

Podstawy, koncepcje i idee programowania

Programowanie

Z programowaniem mamy do czynienia od wielu lat zarówno w świecie IT jak i codziennym. W świecie IT to rzecz normalna, w pozostałych obszarach nie zawsze to widać. Przykład: instalator montujący system alarmowy programuje centralę alarmową poprzez najpierw ustalenie co i jak ma działać, kolejno stworzenie np. pliku z zasadami działania centrali a tym samym całego systemu alarmowego i ostatecznie wgrywając czy wprowadzając ten plik do centrali. Centrala ładuje tak zapisany program do pamięci i oddaje sterowanie do tegoż kodu tym samym rozpoczynając działanie według określonego algorytmu. Podobnie dzieje się w komputerach samochodowych, przemysłowych i wielu innych urządzeniach elektronicznych czy elektrotechnicznych. Zazwyczaj program działa na zasadzie wejście -> obsługa wprowadzonych danych -> wyjście.

Algorytmy

Jeśli mamy jakiś zestaw czynności do wykonania to za zwyczaj wymaga on określenia czy zdefiniowania kolejnych kroków i ich warunków wykonania. Czasem kroki te muszą być wykonywane kolejno czyli szeregowo (sekwencyjnie, jeden za drugim) a innym razem pojawi się konieczność wykonania ich jednocześnie czyli równolegle. Ten zestaw czynności to może być cokolwiek: upieczenia ciasta, dostanie się do odległej miejscowości, poskładanie silnika samochodowego itd. W wielu przypadkach kolejność wykonywania działań jest krytyczna.

Tworzymy więc zazwyczaj zestaw kroków do wykonania w celu zakończenia planowanego procesu lub inaczej mówiąc tworzymy algorytm postępowania, w którym określamy co, jak, gdzie, kiedy zrobimy oraz czego użyjemy. Ten ostatni element można traktować jako nasze dane.

Tworzone algorytmy można zapisywać w różne sposoby: własnymi słowami, stosując wcześniej zdefiniowaną gramatykę czyli jakiś język zapisu algorytmu co często nazywa się pseudokodem, w sposób wizualny np. schemat blokowy czy inny.

Algorytm może być zatem opisany za pomocą:

- języka naturalnego
- schematu blokowego
- pseudokodu
- kodu w konkretnym języku programowania

Przykład:

upieczenia ciasta

- przygotuj odpowiednie składniki w odpowiednich wagach czy objętościach
- przygotuj naczynia
- wbij jajko do miski
- wsyp 0.1 kg cukru
- dopóki masa NIE jest płynna mieszaj (jeśli masa NIE jest płynna mieszaj dalej)
- wsyp 0.5 kg mąki, drożdży czy co tam trzeba
- mieszaj tak długo jak masa jest miękka
- odstaw ciasto i czekaj 0.5 h
- jeśli ciasto wyrosło daj do pieca, jeśli nie czekaj 15 minut i powtórz sprawdzenie
- włącz piec na 150 stopni
- czekaj 0.5 h
- sprawdź czy upieczone (co to oznacza?)
- jeśli tak to wyłącz piec i wyjmij ciasto, jeśli nie czekaj 10 min i powtórz sprawdzenie
- gotowe!

Albo inny algorytm:

- przygotuj listę zakupów
- przygotuj odpowiednią siatkę
- wyjdź z domu
- idź w stronę przejścia dla pieszych
- jeśli światło czerwone czekaj, jeśli zielone idź

- zdecyduj do jakiego sklepu idziesz (na jakiej podstawie?)
- jeśli osiedlowy przygotuj gotówkę, jeśli hipermarket kartę
- w sklepie odszukaj towary z kartki i wkładaj do koszyka (i tu może się pojawić problem wyboru najkrótszej trasy po sklepie, chyba, że idziemy z żoną wtedy problem wyboru jak najdłuższej trasy...)
- podejdź do kas
- [:
- przelicz ile osób czeka w każdej kolejce
- wybierz kasę z najkrótszą kolejką
- :]
- czekaj aż nadejdzie Twoja kolej czyli kasjerka powie "Dzień dobry"
- [jeśli masz dość pieniędzy płacisz i zabierasz towar, jeśli nie odkładasz część (obliczenie na ile towaru starczy kasy):]
- idź w stronę przejścia dla pieszych
- idź do domu
- wyłóż towar z siatki
- gotowe!

Można zauważyć, że pewne elementy naszego planu działania mogą się powtarzać przy każdej wizycie w sklepie. Przykładem jest przeliczenie ile osób stoi w poszczególnych kolejkach i wybór kasy z najmniejszą ilością klientów. Ujęto to w symbole [: :]. Taką procedurę postępowania można by jakoś zakodować i potem tylko automatycznie wywołać.

Dobrym przykładem może być utworzenie szablonu dokumentu firmowego:

- [:uruchom edytor tekstów
- stwórz nowy, pusty dokument
- wstaw nagłówek
- do nagłówka wstaw po lewej logo, pod logo nazwę firmy
- do nagłówka wstaw po prawej aktualną datę
- wstaw stopkę
- do stopki wstaw adres www, email oraz podziękowania:]
- w głównej sekcji dokumentu wpisz właściwy tekst

Powyższy zestaw czynności zna chyba każdy. Jest w nim wiele elementów powtarzających się w każdym takim dokumencie a co ważniejsze zawierają one identyczne dane (poza datą). Oczywiście można każdorazowo tworzyć taki dokument od zera ale po co? Zazwyczaj przygotowujemy sobie szablony lub makra jakie udostępnia edytor i elementy wypisane kursywą będą wykonywane automatycznie. To kolejny algorytm postępowania tym razem stworzenia dokumentu firmowego.

Jak widać algorytmy tworzymy bardzo często nawet nie zdając sobie z tego sprawy. Używamy także często pseudokodów jak np na kartce typu "co mam dziś załatwić".

Pseudokod

To sposób zapisu naszego algorytmu w nieistniejącym języku programowania najczęściej podobnym do naszego języka i gramatyki. Inaczej mówiąc pseudokod zachowuje charakterystykę prawdziwego kodu ale unika ścisłych jego reguł takich jak typy zmiennych, używanie specyficznych funkcji itp. Po co zatem używać pseudokodu? Aby uprościć i uporządkować proces programowania i budowania kodu właściwego. Poniżej prosty przykład warunku sprawdzającego podany PIN:

```
pobierz PIN
początek warunku
jeśli PIN := 000
idź do koniec warunku
jeśli PIN zgodny z wzorcem
realizuj zamówienie
w przeciwnym razie
{
```

```
    generuj komunikat o błędzie
    żądaj ponownego podania PIN
}
koniec warunku
```

Poniżej przykład algorytmu w pseudokodzie, który wczytuje i sumuje n liczb.

```
Start
Suma:=0
Podaj n
i:=0
Dopóki ( $i < n$ ) wykonuj:
{
    Wczytaj a
    Suma := Suma + a
    i := i + 1
}
Wypisz Suma
Koniec
```

Kolejny przykład do pseudokod działania bramki AND:

```
Start
Wczytaj A
Wczytaj B
Jeśli A równe jest 1 i B równe jest 1 to na wyjście podaj 1, w każdym innym przypadku na wyjście podaj 0
Stop
```

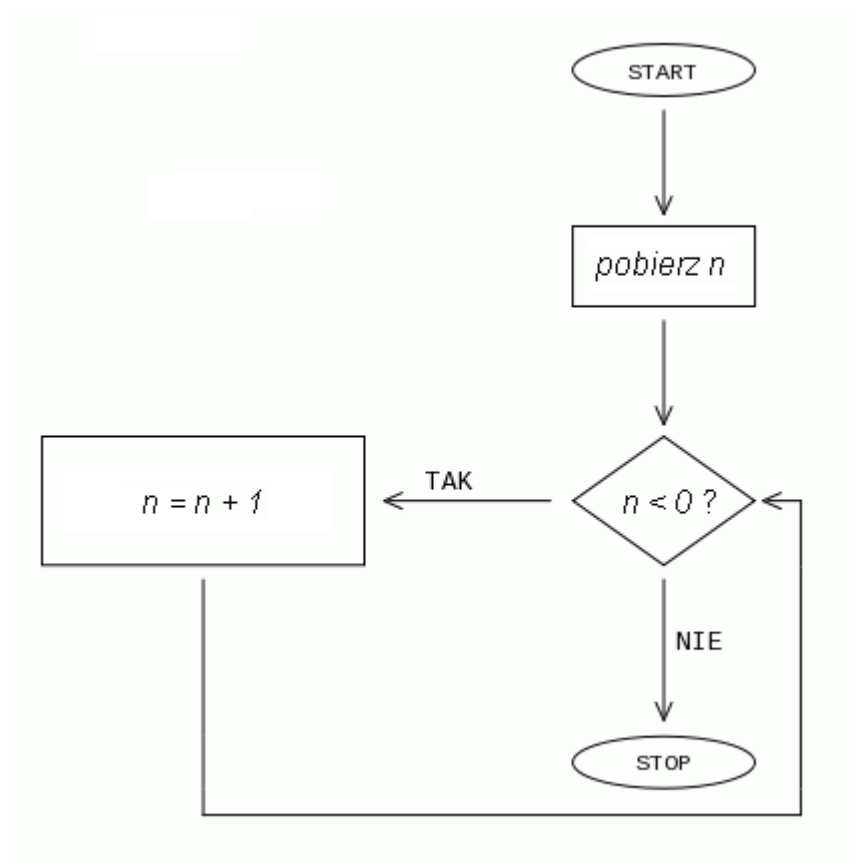
W pseudokodzie zazwyczaj stosuje się różnego rodzaju operatory takie jak przypisanie wartości $:=$ lub $=$, sekcje czy bloki kodu ograniczone nawiasami $\{ \}$ służące do lepszego grupowania zestawów logicznie powiązanych instrukcji i poleceń. Szczególnie ważne są nawiasy $\{ \}$ grupujące instrukcje. Tak jak w matematyce kolejność działań można wymuszać stosując nawiasy tak samo i tu, w zależności od tego czy dana instrukcja jest w bloku nawiasowym czy po za nim może się wykonać lub nie. Poniżej prosty przykład:

```
jeśli  $a > 0$  to
{
    czytaj x
     $x = x + 1$ 
}
jeśli  $a > 0$  to
    czytaj x
     $x = x + 1$ 
```

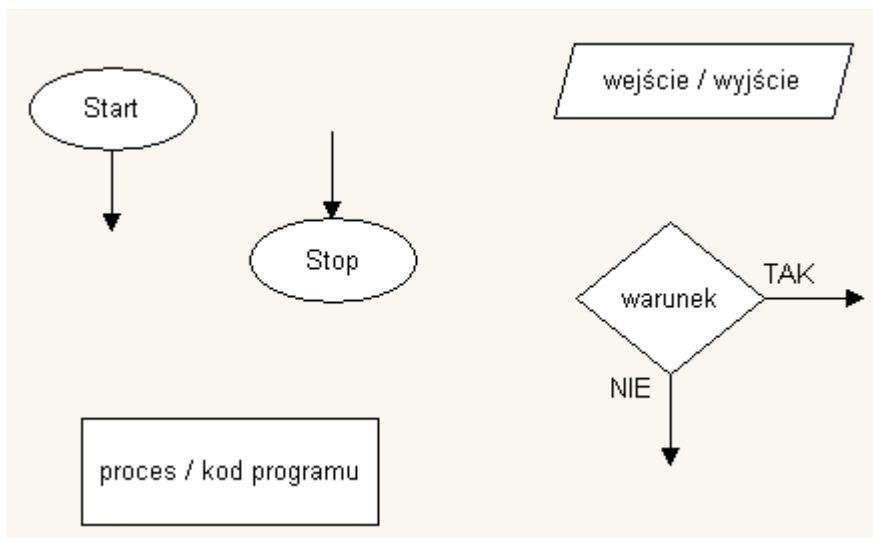
Jaka jest różnica pomiędzy tymi dwoma fragmentami pseudokodu? W pierwszym przypadku powiększenie x o 1 będzie miało miejsce tylko gdy $a > 0$ ale w drugim przypadku x zostanie powiększone o 1 niezależnie od tego jaką wartość ma a . Powodem tego stanu jest ujęcie obu instrukcji w nawiasy i wykonanie obu instrukcji zgodnie z warunkiem. W drugim przypadku tylko instrukcja bezpośrednio po warunku jest od niego zależna, druga instrukcja wykonuje się niezależnie od warunku. Wspomina się o tym ponieważ jest to bardzo częsty błąd logiczny, którego kompilator NIE wykrywa.

Schematy blokowe

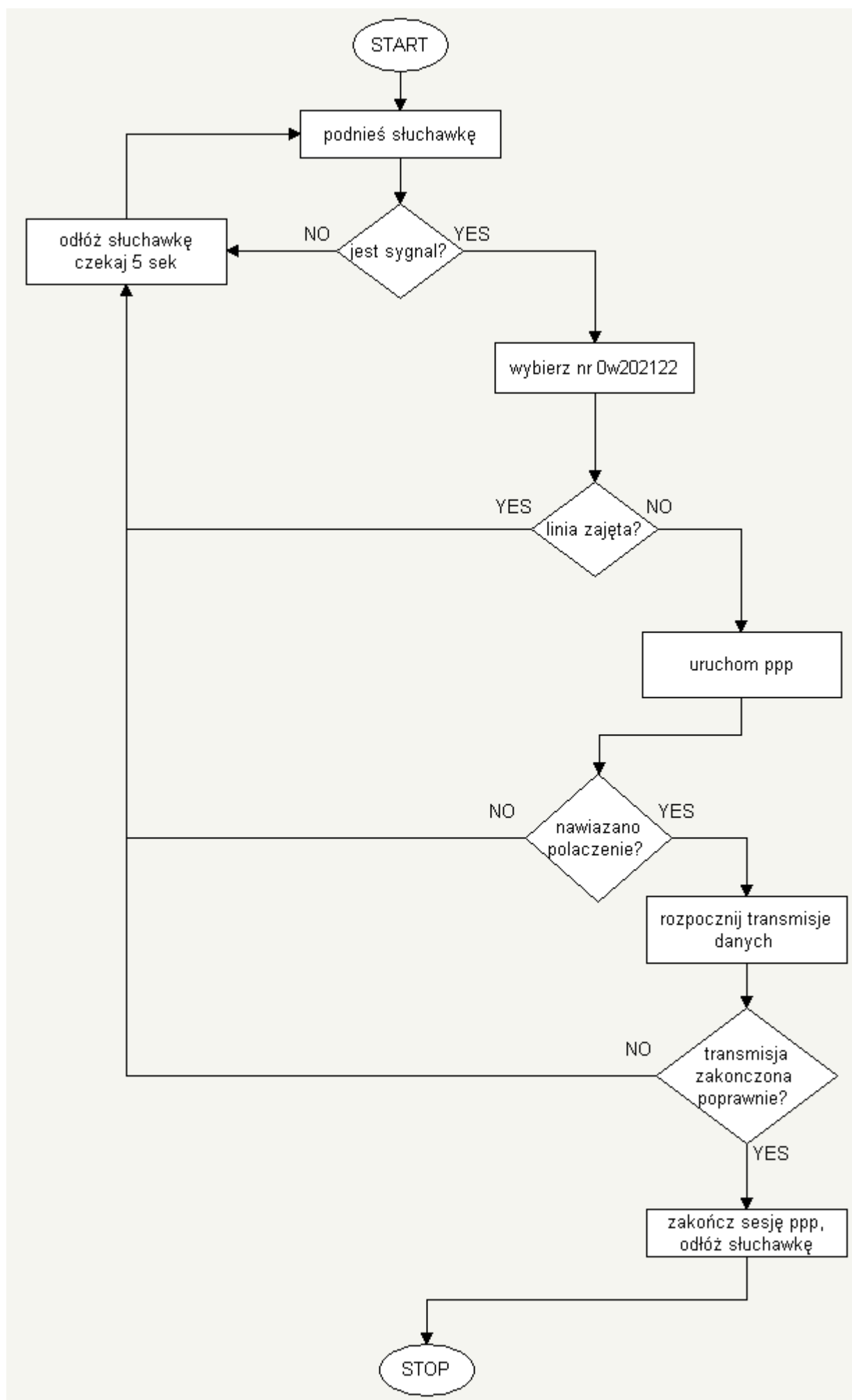
Ponoć dobry rysunek jest więcej wart niż tysiące słów tak więc ktoś wpadł na pomysł aby nasz algorytm wyrazić nie za pomocą potoku słów ale za pomocą rysunku.



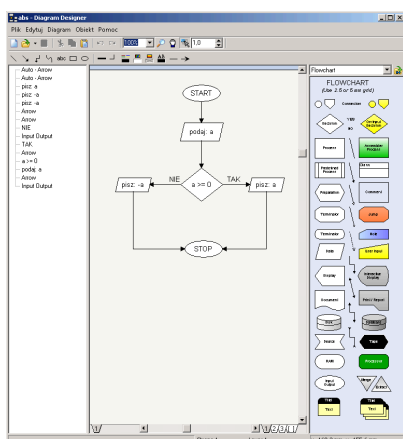
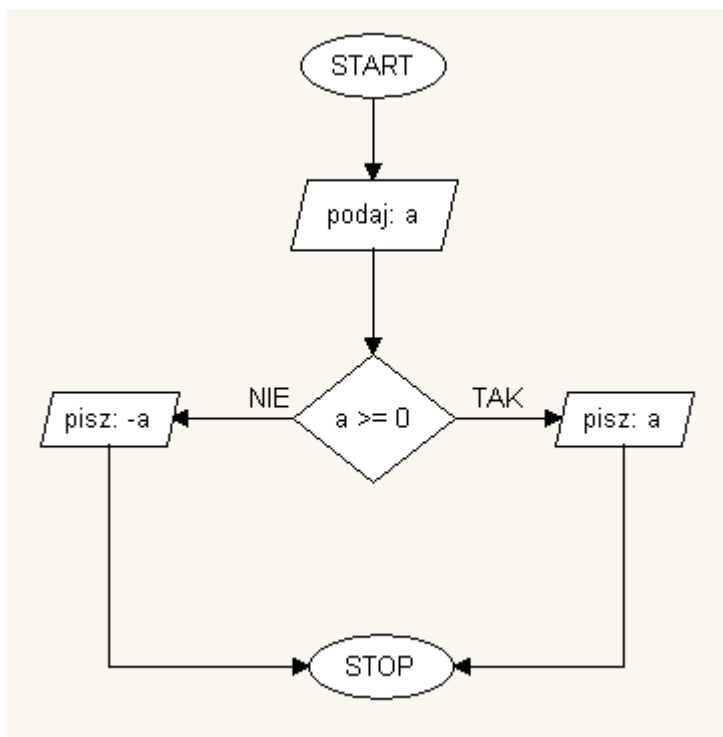
W schematach blokowych najczęściej używa się poniższych symboli, które odpowiadają czynnością jakie ma wykonać nasz algorytm.



Poniżej nieco bardziej skomplikowany przykład.



Poniżej coś bardziej konkretnego, diagram algorytmu wyznaczania wartości bezwzględnej:



Przydatnym programem może być prosty ale użyteczny Diagram Designer, darmowy do kreślenia diagramów oraz Dia prosty i wyposażony w wiele symboli. Można także użyć MS Visio, Magiczne bloczki, Diagram painter i inne. Niektóre pakiety obsługują język UML i pozwalają na wygenerowanie całkiem sporego kawałka kodu źródłowego w zadanym języku, co znacznie upraszcza proces tworzenia programów. Świetnym serwisem online jest <https://www.draw.io> - darmowy wyposażony w wiele przydatnych symboli.

Języki programowania

Generalnie rzecz biorąc językiem programowania można nazwać pewien zbiór zasad określających czy dany ciąg symboli (leksykalnych) tworzy program komputerowy. Przez program najczęściej będziemy rozumieć implementację algorytmu. W rzeczywistości istnieje wiele języków programowania, poniżej wymieniono kilka najpopularniejszych:

- C/C++ (fundament prawie każdego nowoczesnego języka programowania)
- C#
- Java
- Basic i Visual Basic
- Pascal i ObjectPascal
- Fortran
- PHP
- SQL
- HTML
- Python

- Perl
- JavaScript
- Ruby
- AutoLisp
- Algol

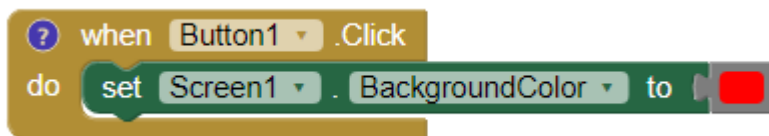
Języki programowania można podzielić na

- liniowe (Basic, Fortran)
- strukturalne (Pascal, C)
- obiektowe (ObjectPascal, C++, C#)
- sterowane zdarzeniami (VisualBasic)
- Skryptowe (PHP, HTML, Perl, języki powłok Linuxa, Windows PowerShell)

Można także dokonać podziału ze względu na umowną generację:

- pierwsza generacja: maszynowe niskiego poziomu czyli binarne kody odpowiadające rozkazom poszczególnych procesorów; kompletny brak przenośności, wysoka trudności kodowania, bardzo trudne wyszukiwanie i poprawianie błędów
- druga generacja: symboliczne tzw asemblery, nadal niskiego poziomu ale kody binarne zastąpiono mnemonikami czyli skrótami nazw kojarzących się z tym co dana instrukcja wykonuje; pojawia się tu nieco przenośności, i wraz z dodatkowym oprogramowaniem nieco łatwiejsze wyszukiwanie błędów
- trzecia generacja: wysokiego poziomu proceduralne lub obiektowe (C, C++, Pascal), jedno polecenie (instrukcja języka) jest tłumaczona na kilka lub nawet kilkanaście instrukcji maszynowych za pomocą kompilacji
- czwarta generacja: bardzo wysokiego poziomu tzw deklaratywne (SQL), gdzie skupiamy się na problemie nie na samej składni języka i sposobie kodowania
- piąta generacja: języki naturalne, sztucznej inteligencji (Prolog) bardzo zbliżone do naturalnego

Programowanie w AI2 opiera się na określaniu *zdarzeń* jakie mogą zajść w tworzonej aplikacji i odpowiednim ich obsłudze za pomocą *metod* wbudowanych w język i system AI2 lub utworzonych przez programistę. Jest to programowanie oparte na *sterowaniu zdarzeniami*. Zdarzeniem może być naciśnięcie przez użytkownika przycisku. Rola programisty (twórcy aplikacji) jest zaprogramowanie odpowiedniej reakcji np zmiana koloru tła ekranu. Poniżej przykład: po zajściu zdarzenia "naciśnięcie przez użytkownika przycisku Button1" kolor tła ekranu zmienia się na czerwony.



Kod źródłowy

To co tworzymy czyli piszemy w edytorach IDE czy innych, a co potem poddajemy procesowi przetwarzania (kompilacji czy interpretacji o czym dalej) to kod źródłowy. Co to oznacza to sugeruje chyba sama nazwa. Może to być program napisany w dowolnym języku programowania wysokiego czy niskiego poziomu. Jest to format dość czytelny dla człowieka ponieważ zawiera konstrukcje gramatyczne podobne do języka jakim posługuje się człowiek. Przykładem takich języków są C, C++, C# (jako kolejne rozwinięcia standardowego acz starego C), Pascal, Fortran, Cobol, SQL, HTML, Java, Asembler. No może z czytelnością tego ostatniego to przesada ale jest to także kod źródłowy. W przypadku AI2 nie piszemy kodu "z palca" ale tworzymy logikę działania aplikacji za pomocą przeciągania i odpowiedniego łączenia bloków. Każdy taki blok to zestaw instrukcji (czynności) jakie będzie wykonywała nasza aplikacja. Można powiedzieć, że jest to programowania wizualne, graficzne polegające na składaniu bloków pewnych działań w logiczną całość.

Kod wynikowy (binarny)

Kod ten powstaje po przetworzeniu przez kompilator lub interpreter. Przetwarzany jest kod źródłowy w kod binarny dostosowany do danej platformy sprzętowo - programowej. Inaczej mówiąc kod wynikowy to nasz program najczęściej przechowywany na nośniku stałym (np dysk). Taki kod wynikowy staje się programem wykonywalnym po wczytaniu do pamięci i przekazaniu doń sterowania. Zajmuje się tym system operacyjny. W

przypadku AI2 można generować aplikacje bezpośrednio dla systemu Android i Google Play (pliki.apk) lub pliki z "kodem źródłowym" (pliki .aia) do późniejszej modyfikacji czy rozbudowy tworzonej aplikacji.

Kod pośredni

Niektóre systemy tworzenia aplikacji generują kod pośredni. Jest to binarny plik wynikowy, który do uruchomienia wymaga specjalnego oprogramowania. W świecie PC przykładem jest .NET (czyt. dotnet) firmy Microsoft. Aby uruchomić na komputerze docelowym aplikację napisaną w .NET potrzebny jest specjalny program uruchomieniowy. Takowy można pobrać z witryny Microsoftu, zainstalować i dzięki niemu uruchamiać programy pisane w środowisku .NET. W świecie mobilnym podobny kod generujemy za pomocą np Appinventora i instalujemy na smartfonie (Android). Aby uruchomić taką aplikację wymagane jest oprogramowanie zwane *wirtualna maszyna Javy*. Jest ona zainstalowana fabrycznie na każdym systemie Android i pośredniczy pomiędzy konkretnym sprzętem (model telefonu, procesor, płyta główna itp) a naszą aplikacją. dzięki temu nasz kod jest niezwykle uniwersalny i możliwy do uruchomienia na dowolnym modelu urządzenia mobilnego z systemem Android. Dla zainteresowanych: [coś o maszynie wirtualnej Javy](#), [coś o programowaniu na Androda](#), [Java i Android](#).

Język wysokiego poziomu, niskiego, maszynowy

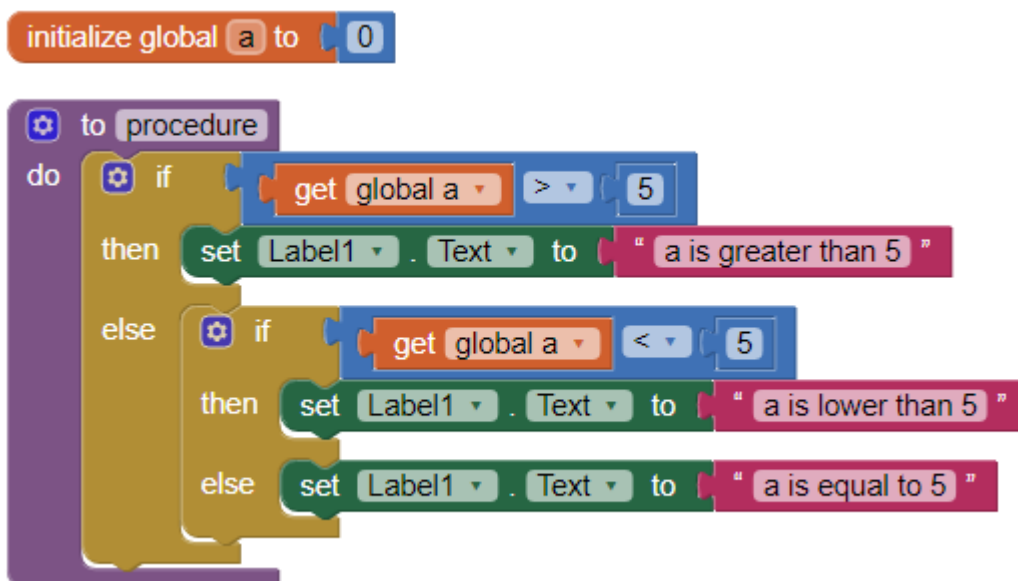
Język *wysokiego* poziomu to język zrozumiały w zapisie dla człowieka, najczęściej zapisany w postaci wyrażień słownych jak C++ lub w postaci graficznej jak język schematów blokowych UML.

Poniżej fragment kodu w Pascalu i odpowiedni w C++:

```
Program IsNumber;
var a:integer;
begin
  readln(a);
  IF (a>5) THEN
    writeln('a is greater than 5')
  ELSE
    IF (a<5) THEN
      writeln('a is lower than 5')
    ELSE
      writeln('a is equal to 5');
  readln;
end.
```

```
#include <iostream>
using namespace std;
int a;
int main()
{
  cin>>a;
  if (a>5)
    cout<<"a is greater than 5"<<endl;
  else
    if (a<5)
      cout<<"a is lower than 5"<<endl;
    else
      cout<<"a is equal to 5"<<endl;
  getchar();
  return 0;
}
```

Poniżej "kod" czyli zestaw bloków w AI2 odpowiadający powyższemu kodowi w Pascalu czy C++.



Język *niskiego* poziomu to język niezbyt czytelny dla człowieka np Asembler. Charakteryzuje się on mnemoniczną składnią czyli nie zawiera popularnych słów i wyrażień typu if, while, do, break itp ale kilkuliterowe symbole, które

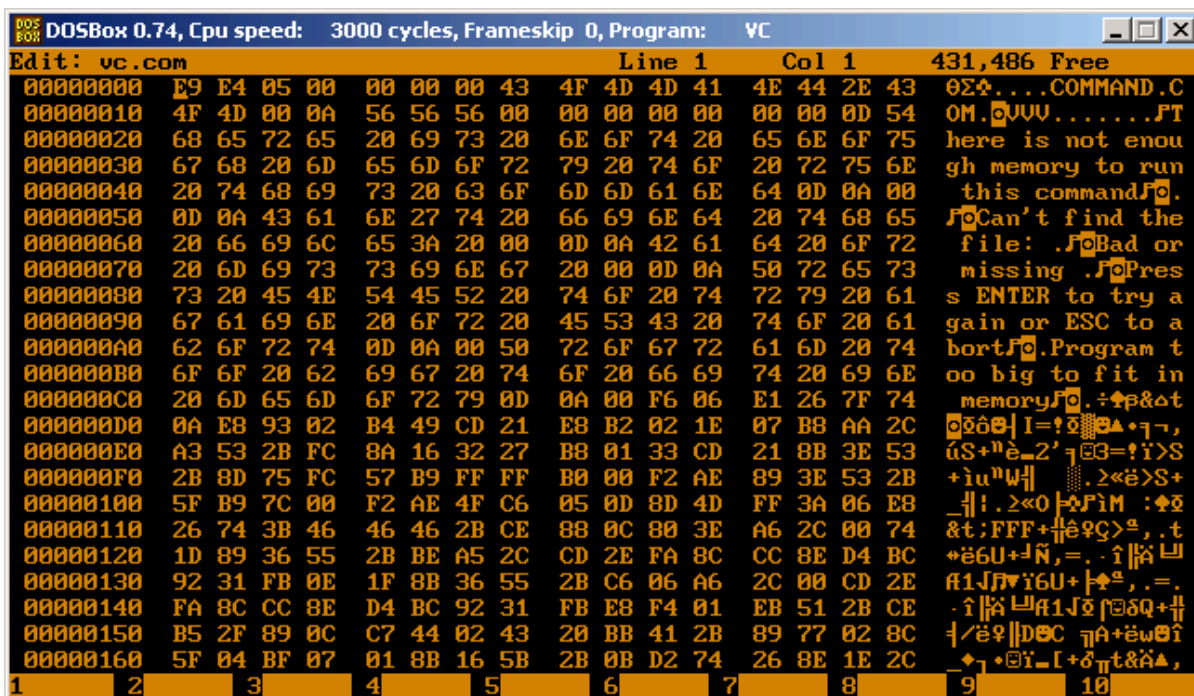
wraz z parametrami kodują konkretne zestawy operacji sprzętowych na danej maszynie. Poniżej fragment kodu w Asemblerze (wyświetla na ekranie literę A).

```
mov ah, 2
mov dl, 41h
int 21h
```

W Turbo Pascalu można było kodować też tak na jeszcze niższym poziomie, co było już prawie na poziomie kodu maszynowego:

```
begin
  inline($C4/$7E/<Dest/$8B/$4E/<Count/$8B/$46/<Data/$FC/$F3/$AB);
end.
```

Język maszynowy (często określany binarnym) to, jak wcześniej powiedziano, to język zrozumiały dla danej platformy, zapisany w kodach bezpośrednio związanych z maszyną na jakiej pracuje. Poniżej widok małego programu (dla systemu DOS/Windows platforma Intel/AMD) w postaci kodu maszynowego (szesnastkowo) i w postaci znaków ASCII (po prawej). Fajnym programem do przeglądania plików binarnych jest HexEdit.



Interpretacja, kompilacja

Aby przenieść kod źródłowy do kodu maszynowego potrzebne jest działanie zwane interpretacją lub kompilacją.

Interpretacja to proces tłumaczenia kodu źródłowego na kod maszynowy dla danej platformy "w locie". Nie jest generowany żaden plik wynikowy, wszystko dzieje się w pamięci operacyjnej. Interpretacja polega na wczytaniu kodu źródłowego przez interpreter i wykonywaniu go linia po linii czy polecenie po poleceniu. Niewątpliwą zaletą jest prostota i elastyczność tego rozwiązania - kod źródłowy można posłać na maszynę docelową i tam zostanie zinterpretowany. Rozbudowa i modyfikacje kodu również są bardzo elastyczne: wykonuje się je w centralnym miejscu (jak modyfikacja strony www na serwerze) i od razu są dostępne dla użytkowników. Świadczy o tym popularność np HTML i stron www. Wad jest kilka: za każdym razem powtarzany jest cały proces od nowa, przy każdym wywołaniu programu w pamięci rezyduje interpreter jak i program co zwiększa zapotrzebowanie zarówno na CPU jak i RAM. Przykładami języków interpretowalnych jest PHP, HTML, Python, Perl, skrypty powłoki Linuxa, Power Shell Windowsa. Przykładowo w przypadku HTML i PHP interpreterem jest przeglądarka, w przypadku skryptu powłoki Linuxa sama powłoka. Zazwyczaj program poddawany interpretacji działa wolniej niż program kompilowany.

Kompilacja przebiega inaczej: najpierw kod źródłowy jest za pomocą kompilatora zamieniany na kod maszynowy (może to być proces kilkietapowy jak w C) i w wyniku tego otrzymujemy program wykonywalny ale tylko na daną

platformę sprzętowo-programową. Jeśli nasz program ma pracować na innym systemie potrzebna będzie ponowna kompilacja i wygenerowanie kodu dostosowanego do platformy docelowej. Przykładem języków kompilowalnych jest C, C++, Pascal. W systemach Windows i dawniej DOS kompilator generuje pliki wykonywalne *.exe lub *.com (zazwyczaj z Asemblera) ewentualnie dodatkowe biblioteki *.ovr, *.ovl, *.dll ponieważ w tych systemach pliki wykonywalne muszą mieć rozszerzenie exe lub com. W systemach Linux wymagania takiego nie ma ale zazwyczaj pliki wykonywalne lub po kompilacji mają rozszerzenia *.o lub *.obj. Zaletą kompilacji jest to, że do uruchomienia programu nie trzeba nic po za systemem operacyjnym na jakim będzie on pracował. Wadą jest niewątpliwie problematyczność zmian programu. W przypadku wprowadzenia poprawek czy nowych wersji programu trzeba go dystrybuować do miejsc docelowych czyli tam gdzie został zainstalowany.

Zatem kompilacja czy interpretacja? Wszystko zależy od wymagań i zastosowań a jak widać obie metody mają ich wiele.

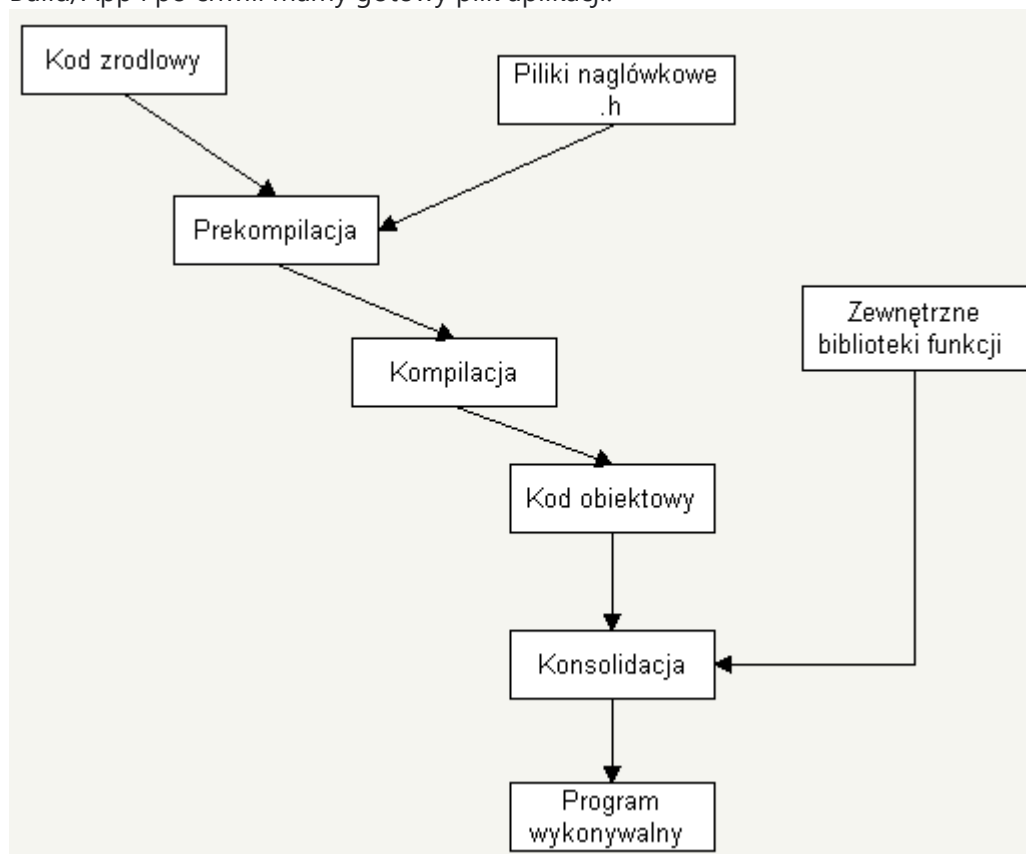
W przypadku AI2 generowany kod wynikowy (.apk) to kod tzw pośredni, który uruchamiany jest na smartfonie czy tablecie w wirtualnej maszynie Javy. Jest to specjalne oprogramowanie dostosowane do konkretnego modelu i sprzętu urządzenia mobilnego. Dzięki temu zabiegowi kod (aplikacja) jest jeden a może być uruchamiany na wielu różnych platformach sprzętowych różnych producentów z różnymi procesorami, płytami głównymi itd.

Kompilacja, łączenie, budowanie

Cały proces zamiany kodu źródłowego na wynikowy ma zazwyczaj kilka etapów:

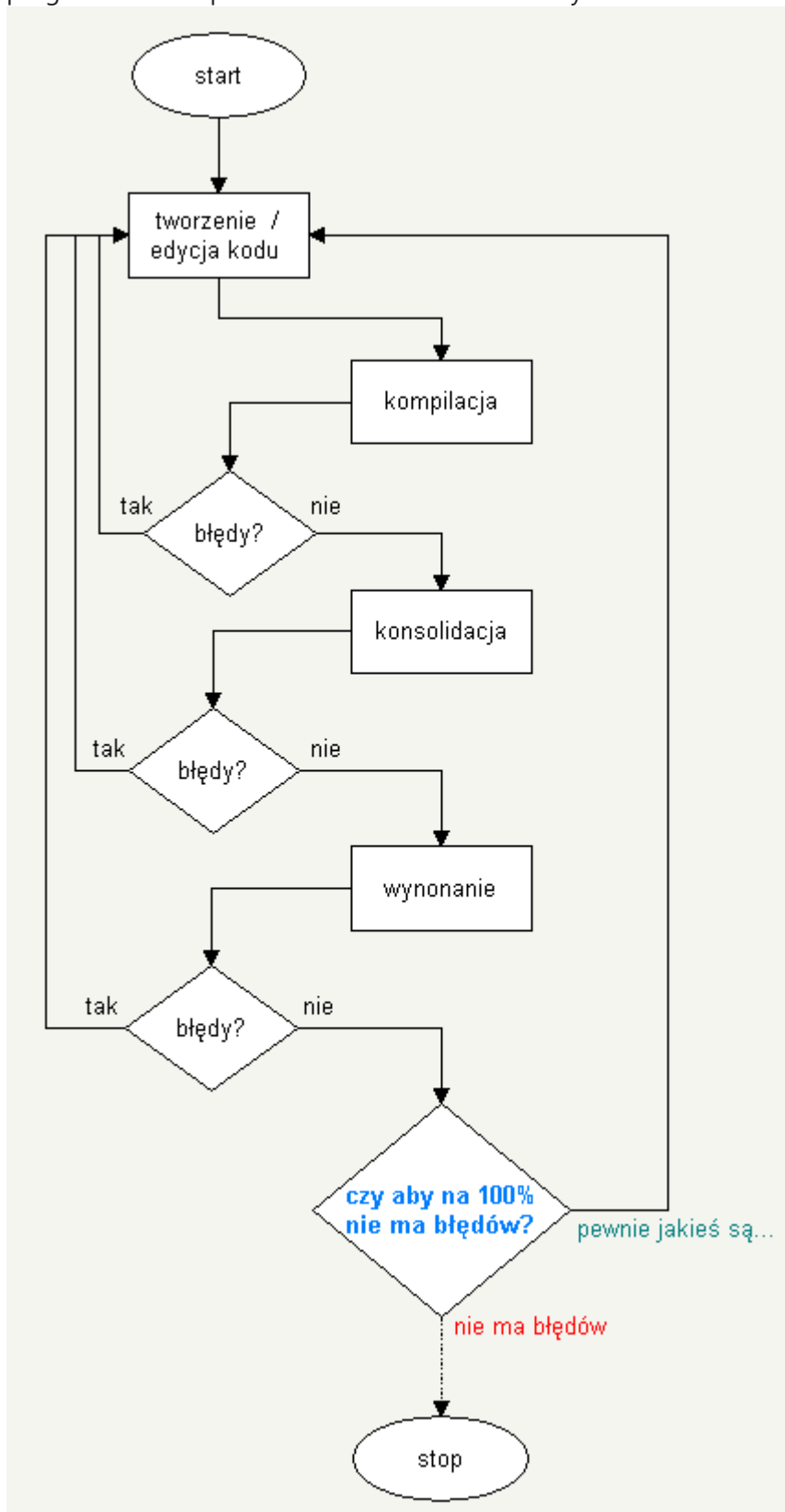
- *prekompilacja* czyli wykonanie dyrektyw preprocesora (#, język C/C++)
- *kompilacja i optymalizacja* kodu sprawdza poprawność składni kodu źródłowego, budowany jest pośredni plik specjalnego asemblera, wewnętrznego języka kompilatora
- *aseblacja* to budowanie kodu maszynowego (binarnego) ale nadal pośredniego do dalszych działań kompilatora
- *łączenie* (ang. linking) inaczej mówiąc konsolidacja pozwala połączyć użyte w programie funkcje (nie zdefiniowane w nim ale w bibliotekach zewnętrznych) z właściwymi bibliotekami funkcji.

W większości nowoczesnych systemów tworzenia aplikacji cały powyższy proces jest przed programistą ukryty a utworzenia aplikacji polega na naciśnięciu najczęściej jednego przycisku. Tak jest w AI2: wybieramy z menu opcje Build/App i po chwili mamy gotowy plik aplikacji.



Etapy tworzenia programu

Programy żyją własnym życiem, są ciągle modyfikowane, rozwijane, ulepszone. Ciągłe niestety w programach są znajdowane błędy co powoduje konieczność ich poprawiania niezależnie od języka w jakim są pisane. Cykl życia programu można przedstawić schematem blokowym:



Zastosowania

W związku z ogromną ilością dostępnych języków programowania jak i systemów tworzenia oprogramowania jest w czym wybierać :)

- aplikacje webowe: PHP, Java, JavaScript, HTML, XHTML
- bazy danych: SQL (często + PHP)
- programy "stacjonarne", uniwersalne: C, C++
- skomplikowane obliczenia inżynierskie: Fortran (ew. C/C++ ze specjalnymi bibliotekami)

- programy wizualne: Delphi, Visual Basic, Visual C
- opis strony tekstu: PostScript, HTML, PHP
- przetwarzanie tekstów: perl, python
- sztuczna inteligencja: List, Prolog
- obsługa grafiki: OpenGL, OpenCL
- specjalistyczne, dedykowane: języki makr Worda, formuł Excella, GoogleCalc itp.
- języki "sieciowe": MikroTik, Cisco, Juniper
- języki skryptowe: powłoki shell systemów takich jak Linux, Windows, VMWare

Oczywiście programy jak i systemy mogą się przeplatać, dobrym przykładem są aplikacje webowe korzystające z np: PHP, Ruby, Java, CGI po stronie serwera, JavaScript, HTML po stronie klienta.

Rzemiosło czy sztuka?

Jedni twierdzą, że programowanie to rzemiosło inni, że to sztuka. W sumie to połączenie sztuki i rzemiosła ponieważ najpierw trzeba program "wymyślić", potem zaprojektować, zakodować, przetestować, rozpowszechnić a następnie twórców czeka nieustający proces poprawek, modyfikacji i rozwoju programu. Każdy z tych etapów zawiera elementy sztuki i rzemiosła.

Sprzęt i oprogramowanie

Aktualnie istnieje równowaga logiczna pomiędzy sprzętem a oprogramowaniem. Funkcje budowane programowo mogą zostać zaimplementowane bezpośrednio w sprzęt (pozostaje kwestia opłacalności budowy takiego układu) natomiast te opracowane dla wykonania sprzętowego mogą być realizowane (emulowane) programowo (pozostaje kwestia wydajności ponieważ funkcje wykonywane sprzętowo budowane są zazwyczaj dla osiągnięcia maksymalnej wydajności).