
3.1.1.1.2.4.1.	<wyrażenie-ON>/<wyrażenie-ON> : : = <PROCEDURE (nazwa)>/<wyrażenie-logiczne>/<wyrażenie-arytmetyczne>
3.1.1.1.2.5.	<zdanie-INVOLVED> : : = <typ-obiektu> INVOLVED : <lista nazw-obiektu-danego-typu)>
3.1.1.1.2.6.	<zdanie-INVOLVED IN> : : = <typ-obiektu> INVOLVED IN <typ-obiektu> : <(lista-typów-obiektu)>
3.1.1.1.2.7.	<zdanie-CONTAINED IN> : : = <(nazwa obiektu)> CONTAINED IN <(nazwa obiektu)>
3.2.	<deklaracja zewnętrzna> : : = <typ-obiektu> <(nazwa-obiektu)> OF <typ-obiektu> <(nazwa-obiektu)> : : <(lista)>

• zdanie RELATED:
<typ-obiektu RELATED TO <typ-obiektu> ...
(patrz zwrot 3.1.1.1.2.4. w tab. 1)
• zdanie INVOLVED:
<typ-obiektu> INVOLVED : (lista)
INVOLVED IN <typ-obiektu> : (lista)

• zdanie PRECEDENCE OF <typ-obiektu> : (lista).

Wyrażenia określające strukturę spełniają różnorodne zadania. Pierwsze cztery służą do określenia dekompozycji konceptualnej, niezbędnej w procesie poznawczym (np. przy opisie problemu) i stanowiącej podstawę do wygenerowania kwalifikowanych nazw. Za pomocą tych nazw można otrzymać np. opisy powiązanych logicznie obiektów. Funktor RELATED powoduje utworzenie wskaźników adresowych w przypadku wystąpienia warunków podanych w wyrażeniu ON. Jeśli jako typu obiektu występuje w tym zdaniu RECORD wówczas odnosi się to do struktur sieciowych lub liniowych (sieciowe struktury uzyskiwać można poprzez nałożenie wielu liniowych). Jeśli natomiast deklarowana jest relacja pomiędzy OWNER i MEMBER wówczas chodzi o struktury hierarchiczne (lub sieciowe hierarchizowane). Funktor INVOLVED nie pociąga za sobą skutków wykonawczych (element listy nie musi być zdefiniowany jako obiekt, jest więc zwrotem stwierdzającym występowanie danego elementu w środowisku obiektu definiowanego (o nazwie umieszczonej w zwrocie DEF). Funktor INVOLVED IN wskazuje na listę „inwersyjną" (w jakich obiektach innego typu występuje obiekt definiowany). Wreszcie funktor PRECEDENCE OF służy do wyrażenia hierarchii sterowania (nie zaś hierarchii dekompozycyjnej). Znajduje on zastosowanie w szczególności przy opisie programu.

Przedstawione środki tworzą w sumie elastyczny aparat określający struktury obiektów prostych i złożonych.

Przykłady opisów specyfikacyjnych dla wybranych typów obiektów:

```
A. DEF1 PROBLEM (nazwa-problemu)
   FNAME = (pełna-nazwa-problemu)
   CLASS = (kod-klasyfikacyjny-problemu)
   PRIORITY = (priorytet-problemu)
   DEPARTMENTS INVOLVED : (lista-jednostek-organizacyjnych)
   ANALYSTS INVOLVED : (lista-analityków-problemu)
   USERS INVOLVED : (lista-użytkowników)
   ALGORITHMS INVOLVED : (lista algorytmów)
   ATTRIBUTES : (lista-atrybutów-definiowanych-przez-
   -użytkownika)
   <atrybut> = ...
   DECISIONS : (lista-strukturalna)
   OBJECTIVES : (lista-strukturalna)
   NOTE = LISTA STRUKTURALNA MOŻE BYĆ ZBUDOWANA
     HIERARCHICZNIE WG UKŁADU : (cel 1 (podcel 1.1., podcel
     1.2.), cel 2 (. .)).
   ACTIVITIES : : (aaa1, aaa2, . . . , aaan) (lista obszarów działań)
   DEF2 ACTIVITY (aaa1)
     . . . opis atrybutowy
     PARTS : : (bbb1, bbb2, . . . bbbn) (dodatkowy podział nie
     uwidoczniony w metastrukturze)
     DEF3 (bbb1)
     . . .
     ENDDEF2
     PROCESSES OF (bbb1) : : (ccc1, ccc2, . . . , cccn)
     ACTIONS OF (ccc1) : : (ddd1, ddd2, . . . , dddn)
     EVENTS OF (eee1, eee2, . . . , eeen)
   ENDDEF1

B. DEF PROCESS (nazwa-procesu)
   . . .
   OBJECTIVES : : (lista-celów)
```

Tabela 2. Specyfikacja typów obiektów języka SL

ACTION	akcja (element strukturalny procesu)
ALGORITHM	algorytm (obiekt używany na poziomie opisu problemu)
ACTIVITY	działalność/działanie (obszar problemu)/(zestaw procesów)
BEGIN	etykieta wsadu (czołowa)
BINDER	zestaw różnorodnych obiektów
CLUSTER	zestaw obiektów należących do tego samego typu
COMPUTER	komputer (charakterystyka techniczno-eksploatacyjna)
DBASE	baza danych
DBMS	system zarządzania bazą danych
DECISION	decyzja (obiekt używany na poziomie opisu problemu)
DEPARTMENT	jednostka organizacyjna
DESIGNER	projektant (dorobek jego działalności)
DOCUMENT	dokument, w którym dokonuje się ewidencji zdarzenia
DOCUMENTATION	dokumentacja systemu
ECODE	event code — rozpoznawczy kod zdarzenia
END	końcówka etykieta wsadu
ENTITY	obiekt wprowadzony przez użytkownika (nie ujęty w kluczowych typach obiektu)
EVENT	zdarzenie informacyjne (element strukturalny procesu i akcji)
FILE	zbiór danych
FLOW	ciąg zdarzeń w określonym przedziale czasu
FORM	format raportu (zestaw parametrów formatu strony i wiersza)
FOLDER	ciąg logicznie powiązanych raportów
IDKEY	klucz identyfikujący obiektu
INTERFACE	łącze zewnętrzne
INTERACTION	wykaz zdarzeń wspólnych dla kilku akcji
LANGUAGE	charakterystyka języka programowania
LIFE-HISTORY	kronika życia (stanów) obiektu
MODIFIER	modyfikator (element ciała obiektu podlegający modyfikacji)
MODULE	moduł (zwykle moduł programowy)
NOTE	notatka
OBJECTIVE	cel (jeśli wymaga szczegółowego opisu)
OPER-SYSTEM	charakterystyka systemu operacyjnego komputera
QUERY	skatalogowany wzorzec zapytania w języku &DL
PACKAGE	wygenerowany pakiet programowy (zwykle ukierunkowany na problem)
PARAMETER	charakterystyka danych sterujących wprowadzanych „z zewnątrz" przy eksploatacji pakietu
PART	charakterystyka składnika obiektu nie ujętego w niniejszej specyfikacji
POINTER	charakterystyka niestandardowego wskaźnika adresowego
PROCESS	sekwencja akcji i zdarzeń, składająca się ze zdarzenia inicjującego (TRIGGER), zdarzeń wykonawczych i zdarzenia końcowego (TERMINAL)
PCODE	kod rozpoznawczy procesu
PROBLEM	problem podlegający opisowi specyfikacyjnemu
PROCEDURE	procedura programowa
PROGRAM	wykonywana jednostka programowa
PROGRAMMER	programista (dorobek jego działalności)
PROJECT	projekt (ciąg dokumentacji projektowych, programowych)
RECORD	zapis (element zbioru)
REPORT	tabulogram
RUN	przebieg
SCHEMA	schemat bazy danych
SUBSCHEMA	podschemat bazy danych
SESSION	sesja obsługi wsadu komend języka &DL
SET	logiczny zestaw danych
SITUATION	sytuacja powiązanych logicznie obiektów (raport stanów)
STATE	stany danego obiektu
STREAM	strumień zdarzeń
STRUCTURE	funktor strukturotwórczy (do definiowania metastruktur)
TASK	moduł przetwarzania, przeznaczony m.in. do realizacji takich obiektów wynikowych, jak SITUATION czy INTERSECTION
TEXT	dowolny tekst skatalogowany (obiekty niestrukturalne — czarna skrzynka)
USER	użytkownik
VECTOR	wektor danych lub wektor powiązań (w wektorowych bazach danych)

```
   INVOLVED IN ACTIVITIES : (lista obszarów działań, w których występuje dany proces)
   PRECEDENCE OF EVENTS : : (nazwa-zdarzenia — inicjującego,
                             nazwy-zdarzeń-wykonawczych,
                             nazwa-zdarzenia-kończącego)
   DURATION = czas-trwania
   FREQUENCY = częstotliwość
   CONDITIONS = warunki-występowania
   INPUT (nazwa-obiektu) DERIVED FROM/CONTAINED IN
   (nazwa-„ośrodka"-przechowującego)
   OPERATION (nazwa) USING (nazwa-narzędzia np. DBMS)
   OUTPUT (nazwa-obiektu) DIRECTED TO (nazwa-„ośrodka"-
   -odbierającego)
   ...
ENDDEF
C. DEF1 PROGRAM (nazwa-programu)
   LANGUAGE = nazwa-języka (języków)
   COMPILER = nazwa-kompilatora (kompilatorów)
   PROGRAMMER = nazwisko-programisty-wiodącego
   COMPUTER = model-komputera
   OPER-SYSTEM = nazwa-systemu-operacyjnego
   ...
   PRECEDENCE OF PARAMETERS : (lista-parametrów-sterujących- w-kolejności-ich-wprowadzania)
   PRECEDENCE OF MODULES : (moduł 1 ((moduł 2 (moduł 3,
   moduł 4)) (moduł 5)
   DEF2 MODULE (moduł 1)
   ...
     DEF3 PROCEDURE (procedurax)
       MODIFIERS : (lista-modyfikatorów)
     ENDDEF3
   ENDDEF2
   DEF2 MODULE (moduł 2)
   ...
   ENDDEF2
   ...
ENDDEF1
D. DEF SET (nazwa-zestawu-logicznego-danych)
   TYPE = COSET/LINE/TREE/FREENETWORK/UNDEFINED
   RECORD (nazwa) RELATED TO RECORD (nazwa) ON wyrażenie-1 BY wyrażenie-2
   DBASE = nazwa-bazy-danych
   DBMS = nazwa-systemu-zarządzania-bazą-danych
   ......
ENDDEF
```

* * *

Artykuł niniejszy jest wprowadzeniem do składni języka SL, na podstawie którego Czytelnik może wyrobić sobie pogląd o proponowanych możliwościach eksploatacyjnych. Nie dokonuję analizy porównawczej z językiem PSL (Problem Statement Language), gdyż różnice — szczególnie w zakresie opisu strukturalnego — wydają się oczywiste.

Gwoli ścisłości dodam, że jest to zerowa wersja języka, dla której jeszcze nie opracowano analizatora i innych elementów oprogramowania, warunkujących zastosowanie. Koncepcja języka i jego specyfikacja nie były finansowane przez żadną instytucję, nie reprezentuje więc interesów żadnej organizacji. Jako autor nie widzę przeszkód w praktycznym wykorzystywaniu tej publikacji przez kogokolwiek. Oczekuję ewentualnej dyskusji i polemiki na łamach INFORMATYKI.

JAN DAWIDOWSKI, PIOTR OWCZARCZAK
Akademia Ekonomiczna
Poznań

Zastosowanie pakietu PSL/PSA do prowadzenia słownika-skorowidza bazy danych

Relatywna niezależność tworzenia poszczególnych systemów informatycznych (SI) powoduje, że złożoność informacyjna, wywodząca się z „otoczenia" SI, jest w rezultacie w znacznym stopniu obarczona redundancją informacji, niespójnością czy ich metodyczną niejednoznacznością. Z problemami tymi projektant SI musi sobie radzić na etapie tworzenia projektu. Może to być osiągnięte jedynie przez zastosowanie nowego jakościowo podejścia, respektującego wymagania podejścia systemowego. Dotyczy to zarówno samej teorii, jak i praktyki projektowania SI (por. [9]). Z tych też idei wywodzą się koncepcje stosowania baz danych w SI.

Problemy te omawia syntetycznie L. A. Maciaszek w INFORMATYCE nr 7—8/81 [8]. Niestety, wśród wielu znanych na świecie pakietów programowych wspomagających proces tworzenia i konserwacji słowników-skorowidzów baz danych, wymienionych przez autora, zabrakło również znanego na świecie (i — co ważniejsze — dostępnego w Polsce) pakietu programowego PSL/PSA. Pakiet ten, wspomagając proces tworzenia systemów informatycznych zarządzania (SIZ), szczególnie szeroko wspomaga proces tworzenia struktur danych, stanowiący jeden z najważniejszych aspektów tworzenia SIZ. Z tego też względu chcielibyśmy przedstawić jego zastosowanie.

OGÓLNA KONCEPCJA PAKIETU PSL/PSA

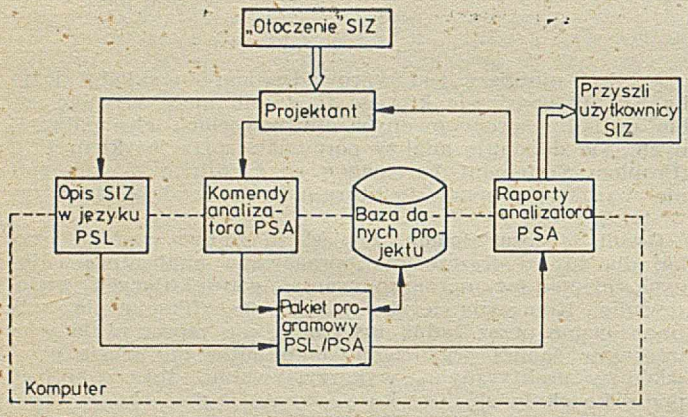
Pakiet programowy PSL/PSA (Problem Statement Language)/Program Statement Analyzer) należący do grupy oprogramowania narzędziowego systemów komputerowych wspomagających proces tworzenia i konserwacji systemów informatycznych zarządzania. Powstał on w pierwszej połowie lat 70-tych na Uniwersytecie w Michigan (Stany Zjednoczone) w ramach projektu ISDOS (Information System Design and Optimization System) pod kierunkiem prof. Daniela Teichroew'a. Pakiet ten składa się z dwóch części:

- języka opisu systemu (PSL)
- analizatora opisu systemu (PSA).

Język PSL służy do opisu istniejącego lub projektowanego systemu informacyjnego (informatycznego) w ramach formalizmów językowych, wywodzących się z hipotezy naukowej znanej pod nazwą ERA (Entity — Relation — Attribute), wyróżniających następujące kategorie opisu:

- obiekty systemu
- relacje występujące pomiędzy tymi obiektami
- atrybuty (cechy) obiektów.

Analizator PSA stanowi zestaw kilkudziesięciu programów komputerowych, których zadaniem jest tworzenie i aktualizacja bazy danych, zawierającej wszystkie informacje związane z prowadzonym projektem SIZ. Programy te, dzięki specjalnej technologii implementacji są łatwe do przenoszenia na dowolny typ komputera o określonych minimalnych możliwościach sprzętowych i programowych. Dzięki tej własności rozważania prezentowane w tym artykule powinny nabrać charakteru bardziej uniwersalnego, umożliwiającego ich wykorzystanie na sprzęcie krajowym.



Komputerowe wspomaganie procesu projektowania systemów informatycznych zarządzania w oparciu o pakiet programowy PSL/PSA

Schemat-procedurę współpracy projektanta SIZ z pakietem programowym PSL/PSA ilustruje rysunek 1. Zadania wykonywane przez projektanta w trakcie projektowania SIZ można scharakteryzować następująco:

etap 1: zbieranie informacji o potrzebach informacyjnych „otoczenia" projektowanego SIZ

etap 2: formalne opisanie tych informacji w języku PSL

etap 3: wprowadzenie za pomocą analizatora PSA przygotowanego opisu bazy danych projektu

etap 4: wygenerowanie raportów analitycznych i dokumentacyjnych dotyczących aktualnego stanu projektowanego SIZ

etap 5: analiza aktualnego stanu zaawansowania projektu; jeżeli wynik analizy wskazuje na konieczność wprowadzenia w projekcie zmian, to należy je wykonać zgodnie z wymienionymi wyżej etapami.

W trakcie wykonywania wymienionych zadań pakiet PSL/PSA stwarza szereg ułatwień i w konsekwencji uwalnia projektanta od wielu żmudnych i pracochłonnych czynności. Pakiet wspiera:

etap wywiadu z bezpośrednimi użytkownikami projektowanego SIZ przez:

— udostępnienie szerokiej gamy możliwości dokumentowania aktualnego stanu projektu, od raportów poglądowych (rysunków) — poprzez selektywne grupowanie informacji — do ich pełnego zestawienia

— komputerową analizę projektu, zapewniającą takie cechy projektu, jak: spójność, niesprzeczność i zupełność (zarówno w sensie struktury, jak i działania)

— narzucenie pewnych formalizmów i sposobów porozumiewania się projektanta z „otoczeniem", zapewniających jednoznaczność i pełną wymianę informacji

etap formułowania opisu projektu SIZ w języku PSL i wprowadzenie go do bazy danych projektu poprzez:

— formalną i merytoryczną kontrolę wprowadzanych do bazy informacji — pod względem poprawności ich konstrukcji językowych i niesprzeczności z aktualnym stanem zapisu projektu w bazie

— znaczne możliwości wprowadzania w bazie danych zmian dotyczących zarówno poszczególnych obiektów, jak i powiązań (relacji) między tymi obiektami

etap generowania raportów analitycznych i dokumentacyjnych i ich późniejszą analizę w zakresie stanu zaawansowania przechowywanego w bazie projektu SIZ.

MOŻLIWOŚCI RAPORTOWANIA AKTUALNEGO STANU PROJEKTU

Jak już wspomniano, zasadniczym celem opracowania analizatora PSA było komputerowe wspomaganie procesu projektowania SIZ. Głównym elementem tego wspomagania są dostarczane przez analizator PSA raporty, o aktualnym stanie zaawansowania prac projektowych. Raporty te podzielić można na:

- dokumentacyjne
- analityczne
- techniczne.

Raporty dokumentacyjne zawierają kompleksowe zestawienia informacji dotyczących projektowanego SIZ, natomiast raporty analityczne — elementy oceny merytorycznej projektu: analizę porównawczą informacji opisujących system oraz zestawienie przypuszczalnych błędów logicznych projektu. Reporty techniczne związane są z procesem wprowadzania zmian w bazie danych i mają wyraźnie roboczy charakter.

W fazie projektowania zasadnicze znaczenie mają raporty analityczne, które ten proces silnie wspomagają, natomiast w fazie dokumentowania — raporty dokumentacyjne, zawierające jednoznaczny i uporządkowany opis projektowanego systemu.

Raporty dokumentacyjne

Ze względu na zawartość informacyjną raporty dokumentacyjne podzielić można na:

- listy nazw obiektów
- raporty poglądowe
- raporty słownikowe
- raporty opisu pełnego
- raporty statystyczne.

Listy nazw obiektów są najprostszymi raportami dokumentacyjnymi analizatora PSA. Zawierają one uporządkowane, alfabetycznie lub według typów obiektów, zestawienia nazw określonych i zdefiniowanych w bazie danych obiektów. Do grupy tej należą raporty: NAME-GENERATION (lista nazw wybranych obiektów), NAME-LIST (lista nazw wszystkich obiektów) i KWIC (indeks permutowany wybranych obiektów).

Raporty poglądowe są najbardziej obrazowymi raportami analizatora PSA, przedstawiającymi poglądowo ogólną koncepcję projektowanego systemu. Zawierają one jedno- lub wieloaspektowe „graficzne" zestawienia informacji dotyczących powiązań obiektów w SIZ. Do grupy tej należą: PICTURE (przedstawiający przepływ, strukturę i przetwarzanie danych lub procesów) oraz FREQUENCY (przedstawiający strukturę i wzajemne zależności obiektów typu o k r e s c z a s o w y).

Raporty słownikowe zawierają uporządkowane, alfabetycznie lub według typów obiektów, zestawienia nazw wybranych obiektów bazy danych z pewnymi informacjami uzupełniającymi w formie opisów słownych. Do grupy tej należą raporty: DICTIONARY (słownik ogólny), PROCESS-INPUT-OUTPUT (słownik procesów) i PRINT-ATTRIBUTE-VALUE (słownik cech obiektywów).

Raporty opisu pełnego zawierają kompletne opisy wybranych obiektów BDP. Do grupy tej należy raport FORMATTED-PROBLEM-STATEMENT.

Raporty statystyczne zawierają zestawienia sumaryczne dotyczące ilości zdefiniowanych w BDP obiektów. Do grupy tej należy raport SUMMARY.

Raporty analityczne

Ze względu na zawartość informacyjną, raporty analityczne można podzielić na:

- analizy struktur formalnych obiektów
- analizy struktur informacyjnych jednostek danych
- analizy wykorzystania jednostek danych i procesów.

Analizy struktur formalnych obiektów pozwalają stwierdzić spójność formalną projektowanego SIZ. Przedstawia-

ją one zależności pomiędzy obiektami, związane z hierarchiczną lub sieciową strukturą tych obiektów w projektowanym SIZ. Pozwalają one na zwięzłe przedstawienie, w postaci list hierarchicznych lub macierzy, powiązań obiektów — z wyróżnieniem tych obiektów, które nie mają powiązań. Do tej grupy należą raporty: STRUKTURE (struktura obiektów w postaci listy hierarchicznej), CONTENTS (struktury sieciowe obiektów w postaci listy) oraz CONSISTS-MATRIX (powiązania sieciowe obiektów w postaci macierzy).

Analizy struktur informacyjnych jednostek danych pozwalają stwierdzić stopień redundancji i spójności jednostek danych. Przedstawiają one w formie macierzy powiązań analizę ich zawartości informacyjnej. Do grupy tej należą raporty: CONSISTS-COMPARISON (analiza jednostek danych pod względem zawartości informacyjnej) oraz ENTITY-IDENTIFIER (analiza identyfikatorów rekordów).

Analizy wykorzystania jednostek danych i procesów obrazują w postaci macierzowej powiązania tych obiektów. Do grupy tej należy raport DATA-PROCES. Raport ten pozwala stwierdzić kompletność oraz spójność projektu SIZ. Dokładniejsze omówienie zawartości i stosowania raportów analizatora PSA można będzie znaleźć w publikacji [6].

●

Ze względu na możliwości pakietu PSL/PSA celowym wydaje się polecenie tego narzędzia jako „typowego" pakietu obsługi słownika-skorowidza bazy danych. Wynika to nie tylko z zaspokajania przez ten pakiet typowych wymagań użytkowników baz danych (w pełni — w zakresie opisu: jednostek danych i stosowanych norm opisu oraz powiązań i kontroli spójności logicznej, natomiast w zakresie generowania kodu schematu bazy danych — sprowadza je do poziomu wspólnego dla wielu języków bazowych). Cechą o wiele bardziej przemawiającą za zastosowaniem tego narzędzia do prowadzenia słowników-skorowidzów baz danych jest wysoki stopień niezależności stosowanego w tym pakiecie aparatu teoretycznego i to zarówno od stosowanych komputerów instrumentalnych (tzn. komputerów, na których to narzędzie jest stosowane), jak i komputerów docelowych (tzn. komputerów, na których będą eksploatowane projektowane bazy danych). Umożliwia to z jednej strony specjalna technologia implementacji analizatora PSA, a z drugiej — konsekwentne realizowanie modelu opisu SIZ wywodzącego się ze wspomnianej na wstępie hipotezy ERA.

Niemałe znaczenie ma również fakt możliwości komputerowego wspomagania procesu projektowania nie tylko bazy danych, ale i całego zbudowanego na niej systemu informatycznego, a także automatyzacji tworzenia dokumentacji projektowej oraz implementacji oprogramowania systemu połączonej z symulacyjnym optymalizowaniem jego działania.

Warto też podkreślić, że pakiet radykalnie zmniejsza pracochłonność konserwacji baz danych i oprogramowania wykonanego za pomocą tej technologii, w wyniku zwolnienia projektanta z wykonywania wielu rutynowych czynności.

Zastosowanie pakietu stwarza techniczną podstawę do budowy branżowych, regionalnych lub krajowych bibliotek projektów typowych, wielokrotnie zmniejszających krajowe nakłady na realizację nowych systemów informatycznych, przy jednoczesnej poprawie ich jakości, skróceniu czasu realizacji oraz wydłużenia cyklu życia.

* * *

Akademia Ekonomiczna w Poznaniu od 1977 r. prowadzi badania w dziedzinie komputerowego wspomagania procesu tworzenia i konserwacji SIZ. Zakres prowadzonych tutaj prac zawarto w publikacji [3]. Do praktycznych wyników tych prac można zaliczyć:

w 1979 roku — implementację pakietu PSL/PSA (wersja 2.1R5) na komputerze ODRA 1305 (system operacyjny GEORGE 3, wersja 8.65)

w 1980 roku — a) wdrożenie do eksploatacji pakietu PSL/PSA w skali zarówno „dydaktycznej" (dla studentów Studiów Podyplomowych), jak i „praktycznej" — do realizacji wielu tematów naukowo-badawczych (np. projekty banków danych SIZ, systemów komputerowego wspomagania projektowania SIZ); b) opracowanie pakietu programowego GD (generator dokumentacji), wspomagającego proces tworzenia dokumentacji różnych typów; c) eksperymentalne automatyczne generowanie oprogramowania SIZ w oparciu o pakiety programowe PSL/PSA, GD i DMS-2

w 1981 roku — opracowanie technologii komputerowo wspomaganego projektowania SIZ, z wykorzystaniem pakietów PSL/PSA, GD i DMS-2.

Dotychczasowe wyniki prac prezentowaliśmy na konferencjach: INFOGRYF'80 ([1,5]), „Banki danych w skomputeryzowanych systemach zarządzania" ([2,7]) oraz „Wielodostęp w zarządzaniu, badaniach naukowych i dydaktyce" ([4]). Mając na względzie szeroko pojęte dobro naszej informatyki chcielibyśmy zaoferować ze swej strony wszystkim zainteresowanym swoje dotychczasowe doświadczenia i wyniki.

LITERATURA

[1] Benc Cz., Danek A., Dawidowski J., Owczarczak P.: Wykorzystanie pakietu PSL/PSA w projektowaniu systemów informatycznych. Mat. Konf. INFOGRYF'80, Szczecin-Kołobrzeg 1980

[2] Benc Cz., Danek A., Dawidowski J., Kryszak E., Owczarczak P., Wiśniewski M.: Technika i narzędzia projektowania banku danych dla zarządzania kompleksem rolno-spożywczym górnej Noteci. Mat. narady „Bank danych...", Poznań, 1979

[3] Benc Cz., Dawidowski J., Owczarczak P.: Kompleks narzędzi komputerowego wspomagania procesu tworzenia i konserwacji systemów informatycznych zarządzania. Biul. Techniczny MERA 3/81

[4] Benc Cz., Dawidowski J., Owczarczak P.: Komputerowo wspomagane projektowanie systemów informatycznych z bankiem danych za pomocą pakietu PSL/PSA na komputerze ODRA 1305. Mat. kursokonf. „Wielodostęp...", Poznań, 1980

[5] Benc Cz., Dawidowski J., Owczarczak P.: Pakiet PSL/PSA narzędziem wspomagania procesu projektowania systemów informatycznych na bazie komputera ODRA 1305. Mat. konf. INFOGRYF'80, Szczecin-Kołobrzeg, 1980

[6] Dawidowski J., Owczarczak P.: Opis narzędzia projektowo-programowego PSL/PSA (w przygotowaniu)

[7] Dawidowski J., Owczarczak P., Wiśniewski M.: Automatyczne generowanie dokumentacji projektowej. Mat. narady „Bank danych...", Poznań, 1981

[8] Maciaszek L. A.: Słownik-skorowidz w procesie powstawania i utrzymywania baz danych. INFORMATYKA 7—8/81

[9] Soltés D.: Informačné zabezpečenie automatizovaného informačného systému. MECHANIZACE A AUTOMATIZACE ADMINISTRATIVY 5/81.

HALINA SONDEJ, JANUSZ WIŚNIEWSKI
Ogólnouczelniany Ośrodek Obliczeniowy
Uniwersytet im. Mikołaja Kopernika
Toruń

MOLATO — system wspomagający nauczanie

Od czasu pierwszych prób zastosowania komputerów w nauczaniu minęło już ponad 20 lat [7]. Komputer wprawdzie nie zrewolucjonizował światowej dydaktyki, niemniej przyczynił się do usprawnienia procesu nauczania, szczególnie w szkolnictwie wyższym. Bogaty przegląd badań prowadzonych w tej dziedzinie przez różne ośrodki naukowe przedstawił w swej książce S. Jarmak [6]. Indywidualizacja procesu nauczania, możliwość symulacji zjawisk i spadek kosztu kształcenia [1] — to trzy najczęściej wymieniane czynniki uzasadniające intensywne wykorzystanie informatyki w dydaktyce akademickiej.

W ostatnich latach ukształtowały się następujące kierunki zastosowań metod informatycznych w nauczaniu:

- interakcyjne programowanie połączone z rozwiązywaniem problemów
- ćwiczenia w dialogu z komputerem
- testowanie wyników uczenia się
- symulacja procesów na ekranie monitora
- ćwiczenia oraz utrwalenie materiału nauczania w systemie repetytoryjnym
- nauczanie: przekazywanie i utrwalanie wiadomości.

Tak rozumiane wspomaganie nauczania objęło nie tylko informatykę, fizykę, chemię, część matematyki czy projektowanie techniczne, ale wkroczyło także do nauk społecznych. Przykładem niech będzie jeden z najbardziej znanych — system PLATO (Programmed Logic for Automatic Teaching Operations) [2], opracowany w Coordinated Science Laboratory (University of Illinois, Urbana) pod kierunkiem D.L. Bitzera. Funkcje i możliwości systemu były ostatnio szczegółowo przedstawione na łamach INFORMATYKI w artykułach Z. Gackowskiego i M. Hołyńskiego [4, 5].

Systemem, który — podobnie jak PLATO — stanowi jeden z najbardziej efektownych przykładów zastosowania techniki obliczeniowej w nauczaniu jest system SOCRATES, opracowany w kalifornijskim State University and Colleges. Jest to największy w świecie system wspomaganego komputerem sprawdzania wiadomości z różnych dziedzin wiedzy. Pozwala on na obsługę ponad 250 tys. studentów z kilkunastu uczelni amerykańskich.

Systemy SOCRATES i PLATO to projekty potężne, drogie, wymagające bardzo nowoczesnej technologii. Oprócz nich w różnych krajach świata funkcjonują systemy znacznie mniejsze, realizowane w oparciu o małe instalacje komputerowe i przeznaczone dla nauczania wybranych dziedzin wiedzy. Prezentowany poniżej system MOLATO [1], służący nauczaniu fizyki atomowo-molekularnej, jest przykładem krajowych prac prowadzonych w tej dziedzinie. Przystosowany do istniejących w Polsce konfiguracji sprzętowych — pozwala na zapewnienie indywidualizacji procesu nauczania oraz bieżące lub okresowe sprawdzania jego wyników. Doświadczenia zebrane podczas dotychczasowej eksploatacji w Uniwersytecie Mikołaja Kopernika świadczą o jego walorach użytkowych oraz atrakcyjności dla korzystających z niego studentów.

STRUKTURA SYSTEMU

System dydaktyczny MOLATO służy do wspomagania nauczania fizyki atomowo-molekularnej oraz innych dowolnie wybranych przedmiotów. Komunikacja z systemem odbywa się w sposób konwersacyjny, przy wykorzystaniu w tym celu monitoru ekranowego. MOLATO składa się z czterech modułów:

- CZDAT — umożliwiającego modelowanie potencjału cząsteczek dwuatomowych oraz obliczanie energii poziomów oscylacyjnych tych cząsteczek
- CNINDO — służącego do teoretycznego badania w trybie konwersacyjnym struktury elektronowej układów wieloatomowych metodami CNDO/2 i INDO, w schemacie zamknięto- i otwartopowłokowym
- MSW — służącego do sprawdzania wiadomości studentów z różnych przedmiotów; może być on także wykorzystany w celach repetytoryjnych oraz jako środek do prezentacji wiedzy z wybranej dziedziny
- POMOC — składającego się z programów umożliwiających aktualizację zbiorów wykorzystywanych przez moduł MSW.

Moduł CZDAT — na podstawie podanego przez studenta kształtu potencjału cząsteczki dwuatomowej oraz liczb masowych tworzących ją atomów — oblicza energię szesnastu najniższych poziomów oscylacyjnych cząsteczki o-

---

[1]) Opisany system został opracowany w Ogólnouczelnianym Ośrodku Obliczeniowym oraz Zakładzie Metodyki Nauczania Fizyki Instytutu Fizyki Uniwersytetu Mikołaja Kopernika w Toruniu przez zespół w składzie: R. Bobkowski, T. Orlikowski, H. Sondej, J. Wiśniewski, M. Żurawska i B. Żurawski;
MOLATO — nazwa utworzona ze słów MOLekularna i ATOmowa, akcentująca przeznaczenie systemu do wspomagania nauczania fizyki atomowo-molekularnej oraz nawiązująca (brzmieniowo) do nazwy PLATO

Mgr HALINA SONDEJ ukończyła w 1979 r. fizykę w Uniwersytecie Mikołaja Kopernika w Toruniu. Pracuje w Zakładzie Informatyki Stosowanej Ogólnouczelnianego Ośrodka Obliczeniowego UMK w Toruniu na stanowisku asystenta. Oprócz pracy dydaktycznej zajmuje się projektowaniem systemów informatycznego wspomagania nauczania i badań naukowych.

Mgr JANUSZ WIŚNIEWSKI ukończył w 1979 r. fizykę w Uniwersytecie Mikołaja Kopernika w Toruniu. Posiada oprócz tego wyższe wykształcenie ekonomiczne. Pracuje w Zakładzie Informatyki Stosowanej Ogólnouczelnianego Ośrodka Obliczeniowego UMK w Toruniu na stanowisku asystenta. Poza pracą dydaktyczną zajmuje się projektowaniem systemów informatycznego wspomagania nauczania i badań naukowych.

raz wykreśla krzywą potencjału wraz z naniesionymi na nią czterema najniższymi poziomami oscylacyjnymi (naniesienie większej liczby poziomów uczyniłoby wykres nieczytelnym). Kształt krzywej potencjału można aproksymować numerycznie — krzywą Morse'a lub krzywą Lenarda-Jonesa. Poziomy oscylacyjne są obliczane przez przybliżone rozwiązanie równania Schrödingera dla układu dwuatomowego metodą opisaną przez Cooley'a [3].

Do modułu CNINDO student wprowadza liczby atomowe atomów tworzących badany układ, współrzędne geometryczne opisujące wzajemne rozmieszczenie tych atomów oraz multipleksowość stanu podstawowego i całkowity ładunek układu. Moduł ten pozwala uzyskać o układzie m. in. następujące informacje: wartość energii elektronowej, całkowitej i energii wiązania, energie orbitalne, orbitale molekularne przedstawione w postaci współczynników rozwinięcia na funkcje bazowe (którymi są orbitale atomowe typu Slatera), rozkład ładunków w układzie, momenty dipolowe oraz stałe sprzężenia nadsubtelnego. Moduł umożliwia badanie zależności wymienionych wielkości od różnych konfiguracji geometrycznych rozważanego układu.

Moduł MSW prezentuje zestaw pytań testowych wraz z kilkoma (co najmniej pięcioma) odpowiedziami do wyboru. Po podaniu numeru wybranej odpowiedzi student jest informowany o liczbie punktów zdobytych za daną odpowiedź. Przystępując do testowania można dokonać wyboru tematyki testów spośród aktualnie dostępnych, tzn. dotychczas opracowanych bloków tematycznych. Po zakończeniu testowania z wybranej lub zaleconej przez prowadzącego zajęcia tematyki student otrzymuje informację o sumie zdobytych punktów, natomiast prowadzący zajęcia ma możliwość poznania dorobku punktowego studenta, grupy studenckiej lub wszystkich studentów zarejestrowanych w systemie, tzn. podlegających testowaniu. Moduł ten może być również wykorzystywany przez studentów w celach repetytoryjnych, aby sprawdzić stopień opanowania przez siebie wybranej dziedziny wiedzy.

System umożliwia prezentowanie usystematyzowanej wiedzy z dowolnej dziedziny, przy czym możliwy jest — dzięki dowolności sekwencji prezentowanych zagadnień — wybór indywidualnej drogi.

Moduł MSW korzysta z pięciu zbiorów o organizacji indeksowo-sekwencyjnej: BANK, DYDA, INIT, STAT i WYK.

W zbiorze BANK przechowywane są pytania testowe z dowolnych dziedzin wiedzy, pogrupowane w bloki tematyczne. Każdy rekord tego zbioru zawiera jeden test wraz z odpowiedziami do wyboru i jest zaopatrzony w klucz będący numerem testu. Prowadzący zajęcia mogą stale uzupełniać oraz poprawiać zbiór testów.

Zbiór DYDA zawiera macierz sekwencji testów, sterującą kolejnością prezentowanych pytań oraz liczbą przyznawanych punktów. Kluczami rekordów zbioru DYDA są numery testów umieszczonych w zbiorze BANK. Każdy rekord zawiera liczbę punktów uzyskiwanych za podanie każdej z pięciu odpowiedzi do wyboru oraz numer testu, który ma zostać wyświetlony jako następny, zależnie od tego, którą z podanych odpowiedzi student wybierze. Sekwencja pojawiających się na ekranie monitora testów zależy zatem od odpowiedzi udzielanych przez studenta i jest ustalana przez prowadzącego zajęcia — zgodnie z punktacją za odpowiedzi. Jeżeli podana przez studenta odpowiedź jest częściowo lub całkowicie błędna, możliwe jest stawianie mu różnych pytań pomocniczych, w zależności od rodzaju popełnionego błędu. Osiągnięto w ten sposób zindywidualizowany do pewnego stopnia proces testowania.

Zbiór INIT zawiera takie informacje o studentach uprawnionych do testowania, jak: numer indeksu, imię, nazwisko, nazwa uczelni, rok studiów, specjalność i numer grupy. Student przystępujący do testowania jest proszony o podanie numeru swego indeksu oraz numeru grupy. W przypadku, gdy zbiór INIT nie zawiera podanego numeru indeksu, student nie jest dopuszczony do testowania.

W zbiorze STAT przechowywany jest dorobek punktowy studentów, zgromadzony podczas ostatniego testowania. Zbiór ten może być przeglądany przez prowadzącego zajęcia.

Zbiór WYK zawiera wykaz tematyki testów umieszczonych w zbiorze BANK. Zawartość tego zbioru jest wyświetlana przy przystępowaniu do testowania.

Prowadzący zajęcia we wszystkich tych zbiorach mogą dopisywać nowe rekordy lub zmieniać zawartość rekordów już istniejących. Służy do tego celu moduł POMOC, składający się z programów aktualizujących poszczególne zbiory. Została zatem stworzona dla prowadzących zajęcia możliwość umieszczania w zbiorze BANK własnych testów z dowolnej dziedziny oraz realizowania własnych koncepcji dotyczących sekwencji prezentowanych testów i punktacji za odpowiedzi. Na szczególne podkreślenie zasługuje fakt, że aktualizacji zbiorów dokonuje się w trybie konwersacyjnym i że nie jest w tym celu wymagana znajomość żadnego języka programowania. Prowadzący zajęcia, po wywołaniu programu aktualizującego dany zbiór, jest krótko informowany o formacie, w którym powinien podać zawartość dodawanego czy zmienianego rekordu w zbiorze, po czym przystępuje do podawania tych danych.

OPIS SPRZĘTU I UWAGI EKSPLOATACYJNE

Do prawidłowej pracy systemu wymagana jest następująca konfiguracja sprzętowa:
— jednostka centralna R-32 z pamięcią o pojemności co najmniej 512 K bajtów
— zestaw monitorów ekranowych w wersji lokalnej lub zdalnej
— pamięci 2 dyskowe po 30 M bajtów oraz dostęp do 36 cylindrów na trzecim dysku o analogicznej pojemności
— jednostka pamięci taśmowej.
Praca przebiega pod nadzorem systemu operacyjnego OS w wersji MVT z opcją TSO (Time Sharing Option) i metodami dostępu TCAM.

Moduły systemu MOLATO w formie przystosowanej do eksploatacji oraz dokumentacja projektowa, eksploatacyjna, przewodnik dla prowadzącego zajęcia i przewodnik dla studentów są dostarczane przez Zakład Elektronicznej Techniki Obliczeniowej w Poznaniu bądź Ogólnouczelniany Ośrodek Obliczeniowy UMK w Toruniu. Wszelkich informacji eksploatacyjnych udzielają również bezpośrednio jego autorzy.

Moduły CZDAT i CNINDO zostały napisane w języku FORTRAN IV, natomiast moduły MSW i POMOC w języku PL/1.

* * *

Działanie prezentowanego systemu zostało sprawdzone podczas zajęć ze studentami z różnych kierunków. Studenci bardzo szybko przyswajali sobie zasady obsługi monitorów ekranowych i uznali, że tego typu zajęcia są o wiele bardziej atrakcyjne od tradycyjnych ćwiczeń. Prezentowany system, tak jak i inne komputerowe systemy dydaktyczne, umożliwia efektywne wykorzystanie czasu każdego studenta podczas zajęć, rozwinięcie jego samodzielności, indywidualizację nauczania zgodnie z jego predyspozycjami i wiedzą, a także bardziej obrazowe niż w tradycyjnych metodach przedstawianie materiału. Jest to ponadto system uniwersalny, nadający się do nauczania dowolnych przedmiotów.

Dalsze prace nad systemem obejmują m.in. zastosowanie monitorów graficznych, mające na celu poszerzenie zastosowań systemu.

LITERATURA

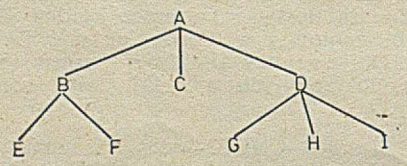
[1] Bark A.: Learning with computers — today and tomorrow. IFIP, North-Holland, Publishing Company, 1975
[2] Bitzer D., Easly I. A.: PLATO: A Computer Controlled Teaching System. Coordinated Science Laboratory. University of Illinois, Urbana, I11
[3] Cooley J. W.: Moth. Comp. 15, 363 1961
[4] Gackowski Z.: Informatyka w amerykańskiej dydaktyce INFORMATYKA 9, 1980
[5] Hołyński M.: Symulacja procesu nauczania w systemie PLATO. INFORMATYKA, 2, 1981
[6] Jarmak S.: Komputery w dydaktyce szkoły wyższej. PWN, Warszawa, 1979
[7] Klotz G.: Computer als Legrmaschinen. Zeitschrift für Datenarbeitung, 5, 1967.

Procedura generująca drzewa etykietowane

Przedstawimy poniżej procedurę generującą w sposób losowy (z rozkładem równomiernym) drzewa etykietowane o n wierzchołkach z wyróżnionym korzeniem. Dla danej liczby n są, dzięki niej, generowane drzewa etykietowane o n wierzchołkach — tak, aby prawdopodobieństwa otrzymania każdego z drzew były równe.

Motywacja

Symulacja cyfrowa jest często jednym z kluczowych narzędzi do badania dużych systemów, w których zmiany stanów są trudne do przewidzenia. Z dużej liczby modeli symulacyjnych, jako dane podstawowe, można wyodrębnić d r z e w a. Przykładem może tu być symulacja procesów w systemie operacyjnym. Rozważmy pojedynczy proces A wygenerowany przez system. Wiemy o nim tylko tyle, że może on generować inne procesy o analogicznej strukturze, przy czym ich liczba jest niemożliwa do przewidzenia. Powstająca struktura jest drzewem. Przykładową sytuację obrazuje rysunek 1. Proces A generuje procesy B, C, D; proces B — procesy E, F itd. Dobrze przeprowadzona symulacja wymagać będzie dokonania analizy dużej liczby drzew powstałych zupełnie przypadkowo.



Rys. 1

Prezentowany algorytm jest ważnym fragmentem każdego programu badającego własności takich systemów, w których podstawowe elementy składowe dają się opisać jako drzewa.

Drugą, nie mniej istotną, funkcją prezentowanej procedury jest możliwość jej użycia przy testowaniu programów operujących na drzewiastych strukturach danych.

Opis algorytmu

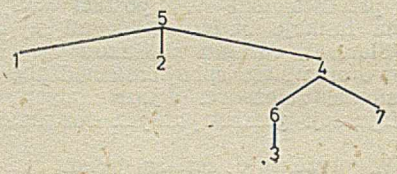
Pomysł algorytmu opiera się na spostrzeżeniu, że liczba drzew etykietowanych o n wierzchołkach z wyróżnionym korzeniem jest równa liczbie $(n-1)$-wyrazowych ciągów utworzonych z n elementów (por. Knuth D. E.: The Art of Computer Programming. vol. 1, § 2.3.4.4).

Pierwszym krokiem jest wygenerowanie $(n-1)$-wyrazowej wariacji z powtórzeniami ze zbioru $\{1, ..., n\}$. Następnie z uzyskanego ciągu tworzymy drzewo według metody, której szkic czytelnik może znaleźć w wyżej wymienionej monografii Knutha w ćwiczeniu 18, § 2.3.4.4:

Załóżmy, że mamy dany ciąg $s_1, ..., s_{n-1}$ liczb ze zbioru $\{1, ..., n\}$. Definiujemy s_1 jako korzeń drzewa. Kolejno do s_1, s_2,... dołączamy wierzchołki, używając procedury: jeśli wierzchołek s_k występował w ciągu wcześniej, to dołączamy go do s_{k-1} nie nadając mu żadnej etykiety, w przeciwnym przypadku dołączamy go do s_{k-1} nadając mu etykietę s_k.

Po (n-1)-krotnym wykonaniu powyższej instrukcji etykietujemy wszystkie nie nazwane wierzchołki numerami, które dotychczas nie wystąpiły, używając ich w porządku rosnącym.

Na przykład — z ciągu 5,5,5,4,6,4 uzyskujemy drzewo przedstawione na rysunku 2.



Rys. 2

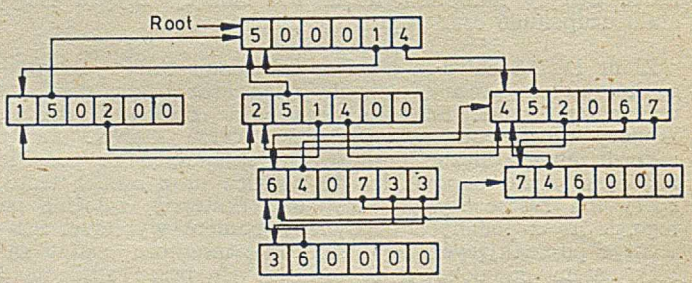
Zauważmy, że różnym ciągom zostaną w ten sposób przyporządkowane różne drzewa, co implikuje poprawność rozwiązania. Każde drzewo generowane jest oczywiście z prawdopodobieństwem równym $1/(n\ n-1)$, zatem uzyskany rozkład prawdopodobieństwa jest równomierny.

Struktura danych

Drzewo „pamiętane" jest w tablicy dwuwymiarowej $6 \times n$. Każdy wierzchołek opisywany jest przez sześć pól:

- etykieta wierzchołka
- wskaźnik (indeks tablicy) do ojca
- wskaźnik do lewego brata
- wskaźnik do prawego brata
- wskaźnik do pierwszego syna
- wskaźnik do ostatniego syna.

W przypadku, gdy dany element nie ma np. synów — odpowiednie wskaźniki są równe *0*. Na przykład, drzewo z przykładu prezentowanego w opisie algorytmu przedstawione jest na rysunku 3.



Rys. 3

Opis procedury gentree

Procedura ma trzy parametry:

- liczba naturalna n określająca ilość wierzchołków drzewa
- całkowita tablica *tree* o wymiarach *[1:6, 1:n]*, reprezentująca drzewo uzyskane w wyniku działania procedury
- liczba naturalna *root*, będąca indeksem do elementu tablicy opisującego korzeń drzewa.

Procedura wywołuje funkcję *random (k, m)*, generującą losowo liczby naturalne z przedziału *[k, m]* z rozkładem równomiernym. Funkcje takie są dostępne w przypadku większości maszyn cyfrowych, więc nie będziemy przytaczać ich treści (można je znaleźć również w dru-

gim tomie wspomnianej monografii Knutha lub w wydanej przez WNT książce R. Zielińskiego „Generatory liczb losowych").

Nietrudno zauważyć, że *gentree* działa w czasie liniowym w zależności od *n*. Oznacza to, że liczba instrukcji wykonywanych podczas działania procedury jest wprost proporcjonalna do liczby wierzchołków generowanego drzewa.

Wyniki empirycznych pomiarów wykonanych przy użyciu minikomputera MERA 400

Liczba wierzchołków	Czas działania (s)
100	0,4
300	1,1
500	1,8
1000	3,6
1500	5,5
2000	7,4

```
procedure gentree (n, tree, root); integer n, root;
                                   integer array tree [1:6,
                                   1:n];
comment procedura generuje losowo (z rozkładem równo-
        miernym) drzewa etykietowane o n wierzchoł-
        kach z korzeniem;
begin integer wait, i, j, ix, jx, x, father;
if n ⩾ 1 then
  begin wait: = 0;
    for i: = 1 step 1 until n do
      begin tree [1, i]: = random (1, n);
        tree [3, i]: = 1;
        tree [5, i]: = tree [6, i]: = 0
      end;
    root: = tree [1, 1]; tree [1, n]: = root;
    tree [2, root]: = tree [3, root]: = 0;
    for i: = 1 step 1 until n-1 do
      begin ix: = tree [1, i+1]; jx: = tree [1, i]; tree [1, i]: =
      = i;
        if tree [3, ix] = 0 then begin wait: = wait+1;
tree [4, wait]: = jx
end
        else tree [2, ix]: = jx;
        tree [3, ix]: = 0
      end;
j: = 1; tree [1, n]: = n;
for i: = 1 step 1 until wait do
  begin
    while tree [3, j] = 0 do j: = j+1;
    tree [2, j]: = tree [4, i];
    j: = j+1
  end;
comment w tym momencie tree [2, i] pokazuje na ojca i-
-tego wierzchołka;
for i: = 1 step 1 until n do tree [3, i]: = tree [4, i]: = 0;
for i: = 1 step 1 until root-1, root+1 step 1 until n do
  begin father: = tree [2, i];
    if tree [5, father] = 0 then tree [5, father]: =
                                 tree [6, father]: = i
    else begin x: = tree [6, father]; tree [4, x]: = i;
      tree [3, i]. = x; tree [6, father]: = i
    end
  end
end gentree;
```

ANDRZEJ SZAŁAS
ZBIGNIEW ŚWIRSKI

Sytuacja kadry w ośrodkach informatyki

Na początku 1982 r. Sekretariat Komitetu Informatyki otrzymał informacje wskazujące na zaniepokojenie środowiska informatycznego możliwością znacznej redukcji zatrudnienia. Dla zbadania skali zjawiska oraz jego przyczyn, Sekretariat zlecił Zakładowi Projektowania i Obsługi MNSzWiT zanalizowanie problemu.

Analizy sytuacji kadrowej dokonano na podstawie przeprowadzonej ankiety oraz danych za rok 1980 (ze zbioru GUS — A01 „Działalność Ośrodków Informatycznych"). Ankieta miała charakter dobrowolny i dotyczyła głównie liczby pracowników przewidzianych do zwolnienia w bieżącym roku.

ANKIETA

Ankietę rozesłano do 1256 ośrodków, zatrudniających powyżej pięciu osób. Stanowi to 78% wszystkich ośrodków (1603 — wg sprawozdań GUS za 1980 r.). Do 20 marca br. otrzymaliśmy 218 pisemnych odpowiedzi, co stanowi 17,4% wysłanych ankiet (tab. 1).

Analiza wielkości i struktury zwolnień

Ankieta była adresowana do ośrodków, w których reorganizacja gospodarki oraz trwający kryzys spowodowały konieczność zmniejszenia zatrudnienia. Można wobec tego uważać, że ośrodki, które nie odpowiedziały na ankietę, nie przewidują istotniejszych zwolnień pracowników.

32% analizowanych ośrodków przewiduje w bieżącym roku zwolnienia pracowników. Stanowi to ok. 5,6% ogólnej liczby ośrodków, do których ankieta została wysłana. Ośrodki te zatrudniały pod koniec 1980 r. 5226 pracowników i w ciągu 1981 r. zwolniły 663 osoby (13%). Na początku 1982 r. ośrodki te zatrudniały 4563 osoby i przewidują zwolnić w tym roku 650 osób, czyli — 14%. Tak więc proces przewidywanych zwolnień nie będzie odbiegać od sytuacji w roku ubiegłym.

Zmieniła się natomiast zasadniczo struktura zatrudnienia zwalnianych osób. W 1981 r. zwalniano przede wszystkim osoby nisko kwalifikowane (według nomenklatury GUS — pozostali pracownicy działalności podstawowej), zaś w 1982 r. przewiduje się zwolnienie 23% ogółu zatrudnionych programistów (w 70 ośrodkach) oraz 20% projektantów systemów i analityków. Ma więc odejść kadra, w którą społeczeństwo najpoważniej zainwestowało.

Tabela 1. Dane statystyczne stanowiące podstawę ankiety

	Ogółem	Województwo warszawskie
Liczba ośrodków (31.12.80)	1 603	311
Liczba osób zatrudnionych w ośrodkach (31.12.80)	56 379	13 466
Liczba ankietowanych ośrodków	1 256	249
Liczba osób zatrudnionych w ośrodkach ankietowanych	53 913	13 018
Liczba ośrodków które nadesłały odpowiedzi	218	38
Liczba ośrodków, które przewidują zwolnienia	70	11
Liczba osób przewidzianych do zwolnienia w 1982 r.	650	195

Strukturę zatrudnienia osób przewidzianych do zwolnienia w 1982 r. przedstawia tabela 2.

W 1980 r. poza nielicznymi wyjątkami w dużych aglomeracjach liczba zwolnionych pracowników wahała się w granicach 10%. W 1982 r. ilościowo najwięcej zwolnień przewiduje się w Warszawie (195 osób) oraz w województwach: łódzkim (101 osób), wrocławskim (71 osób), krakowskim (64 osoby i gdańskim (140 osób) — tabela 3. Województwa zatrudniające wielu informatyków również wielu ich zwalniają. Wyjątek stanowi województwo katowickie, niemniej liczba informatyków w stosunku do liczby mieszkańców wynosi tam tylko 2,1%, podczas gdy np. w województwie warszawskim 5,8%, wrocławskim 3,6%, łódzkim 3,1%.

Podobnie wygląda struktura zwolnień pracowników w układzie resortowym (tab. 4). Największej grupy informatyków pozbywają się resorty, w których ich liczba była największa.

Tabela 2. Struktura zatrudnienia pracowników przewidzianych do zwolnienia w 1982 r.

		Ogółem liczba zatrudnionych w kraju w 1980 r.	Wyniki ankiet z 70 ośrodków				
			liczba zatrudnionych na 31.12.80 r.	liczba zatrudnionych na 31.01.82 r.	liczba osób przewidzianych do zwolnienia w 1982 r.	stosunek rubryk 5:4 (%)	stosunek rubryk 6:5 (%)
1	2	3	4	5	6	7	8
01	OGÓŁEM (02+10)	53 913	5 226	4 553	650	87	14
02	Pracownicy działalności podstawowej	48 530	4 549	3 928	563	86	12
03	Projektanci systemów i analitycy	6 405	867	813	155	93	20
04	Programiści	6 530	607	494	165	81	23
05	Operatorzy maszyn	17 206	1 247	1 101	114	88	10
06	Operatorzy systemów	2 241	209	266	42	127	16
07	Konserwatorzy	5 241	506	555	40	109	7
08	Kontrolerzy WE-WY	3 759	170	196	20	115	10
09	Pozostali pracownicy działalności podstawowej	7 138	943	503	55	53	11
10	Pracownicy działalności pomocniczej	5 383	677	635	87	93	14
11	Pracownicy adm.-biur.	3 212	413	359	50	89	14
12	Pozostali pracownicy działalności pomocniczej	2 171	264	266	37	100	14

Tabela 3. Zatrudnieni w 70 ośrodkach obliczeniowych przewidziani do zwolnienia w 1982 r. wg województw

Wybrane województwa	Zatrudnienie ogółem 31.12.80 r.)	Zatrudnienie w ośrodkach zwalniających (31.01.82 r.)	Liczba pracowników przewidzianych do zwolnienia w 1982 r.	Stosunek rubryk 4:3 (%)	Stosunek rubryk 4:2 (%)
1	2	3	4	5	6
St. warszawskie	13 466	819	195	23	1,5
Bielskie	878	47	8	19	0,9
Bydgoskie	2 082	85	9	11	0,4
Częstochowskie	568	25	9	36	1,6
Gdańskie	3 039	349	40	11	1,3
Katowickie	7 839	881	7	1	0,1
Kieleckie	1 204	59	12	20	1,0
Krakowskie	3 237	760	64	8	2,0
Łódzkie	3 539	496	101	20	2,9
Olsztyńskie	1 181	132	8	6	0,7
Opolskie	779	39	11	28	1,4
Piotrkowskie	159	11	3	27	1,9
Poznańskie	2 755	50	30	60	1,1
Radomskie	774	89	4	4	0,5
Rzeszowskie	860	13	5	38	0,6
Siedleckie	122	1	1	100	0,8
Skierniewickie	204	10	4	40	2,0
Szczecińskie	1 494	40	40	100	2,7
Toruńskie	682	85	16	19	2,3
Wałbrzyskie	577	233	7	3	1,2
Wrocławskie	3 922	305	71	23	1,8
Zamojskie	23	16	2	13	8,7
Zielonogórskie	706	23	3	13	0,4
Razem	50 090	4563	650	14	1,3

Niepokój budzi wysoki procent pracowników przewidzianych do zwolnienia w 1982 r. w jednostkach ministerstw Rolnictwa, Przemysłu Lekkiego oraz Przemysłu Spożywczego i Skupu, które w aktualnej sytuacji kraju — wydawałoby się — powinny się rozwijać.

PRZYCZYNY ZWOLNIEŃ

Przyczyny zwolnień pracowników wiążą się z sytuacją ośrodków informatycznych. Generalnie można je sklasyfikować następująco:

Niewłaściwe przepisy prawne

Obowiązujące dotychczas zasady finansowania przedsiębiorstw nie stwarzały zachęty do wdrażania postępu organizacyjnego. Również istniejące relacje między kosztami sprzętu informatycznego i wartością pracy ludzkiej w warunkach samofinansowania przedsiębiorstw przemawiają przeciwko stosowaniu informatyki (odwrotnie niż to ma miejsce w wielu krajach zachodnich). Krytycznie odnoszono się też do obowiązującego taryfikatora płac w ośrodkach informatycznych, który w obecnej sytuacji zapewnia płace niższe od średniej krajowej.

Brak środków finansowych

Chodzi tu o środki na prace naukowo-badawcze i projektowe, które w tym roku są znacznie ograniczone, oraz o Fundusz Postępu Techniczno-Ekonomicznego. Ośrodki informatyczne, które pracują w oparciu o te środki finansowe mają szczególne trudności.

Reorganizacja gospodarki

Likwidacja zjednoczeń oznacza zlikwidowanie jednostek informatycznych, które wchodziły w ich skład. Następuje też zmniejszenie liczby zamówień na usługi ze strony dotychczasowych zjednoczeń i tym samym — ograniczenie zatrudnienia w tych ośrodkach.

Brak materiałów eksploatacyjnych i części zamiennych

Dotyczy to szczególnie materiałów reglamentowanych, np. papieru tabulogramowego do drukarek, oraz limitów dewizowych.

Zła jakość sprzętu komputerowego i wysoki jego koszt

Ogólny brak efektów stosowania informatyki

W ocenie przedsiębiorstw zwalniających pracowników nastąpił brak wystarczających efektów ekonomicznych w wyniku zastosowania systemów informatycznych, co z kolei spowodowało zmniejszenie zapotrzebowania na te systemy.

• • •

Jak wykazała ankieta, ruch kadrowy wywołany reorganizacją gospodarki oraz sytuacją kryzysową nie stanowi istotnego zagrożenia dla całości informatyki. Bardziej niepokojącym zjawiskiem jest stały odpływ ludzi wywołany malejącą atrakcyjnością pracy w tej branży.

Malejące zatrudnienie w ośrodkach, które odpowiedziały na ankietę, jest — co ważne — kontynuacją trendu z roku poprzedniego. Można więc przypuszczać, że istnieją konkretne, niejako dodatkowe przyczyny redukowania ośrodków. Do tych przyczyn można zaliczyć np. niewłaściwe zlokalizowanie ośrodka — w miejscu, gdzie brakuje odpowiednio dużego zapotrzebowania na usługi informatyczne. Może to być też zła koncepcja organizacyjna ośrodka, hamująca jego rozwój oraz brak właściwego przygotowania środowiska do stosowania informatyki. Przyczyny te są obiektywne i występują niezależnie od sytuacji w gospodarce.

GRAŻYNA KLAJN-ZIENKIEWICZ
Sekretariat Komitetu Informatyki

Tabela 4. Zatrudnieni w ośrodkach obliczeniowych przewidziani do zwolnienia w układzie resortowym

Ministerstwa	Zatrudnienie ogółem 31.12.80 r.)	Zatrudnienie w ośrodkach zwalniających 31.01.82 r.)	Liczba osób przewidzianych do zwolnienia w 1982 r.	Stosunek rubryk 4:3 (%)	Stosunek rubryk 4:2 (%)
1	2	3	4	5	6
NSzWiT	8462	775	90	12	1,1
Energetyki i Energii Atomowej	2312	128	12	9	0,5
Górnictwa	2611	881	7	1	0,3
Hutnictwa	2471	14	2	14	0,1
Leśnictwa i Przemysłu Drzewnego	577	25	7	28	1,2
Przemysłu Maszynowego	7849	469	140	29	1,8
Przemysłu Lekkiego	1771	315	97	29	5,5
Przemysłu Chemicznego	2519	203	37	18	1,5
Przemysłu Spożywczego i Skupu	1197	138	40	29	3,3
Przemysłu Maszyn Ciężkich i Rolniczych	2952	92	13	14	0,4
Rolnictwa	858	250	62	25	7,2
Komunikacji	3313	241	18	8	0,6
Budownictwa i Przemysłu Materiałów Budowlanych	4103	338	48	14	1,2
Łączności	2102	145	6	4	0,3
Administracji, Gospodarki Terenowej i Ochrony Środowiska	1394	267	27	10	1,9

POLSKIE TOWARZYSTWO INFORMATYCZNE

Z prac PTI

• Jak już wspominaliśmy na łamach INFORMATYKI, powstaje stałe, organizowane przez Andrzeja J. Bliklego, **seminarium PTI.** Tematem tego seminarium będą nowe, ważne osiągnięcia w dziedzinie informatyki. Dopiero dziś możemy podać bliższe dane. Począwszy od maja br., seminarium będzie się odbywać w Pałacu Kultury i Nauki, X piętro, sala nr 1044, w ostatnie środy każdego miesiąca, o godzinie 16^{00}.

Oto terminarz i tematy pierwszych spotkań:

26 maja — **„Język programowania Ada"** (Andrzej Salwicki)

30 czerwca — **„Język Loglan. Ogólna charakterystyka, pokaz funkcjonowania"** (Antoni Kreczmar),

a po przerwie wakacyjnej:

29 września — **„Rekonfigurowalna maszyna ze zmienną strukturą — TRAC"** (Romuald Marczyński)

24 października — **„O metakompilacji"** (Jan Borowiec)

PTI zaprasza swych członków i inne zainteresowane osoby do wzięcia udziału w tym seminarium.

• Przypominamy, że szczegółowe informacje o tworzonych sekcjach i komitetach PTI, wraz z nazwiskami ich opiekunów, znajdują się w poprzednim numerze INFORMATYKI.

• Na posiedzeniu Zarządu Głównego PTI w dniu 22 kwietnia br., podjęto uchwałę dotyczącą rozpoczęcia **działalności Towarzystwa w sferze usług informatycznych.** Powołano w tym celu specjalną Komisję, w składzie: J. Muszyński (przewodniczący), R. Dąbrówka (członek) i M. Muraszkiewicz (członek, przedstawiciel Komisji Rewizyjnej). Do zadań Komisji — w ramach statutowej działalności PTI — należy m.in.:

— inicjowanie i prowadzenie różnorodnych prac badawczych i wdrożeniowych, ze szczególnym uwzględnieniem projektowania i programowania systemów informatycznych

— organizowanie wszelkiego rodzaju doradztwa i ekspertyz

— organizowanie działalności szkoleniowej.

Przewiduje się, że działalność PTI dotycząca usług informatycznych będzie prowadzona przy współpracy z innymi towarzystwami i stowarzyszeniami, a także jednostkami gospodarczymi; m.in. na ich zlecenie członkowie PTI będą mogli wykonywać różne prace.

Zarząd Główny PTI rozważa ponadto możliwość podjęcia przez Towarzystwo wyodrębnionej działalności gospodarczej.

Osoby pragnące nawiązać z nami kontakt w tej sprawie prosimy o listy kierowane pod adresem Zarządu Głównego PTI, z dopiskiem na kopercie: „Komisja Usług".

• Z miesiąca na miesiąc zwiększa się liczba członków naszego Towarzystwa.

• PTI prezentuje sprawy polskiej informatyki w organach rządowych. Powołany w wyniku starań Towarzystwa do Zespołu 11 (Postęp techniczny) Komisji ds. Reformy Gospodarczej, Jerzy Kisielnicki przedstawił na tym forum referat na temat roli oraz kierunków rozwoju informatyki i jej zastosowań w reformowanej gospodarce Polski. W opracowaniu tym uzasadniono konieczność odbudowy i rozwoju informatyki mimo istniejącego kryzysu gospodarczego, zwracając m.in. uwagę na to, że wstrzymanie w chwili obecnej informatyzacji kraju uniemożliwiłoby w niezbyt dalekiej przyszłości, za jakieś 10 lat, sprawną wymianę informacji, działalność handlową, uczestniczenie w międzynarodowym podziale pracy, prowadzenie naprawdę opłacalnego eksportu. W referacie wskazano także kierunki zastosowań informatyki dotychczas nie rozwijane, zaproponowano włączenie ośrodków obliczeniowych do konkretnych prac związanych z reformowaniem gospodarki (m.in. utworzenie mechanizmu informacyjnego dla racjonalnych powiązań kooperacyjnych towarzyszących wytwarzaniu wyrobów). Sprecyzowano także kierunki działań natury organizacyjnej i ekonomicznej, mających na celu powstrzymanie degradacji informatyki i zapewniających jej prawidłowy rozwój oraz racjonalizację zastosowań.

• Z inicjatywy PTI podjętej w Komitecie Informatyki PAN, a także przy poparciu Zarządu Informatyki Wojska Polskiego, wystąpiono do organów rządowych o powołanie Państwowej Agencji Informatyki. Oddziałując na samodzielne i samofinansujące się ośrodki informatyki głównie za pomocą mechanizmów ekonomicznych (kredyty, podatki, cła itp.), Państwowa Agencja Informatyki stałaby się reprezentantem ogólnonarodowego interesu w stosunkach z organizacjami gospodarczymi. Jej podstawowe zadanie — to sprawowanie państwowego mecenatu nad rozwojem informatyki i jej zastosowań.

Przypominamy nasz adres do korespondencji: Zarząd Główny Polskiego Towarzystwa Informatycznego ul. Jasna 14/16, pok. 338, 00-041 Warszawa

Sekretariat (w godz. 9—15) prowadzi p. Anna Pykało tel. 26-82-61 wew. 122.

(B.O.)

KONFERENCJE

Convention Informatique '82

W dniach 20—24 września br. odbędzie się w Paryżu kolejna doroczna międzynarodowa konferencja informatyczna „Convention Informatique". Konferencja ta coraz silniej umacnia swoją pozycję, wysuwając się na czołowe miejsce wśród wielkich europejskich imprez informatycznych. Potwierdzeniem tego było 3175 uczestników w ub. r., z czego ok. 700 stanowili delegaci zagraniczni.

Hasłem przewodnim konferencji tegorocznej jest „Nowa informatyka" („Une nouvelle informatique"), które ma zaakcentować wszystkie przemiany, jakie zaszły w ostatnich latach w wyniku postępu technicznego (teleprzetwarzanie, mikroprocesory, sieci komputerowe, bazy danych, komunikacja w językach naturalnych, sztuczna inteligencja, robotyka, automatyzacja biur). Program konferencji, akcentujący konieczność uwzględnienia tych przemian, przewiduje zgodnie z tradycyją podział obrad na następujące cztery sekcje problemowe:

1) **technologia** (elementy, architektura systemów, mikroprocesory, niekonwencjonalne we/wy, zarządzanie informacjami, transmisja danych, produkcja oprogramowania)

2) **zastosowania dziedzinowe** (gospodarka żywnościowa, bankowość, automatyzacja biur, handel detaliczny, sterowanie produkcją, administracja i organizacje społeczne, teleprzetwarzanie)

3) **aspekty społeczne i prawne** (wpływ na zatrudnienie, organizację i warunki pracy, kształcenie i szkolenie wspomagane, praca zdalna, prawo komputerowe)

4) **aspekty ekonomiczne** (zarządzanie informatyką, skutki ekonomiczne nowych metod, nowe specjalizacje informatyki profesjonalnej).

W br. organizatorzy dołączyli do dotychczasowych języków urzędowych konferencji (francuskiego i angielskiego) również język niemiecki.

W.K.

zjednoczenie informatyki ZETO

Badania ankietowe użytkowników systemów informatycznych

Celem prowadzonych badań ankietowych jest podniesienie jakości usług świadczonych przez sieć ZETO, na etapie wdrażania i pierwszego okresu eksploatacji systemów informatycznych, a także zebranie uwag na temat potrzeb użytkowników i rozwiązań technologicznych w projektowanych systemach. Badaniami, przedstawianymi poniżej, zostały objęte systemy powielarne, głównie wspomagające proces zarządzania, wdrażane przez sieć ZETO w latach 1979—1980. W oparciu o ankiety nadesłane przez użytkowników, badania opracowało ZETO Szczecin.

Za systemy powielarne uznano takie, które ze względu na zastosowane sparametryzowane rozwiązania projektowo-programowe oraz przewidziany maksymalny zakres informacyjny — mogą być zastosowane u wielu użytkowników w różnych branżach.

Przyjęta w sieci ZETO zasada projektowania systemów powielarnych wynika głównie z następujących przesłanek:

- lepszego dopracowania technologii przetwarzania
- zmniejszenia kosztów zakupu systemu przez indywidualnego nabywcę
- wdrażania systemów już sprawdzonych u innych użytkowników, po wyeliminowaniu błędów, tkwiących zwykle w oprogramowaniu, a wykrywanych dopiero przy pierwszych wdrożeniach
- skrócenie czasu wdrażania systemu.

Badaniom poddano systemy opracowane zarówno przez zakłady ZETO, jak i przez inne jednostki autorskie. Chociaż dane uzyskane z ankiet zawierać mogą pewne niedokładności, to jednak wydaje się, że materiał w nich zawarty obejmuje wiele interesujących danych faktograficznych i może stanowić podstawę do sprecyzowania szeregu istotnych uwag i wniosków na ten temat.

CHARAKTERYSTYKA ANKIET I OBSZARU BADAŃ ANKIETOWYCH

W badaniach za rok 1979 poddano ocenie 434 systemy informatyczne, natomiast za rok 1980 — 328. W oparciu o nadesłane przez użytkowników ankiety, badania przeprowadzono:

- za rok 1979 w dwóch etapach
- za rok 1980 w jednym etapie.

Pierwszy etap badań, którym objęte zostały systemy wdrożone w roku 1979 i 1980, dotyczył wdrożenia systemu i zawierał oceny następujących zagadnień:

- ogólnego zadowolenia z wdrożonego systemu
- przeszkolenia personelu użytkownika w zakresie eksploatacji systemu
- przejrzystości i czytelności dokumentacji użytkowej
- terminowości wdrożenia systemu.

Każde z powyższych zagadnień oceniane było przez użytkowników w skali od 4 do 1:
4 — ocena bardzo dobra
3 — ocena dobra
2 — ocena dostateczna
1 — ocena niedostateczna.

Drugi etap badań (dotyczący tylko wdrożeń roku 1979) miał na celu weryfikację ocen systemów uzyskanych w pierwszym etapie, po co najmniej sześciomiesięcznej eksploatacji ciągłej. Badania te dotyczyły następujących zagadnień:

- terminowości wdrożenia systemu
- oceny efektów uzyskanych z eksploatacji systemu
- oceny przydatności informacji, uzyskiwanych z systemu
- oceny dokumentacji eksploatacyjnej oraz terminowości uzyskiwanych wyników obliczeń.

W drugim etapie badań odpowiedzi udzieliło tylko 72,6% respondentów objętych pierwszym etapem, w tym pewna część ankiet wypełniona była częściowo, a kilka ankiet w ogóle nie zawierało odpowiedzi na zawarte w niej pytania. Przypadki takie dotyczyły w szczególności systemów autorstwa ośrodków branżowych lub resortowych. Użytkownicy tych systemów tłumaczyli swoje stanowisko tym, że nie są kompetentni do oceny systemów stworzonych przez ich jednostkę nadrzędną.

Na tym etapie badań wystąpiły znaczne różnice w ocenach użytkowników odnośnie przejrzystości i czytelności dokumentacji eksploatacyjnej w porównaniu do ocen z pierwszego etapu badań. Oceny drugiego etapu były w tej kwestii mniej korzystne w porównaniu do pierwszych ocen. Ponieważ półroczny okres eksploatacji jest zbyt krótki dla uzyskania pełnych efektów zakładanych przez systemy, dlatego też uzasadnionym wydaje się wniosek, by w przyszłości takimi badaniami objęte zostały wszystkie systemy eksploatowane przez dłuższy czas.

W oparciu o obowiązujący GUS-owski dziedzinowy podział zastosowań, najwięcej wdrożeń w roku 1979 i 1980 miały systemy dotyczące:

	Rok 1979		Rok 1980	
	liczba	%	liczba	%
rachunku kosztów i rozliczeń finansowych	122	28,1	54	16,5
gospodarki materiałowej	114	26,3	73	22,2
oprogramowania narzędziowego (m.in. pakiet badania i dokumentowania programów ZETOFLOW II, zmodyfikowany translator języka BASIC, egzekutor EGRM z systemem symulacji drukarki wierszowej, vademecum programisty, pakiet konwersji PAKON OS itp.)	52	12,0	57	17,4
gospodarki środkami trwałymi	46	10,6	45	13,7
opracowań statystycznych	29	6,7	27	8,2

Bardzo mało było natomiast w 1979 r. wdrożeń w zakresie:

	Liczba	%
informacji naukowo-technicznej	2	0,5
analiz ekonomicznych	6	1,4
planowania produkcji	9	2,1%
technicznego przygotowania produkcji	11	2,5%

a w 1980 r. w zakresie:

	Liczba	%
planowania i kontroli działalności podstawowej oraz pomocniczej	1	0,3%
technicznego przygotowania produkcji	6	1,8%
obliczeń optymalizacyjnych	6	1,8%
gospodarki wyrobami gotowymi i towarami	9	2,7%

Małą liczbę wdrożeń miały również systemy z dziedziny zatrudnienia i płac (1979 r. — 18, a w 1980 r. — 16), przy czym w większości dotyczą one raczej zagadnień zatrudnienia, niż płac. Wynika to z jednej strony z faktu, że dotąd stosunkowo mało jest uniwersalnych, powielarnych systemów zatrudnieniowo-płacowych, z drugiej zaś, że — systemy te są bardziej pracochłonne przy wdrożeniu oraz wymagają przestrzegania przez użytkownika znacznej dokładności i terminowości w zakresie wykonywania określonych czynności — zarówno na etapie wdrażania, jak i podczas eksploatacji systemu.

Spośród typowych powielarnych systemów najwięcej wdrożeń miały w 1979 r.:

Nazwa systemu	Jednostka autorska	Typ komputera	Liczba wdrożeń
SEMO — gospodarka materiałowa	ZETO Bydgoszcz	ODRA	25
CZYNSZ — rozliczanie członków spółdzielni mieszkaniowych	ZETO Olsztyn	ODRA	24
GM-KTM — gospodarka materiałowa	ZETO Wrocław	ODRA	18
GOSMAT — gospodarka materiałowa	ZETO Katowice	RIAD	14
PESTO — środki trwałe	ZETO Bydgoszcz	ODRA	9

w 1980 r.:

ZETOFLOW — pakiet badania i dokumentowania	ZETO Łódź	ODRA	15
F-K — system finansowo-kosztowy	ZETO Bydgoszcz	ODRA	12
F-K JS — system finansowo-kosztowy	ZETO Bydgoszcz	RIAD	12
PESTO — środki trwałe	ZETO Bydgoszcz	ODRA	12
GOSMAT — gospodarka materiałowa	ZETO Katowice	RIAD	10

W 1979 r. z ogólnej liczby 434 wdrożonych systemów — 100 (23%) stanowiły systemy eksploatowane na komputerach Jednolitego Systemu, natomiast w 1980 r. z 328 — 132 systemy (40%). Świadczy to o wzrastającym ukierunkowaniu prac projektowo-programowych i wdrożeniowych na ten rodzaj sprzętu komputerowego.

Wzrost liczby wdrożeń systemów przeznaczonych do eksploatacji na komputerach Jednolitego Systemu idzie w parze ze zwiększającym się wyposażeniem ośrodków ZETO w tego typu maszyny cyfrowe.

Liczba wdrożonych systemów na komputery JS EMC w podziale wg dziedzin zastosowań przedstawia się następująco:

	1979 r.	1980 r.
rachunek kosztów i rozliczenia finansowe	34	24
gospodarka materiałowa	28	28
środki trwałe	14	18
obliczenia optymalizacyjne	8	4
systemy bliżej nie sprecyzowane	6	31
gospodarka wyrobami gotowymi	3	2
analizy ekonomiczne	3	13
zatrudnienie i płace	2	2
planowanie produkcji	2	—
informacja naukowo-techniczna	—	7
opracowania statystyczne	—	3

W 1979 r. najwięcej systemów informatycznych wdrożono w resortach:

- **przemysłu maszynowego** — 86 — 19,8%
- **administracji i ochrony środowiska** — 50 — 11,5%
- **leśnictwa i przemysłu drzewnego** — 39 — 9,0%
- **hutnictwa** — 30 — 6,9%

natomiast w 1980 r. w resortach:

- **nauki, szkolnictwa wyższego i techniki** — 46 — 14,0%
- **przemysłu maszynowego** — 46 — 14,0%
- **leśnictwa i przemysłu drzewnego** — 26 — 7,9%
- **przemysłu lekkiego** — 21 — 6,4%

W 1979 r. bardzo mało było wdrożeń w następujących resortach:

- **spraw wewnętrznych** — 1 — 0,2%
- **zdrowia i opieki społecznej** — 2 — 0,4%
- **łączności** — 2 — 0,4%
- **handlu zagranicznego i gospodarki morskiej** — 3 — 0,7%
- **handlu wewnętrznego i usług** — 4 — 0,9%

a w 1980 r. w resortach:

- **kultury i sztuki** — 1 — 0,3%
- **obrony narodowej** — 1 — 0,3%
- **górnictwa** — 3 — 0,9%
- **handlu wewnętrznego i usług** — 3 — 0,9%

WYNIKI BADAŃ

Wyniki uzyskane w pierwszym etapie badań ankietowych przedstawiają się następująco:

Ocena w zakresie ogólnego zadowolenia z wdrożonego systemu

Oceny	1979 r.		1980 r.	
bardzo dobre	269	62,1%	180	55,0%
dobre	137	31,6%	114	34,9%
dostateczne	27	6,3%	31	9,5%
niedostateczne	—	—	2	0,6%

Ocena przeszkolenia personelu użytkownika na potrzeby eksploatacji systemu

Oceny	1979 r.		1980 r.	
bardzo dobre	269	69,8%	181	57,3%
dobre	109	27,7%	110	34,8%
dostateczne	16	3,8%	23	7,3%
niedostateczne	3	0,7%	2	0,6%

W roku 1979 w 10 przypadkach użytkownicy nie wystawili ocen (ponieważ szkolenia nie przeprowadzono), a w roku 1980 — w 12 przypadkach, nie podając jednak przyczyny.

Ocena przejrzystości i czytelności dokumentacji

Oceny	1979 r.		1980 r.	
bardzo dobre	297	69,4%	200	61,0 %
dobre	120	27,7%	102	31,1%
dostateczne	16	3,7%	23	7,0%
niedostateczne	1	0,2%	3	0,9%

W poszczególnych rodzajach ocen w 1980 r., w porównaniu do roku 1979, wystąpił spadek udziału ocen bardzo dobrych.

Ocena w zakresie terminowości wdrożenia systemu

W tym punkcie ankiety respondenci stwierdzili, że opóźnienie wdrożenia systemów w stosunku do terminów umownych wystąpiło w 1979 r. w 12 przypadkach na łącznie 416 dni, w 1980 r. w 16 przypadkach na łącznie 1013 dni. Opóźnienia te — jak wynika z odpowiedzi — były spowodowane różnymi przyczynami, subiektywnymi i obiektywnymi. Tym niemniej wielkości te są zbyt duże i wymagałyby przeprowadzenia oddzielnej, szczegółowej analizy.

W drugim etapie badań (za 1979 r.), mimo uzyskania zaledwie 72,6% odpowiedzi w stosunku do łącznej liczby wdrożonych systemów, wyniki można jednak uznać w pełni za reprezentatywne, a tym samym pozwalające na uogólnienia i wyprowadzenie pewnych prawidłowości i wniosków.

Analiza ocen dotyczących uzyskanych efektów z eksploatowanych systemów wykazuje, że największe korzyści osiągnięto w zakresie doskonalenia organizacji w przedsiębiorstwie (87,5% odpowiedzi) oraz zwiększenia zakresu informacji (85,4% odpowiedzi). Mniejsze natomiast korzyści (71,1%) osiągnięto w zakresie przyspieszania uzyskiwania informacji, co na pewno jest wielkością niewystarczającą. Wskaźnik ten będzie z pewnością wzrastał w miarę wdrażania do eksploatacji systemów teleprzetwarzania.

Pewne efekty osiągnięto w wyniku wdrożenia systemów w zakresie zmniejszenia zatrudnienia. Pomimo niezbyt długiego okresu ich eksploatacji nastąpiło łączne zmniejszenie zatrudnienia o 135 etatów.

Największe oszczędności etatowe osiągnięto w dziedzinach:

- planowania i kontroli działalności podstawowej oraz pomocniczej — 2 etaty/1 system
- gospodarki materiałowej — 0,6 etatu/1 system
- rachunku kosztów i rozliczeń finansowych — 0,3 etatu/1 system

Z wypowiedzi respondentów wynika, że wdrożone systemy tylko w 19,1% przypadków spowodowały obniżenie kosztów przetworzenia informacji, natomiast pewna część użytkowników stwierdziła, że zamiast obniżenia kosztów, nastąpiło ich podwyższenie (!) w porównaniu do uzyskiwania informacji systemem tradycyjnym. Można jednak sądzić, że efekty osiągane w zakresie: doskonalenia organizacyjnego, zwiększenia zakresu informacyjnego, przyspieszenia procesu uzyskiwania informacji oraz zmniejszenia liczby etatów — rekompensują niską ocenę uzyskanej obniżki kosztów przetworzenia informacji po wdrożeniu systemu informatycznego w przedsiębiorstwie. Zagadnienie to nabiera szczególnego znaczenia w warunkach reformy gospodarczej i wymaga podjęcia odpowiednich działań przez przedsiębiorstwa ZETO. Większość respondentów wyraziła pozytywne opinie o przydatności otrzymywanych informacji. Dokładność informacji stwierdza 94,0% respondentów, kompletność 90,0%, aktualność 85,0%, wszechstronność 79,9%, a przydatność informacji do podejmowania decyzji 78,3%. Wskaźniki powyższe można uznać za zadowalające.

Niezbyt korzystnie kształtują się natomiast oceny użytkowników dotyczące terminowości otrzymywania wyników obliczeń. Wskaźnik 81,8% pozytywnych odpowiedzi świadczy o brakach w organizacji współpracy ośrodków obliczeniowych i użytkowników, występujących w pierwszym okresie po wdrożeniu systemu. Za taki stan rzeczy najczęściej winę ponoszą obydwaj partnerzy: klient i ośrodek obliczeniowy. W szczególności — najczęściej występującą przyczyną takiego stanu rzeczy jest nieterminowe dostarczanie danych przez użytkownika, bądź dostarczanie danych z dużą liczbą błędów, co w konsekwencji wpływa w sposób zasadniczy na przedłużanie terminów wykonania prac obliczeniowych przez ośrodek. Czasami też nieterminowe wykonanie obliczeń wynika z przyczyn obiektywnych lub subiektywnych, związanych z działalnością ośrodka obliczeniowego, na które użytkownik nie ma żadnego wpływu. Obydwie strony powinny zatem dążyć do usunięcia przyczyn powodujących opóźnienie wyników przetwarzania.

Przy szczegółowej analizie wyników ankiet można zauważyć znaczną różnorodność odpowiedzi odnoszących się do tego samego systemu. Różnorodność ta — zdaniem autorów niniejszego artykułu — wynika z następujących przyczyn:

- niejednakowych warunków i układów współpracy pomiędzy użytkownikiem a autorami systemu i ośrodkiem wykonującym obliczenia
- różnego poziomu przygotowania fachowego kadry u użytkownika, co warunkuje możliwości pełnego i wszechstronnego krzystania z wyników systemu
- z przesłanek mających decydujący wpływ na wdrożenie systemu w przedsiębiorstwie (czy wdrożenie wynikało z konkretnych potrzeb i inicjatywy użytkownika, czy też na skutek nakazu jednostki nadrzędnej).

WNIOSKI

Przedstawione wyżej wyniki badań ankietowych użytkowników systemów informatycznych dają ogólny obraz opinii części odbiorców usług sieci ZETO. Z tego powodu materiał ten powinien być szczególnie pożyteczny dla przedsiębiorstw ZETO.

Uzyskane opinie użytkowników można pogrupować również wg poszczególnych systemów informatycznych oraz wdrażających je przedsiębiorstw ZETO. Takie cząstkowe wyniki badań mogą być udostępnione na życzenie zainteresowanych ZETO i dawać bardziej szczegółowe wskazówki odnośnie:

- ulepszenia rozwiązań oraz dokumentacji użytkowej niektórych systemów informatycznych
- poprawy organizacji procesu wdrożeniowego.

Z drugiej strony — po analizie takich cząstkowych wyników badań — ośrodki ZETO mogą zgłosić propozycje modyfikacji, bądź rozszerzenia ankiety oraz sposobu zbierania odpowiedzi ankietowych.

Uważamy, że badania ankietowe użytkowników systemów informatycznych powinny być kontynuowane również w latach następnych. Pozwalają one bowiem prognozować sytuację na rynku usług informatycznych, która w warunkach reformy gospodarczej kształtować będzie sytuację ekonomiczną przedsiębiorstw ZETO.

KAZIMIERZ DUDEK
JÓZEF DZIEDZICKI

Przemysł komputerowy USA w roku 1980

Miesięcznik DATAMATION ogłasza co roku listę stu największych firm komputerowych w USA przedstawiającą kolejność dochodów tych firm. Przytoczymy tu pierwszą dziesiątkę z tej listy, a także omówimy — za cytowanym źródłem — tendencje wynikające z analizy pełnej listy.

Na tle ogólnej recesji przemysł komputerowy USA wykazywał w 1980 r. dość dużą dynamikę. Dochody firm z listy DATAMATION wzrosły (w porównaniu z dochodami w 1979 r. o 20,4% — do kwoty 55,6 mld dolarów, przy czym procentowy udział IBM zmalał do 38,4% (z 40% w 1979 r. i 43,2% w 1978 r). Bez IBM wzrost dochodów wyniósł 24%. Zyski brutto zmalały z 18,5% do 17%. Najszybszy rozwój wystąpił w działach mikrokomputerów, przetwarzania tekstów oraz wspomaganego projektowania i wytwarzania (CAD/CAM). Nastąpiły również znaczne zmiany na liście w stosunku do ubiegłego roku. Ze względu na roszerzenie pojęcia informatyki o dziedzinę przetwarzania tekstów i transmisji danych oraz zmiany organizacyjne w przedsiębiorstwach — na liście pojawiło się 19 nowych firm.

Pierwsza dziesiątka firm realizowała 71% dochodów całej listy oraz zatrudniała 75% personelu, przy czym IBM, NCR, SPERRY-UNIVAC, HONEYWELL i HEWLETT-PACKARD utrzymały swoje dotychczasowe pozycje, natomiast na pozostałych miejscach nastąpiły zmiany.

Biorąc pod uwagę bezwzględny przyrost dochodu — na pierwszym miejscu znajduje się IBM (3029 mln $), a następnie DEC (712), CONTROL DATA (518), HEWLETT-PACKARD (430) i NCR (312). Dochód o ponad 100 mln dolarów powiększyło łącznie 16 firm.

Wśród wymienionych dziesięciu firm największy procentowy przyrost dochodu wykazał HEWLETT-PACKARD (37,5%), jednakże największą dynamikę wykazują małe firmy, wśród których przyrost 208,5% wykazał SANDERS ASSOCIATES (częściowo dzięki wchłonięciu firmy CALCOMP) i awansował z 82 pozycji w 1979 r. na 49 w 1980 r. W następnej kolejności są: APPLE COMPUTER (175,1% i skok z 74 pozycji na 47), PHILIPS INFORMATION SYSTEMS (100%, spoza listy na 98 pozycję), TANDEM (93,9% z 66 na 53 pozycję) i INTERGRAPH (91,3% z 99 na 90 pozycję).

Tylko jedna firma (NORTHERN TELCOM) wykazała spadek dochodu (o 26%), podczas gdy w 1979 r. spadek taki miało aż pięć firm. Wśród firm o najniższym względnym wzroście dochodu figurują: BURROUGHS (1,5%), CENTRONICS i ITT (po 2%) oraz MEMOREX (4,3%).

Zyski brutto (bez uwzględniania dywidend i podatków) wzrosły w 1980 r. o 10% — do 8,5 mld dolarów, z czego 62% przypada na IBM (5231 mln, wzrost 12,5% od 1979 r.). Na dalszych miejscach znajdują się DEC (434 mln, 32,6%), NCR (390 mln, 12%), HEWLETT-PACKARD (261 mln, 52,7%) i SPERRY UNIVAC (255 mln, 23,7%). BURROUGHS wykazał spadek zysku o 68,2%, natomiast najszybszy wzrost wykazywały firmy WANG (65%) i wspomniany HEWLETT-PACKARD.

Nakłady inwestycyjne największe były w IBM (1985 mln $, 28,2% wzrostu w porównaniu do 1979 r), następnie w DEC (321 mln, 56,4%), CONTROL DATA (298 mln, 42,8%), NCR (156 mln, 36,2%) i HEWLETT-PACKARD (148 mln, 28,7%). Średni wzrost tych składników dla całej listy w porównaniu do 1979 r. wyniósł 37,1% (w 1979 r. 30%, w tym IBM — 35%).

Przemysł komputerowy, ze względu na swą specyfikę, przeznacza szczególnie duże nakłady na badania. Podane poniżej kwoty obejmują wydatki wyłącznie na te wyroby i usługi, które nie zostały jeszcze wprowadzone na rynek. Wyniosły one w 1980 r. 3,7 mld dolarów i wzrosły o 21,5% w porównaniu do 1979 r., kiedy to analogiczny wzrost wyniósł 18%. Wydatki na badania stanowiły 7,6% sum uzyskanych ze sprzedaży (7,5% w 1979 r.). Przedsiębiorstwa specjalizujące się w dziedzinie przetwarzania tekstów wykazały 102% wzrostu wydatków na badania, natomiast w dziedzinie wspomaganego projektowania i wytwarzania — 78%, przy czym większość tych wydatków dotyczy sprzętu.

Zatrudnienie w większości firm (reprezentujących 93% dochodów) wyniosło w 1980 r. 863 tys. osób i w porównaniu do roku poprzedniego wzrosło zaledwie o 7,1%, co oznacza znaczny wzrost wydajności określanej wskaźnikiem dochodu na jednego zatrudnionego — z 53 600 do 60 100 $. IBM zatrudnia tylko 32% pracowników przemysłu komputerowego (a więc mniej niż procentowy udział tej firmy w dochodach i zyskach) i w 1980 r. zwiększył zatrudnienie zaledwie o 2,9%, podobnie zresztą jak NCR (1,6%) oraz BURROUGHS i CONTROL DATA (po 1,4%). Natomiast DEC zwiększył zatrudnienie aż o 21%, a HEWLETT-PACKARD — o 12%.

Dochody ze sprzedaży systemów komputerowych stanowiły niespełna połowę (46,1%) wszystkich dochodów i wykazały wzrost zbliżony do ogólnego (odpowiednio 21 i 20,9%). W dochodach tych największy rozwój wykazały mikrokomputery (84,9% wzrostu), chociaż ich udział w całości dochodów zwiększył się z 0,9% w 1979 r. do 1,4% w 1980 r. Następne w kolejności dynamiki dochodów były systemy przetwarzania tekstów (63,8% wzrostu i zwiększenie udziału z 1,2 do 1,6%) oraz minikomputery (27,8% wzrostu i zwiększenie udziału z 15 do 15,9%), natomiast systemy konwencjonalne wykazały zaledwie 13,8% wzrostu, a ich udział spadł z 29 do 27,2%. Spośród innych działów największy rozwój wykazywało oprogramowanie (29% wzrostu i zwiększenie udziału w dochodach z 2,9 do 3,1%), następnie — uniwersalne urządzenia peryferyjne (26,9% wzrostu i zwiększenie udziału z 6,8 do 7,1%) oraz transmisja danych (23,1% wzrostu i zwiększenie udziału z 2 do 2,1%). Inne działy, takie jak konserwacja i obsługa, utrzymały swój dotychczasowy udział w dochodzie (odpowiednio 16 i 11,6%), natomiast specjalizowane urządzenia peryferyjne zmniejszyły ten udział (z 12,9 do 12,4%), przy wzroście tylko o 16,3%.

Największe dochody ze sprzedaży systemów przetwarzania tekstów wykazała firma WANG (252,3 mln $, co stanowi 37% jej dochodów ze sprzętu

Porównanie potencjału i pozycji rynkowej 10 największych amerykańskich firm przemysłu komputerowego

Pozycja w roku 1980	Pozycja w roku 1979	Firma	Dochód (mln $)	Procent wzrostu	Zatrudnienie
1	1	IBM	21 357	16,5	278 200
2	2	NCR	2 840	12,3	65 600
3	4	CONTROL-DATA	2 791	22,8	49 000
4	6	DEC	2 743	35,0	60 000
5	5	SPERRY-UNIVAC	2 552	12,4	47 435
6	3	BURROUGHS	2 478	1,5	57 300
7	7	HONEYWELL	1 636	12,5	29 000
8	8	HEWLETT-PACKARD	1 577	37,5	28 000
9	10	XEROX	770	35,1	—
10	9	MEMOREX	686	4,3	10 700

komputerowego i wzrost o 112% w porównaniu do 1979 r.), następnie LANIER (110,1 mln, 86% i 48,6%), DEC (82,3 mln, 3% i 102,6%), XEROX (69,3 mln, 9 i 35,1%) i RAYTHEON (67,5 mln, 30 i 37,8%).

W dziedzinie transmisji danych trudno zdecydować, który sprzęt należy zaliczyć do tego działu. Można jednak na pewno stwierdzić, że największym dostawcą jest tu IBM (blisko miliard dolarów dochodu), natomiast BURROUGHS, MEMOREX i NCR są głównymi dostawcami komputerów wstępnie przygotowujących dane do przesyłania (z 85 do ponad 100 mln $). Do dziesięciu czołowych firm w tej dziedzinie można zaliczyć też dostawców modemów: RACAL, MOTOROLA, GENERAL DATA COMMUNIKATION oraz PERADYNE.

Rynek międzynarodowy jest wciąż atrakcyjny dla firm amerykańskich, mimo widocznej konkurencji Europy i Japonii oraz istniejącej recesji gospodarczej. Wzrost wpływów firm amerykańskich na tym rynku wyniósł w 1980 r. 27% w porównaniu do 16% w 1979 r. Dla ponad 20 firm dochody z rynku zagranicznego przekraczają 40% wszystkich dochodów, a dla siedmiu firm — ponad 50%. Na czele jest tu COMMODORE INTERNATIONAL (udział 75,7%, 75 mln $ z wpływów zagranicznych, 105% wzrostu), następnie NORTHERN TELECOM (odpowiednio 61%, 132 mln i spadek o 32%), MOHAWK DATA SCIENCES (58%, 161 mln, wzrost 12%), NCR (55%, 1560 mln, 14%) i C. ITOH ELECTRONICS (53%, 101 mln, spadek o 1%).

Na zakończenie warto przytoczyć krótkie charakterystyki pięciu czołowych firm.

IBM osiągnęła 57% wpływów ze sprzedaży systemów komputerowych, 28% — za oprogramowanie i usługi, a 15% za terminale. W 1980 r. poczyniono znaczne inwestycje w kwocie 6,6 mld $. Zakończono budowę ponad 370 tys. m² powierzchni produkcyjnej i laboratoryjnej oraz ponad 90 tys. m² powierzchni handlowej. W końcu roku w trakcie budowy znajdowało się ponad 740 tys. m² powierzchni produkcyjnej i laboratoryjnej oraz ok. 280 tys. m² powierzchni handlowej. Zapowiedziano nowe systemy komputerowe do zastosowań biurowych, m.in. rozproszony system do maszyny 8100 oraz system rozdziału dokumentów administracyjnych o symbolu 5520. Zapowiedziano przy tym, że w przyszłości zapewniona będzie możliwość komunikacji pomiędzy tymi systemami. Z pewnym opóźnieniem rozpoczęto dostawy systemu 38, którego pod koniec roku wytwarzano ok. 500 sztuk miesięcznie. Ceny dzierżawy wzrosły ok. 12%. Pojawiły się nowe modele rodziny 4300. Zapowiedziano również pierwszy model nowej serii H (3081) o największej szybkości obliczeniowej, zawierający 750 000 układów logicznych w kostce o objętości ok. 28 dcm³. Poprawiono parametry eksploatacyjne serii 3033 N oraz zapowiedziano najmniejszy model tej serii nazwany symbolem S. Zastosowano holografię w urządzeniu odczytującym etykiety identyfikacyjne wyrobów sprzedawnych w domach towarowych (3687). Rozpoczęto prace w dziedzinie użytkowego zastosowania techniki wizyjno-dyskowej.

NCR miała podobną strukturę dochodów (ok. 49% systemy komputerowe, 36% oprogramowanie i usługi, ok. 15% terminale). Na początku 1980 roku wystąpiły pewne perturbacje, co spowodowało spadek zysków wypłacanych akcjonariuszom o 8% w porównaniu z pierwszym półroczem 1979 r. W drugiej połowie roku sytuacja poprawiła się. Zamówienia natomiast w pierwszym półroczu były większe i zmniejszyły się dopiero pod koniec roku zarówno w USA, jak i za granicą. Firma wchłonęła APPLIED DIGITAL SYSTEMS (w 1979 r. COMTEN). Największy rozwój w sprzedaży wykazywały systemy i terminale bankowe (wzrost 27%) oraz uniwersalne systemy komputerowe (15%), natomiast dzierżawa systemów i terminali wykazywała zmniejszenie dochodów (o 5%). Mimo to firma NCR umocniła swą pozycję na rynku dzierżawy systemów zwłaszcza przeznaczonych dla obsługi domów towarowych. Zapowiedziano przystąpienie do produkcji laserów. Już drugi rok prawie 60% nakładów na badania przeznaczone jest na rozwój oprogramowania. Inwestycje obejmowały budowę fabryki systemów dostarczanych na zasadzie dzierżaw oraz rozbudowę istniejących zakładów, zwłaszcza pod kątem realizacji rozwiązań mikroelektronicznych oraz wprowadzenia usprawnień produkcyjnych. NCR rozszerza produkcję własnych elementów półprzewodnikowych i wkrótce będzie w stanie zaspokoić dwie trzecie swego zapotrzebowania na tego rodzaju elementy.

CONTROL DATA już szósty kolejny rok zwiększa swe dochody, mimo że wskaźniki przedsiębiorstwa obniżają działy pozakomputerowe (jak np. COMMERCIAL CREDIT), które nie wykazywały wzrostu. Największy wzrost wykazały urządzenia peryferyjne, które stanowiły 42% całkowitych dochodów z działalności komputerowej i w porównaniu do 1979 r. wzrosły o 31,5%, przy czym największą dynamikę zanotowano w grupie urządzeń uniwersalnych. Urządzenia te wytwarza firma we własnych zakładach bądź powiązanych z nią przedsiębiorstwach (np. MAGNETIC PERIPHERALS lub COMPUTER PERIPHERALS, w których CDC ma 60—70% udziału. Usługi komputerowe (37% dochodów z całkowitej działalności komputerowej) wzrosły w 1980 r. o 17,3%, przy czym największy wzrost (25%) wykazała działalność szkoleniowa i konsultacyjna, gdzie stosowano m.in. komputerowy system dydaktyczny PLATO, który w latach poprzednich pochłonął znaczne sumy nakładów inwestycyjnych. Trzecim odcinkiem działalności firmy są systemy komputerowe (21% dochodów) o średnim wzroście dochodów 16,7%, w ramach których największy wzrost (31%) odnotowano w zakresie dostaw systemów dla instytucji rządowych (poprawa w porównaniu do 1979 r.). Głównym przedstawicielem tej linii systemów jest CYBER 205, który pojawił się w 1980 r. jako jeden z najpotężniejszych systemów na świecie (800 mln instrukcji zmiennego przecinka na sekundę). Przedstawione tendencje w wynikach działalności firmy kontynuowane są w bieżącym roku.

DEC (DIGITAL EQUIPMENT CORPORATION) jest jedyną firmą w pierwszej dziesiątce, ktora poprawiła swą pozycję o dwa miejsca. W ciągu sześciu lat powiększyła ona swe dochody z 0,5 do 3 mld dolarów, co np. IBM zajęło dziewięć, a XEROX — siedem lat. Firma ta ma najwyższy udział systemów komputerowych w strukturze dochodów (63%), podczas gdy usługi i oprogramowanie stanowią tylko 25%, uniwersalne urządzenia peryferyjne — 7%, a terminale — 5%. Pewne zmniejszenie zamówień spowodowane ogólną recesją gospodarczą pozwoliło firmie znacznie skrocić czas oczekiwania na dostawę, co najbardziej dotknęło jej konkurentów. Oprogramowanie systemów VAX osiągnęło już poziom zadowalający większość klientów i jest nadal usprawniane. W październiku 1980 r. pojawił się model VAX-11/750, który mając 60% mocy obliczeniowej VAX-11/780 kosztuje tylko 40% jego ceny. Wpływy firmy z usług wzrosły o 39%.

SPERRY UNIVAC — ostatni z omawianych liderów tabeli — stanowi konglomerat o łącznych dochodach ponad 5 mld dolarów, z tym że działalność tej firmy obejmuje również produkcję maszyn rolniczych oraz urządzeń hydrauliki i sterowania. W działalności komputerowej największy udział miały terminale (40%), a następnie systemy komputerowe (35%) oraz usługi i oprogramowanie (25%). Większy (o 37%) przyrost wykazały dochody z rynków zagranicznych. Główne dziedziny zainteresowań firmy to zastosowania komputerów w zarządzaniu, energetyce, liniach lotniczych, handlu i obronności kraju. Szczególny duży wzrost (ok. 50%) wykazują usługi i oprogramowanie. Firma utworzyła ostatnio również dział produkcji półprzewodników, dla pokrycia własnego zapotrzebowania na układy dużej i bardzo dużej skali integracji. Chociaż na wprowadzony do produkcji w końcu 1979 r. system 1100/60 uzyskano już ponad 550 zamówień, w 1980 r. zapowiedziano produkcję nowych modeli. W grupie komputerów średniej mocy obliczeniowej System 80 uzyskał już ponad 500 zamówień i był wytwarzany w liczbie ok. 100 egzemplarzy kwartalnie. Wprowadzono także nowe modele rodziny minikomputerów V77 oraz zapowiedziano nową rodzinę mikroprocesorowych terminali rozproszonego przetwarzania UTS-4000. W końcowej fazie testowania znajduje się również kompleksowy system do zastosowań biurowych.

Opracował JAN RYŻKO
na podstawie czasopisma
DATAMATION nr 6/1981

Dysk optyczno-numeryczny zamiast mikrofilmowania?

Od momentu opublikowania w 01-HEBDO nr 617 ogólnego opisu dysku optyczno-numerycznego, na rynku informatycznym wiele się zmieniło. Publikacje STRATEGIE INC [1]) wskazują, że w 1981 r. grupę producentów oferujących elektroniczne systemy archiwowania i wyszukiwania dokumentów stanowiły m.in. firmy PHILIPS, TOSHIBA i THOMSON.

System MEGADOC firmy PHILIPS wyposażony jest w sterowane mikrokomputerem P 857 urządzenie typu „juke box" („szafa grająca") obejmujące 64 dyski optyczno-numeryczne. Urządzenie to jest wyposażone w laserowy skaner, który w oparciu o zasadę telekopiowania przekształca treść dokumentów tradycyjnych do postaci danych binarnych. Ma ono możliwość umieszczenia na obu powierzchniach jednego dysku 2500 dokumentów (typu rysunków). W przypadku zastosowania upakowanej postaci zapisu binarnego określanej przez PHILIPSA jako „run length coding", pojemność dysku wzrasta do 25 tys. dokumentów. Zmniejszenie gęstości analizy dokumentu przez skanera z 2400 do 1200 linii na cal pozwala zarejestrować na dysku 50 tys. dokumentów.

W przypadku zapisu tekstów alfanumerycznych pojemność jednego dysku może osiągnąć 500 tys. stron formatu A4, co oznacza, że jeden „juke box" umożliwia zarejestrowanie 32 mln takich stron. Biorąc średnio 50 wierszy po 80 znaków alfanumerycznych na stronę, system MEGADOC dysponuje możliwością zapisu 128 mld bajtów. Należy dodać, że rysunki i wykresy formatu A4 kodowane są w postaci matrycy o rozmiarach 2376 × × 1728 punktów, co w chwili obecnej jest jednym z najlepszych rozwiązań w tej dziedzinie. Dysk optyczny wchodzi w zestaw komputerowy zamiast klasycznego dysku magnetycznego.

Urządzeniem wejścia jest w systemie MEGADOC czytnik optyczny dokumentów ODR (Original Document Reader) o prędkości odczytu 4 Mbitów/s, co odpowiada możliwości zarejestrowania jednej strony maszynopisu formatu A4 w ciągu sekundy. Systemy MEGADOC mogą być łączone i współpracować w konfiguracji sieciowej z szybkością przesyłania 4 bitów/s. W przygotowaniu znajdują się nowe generacje systemów, które pozwolą osiągać prędkości rzędu 300 Mbitów/s.

System operacyjny MEGADOC składa się z trzech podsystemów: rejestracji, zarządzania oraz wyszukiwania dokumentów.

Firma THOMSON-CSF rozwija współpracę z firmą XEROX nad opracowaniem dysku optyczno-numerycznego, którego prototyp ujrzy światło dzienne w 1982 r. Na ten temat nie podano jeszcze dokładniejszych informacji, wiadomo tylko, że rozwiązania zostały oparte na zasadzie działania „inteligentnej kopiarki", a więc nie, jak w przypadku systemu MEGADOC, w oparciu o laserowy skaner. Przewiduje się, że ceny tych urządzeń na rynku amerykańskim będą wynosić od 14 000 do 20 000 dolarów. STRATEGIE INC zaznacza, że główne trudności techniczne są wspólne dla wszystkich rozwiązań dysków optycznych i tkwią one w fazie produkcji. Zastosowana przez firmy THOMSON i XEROX technika archiwowania pozwala złagodzić dotychczasowe trudności zastosowań, między innymi przez wprowadzenie nowej struktury zbiorów. Zastosowane rozwiązania techniczne pozwalają w pełni wykorzystać olbrzymią pojemność dysku optycznego.

Urządzeniem najbardziej perspektywicznym ze wszystkich systemów elektronicznego archiwowania może się stać system firmy TOSHIBA. Konstruktorzy japońscy znajdują się w końcowej fazie realizacji własnego systemu archiwowania laserowego OF 2000. System ten składa się z czterech podstawowych elementów:

- czytnika laserowego z matrycą 8,5 punktów/mm, co umożliwia numeryczny odczyt dokumentów w postaci zarówno maszynopisów jak i rysunków
- dysku optyczno-numerycznego o pojemności 10 000 stron
- drukarki laserowej o prędkości druku 17 dokumentów/min.
- jednostki sterującej z klawiaturą i ekranem.

Produkt firmy TOSHIBA jest bardziej konkurencyjny w stosunku do uprzednio zaprezentowanych, ponieważ jest to jedyna firma, która oferuje system wyposażony w drukarkę, a więc znakomicie przystosowany również do prac biurowych.

W początkowej fazie opracowania we wszystkich systemach nie brano pod uwagę metod logicznego wyszukiwania dokumentów. Dopiero w końcowym etapie prac nad systemem następuje opracowanie metod dostępu, które warunkują efektywną pracę całego systemu. Znając możliwości procesora oraz urządzeń we/wy można opracować szczególnie efektywny, bo działający w czasie rzeczywistym, system wyszukiwania dokumentów.

Tak w największym skrócie przedstawiają się produkty trzech producentów, którzy obecnie przygotowują się do wejścia na rynek z dyskami optyczno-numerycznymi. Przyszłość tych urządzeń zależeć będzie oczywiście od efektywności zastosowania oraz cen. Według opinii STRATEGIE INC, będą one stopniowo wypierały komputerowe mikrofilmowanie (metoda COM) i w 1985 r. opanują ok. 20% rynku. Dyski optyczno-numeryczne staną się niezbędnym uzupełnieniem dysków magnetycznych, które dzięki stałej poprawie parametrów zapisu/odczytu oraz czasów dostępu utrzymują nadal część tego rynku.

Jeżeli jednak w przyszłości dyski optyczno-numeryczne uzyskają zbliżone do dysku magnetycznego możliwości zapisu/odczytu, to bez wątpienia wyprą one całkowicie taśmy magnetyczne. Aktualne koszty poszczególnych rodzajów nośników oraz rejestrowanie na nich informacji ilustruje załączona tabela.

We Francji archiwa przedsiębiorstw liczą ok. 400 mld stron, z czego 95% ma nadal formę dokumentów papierowych, 3% znajduje się na nośnikach magnetycznych, a 2% na mikrofilmach. Dyski optyczno-numeryczne zapewne zmienią te proporcje. Przewiduje się, że systemy z takimi dyskami będą również stosowane do rejestracji i odtwarzania obrazu i dźwięku telewizyjnego.

Oprac. P. A. MILEWSKI
na podstawie czasopisma 01 HEBDO nr 644 z 11 maja 1981 r.

[1]) Impact of optical memories on existing media and equipment, vol. 2. STRATEGIC INC, 4320 Stevens Creek boulevard, Sant Jose, California.

Porównanie kosztów różnych nośników informacji do rejestracji i archiwowania dokumentów

Rodzaj urządzenia	Koszt jednostkowy ($)	Liczba stron dokumentów na jednostkę	Koszt na jeden M bit informacji ($)
Microfisza COM 72 X	3—5	670	0,0045—0,0075
Microfisza COM 48 X	1,50—4	270	0,01—0,015
Microfisza 24 X	2,50—4	98	0,026—0,041
Mikrofilm (100 stóp)	12—16	300	0,004—0,0053
Dysk magnetyczny	400	640	0,625
Taśma magnetyczna (2400 stóp)	24	1400	0,017
Dysk optyczny	200—350 (obecnie)	20000	0,0126—0,0175
	20—50 (w przyszłości)		0,0012—0,006

Sterowanie komunikacją w sieciach komputerowych

Problematyka sieci komputerowych staje się przedmiotem coraz liczniejszych publikacji. Wydawnictwa ostatniego dziesięciolecia, dotyczące tej dziedziny, koncentrowały się na podstawach technologicznych systemów teleprzetwarzania, metodach i technikach tele- i radiokomunikacji (np. protokoły komunikacji), stosowanych w sieciach komputerowych, oraz na teorii analizy i syntezy komunikacji komputerowej w systemach teleinformatycznych. Pozycje wydawnicze tego typu zostały udostępnione polskiemu czytelnikowi zarówno w formie tłumaczeń, jak i opracowań krajowych. Na tym tle omówiona tutaj praca [1]) pod względem treści i metody prezentacji problematyki systemów teleinformatycznych — jest całkowitą nowością.

Istota tej książki polega na sposobie podejścia autorów do sterowania komunikacją, któremu podporządkowana została analiza dotychczasowych osiągnięć w dziedzinie metod i technik komunikacji komputerowej. W pracy zaakcentowano dynamiczne aspekty komunikacji, występujące zarówno w fazie projektowania, jak i eksploatacji systemów realizujących usługi rozproszonego przetwarzania danych. Osiągnięcie przez autorów tak ambitnego celu jest wynikiem ich specjalizacji w dziedzinie telekomunikacji (J. Pužman) i architektury systemów komputerowych (R. Pořizek). Praca stanowi kompendium wiedzy o metodach i technikach sterowania komunikacją komputerową, co uzasadnia zaliczenie jej do bardziej wartościowych pozycji monograficznych w dziedzinie teleinformatyki.

Książka składa się z czterech części oraz dodatków zawierających szczegółowe informacje o osiągnięciach standaryzacyjnych w dziedzinie komunikacji komputerowej (liczne tabele dotyczące zalecanych parametrów systemowych). Każda część książki zawiera szczegółową analizę związanej z nią literatury oraz bogaty zestaw bibliografii, co znakomicie ułatwia czytelnikowi posługiwanie się tą pracą w badaniach naukowych.

●

Obszerne wprowadzenie w ogólne aspekty komunikacji komputerowej i jej rolę w systemach rozproszonego przetwarzania danych stanowi część pierwszą. Scharakteryzowane w niej zostały podstawowe techniki i rodzaje struktur systemów teletransmisji danych oraz ich wpływ na jakość usług tele- i radiokomunikacyjnych (z wykorzystaniem np. łączy cyfrowych analogowych i różnych technik komutacji). Wpływ funkcji komunikacyjnych na efektywność dostępu do zasobów sieci i transferu informacji przedstawiono na tle właściwości sprzętowych i programowych komponentów sieci, metod komutacji informacji (SITA) i pakietów (DATAPAC, ARPA i inne) oraz architektury sieciowych systemów komputerowych czołowych producentów sprzętu informatycznego, jak: SNA (IBM), DNA (DEC), DCA (UNIVAC).

Część druga dotyczy analizy technik sterowania komunikacją w sieciach komputerowych i stanowi obszerne wprowadzenie do syntezy tych metod przeprowadzonej w trzeciej i czwartej części książki. Sterowanie komunikacją jest rozpatrywane w dwóch nurtach: sprawności systemu komunikacji oraz implementacji oprogramowania komunikacyjnego w sieci.

Sprawność komunikacji jest wyrażona zbiorem takich parametrów, jak: przepustowość, czas transmisji, opóźnienie w pętli, rozkład błędów, stopień wykorzystania węzłów lub linii (kanałów). Ich wartości są rozpatrywane z punktu widzenia ogólnych wymagań użytkowych stawianych usługom sieciowym, jak: dokładność, gotowość, niezawodność oraz koszt eksploatacji i budowy sieci. Optymalizacja funkcji sterujących komunikacją w sieci polega na doborze odpowiedniego algorytmu (mechanizmu sterującego), realizującego operacje za pomocą różnych jednostkowych mechanizmów sterujących — implementowanych zarówno w komputerach, jak i komponentach komunikacyjnych sieci.

Implementacja jest związana z oprogramowaniem funkcji sterujących komunikacją komputerową, realizowanymi za pomocą zbioru operacji transferu danych między komponentami sieci. W projektowaniu mechanizmów komunikacji występują dwa wzajemnie uzupełniające się podejścia, dotyczące statycznych i dynamicznych komponentów sieci. Implementacja funkcji sterujących jest uwarunkowana projektowaniem takich mechanizmów sterujących i algorytmów, które zapewnią odpowiednią dynamikę sterowania komunikacją na poszczególnych poziomach architektury sieci (zgodnie z protokołami sieciowymi).

Punktem wyjścia analizy sterowania komunikacją są funkcje komunikacyjne i usługi sieciowe. Funkcje komunikacyjne są rozpatrywane w aspekcie faz komunikacji (nawiązywania połączenia i transferu danych), rodzaju łączy (dzierżawionych, komutowanych) i sposobów implementacji tych funkcji na poziomie węzłów komunikacyjnych sieci. Szczególne znaczenie w realizacji funkcji komunikacyjnych przypisano technikom synchronizacji operacji komunikacyjnych i sterujących. Analiza jakościowa parametrów przeprowadzona dla różnych faz sterowania komunikacją (sterowanie szeregowe, równoległe) dała podstawę wyznaczenia kryteriów klasyfikacji funkcji komunikacyjnych.

Wpływ architektury sieci na sposób sterowania komunikacją jest rozpatrywany z punktu widzenia organizacji logicznej (relacje sterujące między elementami logicznymi sieci) oraz konfiguracji sprzętu komunikacyjnego sieci. Za podstawę analizy przyjęto warstwową architekturę sieci, tryby komunikacji międzywarstwowej (komunikacji V, nazywana kaskadową), protokoły komunikacyjne: międzywarstwowe i warstwowe oraz zbiory operacji sterujących występujących na różnych poziomach architektury sieci.

Dekompozycja funkcji protokołowych w warstwie posłużyła do wydzielenia bloków sterowania w danej warstwie oraz formatów danych i komunikatów wykorzystywanych do sterowania komunikacją międzywarstwową. Protokół jest traktowany jako podstawowy komponent służący do sterowania dwóch komunikujących się stacji (warstw) sieci. W tym ujęciu protokoły wyznaczają reguły sterowania i mogą być wyrażone za pomocą procedur lub faz komunikacji, przy użyciu takich parametrów, jak: długość informacji, liczba modułów, wymiar tzw. okna, maksymalna liczba retransmisji, częstotliwość „time-out'ów". Aspekty implementacyjne funkcji sterujących autorzy analizują na tle stosowanych formalnych metod i języków specyfikacji protokołów komunikacyjnych sieci.

Przedmiotem dalszej analizy sterowania komunikacją są rola i struktury formatów w architekturze sieci. Format jest traktowany jako kmpozycja rozkazów i danych o różnych długościach (nagłówka oraz pól danych, służących do sterowania i wykrywania błędów). Rola formatu jest rozpatrywana z punktu widzenia hierarchii sterowania dostępem komunikatów na poziomie danych, rekordów, segmentów i plików. Bierze się również pod uwagę formaty orientowane bitowo, znakowo itp., wpływ poziomu ich przezroczystości kodowej na transmisję danych sterujących oraz operacje na formatach dotyczące całego lub części bloku danych (np. transformacja lub modyfikacja formatów).

[1]) Josef Pužman, Radosław Pořizek: Communication Control in Computer Networks. John Wiley and Sons, Chichester, 1980, str. 296

W podsumowaniu tej analizy autorzy dokonali obszernego przeglądu roli i ewolucji prac standaryzacyjnych prowadzonych m.in. w ramach organizacji międzynarodowych CCITT, ISO, (TC.6), ECMA, IFIP (WG 6.1); oraz krajowych: ANSI, NBS, AFNOR, DNA.

Synteza technik sterowania komunikacją została przeprowadzona na tle funkcji (część trzecia) i protokołów (część czwarta) komunikacyjnych. Rozpatrywane są takie funkcje, jak: ustanawianie i wykonywanie połączeń, synchronizacja i fazowanie, adresowanie, kontrola błędów, transfer danych i zapobieganie przeciążeniom, marszrutowanie oraz podział pojemności łącza transmisyjnego (zwłaszcza w przypadku łączy radiowych).

Ustanawianie i wykonywanie połączeń dla usług datagramowych i wirtualnego kanału, traktowanych jako postać zbioru procedur jest opisywane za pomocą diagramów stanu, na poziomie stacji DTE/DCE (wg zaleceń CCITT serii V i X) stosowanych w publicznych sieciach danych.

Synchronizacja procesów jest rozpatrywana w przypadku wyższych warstw architektury sieci z punktu widzenia występowania zastoju (ang. *deadlock*) dla różnych poziomów centralizacji funkcji sterujących (za pomocą semaforów). Natomiast rozkazy synchronizacji i ich transfer między stacjami na poziomie bitów są modelowane dla warstw komunikacyjnych (np. protokół HDLC) ze względu na częstotliwość sygnałów i fazowanie transmisji. Fazowanie jako metoda separacji (np. bloków w formacie) jest wykorzystywane dla synchronizacji bitów sterujących (np. dekodowania, inicjacji połączenia).

Wyróżnono pewne kryteria służące za podstawę klasyfikacji sposobów adresacji procedur sterujących w sieci, ze względu na:

- liczbę poziomów w hierarchii struktury sieci
- rodzaj oferowanych usług
- liczbę adresowanych stacji
- tryb alokacji adresów.

Adresowanie może być dokonywane fizycznie (jedno- lub wielopunktowo), logicznie (grupowo) i hybrydowo (w sposób mieszany).

Metody wykrywania błędów zależą od miejsca ich występowania oraz od rodzaju błędów. Na potrzeby sterowania komunikacją wykorzystywane są wszystkie podstawowe techniki kodowania (kody korekcyjne, kody cykliczne), metody sprzężenia zwrotnego (decyzyjnego ARQ, informacyjnego, kombinowane) oraz niektóre metody automatycznego powtarzania transmisji (ARQ z różnymi kryteriami decyzyjnymi).

Kryterium oceny sterowania przepływem danych i dostępu do zasobów sieci jest miara wydajności na poziomie liczby przesyanych pakietów między węzłami komunikacyjnymi. Celem sterowania nadmiarem (zapobiegania przeciążeniom) jest zachowanie odpowiedniego stosunku liczby danych wprowadzanych do sieci do wielkości transferu danych. Stosunek ten zależy od przyjętych parametrów dla mechanizmów rozpoczynania danych i realizacji transferu danych. Brane są pod uwagę zarówno parametry statyczne (np. okna w protokole HDLC), jak i dynamiczne (np. kredyty w protokole INWG 96), a także techniki zarządzania buforami i alokacją zasobów (logicznych kanałów, stacji abonenckich, sesji procesów użytkowych itp.). Zaproponowano następujące trzy sposoby sterowania procesami służącymi do zapobiegania przeciążeniom w sieci:

- ograniczanie strumieni wejść
- zmniejszanie istniejącego obciążenia podsieci komunikacji
- lokalne rozpraszanie strumieni danych.

Ostatnia z metod, znana pod nazwą marszrutowania (ang. *routing*) może być realizowana technikami adaptacyjnymi lub prostymi, w zależności od stopnia centralizacji funkcji sterujących.

Metody projektowania i implementacji protokołów komunikacji (część czwarta) są rozpatrywane w kontekście narzędzi zapewniających uzyskanie sprawnych mechanizmów sterowania komunikacją w sieci. W myśl definicji autorów protokół wyraża jakość (sprawność) oczekiwanych od sieci usług komunikacyjnych. Realizacja protokołów jest dokonywana w następujących etapach:

- projektowania (wybór formalnych metod modelowania)
- implementacji (dobór środków technicznych i ich integracja na poziomie programów w architekturze sieci)
- ocena (weryfikacja poprawności, kompletności i adekwatności specyfikacji protokołów).

Ze względu na rodzaj stosowanych komponentów i poziomy odwzorowania architektury sieci, modele protokołów komunikacji mogą się różnić pod względem struktury oraz rodzaju i liczby parametrów. Z tego m.in. względu w modelowaniu protokołów stosuje się różny aparat formalny, a konkretnie: teorię automatów skończonych, diagramy stanów, macierze przejść, sieci Petriego z elementem czasu, gramatyki formalne i języki programowania wysokiego poziomu (np. PASCAL). Przyszłość modelowania zależy jednak od możliwości adekwatnego odwzorowania w modelu protokołu mechanizmów współbieżności operacji dwóch niezależnych stacji protokołu (w odróżnieniu od dotychczasowych metod, biorących pod uwagę operacje tylko jednej stacji).

Spośród około 17 znanych aktualnie technik modelowania autorzy preferują metody: dwóch interlokutorów (jako stacji protokołowych), sieci Petriego z mechanizmem czasu oraz języki formalne. Weryfikacja protokołów w kontekście sterowania komunikacją jest dokonywana za pomocą aparatu formalnego na etapie modelowania protokołów oraz za pomocą pomiarów i testów na etapie implementacji protokołów. Pierwsza jest formalną weryfikacją specyfikacji protokołu oraz jego parametrów za pomocą odpowiedniego algorytmu weryfikacji. Przedmiotem badań są w tym przypadku transformacje formalne, m.in. wymaganych funkcji komunikacyjnych na parametry protokołu, zbioru błędów transmisji na jakość urządzeń komunikacji oraz wpływ błędów na realizację sterowania komunikacją i synchronizację procesów współbieżnych. Ocena protokołów za pomocą pomiarów jest dokonywana według kryteriów użytkowania sieci oraz jakości wykorzystywanych technik pomiarowych. Szczególne znaczenie ma jednak ocena jakościowa transferu danych (wykrywanie błędów, sterowanie przepływem i odzyskiwanie połączeń). Za podstawowe kryterium przyjmuje się czas odpowiedzi systemu (mierzony parametrem gotowości — wyrażony przepustowością · i opóźnieniem transferu) oraz dokładność usług (mierzona rozkładem błędów).

W ośmiu dodatkach (A—H) zamieszczono szczegółowe informacje o zaleceniach standaryzacyjnych, dotyczące: podstawowych procedur przezroczystego trybu sterowania komunikacją wg ISO 1745—2629 na bazie (A), kontroli błędów w systemie z niezależnością kodową wg CCITT V41 (B), zaleceń dla procedur protokołu HDLC wg ISO 3309, 4335 (C), zaleceń CCITT wg X25, X3, X28, X29, X75 (D), zaleceń INWG 96 i 96. 1 dla usług transportowych (E), zaleceń ISO dla architektury systemów otwartych — OSA (F), oraz dodatkowe informacje bibliograficzne (H).

W swoim ujęciu recenzowana praca stanowi doskonały podręcznik dla inżynierów zajmujących się projektowaniem systemów komunikacji komputerowej. Może ona okazać się szczególnie przydatna dla prac związanych z oprogramowaniem komponentów komunikacyjnych sieci. Służyć może jako podręcznik dla specjalistów informatyki oraz dla studentów szkół wyższych kształcących się w kierunku specjalności teleinformatycznych. Uważam, że podjęcie starań o jej wydanie w języku polskim byłoby w pełni uzasadnione i to ze względu na lukę, jaką wypełniłaby ona w krajowych wydawnictwach z dziedziny teleinformatyki, a także na jej aktualne zapotrzebowanie wynikające ze światowych trendów rozwojowych systemów teleinformatycznych.

Zasługującym na szczególną uwagę recenzenta jest fakt, że wśród licznie cytowanych doniesień literaturowych poważną pozycję zajęły wyniki badań przedstawiane od 1976 r. na corocznych polsko-czechosłowackich szkołach naukowych „Sieci komputerowe", organizowanych przez Politechnikę Wrocławską. Pewne problemy dotyczące funkcji komunikacyjnych i protokołów (omówione w trzeciej i czwartej części książki), znalazły swoje miejsce w publikacjach autorów zamieszczonych w wydawnictwie Politechniki Wrocławskiej w serii „Biblioteka WASC" „Własności i funkcje sieci komputerowych". Część I: Protokoły; Część II: Komunikacja w latach 1979—1980.

MIECZYSŁAW BAZEWICZ

Terminologia języka ADA (cd.)

Jedna z najbardziej interesujących właściwości języka ADA dotyczy możliwości programowania systemów czasu rzeczywistego. Podstawowym terminem związanym z programowaniem takich systemów jest zadanie. Według podręcznika języka (Reference Manual for the ADA Programming Language, July, 1980) zadanie jest jednostką programową wykonywaną równolegle z innymi jednostkami programowymi, a więc — z zadaniami (ang. *a program unit that may operate in parallel with other program units).*

Należy przyznać, że określenie to jest intuicyjnie zrozumiałe i można by poprzestać na jego zacytowaniu. Ponieważ jednak prace nad określeniem operacji elementarnych (ang. *primitives*) do przetwarzania równoległego rozpoczęto stosunkowo niedawno i terminologia nie jest jeszcze ustalona, warto głębiej zastanowić się, co się kryje za tak lakonicznym sformułowaniem.

Zacznijmy od ustalenia, co rozumie się przez równoległość. Według normy ISO 2382, praca równoległa (ang. *parallel operation*) to taki tryb przetwarzania, w którym operacje wykonuje się albo współbieżnie na jednym urządzeniu, albo współbieżnie lub jednocześnie na dwóch urządzeniach.

Praca współbieżna (ang. *concurrent operation*) to taki tryb przetwarzania, w którym w określonym przedziale czasu wykonuje się dwie lub wiele operacji. Natomiast, praca jednoczesna (ang. *simultaneous operation*) to taki tryb przetwarzania, w którym w tym samym czasie następuje dwa lub wiele zdarzeń. Na jednym urządzeniu można zatem wykonywać operacje współbieżne, ale nie jednocześnie. Dlatego przez wieloprogramowanie rozumie się tryb pracy umożliwiający naprzemienne wykonywanie dwóch lub wielu programów komputerowych na pojedynczym procesorze, a wieloprzetwarzanie jest to tryb pracy polegający na przetwarzaniu równoległym na dwóch lub wielu procesorach (wszystkie określenia wg normy ISO 2382).

Dodajmy, że przez wielozadaniowość (ang. *multitasking*) w wymienionej normie rozumie się tryb pracy umożliwiający działanie współbieżne lub naprzemienne wykonywanie dwu lub wielu zadań.

Do bardziej szczegółowego zanalizowania pojęcia zadania skłoniło mnie stwierdzenie twórców języka ADA (O. Roubine, J. C. Heliard), że zadaniami nazywa się procesy równoległe.

Według normy ISO 2382 proces (w przetwarzaniu danych) jest to bieg zdarzeń, następujący zgodnie z zamierzonym celem lub skutkiem. W polskiej terminologii komputerowej proces rozumie się podobnie, a więc jako wykonywanie przez procesor kolejnych instrukcji, których ciąg jest programem procesu. [1])

Porównując oba określenia procesu stwierdzamy, że „zamierzony cel lub skutek" (w definicji ISO) może być określony w programie. Wyjaśnienie zatem sformułowania zawartego w normie ISO brzmiałoby: proces jest to bieg zdarzeń następujący zgodnie z programem. Wynika stąd, że w rzeczywistości istnieją dwa odrębne, choć ściśle związane pojęcia — proces i określający go program (opis procesu). Do którego z nich odnosi się więc termin zadanie?

W jednej z niedawno wydanych prac spotkałem następujące określenie: proces (zwany niekiedy „zadaniem") jest działalnością wynikłą z wykonywania programu wraz z jego danymi przez procesor. [2]). Natomiast, w wymienionej normie ISO, przez zadanie w środowisku wieloprogramowym lub w wieloprzetwarzaniu rozumie się jeden lub więcej ciągów instrukcji traktowanych przez program sterujący jako elementarna praca wykonywana przez komputer. Prawie identyczne określenie podano w normie ANSI. [3])

Tak więc przez zadanie potocznie rozumie się zarówno sam proces, jak i odpowiadający mu segment programu (choć nie zawsze jedno i drugie razem). Jeżeli przyjrzymy się dokładniej określeniu zadania podanemu w podręczniku języka ADA, to stwierdzimy, że zadaniem nazywa się odpowiedni segment programu, lecz — implicite — także przebieg wykonywania tego programu. Należy więc pamiętać, że termin zadanie (w przeciwieństwie do terminu proces) odnosi się do obu wymienionych pojęć.

Zadania (w drugim znaczeniu) komunikują się za pomocą tzw. spotkań. Spotkanie (ang., franc. *rendezvous)* polega na współdziałaniu dwóch równoległych zadań, gdy jedno z nich wywołało drugie, które w odpowiedzi wykonało odpowiednie czynności (omówienie określenia z podręcznika ADY). Mówiąc obrazowo, zadanie wywołujące podaje „hasło" (zawierające nazwę zadeklarowaną w zadaniu wywoływanym jako entry), a zadanie wywoływane odpowiada „odzewem" (wykonaniem instrukcji accept).

W tym wypadku użycie nazwy spotkanie nie powinno budzić wątpliwości (wyjaśniając znaczenie spotkań używa się także słowa *meeting*).

Inne ważne pojęcie języka ADA dotyczy reakcji na sytuacje wyjątkowe, określane angielskim słowem *exception*. Według podręcznika *exception* jest to zdarzenie powodujące zawieszenie normalnego wykonywania programu. Słowo *exception* najłatwiej byłoby przetłumaczyć na język polski jako wyjątek. Jednak taka nazwa nie odpowiadałaby treści pojęcia. Wydaje mi się, że odpowiedniejszym określeniem jest słowo wypadek. Według „Słownika poprawnej polszczyzny" (pod red. W. Doroszewskiego, PWN, Warszawa, 1980) słowo wypadek oznacza zdarzenie, zajście, fakt, a więc dokładniej odpowiada przedstawionemu znaczeniu słowa *exception*.

Z wypadkami w języku ADA wiążą się dwie ważne czynności (operacje) określane po angielsku jako *„raising the exception"* oraz *„handling the exception"*. Czynność *„raising the exception"* polega na zwróceniu uwagi na zdarzenie (ang. *drawing attention to the event*, str. 11.1 podręcznika; *bringing an exception to attention*, str. D-2), dlatego proponowałbym określić ją jako stwierdzenie wypadku (lub ogłoszenie wypadku). Czynność *„handling the exception"* polega na wykonaniu działań będących reakcją na wypadek (ang. *executing some actions, in response to the occurence of an exception*, str. 11.1; *execution of a program text specifying a response to the exception*, str. D-2) i chyba nie będzie błędem nazwanie jej po polsku — obsługą wypadku.

Na zakończenie warto zauważyć, że sytuacje wyjątkowe powstałe wskutek wypadków są analogiczne do przerwań. Choć nie jest to jawnie powiedziane w podręczniku, podstawową różnicę rozumiem tak, że wypadki są bezpośrednią konsekwencją wykonywania programu, natomiast sygnały przerwania nie są związane z aktualnie wykonywanym programem (porównaj VAX — 11/780 Processor handbook).

Przedstawionego rozumowania, jak i poszczególnych propozycji nie należy traktować jako ostatecznych. Uważam je jedynie za przyczynek do doskonalszych ustaleń w zakresie słownictwa języka programowania, który ma stać się dominujący w informatyce w ciągu najbliższych lat.

Janusz ZALEWSKI

[1]) Określenie skrócone użyte przez J. Olszewskiego w pracy „Wybrane zagadnienia systemów operacyjnych" (pod red. W. M. Turskiego), PWN, Warszawa, 1971; por. także J. Olszewski, „Projektowanie struktur systemów operacyjnych", WNT, Warszawa, 1981

[2]) Według polskiego wydania książki A. C. Shaw „Projektowanie logiczne systemów operacyjnych", WNT, Warszawa, 1980

[3]) American National Dictionary of Information Processing, X3/ /TR — 1 — 77, CBEMA, Washington, DC, September, 1977

Kilka uwag o „ALGORYTMACH"

W INFORMATYCE (nr 11—12/81) pojawiła się nowa rubryka — „Algorytmy". Pomysł „Podręcznej Biblioteki Programisty" jest trafny, a propozycje pp. A. Szałasa i Zb. Świrskiego — ciekawe. Warto jednak przy okazji zastanowić się nad kilkoma sprawami.

- *Czy jest uzasadnione ograniczanie zakresu tematycznego — jak proponują Autorzy — do algorytmów nienumerycznych?*
- *Część nowego działu mogłaby być poświęcona krótkim opisom „referującym" algorytmy opublikowane gdzie indziej. (Należy wziąć pod uwagę, że cięcia dewizowe spowodowały drastyczne ograniczenie dostaw czasopism zagranicznych. Największe ośrodki — przeważnie znajdujące się w Warszawie — będą zapewne otrzymywały jedyne egzemplarze w kraju. Warto więc prowadzić przegląd literatury w dziedzinach interesujących szersze grono osób. Ten wniosek może mieć charakter bardziej ogólny, dotyczący profilu całego czasopisma).*
- *Najwyższa pora, aby zająć się poważnie na łamach INFORMATYKI algorytmami i programami dla systemów mikroprocesorowych! Wypada przypomnieć, że w zagranicznych czasopismach istnieje wiele „kącików" poświęconych konkretnym programom tego typu. Wzięcie przykładu z „μComputerist Corner" czy „Software Notes" nikomu ujmy nie przyniesie.*

Ze swej strony chciałbym zaproponować jeszcze jeden punkt widzenia na popularyzację wartościowych algorytmów. Swój pomysł adresuję do grona praktyków — użytkowników informatyki; do zespołów, które użytkują minikomputery lub mają końcówki systemów wielodostępnych, ale nie są ośrodkami obliczeniowymi. Kieruję go też do posiadaczy i użytkowników komputerów osobistych (dziś jest ich niewielu, ale sądzę, że za kilka lat będą liczącą się siłą w krajowej informatyce; wierzę, że to oni pchną ją na nowe drogi rozwoju). Proponuję — regularne publikowanie programów szerokiego użytku w języku BASIC!

System BASIC nie zawiera z reguły biblioteki podprogramów, co w pewnym stopniu wynika ze struktury samego języka. Przeważnie użytkownicy tworzą swoje biblioteki programów z „cegiełek", nad którymi jednocześnie pracują inni. Na łamach INFORMATYKI mogłaby powstać podręczna bibioteka programisty-użytkownika, posługującego się popularnym już w Polsce BASIC'em.

BASIC oparł się próbie czasu, rozpowszechnia się jako standard dla małej maszyny. Jest prosty i przejrzysty. Programowanie musi być umiejętnością powszechną wśród specjalistów nieinformatyków. Pora wyjść im naprzeciw. Do tego zaś celu BASIC jest idealny.

JACEK ZEBROWSKI
Łódź

W odpowiedzi

Postulaty p. J. Żebrowskiego są zasadne. Chcielibyśmy więc na nie odpowiedzieć.

- Zakres tematyczny rubryki ograniczyliśmy do algorytmów nienumerycznych ponieważ uważaliśmy, że w polskiej literaturze fachowej na ten temat istnieje poważna luka. Natomiast dostępne są na naszym rynku wydawniczym dobre opracowania algorytmów numerycznych (np. wydane przez PWN: Jankowscy J., M.: „Przegląd metod i algorytmów numerycznych" lub Zorychta K., Ogryczak W.: „Programowanie liniowe i całkowitoliczbowe").
- W omawianej rubryce staramy się również przedstawiać ciekawe algorytmy ukazujące się aktualnie w publikacjach zagranicznych. Jesteśmy zdania, że proponowany przez Pana rodzaj bibliografii dostępnych w Polsce źródeł byłby z pewnością interesujący jako osobne opracowanie. Jednak ze względu na objętość rubryki musiałby być bardzo fragmentaryczny i chyba nie spełniłby oczekiwań Czytelników.
- Zajęcie się mikroprocesorami byłoby obecnie nieco na wyrost, gdyż nie ma powszechnie znanego uniwersalnego języka, służącego do programowania mikroprocesorów.
- Wybrany przez nas język programowania ma według nas dużą przewagę nad BASICEM, gdyż można w nim pisać programy w sposób czytelny i z uwzględnieniem zasad programowania strukturalnego. Ponadto notacja algolowa jest znacznie krótsza niż zapis proponowany przez Pana, a tłumaczenie nierekurencyjnych procedur algolowych na BASIC jest proste nawet dla niewprawnego programisty. Jeśli jednak napłyną do Redakcji ciekawe i użyteczne programy w BASICU — wydrukujemy je.

A. SZAŁAS, Z. ŚWIRSKI

Szanowna redakcjo

Chęć napisania tego listu wyzwoliło ostatnie zdanie artykułu Pana doc. Orłowskiego w INFORMATYCE nr 11—12 z 1981 r. W związku z tym chciałbym przedstawić Redakcji swoje refleksje.

Jestem przekonany, że temat „przemysł minikomputerowy a informatyka" powinien być rozwijany w swej drugiej części, tj. dotyczącej zastosowań przemysłowych informatyki. Uważam, że właśnie zastosowania informatyki (czy produktów przemysłu komputerowego) dają możliwość oceny przeszłości, stanu obecnego i wytyczenie kierunków działania w zakresie wytwarzania komputerów oraz ich oprogramowania.

Przemysł komputerowy nie może być traktowany jako twór wyizolowany, dostarczający użytkownikom zagranicznym lub krajowym procesorów, pamięci, urządzeń peryferyjnych, programów itp. Musi być także traktowany jako dostawca środków ułatwiających produkcję innych dóbr lub świadczenie usług. Takie podejście odwróci uwagę od kompleksów związanych z nowoczesnością naszego sprzętu, nie hamując prac nad gruntownym usunięciem źródeł tych kompleksów.

Właściwości funkcjonalne naszych komputerów oraz ogromne rezerwy w ich wykorzystaniu mogą zaspokoić z nadmiarem dużą część potrzeb (nie ambicji). Duże możliwości wykorzystania komputerów, a zwłaszcza minikomputerów, widzę w projektowaniu i technicznym przygotowaniu produkcji. Dopiero w tych dziedzinach mogą stać się niezastąpione, przyczyniając się do powstawania najrozmaitszych produktów będących pośrednio wizytówką przemysłu komputerowego. Wtedy właśnie będzie można mówić o opłacalnym eksporcie zarówno efektów zastosowań komputerów jak i ich samych wraz z oprogramowaniem. Nie należy tu jednak zapominać o zwiększeniu asortymentu potrzebnych urządzeń zewnętrznych o plotery i monitory graficzne.

W artykule Pana doc. H. Orłowskiego przewija się pogląd o celowości rozwijania, a następnie eksportu systemów użytkowych opartych na minikomputerach SM. Trudność może sprawić realizacja tej słusznej myśli, skoro szanse jej urzeczywistnienia mają nieliczni posiadacze komputerów PDP i SM. Moim zdaniem należy rozwijać systemy użytkowe, niezależnie od sprzętu, na którym będą realizowane. Z autopsji wiem, że jest to możliwe.

Nie całkowicie zgadzam się z poglądem autora nt. prac nad adaptacją cudzych rozwiązań. Uważam, że w pewnych przypadkach jest to metoda na szybsze zwiększenie efektywności zastosowań informatyki. Sprzedaż własnych systemów użytkowych jest oczywiście korzystna, ale najpierw trzeba takie systemy umieć zrobić.

Gorąco popieram postulaty uzupełnienia istniejących zestawów komputerowych i zgłaszam postulat własny, dotyczący ożywienia szczątków systemu MERA 300 (głównie MERY 305 ze względu na dysk) poprzez ich różnorakie wykorzystanie.

JAN NADOLSKI
Poznań

Trudne chwile informatyki?

Problematyka informatyczna — od momentu rozpoczęcia produkcji sprzętu na skalę przemysłową — wzbudzała w naszym kraju wiele namiętności. Jej rola i zadania przybierały różne rozmiary. Patrząc jednak retrospektywnie, informatyka była częściowo zawieszona w próżni, chociaż przypisywano jej nawet strategiczne zadania, łącznie z uzdrowieniem gospodarki narodowej.

Euforia lat siedemdziesiątych była jednak krótkotrwała; znaczne kwoty wydatkowane na informatykę nie dały bowiem spodziewanych rezultatów. Główne przyczyny tego stanu można sprowadzić do następujących faktów:

- nakłady przeznaczone na informatykę służyły głównie uruchamianiu produkcji przemysłowej sprzętu komputerowego, traktowanego tak jak inne urządzenia techniczne
- potencjalne możliwości kadry nie były w pełni wykorzystane w wyniku różnych rozgrywek służących partykularnym interesom
- nie podjęto intensywnych działań zmierzających do zbudowania systemów informacyjno-decyzyjnych, w ramach których informatyka byłaby elementem normalnego cyklu technologicznego
- nie przełamano barier pomiędzy informatykiem a użytkownikiem; dyrektorom „komputer" słusznie kojarzył się z naukową metodą zarządzania, nie zawsze jednak dostrzegali w informatyce dogodne dla siebie narzędzie.

Informatyka pomimo sporych osiągnięć, tak w dziedzinie technologiczno-sprzętowej, jak i w dziedzinie zastosowań znalazła sobie u nas — moim zdaniem — trwałe miejsce jedynie w systemie szkolnictwa wyższego. Nie spełniła swej roli w całym systemie gospodarczym państwa, ani w jego poszczególnych ogniwach. Nie zyskała sobie również społecznej akceptacji, co — według mnie — jest niezmiernie istotnym czynnikiem warunkującym tempo jej dalszego rozwoju.

Brak wyraźnych efektów wykorzystania informatyki wywołał spadek zainteresowania nią ze strony potencjalnych użytkowników. Spotęgowała to szybko pogarszająca się sytuacja gospodarcza kraju. W rezultacie wielu ośrodkom obliczeniowym zaczęła grozić likwidacja.

Warto przy tym przypomnieć, że stan zastosowań informatyki w jednostkach i centralach decyzyjnych, a także stan przemysłu komputerowego, opartego w znacznej mierze na imporcie — jest co najmniej zły. Stwarza to dodatkowe trudności, jeśli się uwzględni fakt, że nakłady na informatykę w obecnej sytuacji nie mogą być duże, zwłaszcza w warunkach postulowanego samofinansowania. Nie można już oczekiwać na fundusze centralne. Trzeba liczyć przede wszystkim na siebie, na własną operatywność. Będzie to tym trudniejsze, że:

- produkcja sprzętu i możliwości zakupu zostaną bardzo ograniczone
- zainstalowany sprzęt ulega szybkiemu starzeniu się (w 1980 r. komputery starsze niż sześć lat stanowiły około 60% ogółu zainstalowanych)
- wymogi użytkowników znacznie wzrastają — tak w zakresie jakości oraz terminowości informacji, jak i kosztów ponoszonych na przetwarzanie.

Uważam, że w najbliższym okresie działania w skali całej informatyki powinny umożliwić:

- utrzymanie osiągniętego poziomu zastosowań
- wyznaczenie prawidłowych kierunków produkcji sprzętu komputerowego
- współuczestnictwo w pracach reorganizujących gospodarkę na wszyskich poziomach zarządzania
- stworzenie warunków do rozwoju informatyki jako czynnika wspomagającego efektywność gospodarowania.

Przyjęcie powyższych kierunków działań oznaczałoby stagnację w porównaniu z poprzednim okresem, jednak ze względu na aktualną sytuację gospodarczą jest to — moim zdaniem — jedyna możliwa droga.

Główną płaszczyzną działań informatyków — powinno być utrzymanie już osiągniętego poziomu zastosowań. Nie możemy dopuszczać do rezygnowania użytkowników z usług informatycznych. Zwłaszcza w zakresie: systemów przygotowywania i planowania produkcji, obsługi księgowo-finansowej (rozliczanie kosztów), systemów wspomagających sterowanie produkcją, obsługi bankowej, automatyzacji prac projektowych, czy wspomagania zarządzania na różnych szczeblach gospodarki narodowej. Konieczne jest zatem, dokonanie analizy eksploatowanych systemów pod kątem ich użyteczności. Bardziej niż dotychczas należy zwracać uwagę na indywidualne potrzeby użytkowników. Należy poprawić jakość i terminowość realizowanych opracowań. Tylko dobre, rzetelnie świadczone usługi będą teraz miały nabywców.

Musimy przy tym wziąć pod uwagę zawodność sprzętu komputerowego, co zmusza ośrodki do: szczególnej dbałości o jego stan techniczny, do „zdobywania" urządzeń uzupełniających konfigurację zestawów oraz do poprawy organizacji i technologii przetwarzania. Oczywiście, rozstrzygające będzie tu współdziałanie z producentami. Nie może istnieć — jak do chwili obecnej — wyłącznie rynek producenta. Przemysł komputerowy powinien produkować pod kątem potrzeb użytkowników, z uwzględnieniem praw ekonomicznych, co oznacza, że ceny sprzętu powinny gwarantować opłacalność stosowania informatyki.

Należy doprowadzić do sytuacji, że użytkownik — reprezentowany przez organizacje informatyczne — stanie się głównym stymulatorem rozwoju i produkcji sprzętu. Działanie nasze będzie tym skuteczniejsze, im większy będzie nasz udział w pracach przy obecnej reorganizacji gospodarki. Dotyczyć to powinno głównie zagadnień ekonomiczno-finansowych i informacyjno-decyzyjnych, począwszy od podstawowego ogniwa gospodarki narodowej — przedsiębiorstwa — a skończywszy na urzędach centralnych.

Trudno sobie wprawdzie wyobrazić, aby gospodarka mogła dzisiaj nie wykorzystać tak efektywnego narzędzia, jakim jest informatyka. Trzeba jednak zaznaczyć, że my — informatycy, jesteśmy bardzo mało widoczni i zbyt pasywni. Musimy wykazać więcej inicjatywy w rozwiązywaniu problemów gospodarczych, zwłaszcza — aby zyskać społeczne zrozumienie i zainteresowanie informatyką jako dziedziną skutecznie wspomagającą procesy gospodarcze. Informatyka stosowana nie może być bowiem — jak poprzednio — zawieszona w próżni. Musi zostać wprzęgnięta do budowanego nowego systemu informacyjno-decyzyjnego, a także do normalnego cyklu technologicznego przedsiębiorstw.

Nie pora dzisiaj na różne rozgrywki partykularne — czas podjąć wspólne wysiłki dla zachowania ciągłości rozwoju naszej dziedziny.

BARTŁOMIEJ KRUSZELNICKI
Wrocław

4 lutego 1982 r.

robotron

Wszechstronny specjalista wkracza do biur Nowy przedstawiciel rodziny A 5100

Wszędzie poszukiwani są specjaliści, zwłaszcza o dużych walorach adaptacyjnych. Nasz system A 5120 spełnia wszystko to, co przyrzekliśmy. Dosłownie wszędzie: w handlu, przemyśle, ośrodkach obliczeniowych, komunikacji i instytucjach finansowych.

A 5120 stosowany być może również w trybie zdalnym jako terminal. Składa się on z monitora oraz jednostki pamięci na dysku elastycznym lub jednostki pamięci kasetowej. Jego wydajność znacznie wzrasta przez wyposażenie w drukarkę oraz dodatkowe jednostki pamięci na dysku elastycznym lub pamięci kasetowej.

Rejestrowane dane można wprowadzać na zewnętrzne nośniki informacji lub bezpośrednio do jednostki centralnej. I jeszcze jedna istotna cecha: dzięki przyjęciu koncepcji pól ekranu A 5120 jest szczególnie dobrze przystosowany do specyfiki rejestracji danych masowych.

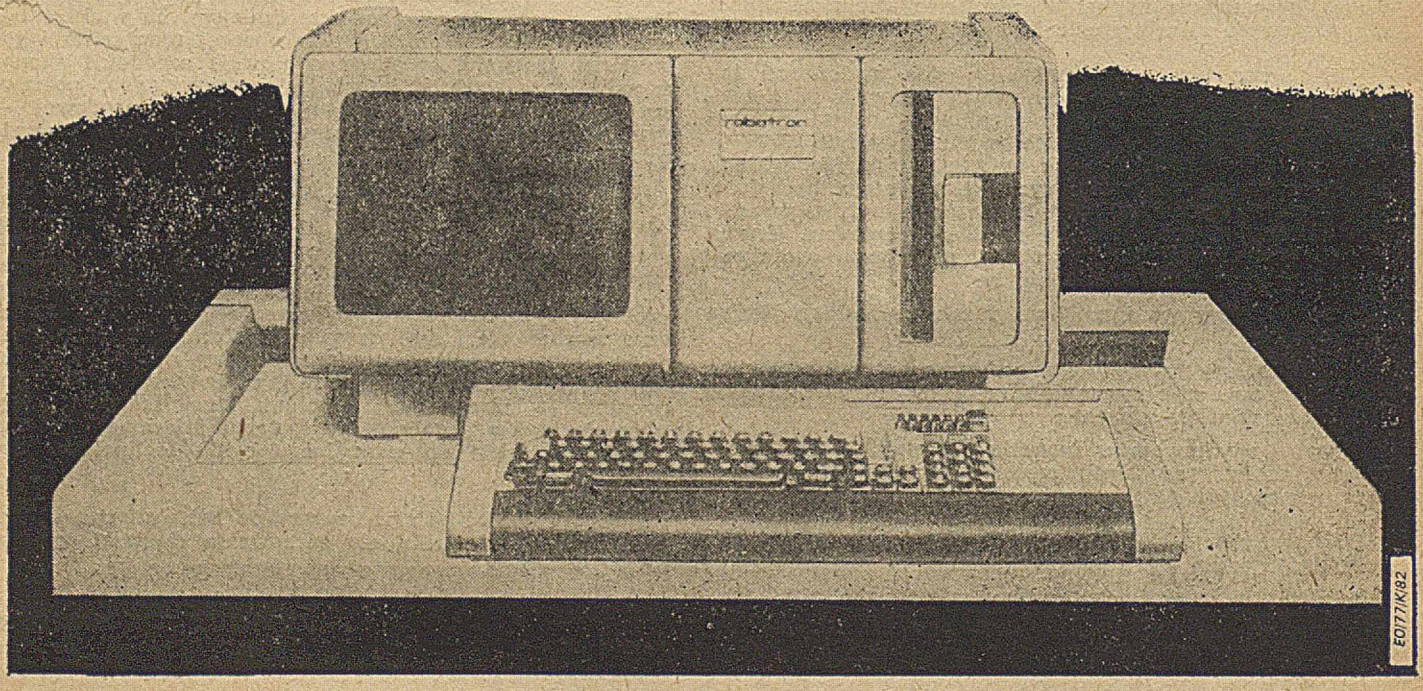
Użytkownik ma do dyspozycji gotowe moduły programów oraz kompletne systemy programowe.

Prosimy zwrócić się o szczegółowe informacje oraz dokładnie obejrzeć A 5120, a następnie przygotować miejsce dla zainstalowania tego systemu księgująco-obrachunkowego.

robotron

Robotron Export-Import
Państwowe Przedsiębiorstwo
Handlu Zagranicznego
Niemieckiej Republiki
Demokratycznej
DDR 1080 Berlin,
Friedrichstrasse 61

A 5120



EO177/K/82