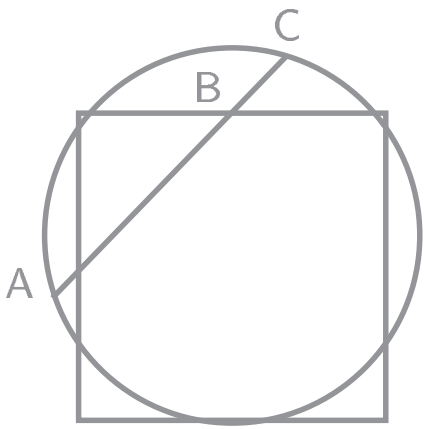
Inżynieria oprogramowania

Radosław Klimek

2015-23



http://home.agh.edu.pl/rklimek

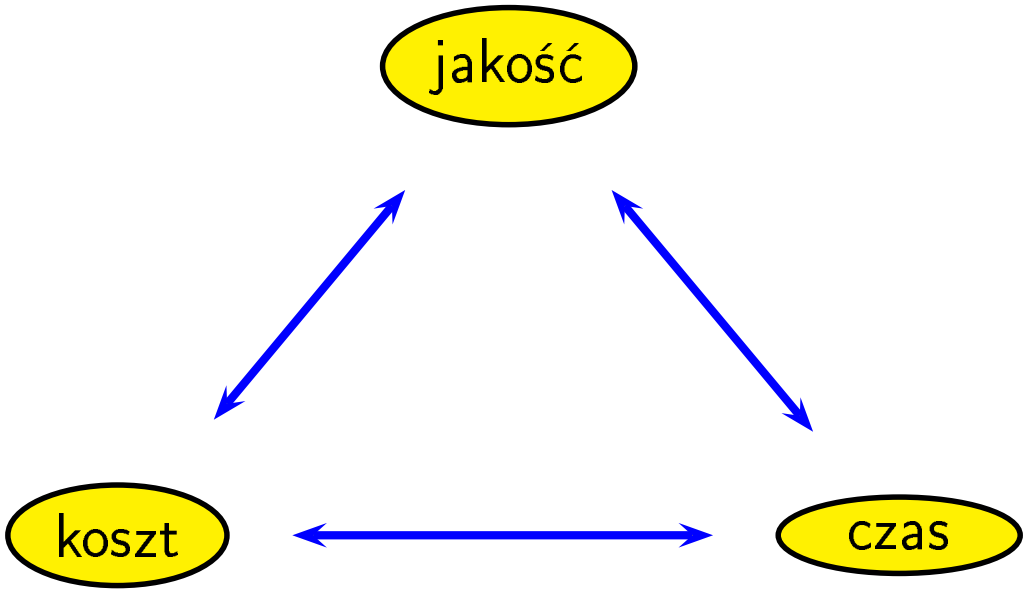
1 Szacowanie projektów informatycznych

Szacowanie projektów informatycznych

Quentin METSYS: *Bankier z żoną*

„Magiczny trójkąt" projektu informatycznego

Na parametry zarządzania i oceny każdego projektu składają się trzy wielkości, oddziaływujące na siebie wzajemnie.



L-OOP

Tymczasem częstym zjawiskiem, zaobserwowanym już w latach sześćdziesiątych jako tzw. kryzys oprogramowania, jest syndrom oznaczany angielskim akronimem **OOP**:

- **O** ver time,
- **O** ver budget,
- **P** oor quality.

Trudności szacowania

Główne trudności procesu szacowania oprogramowania:

- subiektywna natura szacowania – np. istnieje tendencja do niedoszacowywania skomplikowania systemów w przypadku małych projektów oraz przeszacowywania dla projektów dużych;
- polityczne implikacje – np. odmienne cele poszczególnych instytucji, działów czy grup ludzi.

Konieczność posiadania poważnych danych historycznych.

Trudności szacowania – tabela

Program	Projektow. o-m	%	Implement. o-m	%	Testowanie o-m	%	Razem o-m	SLOC
a	3,9	23%	5,3	32%	7,4	44%	16,7	6.050
b	2,7	12%	13,4	59%	6,5	26%	22,6	8.363
c	3,5	11%	26,8	83%	1,9	6%	32,2	13.334
d	0,8	21%	2,4	62%	0,7	18%	3,9	5.942
e	1,8	10%	7,7	44%	7,8	45%	17,3	3.315
f	19,0	28%	29,7	44%	19,0	28%	67,7	38.988
g	2,1	21%	7,4	74%	0,5	5%	10,1	38.614
h	1,3	7%	12,7	66%	5,3	27%	19,3	12.762
i	8,5	14%	22,7	38%	28,2	47%	59,5	26.500

SLOC – **liczba wierszy kodu źródłowego** (ang. *source number of line of code*)

Specyfika wytwarzania oprogramowania

Specyfikę procesu wytwarzania oprogramowania można opisać poprzez następujące fakty:

- dominacja procesu analizy i projektowania – procesy analizy i projektowania przeważają nad procesami produkcji, odwrotnie niż w innych działach – np. wytworzenie nowej wersji programu (bez procesu projektowania) może być mniej kosztowne;
- problem wizualizacji w trakcie początkowych faz – wytwarzone oprogramowanie jest widoczne zasadniczo dopiero w końcowych fazach produkcji, wcześniej mogą to być modele i reprezentacje pośrednie, nie zawsze oddające dziedzinową rzeczywistość;
- nie zużywanie się oprogramowania – nie występuje zjawisko starzenia się charakterystyczne dla innych branż, gdzie zupełnie naturalnym jest np. rozważanie czasu pracy bezawaryjnej;

Specyfika wytwarzania oprogramowania (cd.)

- duża złożoność oprogramowania – oprogramowanie cechuje się dużą złożonością (wewnętrzną), bardzo często nieuświadomioną przez odbiorców, np. kwestia współdzielenia, liczne problemy synchronizacji, propagacja wyjątków, itd.;
- duża dowolność struktury – w trakcie wytwarzania oprogramowania może istnieć problem pewnej dowolności zarówno stosowanych narzędzi programistycznych mających wpływ na produkt, a także samej wytwarzanej – czasem nieregularnej i trudnej do zrozumienia **ad hoc** – struktury oprogramowania;

Specyfika wytwarzania oprogramowania (cd.)

- charakter fizyczny oprogramowania – z jednej strony brak ograniczeń fizycznych np. w przekazaniu sterowania w odległe miejsce, co może rodzić swoje poważne konsekwencje, z drugiej strony fizyczne uszkodzenie w jednym miejscu oprogramowania może uniemożliwić działanie w całości;
- pozorna łatwość zmian – oprogramowanie jest produktem dość łatwo poddającym się „drobnym" i „niepozornym" zmianom i modyfikacjom, które mogą nieść swoje poważne konsekwencje.

Miejsca szacowania w projekcie informatycznym

W trakcie całego procesu wytwarzania oprogramowania możemy mieć do czynienia z szacowaniem oprogramowania w kilku miejscach procesu:

- planowanie strategiczne,
- studium wykonalności,
- specyfikacja systemu,
- ocena propozycji zmian,
- planowanie projektu.

Szacowanie (rozmiarów) oprogramowania można by odnieść – i warto to kiedyś zrobić – do poszczególnych faz cyklu życia oprogramowania.

Trudności szacowania – nieformalne spojrzenie

Zasada Parkinsona: Praca nad projektem ma tendencję do wypełnienia całego dostępnego czasu.

Zasada Brooksa: Skierowanie większej liczby osób do projektu opóźnionego powoduje jeszcze większe jego opóźnienie.

Cele szacowania

Podstawowe cele szacowania:

- ocena nakładu pracy;
- utworzenie odpowiedniego harmonogramu;
- podjęcie właściwych zobowiązań;
- inne.

Zbierane dane (przykładowo):

- nazwa projektu,
- liczba linii kodu,
- wysiłek w osobo-miesiącach,
- koszt w USD,
- liczba stron wytworzonej dokumentacji,
- liczba błędów przed przekazaniem systemu użytkownikowi,
- liczba defektów po przekazniu systemu użytkownikowi, np. w trakcie pierwszego roku eksploatacji,
- liczba osób w projekcie.

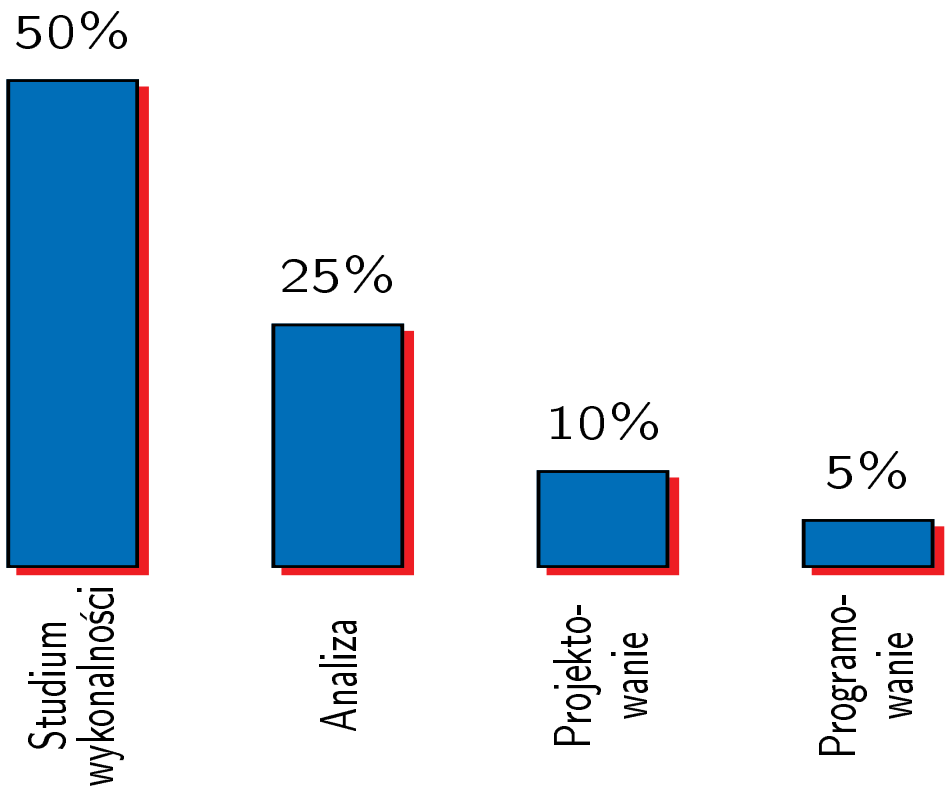
Cele szacowania – tabela

Projekt	LOC	Nak. (o-s)	USD (tys.)	Dok. (str.)	Bł.	Def.	Os.
Alfa	12.100	24	168	365	134	29	3
Beta	27.200	62	440	1224	321	86	5
Gamma	20,200	43	314	1050	256	64	6
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Na podstawie tych oraz innych pomiarów, metryk i zestawień można wyliczyć szereg interesujących wielkości dotyczących charakteru i jakości projektów powstających w firmie (np.: gęstość błędów, liczba spodziewanych błędów podczas testów, itd.).

Dokładność szacowania

E. Yourdon podaje dokładność szacowania w projekcie informatycznym w zależności od fazy realizacji projektu informatycznego.



E. Yourdon: **Współczesna analiza strukturalna**. Wydawnictwa Naukowo-Techniczne 1996.

Dokładność szacowania – uwagi

- Należałoby sie zastanowić czy dane z Yourdona są (na dziś) dokładne.
- Mała dokładność w początkowych fazach projektu wynika z wielu niewiadomych, które w trakcie postępu prac znikają.
- Warto jednak szacować – pozwala to na opracowywanie strategii projektu, działań operacyjnych, harmonogramów itd.
- Z drugiej strony Studium Wykonalności jest bardzo wstępną fazą projektu, w zasadzie jest jakby przed-projektem. Szacowanie w tym momencie może być punktem wyjścia do gry negocjacyjnej.
- Szacowanie już w trakcie programowania jest działaniem powykonawczym/kontrolnym.

Miary stosowane w szacowaniu

- Co to znaczy dokonać pomiaru?
  („Program A jest dwa razy bardziej złożony/trudniejszy do pielęgnacji niż program B”).
- Obszary mierzenia oprogramowania:
  - koszty,
  - pracochłonność,
  - jakość,
  - niezawodność,
  - złożoność (skomplikowanie/trudność),
  - złożoność obliczeniowa algorytmu;

Miary stosowane w szacowaniu (cd.)

- Miary złożoności:
  1. miary objętości:
     - linie kodu (LOC),
     - liczba zdań źródłowych,
     - liczba operacji i operandów,
     - liczba procedur,
     - średnia długość procedury,
     - liczba zmiennych;
  2. miary organizacji sterowania:
     - liczba cyklomatyczna (McCabe),
     - miara pętli (Woodward i Hennel),
     - minimalna liczba przecięć (Chen),
     - średni poziom zanurzenia (Dunsmore);
  3. miara organizacji danych:
     - miara powiązań danych.

Klasyczna dziedzina szacowania

Podstawowa – wynikająca z przedstawionych już parametrów projektu – dziedzina szacowania obejmuje:

- **nakład pracy** (**pracochłonność**) (ang. *effort*) – osobotygodnie lub osobomiesiące;
- **czas trwania** (ang. *duration*) projektu – tygodnie lub miesiące;
- **obciążenie ludzi** (ang. *manpower loading*) – liczba wymaganych pracowników przydzielonych do projektu w funkcji czasu.

Koszt systemu jest zwykle szacowany w funkcji tych właśnie wielkości.

Klasyczna dziedzina szacowania

Podstawowa – wynikająca z przedstawionych już parametrów projektu – dziedzina szacowania obejmuje:

- **nakład pracy** (**pracochłonność**) (ang. *effort*) – osobotygodnie lub osobomiesiące;
- **czas trwania** (ang. *duration*) projektu – tygodnie lub miesiące;
- **obciążenie ludzi** (ang. *manpower loading*) – liczba wymaganych pracowników przydzielonych do projektu w funkcji czasu.

Koszt systemu jest zwykle szacowany w funkcji tych właśnie wielkości.

Oszacowania robione są dla całego projektu, dla pojedynczej fazy, zadania, a nawet czynności. Są one nieodzowne przed startem projektu (m.in. w celu analizy kosztów i zysku oraz negocjacji kontraktu), podczas realizacji projektu (dla potrzeb planowania, monitorowania i kontroli), muszą też być aktualizowane w trakcie trwania projektu.

Trudności szacowania

Inne trudności i błędy szacowania w projekcie wynikają z faktów:

- natura oprogramowania i jego specyfika – złożoność, „niematerialność” oraz jednostkowość (unikatowość) kolejnych zastosowań;
- inne problemy związane z samym szacowaniem – brak danych historycznych, porównawczych, trudność w doborze miar pracy, złożoność procesu oceny.

Do tego dochodzą problemy procesu wytwarzania: zmiany technologii, brak możliwości przeniesienia doświadczeń projektowych, a niekiedy także rozdźwięk pomiędzy grupami wytwarzającymi poszczególne komponenty składowe systemu.

Podstawowe strategie szacowania

- **szacowanie przez analogię** – porównanie z charakterystykami wcześniej realizowanych podobnych projektów;
- **metody delfickie** (ang. *Delphi techniques*) – oceny eksperckie lub dedykowane oprogramowanie;
- **cena do uzyskania** (ang. *price to win*) – „ile też zleceniodawca zgodzi się zaakceptować”;
- **dostępne środki** (ang. *design to cost*) – ocena na podstawie założenia o środkach i zasobach;

Podstawowe strategie szacowania (cd.)

- **mikrotechnika**, szacowanie „od dołu” (ang. *bottom-up*) – począwszy od czynności i zadań elementarnych, następuje składanie oszacowań w większe całości;
- **makrotechnika** podejście „od góry” (ang. *top-down*) – szacowanie na najwyższym poziomie ogólności, a następnie dekomponowanie na zadania składowe; po oszacowaniu zadań najniższego poziomu zazwyczaj korekta całości prac;
- metoda **analizy punktów funkcyjnych** FPA;
- metoda **COCOMO**;
- szereg innych metod.

Metody delfickie

Podstawowe informacje o metodach delfickich:

- eksperci o różnych preferancjach i nastawieniach (m.in.: optymiści, pesymiści, nastawieni na wygraną, nastwieni na porażkę, umotywowani politycznie);
- **metoda delficka standardowa** (ang. *Delphi technique*) została zaproponowana w Rand Corporation (1948), jako sposób na znajdowanie **consensusu** w gronie eksperckim, dokonującym planowania i oceny przyszłych kosztów przedsięwzięcia;

Metody delfickie

Podstawowe informacje o metodach delfickich:

- eksperci o różnych preferancjach i nastawieniach (m.in.: optymiści, pesymiści, nastawieni na wygraną, nastwieni na porażkę, umotywowani politycznie);
- **metoda delficka standardowa** (ang. *Delphi technique*) została zaproponowana w Rand Corporation (1948), jako sposób na znajdowanie **consensusu** w gronie eksperckim, dokonującym planowania i oceny przyszłych kosztów przedsięwzięcia;

Metoda delficka próbuje „pogodzić" różne i odmienne nastawienia ekspertów. Technika delficka została pomyślana była jako pomocna w prognozowaniu trendów rynkowych. Metoda delficka odnosi się do różnych przedsięwzięć, a może być także zastosowana w inżynierii oprogramowania.

Metody delfickie (cd.)

- metoda delficka, która może być z powodzeniem stosowana w zagadnieniach inżynierii oprogramowania, została zmodyfikowana – **rozszerzona metoda delficka** (ang. *Wideband Delphi technique*) – tak aby zapewnić lepszą komunikację uczestników procesu estymacji w dochodzeniu do porozumienia.

Delfy – miasto w starożytnej Grecji, miejsce kultu Apollina z wyrocznią (zręcznie sterującą sprawami kolonizacji i polityki), uchodzącą w starożytności za najsłynniejszą.

Ogólne zasady metod delfickich

Ogólne zasady:

- członkom zespołu oceniającego dostarcza się istotnych informacji odnośnie dziedziny estymacji (cele, założenia, zakres, itd.);
- każdy z uczestników indywidualnie, anonimowo dokonuje oceny;
- wyniki są zbierane, opracowywane i po prezentacji proces estymacji może być powtarzany aż do osiągnięcia porozumienia.

Ogólne cechy metod delfickich

Główne cechy metod delfickich:

- anonimowość,
- analiza statystyczna,
- logiczna argumentacja.

Pytia – wieszczka Apollina w Delfach, w miejscu niedostępnym dla wiernych udzielała odpowiedzi na pytania stawiane za pośrednictwem kapłana; przed udzieleniem odpowiedzi Pytia musiała pościć i odbyć rytualną kąpiel w Źródle Kastalskim; jej wypowiedzi cechowała wieloznaczność, stąd „odpowiedź pytyjska".

Procedura standardowej metody delfickiej

Standardowa metoda delficka obejmuje następujące pięć kroków/reguł:

1	moderator przedstawia każdemu ekspertowi specyfikację i formularz gdzie zapisywane są oceny;
2	eksperci anonimowo wypełniają formularz (mogą kierować zapytania do moderatora, ale nie mogą dyskutować ze sobą);
3	moderator opracowuje wyniki na specjalnym formularzu, który jednocześnie stanowi zaproszenie do rundy kolejnej;
4	eksperci ponownie, anonimowo wypełniają formularz, a sam proces estymacji jest powtarzany tak długo jak to jest niezbędne;
5	podczas całego procesu estymacji nie są prowadzone żadne dyskusje grupowe i pomiędzy ekspertami.

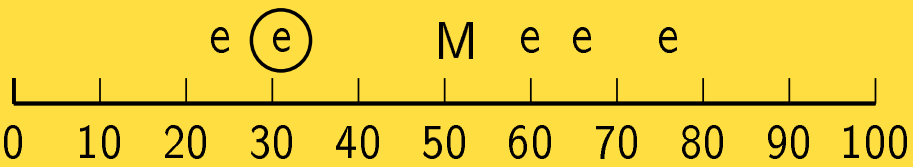
Formularz eksperta – przykład

Formularz metody delfickiej

Projekt: **System obsługi dziekanatu**

Ekspert: **Jan Kowalski** Data: **24.05.2002**

Oszacowania z rundy nr: 1



Prosze podać estymację w kolejnej rundzie

Prosze podać uzasadnienie kolejnej estymacji:

e – ocena ekspercka,

(e) – twoja ocena ekspercka,

M – średnia ocen eksperckich.

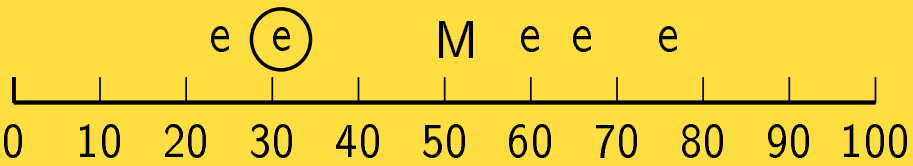
Formularz eksperta – przykład

Formularz metody delfickiej

Projekt: **System obsługi dziekanatu**

Ekspert: **Jan Kowalski** Data: **24.05.2002**

Oszacowania z rundy nr: 1



Prosze podać estymację w kolejnej rundzie

Prosze podać uzasadnienie kolejnej estymacji:

e – ocena ekspercka,

(e) – twoja ocena ekspercka,

M – średnia ocen eksperckich.

Formularz (ma znaczenie dydaktyczne i) jest nie do końca precyzyjny: nie wiadomo dokładnie czego dotyczy (np. rozmiar oprogramowania, czas projektu), jaka jest jednostka estymowanej wielkości, itd. – formularz mógłby/powinien uwzględniać te informacje.

Procedura rozszerzonej metody delfickiej

1	moderator przedstawia każdemu ekspertowi specyfikację i formularz estymacyjny;
2	moderator inicjuje spotkanie ekspertów, na którym eksperci dyskutują z moderatorem oraz ze sobą zagadnienia estymacji (cele projektu, założenia, problemy oceny, itd.);
3	eksperci anonimowo wpełniają formularze estymacyjne;
4	moderator opracowuje wyniki na specjalnym formularzu (podobny do poprzedniego, jednak bez pola z pisemnym uzasadnieniem);
5	moderator inicjuje spotkanie ekspertów, na którym eksperci dyskutują wyniki, w szczególności omawiane są oceny znacznie różniące się od siebie;
6	eksperci ponownie, anonimowo wypełniają formularze, a kroki od 4 do 6 są powtarzane tak długo jak to jest niezbędne.

Metoda analizy punktów funkcyjnych

Metoda **analizy punktów funkcyjnych** FPA (ang. *function point analysis*) została zaproponowana przez Allana J. Albrechta (na podstawie prac zespołu w firmie IBM) w roku 1979:

- celem metody jest oszacowanie nakładów i kosztów projektu na podstawie funkcji użytkowych, które system ma realizować, na podstawie wielkości i złożoności danych;
- metoda nie jest skierowna do kodu źródłowego ale do szczegółowej specyfikacji systemu;
- po obliczeniu funkcjonalności informacje-dane tak uzyskane są mnożone przez zadane z góry wagi i sumowane; rezultatem jest liczba niewyregulowanych („surowych") punktów funkcyjnych;

Metoda analizy punktów funkcyjnych (cd.)

- punkty funkcyjne mogą być następnie modyfikowane zależnie od dodatkowych czynników i złożoności oprogramowania;
- istnieją przeliczniki punktów funkcyjnych na liczbę linii kodu, co może być podstawą np. dla metody COCOMO.

Metoda ta może być stosowana dopiero wtedy, gdy funkcje te i funkcjonalności są z grubsza znane, tj. gdy znany jest model logiczny systemu (np. DFD).

Kategorie punktów funkcyjnych

W metodzie analizy punktów funkcyjnych FPA, ze względu na klasy funkcji operujących na danych, wyróżnia się następujące pięć rodzajów punktów funkcyjnych:

- *I* – **zewnętrzne wejściowe** (ang. *external inputs*) – element danych użytkownika lub sterowania przekazywany do aplikacji, powodujący uaktualnienie wewnętrznych plików systemu (przykładowo formatki ekranowe);
- *O* – **zewnętrzne wyjściowe** (ang. *external outputs*) – element danych użytkownika lub sterowania wytwarzany przez aplikację i przekazywany/przesyłany użytkownikowi (raporty, formatki wyjściowe, ekrany o ile nie są to transakcje zapytania – por. poniżej);

(cdn)

Kategorie punktów funkcyjnych (cd.)

E – **transakcje zapytania** (**usługi**) (ang. *enquiry transactions*) – kombinacja wejścia/wyjścia, transakcje zainicjowane przez użytkownika, które dostarczają wymaganych informacji ale nie powodują uaktualnienia wewnętrznych plików systemu (przykładowo wejście powodujące natychmiastową odpowiedź);

L – **wewnętrzne dane logiczne** (**logiczny plik główny**, czasem **zbiór wewnętrzny**) (ang. *logical master files*) – logiczne, niekoniecznie w postaci fizycznych plików, pamięci danych (gupy danych), na które oddziaływuje użytkownik, ogólnie dane utrzymywane w systemie lub też pliki robocze (np. pliki indeksowe);

(cdn)

Kategorie punktów funkcyjnych (cd.)

F – **dane interfejsowe** (**plik interfejsowy**, czasem **zbiór zewnętrzny**) (ang. *interfaces*) – plik lub dane wejścia/wyjścia używane przez inną aplikację (transmisja danych w obu kierunkach), tworzące powiązania systemu z otoczeniem, ogólnie sprzęgi z otoczeniem (np.: pobranie szczegółowych informacji z aplikacji księgowej do modułu wystawiania/rozliczania faktur, wytworzenie pliku z rachunkami debetowymi, ogólnie pliki dzielone przez różne aplikacje).

Schemat postępowania w metodzie FPA

Schemat postępowania w metodzie analizy punktów funkcyjnych może obejmować następujące kroki:

- wyróżnienie modułów (podsystemów) projektu i określenie głównych typów danych każdego modułu;
- ocena wyróżnionych typów danych i przypisanie im wag w zależności od poziomu złożoności;
- obliczenie początkowej (surowej) liczby punktów funkcyjnych;
- analiza parametrów korygujących oraz określenie stopnia ich wpływu;
- obliczenie ostatecznej (właściwej) liczby punktów funkcyjnych;
- określenie nakładu pracy (np. w osobomiesiącach), na podstawie znanych odwzorowań nakładu pracy, wymaganego do zrealizowania danej liczby punktów funkcyjnych.

Złożoność systemu a nieskorygowane *FP*

Niewyregulowane (nieskorygowane) punkty funkcyjne *UFP* (ang. *unadjusted function points*) wylicza się z uwzględnieniem wag złożoności analizowanego systemu.

(i)		Elementy przetwarzania	Poziom złożoności (j)		
			prosty	średni	złożony
1	I	Wejścia	3	4	6
2	O	Wyjścia	4	5	7
3	E	Zapytania	3	4	6
4	L	Dane wewnętrzne	7	10	15
5	F	Dane interfejsowe	5	7	10

Wagi dla poszczególnych elementów systemu zostały zaproponowane na podstawie wielu praktyk wykonania systemów informatycznych.

Złożoność systemu a nieskorygowane *FP* (cd.)

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{i,j} \cdot P_{i,j}$$

gdzie: w – wartość współczynnika wagi,
P – liczba elementów w projekcie (systemie),
i – element przetwarzania,
j – poziom złożoności.

Przykład formularza dla *UFP*

Rodzaj parametru	Punkty		Wagi Pr	Śr	Zł		Razem *UFP*
Wejścia		x	3	4	6	=	
Wyjścia		x	4	5	7	=	
Zapytania		x	3	4	6	=	
Pliki danych		x	7	10	15	=	
Interfejsy		x	5	7	10	=	
Razem (nieskorygowane)							

Złożoność techniczna realizacji systemu (1)

- Rozważmy obecnie uwarunkowania realizacji technicznej systemu, które to uwarunkowania będą służyły do korelacji „surowych" punktów funkcyjnych.

- Same indeksy, ich kolejność, przy *DI* są umowne.
- Istnieje wiele opisywanych uwarunkowań realizacji technicznej systemu – przedstawione tutaj są klasyczne i najczęściej spotykane.
- Istnieje także inny problem wynikający z faktu, że gdy definiowano czynniki technicznej realizacji systemów stan wiedzy i doświadczenia informatycznego był inny niż obecnie.

Złożoność techniczna realizacji systemu (2)

Czternaście (podstawowych) **czynników korygujących** $DI_{1...14}$, uwzględnianych przy ocenie złożoności technicznej wykonania systemu:

DI_1 **komunikacje danych** (ang. *data communications*) – czy dane w systemie są przesyłane poprzez urządzenia komunikacyjne? (0 – przetwarzanie wsadowe, 5 – przetwarzanie całkowicie on-line);

DI_2 **rozproszenie przetwarzania/funkcjonalności** (ang. *distributed function/functionality*) – funkcjonalności systemu lub jego dane są rozproszone (0 – całkowicie scentralizowane przetwarzanie w pojedynczej aplikacji, 5 – całkowite rozproszenie poprzez sieć);

DI_3 **szybkość działania** (ang. *performance*) – czy parametry szybkościowe działania muszą być uwzględnione w trakcie projektu, implementacji i pielęgnacji systemu? (0 – standardowe przetwarzanie, 5 – szybkość jest krytyczna);

Złożoność techniczna realizacji systemu (3)

DI_4 **mnogość wykorzystywanych konfiguracji** (ang. *heavily used configurations*) – czy aplikacja będzie wykorzystywana (uruchamiana) w środowisku o różnych i zmiennych wymaganiach (np. dzielony procesor)? (0 – bez ograniczeń/wymagań środowiskowych, 5 – silne ograniczenia/wymagania środowiskowe i systemu operacyjnego);

DI_5 **czestotliwość/liczba transakcji** (ang. *transaction rates*) – czy system będzie obciążony dużą liczbą transakcji, koniecznych do uwzględnienia w trakcie projektu, implementacji oraz pielęgnacji systemu? (0 – przeciętna częstotliwość transakcji, 5 – krytyczna częstotliwość transakcji);

DI_6 **wprowadzanie danych w trybie on-line** (ang. *on-line data entry*) – czy dane w systemie są wprowadzane bezpośrednio (w trybie on-line)? (0 – brak interakcyjnego wprowadzania danych, 5 – interaktywne wprowadzanie danych, więcej niż 30% danych wprowadzanych on-line);

Złożoność techniczna realizacji systemu (4)

DI_7 **wydajność użytkownika końcowego** (ang. *end-user efficiency*) – czy użytkownik końcowy/docelowy wymaga prostego, graficznego interfejsu oraz łatwości obsługi? (0 – brak szczególnych wymagań, 5 – duże wymagania odnośnie prostoty obsługi);

DI_8 **dane w trybie on-line** (ang. *on-line update*) – czy bazy danych i logiczne pliki główne są modyfikowane on-line? (0 – brak przetwarzania on-line, 5 – duża liczba danych przetwarzanych on-line);

DI_9 **złożoność przetwarzania** (ang. *complex processing*) – czy wewnętrzna logika przetwarzania jest skomplikowana (także, czy istnieje np. złożona obsługa wyjątków)? (0 – proste przetwarzanie, 5 – skomplikowana logika przetwarzania, bezpieczeństwo, uwarunkowania obsługi wyjątków, złożone formaty wejścia/wyjścia);

Złożoność techniczna realizacji systemu (5)

DI_{10} **ponowne wykorzystanie kodu** (ang. code reusability) – czy tworzony kod będzie ponownie wykorzystywany? (0 – 0%–10%, 1 – 10%–20%, 2 – 20%–30%, 3 – 30%–40%, 4 – 40%–50%, 5 – 50%-100%);

DI_{11} **łatwość instalacji** (ang. *conversion/installation ease*) – czy uwarunkowania instalacyjne są uwzględnione już na etapnie projektowania i implementacji systemu, a także przetestowane w trakcie testów ogólnych? (0 – brak szczególnych wymagań związanych z instalacją, 5 – szczególne wymagania instalacyjne, także wymagane specjalne narzędzia do instalacji);

Złożoność techniczna realizacji systemu (6)

DI_{12} **łatwość obsługi** (ang. *operational ease*) – czy uwarunkowania obsługi, np. uruchamianie, kopie zapasowe oraz inne czynności operatorskie, są uwzględnione w trakcie projektowania i implementacji systemu, a także przetestowane podczas testów ogólnych? (0 – brak szczególnych wymagań związanych z obsługą, 5 – szereg trudnych do zaplanowania czynności operatorskich);

DI_{13} **różnorodność i rozproszenie platform** (ang. *multiple site installation*) – czy system jest projektowany, implementowany i testowany z przeznaczeniem dla wielu różnych miejsc i stanowisk? (0 – pojedyncze stanowisko, 5 – wiele różnych stanowisk);

Złożoność techniczna realizacji systemu (7)

DI_{14} **zmienność wymagań** (ang. *facilitate change*) – czy aplikacja jest projektowana, implementowana i przeznaczona do pielegnacji z możliwością wprowadzania zmian odnośnie podstawowych/biznesowych parametrów systemu w trakcie eksploatacji (możliwość zmian wbudowana w kod systemu)? (0 – brak wymagań odnośnie możliwości zmian, 5 – odpowiednie, kluczowe dane wbudowane zarządzane przez użytkownika).

Obliczanie końcowej liczby *FP*

W oparciu o wszystkie (przedstawione) czynniki korygujące obliczamy kompleksowy **współczynnik złożoności technicznej** *CM*:

$$CM = 0,65 + 0,01 \cdot \sum_{i=1}^{14} DI_i$$

Dla czternastu czynników wpływu (każdy oceniany 0–5 punktów), współczynnik *CM* może więc przyjmować wartości od $0,65$ do $1,35$.

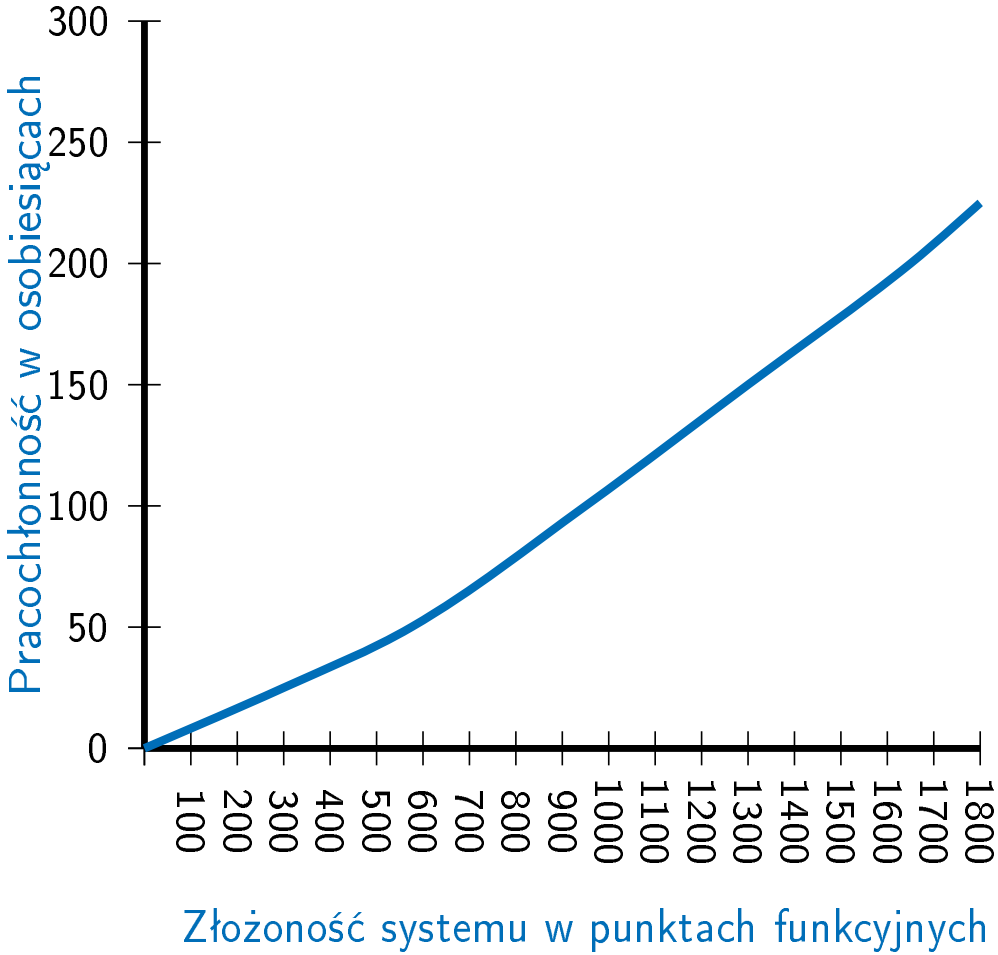
Współczynnik *CM* służy do skorygowania „surowej" liczby punktów funkcyjnych i obliczenia (ostatecznej) liczby punktów funkcyjnych *FP*:

$$FP = UFP \cdot CM$$

Wzór ten jest podstawowy dla metody analizy FPA.

Współczynnik *CM* jest oznaczany równoważnie także jako *TCF* (ang. *Technical Complexity Factor*) lub *PCA* (ang. *Processing Complexity Adjustment*) lub też *TCA* (ang. *Technical Complexity Adjustment*).

Obliczanie pracochłonności



Krzywa jest aproksymacją wielu danych pochodzących z różnych projektów.
Krzywa powinna się lekko podnosić do góry.

Metoda FPA a złożoność algorytmiczna

Typ aplikacji	Punkty funkcyjne	Punkty charakterystyczne
Przetwarzanie wsadowe	1	0,80
Przetwarzanie interaktywne	1	1,00
Systemy telekomunikacyjne	1	1,20
Systemy kontroli	1	1,28
Wbudowane systemy czasu rzeczywistego	1	1,35
Automatyka przemysłowa	1	1,50

Punkty funkcyjne nie oddawały szczególnie złożoności algorytmicznej systemu. Czasem próbuje się odnieść punkty funkcyjne do tzw. punktów charakterystycznych uwzględniających złożoność algorytmiczną poszczególnych klas systemów.

Zalety metody FPA

- Możliwość porównania różnych systemów, niezależnie od klas zastosowań;
- Możliwość porównania wydajności różnych zespołów wykonawczych;
- Możliwość przeliczenia innych wielkości na punkt funkcyjny, np.:
  - liczba wymaganych testów oprogramowania,
  - koszt wykonania jednego punktu funkcyjnego,
  - koszt pielęgnacji (utrzymania) jednego punktu funkcyjnego,
  - koszt eksploatacji jednego punktu funkcyjnego,
  - wydajność programisty,
  - inne;

Zalety metody FPA (cd.)

- Poszczególnie wielkości mogą mieć charakter uniwersalny lub być charakterystyczne dla danej firmy (czy zespołu wykonawców). Znacznie ważniejsze są jednak obliczenia własne wykonane dla konkretnych warunków, nawet jeśli nie jest to liczba zbyt wielu projektów – takie wyniki i porównania są zazwyczaj dokładniejsze.

Zawsze warto zaufać wynikom historycznym własnym, jako powstałym w konkretnych warunkach – dotyczy to także samego wyliczeniu punktów funkcyjnych.

Wyniki uniwersalne są publikowane m.in. przez firmę **Software Productivity Research**, por. `http://www.spr.org`.

Efektywność *LOC/FP*(języki programowania)

Język programowania	*LOC/FP*
Asembler	320
C	128
Cobol	106
FORTRAN	106
Pascal	90
C++	64
Ada95	53
Visual Basic	32
Smalltalk	22
SQL	16
Arkusze kalkulacyjne	6

Efektywność *LOC/FP* (języki baz danych)

Język baz danych	Poziom	*LOC/FP*
Access	8,50	38
ANSI SQL	25,00	13
CLARION	5,50	58
CA Clipper	17,00	19
dBase III	8,00	40
dBase IV	9,00	36
DELPHI	11,00	29
FOXPRO 2.5	9,50	34
INFORMIX	8,00	40
MAGIK	15,00	21
ORACLE	8,00	40
Oracle Developer/2000	14,00	23
PROGRESS v. 4	9,00	36
SYBASE	8,00	40

Wg. *Software Productivity Research*. Organizacja SPR również publikuje dane odnośnie linii kodu w stosunku do punktów funkcyjnych, wiele różnych danych.

Punkty funkcyjne a wydajność zespołu

Poziom języka wg SPR	Produktywność FPs/osobomiesiąc
1–3	5–10
4–8	10–20
9–15	16–23
16–23	15–30
24–55	30–50
>55	40–100

Metoda FPA – podsumowanie

- Metoda wymaga (ogólnej) znajomości systemu, konieczności sporządzenia specyfikacji logicznej;
- Subiektywność doboru parametrów, pracochłonność;
- Metoda dość popularna i chętnie wykorzystywana, może dostarczać wiarygodnych oszacowań;
- Stosowana jako metryka rozmiaru zamiast LOC (w innych metodach szacowania);
- Niezależność od języka programowania oraz możliwość obliczenia we wczesnej fazie projektu;

Metoda FPA – podsumowanie (cd.)

- Metoda jest rozwijana i doskonalona przez organizację IFPUG (ang. *International Function Point User Group*), por. także `http://www.ifpug.org`;
- Modyfikacje metody FPA:
  - **Feature points** – C. Jones 1991;
  - **Object points** – R. Kauffman, R. Kumar 1993;
  - **3D function points** – S. A. Whitmire 1995.

Punkty funkcyjne – przykłady aplikacji

Przykłady różnych aplikacji wyrażone w punktach funkcyjnych (FP).

FP	przykład aplikacji
1	ok. 125 instrukcji w języku C; ok. 50 instrukcji w języku Ada 95
10	typowy mały program tworzony samodzielnie przez miesiąc
100	większość typowych aplikacji; aplikacja tworzona samodzielnie przez ok. pół roku
1 000	komercyjne aplikacje dla MS Windows; małe aplikacje typu klient-serwer (10 osób, ok. 1 roku)
10 000	duże systemy (100 osoób, ok. 1,5 roku)
100 000	MS Windows 95, MVS, systemy militarne

Ryzyko projektu a punkty funkcyjne

Projekt	Złoż. (FP)	Zakończenie projektu (%) przed	w term.	po	Nie-wyk.
Małe programy	1	14,68%	83,15%	1,92%	0,25%
Aplikacje użytkow-nika	10	11,08%	81,25%	5,67%	2,99%
Duże aplikacje	100	6,06%	74,77%	11,83%	7,34%
Programy komer-cyjne	1 000	1,24%	60,76%	17,67%	20,33%
Systemy informa-tyczne	10 000	0,14%	28,03%	23,83%	48,00%
Duże systemy informat.	100 000	0,00%	13,67%	21,33%	65,00%

Metoda COCOMO

Model COCOMO (ang. *COnstructive COst MOdel*) został zaproponowany w 1981 r. przez Barry'ego Boehma:

- umożliwia szacowanie pracochłonności oraz czasu trwania projektów informatycznych lub ich faz;
- jest modelem parametrycznym – wzory w celu szacowania wielkości nakładów;
- istnieje wiele odmian modeli, natomiast model podstawowy budowany jest wokół równania:

$$E = a \cdot (size)^b$$

gdzie E to oszacowanie (np. nakładu pracy), *size* jest rozmiarem/wielkością stanowiącą punkt wyjścia, a parametry a oraz b są parametrami uzależnionymi od rodzaju projektu;

Metoda COCOMO (cd.)

- w poszczególnych modelach rozróżnia się rodzaje projektów, zależnie np. od dziedziny aplikacyjnej, wiedzy, doświadczenia oraz umiejętności zespołu realizacyjnego;
- w modelu COCOMO korzysta się z pojęcia osobomiesiąca, przyjmując że jeden osobomiesiąc jest równy 152 roboczym godzinom oraz 19 roboczym dniom.

Pierwszy model COCOMO opracowany został na podstawie danych o stosunkowo niedużej liczbie projektów (63).

Klasy analizowanych systemów

W modelu COCOMO wyróżnia się trzy podstawowe klasy budowanych i szacowanych systemów. Uzależnione są one (w oryginale ang. *mode*) od technicznej natury systemu oraz otoczenia projektowego:

1. **projekt organiczny** (ang. *organic mode*) – autonomiczne systemy przetwarzania danych, systemy względnie małe i łatwe, tworzone przez raczej małe zespoły;
2. **projekt wbudowany** (ang. *embedded mode*) – systemy czasu rzeczywistego lub systemy podlegające dodatkowym (silnym) ograniczeniom, zazwyczaj komplikujące rozwój systemu, zmiany w systemie są kosztowne, ogólnie systemy trudne;

Klasy analizowanych systemów (cd.)

3. **projekt częściowo-wydzielony** (ang. *semi-detached mode*) – systemy sytuujące się pomiędzy systemami organicznymi a wbudowanymi, tj. posiadające charakterystykę którą można zaliczyć w pewnych częściach pomiędzy obydwoma wymienionymi systemami.

Głównym **założeniami** podstawowego, historycznego modelu COCOMO są zależności wykładnicze oraz kaskadowy cykl wytwarzania oprogramowania.

Model COCOMO – nakłady pracy

W podstawowym modelu COCOMO szacujemy nakłady pracy na podstawie wzoru, w którym za punkt wyjścia bierze się **liczbę dostarczanych instrukcji kodu** *DSI* (ang. *delivered source-code instructions*):

$$MM = a \cdot (KDSI)^b$$

gdzie *MM* to (w oryginale) oszacowanie nakładu pracy wyrażone w **osobo-miesiącach** (ang. *man-month*), *KDSI* oznacza tysiące *DSI*, natomiast stałe *a* oraz *b* wynikają z rodzaju projektu:

System	*a*	*b*
organiczny	2,4	1,05
częściowo-wydzielony	3,0	1,12
wbudowany	3,6	1,20

Model COCOMO – nakłady pracy – przykład

Przykład:
Dla systemu organicznego, przy założeniu 10.000 *DSI*, otrzymamy

$$2,4 \cdot (10.000)^{1,05} = 26,92$$

co w przybliżeniu daje 27 osobo-miesięcy.

Model COCOMO – prognozowanie czasu

Nakłady czasowe wyznaczamy korzystając z modelu podstawowego dotyczącego nakładu pracy. Obecnie jednak za punkt wyjścia bierze się oszacowany nakład pracy wyrażony w osobomiesiącach *MM*:

$$TDEV = a \cdot (MM)^b$$

gdzie *TDEV* oznacza **całkowity czas projektu** (ang. *total development time*), natomiast stałe *a* oraz *b* wynikają z rodzaju projektu:

System	*a*	*b*
organiczny	2,5	0,38
częściowo-wydzielony	2,5	0,35
wbudowany	2,5	0,32

Model COCOMO – prognozowanie czasu – przykład

Przykład (kontynuacja):
Wracając do ostatniego przykładu można teraz wyliczyć całkowity czas projektu:

$$2,5 \cdot (26,92)^{0,38} = 8,73$$

a więc w przybliżeniu 9 miesięcy.

Klasy modeli COCOMO

Model COCOMO istnieje w trzech swoich zasadniczych klasach:

1. **model podstawowy** (ang. *basic COCOMO*) – celem modelu jest podanie jedynie oceny rzędu wielkości nakładów na wczesnym etapie projektowania;
2. **model pośredni** (ang. *intermediate COCOMO*) – model pośredni przewidziany jest do użycia, gdy główne komponenty systemu zostały już zidentyfikowane, uwzględnia on szereg szczegółowych aspektów projektu;

Klasy modeli COCOMO (cd.)

3. **model szczegółowy (model zaawansowany)** (ang. *detailed COCOMO*) – model szczegółowy używany jest po zdefiniowaniu poszczególnych modułów systemu.

W przypadku modelu pośredniego oraz szczegółowego dochodzi dodatkowy parametr, którym jest „napędzający koszty" mnożnik ustalany na podstawie szeregu atrybutów systemu:

$$E = (a \cdot size^b) \cdot F$$

Wersja pośrednia i szczegółowa COCOMO zawiera 15 elementów napędzających koszty, dla których podaje się odpowiednie mnożniki korygujące (wagi).

Rozwój metody – COCOMO II

Rozwój metody podstawowej:

- rozszerzeniem klasycznego modelu COCOMO jest opracowany pod kierunkiem Barry'ego Boehma **model COCOMO II**;
- nowy model wspiera dodatkowo spiralny cykl rozwoju oprogramowania, może być także zastosowany do oprogramowania zorientowanego obiektowo;
- w COCOMO II rezygnije się z trzech podstawowych modeli znanych z modelu klasycznego, a wykładniczy współczynnik funkcji uzależniony został od pięciu **czynników skali** (ang. *scaling factors*);

Rozwój metody – COCOMO II (cd.)

- uwzględnienia się wiele czynników, m.in.
  - atrybuty produktu (wymagana niezawodność, złożoność, rozmiar bazy danych),
  - cechy środowiska docelowego (szybkość pracy, pojemność pamięci komputera, i inne),
  - cechy zespołu pracującego nad projektem (umiejętności analityczne, programistyczne, doświadczenie, itp.), a także
  - atrybuty procesu wytwórczego (nowoczesność rozwiązań, wykorzystanie narzędzi, presję harmonogramu);
- model COCOMO II bazuje teraz na **liczbie logicznych linii kodu**, a nie liczbie linii kodu;

Rozwój metody – COCOMO II (cd.)

- nowy model wspiera także inne metody szacowania, jak np. metoda analizy punktów funkcyjnych;
- istnieje tendencja do tworzenia oddzielnych modeli (a nie np. COCOMO III), i tak w efekcie powstają:

  COSEMO – przeznaczony dla małych projektów (do 64-osobomiesiące),

  CORADMO – przeznaczony dla projektów wykonanych w technologii RAD,

  COCOTS – przeznaczony dla projektów realizowanych w technologii COTS.

Metoda COCOMO – podsumowanie

Krótkie podsumowanie metod z rodziny COCOMO:

- najpopularniejszy przedstawiciel modeli parametrycznych – w celu szacowania poszczególnych wielkości bazuje się na odpowiednich wzorach;
- szacowanie na podstawie wielkości, które nie są znane – głównym problemem staje się trudność trafnego oszacowania **a priori** parametrów modelu (w tym przede wszystkim liczby linii kodu);
- uwidacznia się ranga danych „historycznych”, pozwalających, w przypadku ustabilizowanego procesu wytwarzania, na uzyskiwanie dobrych rezultatów poprzez analizę statystyczną nagromadzonych danych i wykorzystanie ich do kalibracji modeli;

Metoda COCOMO – podsumowanie (cd.)

- w nowszych wersjach modelu COCOMO wprowadza się nowe, kolejne parametry modelu, stanowiących odbicie wielu aspektów rzeczywistych projektów – czasem doprowadza to do bardzo dużej liczby parametrów stwarzających trudności interpretacyjne (stąd m.in. tendencja do tworzenia modeli szczegółowych).

Z uwagi na niedokładność i subiektywizm metod szacowania nakładów zalecana jest estymacja projektu przy użyciu kilku z nich i uśrednianie uzyskiwanych rezultatów.

Propozycja

Propozycja do rozważenia:

- po analizie wymagań:
  - metoda FPA,
  - metoda COCOMO;
- po projekcie ogólnym
  - metoda delficka,
  - inne (metoda wartości rozmytych, metoda standardowego składnika);

Metryka nauki o programach Halsteada

Została zaproponowana przez M. Halsteada tzw. **nauka o programach** (ang. *Halstead's software science*). Linie kodu są nieistotne, a znaczenie mają jedynie jednostki syntaktyczne. Wyróżnia się **operatory** i **operandy**:

n_1 – liczba różnych operatorów w P;
n_2 – liczba różnych operandów w P;
N_1 – całkowita liczba wystąpień operatorów w P;
N_2 – całkowita liczba wystąpień operandów w P.

Słownik programu P zawiera $n = n_1 + n_2$ elementów. **Wielkość słownika** programu P wynosi $N = N_1 + N_2$. **Wolumin** programu P to minimalna liczba bitów niezbędna do zapisania programu.

Metryka nauki o programach Halsteada (cd.)

Twierdzenie

Szacunkowa wartość N wynosi $n_1 \log n_1 + n_2 \log n_2$. ⌟

Twierdzenie

Wysiłek potrzebny do wytworzenia P wynosi

$$E = \frac{n_1 N_2 N \log n}{2 n_2}$$

elementarnych jednostek. ⌟

Twierdzenie

Czas potrzebny do wytworzenia P wynosi $T = E/18 sek$. ⌟

M. Halstead zdefiniował szereg metryk opartych o elementy składniowe programu.

Liczba cyklomatyczna McCabe'a

Została zaproponowana przez T. McCabe'a tzw. **liczba (złożoność) cyklomatyczna** (ang. *McCabe's cyclomatic number*). Metoda bazuje na reprezentacji programu jako grafu sterowania – złożoność programu uzależniona od wewnętrznej struktury a nie fizycznej długości.

Jeżeli graf G jest schematem blokowym programu P i G posiada e **krawędzi (łuków) niezależnych przepływów sterowania** oraz n **węzłów instrukcji sterujących**, to złożoność cyklomatyczna CV:

$$CV = e - n + p$$

gdzie p to liczba składowych spójności (np. podprogramów).

Liczba cyklomatyczna McCabe'a (cd.)

Inne metryki McCabe'a:

złożoność projektu struktury modułowej programu – złożoność struktury hierarchii modułowej bez wnikania do treści wewnętrznej modułów;

złożoność struktury wewnętrznej kodu modułu – złożoność wnętrza modułów w odniesieniu do struktury programu (badanie np. „rozwlekłości" kodu).

Rozwinięciem podejścia McCabe'a są metryki McClure'a.

Waskaźniki zarowno Halsteada jak i McCabe'a były krytykowane z różnych punktów widzenia, twierdzono m.in., że nie są lepszymi wskaźnikami niż LOC.

Wybrane przykłady metryk obiektowych

Przykłady metryk obiektowych (pochodzące od różnych autorów i z różnych prac):

DIT – głębokość drzewa dziedziczenia obiektów;

NOC – liczba potomków w ramach dziedziczenia dla konkretnej klasy;

PCTPUB – procent danych publicznych (jako stosunek do danych prywatnych);

PUBDATA – dostępność danych publicznych (liczba dostępów);

PCTCALL – stosunek liczby nieprzeciążonych wywołań metod do wszystkich wywołań;

Wybrane przykłady metryk obiektowych (cd.)

ROOTCNT – globalna liczba korzeni w hierarchii dziedziczenia klas;

FANIN – liczba klas, z których wywodzi się analizowana klasa;

i naprawdę szereg, szereg innych.

Wybrane przykłady metryk dot. współbieżności

Przykady różnych metryk (szacowanie na podstawie zapisu projektu) dotyczących współbieżności:

GOLB – Graniczna Oczekiwana Liczba Blokad;

GOLZ – Graniczna Oczekiwana Liczba Zagłodzeń;

GPUB – Graniczne Prawdopodobieństwo Usunięcia Blokady (stosunek liczby blokad usuniętych do liczby zarejestrowanych blokad);

Wybrane przykłady metryk dot. współbieżności (cd.)

GOCO – Graniczny Oczekiwany Czas Odpowiedzi (na zadawane wymuszenia);

GPURC – Graniczne Prawdopodobieństwo Utrzymania Reżimów Czasowych (stosunek liczby prawidłowych odpowiedzi systemu spełniających wymagania czasowe do liczby wszystkich zachowujących i przekraczających wymagane czasy odpowiedzi);

GOOK – Graniczne Oczekiwane Obciążenie Komunikatami (intensywność przepływu komunikatów);

oraz inne.

Systemy do analizy kodu

Jest wiele systemów do statycznej analizy kodu, np.:

SonarQube narzędzie do ciągłej analizy statycznej kodu pod względem jakości. Umożliwia automatyczne wykrywanie błędów, słabych punktów powodujących obniżenie bezpieczeństwa. Potrafi też odnaleźć fragmenty, które zostały powielone, co pozwala zredukować ilość kodu. Głównymi korzyściami jest poprawa jakości i bezpieczeństwa tworzonego kodu.