Wstęp do Informatyki #1

Tomasz Przechlewski

Dane, informacja, wiedza

- **Informacja** to jest przyrost **wiedzy**, który może być uzyskany na podstawie **danych**. (Tstchizris i Lochovsky)
- Dane to fakty (symbole). Informacja to zinterpretowane dane umieszczone w znaczącym kontekście. Informacja ma charakter subiektywny. Te same dane mogą być różnie interpretowane. Wiedza to jest informacja zintegrowana z wiedzą istniejącą.
- **Informatyka** jest nauką o przetwarzaniu informacji zwłaszcza przy użyciu automatycznych środków pomocniczych (W. Turski, *Propedeutyka Informatyki*)
- Informatyka dotyczy metod konstruowania procesów przetwarzania danych (P. Naur, *Zarys Metod Informatyki*)

- Sposób **reprezentacji danych** musi być wybrany z należytym uwzględnieniem transformacji, którą należy osiągnąć i środków będących w dyspozycji dla przetwarzania tych danych. (P. Naur, *Zarys Metod Informatyki*)

Reprezentacja danych

Współczesne komputery to *maszyny cyfrowe* (*digital machines*). Przetwarzane dane są *reprezentowane* w postaci ciągu wartości binarnych, nie posiadających oczywistej dla człowieka interpretacji:

`010010111010101101010110...`

Reprezentacja informacji w systemie cyfrowym oznacza zamianę danych zrozumiałych dla człowieka na ciąg bitów. Zbiór reguł według których dokonywana jest ww. zamiana nosi nazwę *formatu danych* (albo *notacji*). Znajomość formatu danych pozwala *zdekodować* dane binarne na postać zrozumiałą dla człowieka. Formaty *tekstowe* i *binarne*. Jawne i niejawne formaty danych.

- *Struktura danych* to wewnętrzna reprezentacja danych w pamięci komputera zaprojektowana pod kątem ich przetwarzania. Struktury i formaty dla tych samych danych są zwykle różne. Serializacja i deserializacja (*parsowanie*, *ładowanie*).

- *Model danych* to sformalizowana, *logiczna* struktura danych oraz (zwykle) sposoby manipulowania nimi (np. modyfikowania, wyszukiwania itp...). Przykładem modelu danych jest *model relacyjny* będący podstawą współczesnych baz danych.
- *Schemat danych* to opis struktury danych zgodnej z określonym formalnym *modelem danych*: Model danych -> Schemat danych -> Struktura/format danych

System informacyjny

- *System informacyjny* (SI) to model pewnego fragmentu świata. Systemy *formalne* i *nieformalne*. Podstawą (formalnego) SI jest *model*/*schemat danych* wykorzystywany do przechowywania danych. Schemat determinuje *funkcjonalność* systemu.
- Systemy nieformalne: np. zbiór plików w formacie MS Word zawierających zestawienie zawartości płyt CD. Większość systemów informacyjnych składa się z systemu formalnego + pewnej liczby systemów nieformalnych
- Systemy formalne: ontologie -> model danych -> schemat danych.

```
From: A. Slomka <topcat@yogi.elfy.pl>
To: Tomasz Przechlewski <tomasz@gnu.univ.gda.pl>
```

```
Subject: Fajny rysunek
Date: Mon, 5 May 2003 23:28:45 +0200
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="----=_NextPart_000_0005_01C3135E.123C6F70"
X-Priority: 3
X-Mailer: Microsoft Outlook Express 6.00.2600.0000
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2600.0000

This is a multi-part message in MIME format.

------=_NextPart_001_0006_01C3135E.123C6F70
Content-Type: text/plain;
        charset="iso-8859-2"
Content-Transfer-Encoding: quoted-printable

Przesyłam fajny rysunek z prośbą o wyrażenie opinii
Pozdrawiam
--AS

------=_NextPart_001_0006_01C3135E.123C6F70
ontent-Type: image/gif;
  name="IMSTP.gif"
Content-Transfer-Encoding: base64
Content-ID: <871735ED-400C-408A-AF7F-16B86E999352>
```

```
R0lGODlhFAAPALMIAP9gAM9gAM8vAM9gL/+QL5AvAGAvAP9gL////wAAAAAAAAAAAAAAAAAAAAA
AAAAACH/C05FVFNDQVBFMi4wAwEAAAAh+QQJFAAIACwAAAAAFAAPAAAEVRDJSaudJuudrxlEKI6E
URlCUYyjKpgYAKSgOBSCDEuGDKgrAtC3Q/R+hkPJEDgYCjpKr5A8WK9OaPFZwHoPqm3366VKyeRt
BAVkAAgALAAAAAAUAA8AAAQSEMlJq7046827/2AojmRpnmgEADs=

------=_NextPart_000_0005_01C3135E.123C6F70--
```

Przykład: reprezentacja liczb w maszynie cyfrowej

Liczby zmiennopozycyjne

- Za pomocą N bitów można zdefiniować co najwyżej 2^N różnych ich kombinacji, a co za tym idzie można zdefiniować 2^N różnych liczb. W przypadku maszyn liczących mówimy o liczbach **zmiennopozycyjnych** (*floating point numbers*) a nie o **liczbach rzeczywistych**.

- Liczby zmiennopozycyjne *są przybliżeniem* liczb rzeczywistych z następujących dwóch powodów: 1) dla wielu liczb rzeczywistych *nie istnieje* odpowiadająca im co do dokładnej wartości liczba zmiennopozycyjna i w związku z tym wybierana jest liczba „najbliższa”. 2) Wszystkie operacje arytmetyczne wprowadzają błąd zaokrąglenia (por. przykład niżej).

- Wymagania dotyczące zapisu zmiennopozycyjnego: zbiór liczb powinien być możliwie „gęsty”, rozstęp powinien być możliwie duży, liczba bitów użyta do reprezentacji winna być *stała* i możliwie mała, liczby takie winny w maksymalnym stopniu „zachowywać się jak liczby rzeczywiste”.

Liczba zmiennopozycyjna jest zapisywana jako trzyelementowy obiekt: bit znaku, wykładnik (e) i mantysa (f):
$x = f \times (2^e)$
gdzie: e jest liczbą, a f jest liczbą rzeczywistą. Mantysa może dodatkowo spełniać warunek normalizacji, np.:
$1 <= |f| < 2$ (IEEE754) lub $1/2 <= |f| < 1$ (DEC)
Przykład: Załóżmy, że maszyna liczy w systemie dziesiętnym używając mantysy o dwóch cyfrach znaczących. Wtedy, np:

```
+ 0,12E+02
  0,34E+00 = 12,00E+00
           +  0,34E+00 = 12,34E+00
                       =  0,12E+02
```

Wniosek: tak naprawdę zamiast liczenia X + Y, maszyna wykonuje obliczenie według schematu: round(round(X) + round(Y)).

Przykład: reprezentacja znaków graficznych

Kodowanie 7- i 8-bitowe

ANSI i Latin-1. Standard ISO-8859-2 na kodowanie jednobajtowe. Popularne, ale niezgodne ze standardem rozwiązania specyficzne dla różnych producentów: CP852, CP1250 (Microsoft). Kodowania wielobajtowe.

Unicode i ISO 10646 (UCS – Universal Character Set)

ISO 10646 definiuje kodowanie 4 bajtowe (teoretycznie 2^{32} znaków = 4,294,967,296 znaków). Póki co zagospodarowano pierwsze 65,534 (od 0x0000 do 0xFFFD). Podzbiór ten jest nazywany BMP (*Basic Multilingual Plane*). UCS definiuje numer znaku i jego oficjalną nazwę, np: A = U+0041 = "Latin capital letter A".
Unicode to projekt różnych producentów komputerowych powołany w celu opracowania standardu kodowania znaków. Ponieważ przedmiot standaryzacji jest identyczny z tym, którym zajmuje się ISO 10646, oba ciała koordynują swoje wysiłki i oba standardy są w zasadzie identyczne (praktycznie identyczne).

UTF-8

UTF to sposób kodowania znaków za pomocą zmiennej liczby bajtów. Znak z zestawu ASCII (U+0000 – U+007F) jest kodowany za pomocą jednego bajtu a pozostałe znaki za pomocą ciągu od dwóch do sześciu bajtów. Bajt „zaczynający się" od bitu 0 to znak ASCII, pozostałe bajty są interpretowane jako kody znaków spoza ASCII.

```
0x00000000 -- 0x0000007F  =>  0xxxxxxx
0x00000080 -- 0x000008FF  =>  110xxxxx 10xxxxxx
0x00000800 -- 0x0000FFFF  =>  1110xxxx 10xxxxxx 10xxxxxx
0x00010000 -- 0x001FFFFF  =>  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0x00200000 -- 0x03FFFFFF  =>  111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0x04000000 -- 0x7FFFFFFF  =>  1111110x 10xxxxxx 10xxxxxx \
                                       10xxxxxx 10xxxxxx 10xxxxxx
```

Technologia informacyjna

- **Technologia informacyjna** (IT) obejmuje trzy podstawowe funkcje: *przetwarzanie*, *przechowywanie* i *przesyłanie* informacji.

  Przykład: Intel, Sun Microsystems, Compaq specjalizują się w przetwarzaniu, EMC w przechowywaniu a Cisco w przesyłaniu. IBM to przykład firmy zajmującej się przetwarzaniem i przechowywaniem.

- **Dobra materialne i niematerialne**. Dobra materialne posiadają takie cechy fizyczne, jak: masa, temperatura itp. Dobro niematerialne ma charakter abstrakcyjny: pomysł, pogląd, cena, informacja.
- **Hardware** (sprzęt) jest materialną częścią IT, działanie tej części wynika bezpośrednio z praw fizyki: elektroniki, optyki. **Software** (oprogramowanie) jest częścią niematerialną. Software-hardware są zamienne, np. można sobie wyobrazić działający system IT zawierający wyłącznie sprzęt.

  Komputer jest produktem, którego funkcjonalność nie jest określona w momencie jego wytworzenia, ale później po dołączeniu oprogramowania. **Programowalność** to główna cecha maszyny cyfrowej.

Prawo Moore'a

Moc obliczeniowa komputerów podwaja się co 18 miesięcy. Albo: Wydajność sprzętu IT na jednostkę kosztu rośnie wykładniczo.

Technologia	Miara	Podwojenie (m-ce)	% rok	Mnożnik 10 lat
Szybkość przetwarzania danych (układ scalony)	Bity/t	18	59%	102
Szybkość przetwarzania danych (komputer)	Bity/t	21	49%	49
Pojemność zapisu	Bity	18	59%	102
Szybkość odczytu/zapisu	Bity/t	36	26%	10
Szybkość przesyłu	Bity/t	12	100%	1024

Źródło: Messerschmitt i Szyperski, *Software ekosystem*, 2003

Warstwy technologii informacyjnej

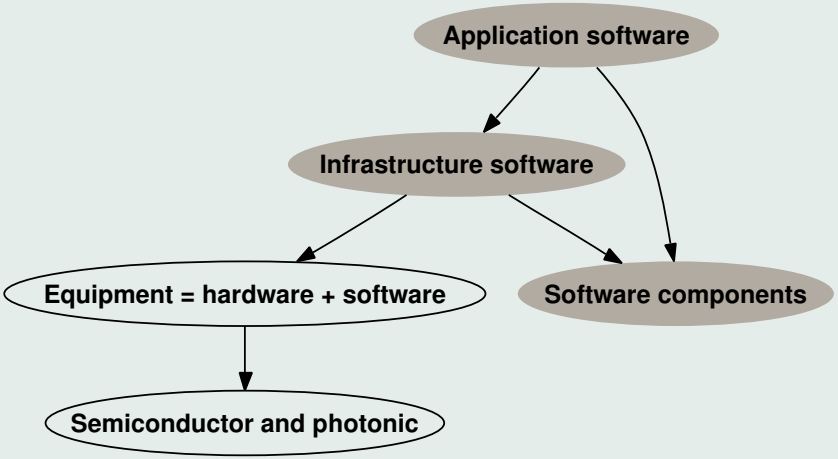
- **półprzewodniki**, itp. elementy;
- **wyposażenie**, tj. sprzęt + oprogramowanie wbudowane (embedded software), np. komputer, ruter;
- **oprogramowanie strukturalne**, zapewnia niezbędne usługi warstwie aplikacji, np. system operacyjny. Jednym z zadań jest oddzielenie warstwy aplikacji od szczegółów warstwy sprzętowej. Pozwala to m.in. na niezależny i nieskoordynowany rozwój *wyposażenia* i *aplikacji*;

- **Warstwa aplikacji**, zapewnia usługi niezbędne dla określonego użytkownika końcowego.
- Rodzaje oprogramowanie: wbudowane, strukturalne, komponentowe, aplikacyjne. Oprogramowanie wbudowane jest ważną częścią *wyposażenia* związanego z przechowywaniem i przetwarzaniem danych.

  Przykład: telefon komórkowy, system operacyjny, program poczty elektronicznej, edytor.

Klasyfikacja oprogramowania

- **Podział ze względu na funkcję: systemowe, narzędziowe, użytkowe**. Systemy operacyjne wraz z oprogramowaniem pomocniczym (wspomagające); Podstawową funkcją SO jest zarządzanie zasobami komputera. Zasoby to: z. sprzętowe (czas procesora, pamięć operacyjna, urządzenia we/wy); z. systemowe (programy wykonywalne, pliki, pamięć operacyjna).
- **Oprogramowanie wspomagające**: translatory języków oprogramowania, fonty, skrypty HTML etc.
- **Narzędziowe**: Edycja tekstu, zarządzanie bazami danych, opracowywanie arkuszy kalkulacyjnych



Rysunek 1: Warstwy technologii informacyjnej

- **Użytkowe**: tj. wykorzystywane przez specjalistów
- **Maszyna rzeczywista i maszyna wirtualna** Maszyna rzeczywista to sprzęt i mikroprogram nią sterujący. Maszyna wirtualna. Maszyna użytkownika. System operacyjny to maszyna wirtualna najniższego poziomu. Podstawową funkcją jest zarządzanie zasobami komputera.
- **Tryby pracy** Pośredni (wsadowy, batch), bezpośredni (interakcyjny,

on-line), w czasie rzeczywistym (real time).

- **Podział SO ze względu na możliwość realizacji wielu procesów i pracy wielu użytkowników**. Z tego względu systemy dzielimy na jedno/wielo programowe, jedno/wielo dostępne.

Tworzenie oprogramowania

- Tworzenie oprogramowania to działalność wieloaspektowa. Tworzenie oprogramowanie $\neq$ programowanie, tak jak tworzenie filmu nie jest tożsame np. pracą ekipy zdjęciowej na planie [M&S2003].
- Tworzenie oprogramowania to: wyszukiwanie/ocena możliwości/okazji biznesowych, podejmowanie decyzji biznesowych (Producent); opracowanie funkcjonalności (Scenarzysta); uściślenie i udoskonalenie pomysłów z potencjalnymi użytkownikami (*focus groups*); utworzenie architektury (*storyboard*); zarządzanie zespołem programistów (reżyser); implementacja oprogramowania, tj. programowanie (praca na planie, obróbka w studio); testowanie (prescreening); obsługa (zarządzanie sprzedażą, wersje obcojęzyczne, alternatywne media itp...) [M&S2003].

  focus group – noun, a group of people assembled to assess a new product, political campaign, television series, etc.

storyboard – noun, a sequence of drawings representing the shots planned for a film or television production.

- Celem powyższych działań jest zaspokojenie potrzeb użytkownika i/lub rozwiązanie jego problemów. Kreatywność w tworzeniu oprogramowania to identyfikacja nowych możliwości i nowych sposobów ich realizacji. Wymaga to zrozumienia możliwości/ograniczeń technologii IT, znajomości użytkowników oraz reguł rynku IT.
- Tworzenie oprogramowanie jest zarówno wyzwaniem technicznym jak i organizacyjnym.

Przykład: Aplikacja typu edytor/arkusz kalkulacyjny wymaga zespołu składającego się z ca. 100 członków, wytwarzającego kolejną wersję co kilka lat.
Duży projekt oprogramowania strukturalnego, np. system operacyjny wymaga zespołu od 1000 do kilku tysięcy członków, wytwarzającego kolejną wersję co kilka lat.
Podział zespołu wg. zasadniczych kategorii wygląda następująco: 30% deweloperzy (programiści), 30% testerzy, 30% menażerowie (tworzenie specyfikacji, nadzór), 10% inni (usability, accessibility, internationalization etc...)

Użytkownicy

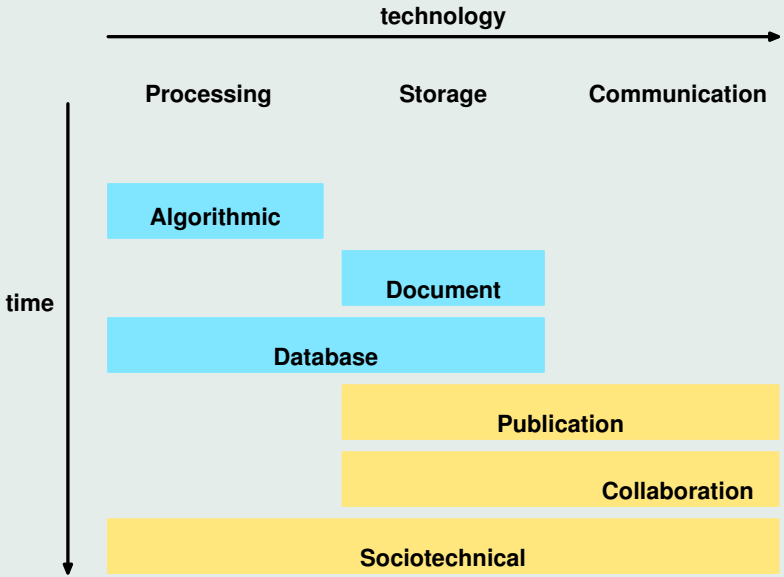
- Funkcją oprogramowania aplikacyjnego jest „zaspokajanie potrzeb" użytkowników końcowych: jednostek, organizacji, społeczeństwa.
- Historycznie trzy obszary IT: przetwarzanie, przechowywanie, przesyłanie.
- Typ użytkownika: **jednostka** (*algorithmic*, *document*, *publication*), **grupa jednostek** (aplikacje grupowe, *database*, *collaboration*), **systemy IT w organizacji** (dwie główne kategorie: transakcyjne, tj. automatyzujące procesy biznesowe oraz systemy wspomagania zarządu)
- podział oprogramowania ze względu na *specification-driven*, oraz *satisfaction-driven*.

Modele tworzenia oprogramowania

Sekwencyjny (kaskadowy, waterfall); Spiralny; Open-Source (community-based).

Sekwencyjny (kaskadowy, waterfall)

1. Konceptualizacja



Rysunek 2: Klasyfikacja oprogramowania aplikacyjnego

Wypracowanie wizji funkcjonalności oprogramowania, uzasadnienie dlaczego jest to dobra 'okazja biznesowa'

Dokument, który powinien przekonać zarząd do przydzielenia środków na następną fazę

2. Analiza

   Uściślenie pomysłu, w celu uzasadnienia wydatków koniecznych w następnych fazach. Dokładne określenie funkcjonalności

   Biznes plan, łącznie z szacowanymi kosztami projektowania/wykonania, utrzymania i rozwoju. Ocena potencjalnego rynku

3. Architektura

   Podział zadania na moduły

   Plan określający poszczególne moduły, wymagania dla każdego modułu, określenie sposobu współdziałania modułów między sobą

4. Implementacja

   Programowanie poszczególnych modułów

   Działające aplikacje-moduły

5. Integracja

Połącznie modułów w jedną całość. Często integracja obejmuje zewnętrzne komponenty

Działająca aplikacja

6. Testowanie

   Sprawdzenie czy aplikacja działa zgodnie z założeniami i/lub potrzebami użytkowników

   Oprogramowanie gotowe do zainstalowania

7. Utrzymanie (maintenance)

   Poprawki i ulepszenia na podstawie raportów użytkowników/operatorów

   Poprawki (*service pack*)

8. Rozwój (upgrade)

   W odpowiedzi na reakcje użytkowników ulepszenie oprogramowania

   Nowa wersja oprogramowania

Krótka historia systemów operacyjnych

- **Ewolucja SO jest zdeterminowana relatywnym kosztem sprzętu (hardware) i użytkowników**

Relatywny koszt sprzętu był astronomiczny na początku (w porównaniu do użytkowników) a później zawsze się obniżał. Wielkość ta była jednym z ważnych przyczyn zmian w sposobie tworzenia SO i oprogramowania w ogóle.

Dawniej: Drogi sprzęt, tani użytkownicy. **Cel**: maksymalizacja wykorzystania sprzętu.

Teraz: Tani sprzęt, drodzy użytkownicy. **Cel**: tworzenie systemów łatwych w użyciu.

- **Podział systemów operacyjnych na generacje** *Adrew S. Tannenbaum,* Modern Operating Systems*, Prentice Hall, 2000*

  Pierwsza (1945–1955), druga (1955–1965), trzecia (1965–1980), czwarta (1980–).

- **Pierwsza generacja:** Komputery oparte o lampy próżniowe programowane za pomocą tablic przełączników lub (później) kart dziurkowanych. Programowanie w języku maszynowym, często poprzez zmianę ustawień na tablicy przełączników. Pojęcie systemu operacyjnego jest nieznane.

- **Druga generacja:** Komputery budowane w oparciu o tranzystory są na tyle niezawodne że są sprzedawane (podział na projektantów, producentów, operatorów i programistów). Praca jednozadaniowa w trybie wsadowym. Problemy z wykorzystaniem CPU.

- **Trzecia generacja:** Komputery budowane w oparciu o układy scalone (o niskiej skali integracji). Komputery z serii IBM/360. Znacznie tańsze od poprzedników. Wprowadzenie koncepcji wieloprocesowości w celu zwiększenia wykorzystania CPU. Wprowadzenie buforowania operacji wejścia/wyjścia (spooling). Ciągle brak interakcji.

  CTSS pierwszy SO z podziałem czasu (*timesharing*). MULTICS: wieloprocesowy system operacyjny z podziałem czasu. Zaprojektowany na komputer o możliwościach porównywalnych z PC 386! obsługiwał kilkuset użytkowników.

  Powstanie minikomputerów: serie DEC PDP. Pierwszy z serii: PDP-1 (1961 r.) kosztował 120,000 $ (mniej niż 5% ceny komputera IBM 7094). Był wyposażony w pamięć 4K 18 bitowych słów.

  Powstanie Unixa (PDP-7)

- **Czwarta generacja:** Komputery budowane w oparciu o układy scalone o dużej skali integracji.

  - **Intel** konstruuje pierwszy procesor 8-bitowy 8080. Gary Kildal opracowuje system operacyjny CP/M do obsługi mikrokomputera wyposażonego w 8 calowy dysk miękki (dyskietka). IBM konstruuje IBM PC i kupuje licence od niejakiego **B. Gatesa** na system operacyjny DOS + interpretator języka BASIC do tegoż (1980). BTW: system DOS

nie jest pomysłem Gatesa ale jest przez niego odkupiony (za 50,000$?) od firmy Seattle Computer Products.

– **Dug Engelbert** ze Stanford Research Institute opracowuje w latach 60-tych koncepcję GUI, czyli sposobu interakcji z komputerem w oparciu o okna, ikony, menu i mysz. Koncepcja ta zostaje rozwinięta w **Xerox PARC**.
– **Steve Jobs** po wizycie w Xerox PARC uświadamia sobie, że GUI umożliwia zbudowanie komputera „dla ludzi" (Kierownictwo Xerox jakoś na to nie wpadło) i projektuje Apple Macintosh odnosząc wielki sukces komercyjny.

  Apple Macintosh komputer przyjazny dla użytkownika (user friendly).
– Microsoft kopiuje pomysł Jobsa sprzedając graficzną nakładkę na DOS-a zwaną Windows (1985). W 1995 roku Microsoft opracowuje „prawdziwy" system okienkowy Windows 95, rozwijany później i sprzedawany w wersjach 98, NT, 2000 Millenium etc.

  Inny system wykorzystywany na komputerach PC i stacjach roboczych to Unix w jego różnych wariantach (zwłaszcza Linux).

Unix

- Wydajny, wielodostępny i wielozadaniowy system operacyjny

- Powstał w 1969 w firmie AT&T Bell Labs w oparciu o system Multics, opracowany wcześniej przez zespół z GE, ATT i MIT
- Napisany w języku C (Ritchie/Thompson) – przenośność
- Model warstwowy struktury systemu Unix: jądro, powłoka, warstwa aplikacji Powłoka (shell): zapewnia interfejs pomiędzy użytkownikiem a jądrem. Interpretator poleceń użytkownika. Rodzaje powłok: Bourne'a (bash), Korna (ksh).
- **Darmowa licencja** dla uniwersytetów, dostępny wraz z kodami źródłowymi i prawem do ich modyfikacji
- Wersje Unixa: BSD (rozwinięta przez Uniwersytet Kalifornijski w Berkeley), System V (ATT), Xenix (MS), HP-UX, SunOS, AIX itp.
- Standard POSIX (Portable Operating System Interface for Computer Environments). Definiuje, jak systemy uniksowe mają działać, określa wywołania systemowe i interfejsy
- System GNU/Linux, System uniksopodobny zgodny ze standardem POSIX
- **Projekt GNU** opracowanie kompletego, powszechnie dostępnego systemu operacyjnego, zgodnego z Uniksem. Projekt GNU/HURD.

Fundacja FSF. Licencje podobne do GPL. Ochrona patentowa na oprogramowanie

- **Copyleft i GPL** Kopiowanie i dystrybucja oprogramowania wraz z kodami źródłowymi, nie są ograniczone. Razem z programem **musi** być rozpowszechniany kod źródłowy. Oprogramowanie powstałe w oparciu o oprogramowanie na licencji GPL musi być rozpowszechniane także na takich warunkach. Przykłady: `gcc`, `emacs`

Programowanie

- Akcja to zdarzenie o skończonym czasie trwania i zamierzonym efekcie (skutku). Akcja może być opisana w terminach jakiegoś języka; opis akcji nazywamy *instrukcją*.

  Akcja dotyczy obiektu, wynikiem akcji jest zmiana stanu obiektu.

- Akcję podzielona na wiele akcji składowych nazywamy *procesem*. Proces sekwencyjny, to proces, którego składowe występują po sobie kolejno w czasie. Ciąg instrukcji opisujący proces nazywamy *programem*.

- Akcje są wykonywane (proces jest wykonywany) zgodnie z programem przez *procesor* (człowiek lub maszyna).

Przykład:
Dana jest istrukcja: *oblicz iloczyn dwóch liczb naturalnych* x *oraz* y. *Wynik oznacz przez* z.
Gdyby procesor rozumiał w/w instrukcje, co oznacza, że znałby pojęcie „liczba naturalna" oraz „oblicz iloczyn", żadne dalsze wyjaśnienie nie jest potrzebne. Załóżmy zatem dodatkowo, że: procesor: nie rozumie języka naturalnego tylko pewien język formalny; nie potrafi mnożyć, lecz potrafi dodawać.
Przy takim założeniu program można opisać nieformalnie w dwóch krokach:

```
1. z:=0 u:=x
2. Powtarzaj instrukcje z:=z+y; u:=u-1 aż do u=0
```

Podane przykłady pokazują, że każdy program opisuje ciąg przekształceń stanów zmiennych. Opis takiego wzorca zachowania bez odwołania się do konkretnego procesora nazywamy *algorytmem*. Algorytm opisany w sposób umożliwiający wykonanie go przez określony typ procesora nazywa się *programem*.
Przykład:
Podziel liczbę naturalną x przez liczbę naturalną y; oznacz iloraz całkowity przez q i resztę przez r.
$x = q * y + r$ gdzie: $0 \leq q < y$
Załóżmy, że procesor nie potrafi wykonać tak zdefiniowanej operacji dzielenia; potrafi natomiast dodawać i odejmować:

```
1. q := 0; r:= x
```

```
2. dopóki r >= y, powtarzaj:
     q := q+1; r := r-y
```

Obydwa przykłady opisują procesy sekwencyjne, typowe dla programowania.

Krótki przegląd budowy komputerów

We współczesnych typach komputerów można wyodrębnić dwie główne składowe:

- **Pamięć**. Zawiera ona zakodowane obiekty reprezentujące *dane*. Wydajność pamięci to jej pojemność oraz szybkość pobierania/zapisywania danych. Pojemność pamięci jest zawsze skończona.
- **Procesor**. Wykonuje: dodawanie, mnożenie, porównywanie itp. proste instrukcje. Dane są pobierane z pamięci; wyniki są zapisywane w pamięci.

Procesor pamięta tylko dane niezbędne do wykonania operacji. Jednostki pamięci procesora to *rejestry*.
Przykład: oblicz: $a \cdot b + c \cdot d$

```
r1 := a
r2 := b
r1 := r1 r2
z  := r1
r1 := c
r2 := d
r1 := r1 r2
r2 := z
r1 := r1 + r2
```

r1, r2 – rejestry procesora, z – wynik pośredni przechowywany w pamięci. Wynik obliczenia znajduje się w rejestrze r1. Obliczenie wyrażenia realizuje zatem program zawierający tylko trzy rodzaje instrukcji: pobranie danych z pamięci do rejestru; operacje arytmetyczne na danych w rejestrach; przesłanie danych z rejestrów do pamięci.
Program i dane są umieszczone w pamięci komputera. Instrukcje programu są kodowane.

Programowanie

- Aż do końca lat 50-tych programowanie polegało na *kodowaniu*, tj. tłumaczeniu instrukcji na postać dwójkową, ósemkową lub szesnastkową i na ich szeregowaniu w żądany program. Taki sposób postępowania ma następujące wady: programy są dopasowane do

konkretnego komputera (wymiana programów między różnymi komputerami niemożliwa); tak zwany *kod maszyny*, będący liniowym ciągiem istrukcji jest mało czytelny dla człowieka. Uwypuklenie struktury stanowi podstawę projektowania złożonych programów.

- Powyższe braki (i rozwój sprzętu) doprowadziły do opracowania tzw. „języków programowania wyższego poziomu". Za pomocą tych języków instruuje się pewien wyidealizowany, hipotetyczny komputer, opracowany nie pod kątem możliwości współczesnej technologii ale zgodnie z nawykami i możliwościami człowieka.
- W rezultacie: komputer A – „prawdziwy" niewygodny dla człowieka, komputer B – „przyjazny" dla człowieka ale istniejący tylko na papierze. Oprogramowanie wiąże oba w/w typy komputerów: program C wykonany na komputerze A potrafi *przetłumaczyć* program napisany dla komputera B na program zrozumiały dla komputera A. Program C nazywamy kompilatorem (*compiler*) bądź interpretatorem (*interpreter*).
- Użycie kompilatora uwalnia programistę od znajomości szczegółów komputera A, ale nie zwalnia go od znajomości ograniczeń komputerów (skończona pojemność pamięci, dokładność, czas obliczeń).

Cztery generacje języków programowania

- *Języki maszynowe*: Instrukcje zapisywane jako ciągi zer i jedynek, Programowanie pracochłonne i podatne na błędy, wymaga wysoko wyszkolonych programistów

  ```
  1101 11001011
  1011 10111010 11001011
  1110 10100011
  ```

- Asemblery: Zastąpienie ciągów zer i jedynek nazwami symbolicznymi (zarówno w przypadku instrukcji, jak i adresów danych). Do zamiany tak przygotowanego programu potrzebny był program tłumaczący zwany asemblerem.

  ```
  LDA B
  ADD B C
  STA A
  ```

- Języki wysokiego poziomu (trzeciej generacji – 3GL): składnia różna od składni języka maszynowego. Języki czwartej generacji (4 GL): wyspecjalizowane języki o instrukcjach zorientowanych na problem, którego dotyczą.

  ```
  a = b +c
  ```

Języki proceduralne (imperatywne), języki deklaratywne:

Języki deklaratywne (LISP, języki funkcjonalne). Programowanie w językach funkcyjnych polega na definiowaniu układów funkcji, najczęściej wzajemnie uzależnionych, a następnie wyliczaniu wartości tych funkcji dla zadanych argumentów. W „czystych językach funkcyjnych" nieobecne są instrukcje, w szczególności nie ma instrukcji przypisania. Zmienne w j. funkcyjnych są symbolami reprezentującymi wartości, ale gdy ustalimy jaką wartość ma mieć zmienna będzie ona mieć tę wartość przez cały okres swojej ważności. Nie można zmienić tej wartości przez wykonanie instrukcji *przypisania*.
Programy funkcyjne łatwiej niż imperatywne poddają się analizie matematycznej, gdyż obiekty w nich występujące mają prostszą interpretację matematyczną.
Większość języków proceduralnych jest potomkami *Algolu*. Wspólne cechy tych języków to: algebraiczna notacja wyrażeń, zakres zmiennych kontrolowany za pomocą struktury blokowej programu, procedury/funkcje wykonywane dla obliczenia zmiennej lub osiągnięcia jakiegoś efektu.
Przykłady: Fortran, Algol, Pascal, C, C++
Błędy składniowe i błędy logiczne (semantyczne)

Języki Skryptowe

Skrypt to program, którego kod źródłowy jest jednocześnie wykonywalny.

Wykonanie skryptu: proste: kod źródłowy -> wykonanie; zamiast kod źródłowy -> kompilacja -> kod wynikowy, w przypadku języków kompilowanych.
Do wykonania skryptu potrzebny jest **interpretator**.
Zalety: szybkość uruchamiania i modyfikacji; wady: szybkość działania.

Cechy języków skryptowych

- Dostępne na zasadach *Open Source* [Dokumentacja, Biblioteki modułów]. Dostępne na wielu platformach systemowosprzętowych.
- Brak konieczności kompilacji – szybkość uruchamiania programów. [Większość języków skryptowych nie jest „klasycznymi" interpretatorami, tj. nie interpretuje programu linia po linii ale raczej kompilują program na specyficzny kod bajtowy *bytecode*, z tym, że nie wymaga to świadomego działania użytkownika.]
- Brak sztywnej struktury programu. Brak ścisłej typizacji danych [Zmienne i typy zmiennych nie muszą być deklarowane; może to prowadzić do problemów w przypadku większych aplikacji.]
- Gotowe do wykorzystania skomplikowane struktury danych [Napisy, tablice, stosy, kolejki, tablice asocjacyjne, wyrażenia regularne. Łatwość manipulowania danymi.]

- Mechanizmy zarządzania wieloma procesami w środowiskach wielozadaniowych, wieloprocesorowych i rozproszonych (sieciowych). [Języki skryptowe są wykorzystywane do obsługi serwisów internetowych, za pośrednictwem standardu CGI].
- **Zastosowanie języków skryptowych**: **Łączenie różnych programów, transformacja danych** [Efektywne mechanizmy wejścia-wyjścia, elastyczne struktury danych]; **Prototypowanie**; **Zarządzanie serwisami WWW** [mod_perl, ZOPE, CGI, PHP, JavaScript etc.]; **Graficzne Interfejsy Użytkownika**.

Języki Skryptowe: przykłady

shell: skrypty shellowe typowo nie są „samowystarczalne" – uruchamia się w nich typowo najróżniejsze programy do wykonania właściwego zadania. Kod skryptu zwykle służy do połączenia kolejności wywołania programów w jedną całość. Podstawowe mechanizmy to **podstawienie** *substitution*, cytowanie *quoting* oraz potoki *pipelines*.
Perl: CPAN, wyrażenia regularne, najpopularniejszy język do tworzenia interaktywnych serwisów WWW
Python: Elegancki język wspierający z założenia obiektowy paradygmat programowania.
Tcl: zdecydowanie najprostszy z popularnych języków skryptowych. Wszystko jest tekstem: jeden typ danych – napis. Skryptowy standard

GUI: Tk.
PHP: