Zaawansowana inżynieria oprogramowania

Szacowanie pracochłonności

Koncepcja wykładu: **Jerzy Nawrocki**
Mirosław Ochodek
Slajdy/Lektor/Montaż: **Mirosław Ochodek**

Witam Państwa serdecznie.

Wykład, w którym wezmą Państwo udział, będzie dotyczył szacowania pracochłonności wytwarzania oprogramowania.

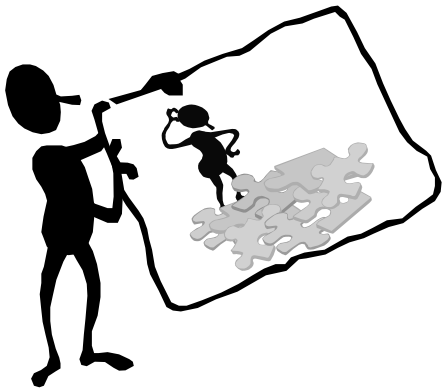
Zapraszam do wysłuchania i życzę miłych wrażeń.

Plan wykładów

Standardy serii ISO 9000
Model dojrzałości CMMI
Zarządzanie przedsięwzięciami i PRINCE2, cz. I
Zarządzanie przedsięwzięciami i PRINCE2, cz. II
Personal Software Process, cz. I
Personal Software Process, cz. II
Metodyki programowania: TSP i RUP
Pozyskiwanie i dokumentowanie wymagań (IEEE 830)
Wymagania pozafunkcyjne i ISO 9126
Zarządzanie ryzykiem
Systemy krytyczne i HAZOP
Szacowanie rozmiaru oprogramowania
Szacowanie pracochłonności

Szacowanie pracochłonności wytwarzania oprogramowania jest ostatnim wykładem z serii Zaawansowana inżynieria oprogramowania. Tematyka poruszana w jego trakcie jest ściśle powiązana z wcześniejszym wykładem poświęconym szacowaniu rozmiaru oprogramowania.

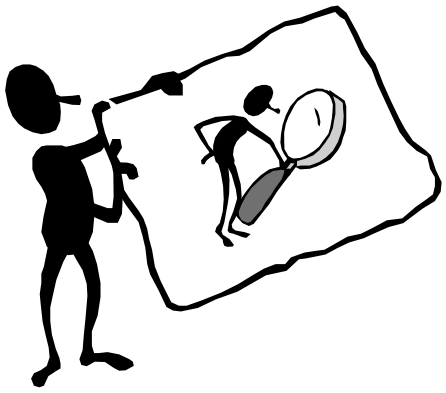
Plan wykładu



Wprowadzenie
Metoda delficka
COCOMO II
Use Case Points

Agenda wykładu przedstawia się następująco:

- **Wprowadzenie** - czyli zdefiniowanie problemu estymacji pracochłonności oraz przedstawienie zastosowań praktycznych.
- **Metoda delficka** - opierająca się na wspólnym szacowaniu wielu ekspertów.
- **COCOMO II** - metoda bazująca na rozmiarze oprogramowania.
- **Use Case Points** - szacowanie pracochłonności w oparciu o specyfikację wymagań.



Wprowadzenie

Metoda delficka

COCOMO II

Use Case Points

Przyjrzyjmy się bliżej problemowi szacowania pracochłonności i zastanówmy się, w jaki sposób można wykorzystać otrzymane estymacje.

Aby zobrazować istotę szacowania pracochłonności wytwarzania oprogramowania, posłużymy się przykładem **Gry Planistycznej**, zaproponowanej w metodyce **eXtreme Programming**. Metoda ta ma na celu ułatwienie pertraktacji zakresu funkcjonalności, przeznaczonej do realizacji w ramach następnego przyrostu.

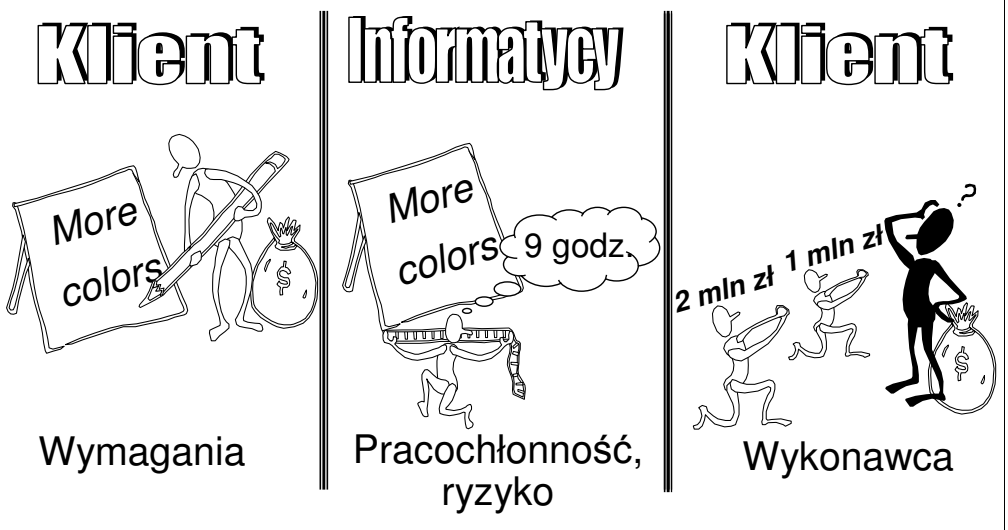
Pierwszym ogniwem jest Klient i wymagania przez niego sprecyzowane. Z perspektywy Klienta, wymagania mają przeważnie różne priorytety.

Po drugiej stronie znajduje się Dostawca oprogramowania, który musi ocenić, czy jest w stanie zaimplementować daną funkcjonalność, w przewidzianym na to czasie. W tym właśnie miejscu pojawia się szacowanie pracochłonności.

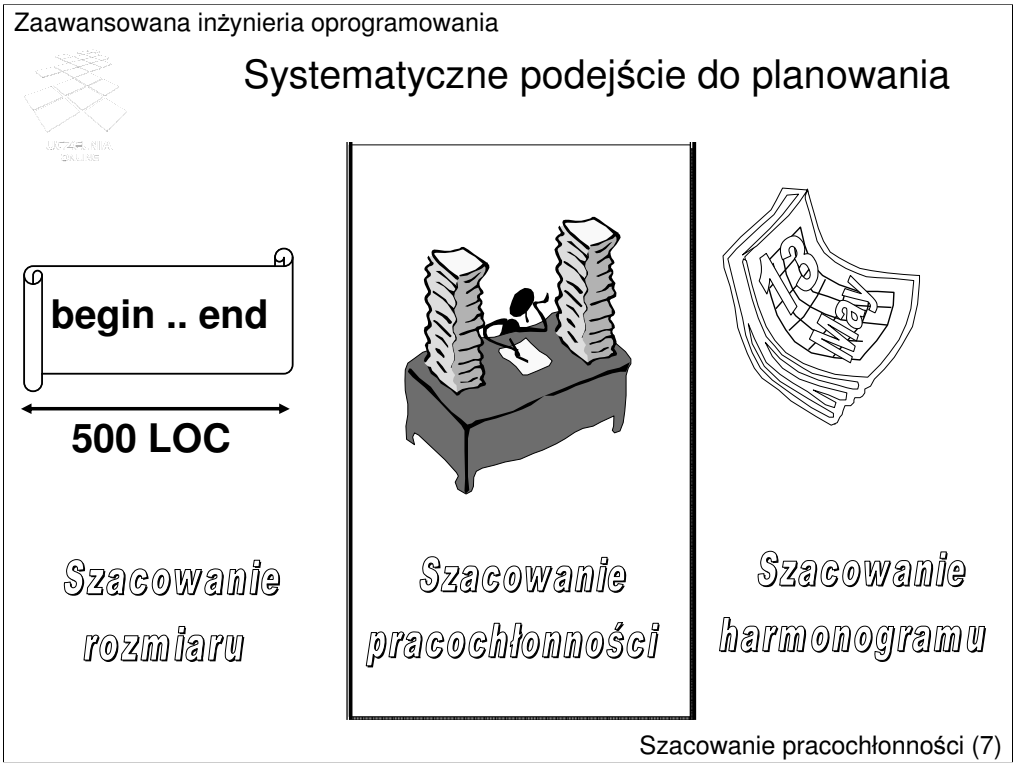
Rolą Klienta w Grze Planistycznej, jest sprecyzowanie wymagań i uszeregowanie ich według własnych priorytetów. Natomiast Dostawca oprogramowania powinien zbadać wykonalność i czas implementacji poszczególnych funkcji. Innymi słowy, należy oszacować pracochłonność, czyli określić sumaryczny czas potrzebny na wytworzenia oprogramowania.

Ostatnim etapem negocjacji z Klientem jest wybór funkcjonalności do następnego wydania. Dysponujemy priorytetami Klienta, które w połączeniu z oszacowaniem pracochłonności, umożliwiają dokonanie wyboru satysfakcjonującego obie strony.

Szacowanie pracochłonności w przetargach



Podobną sytuację można zaobserwować przyglądając się przetargom na tworzenie systemów informatycznych. Klient także definiuje wymagania odnośnie przyszłego systemu. Natomiast organizacje startujące do przetargu muszą ocenić pracochłonność przedsięwzięcia, aby móc oszacować koszt wytworzenia oprogramowania i przedłożyć własną ofertę.



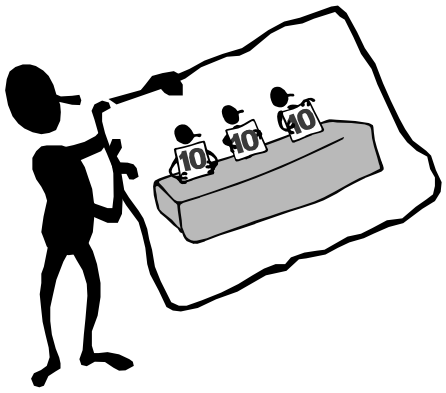
Aby, lepiej wyjaśnić różnicę między szacowaniem rozmiaru oprogramowania, a pracochłonnością posłużmy się przykładem systematycznego podejścia do planowania. Podejście to zakłada trzy filary postępowania, które umożliwiają sprawne planowanie.

Pierwszym etapem jest określenie rozmiaru oprogramowania. Istnieje wiele metryk pozwalających na przedstawienie złożoności, czy też rozmiaru oprogramowania. Jedną z najbardziej intuicyjnych jest liczba **źródłowych linii kodu** (**SLOC**).

Na podstawie oszacowanego rozmiaru oprogramowania, możemy starać się ocenić pracochłonność, wykorzystując dostępne modele zależności wiążących ze sobą te dwie wielkości.

Posiadając oszacowania pracochłonności można natomiast przystąpić do tworzenia harmonogramu.

Metoda Delficka



Wprowadzenie

Metoda delficka

COCOMO II

Use Case Points

Najprostsza metoda szacowania opiera się na wiedzy i intuicji ekspertów. Jedną z metod wykorzystujących to podejście jest metoda delficka.

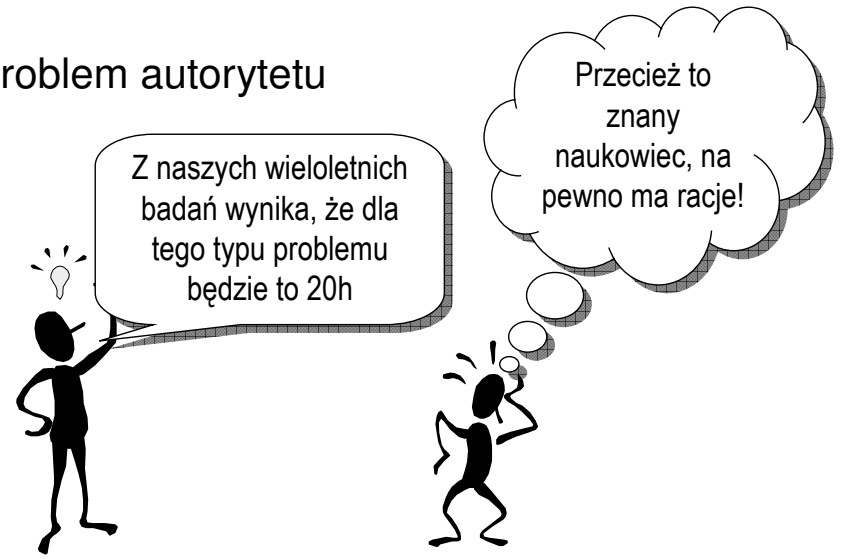
Opieranie się na wiedzy, jest praktyką wykorzystywaną dość powszechnie. Jednak proces negocjacji, w szerszym gronie ekspertów niesie ze sobą pewne problemy, związane z ludzką psychiką.

Oparcie szacowania o większą liczbę ekspertów ma na celu zminimalizowanie ewentualnego błędu subiektywnej oceny jednostek.

Oczywiście wiążą się z tym pewne problemy. Załóżmy na przykład, że istnieje między ekspertami hierarchiczna zależność, jak na przykład między szefem i pracownikiem. Wówczas sprzeciwienie się przełożonemu na forum, z pewnością nie jest łatwym zadaniem i negatywnie wpływa na obiektywizm takiego eksperta.

Problemy podczas negocjacji

Problem autorytetu



Podobnym problemem jest różnica autorytetów. Jeśli jedna osoba, w grupie jest na przykład szanowanym naukowcem, może ona wpływać na innych wykorzystując swój autorytet. Podobnie jak w przypadku hierarchicznej zależności, może utrudniać to prowadzenie negocjacji jak równy z równym.

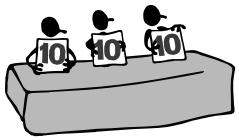
Problemy podczas negocjacji

Problem urażonej dumy

Natura ludzka sprawia, że niejednokrotnie trudno jest nam przyznać się do błędu. Problem ten dotyczy także procesu negocjacji. Może zdarzyć się, że osoba świadoma swojego błędu, będzie podtrzymywała swoje zdanie unosząc się dumą.

Metoda delficka - informacje

- Rand Corporation, koniec lat 40-tych XX wieku
- Kilku ekspertów indywidualnie szacuje nakład (rozmiar)
- Stosując metodę delficką dochodzi się do konsensusu
- Jest wiele różnych odmian metody (np. Wideband Delphi), ale idea jest wspólna



Jedną z metod, próbujących niwelować problemy negocjacji jest metoda delficka. Została ona zaproponowana pod koniec lat 40 XX wieku przez **Rand Corporation**.

Idea polega na indywidualnym, niezależnym szacowaniu przez kilku ekspertów.

Stosując metodę delficką staramy się osiągnąć konsensus, czyli wypracować wspólne zdanie.

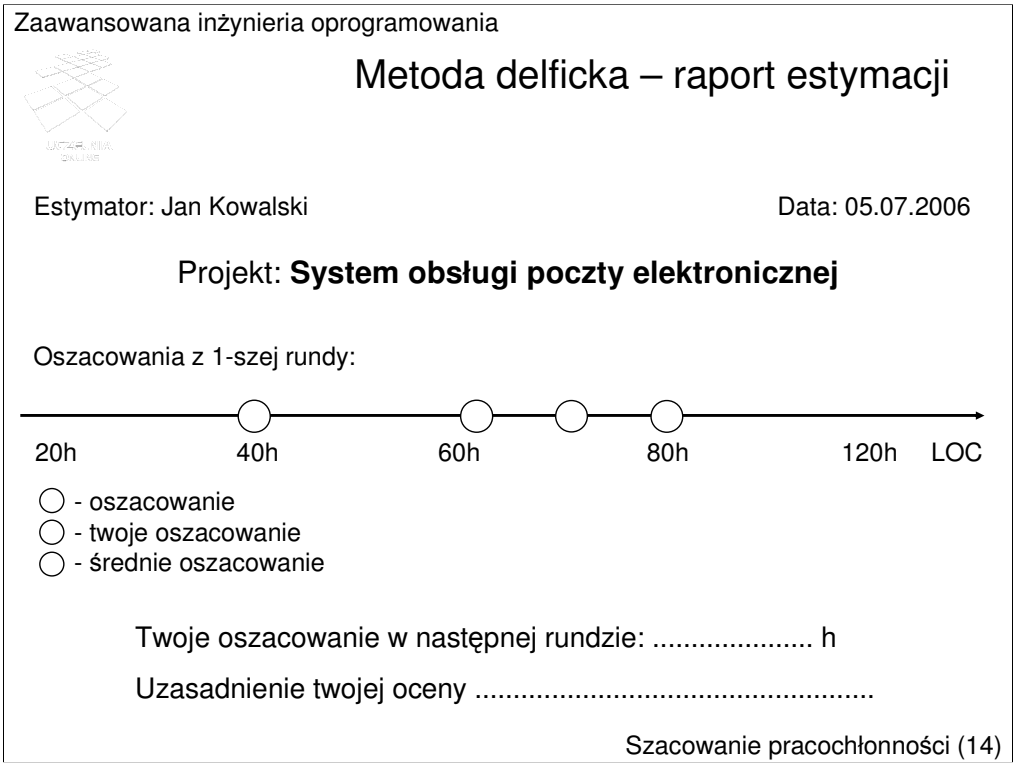
Metoda zyskała sobie dość duża popularność, co zaowocowało powstaniem wielu modyfikacji. Jedna z najbardziej znanych wersji jest wariant **Wideband Delphi**. Właśnie ten wariant zostanie omówiony podczas tego wykładu.

Metoda delficka - algorytm

1. Eksperci dostają specyfikację systemu i formularz estymacyjny.
2. Spotykają się by przedyskutować: cele projektu, założenia, problemy estymacji.
3. Ekspert anonimowo ocenia zadania i szacuje nakład (rozmiar).
4. Szacunki trafiają do moderatora, który opracowuje wyniki i przedstawia je ekspertom.

Algorytm metody delfickiej, w wersji **Wideband Delphi**, składa się z siedmiu kroków.

1. Pierwszym etapem jest zaopatrzenie ekspertów w odpowiednie materiały przybliżające zakres projektu (np. dokument specyfikacji wymagań) oraz specjalny formularz estymacji. Każdy z ekspertów otrzymuje również czas, aby przeanalizować dokumentację.
2. Kolejnym etapem jest wymiana poglądów na temat projektu w gronie ekspertów. Eksperci dzielą się uwagami i spostrzeżeniami wyniesionymi podczas lektury dokumentacji.
3. Po przeprowadzeniu indywidualnej i grupowej analizy zebranej dokumentacji, można przystąpić do głosowania. Każdy z ekspertów precyzuje własne oszacowanie pracochłonności. Niezwykle istotne jest, aby głosowanie przebiegało anonimowo.
4. Oszacowania ekspertów trafiają do moderatora, czyli osoby nadzorującej proces negocjacji. Jego zadaniem jest odpowiednie opracowanie wyników głosowanie i przeniesienie ich na formularze estymacji poszczególnych ekspertów.



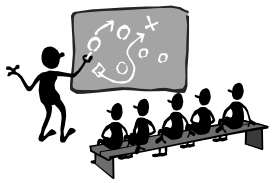
Sporządzony przez moderatora raport estymacji, jest unikalny dla każdego eksperta. Oprócz danych informacyjnych, jak nazwisko eksperta, data głosowania czy nazwa projektu, raport zawiera umiejscowienie oceny eksperta na tle pozostałych oszacowań. Przedstawiona zostaje ocena własna eksperta, oceny pozostałych ekspertów (bez informacji o personaliach głosujących) oraz średnie oszacowanie.

Umożliwia to weryfikację własnego stanowisko i jego krytyczną ocenę. Ekspert ma możliwość porównania wystawionych ocen ze średnią wartością oraz ocenami pozostałych osób. Nie jest natomiast podawana informacja umożliwiająca powiązanie ocen z konkretnymi osobami.

Analiza danych przedstawionych w raporcie ma przede wszystkich na celu skłonić autora oszacowań do refleksji i powtórnego przyjrzeniu się problemowi.

Metoda delficka - algorytm

5. Eksperci spotykają się by przedyskutować wyniki. Omawiają zadania, które zdefiniowali, ale nie dyskutują ich szacunkowych nakładów (rozmiarów)



6. Procedura jest powtarzana od kroku 3, aż szacunki ekspertów są dostatecznie zbliżone.

5. Po przeanalizowaniu raportów estymacji, eksperci ponownie zasiadają do dyskusji. Jest ona okazją do powtórnego przeanalizowania zadań oraz czynników wpływających na pracochłonność.

Aby zachować anonimowość negocjacji należy wystrzegać się podawania dokładnych wartości oszacowań. Bardziej istotne jest nakreślanie problemów, oraz obrazowanie przesłanek, w oparciu o które eksperci podejmowali swoje decyzje.

6. Cała procedura jest powtarzana dopóki oszacowania ekspertów nie będą wystarczająco zbliżone.

Metoda delficka - algorytm

7. Oszacowanie końcowe

Oszacowanie = (P + 4A + O) /6

P – ocena pesymistyczna
A – ocena średnia
O – ocena optymistyczna

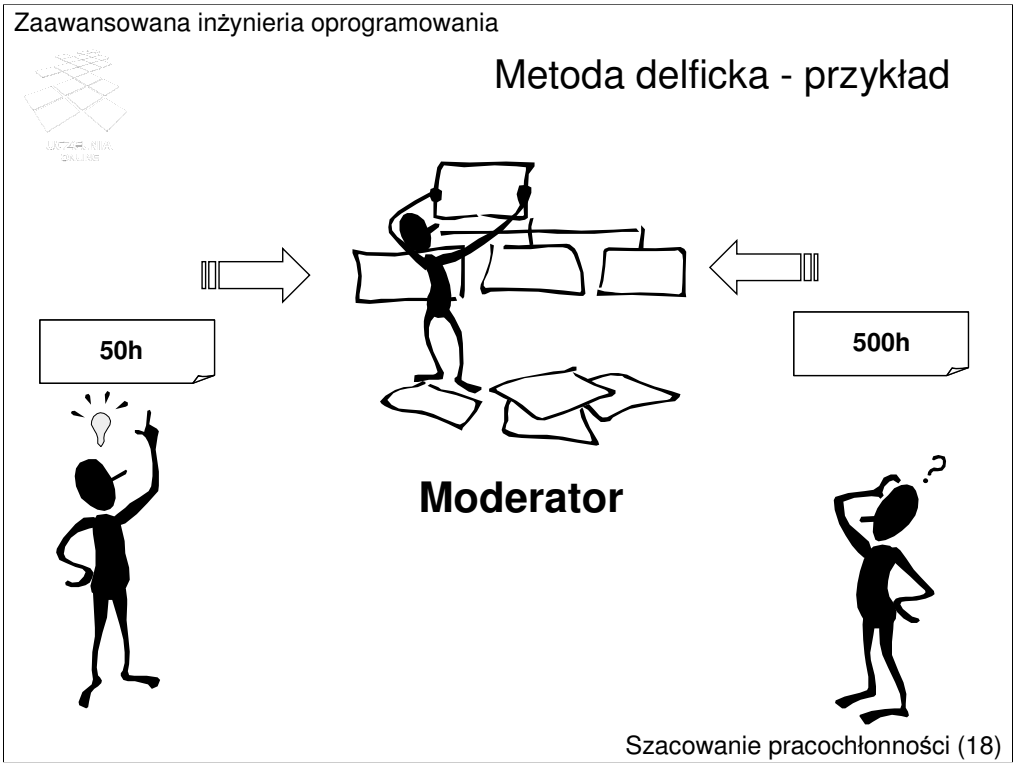
Jeśli nie udało się wypracować wspólnej oceny dla wszystkich ekspertów, należy posłużyć się średnią ważoną, dla określenia oszacowania końcowego. Średnia ta uwzględnia **ocenę pesymistyczną** oraz **optymistyczną z wagą 1** oraz **ocenę średnią z wagą 4**.

Można powiedzieć, że przychylamy się do średniej oceny, mając na uwadze także skrajne oszacowania.

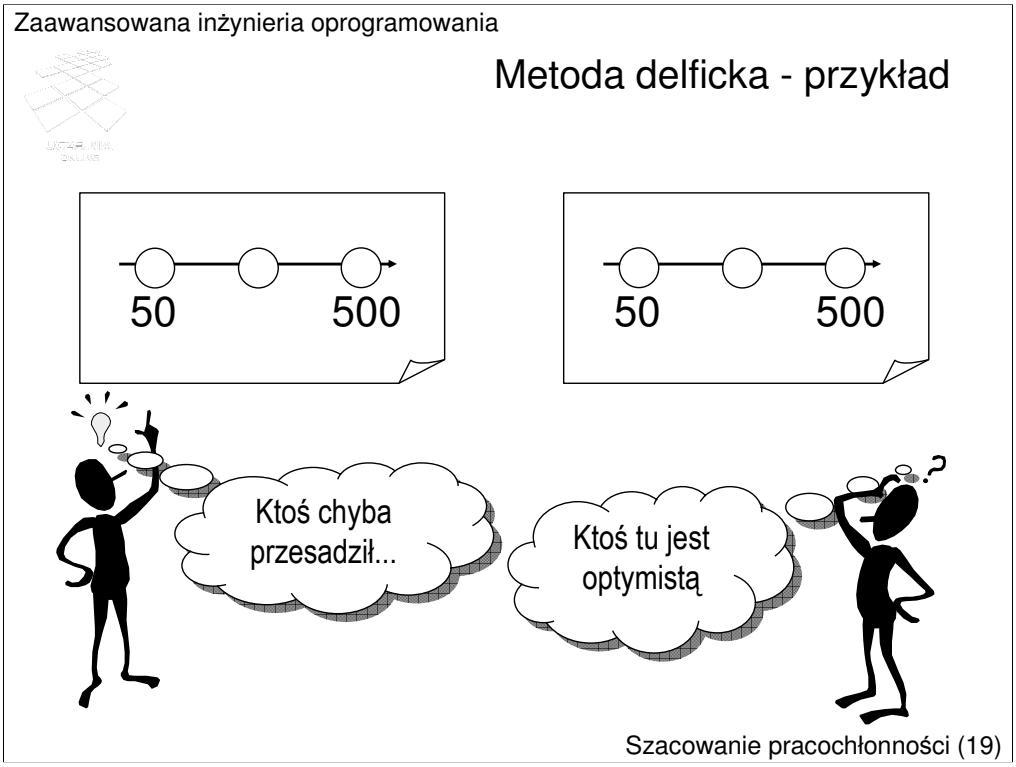
Przeanalizujmy działanie metody na przykładzie szacowania pracochłonności dla prostego edytora języka programowania.

Dla uproszczenia załóżmy, że w negocjacjach bierze udział dwóch ekspertów. Oczywiście w praktyce uniemożliwiłoby to zachowanie anonimowości.

Nasi eksperci różnią się znacznie w swoich ocenach na temat pracochłonności budowy narzędzia. Pierwszy ekspert założył możliwość użycia narzędzi typu RAD (*ang. Rapid Application Development*) i orzekł, że pracochłonność powinna wynieść w granicach 50 roboczogodzin. Natomiast ekspert drugi dostrzegł problem kolorowania składni, który wykracza poza standardowe komponenty. Ta przesłanka skłoniła go do ustalenia pracochłonności na poziomie 500 roboczogodzin.

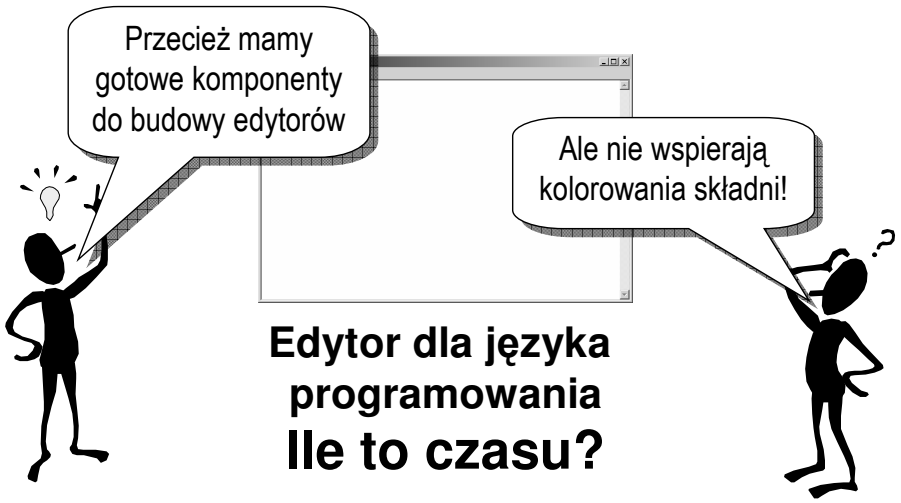


Oceny zostają przekazane moderatorowi z zachowaniem poufności. Ten natomiast sporządza raporty estymacji i rozsyła je do ekspertów.

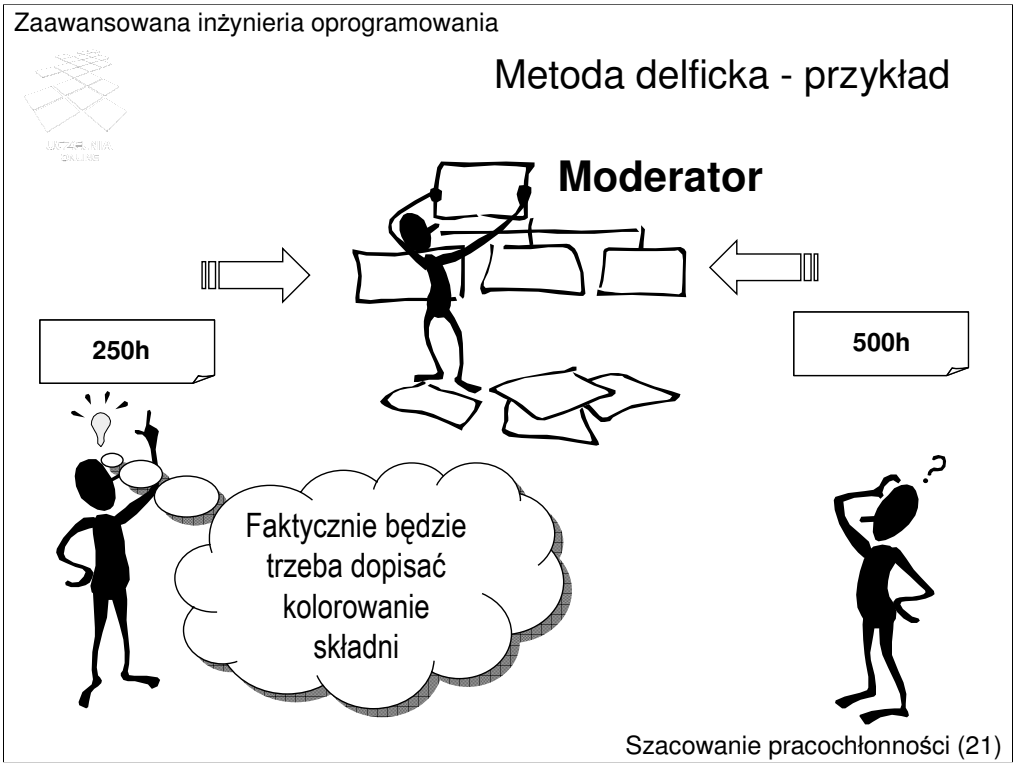


Eksperci analizują swoje głosy na tle innych. Każdy z nich widzi wyraźnie, że jego ocena odbiega od pozostałych. Mechanizm utrzymania anonimowości nie pozwala na skojarzenie poszczególnych ocen z osobami ekspertów.

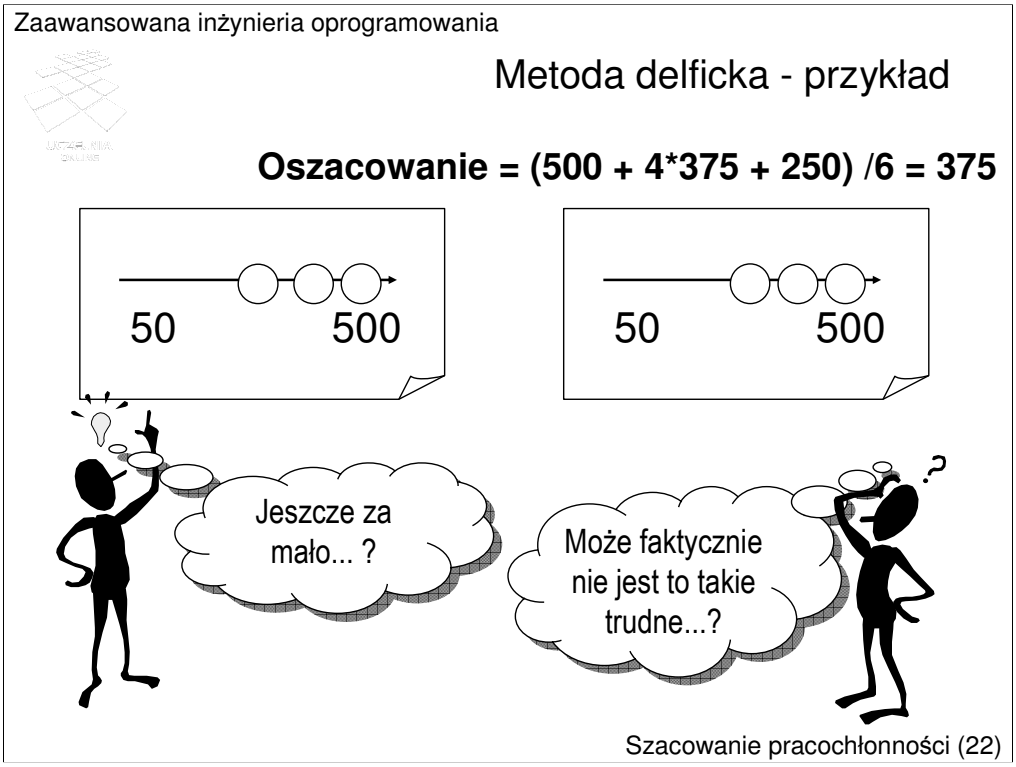
Metoda delficka - przykład



Eksperci spotykają się by omówić wyniki estymacji. Wymieniają swoje spostrzeżenia na temat zadań i problemów.

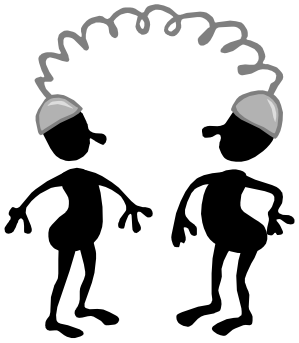


Analiza ocen pozostałych ekspertów oraz dyskusja mogą sprawić, że ekspert zweryfikuje swoją ocenę. W naszym przykładzie pierwszy ekspert doszedł do wniosku, że nie uwzględnił problemu kolorowania składni.



Proces ten przebiega iteracyjnie. W większości przypadków sprawia, że przedział ocen ekspertów stopniowo zawęża się. Gdy oszacowania są już wystarczająco spójne można wyznaczyć wartość końcową.

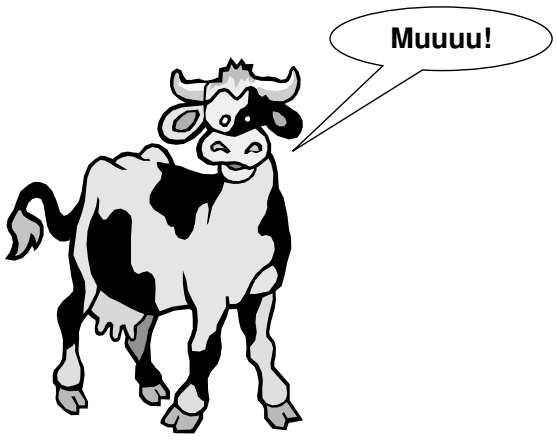
Metoda delficka - burza mózgów



Gra delficka jest jak burza mózgów...

Gra delficka ma charakter burzy mózgów. Wymiana informacji między ekspertami stymuluje proces negocjacji i umożliwia spojrzenie na problem z punktu widzenia wielu osób. Analiza raportów zmusza do refleksji nad własną oceną i rodzi potrzebę poszukiwania przyczyn rozbieżności ocen.

Metoda delficka – zmiana zdania

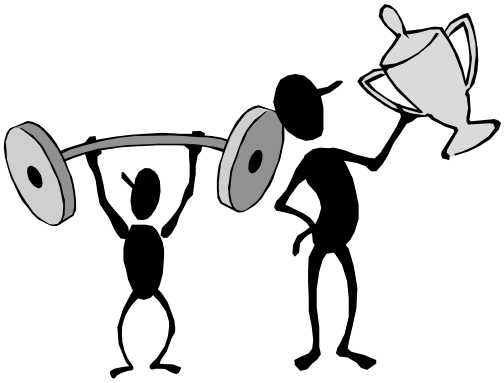


„Tylko krowa nie zmienia zdania ... bo go nie ma...”

Należy stworzyć warunki do wycofania się ze swojego zdania

Jedną z największych zalet metody jest możliwość wycofania się z wcześniejszych decyzji, na każdym etapie negocjacji. Każdy, nawet ekspert posiadający ogromne doświadczenie i wiedzę może się mylić. Jednak nie każdy potrafi się do tego przyznać. Metoda delficka niweluje ten problem, wprowadzając anonimowe głosowania.

Metoda delficka – nie tylko do szacowania

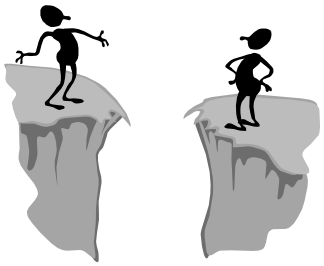


Trening czyni mistrza...

Aby metoda była skuteczna powinna stać się nawykiem

Algorytm gry delfickiej niesie ze sobą drobne utrudnienia w porównaniu z typowym przebiegiem negocjacji w zespole. Warto zatem zadbać o to, by metoda stała się naturalną formą podejmowania kluczowych decyzji. Aby lepiej przyswoić sobie jej zasady, można wykorzystywać ją także, poza biurem. Na przykład szacując koszty wycieczki z przyjaciółmi. Dopiero intensywne wykorzystywanie na każdym kroku, pozwoli stać się jej dobrze zakorzenionym nawykiem.

Metoda delficka – problem



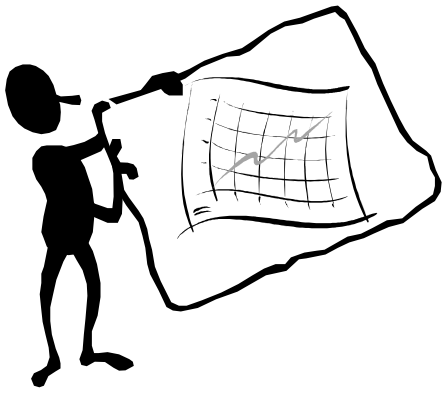
Eksperci nie mogą dojść do porozumienia

Oczywiście metoda delficka jest także podatna na pewne problemy związane z wykorzystaniem metod bazujących na wiedzy ekspertów. Podstawowy problem polega na tym, że w ramach metody, zakłada się stopniową poprawę ocen wraz z postępem negocjacji.

Co jeśli eksperci nie dojdą do porozumienia?

Metoda delficka posiada wprawdzie mechanizm obliczania oceny końcowej, jako średniej ważonej z ocen ekspertów. Nie oznacza to niestety, że w przypadku skrajnych ocen ekspertów otrzymamy trafne oszacowanie.

Plan wykładu

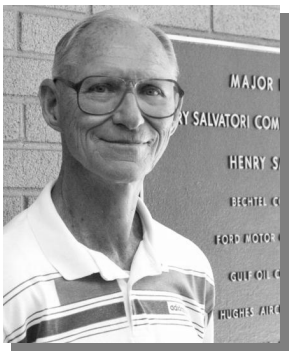


Wprowadzenie
Metoda delficka
COCOMO II
Use Case Points

Inne podejście od szacowanie pracochłonności proponuje metoda **COCOMO II** , bazująca na **rozmiarze oprogramowania**. Metoda opiera się na równaniach wyprowadzonych w oparciu o dane historyczne przeszło stu rzeczywistych projektów informatycznych.

COCOMO II

COCOMO - COnstructive COst MOdel



Barry W. Boehm

1957: BA, Matematyka, Harvard

1961: MS, Matematyka, UCLA

1964: PhD, Matematyka, UCLA

1959-73: Rand Corporation

1973-89: TRW

1989-92: Department of Defence (DoD)

1993-teraz: USC Center for SE

Metoda została zaproponowana przez **Barrego Boehma** i jej nazwa COCOMO jest skrótem od **COnstructive COst MOdel**. Twórca metody jest z wykształcenia matematykiem, posiada szerokie doświadczenie zarówno akademickie, jak i przemysłowe. Pracował między innymi w Rand Corporation, czyli w firmie, która stworzyła metodę delficką. Co więcej w latach siedemdziesiątych współtworzył metodę Wideband Delphi, czyli wariant metody omawiany we wcześniejszej części wykładu.

Następnym dziełem Boehma, w zakresie szacowania pracochłonności było stworzenie metody COCOMO, zwanej również COCOMO81, od roku powstania. Właśnie na bazie tej metody powstała metoda COCOMO II.

COCOMO II - modele

- Rozwinięcie metody COCOMO81
- Dostosowanie do przyrostowego wytwarzania oprogramowania (modele):
  - Application Composition Model
  - The Early Design Model
  - The Post-Architecture Model

Wraz z rozwojem technik wytwarzania oprogramowania okazało się, że metoda COCOMO81, posiada pewne braki. Przyczyną takiego stanu rzeczy, był gwałtowny postęp w rozwoju, zarówno narzędzi programistycznych jak i samych metodyk zarządzania przedsięwzięciami informatycznymi.

Odpowiedzią na ten problem było stworzenie metody COCOMO II, lepiej przystosowanej do nowoczesnych metod wytwarzania oprogramowania.

Metoda ta uwzględnia trzy modele, lepiej wpasowujące się w przyrostowe cykle wytwarzania oprogramowania.

•Pierwszy model – **Application Composition Model** – jest przeznaczony dla projektów wykorzystujących nowoczesne narzędzia do tworzenie (generowania) interfejsów użytkownika (czyli na przykład dla prototypów),

•**Early Design Model** można wykorzystywać we wczesnym stadium projektu (w momencie tworzenia wstępnej architektury),

•Wreszcie **Post-Architecture Model**, który jest najbardziej szczegółowym w obrębie metody. Można go stosować, gdy została już opracowana architektura systemu. Jest to również najbardziej przydatny i najczęściej stosowany model. Uwarunkowane jest to, przede wszystkim tym, że metoda COCOMO bazuje na rozmiarze oprogramowania, który o wiele łatwiej oszacować posiadając zdefiniowaną architekturę systemu (chociażby poprzez użycie metody punktów funkcyjnych).

COCOMO II – model post-architektoniczny

- Miara **PM** (*Person Month*) – zakłada miesiąc pracy jednej osoby w ramach projektu
  - Nie wliczono świąt / dni wolnych
  - Wliczono weekendy
- $PM_{nominal}$ – podstawowy wzór
  - $PM_{nominal} = A \times (Size)^{E}$
    - Stała
    - Stała
    - Size w KSLOC
    - Czynniki skali (*Scale Factors*)
    - E – czynnik skali ($E = B + 0{,}01 \times \sum_{i=1}^{5} SF_i$)

Jednostką, w której mierzy się pracochłonność w metodzie **COCOMO** jest **osobomiesiąc**. Jeden osobomiesiąc oznacza miesięczny czas pracy jednej osoby nad danym projektem, przy czym:

•nie wliczono dni świątecznych i innych dni wolnych od pracy,

•wliczono natomiast weekendy.

Podstawowy wzór określający pracochłonność (**PM nominal**) wyznaczono jako: A razy Size do potęgi E. Przy czym rozmiar (Size) przyjmowany jest w jednostce **KSLOC**, czyli tysiącach źródłowych linii kodu.

Natomiast **E** , stanowi czynnik skali dostosowujący pracochłonność. Jest to suma stałej B oraz iloczynu liczby 0,01 przez sumę pięciu współczynników skali.

COCOMO II – model post-architektoniczny

- $PM_{adjusted} = PM_{nominal} \times \prod_{i=1}^{16} EM_i$

Mnożnik Pracochłonności (*Effort Multiplier*)

- $PM_{adjusted} = A \times (Size)^E \times \prod_{i=1}^{16} EM_i$
  - E – czynnik skali ($E = B + 0{,}01 \times \sum_{i=1}^{5} SF_i$)
  - Wartości A, B skalibrowane na podstawie 161 projektów:
    - A = 2,94
    - B = 0,91
  - Suma czynników skali $0 \le \sum_{i=1}^{5} SF_i \le 31{,}6$
  - $0{,}91 \le E \le 1{,}226$

Model post-architektoniczny dostosowuje oszacowanie nominalne, wymnażając je przez iloczyn 16 współczynników, zwanych mnożnikami pracochłonności. Stałe współczynniki A i B, zostały natomiast wyznaczone na podstawie 161 projektów i wynoszą:

•A = 2,94

•B = 0,91

Można zatem określić, że wartości brzegowe przedziału współczynników skali to 0,91 i 1,226.

COCOMO II – model post-architektoniczny

- $PM_{adjusted} = 2{,}94 \times (Size)^{E} \times \prod_{i=1}^{16} EM_i$
  - Dla przeciętnego projektu
    - $EM_i = 1$, zatem $\prod_{i=1}^{16} EM_i = 1$
- $PM_{adjusted} = 2{,}94 \times (Size)^{E}$ - przeciętny projekt
  - $0{,}91 \leq E \leq 1{,}226$

Okazuje się, że dla przeciętnego projektu, mnożniki pracochłonności wynoszą 1. Zatem iloczyn tych współczynników także będzie także równy 1. Oznacza to, że współczynniki te są pomijalne dla przeciętnego projektu.

COCOMO II – czynniki skali SF_i

	Very low	Low	Nominal	High	Very high	Extra high
Typowość	6.20	4.96	3.72	2.48	1.24	0.00
Elastyczność	5.07	4.05	3.04	2.03	1.01	0.00
Zarz. ryzykiem	7.07	5.65	4.24	2.83	1.41	0.00
Spójność zespołu	5.48	4.38	3.29	2.19	1.10	0.00
Dojrzałość proc.	7.80	6.24	4.68	3.12	1.56	0.00

W ramach metody **COCOMO II** wyszczególniono **5 czynników skali** projektu. Są to:

•Typowość,

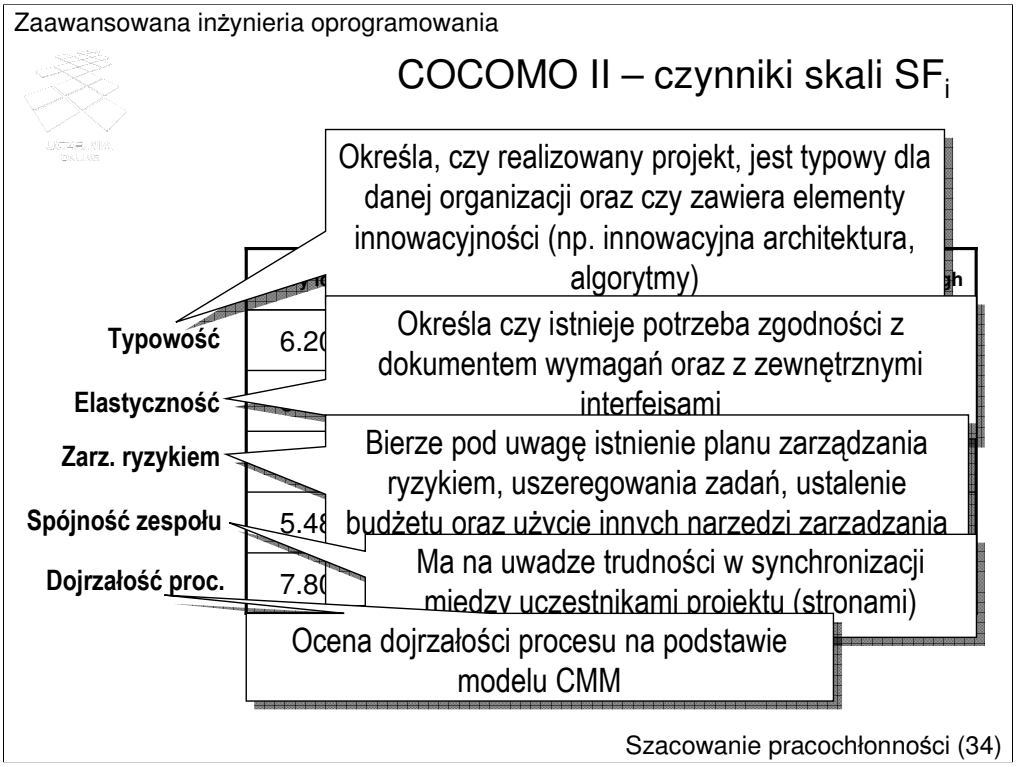
•Elastyczność,

•Zarządzanie ryzykiem,

•Spójność zespołu,

•Dojrzałość procesu wytwarzania.

W zależności od oceny poszczególnych czynników w badanym projekcie, wyznacza się ich wpływ w skali od bardzo niskiego (*very low*), do ekstra wysokiego (*extra high*).



Co kryje się pod nazwami czynników?

•**Typowość** - określa, czy realizowany projekt, jest typowy dla danej organizacji oraz czy zawiera elementy innowacyjności (np. innowacyjna architektura, algorytmy)

•**Elastyczność** - określa czy istnieje potrzeba zgodności z dokumentem wymagań oraz z zewnętrznymi interfejsami

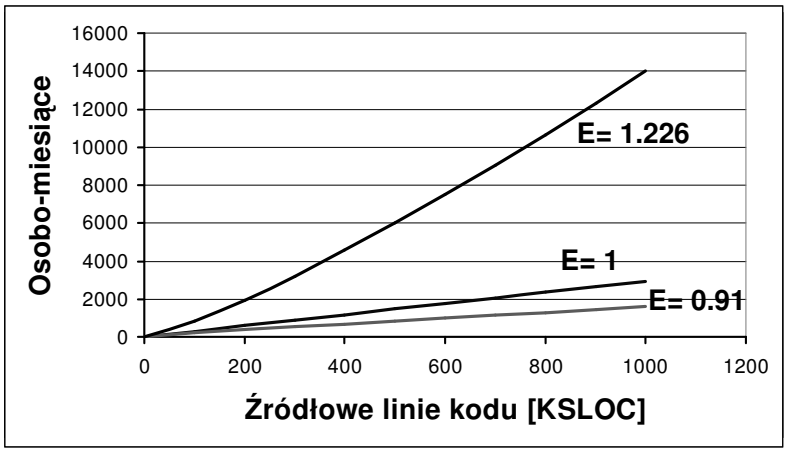
•**Zarządzanie Ryzykiem** - bierze pod uwagę istnienie planu zarządzania ryzykiem, uszeregowania zadań, ustalenie budżetu oraz użycie innych narzędzi zarządzania ryzykiem

•**Spójność zespołu** - ma na uwadze trudności w synchronizacji między uczestnikami projektu (stronami)

•**Dojrzałość procesu** - Ocena dojrzałości procesu na podstawie modelu CMM

Dokładny opis poszczególnych czynników skali można znaleźć w dokumentacji metody.

COCOMO II - wpływ SF_i na pracochłonność



Czynnik Skali E, zależny od cząstkowych czynników skali jest wartością należącą do przedziału <0,91;1,226>. W zależności od wartości współczynnika zmienia się także zależność pomiędzy liczbą linii kodu a pracochłonnością. Jeśli współczynnik ten ma wartość 1, oznacza to liniową zależność pracochłonności od rozmiaru. W pozostałych przypadkach jest to zależność nieliniowa. Może ona narastać łagodniej, jeśli E < 1, lub gwałtowniej, jeśli wartość jest większa od 1.

Oznacza to, że chcąc przyspieszyć czas realizacji zadania, nie możemy w większości przypadków zakładać, że koszt przyspieszenia rośnie liniowo, w zależności od skrócenia czasu realizacji.

COCOMO II – mnożniki pracochłonności EM_i

- Ustalane w zależności od przyjętego modelu
- Opisują cechy takie jak:
  - –Niezawodność
  - –Rozmiar bazy danych
  - –Złożoność produktu
  - –Re-używalność
  - –Umiejętności analityków
  - –Umiejętności programistów
  - –...

Wyznaczenie mnożników pracochłonności różni się w zależności od przyjętego modelu. Współczynniki te skupiają się na własnościach końcowego produktu oraz na zdolnościach pracowników. Są to na przykład:

•Niezawodność

•Rozmiar bazy danych

•Złożoność produktu

•Re-używalność

•Umiejętności analityków

•Umiejętności programistów

Dokładna lista mnożników, znajduje się w dokumentacji metody.

COCOMO II – Przykład

Pewne przedsięwzięcie programistyczne zostało uznane (na etapie planowania) za przeciętne.

Po przeanalizowaniu modelu post-architektonicznego wykonawcy doszli do wniosku, że trzeba będzie napisać ok. 10 000 SLOC w C++.

Typowość i zarządzanie ryzykiem oceniono jako wysokie (High).

Pozostałe czynniki skali oceniono jako bardzo wysokie (Very High).

Jaka będzie pracochłonność tego przedsięwzięcia przy założeniu, że harmonogram będzie budowany bez specjalnej presji czasu?

Spróbujmy oszacować pracochłonność dla pewnego przedsięwzięcia informatycznego uznanego na etapie planowania za przeciętne.

Po przeanalizowaniu modelu post-architektonicznego wykonawcy doszli do wniosku, że trzeba będzie napisać ok. 10000 SLOC w języku C++.

Typowość i zarządzanie ryzykiem oceniono jako wysokie (High).

Pozostałe czynniki skali oceniono jako bardzo wysokie (Very High).

Jaka będzie zatem, pracochłonność tego przedsięwzięcia przy założeniu, że harmonogram będzie budowany bez specjalnej presji czasu?

Model post-architektoniczny

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{16} EM_i$$

Size w KSLOC

gdzie $E = B + 0.01 \times \sum_{i=1}^{5} SF_i$

Wartości **A**, **B** skalibrowane na podstawie 161 projektów:

A = 2.94

B = 0.91

Dla przeciętnego projektu $EM_i = 1.$

$0 \leq \sum_{i=1}^{5} SF_i \leq 31.6$

$$PM_{NS} = 2.94 \times Size^{E}$$

gdzie $0.91 \leq E \leq 1.226$

Powróćmy na chwile do modelu post-architektonicznego i zastanówmy się jakie wartości są nam niezbędne do wyznaczenia pracochłonności.

- Po pierwsze potrzebujemy rozmiaru oprogramowania mierzonego w tysiącach źródłowych linii kodu,
- Konieczne jest wyliczenie współczynnika skali,
- Projekt uznano za przeciętny, zatem mnożniki pracochłonności są pomijalne.

COCOMO II – Przykład

Pewne przedsięwzięcie programistyczne zostało uznane (na etapie planowania) za przeciętne.

Po przeanalizowaniu modelu post-architektonicznego wykonawcy doszli do wniosku, że trzeba będzie napisać ok. 10 000 SLOC w C++.

Typowość i zarządzanie ryzykiem oceniono jako wysokie (High).

Pozostałe czynniki skali oceniono jako bardzo wysokie (Very High).

Jaka będzie pracochłonność tego przedsięwzięcia przy założeniu, że harmonogram będzie budowany bez specjalnej presji czasu?

Wyznaczmy czynniki skali. Typowość i zarządzanie ryzykiem oceniono jako wysokie natomiast pozostałe czynniki skali jako bardzo wysokie.

COCOMO II – Przykład

	Very low	Low	Nominal	High	Very high	Extra high
Typowość	6.20	4.96	3.72	2.48	1.24	0.00
Elastyczność	5.07	4.05	3.04	2.03	1.01	0.00
Zarz. ryzykiem	7.07	5.65	4	2.83	1.41	0.00
Spójność zesp				2.19	1.10	0.00
Dojrzałość pr				3.12	1.56	0.00

$$\sum_{i=1}^{5} SF_i = 8.98 \approx \mathbf{9}$$

Korzystając z tablicy czynników skali wyznaczamy szukane wartości poszczególnych czynników, po czym obliczamy sumę. W tym przypadku jest to wartość 8,98, czyli przyjmujemy w przybliżeniu 9.

COCOMO II – Przykład

$$PM_{NS} = A \times 10^{E} \times \prod_{i=1}^{16} EM_i$$ *Size w KSLOC*

gdzie **E = B + 0.01*9=1**

Wartości **A**, **B** skalibrowane na podstawie 161 projektów:

A = 2.94

B = 0.91

Dla przeciętnego projektu $EM_i = 1.$

$$PM_{NS} = 2.94 \times 10^{1} \approx 30$$

Podstawmy wartości do wzorów.

•Rozmiar oprogramowania oszacowano jako 10 000 SLOC. Do wzoru podstawiamy 10, pamiętając, że miarą rozmiaru dla metody COCOMO są tysiące źródłowych linii kodu,

•Wyznaczamy współczynnik skali E, korzystając z obliczonej sumy cząstkowych czynników skali,

•Natomiast wartości A i B są stałe,

•Jest to przeciętny projekt zatem mnożniki pracochłonności są pomijalne,

Efekt - szacowana pracochłonność wynosi 30 osobomiesięcy.

COCOMO II - podsumowanie

- Opiera się na rozmiarze oprogramowania mierzonym w KSLOC (np. Function Points, Object Points itp.)
- Kalibracji dokonano na podstawie 161 projektów informatycznych (ale nie polskich)
- Aby dokonać kalibracji należy dysponować danymi historycznymi
- Odpowiednia wielkość oprogramowania

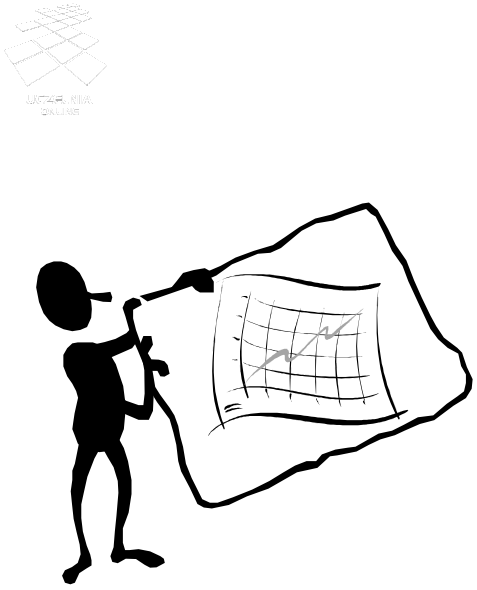
Podsumowując informacje na temat metody **COCOMO II**:

•Jest to metoda bazująca na rozmiarze oprogramowania, mierzonym w tysiącach źródłowych linii kodu. Aby pozyskać tę wartość można wykorzystać metodę punktów funkcyjnych lub specjalnie powstałą na potrzeby COCOMO II metodę Object Points.

•Kalibracji współczynników metody dokonano na podstawie 161 projektów. Niestety nie były to projekty realizowane w Polsce, wiec trudno powiedzieć czy użycie metody w rodzimych projektach jest także skuteczne.

•Problem ten można zniwelować dokonując kalibracji współczynników w oparciu o dane historyczne organizacji. W praktyce jednak, mało która organizacja posiada wystarczającą bazę danych zakończonych projektów.

•Aby metoda była skuteczna, rozmiar oprogramowania powinien być szacowany przynajmniej na kilkanaście tysięcy linii kodu.



Plan wykładu

Wprowadzenie
Metoda delficka
COCOMO II
Use Case Points

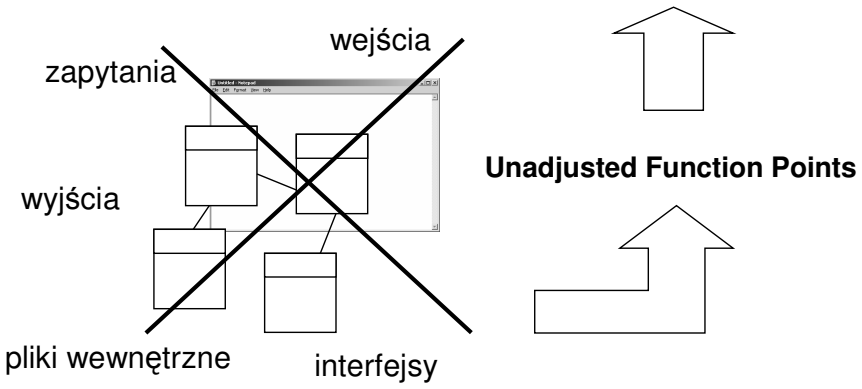
Kolejną ciekawą metodą szacowania pracochłonności, jest metoda punktów przypadków użycia (*ang. Use Case Points*).

Use Case Points

Wyobraźmy sobie sytuację, w której klient zadaje pytanie o czas realizacji projektu na etapie specyfikowania wymagań. Na ich podstawie trudno wyznaczyć rozmiar oprogramowania, ponieważ nie posiadamy jeszcze zdefiniowanej architektury, czy też szkiców ekranów. Należy zatem użyć metody bazującej na wymaganiach systemu. Taką metodą jest metoda punktów przypadków użycia.

Use Case Points – rodowód metody

- Allan Albrecht 1977



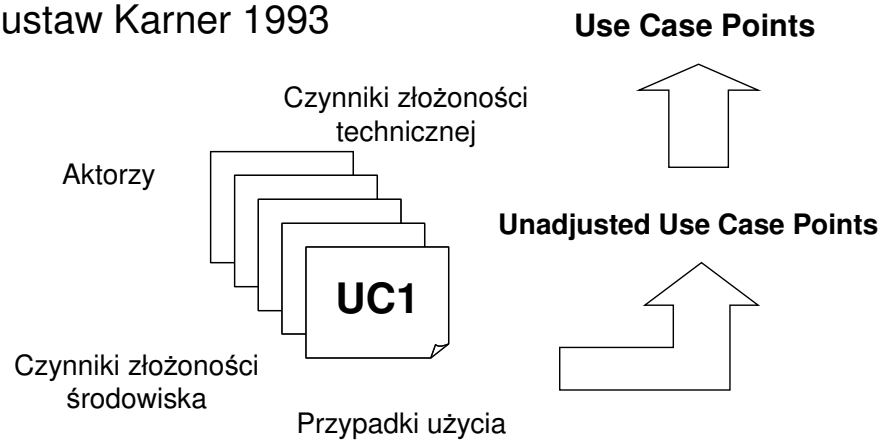
Metoda punktów funkcyjnych

Rodowód **Use Case Points** wywodzi się z innej znanej metody służącej do szacowania rozmiaru oprogramowania, a mianowicie metody punktów funkcyjnych. Metoda zaproponowana przez Allana Albrechta bazuje na projektach ekranów oraz architekturze systemu.

Metoda Use Case Points posiada podobny mechanizm, jednak zrezygnowano z wykorzystania ekranów i architektury.

Use Case Points – rodowód metody

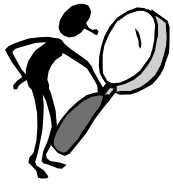
- Gustaw Karner 1993



Metoda punktów przypadków użycia

Twórca metody – **Gustaw Karner** wymienił architekturę i ekrany na specyfikację wymagań zapisaną w postaci przypadków użycia.

Use Case Points – składniki metody



- Czynniki złożoności środowiska
- Czynniki złożoności technicznej
- Aktorzy
- Przypadki użycia (kroki - transakcje)

W metodzie **Use Case Points** podstawowymi artefaktami branymi pod uwagę są:

•Czynniki złożoności środowiska – opisujące w dużej mierze organizację, która wytwarza oprogramowanie,

•Czynniki złożoności technicznej – opisujące własności produktu,

•Aktorzy

•Oraz Przypadki użycia (kroki - transakcje)

Use Case Points – czynniki środowiskowe

Czynniki złożoności środowiska (ECF)

- E1 - Zaznajomienie z projektem *(waga 1,5)*
- E2 - Doświadczenie w tworzeniu aplikacji *(waga 0,5)*
- E3 - Doświadczenie w projektowaniu aplikacji zorientowanych obiektowo *(waga 1,0)*
- E4 – Umiejętności głównego analityka *(waga 0,5)*
- E5 - Motywacja *(waga 1,0)*
- E6 - Stabilność wymagań *(waga 2,0)*
- E7 - Pracownicy pracujący w niepełnym wymiarze *(waga -1)*
- E8 - Trudność języka programowania *(waga -1)*

Czynniki złożoności środowiska (*ang. Environmental Complexity Factors*) charakteryzują zespół/organizacje, która wytwarza oprogramowanie.

Wyszczególniono 8 czynników złożoności środowiska. Każdemu z nich przydzielono także wagę liczbową. Im waga większa tym czynnik bardziej wpływa na zmniejszenie pracochłonności przedsięwzięcia.

Zdefiniowano czynniki takie jak:

•E1 - Zaznajomienie z projektem (waga 1,5) – określa czy zespół jest zaznajomiony z dziedziną problemu i technicznymi aspektami realizowanego projektu . Brane jest tu także pod uwagę zaznajomienie zespołu z metodyką , w ramach której realizowany jest projekt .

•E2 - Doświadczenie w tworzeniu aplikacji (waga 0,5) – ogólnie rozumiane doświadczenie zespołu w wytwarzaniu oprogramowania .

•E3 - Doświadczenie w projektowaniu aplikacji zorientowanych obiektowo (waga 1,0) – określa umiejętność projektowania aplikacji obiektowych , jak i wykorzystanie narzędzi do projektowania , takich jak na przykład język UML .

•E4 – Umiejętności głównego analityka (waga 0,5) – czyli , jego zdolność do pozyskiwania wymagań od klienta oraz zaznajomienie z wiedzą dziedzinową .

•E5 - Motywacja (waga 1,0)

•E6 - Stabilność wymagań (waga 2,0) – bardzo istotny czynnik , określa czy wymagania nie są narażone na częste zmiany .

•E7 - Pracownicy pracujący w niepełnym wymiarze (waga - 1) – określa , czy wśród zespołu znajduje się duża liczba pracowników pracujących w niepełnym wymiarze godzin (na przykład studentów)

•E8 - Trudność języka programowania (waga - 1) – określa jak trudny do opanowania jest język programowania , w którym implementowany będzie system .

Warto zauważyć, że czynniki od E1 do E6 mają dodatnie wagi. Oznacza to, że wzrost ich wpływu redukuje pracochłonność. Natomiast czynniki E7 i E8, posiadają ujemne wagi. Oznacza to, że wraz ze wzrostem ich wpływu zwiększeniu ulegnie także oszacowanie pracochłonności.

Use Case Points – czynniki środowiskowe

- Oceń wpływ każdego czynnika w skali 0-5
- Oblicz ECF (*Environmental Complexity Factor*) korzystając ze wzoru:

$$\mathbf{ECF = 1,4 + (-0,03 * \Sigma w_i * impact_i)}$$

w_i – waga i-tego czynnika środowiskowego,
$impact_i$ – czynnik wpływu

Aby wyznaczyć czynnik złożoności środowiska (*ang. Environmental Complexity Factor*), w skrócie oznaczanego jako **ECF**, należy określić wpływ każdego z ośmiu czynników złożoności środowiska, które zostały wymienione na poprzednim slajdzie, w skali od 0 do 5.

Kolejnym krokiem jest wyliczenie wartości ECF korzystając z podanego wzoru. Warto zauważyć, że w równaniu występuje suma iloczynów wag czynników mnożonych przez czynniki wpływu.

Use Case Points – czynniki techniczne

Czynniki złożoności technicznej

- T1 – System rozproszony *(waga 2,0)*
- T2 - Wydajność *(waga 1,0)*
- T3 – Wydajność dla użytkownika końcowego *(waga 1,0)*
- T4 – Złożone przetwarzanie wewnętrzne *(waga 1,0)*
- T5 – Re-używalność *(waga 1,0)*
- T6 – Łatwość w instalacji *(waga 0,5)*
- T7 – Łatwość użycia *(waga 0,5)*
- T8 - Przenośność *(waga 2,0)*

Czynniki złożoności technicznej określają pewne aspekty tworzonego produktu. Zdefiniowano aż 13 takich czynników. Podobnie jak czynniki środowiskowe, każdemu z nich przypisano wagę oddająca stopień w jakim czynnik wpływa na końcowe oszacowanie.

Wzrost wpływu danego współczynnika złożoności technicznej powoduje zawsze wzrost końcowego oszacowania pracochłonności.

Zdefiniowano następujące czynniki techniczne:

T1 – System rozproszony (waga 2,0) – definiuje czy w systemie wymagane jest rozproszone przetwarzanie danych .

T2 - Wydajność (waga 1,0) – określa wydajność systemu , w odniesieniu do czasu odpowiedzi systemu , szybkości przetwarzania .

T3 – Wydajność dla użytkownika końcowego (waga 1,0) – określa istotność wydajności dla końcowego użytkownika , w kontekście jego percepcji .

T4 – Złożone przetwarzanie wewnętrzne (waga 1,0) – określa czy wymagane są skomplikowane operacje przetwarzania danych , użycie zaawansowanych algorytmów .

T5 – Re-używalność (waga 1,0) – określa czy kod tworzonej aplikacji , będzie w przyszłości wykorzystywany w podobnych projektach .

T6 – Łatwość w instalacji (waga 0,5) – określa sposób instalacji . Czy aplikacja musi być zainstalowana przez specjalistów z firmy wytwarzającej oprogramowanie , czy tez posiadać będzie przyjazny i łatwy w obsłudze instalator .

T7 – Łatwość użycia (waga 0,5) – określa dostosowanie interfejsu użytkownika do jego potrzeb , wygodę w korzystaniu oraz łatwość uczenia .

T8 - Przenośność (waga 2,0) – czy aplikacja powinna działać w różnych środowiskach .

Use Case Points – czynniki techniczne

Czynniki złożoności technicznej (TCF)

- T9 – Łatwość wprowadzania zmian *(waga 1,0)*
- T10 - Współbieżność *(waga 1,0)*
- T11 – Specjalne zabezpieczenia *(waga 1,0)*
- T12 – Zależność od zewnętrznych bibliotek *(waga 1,0)*
- T13 – Dodatkowe szkolenia użytkowników *(waga 1,0)*

•T9 – Łatwość wprowadzania zmian (waga 1,0) – określa czy klient będzie chciał w przyszłości rozbudowywać system . Jeśli tak wymusza to stworzenie odpowiedniej architektury .

•T10 - Współbieżność (waga 1,0) – czy w aplikacji będzie miało miejsce przetwarzanie współbieżne .

•T11 – Specjalne zabezpieczenia (waga 1,0) – określa czy system będzie wymagał wykorzystanie dodatkowych mechanizmów zabezpieczeń .

•T12 – Zależność od zewnętrznych bibliotek (waga 1,0) – w jakim stopniu system bazuje na zewnętrznych bibliotekach . Oczywiście wykorzystanie bibliotek zmniejsza liczbę funkcji , które należy zaimplementować , jednak niesie za sobą pewne problemy z utrzymaniem zgodności , czy też rozbudowy funkcjonalności przez nie udostępniane

•T13 – Dodatkowe szkolenia użytkowników (waga 1,0) – czy będą wymagane dodatkowe szkolenia użytkowników , czy też wystarczy stworzona dokumentacja .

Use Case Points – czynniki techniczne

- Oceń wpływ każdego czynnika w skali 0-5
- Oblicz TCF (*Technical Complexity Factor*) korzystając ze wzoru:

$$\mathbf{TCF = 0{,}6 + (0{,}01 * \Sigma\, w_i * impact_i)}$$

w_i – waga i-tego czynnika technicznego,
$impact_i$ – czynnik wpływu

Aby wyznaczyć czynnik złożoności technicznej (*ang . Technical Complexity Factor*), w skrócie TCF, należy określić wpływ każdego z trzynastu czynników złożoności technicznej, zaprezentowanych na poprzednich slajdach, w skali od 0 do 5.

Kolejnym krokiem jest wyliczenie wartości TCF korzystając z podanego wzoru. Warto zauważyć, że w równaniu występuje suma iloczynów wag czynników mnożonych przez czynniki wpływu.

Use Case Points - aktorzy

Oceniamy złożoność aktora

- Prosty - Aktor, który komunikuje się z systemem przez API
- Średni – aktor, który komunikuje się z systemem przez protokół (np. HTTP, FTP) lub stanowi źródło danych (pliki, baza danych)
- Złożony - aktor komunikujący się z systemem poprzez graficzny interfejs użytkownika

Po określeniu czynników środowiskowych i technicznych można przystąpić do oceny przypadków użycia. Pierwszym etapem jest ocena złożoności aktorów występujących w przypadkach użycia. Wyszczególniono trzy grupy złożoności aktorów:

•Aktor o złożoności prostej który komunikuje się z systemem przez API

•Aktor o złożoności średniej, który komunikuje się z systemem poprzez protokół (np. HTTP, FTP) lub stanowi źródło danych (pliki, baza danych)

•Aktor złożony, komunikujący się z systemem poprzez graficzny interfejs użytkownika

Use Case Points – wagi aktorów

Wyliczamy UAW (*Unadjusted Actors Weight*)

- Zliczamy liczbę aktorów należących do danej kategorii (prosty, średni, złożony)
- Liczbę aktorów w danej kategorii wymnażamy przez współczynnik złożoności z nią związany:
  - Prosty – złożoność 1
  - Średni – złożoność 2
  - Złożony – złożoność 3

$$\mathbf{UAW = \Sigma\, n_i * c_i}$$

n_i – liczność i-tego zbioru złożoności aktorów
c_i – współczynnik złożoności i-tego zbioru

Kolejnym etapem jest wyliczenie **niedostosowanej wagi aktorów** (*ang. Unadjusted Actors Weight*), w skrócie **UAW**. Dokonujemy tego zliczając, w pierwszej kolejności liczbę aktorów zakwalifikowanych do poszczególnych grup. Następnie wymnażamy liczności przez odpowiednie współczynnik złożoności dla każdej z grup, które wynoszą odpowiednio:

•Dla aktorów prostych 1,

•Dla aktorów o średniej złożoności 2,

•Dla aktorów złożonych 3.

Sumując iloczyny liczności przez współczynniki złożoności otrzymujemy współczynnik wag aktorów UAW.

Use Case Points – przypadki użycia

Wyliczamy UUCW (*Unadjusted Use Cases Weight*)

- Należy wyznaczyć transakcje, czyli kroki, lub grupy kroków, które razem stanowią nierozłączną akcje
- W zależności od liczby transakcji przydzielić przypadek użycia do grupy:
  - Prosty – 3 lub mniej transakcji (złożoność 5)
  - Średni – od 4 do 7 transakcji (złożoność 10)
  - Złożony – powyżej 7 transakcji (złożoność 15)

$$UUCW = \sum n_i * c_i$$

n_i – liczność i-tego zbioru złożoności przypadków użycia

c_i – współczynnik złożoności i-tego zbioru

Podobnie do wag aktorów, obliczamy **wagi przypadków użycia** (*ang. Unadjusted Use Cases Weight*) w skrócie **UUCW**. Pierwszą czynnością jest wyznaczenie liczby transakcji w ramach przypadku użycia, czyli kroków lub grup kroków, które mogą zostać wykonane w całości lub w ogóle. Następnie w zależności od liczby transakcji możemy zakwalifikować dany przypadek użycia do jednej z trzech grup złożoności:

•Przypadki proste – jeśli mają 3 lub mniej transakcji (współczynnik złożoności wynosi wówczas 5)

•Średnie – liczba transakcji od 4 do 7 (współczynnik złożoności wynosi wówczas 10)

•Złożone – powyżej 7 transakcji (współczynnik złożoności wynosi wówczas 15)

Należy następnie wymnożyć dla każdej grupy złożoności liczbę zakwalifikowanych do niej przypadków użycia przez współczynniki złożoności. Wagę przypadków użycia UUCW obliczamy sumując powyższe iloczyny.

Use Case Points

- Należy wyliczyć UUCP (*Unadjusted Use Case Points*) sumując wagi aktorów i przypadków użycia

UUCP = UAW + UUCW

- Punkty przypadków użycia (UCP), wyliczamy wymnażając UUCP przez ECF i TCF

UCP = UUCP * TCF * ECF

Po obliczeniu **wag aktorów UAW** i **przypadków użycia UUCW**, możemy obliczyć niedostosowane punkty przypadków użycia (*ang. Unadjusted Use Case Points*) w skrócie **UUCP**, sumując dwie powyższe wartości.

Jednak dopiero wymnożenie **UUCP**, przez czynniki techniczne i środowiskowe umożliwia obliczenie punktów przypadków użycia (*ang. Use Case Points*) **UCP** .

Use Case Points - pracochłonność

20 UCP jaka to pracochłonność ?

- Aby otrzymać pracochłonność należy punkty przypadków użycia przemnożyć przez **czynnik produktywności** PF (przeliczający UCP / roboczogodziny)
- Autor metody podaje PF (*Productivity Factor*) = 20 roboczogodzin
- Inni autorzy podają wartości z przedziału od 15 roboczogodzin do 38 roboczogodzin

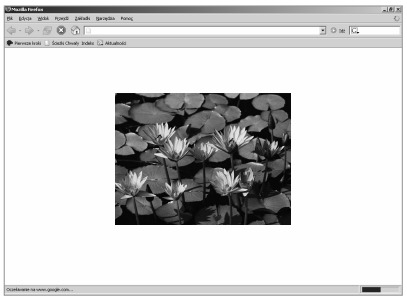
Ostatnim etapem jest przeliczenie punktów przypadków użycia na konkretne oszacowania pracochłonności liczone w osobogodzinach. Dokonujemy tego wymnażając UCP przez współczynnik produktywności (*ang . Productivity Factor*) **PF** , zdefiniowany jako liczba osobogodzin przypadająca na jeden punkt przypadków użycia.

Autor metody - Gustaw Karner zaproponował przelicznik PF równy 20 roboczogodzinom na jeden UCP.

Z wartości podawanych w literaturze wynika, że współczynnik ten może wahać się od **15** do **38** roboczogodzin na jeden punkt przypadków użycia.

Use Case Points - przykład

Załóżmy, że mamy oszacować pracochłonność stworzenia galerii zdjęć, udostępnianych przez Internet.



Spróbujmy przećwiczyć szacowanie za pomocą metody na prostym przykładzie. Oszacujmy pracochłonność stworzenia prostej galerii zdjęć dostępnych przez przeglądarkę internetową.

Use Case Points - przykład

Przyjmijmy, że aplikacja wykonywana jest przez grupę studentów pracujących zgodnie z zaleceniami metodyki TSP, którą znają bardzo dobrze z wykładów wysłuchanych na uczelni.

Studenci stworzyli już wprawdzie kilka aplikacji w czasie studiów, ale jest to ich pierwsza próba implementacji programu, który chcieliby udostępnić szerszemu gronu użytkowników.

Studenci posiadają niezbędną wiedze w zakresie projektowania aplikacji zorientowanych obiektowo. Potrafią wykorzystywać notację UML oraz znają podstawowe wzorce projektowe. Brakuje im tylko odrobiny praktyki.

Przyjmijmy, że aplikacja wykonywana jest przez grupę studentów pracujących zgodnie z zaleceniami metodyki TSP, którą znają bardzo dobrze z wykładów prowadzonych na uczelni.

Studenci stworzyli już wprawdzie kilka aplikacji w czasie studiów, ale jest to ich pierwsza próba implementacji programu, który chcieliby udostępnić szerszemu gronu użytkowników.

Studenci posiadają niezbędną wiedze w zakresie projektowania aplikacji zorientowanych obiektowo. Potrafią wykorzystywać notację UML oraz znają podstawowe wzorce projektowe. Brakuje im tylko odrobiny praktyki.

Use Case Points - przykład

Oprogramowanie tworzone jest przez studentów i dla studentów. Oznacza to, że wymagania precyzować będą sami autorzy programu. O ile bardzo łatwe będzie pozyskanie wymagań, to już na początku prac widać znaczną tendencje do ich ulotności. W każdej chwili może pojawić się zmiana koncepcji, czy zupełnie nowy pomysł.

Studentom nie brakuje oczywiście zapału i chęci do prac.

Niestety trzeba będzie pogodzić rozwój programu z obowiązkami studenta i główne prace wykonywane będą wieczorami.

Studenci wybrali język Java, który znają bardzo dobrze.

Oprogramowanie tworzone jest przez studentów i dla studentów. Oznacza to, że wymagania precyzować będą sami autorzy projektu. O ile bardzo łatwe będzie pozyskanie wymagań, to już na początkowym etapie prac widać znaczną tendencje do ich ulotności. W każdej chwili może pojawić się zmiana koncepcji, czy zupełnie nowy pomysł.

Studentom nie brakuje oczywiście zapału i chęci do prac.

Niestety trzeba będzie pogodzić rozwój programu z obowiązkami studenta i główne prace wykonywane będą wieczorami.

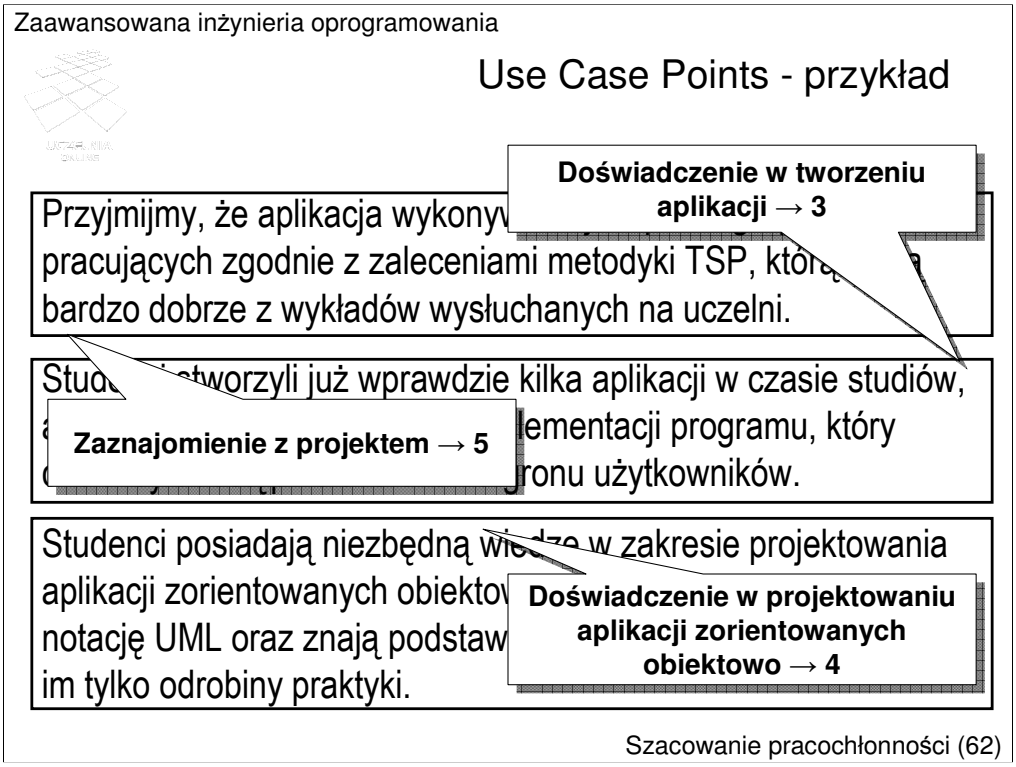
Studenci wybrali język Java, który znają bardzo dobrze.

Use Case Points - przykład

Spróbujmy obliczyć czynnik złożoności środowiskowej.

ECF

Spróbujmy obliczyć czynnik złożoności środowiskowej - ECF.

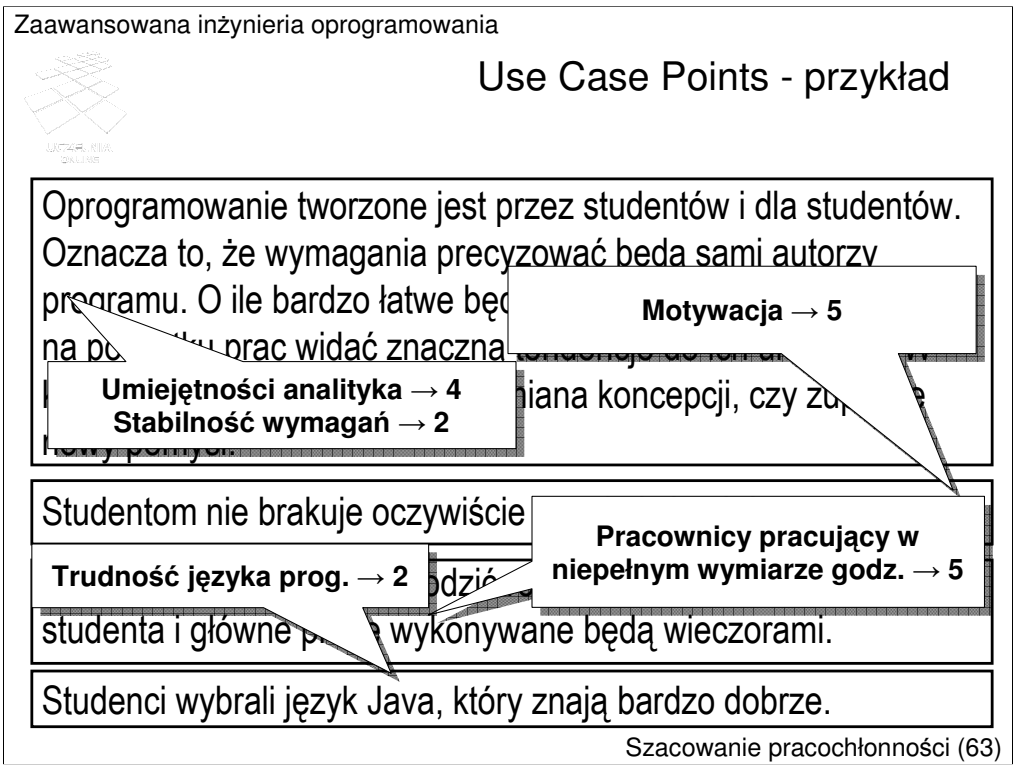


Oszacujmy wpływ poszczególnych czynników złożoności środowiskowej.

Studenci bardzo dobrze rozumieją cel projektu, jak również posiadają bardzo dużą wiedzę na temat metodyki TSP. Możemy zatem ocenić wpływ na poziomie 5.

Jeśli chodzi o doświadczenie w tworzeniu aplikacji, to sytuacja przedstawia się troszkę mniej optymistycznie. Studenci wprawdzie mieli okazje zaimplementować już kilka programów, jednak nie posiadają doświadczenia w zakresie tworzenia aplikacji dla szerszego grona użytkowników. Czynnik wpływu oceńmy na poziomie 3.

Studenci posiadają znaczną wiedze w zakresie projektowania aplikacji zorientowanych obiektowo, jednak ponownie brak im doświadczenia. Wpływ czynnika określmy jako 4.



Oceńmy wpływ kolejnych czynników złożoności środowiska.

Przyjrzyjmy się wymaganiom. Studenci nie mają problemów z pozyskaniem wymagań ponieważ sami są ich źródłem. Znacznym problemem może okazać się brak stabilności wymagań. Oceńmy wpływ obu czynników odpowiednio:

•Umiejętności analityka na 4

•Stabilność wymagań na 2

Studentom z pewnością nie brakuje motywacji, dlatego oceńmy czynnik wpływu na 5.

Niestety wpływ czynnika - Pracownicy pracujący w niepełnym wymiarze, należy ocenić na 5, ponieważ wszystkie osoby wchodzące w skład zespołu pracują w niepełnym wymiarze godzin.

Wybrany język Java, jest dobrze znany przez studentów i tworzenie aplikacji przy jego użyciu nie powinno nastręczyć większych problemów.

Use Case Points - przykład

Wartość ECF:

ECF= 1,4 + (-0,03 *

(1,5*5 + 0,5*3 + 1*4 + 0,5*4 + 1*5 + 2*2 – 1*5 – 1*2)) = **0,89**

$$\mathbf{ECF = 1,4 + (-0,03 * \Sigma w_i * impact_i)}$$

w_i – waga i-tego czynnika środowiskowego,
$impact_i$ – czynnik wpływu

Podstawiając do wzoru otrzymujemy wartość 0,89.

Use Case Points - przykład

Jeśli chodzi o aspekty techniczne tworzonej aplikacji, to dość istotna jest jej wydajność. Jednak nie jest to element kluczowy.

Studenci chcieli by napisać oprogramowanie w taki sposób, by można je w łatwy sposób rozbudowywać.

Autorzy oprogramowania mają także nadzieje, że będą w stanie wykorzystać elementy oprogramowania w przyszłości.

Stworzona aplikacja, będzie w znacznej mierze bazowała na bibliotekach Open Source.

Jeśli chodzi o aspekty techniczne tworzonej aplikacji, to dość istotna jest jej wydajność. Jednak nie jest to element kluczowy.

Studenci chcieliby napisać oprogramowanie w taki sposób, by można je w łatwy sposób rozbudowywać.

Autorzy oprogramowania mają także nadzieje, że będą w stanie wykorzystać elementy oprogramowania w przyszłości.

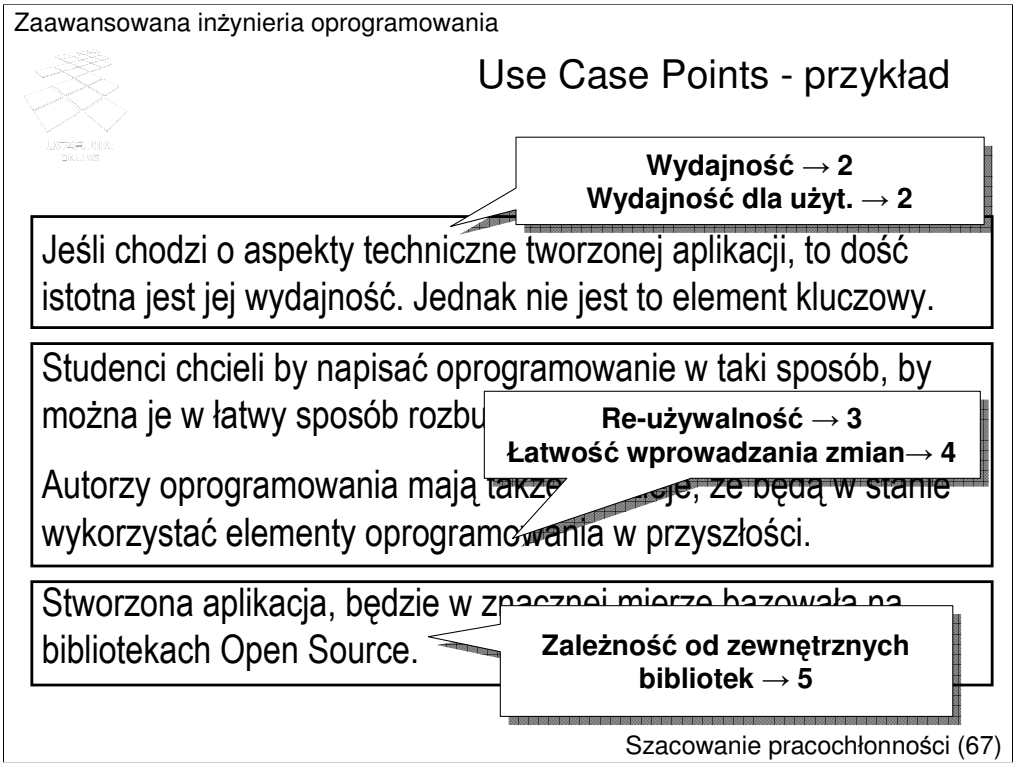
Stworzona aplikacja, będzie w znacznej mierze bazowała na bibliotekach Open Source.

Use Case Points - przykład

Spróbujmy obliczyć czynnik złożoności technicznej.

TCF

Spróbujmy obliczyć czynnik złożoności technicznej - TCF.



Jak wynika z opisu wpływ większości czynników jest znikomy lub zerowy.

Do wymienionych aspektów technicznych, mających wpływ na pracochłonność tworzonego oprogramowania, należy wydajność. Zaznaczono, że nie jest ona bardzo istotna. Oceńmy zatem jej wpływ na wartość 2, zarówno od strony użytych algorytmów, jak i percepcji użytkowników.

Ważnym aspektem jest natomiast łatwość rozbudowy aplikacji i ponowne użycie kodu. Oceńmy wpływ:

•Re-używalności na 3

•Łatwości wprowadzania zmian na 4

Oprogramowanie w dużej mierze bazować będzie na zewnętrznych bibliotekach, dlatego wpływ na pracochłonność oceńmy na poziomie 5.

Use Case Points - przykład

Wartość TCF:

TCF= 0,6 + (0,01 * (1*2 + 1*2 + 1*3 + 1*4 + 1*5)) = **0,76**

$$\mathbf{TCF = 0,6 + (0,01 * \Sigma w_i * impact_i)}$$

w_i – waga i-tego czynnika technicznego,
$impact_i$ – czynnik wpływu

Podstawiając do wzoru otrzymujemy wartość 0,76.

Use Case Points - przykład

Wyszczególniono dwóch aktorów:

- Użytkownik
- Administrator

Aktorzy złożeni

UAW = 0*1 + 0*2 + 2*3 = **6**

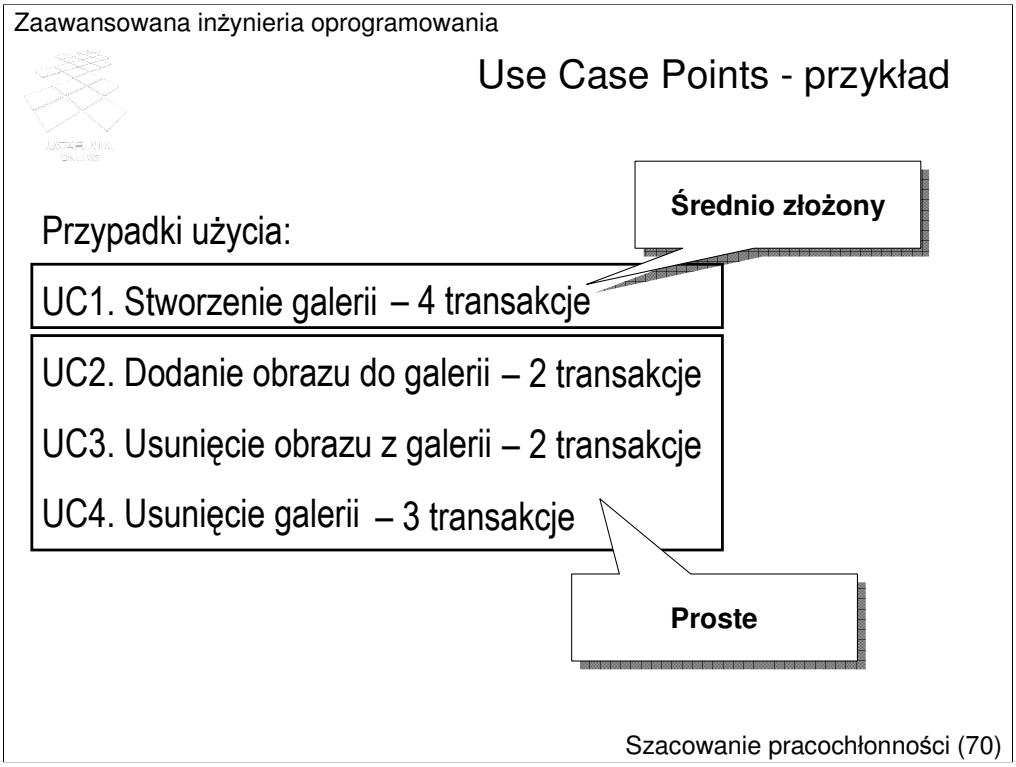
$$\mathbf{UAW} = \sum n_i * c_i$$

n_i – liczność i-tego zbioru złożoności aktorów
c_i – współczynnik złożoności i-tego zbioru

Oceńmy złożoność aktorów i wyliczmy wagi aktorów UAW.

Wyszczególniono dwóch aktorów: Użytkownik i Administrator. Są to aktorzy komunikujący się z systemem za pośrednictwem interfejsu graficznego, dlatego kwalifikujemy ich do grupy aktorów złożonych z wagą 3.

Podstawiając do wzoru otrzymujemy wartość 6.



Wyspecyfikowano także 4 przypadki użycia. Po czym ustalono liczbę transakcji odpowiednio:

UC1. Stworzenie galerii – 4 transakcje

UC2. Dodanie obrazu do galerii – 2 transakcje

UC3. Usunięcie obrazu z galerii – 2 transakcje

UC4. Usunięcie galerii – 3 transakcje

Zatem przypadek UC1 zakwalifikowany został jako średnio złożony, natomiast pozostałe jako proste.

Use Case Points - przykład

Wartość UUCW:

UUCW = 3*5 + 1*10 + 0*15 = **25**

$$\mathbf{UUCW} = \Sigma\, n_i * c_i$$

n_i – liczność i-tego zbioru złożoności przypadków użycia
c_i – współczynnik złożoności i-tego zbioru

Podstawiając do wzoru otrzymujemy wartość 25.

Use Case Points

Wartość UUCP i UCP:

UUCP = 25 + 6 = **31**

Przyjmując PF = 20
Pracochłonność ≈ 419h

UCP = 31 * 0,76 * 0.89 = 20,9684

UUCP = UAW + UUCW

UCP = UUCP * TCF * ECF

Obliczmy wartość punktów przypadków użycia podstawiając wartości do odpowiednich wzorów.

Otrzymaliśmy w przybliżeniu 21 punktów przypadków użycia.

Jeżeli przyjmiemy współczynnik produktywności zaproponowany przez Gustawa Karnera wynoszący 20 UCP na 1 roboczogodzinę, to oszacowana pracochłonność wyniesie w przybliżeniu 419 roboczogodzin.

Use Case Points

Potrafimy już zatem estymować pracochłonność wykorzystując specyfikację wymagań. Co zrobić w przypadku, gdy klient pragnie znać oszacowania jeszcze przed dokładnym sprecyzowaniem wymagań?

Use Case Points – szacowanie do kwadratu

Co jeśli nie ma jeszcze specyfikacji wymagań?

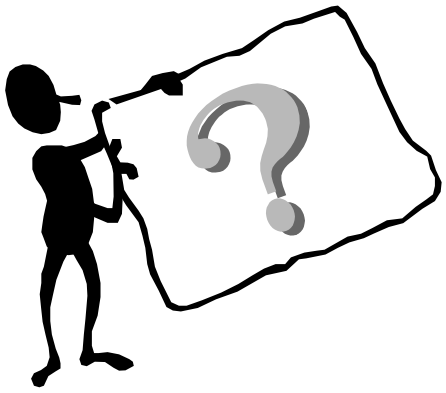
- Możemy spróbować oszacować specyfikacje ...
- Szacując:
  - liczbę przypadków użycia
  - średnią liczbę transakcji
  - liczbę aktorów
  - współczynniki środowiskowe i techniczne
- Możemy oszacować przybliżoną pracochłonność
- UWAGA! Należy wyraźnie zaznaczyć, że szacujemy przy pewnych założeniach!

Można spróbować oszacować specyfikację wymagań, czyli określić orientacyjną liczbę przypadków użycia, ich średnią liczbę transakcji, liczbę aktorów oraz wyznaczyć współczynniki złożoności technicznej i środowiskowej.

Mając takie oszacowania możemy spróbować użyć metody punktów przypadków użycia.

Wówczas należy zaznaczyć, że szacowanie odbyło się przy określonych założeniach. Pozwoli to uniknąć nieporozumień, jeśli rzeczywiste wymagania będą odbiegały w znaczący sposób od oszacowanej przez nas postaci specyfikacji wymagań

Podsumowanie



Wprowadzenie
Metoda delficka
COCOMO II
Use Case Points

Podsumowując. W trakcie wykładu omówiliśmy wykorzystanie metod szacowania w procesie wytwarzania oprogramowania. Przedstawiliśmy także trzy metody szacowania:

- Metodę delficką – bazującą na wiedzy i intuicji ekspertów,
- Metodę COCOMO II – wykorzystującą szacunkowy rozmiar oprogramowania,
- Metodę Use Case Points – umożliwiającą szacowanie na podstawie przypadków użycia.

Dziękuje bardzo za uwagę.

Literatura

- **O. HELMER: *Social Technology. Basic Books*, NY, USA, 1966.**
- **B. W. BOEHM, C. ABTS, A. W. BROWN, S. CHULANI, B. K. CLARK, E. HOROWITZ, R. MADACHY, D. REIFER, B STEECE: Software Cost *Estimation with COCOMO II.* Prentice Hall PTR, 2000.**
- **G. KARNER: *Use Case Points – Resource Estimation for Objectory Projects*. Objective Systems SF, AB, 1993.**

Literatura:

O. HELMER: Social Technology . Basic Books , NY, USA, 1966.

B. W. BOEHM, C. ABTS, A. W. BROWN, S. CHULANI, B. K. CLARK, E. HOROWITZ, R. MADACHY, D. REIFER, B STEECE: Software Cost Estimation with COCOMO II . Prentice Hall PTR, 2000.

G. KARNER: Use Case Points – Resource Estimation for Objectory Projects . Objective Systems SF, AB, 1993.