

Instytut Organizacji i Zarządzania
Politechnika Wrocławska
Raport serii PRE nr 56

**ESTYMACJA NAKŁADÓW PRACY
NA TWORZENIE OPROGRAMOWANIA
PRZY PODEJŚCIU OBIEKTOWYM**

(rozprawa doktorska)

Grażyna Hołodnik – Janczura

Promotor:

dr hab. inż. Jacek Mercik, prof. nadzw. PWr

Słowa kluczowe: model nakładów, rozmiar
programu, miara, regresja,
estymacja

Wydział Informatyki i Zarządzania
Wrocław 1998

SPIS TREŚCI

1. WSTĘP

1. 1. Cel pracy	7
1. 2. Potrzeba budowy nowego modelu szacowania nakładów	8
1. 3. Koszt produktu jako pochodna nakładów	8

2. MIERZENIE W INŻYNIERII OPROGRAMOWANIA

2. 1. Definicja procesu mierzenia	10
2. 2. Aksjomaty poprawności miar programu	12
2. 3. Wymagania i cechy miar	14
2. 4. Miary procesu, produktu i zasobów	18
2. 4. 1. Miary procesu	21
2. 4. 2. Miary programu w technologii strukturalnej	22
2. 4. 2. 1. Długość programu	23
2. 4. 2. 2. Funkcjonalność programu	25
2. 4. 2. 3. Złożoność programu	25
2. 4. 2. 3. 1. Miary złożoności strukturalnej	28
2. 4. 2. 3. 2. Podejście Hendersona – Sellersa do złożoności	30
2. 4. 3. Miary zasobów	31
2. 5. Miary programu obiektowego	32
2. 5. 1. Miary międzymodułowe	33
2. 5. 2. Miary modułowe	36
2. 5. 3. Złożoność strukturalna	39
2. 5. 4. Miary ponownego użycia	39
2. 5. 5. Zestaw metryk Chidambersa i Kemerera	42

3. METODY SZACOWANIA WIELKOŚCI NAKŁADÓW

3. 1. Stosowane podejścia	47
3. 1. 1. Modele algorytmiczne	47
3. 1. 2. Metoda orzeczenia eksperta	49
3. 1. 3. Metoda analogii	49

3. 1. 4. Metoda dekompozycji	50
3. 1. 5. Techniki nieformalne	50
3. 2. Przegląd metod stosowanych w technologii strukturalnej	51
3. 2. 1. Metoda FPA	53
3. 2. 2. Metoda MARK II	57
3. 2. 3. Metoda COCOMO	60
3. 2. 4. Model Pfleegera	65
3. 2. 5. Metoda SLIM	66
3. 2. 6. Metoda COPMO	69
3. 2. 7. Model teoretyczny Halsteada	70
3. 2. 8. Ocena możliwości stosowania przedstawionych metod	71

4. ZAŁOŻENIA DO BUDOWY MODELU NAKŁADÓW

4. 1. Próba adaptacji metody FP	73
4. 2. Przesłanki do ustalenia miar programu obiektowego	80
4. 2. 1. Estymacja nakładów na wczesnym etapie tworzenia	80
4. 2. 2. Etap cyklu życia jako podstawowy wymiar	81
4. 2. 3. Poziom abstrakcji jako kolejny wymiar	81
4. 2. 4. Model etapu analizy w estymacji nakładów	83
4. 2. 5. Wpływ elementów obiektowych na rozmiar programu	84
4. 2. 5. 1. Cechy programu obiektowego i ich wpływ na jego rozmiar	84
4. 2. 5. 2. Cechy procesu obiektowego i ich wpływ na wielkość nakładów	89
4. 2. 6. Rola repozytorium CASE w estymacji nakładów	90
4. 3. Opis procedury ustalania postaci modelu nakładów.....	93
4. 4. Charakterystyki rozmiaru modelu programu.....	95
4. 5. Zestaw potencjalnych zmiennych objaśniających	98

5. CHARAKTERYSTYKA KOLEKCJI PROJEKTÓW

5. 1. Zbiór danych z obserwacji	99
5. 2. Analiza graficzna danych	101

6. KONSTRUKCJA MODELU NAKŁADÓW

6. 1. Badanie charakteru związków między atrybutami rozmiaru programu	106
6. 1. 1. Współczynnik zmienności	106

6. 1. 2. Badanie wyznacznika macierzy współczynników korelacji liniowej	107
6. 1. 3. Weryfikacja hipotezy o wyznaczniku macierzy współczynników	109
6. 2. Test zgodności rozkładu zmiennych z rozkładem normalnym	111
6. 3. Proces doboru optymalnych zmiennych objaśniających	112
6. 3. 1. Metoda analizy macierzy współczynników korelacji	120
6. 3. 2. Metoda taksonomii wrocławskiej	124
6. 3. 3. Metoda selekcji stopniowej	126
6. 4. Estymacja parametrów modelu nakładów	127
6. 5. Weryfikacja modelu nakładów	130
6. 5. 1. Analiza dopasowania modelu do danych z obserwacji	132
6. 5. 2. Badanie wpływu atrybutów rozmiaru na wielkość nakładów	134
6. 5. 3. Analiza składnika losowego modelu nakładów	136
 7. ZAKOŃCZENIE	 146
 8. BIBLIOGRAFIA	 152
 9. ZAŁĄCZNIKI	
9. 1. Opis koncepcji podejścia obiektowego	159
9. 1. 1. Paradygmaty technologii obiektowej	161
9. 1. 2. Definicje podstawowych elementów	164
9. 1. 3. Charakterystyka procesu tworzenia programu	173
9. 2. Porównanie pojęć w metodykach	178
9. 3. Metryki Lorenza i Jacobsona	179
9. 4. Ontologiczna koncepcja systemu informacyjnego	180
9. 5. Przykład obliczania punktów funkcyjnych	184
9. 5. 1. Określenie poziomu aplikacji	184
9. 5. 2. Identyfikacja granicy aplikacji	185
9. 5. 3. Identyfikacja typów danych oraz ustalanie ich złożoności	186
9. 5. 4. Identyfikacja typów funkcyjnych oraz ustalanie ich złożoności	189
9. 5. 5. Obliczanie „niedopasowanych” punktów funkcyjnych	192
9. 5. 6. Obliczanie czynnika dopasowania	192
9. 5. 7. Obliczanie ostatecznej liczby punktów funkcyjnych	194
9. 6. Przykład zastosowania metody COCOMO	195

9. 7. Podejścia do wykonania analizy obiektowej	197
9. 7. 1. Etap analizy w metodyce Jacobsona	198
9. 7. 2. Model przypadków użycia	199
9. 7. 3. Model obiektów dziedzicznych	202
9. 7. 4. Opis interfejsów użytkownika	202
9. 8. Obiekty repozytorium SELECT'a	203
9. 9. Terminy stosowane w metodykach OOSE i SELECT	205
9. 10. Czynniki produktywności w modelu Walstona – Felixa	206
9. 11. Atrybuty nakładów w modelu Bailey'a – Basili'a	207
 10. SPIS TABEL	 208
 11. SPIS RYSUNKÓW	 211

1. WSTĘP

Od czasu, kiedy program komputerowy stał się narzędziem stosowanym przez człowieka prawie w każdej dziedzinie jego działalności, nastąpił znaczny wzrost wymagań dotyczących jakości jego działania, ale i też procesu jego tworzenia. Wymagania rynku producentów oprogramowania powodują autentyczne zainteresowanie tych firm metodami prowadzącymi do usprawnienia tego procesu. Zdarza się bowiem często tak, że zaplanowane terminy nie są dotrzymywane, a budżet przekraczany. Wiele organizacji inwestuje na początku projektu dużo czasu i nakładów w jego oszacowanie. Jednak często nie wpływa to na poprawne określenie terminu jego zakończenia. U wielu producentów oprogramowania zauważa się, że proces realizacji przedsięwzięcia informatycznego jest do tej pory ciągle za mało analizowany i zbyt ogólnie dokumentowany. Bardzo ważne są bowiem, oprócz właściwych metod przewidywania, także dane historyczne z realizacji poprzednich projektów, ich oszacowań oraz danych rzeczywistych, które przeważnie są niedostępne. Taki stan rzeczy znacznie utrudnia stosowanie odpowiednich mechanizmów sterowania jego przebiegiem, a także nie pogłębia wiedzy o zależnościach zachodzących między czynnikami tego procesu. Można oczekiwać, że wymagania jakościowe prowadzące do komputerowego wspomaganie procesu zarządzania przedsięwzięciem informatycznym przyczynią się do zgromadzenia obiektywnej wiedzy o jego przebiegu.

Określenie kosztów i terminów wykonania systemu informatycznego łączy się z jednym z najważniejszych elementów przedsięwzięcia informatycznego, jakim jest zaplanowanie niezbędnych nakładów potrzebnych na jego realizację. O ich wielkości decydują zazwyczaj badania wstępne dotyczące wymagań wobec przyszłego systemu. Wielkość tych nakładów zależy jednak od jego rozmiaru, który można oszacować dopiero po przeprowadzeniu szczegółowych badań. Na ich podstawie możliwe jest dokonanie przybliżonej oceny niezbędnych zasobów czasowych czy też finansowych. W poziomie wiarygodności takiej oceny, jak też i w sposobie zarządzania realizacją takiego przedsięwzięcia można dopatrywać się także warunków jego powodzenia.

Podstawowym czynnikiem nakładów na tworzenie systemu informatycznego jest zatem wielkość tego systemu wynikająca z jego zakresu podobnie, jak dla procesu przemysłowego, jest nim wielkość produkcji określana za pomocą liczby sztuk czy rozmiaru produktu, wyrażonego właściwą dla niego miarą. Dla przedsięwzięcia informatycznego zastosowanie odpowiedniej miary wielkości tworzonego produktu stanowi istotny problem.

Dotyczy on zarówno planowania kosztów jego wykonania, jak i kontroli realizacji całego przedsięwzięcia.

W przypadku systemu informatycznego stosowane są trzy typy miar¹ wielkości: długość, funkcjonalność i złożoność. Brak uniwersalnej miary wielkości produktu informatycznego powoduje wielość rozwiązań i ciągle trwające badania nad poszukiwaniem adekwatnego sposobu mierzenia. Dalszym utrudnieniem jest konieczność pozyskania wystarczających informacji o wielkości planowanego oprogramowania. Zarówno ilość jak i stan posiadanych informacji o powstającym produkcie stanowi istotne ograniczenie dla poszukiwania adekwatnych rozwiązań prowadzących do właściwego oszacowania rozmiarów tego przedsięwzięcia.

Przewidywanie wielkości potrzebnych nakładów i wyznaczanie terminów realizacji systemu informatycznego jest podstawą zarówno opracowania harmonogramu przedsięwzięcia informatycznego jak i możliwości sterowania jego przebiegiem. Rejestrowane zaś, często korygowane na kolejnych etapach tworzenia, wielkości nakładów są źródłem wiedzy dla realizacji podobnych systemów. Analiza tych wielkości oraz badanie przyczyn poziomu ich modyfikacji może w znacznym stopniu przyczynić się do ustalenia zależności występujących między czynnikami procesu tworzenia. Tego rodzaju analizy, prowadzone w przypadku różnego rodzaju działalności gospodarczej, pozwalają na uzyskanie opisu tych zależności za pomocą metod ekonometrycznych. Mimo tego, iż tworzenie programu wymaga specyficznego podejścia, to w celu poszukiwania prawidłowości natury ekonomicznej prowadzi się również badania za pomocą tych samych metod.

Specyfikę tworzenia programu informatycznego podkreślają następujące cechy:

- Tworzenie systemu informatycznego jest tylko częściowo procesem standardowym. *Chociaż zalicza się go do prac inżynierskich, to nie ma tak długiej historii, jak inżynierskie procesy przemysłowe.*
- System informatyczny jest produktem niematerialnym. *Efekt działania jest widoczny na samym końcu procesu tworzenia. Fizyczna miara rozmiaru gotowego programu nie jest wystarczająca do odzwierciedlenia wielkości nakładów potrzebnych do jego otrzymania.*

¹ Miary produktu szczegółowo zostały omówione w rozdziale 2.4 i 2.5.

- Potrzeba zmian.

Częste zmiany zachodzące w funkcjonowaniu przedsiębiorstwa wpływają także na zmianę zidentyfikowanych wcześniej wymagań wobec powstającego systemu.

- Konieczność współpracy pracowników przedsiębiorstwa i twórców systemu.

Brak zainteresowania użytkownika końcowego prowadzonymi pracami analitycznymi, albo ogranicza możliwość tej współpracy, albo prowadzi do ustalenia rzeczywistych wymagań dopiero na etapie wdrożenia lub eksploatacji systemu.

- Konieczność ciągłego szkolenia ludzi tworzących systemy.

Szybkie zmiany w technologii komputerowej i informatycznej wychodzą często poza doświadczenia, zdobyte w trakcie wcześniej wykonanych prac i przez to nie mogą być wykorzystane w nowych przedsięwzięciach.

1. 1. Cel pracy

Celem niniejszej pracy jest znalezienie funkcji estymacji nakładów pracy potrzebnych na tworzenie programu przy podejściu obiektowym, jako produktu o specyficznym charakterze. Funkcji, której postać jest powiązana z mechanizmami natury ekonomicznej, podobnie do innych procesów gospodarczych.

Istnieje wiele różnych metod, które wspomagają menedżera projektu w oszacowaniu wielkości nakładów pracy na tworzenie oprogramowania. Ich praktyczne zastosowanie jest uzależnione od następujących czynników:

- zgodności z technologią tworzenia programu, dla której metoda została opracowana,
- osiągnięcia przynajmniej etapu prac projektowych lub określenia rozmiaru końcowego programu wyrażonego w liniach kodu,
- doświadczenia w korzystaniu z danej metody w celu doboru odpowiednich charakterystyk dla poszczególnych elementów programu.

Aktualnie stosowane metody do szacowania tych nakładów są odpowiednie dla programów powstających w wyniku stosowania metodyk strukturalnych; ich podstawą są wyniki zakończonych prac etapu projektowania bądź kodowania, czyli zbyt późno jak na potrzeby estymacji kosztów tworzenia.

Wobec powyższego zamierza się także znaleźć odpowiedź na następujące pytanie: Czy jest możliwa przybliżona ocena wielkości nakładów pracy na podstawie danych z wczesnego etapu prac projektowych?

1. 2. Potrzeba budowy nowego modelu szacowania nakładów

Do końca lat osiemdziesiątych wiodącym podejściem w tworzeniu oprogramowania było podejście strukturalne. Choć od czasu jego początków minęło ponad 20 lat, to jest ono nadal powszechnie stosowane. Ale pod koniec lat osiemdziesiątych zaczęły pojawiać się pierwsze publikacje na temat formułowania nowej technologii tworzenia oprogramowania, nazywanej technologią obiektową. Zasadnicze zmiany wprowadzone w tym nowym podejściu spowodowały, że metody przewidywania nakładów tworzenia oprogramowania przestały odpowiadać elementom występującym w warunkach technologii obiektowej. Zupełnie nieadekwatne dla tej technologii okazały się te z metod, których mechanizm działania bazuje na empirycznym wnioskowaniu przeprowadzonym w warunkach technologii strukturalnej.

Na podstawie przeglądu literatury zauważono istotny brak formalnego modelu estymacji nakładów dla podejścia obiektowego. Znalaziono jedynie wskazówki, co do możliwości zastosowania proponowanych miar, a także wiele uwag o potrzebie budowy takiego modelu [HEN96, s. 164, FEN96, s. 446, JAC92, s. 472]. W takim modelu nie może zabraknąć charakterystycznych cech programu obiektowego, które wpływają zarówno na jego rozmiar jak i czas potrzebny na jego tworzenie.

1. 3. Koszt produktu jako pochodna nakładów

W analogii do inżynierskich procesów przemysłowych wynik przedsięwzięcia informatycznego jest również nazywany produktem. W celu potwierdzenia słuszności takich porównań prowadzono badania, których efektem jest dokument sporządzony w roku 1986 [SPE86], zawierający analogie między procesem związanym z budową mostu, a procesem tworzenia komputerowego programu. Potwierdzono w nim wiele wspólnych własności obu procesów. To porównanie wykorzystano do analizy pojęcia kosztów tworzenia programu w analogii do kosztów tworzenia produktów przemysłowych. Cały proces związany z tworzeniem wyrobów, który obejmuje zarówno proces przygotowania produkcji, jak i jej wykonania, podobnie jak to jest w przypadku programu, wymaga zaangażowania odpowiednich czynników wytwórczych. Zalicza się go do podstawowej działalności gospodarczej. Jednym z głównych wskaźników tej działalności są koszty, które odzwierciedlają wielkość tego zaangażowania w postaci ponoszonych w tym procesie nakładów [NOW94], [NOW95], [NOW96], [BEG96], [BART74], [BART90], [KLI95], [REK96]. Koszt ten definiuje się jako „... wartość środków gospodarczych zużytych wskutek

wytwarzania przedmiotów lub świadczenia usług” [NOW94, s. 10]. Wartość ta w większości przypadków jest wyrażana kwotą pieniędzy.

Koszt jako wartość zużycia środków gospodarczych, jest wypadkową dwóch czynników

- liczby jednostek zużytych środków gospodarczych,
- jednostkowej ceny zużycia tych środków.

Jego wartość pieniężną oblicza się ze wzoru [NOW94, s. 10]

$$\text{koszt} = \text{zużycie} * \text{cena}. \quad (1.1)$$

Z tego równania wynika, że podstawą kwoty kosztów wyrażonej w pieniądzu jest wielkość zużytych zasobów.

Spośród zaangażowanych w procesie tworzenia produktu zasobów wyróżnia się trzy grupy nakładów [KLI95], [NOW94], [BEG96]:

- pracę żywą,
- środki trwałe,
- środki materiałowe.

Nakłady pracy żywej są miarą pracy ludzkiej, zarówno one, jak i pozostałe dwie grupy, mają swoje ceny, które w pieniądzu przedstawiają ich wartość, ale mają też swoje jednostki naturalne, takie jak godziny pracy, sztuki, kilogramy.

Badania opisywane w niniejszej pracy, związane z zagadnieniem estymacji kosztów tworzenia oprogramowania zostały określone jako estymacja nakładów pracy żywej, czyli grupy nakładów wybranej ze względu na jej podstawową rolę zarówno dla planowania, jak i sterowania przedsięwzięciem informatycznym.

Najważniejsze powody, które zadecydowały o takim podejściu to :

- bezpośredni związek między nakładami pracy żywej i tworzonym produktem;
- możliwość określenia wielkości tego nakładu w niekwestionowanych, naturalnych jednostkach czasu, gdyż w praktyce ceny często się zmieniają;
- potrzeba znalezienia opisu zależności między tymi nakładami i tworzonym oprogramowaniem w celu wykorzystania go w planowaniu i zarządzaniu realizacją przedsięwzięcia informatycznego.

Pojęcie nakładów, występujące w dalszym ciągu niniejszej pracy, odnosi się zatem do nakładów pracy żywej, mierzonych w jednostkach czasu. Nie było rozważane natomiast przewidywanie wielkości pozostałych dwóch grup nakładów, to znaczy wartości środków trwałych i materiałowych.

2. MIERZENIE W INŻYNIERII OPROGRAMOWANIA

Badanie zależności nakładów pracy zużytych na wytworzenia programu od różnych czynników jest nierozłącznie związane z zagadnieniem mierzenia programu. Ze względu na problemy występujące w tym zagadnieniu, przed opisem realizacji procesu estymacji nakładów zapoznano się z podstawami teoretycznymi mierzenia programu.

Każdy proces inżynierski budowlany czy elektryczny byłby trudny do zrealizowania i doprowadzenia do oczekiwanych efektów, gdyby nie stosowano w nim pomiarów. Tworzenie oprogramowania również należy do tego typu procesów, a zagadnienie mierzenia jest nadal nowością, na którą tylko nieliczne firmy produkujące oprogramowanie mogą sobie pozwolić. W większości projektów nie odróżnia się kosztów projektowania od kosztów kodowania czy testowania. Nie prowadzi się też badań kontrolnych w celu określenia, czy nowa technologia jest efektywna, bo do tego niezbędne są pomiary włożonych nakładów i możliwość porównywania ich wyników. Aktualnie dla środowiska obiektowego nie istnieją uniwersalne, zatwierdzone i przetestowane miary [CHI94], [FEN96], [HEN96]. Wielu proponowanym miarom oprogramowania zarzuca się brak podstaw teoretycznych, a niekonsekwencje wymiarowe są szczególnie widoczne w konstrukcji miar złożonych. Istnieje jednak olbrzymia potrzeba mierzenia i kontroli tworzenia oprogramowania. Opis teorii mierzenia został opracowany na podstawie następujących prac [ZUS90], [FEN90 – FEN96], [HEN96], [SHE95].

2. 1. Definicja procesu mierzenia

Mierzenie jest procesem, w którym liczby lub symbole są przydzielane do atrybutów wybranego elementu świata rzeczywistego w sposób wyraźnie zdefiniowany za pomocą odpowiedniej reguły. Proces ten może dotyczyć osoby, przedmiotu lub jakiegoś zdarzenia; choć mówi się o mierzeniu określonej rzeczy, to właściwie mierzona jest istotna dla celu tego pomiaru cecha, nazywana atrybutem.

Miara jest empirycznie i obiektywnie wyznaczoną liczbą lub symbolem, przypisanym do elementu charakteryzowanego specyficznym atrybutem.

Obie definicje pochodzą z reprezentacyjnej teorii mierzenia, która dotyczy zagadnienia sformalizowania intuicyjnego pojmowania zależności między elementami świata rzeczywistego. Zależności te, nazywane relacjami w procesie mierzenia znajdują odwzorowanie w zestawie liczb lub innych umownych znaków. Zatem mierzenie polega na

odwzorowaniu rzeczywistego systemu relacyjnego do numerycznego lub nienumerycznego systemu relacyjnego, który będzie metrycznym i obiektywnym jego odzwierciedleniem. Przykłady takich systemów zawiera tabela 1.

Tabela 1. Empiryczny system relacyjny i odpowiadający mu formalny system relacyjny. Źródło: [ZUS90].

<i>Empiryczny system relacyjny</i>	<i>Formalny system relacyjny</i>
Relacja między drewnianymi stołami	R (liczby rzeczywiste)
Równy lub większy niż: $\bullet \geq$	$\geq$
$b \bullet \geq a$	$\mu(b) \geq \mu(a)$
Binarna operacja $\circ$: konkatencja	$+$
$a \circ b$	$\mu(a \circ b) = \mu(a) + \mu(b)$

Zgodnie z matematyczną definicją mierzenia [FEN96], polega ono na *odwzorowaniu rzeczywistego systemu relacyjnego* D

$$D \equiv (A, R_1, \dots, R_n, O_1, \dots, O_m), \tag{2.1}$$

gdzie:

- A - zbiór obiektów – elementów,
 - $R_1 \dots R_n$ - empiryczne relacje na obiektach - elementach z A (np. większy, mniejszy, równy),
 - $O_1 \dots O_m$ - binarne operacje na obiektach - elementach z A (np. kombinacja),
- do postaci *formalnego systemu relacyjnego* F

$$F \equiv (C, S_1, \dots, S_n, B_1, \dots, B_m), \tag{2.2}$$

w którym:

- C - zbiór elementów (np. liczby rzeczywiste),
- $S_1 \dots S_n$ - formalne relacje na elementach z C (np. $<, >, =$),
- $B_1 \dots B_m$ - binarne operacje na elementach z C (np. $+, -, *$).

To odwzorowanie jest realizowane za pomocą odpowiedniej miary μ , która przekształca empiryczny system D w formalny system F tak, że dla każdego elementu $a \in D$, $\mu(a) \in F$. Co w zapisie formalnym

$$\forall a \in D \exists b \in F, \text{ że } \mu(a) = b \tag{2.3}$$

daje funkcję odwzorowania jako homomorfizm

$$\mu : D \rightarrow F \quad (2.4)$$

Trójka (D, F, μ) jest nazywana skalą, jeśli zachodzą następujące warunki

$$R_i(a_{ij}) \Leftrightarrow S_i(\mu(a_{ik})) \quad (2.5)$$

i

$$\mu(a \circ_j b) = \mu(a) \bullet_j \mu(b) \quad (2.6)$$

dla wszystkich i, j oraz dla wszystkich $a_{ik} \in D$. Jeśli $F=R$, (R jest zbiorem liczb rzeczywistych), to trójka (D, F, μ) jest skalą rzeczywistą.

2. 2. Aksjomaty poprawności miar programu

Dziewięć własności sformułowanych przez Weyukersa [WEY88] stanowi zestaw kryteriów ocen dla proponowanych miar oprogramowania. Pierwsze trzy własności dotyczą wnikliwości i wrażliwości samych miar i nie odzwierciedlają bezpośrednio ani złożoności semantycznej ani syntaktycznej programów, pozostałe zaś dotyczą miar syntaktycznej złożoności programów. Terminologia definicji poszczególnych własności odnosi się do programów, ale wielu badaczy miar oprogramowania odnosi je również do pojęć stosowanych w technologii obiektowej. Zuse [ZUS90] podkreśla brak konsekwencji w definiowaniu własności, np. niektóre aksjomaty są poprawne tylko dla skali proporcjonalnej (własność 9), natomiast inne (własność 6 i 7) nie dotyczą tej skali. Jedna z własności (siódma) nie może być brana pod uwagę wobec nieodpowiedniości dla podejścia obiektowego. Poniżej omówiono dziewięć własności przy zastosowaniu następujących oznaczeń:

P, Q, R – programy,

$\mu(P)$ – miara programu P .

Własność 1. Niepospolitość

$$\exists P \exists Q \mu(P) \neq \mu(Q). \quad (2.7)$$

Własność ta jest zgodna z intuicją, że miara złożoności programu, która ma taką samą wartość dla wszystkich programów nie jest miarą prawdziwą.

Własność 2. Skończona identyczność

Dana jest nieujemna liczba c . Istnieje skończona liczba programów o złożoności c .

Własność wymaga, by była skończona liczba przypadków o tej samej wartości miary. Nie dotyczy ona jednak cyklomatycznej miary złożoności, która może być taka sama dla wielu grafów.

Własność 3. Równoważność

$$\exists P \exists Q \mu(P) = \mu(Q). \quad (2.8)$$

Własność 4. Szczegółowość

$$\exists P \exists Q P \equiv Q \Rightarrow \mu(P) \neq \mu(Q). \quad (2.9)$$

Dane są dwie klasy projektów P i Q , które mają tę samą funkcjonalność, ale nieprawdą jest, że ich miary też muszą mieć identyczną wartość, sposób implementacji decyduje o ich złożoności.

Własność 5. Monotoniczność

$$\forall P \forall Q \mu(P) \leq \mu(P + Q) \text{ i } \mu(Q) \leq \mu(P + Q). \quad (2.10)$$

Złożoność konkatenacji dwóch programów musi być większa od złożoności każdego z nich.

Własność ta nie odpowiada złożoności psychologicznej, dla której miara całego programu jest mniejsza (gdyż wówczas jest lepiej zrozumiały) od miary złożoności jego części.

Własność 6. Nierównoważność interakcji

$$\exists P, \exists Q, \exists R \quad \mu(P) = \mu(Q) \Rightarrow \mu(P + R) \neq \mu(Q + R). \quad (2.11)$$

To oznacza, że interakcja między P i R może być różna niż interakcja między Q i R , która jest wynikiem różnych złożoności dla konkatenacji $P+R$ i $Q+R$.

Własność 7. Nierównoważność permutacji

Są takie dwa ciała programów P i Q , że Q jest utworzony przez permutację wyrazów programu P i wówczas

$$\mu(P) \neq \mu(Q). \quad (2.12)$$

Siódma własność wymaga, by permutacja elementów wewnątrz mierzonego programu mogła zmienić wartość jego miary. Ta własność jest w pełni znacząca w projektach tradycyjnych, gdzie uporządkowanie bloków "if-then-else" może zmienić logikę programu (i w konsekwencji jego złożoność).

W technologii obiektowej klasa jest abstrakcją w przestrzeni problemu i porządek definicji wyrażeń wewnątrz klasy nie ma wpływu na ich wykonanie, np. zmiana kolejności, w której metody są deklarowane nie ma wpływu na porządek, w którym są wykonywane. Z tego powodu Cherniavsky i Smith [CHE91] stwierdzają, że ta własność nie jest przeznaczona dla miar obiektowych.

Własność 8. Niezależność od nazwy

Jeśli program P jest przemianowanym programem Q , to

$$\mu(P) = \mu(Q). \quad (2.13)$$

Zmiana nazwy mierzonego programu nie pociąga za sobą zmiany wartości jego miary.

Własność 9. Złożoność interakcji

$$\exists P \text{ i } \exists Q \text{ taki, że } \mu(P) + \mu(Q) < \mu(P + Q). \quad (2.14)$$

Jeżeli dwie klasy są połączone, to interakcja między nimi może zwiększyć złożoność, a tym samym zwiększyć wartość miary.

Do definicji własności dziewiątej wprowadzono uściślenie [ZUS90]

$$\forall P \forall Q \quad \mu(P) + \mu(Q) \leq \mu(P + Q), \quad (2.15)$$

które nadało tej własności ostrzejsze sformułowanie.

2. 3. Wymagania i cechy miar

W procesie mierzenie należy rozważyć następujące wymagania:

- odpowiedniość: który z numerycznych relacyjnych systemów jest lepszy dla konkretnego odwzorowania,
- możliwość reprezentacji: czy relacyjny rzeczywisty system ma swoją reprezentację w danym numerycznym systemie relacyjnym,
- jednoznaczność: jak wybrać spośród kilku różnych możliwych reprezentacji (kilka skal) w tym samym numerycznym relacyjnym systemie tę odpowiednią.

Jako miary najczęściej przyjmuje się liczby rzeczywiste, a tam, gdzie to jest niemożliwe poszukuje się właściwego podzbioru liczb rzeczywistych, gdyż system numeryczny pozwala na duże możliwości numerycznych manipulacji (których może zabraknąć przy zastosowaniu innych symboli). Problem reprezentacji jest rozwiązywany dla różnych typów relacyjnych systemów przez odwołanie do definicji odwzorowania systemu empirycznego do formalnego systemu relacyjnego (wzory 2.1 i 2.2). Problem jednoznaczności odwzorowania występuje wówczas, kiedy istnieje wiele możliwych reprezentacji. Został on uproszczony przez utworzenie sformalizowanych typów skal. Wówczas, za pomocą dopuszczalnego przekształcenia dla danego typu skali, można znaleźć odwzorowanie tej wielkości w skali innego typu.

Cechy miar

Oprócz wymienionych wymagań w odniesieniu do miar programu sformułowano trzy cechy, które pozwalają stwierdzić ich użyteczność. Proponuje się [HEN96] by miara była poprawna, niezawodna i praktyczna. Wyróżnia się trzy sposoby stwierdzania *poprawności miar*: na podstawie oceny wzrokowej (z wyglądu), wewnętrznej i zewnętrznej zgodności. Po stwierdzeniu poprawności wzrokowej przychodzi się do oceny wewnętrznej zgodności, a następnie zewnętrznej.

Wewnętrzną zgodność miary ocenia się na podstawie analizy wartości mierzonego atrybutu, która powinna w sposób adekwatny przedstawiać różnice występujące w różnych obiektach świata rzeczywistego.

Przykład

Miarą rozmiaru programu jest liczba *SLOC* (atrybutem jest rozmiar). W wyniku porównania dwóch programów *A* i *B*, napisanych w języku *C*, okazało się, że liczba *SLOC* programu *A* jest większa od liczby *SLOC* programu *B*. Sprawdzenie wewnętrznej zgodności tej miary będzie wymagało potwierdzenia, że program *A* jest w rzeczywistości dłuższy niż program *B*.

Wewnętrzna poprawność miary może być badana przez, analizując zawartość, identyfikując powiązane kryteria, analizując podstawy teoretyczne konstrukcji. Poprawność zawartości odnosi się do stwierdzenia, czy miara dobrze określa atrybut. W odniesieniu do miary długości oznacza to ustalenie, czy dobrze odzwierciedla ona ogólnie przyjęte pojęcie atrybutu długości.

Przykład

Czy lepszym odzwierciedleniem długości programu jest liczba *SLOC* czy też pokazanie liczby bajtów potrzebnych do zapamiętania tego programu na nośniku magnetycznym?

Poprawność kryterialna wiąże się z identyfikacją kryterium, które pozwala na określenia, czy miara jest poprawna.

Przykład

Czy miara rozmiaru projektu programu, taka jak liczba oddzielnych modułów na diagramie struktury, może być zastosowana do przewidywania rozmiaru programu mierzonego liczbą funkcji i procedur w kodzie źródłowym? W tym przykładzie miara będzie miała dobrą poprawność przewidywania, jeśli zostanie odkryta zależność między rozmiarem struktury diagramu a rozmiarem programu.

Poprawność konstrukcyjna polega na identyfikacji atrybutów mierzonego obiektu oraz wykazaniu ich zgodności z teorią, na której one bazują. Podstawą teoretyczną nowej miary może być istniejąca, wcześniej sprawdzona miara atrybutu. Należy wówczas wykazać, że nowa miara jest skorelowana z poprzednią. Jednakże nie zawsze korelacja nowej miary z wcześniejszą może być wystarczającym dowodem poprawności konstrukcji.

Kontrola *zewnętrznej poprawności* dotyczy sprawdzenia możliwości uogólnienia jej wyników na inne programy. Podstawowe pytania to: Czy wyniki mierzenia programów napisanych w języku *C* mogą być uogólnieniem dla programów napisanych np. w języku Pascal? Czy wyniki mierzenia mogą być przeniesione poza środowisko, w którym miało miejsce mierzenie?

Niezawodność jest konieczną, ale nie wystarczającą własnością miary. Z niezawodnością wiążą się: stabilność i równoważność. Stabilność wymaga, by miara dawała te same wyniki dla danego obiektu w tym samym środowisku. Jest to trudne do wykazania, bo technika test/retest – osoba biorąca udział w testowaniu może pamiętać odpowiedzi, a mierzenie może spowodować zmiany mierzonego obiektu.

Użyteczność dotyczy aspektu praktycznego, a przede wszystkim kosztów obliczeń. Dodatkowo wymaga się by miary były zrozumiałe, dostarczały wiedzę o tworzonym programie, umożliwiały automatyzację obliczeń, a także były niezależne od języków programowania.

Punkt widzenia

Intuicyjne zrozumienie jest nazywane punktem widzenia. Punkt widzenia jest binarną relacją $\geq$ definiowaną na zbiorze P (zbiór wszystkich możliwych projektów). Dla P, P', P'' muszą być zachowane następujące aksjomaty:

$$P.\geq P' \text{ lub } P'.\geq P \tag{2.16}$$

(kompletność: P jest bardziej złożone niż P' lub P' jest bardziej złożone niż P);

$$P.\geq P', P'.\geq P'' \text{ to } P.\geq P'' \tag{2.17}$$

(przechodniość: jeśli P jest bardziej złożone niż P' i P' jest bardziej złożone niż P'' , to P jest bardziej złożone niż P'').

Analizy statystyczne na miarach

Na miarach o różnych typach skal można wykonywać takie statystyczne analizy, jak np.: wartość średnia, odchylenie standardowe, analiza wariancji. Jednak nie wszystkie analizy są odpowiednie dla każdego typu skali.

Przykład

Zbiór $\{x_1, \dots, x_n\}$ przedstawia miary stopnia trudności dla modułów w systemie X , a zbiór $\{y_1, \dots, y_n\}$ w systemie Y . Należy sprawdzić, który z dwóch systemów ma większą średnią stopnia trudności. Klasyfikacja stopnia trudności jest reprezentowana przez skalę porządkową: bardzo prosty, prosty, średni, złożony, bardzo złożony. W tabeli 2 pokazano w tej skali dwie różne miary M_1 i M_2 .

Tabela 2. Wartości miar stopnia trudności w skali porządkowej. Źródło: [modyfikacja na podstawie FEN96].

Miara	Klasyfikacja stopnia trudności				
	Bardzo prosty	Prosty	Średni	Złożony	Bardzo złożony
M_1	0	1	2	3	4
M_2	0	1	2	3	8

Na podstawie tabeli 3 i 4 wyliczono, że przy zastosowaniu miary M_1 średnia dla systemu X wynosi 2.14, a dla systemu Y wynosi 2.5, natomiast po zastosowaniu miary M_2 średnie odpowiednio wynoszą: 3.28 oraz 2.5. Wartości zestawów średnich dają sprzeczną ocenę stopnia trudności obu systemów; w pierwszym przypadku system Y jest trudniejszy, a

w drugim system *X*. Otrzymane wyniki nie pozwalają na wyciągnięcie poprawnego wniosku. Statystyka polegająca na obliczeniu wartości średniej jest zatem bez znaczenia dla tego typu skali.

Tabela 3. Stopnie trudności modułów w systemie *X*. Źródło: [modyfikacja na podstawie FEN96].

Miary stopnia trudności modułów						
x_1	x_2	x_3	x_4	x_5	x_6	x_7
Bardzo prosty	Średni	Bardzo złożony	Złożony	Prosty	Bardzo złożony	Prosty

Tabela 4. Stopnie trudności poszczególnych modułów w systemie *Y*. Źródło: [modyfikacja na podstawie FEN96].

Miary stopnia trudności modułów							
y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
Złożony	Średni	Średni	Prosty	Złożony	Złożony	Złożony	Złożony

W tabeli 5 przedstawiono najbardziej znane typy skal, ich dopuszczalne przekształcenia oraz znaczące statystyki.

Tabela 5. Typy skal, dopuszczalne przekształcenia i statystyki. Źródło: [FEN96], [HEN96].

Typ skali	Dopuszczalne przekształcenie	Statystyki
Nominalna	Odwzorowanie 1-1	Nieparametryczne, np. dominanta, częstość, mediana
Porządkowa	Funkcja monotonicznie rosnąca	Jak dla nominalnej oraz ranking
Przedziałowa	$g(x)=ax+b \quad (a>b)$	Jak dla porządkowej oraz średnia arytmetyczna, odchylenie standardowe
Proporcjonalna	$g(x)=ax \quad (a>0)$	Nieparametryczne i parametryczne
Absolutna	$g(x)=x$	Nieparametryczne i parametryczne

2. 4. Miary procesu, produktu i zasobów

Opis miar opracowano na podstawie przeglądu literatury na temat mierzenia elementów występujących w procesie tworzenia programu [FEN90 – 96], [ALB79 - 83], [BOE81 - 88], [SOM95], [KAN95], [HEN96], [GAR96], [SHE85 - 95], [SYM88], [VER88 - 92], [VES84], [TIA95], [TAT91], [WAL77], [WEY88], [PUT78 - 79], [PFL91]. Ze względu na aktualnie wyróżniane podstawowe technologie tworzenia programu miary dotyczące programu zostały omówione dla obu technologii: tradycyjnej i obiektowej.

W inżynierii oprogramowania wykonywanie pomiarów wiąże się z identyfikacją obiektów¹ i ich atrybutów, które chcemy zmierzyć. Wyróżnia się trzy klasy obiektów:

- procesy (które są powiązanymi czynnościami),
- produkty (które są wynikiem czynności w procesach),
- zasoby (które są elementami wejść do procesów).

Dla każdej klasy obiektów rozróżnia się atrybuty wewnętrzne i zewnętrzne (tabela 6):

- Atrybuty wewnętrzne to są takie, które mogą być mierzone wyłącznie w terminach związanych z samymi obiektami, w oderwaniu od ich zachowania w środowisku.
- Atrybuty zewnętrzne odzwierciedlają związki tych obiektów z ich środowiskiem.

Atrybuty zewnętrzne są trudne do zmierzenia; definiuje się je przez inne, które są mierzalne, (np. jakość definiuje się za pomocą liczby znalezionych błędów w formalnym testowaniu, gdzie liczba błędów jest wewnętrznym atrybutem procesu). Poszukiwanie relacji między atrybutami zewnętrznymi i wewnętrznymi jest jednym z ważniejszych celów badania procesu mierzenia oprogramowania. Z klasyfikacją wiąże się pojęcie mierzenia bezpośredniego i pośredniego. Z definicji wynika, że wewnętrzny atrybut może być mierzony bezpośrednio. Jednakże nie wyklucza to możliwości pośredniego mierzenia, jeżeli takie będzie odpowiedniejsze. Atrybutów zewnętrznych nie można mierzyć bezpośrednio.

¹ Pojęcia obiekt i atrybut odnoszą się do mierzonych podmiotów oraz ich cech w sensie potocznego rozumienia tych terminów. Nie należy ich łączyć z interpretacją obiektu i atrybutu, które odpowiadają elementom występującym w modelach systemu informacyjnego.

Tabela 6. Przykłady atrybutów dla wyróżnionych klas obiektów. Źródło: [FEN96, s.76].

Klasa obiektów	Atrybut wewnętrzny	Atrybut zewnętrzny
Produkty: <i>Specyfikacje</i>	Rozmiar, modułowość, ponowne użycie, poprawność syntaktyczna, redundacja, funkcjonalność ...	Zrozumiałość, łatwość pielęgnacji ...
<i>Projekty</i>	Rozmiar, modułowość, ponowne użycie, powiązanie, spójność, funkcjonalność ...	Jakość, złożoność, łatwość pielęgnacji ...
<i>Kody programów</i>	Rozmiar, modułowość, ponowne użycie, powiązanie, funkcjonalność, złożoność algorytmiczna ...	Niezawodność, użyteczność, łatwość pielęgnacji ...
Procesy: <i>Tworzenie specyfikacji</i>	Czas, nakład, liczba wymaganych zmian ...	Jakość, koszt stabilności ...
<i>Szczegółowe projektowanie</i>	Czas, nakład, liczba znalezionych błędów specyfikacji ...	Koszt efektywności, koszt ...
<i>Testowanie</i>	Czas, nakład, liczba znalezionych błędów ...	Koszt efektywności, koszt, stabilność ...
Zasoby: <i>Personel</i>	Wiek, stawka godzinowa ...	Wydajność, doświadczenie, inteligencja ...
<i>Zespół</i>	Wielkość, poziom komunikatywności, struktura organizacyjna ...	Wydajność, jakość ...
<i>Oprogramowanie</i>	Cena, wielkość ...	Użyteczność, niezawodność ...
<i>Sprzęt</i>	Cena, szybkość, rozmiar pamięci ...	Niezawodność ...

W literaturze związanej z mierzeniem oprogramowania występują dwa pojęcia: *miara* i *metryka*, oba używane do określenia mierzalnej wartości obiektu ze świata rzeczywistego. Według interpretacji Fentona [FEN91] metryki i miary odróżnia się w następujący sposób:

- metryki charakteryzują proste atrybuty, jak np. długość, liczba decyzji, operatorów czy znalezionych błędów,
- miary są funkcjami metryk, które mogą być zastosowane do oszacowania lub przewidywania bardziej złożonych atrybutów, np. kosztu lub jakości.

Często stosuje się te terminy zamiennie. W pracy przyjęto pojęcie miary w odniesieniu do funkcji nakładów, całego rozmiaru produktu oraz poszczególnych aspektów tego rozmiaru, które przeważnie są funkcjami innych wielkości, np. miara nakładów, miara rozmiaru, miara

złożoności, miara funkcjonalności. W przypadku aspektu długości, kiedy jest mierzony za pomocą pojedynczego atrybutu, przyjęto pojęcie metryki. Również metrykami nazywa się atrybuty elementarnych jednostek obiektów, np. liczbę atrybutów obiektu (elementów modelu systemu informacyjnego), liczbę stron.

Miary pośrednie i bezpośrednie

Z uwagi na złożoność relacji między atrybutami wprowadzono pojęcie bezpośredniego i pośredniego mierzenia (tabela 7). Bezpośrednie mierzenie atrybutu nie zależy od miary innego atrybutu, a pośrednie wymaga odwołania się do miary jednego lub kilku innych atrybutów, a nawet innego obiektu.

Tabela 7. Przykłady miar bezpośrednich i pośrednich stosowanych w inżynierii oprogramowania. Źródło: [opracowanie własne].

Miary bezpośrednie	Atrybuty	Miary pośrednie	Atrybuty
Długość kodu źródłowego	Liczba linii kodu	Produktywność programisty	Liczba linii kodu/miesiąc
Czas trwania procesu testowania	Liczba godzin	Efektywność testu	Liczba pozycji błędnych/całkowita liczba pozycji
Liczba błędów w czasie testowania	Liczba błędów	Zmienność wymagań użytkownika	Liczba nowych wymagań/całkowita liczba wymagań
Liczba wejść	Liczba danych wejściowych	Wskaźnik specjalizacji programu	Liczba podklas/liczba nadklas

2. 4. 1. Miary procesu

Istotnym wskaźnikiem procesu jest czas. Można mówić o procesie analizy, projektowania czy pisania programu, a każdy z tych procesów przebiega w określonym czasie i wymaga odpowiedniego nakładu pracy. Są to przykłady procesów dobrze zdefiniowanych, których spójne działania prowadzą do otrzymania konkretnych produktów: dokumentu specyfikacji wymagań, dokumentacji projektowej, działającego programu. Procesami są też odcinki czasowe, których nie wyznacza otrzymanie produktu, ale np. okres wyznaczony terminem, np. między 10 marca a 30 czerwca 1996 roku dla projektu *X*. W zależności od przyjętej interpretacji procesu różnie będą określane jego miary: wielkość nakładu pracy potrzebna do wykonania projektu (dla procesu zdefiniowanego produktem) lub liczba osób pracujących nad projektem *X* w okresie od 10 marca do 30 czerwca 1996 roku nad projektem *X*.

Miary procesu pełnią szczególną rolę dla efektywności zarządzania przedsięwzięciem informatycznym już od samego jego początku. Zaplanowanie przedsięwzięcia wymaga określenia kosztów jego realizacji oraz terminów czasowych, co byłoby niemożliwe do wykonania bez oceny wielkości jego podstawowej miary, tj. przewidywanej wielkości nakładów pracy. Do przewidywanych miar procesu należą koszty jego realizacji, które ustala się na podstawie modelu estymacji kosztów. Zaplanowane koszty i nakłady pracy są dla menedżera projektu podstawą do prowadzenia kontroli przebiegu realizowanych prac, a także do podejmowania decyzji w sytuacjach krytycznych, szczególnie w dużym projekcie. W tym celu, w czasie prowadzonych prac, przewidywane wielkości porównuje się z rzeczywistymi. Atrybuty zatem są mierzone wielokrotnie, a wyniki tego mierzenia umożliwiają ich analizę i sterowanie nimi, aż do pozytywnego zakończenia.

Bada się zarówno atrybuty procesu wewnętrzne, jak i zewnętrzne, mierzone bezpośrednio. Jest wiele atrybutów zewnętrznych, mierzonych pośrednio, które są kombinacją innych atrybutów, np. efektywność pielęgnacji kodu jest obliczana jako stosunek liczby wykrytych błędów do liczby błędów usuniętych (tabela 8).

2. 4. 2. Miary programu w technologii strukturalnej

Produkty są wynikiem czynności realizowanych w trakcie trwania projektu, powstają w ciągu cyklu życia systemu. Przeważnie są to dokumenty na różnych poziomach szczegółowości, jest nim też prototyp, a produktem końcowym jest działający program. Każdy z produktów, podobnie jak proces, może być mierzony lub jego miara może być przewidywana. Produkty w relacji do procesów są łatwiejsze do zmierzenia za pomocą miar zarówno wewnętrznych, jak i zewnętrznych. Wybór miary zależy od celu mierzenia, np. od badania jakości, rozmiaru czy kosztu. Mierzenie atrybutów zewnętrznych produktu jest możliwe wtedy, gdy jest on zakończony. Za najlepszą cechę produktu, stosowaną do przewidywania wielkości nakładów, uważa się rozmiar produktu. Każdy wytworzony produkt ma fizyczną miarę wielkości – rozmiar, który jest miarą prostą, zrozumiałą i zgodną z zasadami teorii mierzenia. Miara produktu informatycznego sprawia wiele kłopotu, gdyż próbuje się znaleźć uniwersalny atrybut pomiaru wielkości programu, który miałby zastosowanie zarówno w badaniach wielkości potrzebnych nakładów, jak i pojemności na nośniku magnetycznym. W odniesieniu do innych produktów, w zależności od potrzeby, mówi się o objętości, ciężarze, czy długości. Jeszcze trudniej zatem jest mówić o uniwersalnym atrybucie wielkości programu, którego rzeczywisty rozmiar znany jest dopiero na końcu procesu jego wytwarzania (w budowlu znany jest każdy wymiar już w projekcie).

Fenton [FEN96] proponuje spojrzenie na rozmiar programu w różnych aspektach, analogicznie do problemu mierzenia złożonych obiektów w świecie rzeczywistym. Wyróżnia trzy kluczowe aspekty rozmiaru programu:

- długość, która przedstawia rozmiar fizyczny programu,
- funkcjonalność, która pokazuje, co w rzeczywistości użytkownik otrzymuje w wyniku działania tego programu,
- złożoność, która jest różnie definiowana.

Ze względu na to, że miara długości jest fizycznym rozmiarem produktu, różne są miary długości kolejnych produktów pośrednich programu: specyfikacji, projektu i kodu. Natomiast funkcjonalność i złożoność odpowiadają logicznej interpretacji rozmiaru programu, dla których trudno jest wytypować miary, które mogłyby być stosowane na kolejnych etapach tworzenia programu i umożliwiałyby analizy porównawcze.

2. 4. 2. 1. Długość programu

Najczęściej stosowaną miarą długości kodu źródłowego programu jest liczba linii kodu (ang. lines of code, skrót *LOC*). Oblicza się je kilkoma sposobami, ponieważ w kodzie programu są wiersze o różnej zawartości: linie puste, komentarza, deklaracji danych i linie zawierające część instrukcji², jedną całą lub kilka oddzielnych instrukcji programowych. Występują zatem różne miary długości kodu: efektywne linie kodu, czyli bez linii komentarza (ang. effective lines of code, w skrócie *ELOC* lub noncommented line, w skrócie *NCLOC*). Linii komentarza nie można pomijać przy obliczaniu nakładów pracy, dlatego proponuje się miarę całkowitej długości programu

$$LOC = NCLOC + CLOC, \quad (2.18)$$

gdzie *CLOC* (ang. commented lines of code) – linie komentarza programu.

Wydruk programu na etapie testowania może znacznie różnić się długością od wydruku programu po zakończeniu testowania, dlatego wprowadzono dodatkową metrykę - liczbę instrukcji źródłowych napisanych (ang. delivered source instructions, skrót *DSI*) różną od liczby instrukcji źródłowych kodu finałowego. Liczba instrukcji źródłowych nie zawiera linii komentarza, ale uwzględnia deklaracje danych oraz nagłówki.

² Instrukcje źródłowe odpowiadają konstrukcjom syntaktycznym, używanym w językach programowania i nie muszą być równoważne fizycznym liniom kodu.

Liczba znaków

Znaki są podstawową jednostką leksykalną każdego napisanego programu, który następnie w procesie kompilacji jest przetwarzany do postaci binarnej. Jednakże liczba znaków jest zapisywana i prezentowana za pomocą liczby bajtów, która to jest zależna od implementacji sprzętowej. Pojawienie się w drugiej połowie lat 90-tych programowania wizualnego oraz okienkowego środowiska graficznego spowodowało, że przedstawione tradycyjne miary długości programu przestały być istotne dla dołączanych do programów obiektów, dostarczanych przez graficzne środowisko. Na przykład program z ręcznie napisanymi pięcioma wyrażeniami w języku Visual Basic generuje powyżej 200Kb programu wykonywalnego.

Metryka długości Halsteada

Alternatywnym sposobem mierzenia wielkości jest teoretyczna miara programu, sformułowana przez Halsteada [HAL77], ale mocno krytykowana za nieadekwatne i niejasne przedstawienie procedury przewidywania wielkości nakładu. Zaproponowaną miarą wielkości programu jest formuła

$$N = N_1 + N_2, \quad (2.19)$$

w którym:

N_1 - całkowita liczba operatorów,

N_2 - całkowita liczba operandów.

Wielkość programu N jest też nazywana przez autora długością programu.

Długość dokumentacji etapu analizy i projektu

W wyniku etapów analizy oraz projektowania powstają produkty pośrednie, nazywane modelami, które składają się z opisów tekstowych, diagramów i matematycznych symboli. Sposób prezentacji tych produktów zależy od przyjętej metody i stosowanej notacji. W przypadku aspektu długości zliczane obiekty atomowe stanowiły linie kodu, instrukcje źródłowe lub znaki, natomiast dla dokumentacji, zawierającej wymienione produkty, za naturalny obiekt atomowy uważana jest strona. Ponieważ dokumentacja składa się z tekstu i rysunków mierzenie liczby stron daje tylko ogólny pogląd na jej wielkość. Aby otrzymać dokładniejsze przybliżenie rozmiaru tych produktów, stosuje się inne miary, np. miary

specyfikacji De Marco, które jednak nie należą już do miar długości, ale (według autora) do miar funkcjonalności.

2. 4. 2. 2. Funkcjonalność programu

Wielu inżynierów oprogramowania preferuje funkcjonalność, a szczególnie wtedy, gdy ma ona posłużyć za podstawę do wygenerowania wielkości przewidywanych nakładów [FEN96]. Funkcjonalność systemu jest określana za pomocą realizowanego przez ten system przetwarzania informacji, którego opis dostarczany jest w postaci funkcji. Za intuicyjną miarę funkcjonalności uważa się liczbę funkcji wykonanych przez ten produkt [FEN96, s. 258]. Funkcjonalność systemu jest badana na podstawie jego specyfikacji, zatem jako miara jest wykorzystywana do przewidywania nakładów oraz mierzenia produktywności. Wówczas wymaga się spełnienia następującego założenia: jeśli program P jest implementacją specyfikacji S , to P i S mają tę samą funkcjonalność.

Podejście DeMarco

Do mierzenia funkcjonalności zastosowano metryki, które odzwierciedlają elementy składowe modelu struktury informacyjnej oraz struktury funkcjonalnej, zgodnie z podejściem przyjętym w technologii strukturalnej [MAR82]. Definiuje się różne obiekty atomowe dla poszczególnych modeli, które zależą od typu diagramu i rodzaju stosowanych symboli. Obiektami atomowymi dla diagramu przepływu danych są: procesy, obiekty zewnętrzne, zbiorniki danych i przepływy danych. Dla diagramu związków encji obiektami atomowymi są encje i związki. Dla algebraicznych specyfikacji atomowymi elementami są: sortowania, operacje i aksjomaty. Chociaż to podejście zostało zakwalifikowane do miar funkcjonalności, to przez wykorzystywanie metryk, które charakteryzują złożoność strukturalną może być również miarą złożoności strukturalnej.

2. 4. 2. 3. Złożoność programu

Pojęcie złożoności programu³ jest najtrudniejszym elementem rozważań na temat rozmiaru programu. Świadczy o tym chociażby wiele różnych definicji tego pojęcia, przez wielu autorów różnie formułowanych. Złożoność programu jest przeważnie definiowana w zależności od celu jej badania. Jednak, mimo braku jednoznacznej definicji spełnia ona jeden

³ Pojęcie złożoności obliczeniowej posiada precyzyjnie określoną definicję [MIK86], [ZAK90], [HAR92], [HOP94], ale nie stanowi ono przedmiotu rozważań niniejszej pracy.

z podstawowych warunków adekwatności poszukiwanej miary rozmiaru programu, gdyż odpowiada intuicyjnemu wyobrażeniu o wielkości programu. Słownikowa definicja złożoności⁴ wyraża dwoisty charakter tego pojęcia. Z jednej strony określa ona strukturę (elementy składowe) badanej wielkości, a z drugiej wskazuje właśnie na stopień jej skomplikowania.

Estymacja nakładów przeznaczonych na tworzenie programów (rozważana w niniejszej pracy) nie dotyczy programów, dla których wybór algorytmu jest związany z badaniami natury podstawowej, w których poszukuje się odpowiedzi na pytanie o możliwość rozwiązania problemu za pomocą programu komputerowego. Dotyczy tych problemów, dla których algorytmy istnieją, są to problemy związane z przetwarzaniem dużej ilości różnego typu danych, które wymagają ochrony i bezpieczeństwa. Ze względu na stosowaną w nich technikę realizacji nazywane są systemami przetwarzania transakcyjnego⁵. Występujące w nich problemy obliczeniowe stanowią od 5 do 25 % realizowanych przez nie funkcji [BAR92]. Zagadnienie szacowania nakładów tworzenia takich programów jest związane z pojęciem złożoności interpretowanej zgodnie z podejściem Albrechta, Symonsa, Fentona, Henderson-Selllorsa, w których wymaga się odpowiedzi na pytanie, ile czasu pracy potrzebuje zespół informatyków na wykonanie programu.

Metryka McCabe

Jednym ze sposobów mierzenia złożoności programu podejmowane w celu ustalenia jego stopnia trudności to miara złożoności strukturalnej McCabe [CAB76] (również aktualnie stosowana) oraz miara długości Halsteada [HAL77], przez niektórych autorów [HEN96, s. 57] zaliczana też do miar złożoności. Według McCabe złożoność strukturalna programu może być mierzona tzw. cyklomatyczną liczbą grafu przepływu sterowania. Dla programu z grafem G liczba cyklomatyczna jest obliczana jako

$$v(G) = e - n + 2, \quad (2.20)$$

gdzie G ma e krawędzi i n węzłów. Uproszczona formuła obliczania $v(G)$ brzmi:

$$v(G) = d + 1, \quad (2.21)$$

w której d – liczba węzłów decyzyjnych.

⁴ Definicja złożoności według słownika języka polskiego: „... to, że się coś składa z wielu elementów, to, że coś ma budowę skomplikowaną; bycie skomplikowanym, zawiłym, nieprostym” [SŁO81], s. 1033]

⁵ Transakcja jest jednostką przetwarzania i wymiany informacji między lokalnym i zdalnym programem, który wykonuje jakieś działanie [BER92]. Charakteryzuje się następującymi własnościami [DAT86]: niepodzielność, spójność, izolacja, trwałość. Głównym celem technologii z przetwarzaniem transakcyjnym jest zapewnienie

Na rysunku 1 przedstawiono graf odpowiadający przykładowej procedurze, która składa się z wyrażień sekwencji i selekcji.

Przykład

A *Input (dana);*

B *If dana < 0*

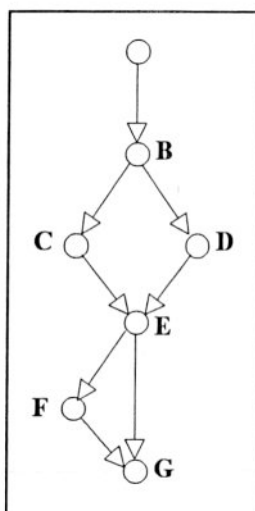
C *then print ('błąd')*

D *else print ('idź dalej');*

E *If dana > 100*

F *then print ('nadmiar');*

G *End*



Rysunek 1. Graf przepływu sterowania dla procedury z przykładu. Ze wzoru (2.20) wyliczona wartość liczby cykloatycznej wynosi $v(G) = 8 - 7 + 2 = 3$. Źródło: [FEN96, s. 298].

Cyklomatyczna liczba jest też stosowana jako wskaźnik badania poziomu skomplikowania programu lub modułu dla procesu testowania i pielęgnacji produktu. Podaje się na podstawie empirycznych ustaleń, że wartość v , która przekracza liczbę 10, może oznaczać wystąpienie w tych procesach istotnych problemów.

Podejście Albrechta i Symonsa

Złożoność programu jest określana na podstawie specyfikacji wymagań i mierzona za pomocą punktów funkcyjnych. Określenie liczby punktów funkcyjnych wynika z przyporządkowania odpowiedniej wagi każdemu z wyspecyfikowanych typów funkcji.

spójności danych oraz wysokiej wydajności systemów informatycznych pracujących w środowisku wielodostępnym.

Wyróżnia się dwa rodzaje złożoności: złożoność funkcjonalną i techniczną. Złożoność funkcjonalna określana jest za pomocą liczby logicznych typów plików, do których odwołuje się każda z funkcji. Złożoność techniczna jest określana za pomocą określonego zestawu charakterystyk. W metodzie Albrechta występuje 14 czynników charakteryzujących aplikację, natomiast w metodzie Symonsa jest ich 20. Sumaryczny poziom udziału tych czynników dla konkretnej aplikacji, ustalony na podstawie empirycznie wyznaczonej skali wpływu, stanowi miarę jej złożoności. Szczegółowy opis metody FP znajduje się w punkcie 3.2.

2. 4. 2. 3. 1. Miary złożoności strukturalnej

Najczęściej spotyka się w literaturze miary złożoności strukturalnej (jako wewnętrzny atrybut programu), które są stosowane do estymacji charakterystyk zewnętrznych. Dzielią się one na wewnątrz modułowe, nazywane też modułowymi i międzymodułowe. Stosowany podział wskazuje, że są to miary odnoszące się do etapu projektowania, czyli dokumentacji projektowej programu, do etapu implementacji, lub też etapu pielęgnacji. Pojęcie modułu⁶ wynikające z definicji w technologii strukturalnej występuje dopiero od etapu projektowania.

Card i Glass [CAR90] przyjęli, że całkowita złożoność systemu C_s jest kombinacją złożoności międzymodułowej i modułowej

$$C_s = C_I + C_M, \quad (2.22)$$

gdzie:

C_I - złożoność międzymodułowa, C_M - złożoność modułowa.

Złożoność międzymodułowa jest równa liczbie powiązań w postaci wywołań podprogramów za pomocą instrukcji *CALL* lub liczbie przekazywanych komunikatów. Modułowa złożoność natomiast odpowiada złożoności danych, która jest przedstawiana jako liczba zmiennych wchodzących do modułu lub wychodzących.

⁶ Pojęcie modułu występuje zarówno w technologii strukturalnej jak i obiektowej. Definicja modułu według Yourdona [YOU79]: „moduł jest sekwencją bliskich wyrażeń programu, ograniczonego przez granicę elementów, które posiadają identyfikator zagregowania”. Definicja modułu według Rumbaugh [RUM91, s. 43]: „Moduł jest logiczną konstrukcją, umożliwiającą grupowanie klas, asocjacji i generalizacji. Moduł obejmuje jedną perspektywę lub przekrój sytuacji. Na przykład: moduł elektryczny, hydrauliczny i wentylacyjny są

Dla systemów obiektowych dodano [CAR90] trzeci element, spójność semantyczną, jako wydzieloną część złożoności modułowej. Wówczas równanie ma postać

$$C_s = C_I + C_M + (C_{h_{\max}} - C_h), \quad (2.23)$$

gdzie:

C_h - spójność semantyczna, która jest częścią C_M ,

$C_{h_{\max}}$ - maksymalna, możliwa spójność dla modułu,

biorąc pod uwagę zminimalizowane połączenia danych w systemach obiektowych, taka notacja złożoności danych na poziomie między modułowym jest redundantna. Dla porównania złożoności systemów wprowadzono relatywną miarę złożoności

$$\overline{C_s} = \frac{C_s}{n}, \quad (2.24)$$

gdzie n jest liczbą modułów w systemie. Wartość ta daje średnią złożoność dla modułu w systemie.

Miary międzymodułowe

Miary międzymodułowe uwzględniają powiązania między modułami, które mogą być skierowane do wewnątrz modułu lub na zewnątrz, stąd istnieją dwa rodzaje powiązań:

- fan – in, odpowiada liczbie miejsc, z których sterowanie jest przekazywane do modułu plus liczba danych,
- fan – out, odpowiada liczbie wymaganych innych modułów plus liczba struktur danych, które są aktualizowane przez badany moduł.

Powiązania te są określane na etapie analizy i projektowania. Card i Glass uwzględniają tylko liczbę powiązań fan – out, które po podstawieniu do równania

$$\overline{C_I} = \frac{C_I}{n} = \frac{\sum^n (fan-out)^2}{n} \quad (2.25)$$

pozwalają na wyliczenie średniego rozkładu powiązań.

Inną miarę przepływu informacji między modułami przyjęli Henry i Kafura [HEN81] w postaci trzech wyrażeń

różnymi przekrojami budownictwa. Granice modułu są ustalane arbitralnie i są przedmiotem opinii”. W metodyce OOSE przy podejściu Jacobsona odpowiednikiem modułu jest blok.

$$fan - in * fan - out, \quad (2.26)$$

$$(fan - in * fan - out)^2, \quad (2.27)$$

$$S_s * (fan - in * fan - out)^2, \quad (2.28)$$

gdzie S_s jest rozmiarem modułu podanego w liniach kodu.

Miary modułowe

Wyszczególniono pięć typów miar modułowych: styl (kształt), rozmiar, struktura danych, struktura logiczna i spójność wewnętrzna. Przy analizie kosztów najczęściej są uwzględniane miary rozmiaru i struktury logicznej. Miary stylu dotyczą techniki pisania komentarzy: stosowanych konwencji nazewnictwa, sposobów ukrywania i identyfikacji. Badania stylu są podobne do problemu mierzenia dokumentacji. Miary struktury danych dotyczą liczby zmiennych lokalnych i ich zakresu, który jest istotny dla zrozumienia i pielęgnacji kodu. Notacja życia zmiennych i zakresu wymaga znacznych nakładów ponoszonych na ich uzyskanie [CON86].

2. 4. 2. 3. 2. Podejście Hendersona – Sellersa do złożoności

Strukturalna złożoność według Hendersona - Sellersa stanowi jedną z trzech części psychologicznej złożoności i jest związana ze strukturą dokumentu. Inaczej niż Fenton postrzega on pojęcie złożoności programu. Klasyfikuje złożoność programu najpierw ze względu na zewnętrzne charakterystyki programu, a następnie identyfikuje wewnętrzne charakterystyki, nazywając je obiektywnymi miarami [HEN96, s.51]. Złożoność programu dzieli na trzy składniki: złożoność problemu, psychologiczną i reprezentacyjną. Na podstawie tej klasyfikacji Henderson – Sellers zaproponował funkcję obiektywnej miary złożoności w postaci:

$$X = X(H, P, S, D, L, C_h, C_s, C), \quad (2.29)$$

gdzie:

H – czynnik ludzkiego pojmowania,

P – złożoność problemu,

S – obiektywne miary wraz z rozmiarem,

D – struktura danych,

L – logiczna struktura,

C_h - spójność wewnętrzną,

C_s - spójność semantyczna,

C – powiązania.

Dla każdego z etapów cyklu życia (dla $i = 1$ do 6), wychodząc z ogólnej postaci równania (2.29) zbudowano podobne, uwzględniając w każdym z nich możliwe do uzyskania na tym etapie składniki. W ten sposób otrzymano sześć następujących równań:

1) etap analizy

$$X_1 = X_1(H, P, C_s, C), \quad (2.30)$$

2) etap projektowania konceptualnego

$$X_2 = X_2(H, P, D, L, C_h, C_s, C), \quad (2.31)$$

3) etap projektowania komponentowego

$$X_3 = X_3(H, P, S, D, L, C_h, C_s, C), \quad (2.32)$$

4) etap kodowania

$$X_4 = X_4(S, D, L, C_h, C_s, C), \quad (2.33)$$

5) etap testowania

$$X_5 = X_5(H, S, D, L, C_h, C_s, C), \quad (2.34)$$

6) etap implementacji

$$X_6 = X_6(H, P, S, D, L, C_h, C_s, C). \quad (2.35)$$

2. 4. 3. Miary zasobów

Do zasobów w realizacji przedsięwzięcia informatycznego zalicza się: personel indywidualny lub zespół, materiały łącznie z usługami, narzędzia programowe i sprzętowe i metody. Sposób mierzenia zasobów ma na celu określenie ich wielkości, np. ile osób w zespole powinno pracować nad projektem, ile będą kosztowały narzędzia, jaka powinna być jakość narzędzi, a także jakie kwalifikacje powinien posiadać personel. Wielkość zasobów ma zasadniczy wpływ na koszt całego przedsięwzięcia, ponieważ zasoby stanowią o wielkości kosztów pochodzących ze składników wejściowych procesu i są odzwierciedlone w kosztach

wyjściowych, czyli w kosztach otrzymanego produktu. Przez zaangażowanie poszczególnych zasobów w procesie tworzenia produktu zachodzi widoczny związek między procesem (składnikami wejściowymi) a produktem (składnikiem wyjściowym).

Koszt zasobów jest definiowany przez wyróżnione składniki, a jego wartość jest zależna od ich ceny rynkowej.

Produktywność

W przypadku zasobów, do których należy personel ważnym atrybutem jest jego produktywność. Jest to atrybut zewnętrzny, gdyż zależy od realizowanego procesu. Jest obliczany jako

$$P = \frac{S}{E}, \quad (2.36)$$

gdzie:

S - wielkość wyjścia,

E - wielkość włożonego nakładu.

Produktywność jest to miarą wynikającą z ekonomicznego punktu widzenia procesu. Dla tworzenia programu miarą wyjścia jest zazwyczaj liczba linii wykonanego kodu traktowanego jako końcowy produkt, a miarą wejścia jest liczba osobo - miesięcy wykorzystana do specyfikacji, projektowania, kodowania i testowania programu. Analogia ekonomiczna jest jednak niewystarczająca dla oprogramowania, ponieważ wielkość wyjścia programowego nie jest powiązana w ten sam sposób z wejściem, jak to jest w przemyśle. Procesy przemysłowe są powtarzalne, a program przeważnie jest tworzony jednostkowo, a nie seryjnie. Dlatego przy obliczaniu produktywności zespołu należy uwzględnić dodatkowe czynniki, np. etap cyklu życia, procent prac badawczych. Na proces i produkt mają też wpływ inne atrybuty zespołu, które mogą być mierzone: indywidualnie (doświadczenie, wiek, inteligencja oraz zespołowo: rozmiar, struktura i komunikatywność).

2. 5. Miary programu obiektowego

W odniesieniu do programów tworzonych przy podejściu obiektowym trudno jest mówić o rozmiarze programu podobnie, jak przy podejściu strukturalnym. Liczba linii kodu źródłowego programu jest znana po jego wytworzeniu i jest równie zależna od języka implementacji, np. metoda jest uznana za małą w Smalltalku, kiedy zawiera około 8 linii kodu, natomiast w C++ około 24 linii (tabela 8).

Tabela 8. Wyniki porównania liczby linii kodu. Źródło: [na podstawie literatury i doświadczeń projektowych]

Rodzaj porównania	Nazwa języka			
	Smalltalk	C++	OO Cobol	Jawa
Liczba linii kodu	8	24	16	24

Jednakże w dziedzinie mierzenia produktów obiektowych nie istnieją tak dobrze opisane i sprawdzone metody, jak w przypadku technologii tradycyjnej. W literaturze spotyka się często stwierdzenia o braku takich metod. Poniżej zostaną omówione metryki należące do miar międzymodułowych, a następnie modułowych.

2. 5. 1. Miary międzymodułowe

Przyjmuje się, że klasy są autonomicznymi modułami. Dlatego powiązania międzymodułowe są mierzone liczbą relacji między klasami lub też między podsystemami. W technologii obiektowej, jak opisano w załączniku 1, wyróżnia się trzy podstawowe rodzaje powiązań: asocjacja, agregacja i dziedziczenie. Mimo istotnych różnic w odniesieniu do tych rodzajów, powszechnie używa się wspólnego określenia "powiązanie". Niektórzy autorzy metryk stosują pojęcie powiązania tylko w odniesieniu do pierwszych dwóch rodzajów powiązań, a niektórzy do wszystkich trzech. W konsekwencji powoduje to dwuznaczność rozumienia proponowanych definicji miar. Na przykład Chidamber i Kemerer [CHI94] definiują powiązanie między obiektami jako połączenie jednej klasy z innymi, mierzone ich liczbą *CBO* (ang. coupling between objects). Definicja z 1991 określała istnienie powiązania tylko wtedy, gdy dwa obiekty oddziaływały na siebie. Wówczas dziedziczenie, które działa tylko w jedną stronę, nie jest powiązaniem według tej definicji. Jednak w definicji metryki powiązań z 1994 uwzględniono związki dziedziczenia. Z jeszcze inną definicją powiązania łączą się metryki Li i Henry [LIH93], w których pojęcie „powiązanie” odnosi się do abstrakcyjnego typu danych (ang. Abstract Data Type, skrót *ADT*). Wówczas powiązanie abstrakcyjnych danych (ang. Data Abstarction Coupling, skrót *DAC*) wyraża się liczbą nieprostych atrybutów różnych typów definiowanych w klasie

$$DAC = \text{liczba danych } ADT. \quad (2.37)$$

Stosuje się zatem kilka różnych zasad mierzenia powiązań, które zostaną omówione na przykładzie wybranych metryk: „fan – in” oraz „fan – out”, *RFC*, *CBO*, *MPC*. Metryki powiązań „fan – in” oraz „fan – out” występują również przy podejściu strukturalnym (opisane punkcie 2.4.) Wartość tych metryk, jak pokazano na rysunku 2, dla każdej pary klas

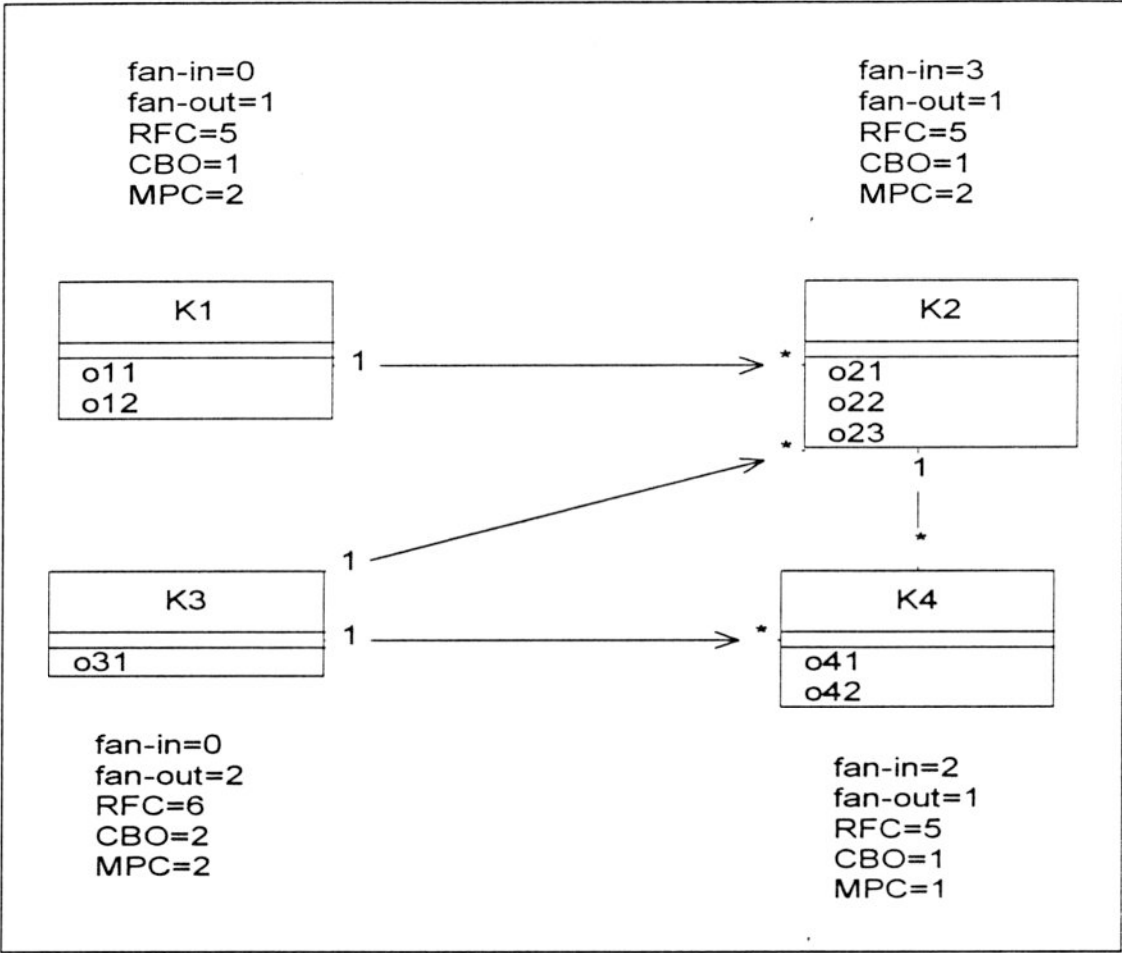
może mieć wartość zero lub jeden. Obliczone wartości metryk *RFC* i *CBO* również zamieszczono dla porównania w tym samym przykładzie (rysunek 2), zaś ich definicje znajdują się w następnym punkcie tego rozdziału, wraz z całym zestawem metryk Chidambersa i Kemerera. Metryka *MPC* (ang. Message - Passing Coupling) jest liczbą przesyłanych wyrażań, które zostały zdefiniowane w danej klasie. Powiązaniem dla tej metryki jest połączenie potrzebne do przekazywania komunikatów. Różnica między metryką „fan - out” a *MPC* jest taka, że „fan - out” jest liczbą połączeń (w jedną stronę) między parą klas, natomiast *MPC* uwzględnia każde przesłanie oddzielnie, np. jeżeli dwie różne metody w klasie *A* mają dostęp do tej samej metody w klasie *B*, to *MPC*=2, a liczba „fan - out” jest równa 1 (przykład na rys. 2: klasy *K1* i *K2*).

Można mierzyć również *jakość powiązania*. Lorenz i Kidd proponują wyliczać średni poziom klasy *PPM*, jako liczbę parametrów przypadających na metodę [LOR94]. Dobry projekt powinien przekazywać obiekty jako parametry (lub argumenty), a wartość *PPM* powyżej 0,7, wykazana empirycznie, wskazuje na właściwy poziom tego projektu.

W pracy [HAY94] przedstawiono sugestię, otrzymaną w wyniku analizy statystycznej, że zachodzi liniowa relacja między powiązaniem klasy i jej rozmiarem. Dla kilku systemów zaprogramowanych w języku Smalltalk otrzymano równanie w postaci

$$\text{powiązanie} = (0,03 \pm 0,0006) * SLOC + 5,5 \pm 1,7. \quad (2.38)$$

Równanie (2.38) zostało wyprowadzone na podstawie systemów wykonanych w jednym języku, a ponieważ odwołuje się do rozmiaru klasy mierzonego za pomocą liczby linii kodu źródłowego (*SLOC*), to wyraża zależność charakterystyczną tylko dla tego języka. Korzystanie z tego równania uzależnione jest od dalszych badań dotyczących: niezależnej walidacji, systemów o różniącej się liczbie klas i wartościach powiązań oraz systemów programowanych w innych językach niż Smalltalk, np. Eiffel i C++.



Rysunek 2. Przykład modelu klas i wartości metryk RFC i CBO. Źródło: [opracowanie własne].

Metryki struktury dziedziczenia jako szczególnego rodzaju powiązanie

Rozważania dotyczące struktury dziedziczenia mogą być prowadzone zarówno na poziomie klas jak i systemu. W celu badania szerokości drzewa wprowadzono metrykę *NOC*, która podaje liczbę potomków dla klasy. Wartość średnia metryki *NOC* na poziomie systemu jest wskaźnikiem wielokrotnego wykorzystania tych samych własności klas. Badanie tej metryki wykazało, że wyższa klasa w hierarchii drzewa powinna mieć więcej podklas, niż klasa niższa.

W celu analizy głębokości drzewa dziedziczenia wprowadzono metrykę *DIT*, która podaje odległość danej klasy od wierzchołka hierarchii. Maksymalna odległość wskazuje na potencjalne możliwości wielokrotnego wykorzystania tych samych własności klas. Metryka *NOC* i *DIT* pochodzą z zestawu metryk Chidambera i Kemerera. Dla przykładu przedstawionego na rysunku 2 wartość metryki *NOC* dla superklasy *A* wynosi 3, dla superklasy *B* *NOC* = 2. Dla klasy *A22* istotna jest metryka *DIT* = 2, gdyż jest to większa odległość od klasy *A* niż od klasy *B*.

Na poziomie systemu spotyka się również szczegółowe metryki dotyczące metod. Na przykład: można obliczyć liczbę klas dziedziczących specyficzną operację, liczbę metod nadpisanych i liczbę metod dodanych. Lorenz i Kidd [LOR94] stosują liczbę metod nadpisanych do obliczenia "indeksu specjalizacji", aby rozróżnić tworzenie podklas przez specjalizację (wysoka wartość indeksu) od tworzenia przez implementację (niska wartość indeksu). Indeks ten jest obliczany dla każdej klasy jako

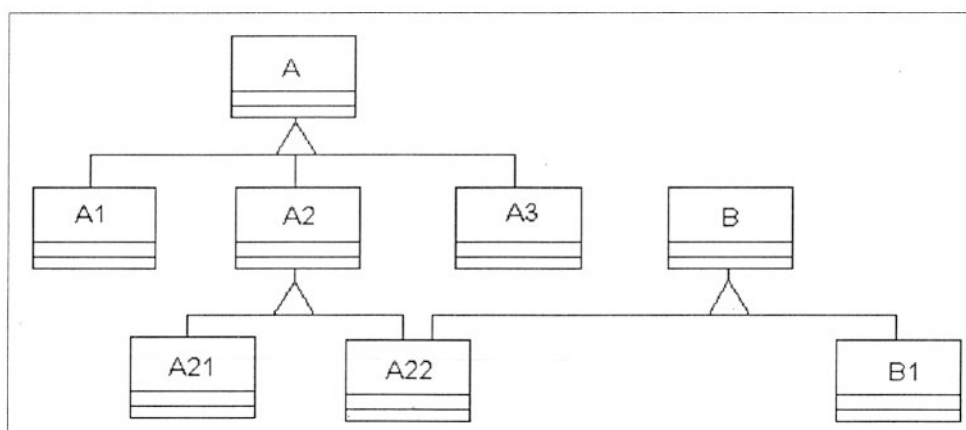
$$\frac{NORM * DIT}{NOM}, \quad (2.39)$$

gdzie:

NORM – liczba metod nadpisanych,

DIT - głębokość drzewa dziedziczenia,

NOM – liczba metod.



Rysunek 3. Hierarchia klas w diagramie klas. Źródło: [opracowanie własne].

2. 5. 2. Miary modułowe

Chociaż miary związane z liczbą linii kodu źródłowego są krytykowane, to jednak występują pod różnymi postaciami również w odniesieniu do programu obiektowego, np. suma *LOC* dla wszystkich metod w klasie, suma wyrażeń przypadających na metodę kolejno dla wszystkich metod w klasie, suma rozmiarów metod liczona w znakach dla danej klasy.

Li i Henry [LIH93] definiują miarę *SIZE1* jako liczbę średników w klasie lub jako liczbę linii kodu źródłowego bez linii komentarza oraz

$$SIZE2 = NOA + NEM, \quad (2.40)$$

gdzie: *NOA* - liczba atrybutów,

NEM - liczba metod zewnętrznych.

Wielkości *NOA* i *NEM* należą do wewnętrznych atrybutów produktu.

Podobnym wskaźnikiem może być metryka rozmiaru klasy wyrażona liczbą metod przypadających na klasę. Taka metryka może być też wykorzystana do kontroli jakościowej produktu, np. bardzo mała liczba metod sugeruje, że powinno się wykonać połączenie klas, a za duża liczba sugeruje przeprowadzenie dekompozycji. Jednak przy takich modyfikacjach należy pamiętać o spójności semantycznej. Przyjmuje się, że średnia liczba, to mniej niż 20 metod przypadających na klasę.

Liczba atrybutów lub zmiennych przypadających na klasę

Lorenz i Kidd [LOR94] wyróżniają liczbę wystąpień zmiennych *NIV* oraz liczbę zmiennych *NCV*, wówczas liczba atrybutów jest równa

$$NOA = NIV + NCV. \quad (2.41)$$

Całkowita liczba atrybutów jest wyliczana na podstawie deklaracji klasy. Jeżeli atrybuty są złożone, to całkowita ich liczba jest obliczana według podanego wzoru (2.37).

Liczba metod przypadających na klasę

Lorenz dzieli metody na zewnętrzne i *NEM* to liczba metod zewnętrznych oraz wewnętrzne, a *NHM* określa ich liczbę. Wprowadza też drugi podział, według którego *NIM* to liczba wystąpień metod, a *NCM* to liczba metod danej klasy [LOR94]. Stąd *NOM* całkowita liczba metod równa się

$$NOM = NHM + NEM = NIM + NCM. \quad (2.42)$$

Rozmiar każdej metody może być oszacowany za pomocą miary *LOC* lub *DSI*. Dla tych miar poszukuje się wartości średnich i wartości standardowych odchyłeń.

Metryką klasy jest suma metod ważonych, przypadających na daną klasę. Obliczana jest według wzoru

$$WMC = \sum_{i=1}^n c_i, \quad (2.43)$$

w którym c_i określa statyczną złożoność każdej z n metod. Jest to metryka pochodząca z zestawu Chidamberra i Kemerera.

Równania (2.42) i (2.43) nie uwzględniają atrybutów. Z opisu miary WMC nie wynika też do jakich metod (publicznych, prywatnych czy wszystkich) ma ona zastosowanie.

Kombinacja atrybutów i metod

W skład deklaracji klasy wchodzi atrybuty i metody. Na podstawie badań empirycznych ustalono równanie rozmiaru klasy, które stanowi kombinację liczby atrybutów i metod. Korzystając z równania (2.40), wyprowadzono definicję [HEN96] rozmiaru klasy S_C jako

$$S_C = NOA + NOM, \quad (2.44)$$

a zastępując liczbę metod przez ich całkowity rozmiar TMS , otrzymano ważony rozmiar klasy jako

$$S_C = NOA + TMS. \quad (2.45)$$

Po rozszerzeniu równania o odpowiednie wagi dla atrybutów i metod [THO89], [HEN93] otrzymano wartość rozmiaru dla i -tej klasy s_i , jako:

$$s_i = (AW_A + MW_M)_i, \quad (2.46)$$

gdzie:

W_A i W_M są odpowiednio wagami dla atrybutów i metod. Wagi te są zależne od języków programowania.

W celu poszukiwania ogólnego wzoru dla rozmiaru systemu wprowadzono wielkość ważoną, odpowiadającą za wskaźnik zmniejszenia długości kodu w programowaniu obiektowym w odniesieniu do programowania strukturalnego (szczególnie ze względu na wysoki stopień modułowości). Zatem równanie to zostało rozszerzone [GRA95] do postaci

$$WC_i = (AW_A + MW_M L_M + RW_R N_R)_i, \quad (2.47)$$

gdzie R jest liczbą zbiorów reguł, N_R jest liczbą reguł przypadających na zbiór reguł, W_R jest wagą powiązania. Czynniki L_M jest odzwierciedleniem nadmiarowej wielkości metody (mierzonej za pomocą $SLOC$) powyżej uzgodnionego standardu dla języka danego programowania, np. powyżej 17 linii. Miarą dla całego systemu będą zsumowane wartości rozmiarów N obiektów na poziomie klas, co daje równanie rozmiaru systemu, składające się z N klas [HEN93]:

$$S = \sum_{i=1}^N s_i = \sum_{i=1}^N (AW_A + MW_M)_i. \quad (2.48)$$

2. 5. 3. Złożoność strukturalna

Miara $V(G)$ Mc Cabe w postaci klasycznej jest również stosowana do obiektowo - zorientowanego systemu, ale nie powszechnie. Nowe podejścia do pomiaru struktury logicznej uwzględniają większą liczbę poziomów abstrakcji i większą dekompozycję modułową w systemach obiektowych niż w tradycyjnych. Rozszerzono dlatego równanie (2.20) do postaci [FEG94]:

$$v(G) = e - n + 2p, \quad (2.49)$$

gdzie p – liczba rozłącznych części grafu.

Również w analogii do miary $v(G)$ sformułowano miarę złożoności związków daną równaniem

$$AC = A - C + 2P, \quad (2.50)$$

gdzie:

A – liczba związków w diagramie klas (krawędź),

C – liczba klas w diagramie klas (węzły),

P – liczba rozłącznych części.

Miara $v(G)$ jest też stosowana do obliczania strukturalnej złożoności klasy. Wówczas kalkulacja złożoności strukturalnej s_i , przeprowadzona dla każdej i -tej klasy zawierającej m metod, ma postać

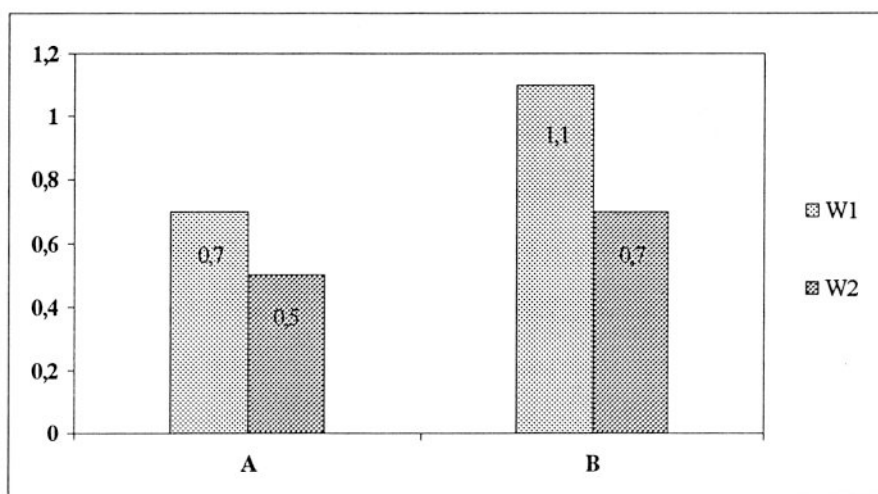
$$s_i = \sum_{j=1}^m V_{ij}(G). \quad (2.51)$$

2. 5. 4. Miary ponownego użycia

Tworzenie programu w środowisku obiektowym szczególnie sprzyja możliwości wielokrotnego wykorzystania tych samych klas (kodu). Według definicji, wtedy występuje ponowne użycie (ang. reuse), gdy w systemie znajduje się pewna liczba logicznych wyrażeń źródłowych ($LW\acute{Z}$) lub fizycznych wyrażeń źródłowych ($FW\acute{Z}$) włączonych, ale i niezmienionych. Zgodnie z powyższą definicją program może być mierzony przez liczbę

LWŻ lub FWŻ, które były na nowo utworzone dla tego systemu lub/i dołączone bez żadnej modyfikacji.

Wiele elementów związanych z tworzeniem programu może być analizowanych pod względem ponownego użycia. Do najczęściej wymienianych należy kod programu, ale ponownie wykorzystane mogą być też fragmenty dokumentacji projektowej czy dane testujące. Podstawowe źródła ponownego użycia to: dziedziczenie wśród nowych klas i korzystanie z istniejących bibliotek klas.



Rysunek 4. Porównanie produktywności wydziału W1 i W2 przy wykonywaniu programu bez ponownego użycia - przypadek A oraz z ponownym użyciem – przypadek B. Źródło: [na podstawie FEN96, s.504].

Znaczenie ponownego użycia dla procesu tworzenia programu pokazuje wykres (rys. 4), na którym przedstawiono porównanie produktywności w dwóch wydziałach wykonujących oprogramowanie firmy Hewlett-Packard. W przypadku pierwszego wydziału nastąpił wzrost produktywności o 57% , a drugiego o 40%.

Obliczanie ponownego użycia na poziomie kodu

Kod programu składa się przeważnie z części całkowicie nowo utworzonej m , części l dołączonej, zmodyfikowanej oraz części k dołączonej bez zmian. Wówczas całkowita liczba klas w systemie może być obliczona z równania

$$N=k+l+m. \quad (2.52)$$

Mierzenie powtórnego wykorzystania kodu przedstawia się za pomocą procentowego udziału wyszczególnionych części w całości produktu. W firmie NASA [FEN96] ustala się cztery poziomy ponownego użycia kodu:

- dosłowny: cały kod w jednostce programu został włączony bez zmian,

- mało zmodyfikowany: mniej niż 25% kodu w jednostce programu pochodzi ze zmodyfikowanego,
- ekstensywnie zmodyfikowany: więcej niż 25% kodu pochodzi ze zmodyfikowanego kodu,
- nowy: żadna część kodu nie pochodzi z wcześniej zbudowanej jednostki.

Obliczanie ponownego użycia na poziomie klas

Mechanizm dziedziczenia dostarcza naturalną możliwość ponownego użycia. Dlatego wprowadzono miarę do oszacowania poziomu ponownego użycia wewnątrz hierarchii, nazywaną stosunkiem ponownego użycia U , który jest obliczany według wzoru

$$U = \frac{s}{k}, \quad (2.53)$$

W którym s to liczba super klas, a k całkowita liczba klas. U zazwyczaj będzie mniejsze od 1. Wartość bliska 1 charakteryzuje pionową, głęboką, liniową hierarchię, a wartość bliska 0 wskazuje poziomą, płaską hierarchię.

Drugą miarę ponownego użycia klas stanowi wskaźnik specjalizacji S , dany jako

$$S = \frac{p}{s}, \quad (2.54)$$

gdzie:

p - liczba podklas,

s - liczba super klas.

Duża wartość S wskazuje na wysoki poziom ponownego użycia przez dziedziczenie. Ekstremalne wartości przebiegają od 1 do nieskończoności. Wartość dla S bliska 1 wskazuje na słaby projekt (stylistycznie ubogi). Ponieważ kod zmodyfikowany według definicji nie jest zaliczany do ponownego użycia, to z tego powodu cecha polimorfizmu nie jest uwzględniana w badaniach poziomu ponownego użycia, chociaż pozwala na ponowne wykorzystanie tej samej metody, ale ze zmiennymi elementami.

Uwzględnienie wpływu dziedziczenia na rozmiar programu i czas jego wykonania z powodu możliwości ponownego użycia tych samych elementów produktu nie wyczerpuje zagadnienia. Drugim źródłem możliwości ponownego użycia są posiadane biblioteki klas, utrzymywane z przewidywaniem takiej możliwości. Każdy zespół tworzący oprogramowanie dopracowuje się własnych bibliotek, składających się z procedur będących na wystarczająco wysokim poziomie uogólnienia, które mogą być stosowane w kolejnych systemach. Efekt

ponownego użycia z obu wymienionych powodów wywiera istotny wpływ na wielkość nakładów potrzebnych do jego wykonania, a w konsekwencji na ponoszone koszty, szczególnie widoczny w przypadku wielu zastosowań.

Przegląd proponowanych miar i metryk obiektowych wskazuje na różnorodność zarówno w samym podejściu do zagadnienia mierzenia programu, jak i propozycjach konkretnych rozwiązań. Niektóre powstały w wyniku adaptacji pomysłów stosowanych w technologii tradycyjnej, ale wymagają dalszych badań w celu ich weryfikacji, uogólnienia i zastosowania.

2. 5. 5. Zestaw metryk Chidambersa i Kemerera

Chidamber i Kemerer [CHI94], zarzucili istniejącym miarom oprogramowania brak podstaw teoretycznych i nieodpowiedniość dla technologii obiektowej. Zaproponowali metryki związane z ontologicznym modelem systemu informacyjnego⁷. Zestaw metryk składa się z sześciu wytypowanych charakterystyk projektu realizowanego w technologii obiektowej. Zarówno stosowana terminologia, jak i koncepcja mierzenia atrybutów projektu wskazują wyraźnie na ich przeznaczenie. Metryki są zaprojektowane do pomiaru wielkości i jakości programu przedstawionego w postaci modelu klas; przeznaczone do szacowania wielkości systemu na podstawie projektu, a nie kodu programu, co jest zgodne z zamierzeniem autorów o niezależności metryk od środków implementacji.

Metryka 1: Metody ważone w klasie WMC

W klasie C_1 są zdefiniowane metody $M_1 \dots M_n$. Niech $c_1 \dots c_n$ będą złożonością tych metod. Wówczas

$$WMC = \sum_{i=1}^n c_i. \quad (2.55)$$

Jeśli złożoności wszystkich rozważanych metod są jednością, to $WMC = n$, czyli równa się liczbie metod.

Metryka WMC jest bezpośrednio związana z definicją złożoności obiektu Bunga, gdzie metody są własnościami klas obiektów, a złożoność jest określana przez liczbę kardynalną zbioru ich własności.

⁷ Koncepcja przedstawienia systemu informacyjnego w postaci ontologicznego modelu bierze swój początek z teorii systemów Bunga [BUN77], [BUN79], a została zaadaptowana przez Wanda i Webera [WAN90] do opisu

Metryka 2: Głębokość drzewa dziedziczenia DIT

Głębokość dziedziczenia⁸ jest miarą klasy. W przypadku wielokrotnego dziedziczenia, *DIT* jest maksymalną długością od węzła do korzenia drzewa.

Metryka *DIT* względem teorii Bunga jest pojęciem zakresu własności⁹. Jest miarą, która wskazuje, jak wiele klas przodków może potencjalnie wpływać na tę klasę.

Metryka 3: Liczba potomków NOC

NOC jest równa liczbie najbliższych podklas, podległych danej klasie w drzewie hierarchii klas.

Metryka *NOC* odnosi się do pojęcia zakresu własności klasy. Wskazuje, jak wiele metod klasy rodziców może być dziedziczonych przez jej podklasy.

Metryka 4: Połączenia między klasami obiektów CBO

CBO dla klasy jest liczbą innych klas, z którymi jest ona powiązana¹⁰.

Metryka *CBO* odzwierciedla połączenia jednego obiektu z innymi, gdy jeden z obiektów działa na inny. Również ta miara wpływa na zakres własności obiektu.

Metryka 5: Odpowiedzi dla klasy RFC

$$RFC = |RS|, \quad (2.56)$$

gdzie *RS* jest zbiorem odpowiedzi dla klasy.

Zbiór odpowiedzi dla klasy może być wyrażony jako:

$$RS = \{M\} \cup_{all(i)} \{R_i\}, \quad (2.57)$$

gdzie:

$\{R_i\}$ - zbiór metod wywoływanych przez *i*-tą metodę,

$\{M\}$ - zbiór wszystkich metod w klasie.

systemu informacyjnego, gdzie elementarnym pojęciem jest byt (thing) przedstawiany za pomocą obiektu. Zarys tej koncepcji został przedstawiony w załączniku 4.

⁸ Głębokość dziedziczenia jest równoważna pojęciu: głębokość klasy w drzewie dziedziczenia. Głębokość węzła drzewa opisuje długość maksymalnej ścieżki od węzła do korzenia drzewa.

⁹ Zakres własności klasy powiększa się przez mechanizm dziedziczenia, stąd głębokość dziedziczenia wskazuje obszar, na który klasa działa przez własności przodka.

¹⁰ Dwie klasy są powiązane wtedy, gdy deklarowane metody w jednej klasie wywołują metody lub zmienne zdefiniowane w innej klasie.

Zbiór odpowiedzi klasy jest zbiorem metod, które mogą być potencjalnie wykonane w odpowiedzi na otrzymany komunikat przez obiekt tej klasy, czyli jest to sposób na dołączenie do metryki metod wywoływanych na zewnątrz klasy. Metryka ta jest także miarą potencjalnej komunikacji między daną klasą a innymi klasami.

Metryka 6: Brak spójności¹¹ w metodach LCOM

W klasie C jest zdefiniowanych n metod $M_1, M_2, \dots, M_n$, I_j = zbiór zmiennych używanych przez metodę M_i .

Jest n takich zbiorów $I_1, \dots, I_n$. Niech $P = \{(I_i, I_j) | I_i \cap I_j = \emptyset\}$ i $Q = \{(I_i, I_j) | I_i \cap I_j \neq \emptyset\}$.

Jeśli każdy z n zbiorów $I_1, \dots, I_n$ jest równy $\emptyset$, to $P=Q$.

$$LCOM = |P| - |Q|, \quad \text{jeśli } |P| > |Q| \quad (2.58)$$

$$LCOM = 0, \quad \text{dla pozostałych.} \quad (2.59)$$

W definicji tej metryki zastosowano pojęcie z teorii Bunga, stopnia podobieństwa metod. Stopień podobieństwa dla dwóch metod M_1 i M_2 w klasie C jest dany przez

$$\sigma() = \{I_1\} \cap \{I_2\}, \quad (2.60)$$

gdzie I_1 i I_2 - zbiory zmiennych używanych przez metody M_1 i M_2 .

$LCOM$ jest liczbą par metod, których podobieństwo jest równe 0 ($\sigma()$ jest zbiorem pustym), od której odejmuje się liczbę par metod, których podobieństwo nie jest równe zero. Metryka $LCOM$ wiąże się ze zmiennymi i metodami klasy, zatem jest również miarą atrybutów obiektu tej klasy.

¹¹ Spójność jest związana ze sposobem tworzenia modułów. W językach proceduralnych aranżacja zawartości modułu jest bardziej naturalna: dekompozycja programu na moduły odbywa się na podstawie kodu zawartego w obszarze danej funkcjonalności, jest to wówczas funkcjonalny typ spójności. Istnieją też inne typy spójności, np. spójność danych, która jest istotna w systemach obiektowych. Miare spójności systemu w technologii strukturalnej [FEN91] oblicza się według wzoru

$$C_f = \frac{n_f}{n},$$

gdzie: C_f - stosunek spójności, n_f - liczba modułów ze spójnością (funkcjonalną), n - całkowita liczba modułów.

Przedstawiony zestaw miar dotyczy pierwszych trzech, wyróżnianych przez Rumbaugh'a i Booc'h'a, etapów tworzenia systemu. Metryka *WMC* dotyczy pierwszego i drugiego etapu, metryka *RFC* drugiego i trzeciego, a pozostałe tylko jednego z etapów. Miary *WMC*, *DIT* i *NOC* odnoszą się do identyfikacji klas, pierwszego etapu w technologii obiektowej, *WMC* w aspekcie złożoności klasy i *DIT* oraz *NOC* bezpośrednio odnoszą się do układu hierarchii klas. *WMC* i *RFC* opisują zachowanie klas, związane z połączeniami między klasami, *LCOM* odnosi się do rozmieszczenia danych i metod wewnątrz definicji klasy, stając się miarą spójności klasy, *WMC*, *RFC* i *LCOM* odnoszą się do drugiego etapu technologii obiektowej, etapu identyfikacji semantyki klas. *RFC* i *CBO* obejmują obszar komunikacji między klasami przez obliczanie połączeń między klasami i ilości zewnętrznych metod mających dostęp do danej klasy, stanowią miarę dla trzeciego etapu, etapu identyfikacji relacji między klasami.

Wszystkie miary spełniają główne własności Weyukera z jednym wyjątkiem, własności szóstej (nierównoważność interakcji). Nie jest ona spełniona przez żadną miarę, ale też nie jest ona podstawową cechą złożoności dla projektowania obiektowego. Z teoretycznego punktu widzenia metryka, która spełnia własność szóstą, nie może być w skali przedziałowej ani stosunkowej. To oznacza, że trudno zastosować ją do następującego wnioskowania: "klasa *A* jest dwa razy bardziej złożona niż klasa *B*". Niespełnienie własności szóstej zatem może być postrzegane raczej jako korzystne dla tego zestawu miar.

Zauważa się też zakłócenie w przypadku metryk *DIT* i *LCOM*. Metryka *DIT* nie spełnia własności czwartej, wtedy, gdy dwie klasy są w relacji rodzic-potomek, ponieważ według tej własności odległość rodzica od korzenia nie może stać się większa niż jednego z jego potomków, co w tym przypadku jest niemożliwe. Również metryka *LCOM*, w warunkach konkatenacji klas może nie spełnić własności czwartej. We wszystkich pozostałych przypadkach własność ta jest spełniona.

Uwagi do zestawu miar

- Możliwość zastosowania metryk dopiero na późniejszym etapie tworzenia programu, na etapie projektowania.
- Stosowane pojęcie własności odnosi się do zmiennych i metod klasy, zatem inaczej niż jest to przyjęte w teorii technologii obiektowej, gdzie własność klasy jest reprezentowana tylko przez zmienne a metody odpowiadają możliwym działaniom danej klasy.
- Metryka czwarta i piąta nie są ortogonalne, gdyż obie odwołują się do metod lokalnych danej klasy oraz należących do innej klasy, które są możliwe do wywołania przez metody

danej klasy. Różnią się liczbą atrybutów lokalnych i liczbą atrybutów dostępnych przez związek, nie uwzględnianych przez metrykę *RFC*, a mierzonych tylko przez metrykę *CBO*.

- Metryka *WMC* określa złożoność klasy, ale nie opisuje reguł wyznaczania tej złożoności.

3. METODY SZACOWANIA WIELKOŚCI NAKŁADÓW

3. 1. Stosowane podejścia

W inżynierii oprogramowania stosuje się wiele różnych sposobów oceny i przewidywania zarówno wielkości nakładów, jak i terminów realizacji. Można je zakwalifikować do trzech podstawowych grup:

- modele algorytmiczne,
- metody polegające na orzeczeniu eksperta, analogii oraz dekompozycji,
- techniki niesformalizowane, do których zalicza się wszystkie pozostałe.

3. 1. 1. Modele algorytmiczne

Modele stosowane do estymacji nakładów identyfikują kluczowe czynniki tych nakładów, które wbudowane w formułę matematyczną odzwierciedlają relacje między nimi a nakładem. Taki model algorytmiczny powstaje na podstawie historycznych informacji o tych zależnościach i ich statystycznej analizie, a zatem należy do dziedziny problemów związanych z budową modelu ekonometrycznego w celu prognozy zjawisk zachodzących w procesach gospodarczych.

W latach sześćdziesiątych stosowano liniowe modele estymacji, które są łatwiejsze w zastosowaniu. Jednak, ze względu na zbadany doświadczalnie przy podejściu strukturalnym [PFL91, s.71], nieliniowy charakter związków między nakładem (mierzonym w liniach kodu) i jego czynnikami, obecnie przyjmuje się nieliniową postać równania wielkości nakładów. Jego podstawą jest następujący ekonometryczny model nieliniowy:

$$E = c_0 + \sum_{i=1}^n c_i x_i^{d_i}, \quad (3.1)$$

gdzie:

E – nakład mierzony w osobomiesiącach lub osobo – godzinach,

x_i - charakterystyki projektu, dotyczące produktu, procesu i zasobów,

c_0 - wyraz wolny,

c_i, d_i - stałe, wyprowadzone na podstawie danych historycznych o zakończonych projektach.

Jeżeli w modelu nakładów podstawową rolę spełnia rozmiar produktu, to w celu jego wyróżnienia, model sprowadzono [PFL91, s.71] do postaci

$$E = (a + bS^c)m(X), \quad (3.2)$$

gdzie:

S - estymowany rozmiar systemu wyrażony w tysiącach linii kodu źródłowego programu,

a, b, c - stałe, wyprowadzone na podstawie analizy regresji,

X - wektor czynników kosztu $x_1 \dots x_n$,

m - współczynnik regulujący dla czynników $x_1 \dots x_n$.

Najczęściej stosowaną w metodach estymacji kosztów ogólną postać [FEN96] nieliniowego równania nakładów przedstawia się jako

$$E = a * S^b, \quad (3.3)$$

gdzie:

E - nakład podany w osobo - miesiącach,

S - miara tworzonego produktu i występuje jako miara rozmiaru programu,

a, b - stałe, b jest zazwyczaj niewiele większe od 1, odzwierciedla wzrost nakładu dla dużego projektu.

Budowa modeli algorytmicznych wiąże się jednak z następującymi problemami:

- Trudno jest oszacować wielkość produktu S na wczesnym etapie realizacji danego projektu.
- Oszacowane parametry a i b nie są obiektywne, gdyż odzwierciedlają warunki środowiska, w którym zostały ustalone.

Z każdym modelem powinien być związany zbiór procedur dla wyznaczenia nieznanymi parametrów oraz określony sposób interpretacji otrzymanych wyników estymacji, związany z ustaleniem marginesu błędu.

3. 1. 2. Metoda orzeczenia eksperta

Technika wymaga przeprowadzenia konsultacji z grupą osób o dużym doświadczeniu w tworzeniu systemów. Każda z nich ocenia koszt danego projektu, korzystając z dowolnej techniki estymacji, ale głównie opiera się na własnej wiedzy. Następnie oceny te porównuje się i dyskutuje aż do ustalenia końcowej wielkości. Jedną z takich technik jest Delphi [BOE81], która odbywa się według następującej procedury:

- 1) Koordynator prezentuje projekt oraz inne znaczące informacje (np. dane o poprzednio realizowanych projektach podobnych) każdemu ekspertowi.
- 2) Grupa ekspertów spotyka się z koordynatorem. Eksperti zadają pytania i dyskutują nad estymacją wyników.
- 3) Każdy z ekspertów dostarcza anonimowe wstępne estymacje, wskazujące na opinie dotyczące całkowitego nakładu tworzenia. Przeważnie są to wartości podane w przedziałach liczbowych.
- 4) Koordynator przygotowuje podsumowujący raport, z wyszczególnieniem indywidualnych estymacji oraz średnią estymowaną wielkością dla grupy.
- 5) Eksperti spotykają się ponownie i dyskutują nad wszystkimi ocenami.
- 6) Eksperti dostarczają anonimowe, ostateczne estymacje. Kroki od 2 do 6 są powtarzane aż do czasu, gdy grupa zdecyduje, że wielkość średnia jest reprezentatywna i satysfakcjonująca.

3. 1. 3. Metoda analogii

Ocenę przez analogię stosuje się wtedy, gdy istnieje skompletowana dokumentacja innych projektów z tej samej dziedziny aplikacji. Wymaga ona zidentyfikowania różnic i podobieństw nowego projektu z poprzednimi na podstawie kluczowych charakterystyk projektu, np. znaczenie wymagań niezawodności, wydajności czy możliwość ponownego użycia.

Ocena wielkości nakładu może być liniowa, ale nie czasu trwania, który powinien być ustalony subiektywnie na podstawie innych, realizowanych projektów, a najlepiej większego i mniejszego, dzięki którym możliwa będzie interpolacja między ich wielkościami nakładów i czasów trwania dla estymacji nowego projektu.

3. 1. 4. Metoda dekompozycji

Metoda dekompozycji wymaga analizy tych atrybutów projektu, które mają wpływ na wielkość nakładu. Analiza dotyczy albo planowanego produktu albo zadań niezbędnych do jego wykonania. W przypadku analizy produktu dokonywana jest jego dekompozycja na części składowe. Estymacja wymaganego nakładu jest prowadzona w odniesieniu do tych składników. W drugim przypadku zadania są dekomponowane na mniejsze zadania i czynności i estymacja nakładów również dotyczy tych wyszczególnionych składników. Wyniki obu estymacji są podstawą do wyliczenia estymowanych wielkości dla całego produktu lub wszystkich zadań. Dla małych projektów jest to przeważnie sumowanie estymacji niskiego poziomu, a dla bardziej złożonych są stosowane specjalne reguły dla połączenia wielkości elementów składowych.

3. 1. 5. Techniki nieformalne

- 1) Zasada *Parkinsona*, według której koszt jest określany raczej przez dostępne zasoby niż przez oszacowaną wielkość produktu. Jeżeli projekt ma być zrealizowany, np. w ciągu 6 miesięcy, a firma może zaangażować do jego wykonania 5 osób, koszt jest szacowany według potrzeb pokrycia zapotrzebowania finansowego na 30 osobo miesięcy.
- 2) Druga z tych technik nazywana jest zasadą "*wygrywa cena*". Jest ona formą intuicyjnej manipulacji kosztem, dopasowywanym do możliwości klienta, a nie funkcjonalności programu. Na początku projektu jest ustalany jego koszt na podstawie ogólnie zarysowanej propozycji systemu, a następnie odbywają się negocjacje na temat szczegółów projektu, ale już w ramach tych kosztów. Na koniec kupujący i sprzedający uzgadniają satysfakcjonujący zakres funkcjonalny systemu. Wymagania projektowe w trakcie realizacji mogą się zmieniać, ale nie wysokość kosztów.

3. 2. Przegląd metod stosowanych w technologii strukturalnej

Spośród metod estymacji wielkości potrzebnych nakładów zaprezentowano wybrane przykłady. Opisano ich założenia, przeznaczenie oraz sformułowano uwagi dotyczące niektórych zasad i możliwości ich stosowania.

Ze względu na obecny we wszystkich metodach i podstawowy ich element - rozmiar tworzonego oprogramowania metody szacowania kosztów można podzielić na dwie grupy:

- w pierwszej rozmiar tworzonego programu jest określany za pomocą liczby linii kodu źródłowego programu (*LOC* – ang. lines of code) lub tysięcy linii kodu (*KLOC* – ang. thousands of lines of code), albo też liczby wykonanych instrukcji źródłowych (*DSI* – ang. delivered source instructions) lub tysięcy wykonanych instrukcji źródłowych (*KDSI* – ang. thousands delivered source instructions);
- druga grupa obejmuje metody modele kosztów, w których miarą rozmiaru tworzonego programu (systemu) są punkty funkcyjne (*FP*, skrót ang. function point), określające rozmiar przez identyfikację składowych systemu, postrzeganych w aspekcie realizowanych funkcji użytkownika (np. liczby wejść i liczby plików).

Możliwości stosowania określonej metody są przede wszystkim uzależnione od przyjętej jednostki pomiaru:

- jeżeli wielkość oprogramowania jest określana liczbą linii kodu źródłowego programu, metoda ta ma ograniczony zakres stosowania, ponieważ: szacowanie potrzebnych nakładów, które odbywa się na podstawie samego kodowania programu (a jest to tylko jeden z etapów realizacji¹²⁾ wytwarza wskaźnik w znacznym stopniu uproszczony i z dużym przybliżeniem; wykonanie takiego szacowania jest możliwe dopiero wtedy, gdy znana jest wielkość programu; miara w postaci liczby linii kodu utrudnia wykonywanie badań porównawczych. Nieraz jest to liczba instrukcji, nieraz liczba linii i nie zawsze wiadomo, co stanowi linię. Jest ona w pełni zależna od środków implementacji, gdyż ten sam problem rozwiązywany za pomocą programu napisanego w różnych językach programowania będzie przedstawiał inną wielkość.

- jeżeli metoda umożliwia ustalanie wielkości oprogramowania za pomocą punktów funkcyjnych, jest to miara wielkości niezależna od środków implementacji [ALB83]. Pozwala zastosować ją na wcześniejszych etapach cyklu życia, kiedy to na podstawie analizy wymagań użytkownika przyszłego systemu określa się jego przybliżoną wielkość.

W metodach szacowania kosztów podejmuje się zagadnienie opisu zależności wpływających na wielkość nakładów w różny sposób. W tabeli 9 klasyfikowano wybrane metody według kryterium stosowanej miary wielkości oprogramowania oraz typu budowanego modelu.

Tabela 9. Klasyfikacja metod szacowania kosztów.
Źródło: [opracowanie własne].

MODEL	MIARA	TYP
FPA	<i>FP</i>	<i>E,S</i>
MARK II	<i>FP</i>	<i>E,S</i>
COCOMO	<i>DSI</i>	<i>H</i>
SLIM	<i>LOC</i>	<i>D</i>
Pfleegera	<i>LOC</i>	<i>S</i>
COPMO	<i>DSI</i>	<i>D</i>
Halsteada	<i>LOC</i>	<i>T,S</i>

Wymienione w tabeli 9 metody stosują modele

- statyczne (*S*), które nie uwzględniają czynnika czasu, proces tworzenia jest sprawdzany tylko w odniesieniu do ilości i poziomu udziału czynników związanych z kosztem,
- dynamiczne (*D*), to znaczy takie, które opisują, jak wielkość nakładu tworzenia rozkłada się na cały cykl życia systemu,
- hybrydowe (*H*).

Większość z nich to metody statyczne, które nie uwzględniają wpływu czasu na wielkość ponoszonych nakładów w trakcie realizacji oprogramowania, co w przypadku średnich i dużych systemów może stanowić istotny element w planowaniu kosztów. Modele dynamiczne pozwalają na uwzględnienie zmienności nakładów zaangażowanych w kolejnych etapach

¹²Zgodnie z metodyką tworzenia systemów informatycznych CASE*Method koszty etapu budowy systemu

tworzenia, na ustalenie daty zakończenia projektu, ale podobnie, jak w przypadku modeli statycznych wymagają też opinii eksperta np. do określenia klasy złożoności czy ustalenia czynników technologicznych. Zbudowane modele na potrzebę estymacji kosztów tworzenia oprogramowania zostały opracowane w wyniku bądź to teoretycznych rozważań, bądź analizy doświadczeń z tworzenia wielu podobnych grup systemów informatycznych. Modele doświadczalne E powstały w wyniku wielu badań empirycznych w warunkach tworzenia systemów i w przypadku zastosowania ich w innym środowisku twórczym z reguły wymagają odwoływania się do opinii ekspertów. Modele T zaliczane do metod teoretycznych natomiast są niezależne od środowiska wytwarzania, ale są też trudne do zastosowania.

3. 2. 1. Metoda FPA

Metoda "Analizy punktów funkcyjnych" FPA (ang. Function Point Analysis) została opracowana przez Allana Albrechta dla firmy IBM i opublikowana w 1979 roku [ALB79]. Jest to metoda empiryczna, otrzymana w wyniku przeprowadzonych doświadczeń z 24 aplikacjami. W celu obliczenia kosztu tworzenia oprogramowania ustalono kolejne etapy obliczeń, prowadzących do otrzymania wskaźnika produktywności, wyrażającego stosunek wielkości produktu do poniesionego nakładu pracy. Nakład pracy mierzono roboczogodzinami, natomiast wielkość produktu, systemu informatycznego, określono za pomocą punktów funkcyjnych. Punkty funkcyjne stanowią wielkość bezwymiarową, wyrażającą istotne składniki systemu, poziom ich złożoności oraz stopień wpływu wybranych charakterystyk systemu. na jego rozmiar.

Wybór punktów funkcyjnych jest niezależny od narzędzi stosowanych w tworzeniu oprogramowania i narzędzi implementacyjnych, ale zastosowanie metody "Analizy punktów funkcyjnych" jest uzależnione od dziedziny rozwiązywanego problemu. Zgodnie z założeniami autora [ALB83] metoda jest przeznaczona do szacowania kosztów tworzenia systemów informatycznych wspomagających zarządzanie firmą. Nie powinna być stosowana w procesach tworzenia aplikacji naukowych i technicznych, zawierających skomplikowane algorytmy obliczeniowe.

(kodowanie i tworzenie bazy danych) stanowią 35% kosztów całego przedsięwzięcia) [MAR89].

W celu ustalenia punktów funkcyjnych bierze się pod uwagę dwa podstawowe elementy:

- rozmiar przetwarzanych informacji (ang. information processing size),
- poziom technicznej złożoności (ang. technical complexity),

które się otrzymuje z analizy wymagań użytkownika. Metoda FPA jest prowadzona w trzech podstawowych etapach:

- I. Obliczanie "niedopasowanych" punktów funkcyjnych na podstawie identyfikacji składników systemu i ich klasyfikacji według trzech poziomów złożoności.
- II. Obliczanie sumarycznego poziomu technicznej złożoności systemu na podstawie oszacowania wielkości wpływu charakterystycznych czynników.
- III. Obliczanie ostatecznej liczby punktów funkcyjnych, którą stanowi liczba "niedopasowanych" punktów funkcyjnych, skorygowana o oszacowaną wielkość wpływu czynników technicznej złożoności systemu.

Po przeprowadzeniu obliczeń w ramach tych etapów należy¹³:

- odczytać, dla wyliczonej liczby punktów funkcyjnych czas potrzebny na realizację takiego systemu (z tabeli otrzymanej w wyniku przeprowadzonych badań), a następnie
- obliczyć wskaźnik produktywności jako stosunek liczby punktów funkcyjnych do czasu realizacji systemu.

Etap I. Obliczanie "niedopasowanych" punktów funkcyjnych

W pierwszym kroku jest ustalany rozmiar systemu przez identyfikowanie jego części składowych, widzianych przez końcowego użytkownika. Następnie przeprowadza się klasyfikację tych elementów według pięciu określonych typów (tab. 10). Znaczenie składników systemu dla mierzonej wielkości produktu jest wyrażane za pomocą liczby wagowej.

¹³ Przykład obliczania punktów funkcyjnych zawarto w załączniku 5 oraz można znaleźć w opracowaniu [CZA94].

Tabela 10. Wagi typów składników według poziomów złożoności. Źródło: [ALB83, s.647].

NAZWA TYPU	POZIOM ZŁOŻONOŚCI		
	PROSTY	ŚREDNI	ZŁOŻONY
Zewnętrzne wejście (<i>external input</i>)	3	4	6
Zewnętrzne wyjście (<i>external output</i>)	4	5	7
Logiczny plik wewnętrzny (<i>logical internal file</i>)	7	10	15
Plik przekazywany lub wspólny z innym systemem (<i>external interface file</i>)	5	7	10
Zewnętrzne zapytanie (<i>external inquiry</i>)	3	4	6

Wielkości wagowe danego typu składników zależą od poziomu ich złożoności (prosty, średni i złożony). Kryterium tego podziału uwzględnia przede wszystkim liczbę elementów danych w każdym typie składnika. Każdy składnik jest przedstawiany jako liczba punktów zależna od jego typu i poziomu złożoności. Natomiast suma punktów dla wszystkich składników, która stanowi tzw. "niedopasowane" punkty funkcyjne *UFP* (ang. *Unadjusted Function Point*), jest obliczana według wzoru:

$$UFP = \sum_{i=1}^5 \sum_{j=1}^3 w_{ij} z_{ij} \tag{3.4}$$

gdzie:

z_{ij} - liczba składników i -tego typu oraz j -tego poziomu

w_{ij} - wielkość wagowa ustalona przez procedurę Albrechta dla składnika z_{ij}

Etap II. Obliczanie sumarycznego poziomu technicznej złożoności TCF (The Technical Complexity Factor)

W drugim etapie obliczeń charakteryzuje się system na podstawie wutypowanych czternastu czynników.

Tabela 11. Lista charakterystycznych czynników aplikacji. Źródło: [ALB83, s. 647].

ID	CHARAKTERYSTYKA	ID	CHARAKTERYSTYKA
C1	Transmisja danych	C8	Aktualizacja on-line
C2	Rozproszone przetwarzanie	C9	Stopień złożoności przetwarzania
C3	Wydajność	C10	Ponowne użycie
C4	Obciążenie konfiguracji	C11	Łatwość instalacji
C5	Tempo przyrostu transakcji	C12	Łatwość eksploatacji
C6	Wprowadzanie danych on-line	C13	Wielokrotność instalacji
C7	Sprawność końcowego użytkownika	C14	Możliwość wprowadzania zmian

Lista tych czynników (tab. 11), zwanych "głównymi charakterystykami aplikacji" (ang. General Application Characteristics), reprezentuje grupę elementów o drugorzędnym znaczeniu dla szacowanego rozmiaru systemu. Odzwierciedlają one parametry techniczne, związane z realizacją przetwarzania danych. Poziom wpływu poszczególnych elementów z tej listy jest ustalany w skali od 0 do 5 (tab. 12). Suma punktów dla wszystkich 14 charakterystyk jest określona jako całkowity stopień wpływu DI , zaś sumaryczny poziom technicznej złożoności systemu TCF jest obliczany przy użyciu formuły:

$$TCF=0.65+0.01*DI. \tag{3.5}$$

Wartość TCF waha się w granicach od 0.65 do 1.35, a liczba "niedopasowanych" punktów funkcyjnych może być modyfikowana w granicach $\pm 35\%$.

Tabela 12. Skala wpływu czynników. Źródło: [ALB83, s. 647].

CHARAKTER WPŁYWU	LICZBA STOPNI
Nieobecny lub brak wpływu	0
Nieznaczny wpływ	1
Umiarkowany wpływ	2
Średni wpływ	3
Znaczny wpływ	4
Silny wpływ, we wszystkich obszarach	5

Etap III. Obliczanie punktów funkcyjnych (FP)

Po wykonaniu rachunków w obu etapach wielkość systemu ustalana w punktach funkcyjnych *FP* ma postać równania:

$$FP = UFP * TCF \quad (3.6)$$

W uzasadnieniu słuszności swojej metody Albrecht podkreśla, że proponowanie punktów funkcyjnych jako miary wielkości systemu pozwala uzyskać [SYM88]:

- odizolowanie faktycznej miary systemu od czynników środowiskowych, co ułatwia badanie czynników wpływających na produktywność,
- miary zgodne z widzeniem systemu przez użytkownika systemu, które są niezależne od technologii,
- miary w postaci punktów funkcyjnych, które mogą być wcześniej określone w cyklu tworzenia, umożliwiając wykorzystanie *FP* w procesie estymacji na etapie planowania i analizy,
- zrozumienie i stosowanie metody przez nietechnicznych użytkowników.

3. 2. 2. Metoda MARK II

Wynikiem alternatywnego podejścia do zastosowania metody punktów funkcyjnych Albrechta jest metoda MARK II, opracowana przez Symonsa [SYM88] i opublikowana jako nowa miara wielkości przetwarzanych informacji. Zachowuje ona podstawowe przesłanki metody Albrechta, tj. niezależność wielkości przetwarzanych informacji od technologii przetwarzania (jak tylko jest to możliwe) oraz zachowanie terminologii łatwo rozpoznawalnej przez użytkownika systemu.

Podstawowe założenia metody MARK II:

- system jest postrzegany jako zestaw logicznych "typów transakcji", takich jak logiczne kombinacje wejście/proces/wyjście,
- pliki przekazywane lub wspólne z innymi systemami są traktowane jako innego rodzaju wejście lub wyjście,
- zapytania są rozważane jako innego rodzaju kombinacja wejścia/procesu/wyjścia,

- koncepcja "logicznego pliku wewnętrznego" z metody Albrechta jest zastąpiona pojęciem "entity" (w literaturze polskiej znane jako encja lub obiekt).

Założenia te modyfikują podejście do klasyfikacji składowych wielkości systemu, wprowadzając pojęcie obiektu, jako nowego elementu systemu. Obiekty funkcjonują od wielu lat w procesie modelowania systemów informacyjnych jako podstawowy element diagramu Obiekt - Związek (ang. Entity Relationship Diagram) [CHE76], stanowiącego model konceptualny danych, który odzwierciedla potrzeby informacyjne użytkownika i jest budowany od samego początku prac związanych z tworzeniem systemu. Na etapie strategii występuje jako model ogólny, a na etapie szczegółowej analizy wymagań użytkownika, jako model szczegółowy.

Zaproponowano składniki i skalę wielkości systemu związaną z nakładem pracy na etapie analizy, projektowania i tworzenia funkcji systemu. Takie podejście jest zrozumiałe i adekwatne z poziomu pracy twórców systemu, a nie jak w metodzie Albrechta, uzależnionej od przekazów użytkownika systemu.

Formuła obliczania wielkości przetwarzanych informacji dla metody MARK II wyrażona w "niedopasowanych" punktach funkcyjnych UFP brzmi

$$UFP = N_I W_I + N_E W_E + N_O W_O, \quad (3.7)$$

gdzie:

N_I - liczba typów wejściowych elementów danych (ang. input data element types),

W_I - waga wejściowego typu danych,

N_E - liczba odwołań¹⁴ do typów obiektów (ang. entity-type references),

W_E - waga odwołania do typu obiektu,

N_O - liczba wyjściowych typów danych (ang. output data element types),

W_O - waga wyjściowego typu danych,

N_I , N_E , N_O są sumą wszystkich typów transakcji.

¹⁴Odwołania reprezentują operacje tworzenia, aktualizacji, odczytywania lub usuwania.

Wagi dla poszczególnych typów transakcji zostały ustalone w wyniku badań konsultacyjnych, w których wykorzystano dane zebrane z 12 systemów o różnych rozmiarach i tworzonych w różnych technologiach. Kryterium wyboru systemów było ograniczone do okresu ich tworzenia, tj. przynajmniej 3-4 miesiące. Po ustaleniu wartości wagowych, równanie wielkości przetwarzanych informacji uzyskało postać

$$UFP = 0.44N_I + 1.67N_E + 0.38N_O, \quad (3.8)$$

W celu wyliczenia czynnika technicznej złożoności TCF wprowadzono, oprócz 14 czynników sformułowanych przez Albrechta, sześć innych:

- interfejs z innymi aplikacjami,
- własności specjalnego zabezpieczenia,
- bezpośredni dostęp dla osób "trzecich",
- specjalne wymagania dotyczące dokumentowania,
- specjalistyczne szkolenie użytkownika,
- definiowanie, wybór i instalacja specjalnego sprzętu komputerowego oraz unikalnego oprogramowania.

Formuła obliczania sumarycznego poziomu technicznej złożoności systemu, przy uwzględnieniu 20 charakterystycznych czynników ma następującą postać

$$TCF = 0,65(1 + Y/X), \quad (3.9)$$

gdzie:

Y – roboczo-godziny poświęcone TCF ,

X – roboczo-godziny poświęcone wielkości przetwarzanej informacji mierzonej UFP .

Zasadą wydzielania składników TCF było wymaganie, by był to czynnik, który nie dotyczy informacji, ale jest istotny i oddziałuje na rozmiar systemu, ale także nie wynika ze środowiska tworzenia systemu.

3. 2. 3. Metoda COCOMO

Metoda COCOMO¹⁵ (CONstruktive COSt MOdel) jest modelem hybrydowym, ponieważ łączy metody doświadczalne z elementami wnioskowania statystycznego. Jest metodą trójpoziomową: na poziomie podstawowym, pośrednim i szczegółowym. Modele budowane na kolejnych poziomach tworzą hierarchię coraz to bardziej szczegółowych modeli: od modelu w skali makro-estymacji, jako funkcji wielkości produktu, do modelu mikro-estymacji. Ponadto metoda COCOMO zawiera formuły dla przewidywania czasu tworzenia i planowania projektu, dzieląc nakłady na fazy i czynności. Pojęcie projektu jest tutaj używane w znaczeniu przedsięwzięcia informatycznego.

Formuła estymacji nakładu tworzenia projektu dla każdego z poziomów budowanego modelu ma postać zgodną z równaniem (3.2):

$$E = aS^b m(X), \quad (3.10)$$

gdzie: $m(X)$ - współczynnik wnioskowania, jest złożoną funkcją 15 czynników kosztu (nazywanych sterownikami kosztu) od x_1 do x_{15} , obliczany zgodnie z równaniem

$$m(X) = \prod_{i=1}^{15} m(x_i), \quad (3.11)$$

w którym:

a, b - parametry ustalone arbitralnie na podstawie doświadczeń autora i innych menedżerów projektów,

S - wielkość projektu wyrażona liczbą linii instrukcji programu.

Metoda COCOMO poziomu pośredniego przebiega następująco:

- 1) Nominalny nakład tworzenia jest estymowany jako funkcja wielkości produktu mierzonej w tysiącach instrukcji kodu źródłowego *KDSI* (ang. Kilo Delivered Source Instructions) oraz ustalonego trybu tworzenia projektu.

¹⁵ W załączniku 6 przedstawiono przykład zastosowania metody COCOMO.

- 2) Zestaw współczynników nakładów jest określany na podstawie zidentyfikowanych wskaźników produktu ze zbioru 15 atrybutów sterujących kosztem.
- 3) Estymowany nakład tworzenia jest otrzymywany przez pomnożenie nominalnej wielkości nakładu przez wszystkie współczynniki.
- 4) W celu określenia, na podstawie przeprowadzonej estymacji, wielkości nakładu w jednostkach monetarnych należy uwzględnić dodatkowe elementy, np. harmonogram tworzenia, fazę i czynności dystrybucji, koszty komputera, koszty pielęgnacji oraz inne potrzebne elementy.

Krok 1. Estymacja nominalnego nakładu

W tabeli 13 zamieszczono charakterystykę trybów tworzenia projektu: organiczny (ang. organic), bliźniaczy (ang. semidetached) i wbudowany (ang. embedded). Na podstawie wymienionych cech ustala się tryb tworzonego projektu. Po przeprowadzeniu analizy 63 projektów autor metody otrzymał dla każdego trybu różne parametry równania estymacji.

Tabela 13. Tryby tworzenia oprogramowania w metodzie COCOMO. Źródło: [BOE84, s. 13].

CECHA	TRYB		
	ORGANICZNY	BLIŹNIACZY	WBUDOWANY
Organizacyjne zrozumienie celów produktu	Kompletne	Znaczne	Ogólne
Doświadczenie w pracy z podobnymi systemami	Obszerne	Znaczne	Umiarkowane
Potrzeba dostosowania oprogramowania do wcześniej ustalonych wymagań	Podstawowa	Znaczna	Pełna
Potrzeba dostosowania oprogramowania do specyfikacji zewnętrznego interfejsu	Podstawowa	Znaczna	Pełna
Równoległość tworzenia w powiązaniu z nowym sprzętem i operacyjnymi procedurami	Niektóre	Umiarkowanie	Obszerne
Potrzeba innowacji architektury przetwarzania danych, algorytmów	Minimalna	Niektóre	Znaczna
Premia za wcześniejsze zakończenie	Niska	Średnia	Wysoka
Zakres wielkości produktu	<50 KDSI	<300 KDSI	Dowolna wielkość

W tabeli 14 zawarto wyniki skalowania równań dla każdego z trybów tworzenia, podano wielkość nakładu w osobo - miesiącach (man-month) jako jako funkcję wielkości projektu (w *KDSI*) i trybu tworzenia. Ogólna postać równania ma postać

$$(MM)_{NOM} = a(KDSI)^b \tag{3.12}$$

Tabela 14. Równania nakładu i planowanego czasu w metodzie COCOMO. Źródło: [BOE84, s. 13].

Tryb tworzenia	Nakład nominalny	Plan
Organiczny	$(MM)_{NOM} = 3,2(KDSI)^{1,05}$	$T_{DEV} = 2,5(MM_{DEV})^{0,38}$
Bliźniaczy	$(MM)_{NOM} = 3,0(KDSI)^{1,12}$	$T_{DEV} = 2,5(MM_{DEV})^{0,35}$
Wbudowany	$(MM)_{NOM} = 2,8(KDSI)^{1,20}$	$T_{DEV} = 2,5(MM_{DEV})^{0,32}$

Na przykład, dla projektu o oszacowanej wielkości 10 000 instrukcji źródłowych i ustalonego trybu "wbudowanego" na podstawie wymagań użytkownika, nominalny nakład tworzenia w osobo - miesiącach wynosi

$$2.8(10)^{1.20} = 44MM \text{ .}$$

Krok 2. Określenie współczynników nakładu

Sformułowano piętnaście czynników uznanych za atrybuty sterujące kosztem (tab. 15). Po ustaleniu kategorii wpływu (bardzo niski, niski, nominalny, wysoki, bardzo wysoki i najwyższy) i oszacowaniu skali otrzymano doświadczalnie, dla każdego z atrybutu, zbiór odpowiadających im współczynników (tab. 16). Współczynnik wskazuje ile razy musi być pomnożona wielkość nominalnego nakładu, aby wyliczyć niezbędną pracę dla danego projektu, przy ustalonej kategorii wpływu danego atrybutu.

Tabela 15. Sterowniki kosztu w metodzie COCOMO. Źródło: [BOE84, s. 14].

<i>Atrybuty produktu</i> RELY DATA CPLX	<i>Cecha</i> Wymagana niezawodność oprogramowania Rozmiar bazy danych Stopień złożoności produktu
<i>Atrybuty komputera</i> TIME STOR VIRT TURN	<i>Cecha</i> Ograniczenie czasu wykonania Ograniczenie pamięci Lotność maszyny wirtualnej Czas cyklu komputera
<i>Atrybuty personelu</i> ACAP AEXP PCAP VEXP LEXP	<i>Cecha</i> Kwalifikacje analityków Doświadczenia aplikacyjne Kwalifikacje programistów Doświadczenie z maszynami wirtualnymi Doświadczenie w programowaniu
<i>Atrybuty projektu</i> MODP TOOL SCED	<i>Cecha</i> Praktyka w stosowaniu nowoczesnego programowania Stosowanie narzędzi tworzenia oprogramowania Wymagany harmonogram tworzenia

W celu obliczenia sumarycznego współczynnika wnioskowania przede wszystkim należy przyporządkować poziom wpływu (niski, wysoki itd.) danego atrybutu, a następnie odczytać z tabeli 16 wartość współczynnika. Tabele zawierające charakterystyki dotyczące podziału atrybutów na kategorie są zamieszczone w publikacjach autora metody [BOE81],[BOE84, s. 15].

Tabela 16. Współczynniki nakładu tworzenia oprogramowania dla odpowiednich sterowników w pośredniej metodzie COCOMO. Źródło: [BOE84, s. 14].

STEROWNIK KOSZTU	KATEGORIA					
	BARDZO NISKI	NISKI	NOMI- NALNY	WYSOKI	BARDZO WYSOKI	NAJ- WYŻSZY
<i>Produkt</i>						
RELY	0,75	0,88	1,00	1,15	1,40	
DATA		0,94	1,00	1,08	1,16	
CPLX	0,70	0,85	1,00	1,15	1,30	1,65
<i>Computer</i>						
TIME			1,00	1,11	1,30	1,66
STOR			1,00	1,06	1,21	1,56
VIRT		0,87	1,00	1,15	1,30	
TURN		0,87	1,00	1,07	1,15	
<i>Personel</i>						
ACAP	1,46	1,19	1,00	0,86	0,71	
AEXP	1,29	1,13	1,00	0,91	0,82	
PCAP	1,42	1,17	1,00	0,86	0,70	
VEXP	1,21	1,10	1,00	0,90		
LEXP	1,14	1,07	1,00	0,95		
<i>Projekt</i>						
MODP	1,24	1,10	1,00	0,91	0,82	
TOOL	1,24	1,10	1,00	0,91	0,83	
SCED	1,23	1,08	1,00	1,04	1,10	

Krok 3. Estymacja nakładu tworzenia

Przybliżony nakład tworzenia oblicza się jako iloczyn nominalnego nakładu tworzenia (wynik kroku 1) i sumarycznego poziomu wpływu piętnastu sterujących atrybutów kosztu (tab. 16), otrzymanego w kroku 3. Przybliżony nakład dla przykładowego projektu wynosi zatem:

$$(44\ MM) * (1.35) = 59\ MM.$$

Krok 4. Estymacja czasu trwania realizacji projektu

Metoda COCOMO umożliwia pokazanie dodatkowych zależności związanych z tworzeniem oprogramowania w celu podania oszacowanych nakładów projektu w jednostkach

monetarnych a także dokonania rozkładu kosztów i nakładów na fazy cyklu życia (analiza wymagań, projektowanie itd.) lub podziału ze względu na występujące w projektach rodzaje czynności (programowanie, planowanie testu, zarządzanie itd.). Te uzupełniające relacje wspomagają estymację budowanego harmonogramu projektu oraz fazę jego dystrybucji. Zatem planowany czas tworzenia dla przykładowego projektu (tryb "wbudowany"), obliczony na podstawie równania z tabeli 14, wynosi

$$T_{DEV} = 2.5(59)^{0.32} = 9MM.$$

Sterowanie, omówionymi wyżej, współczynnikami nakładów pozwala na przeprowadzanie złożonych analiz wrażliwości i analiz porównawczych. Można przeprowadzić badanie wpływu tych współczynników na estymowany nakład, dokonując na przykład redukcji. I tak, korzystając z danych tabeli 16, redukcja atrybutu *TURN* z wartości wskaźnika 1.00 do 0.87 spowoduje zmniejszenie oszacowanego (dla cytowanego przykładu) nakładu

$$(59 MM) * (0.87) = 51 MM.$$

3. 2. 4. Model Pfleegera

Model opracowany w 1988 roku przez S.L. Pfleegera [PFL91, s. 82] bardziej szczegółowo niż inne ujmuje zagadnienie tzw. ponownego wykorzystania (reuse) w innym projekcie tych samych modułów programowych, stanowiących oprogramowanie biblioteczne programisty. Jest to model statyczny, który wyraża rzeczywistą produktywność P_{actual} projektu jako wynik iloczynu jego średniej produktywności P_{avg} i funkcji regulującej, zależnej od zestawu czynników kosztu $(X_1, \dots, X_n)$. Ma postać:

$$P_{actual} = P_{arg} * f(X_1, \dots, X_n). \quad (3.13)$$

W odróżnieniu do innych modeli, zamiast wyboru ze stałego zestawu czynników proponuje użytkownikowi zdefiniowanie własnych czynników, dopasowanych do specyfiki projektu i środowiska, w którym jest realizowany. Dla każdego zidentyfikowanego czynnika X należy wygenerować współczynniki przez badanie kosztu włączenia X do projektu i ustalenia liczby projektów, powyżej której koszty te będą zamortyzowane. Na przykład, jeśli X jest wykorzystywanym narzędziem, to współczynnik A_X jest wyprowadzany na podstawie:

- części projektu stosującej narzędzie X (od 0 do 1),

- kosztu tworzenia narzędzia X ,
- kosztu włączenia narzędzia X do projektu wraz z kosztem szkolenia twórców w jego stosowaniu,
- liczby projektów, powyżej której koszty będą amortyzowane (tj. przewidywanej liczby projektów, w których korzystało się z narzędzia X).

Zatem współczynniki $(A_1, \dots, A_n)$, określone dla czynników kosztu $(X_1, \dots, X_n)$ występują w formule modelu Pfliegera w postaci złożonego czynnika kosztu $f(X_1, \dots, X_n)$, opisującego ich wzajemne oddziaływanie.

W razie ustalania oszczędności, wynikającej z wielokrotnego wykorzystania wytworzonego modułu programowego, zwraca się uwagę, że różnica w wielkości nakładów będzie miała inne znaczenie dla kilku projektów, w których to badany moduł będzie zastosowany, niż w przypadku modułu uniwersalnego, włączonego do biblioteki programów. Wówczas koszt powstania tego modułu może być amortyzowany przez większą liczbę projektów.

Model Pfliegera umożliwia przeprowadzanie analiz porównawczych w celu określenia liczby wymaganych zastosowań dla zapewnienia mniejszego kosztu wykonania wszystkich projektów. Rozważania te proponuje się prowadzić nie tylko w aspekcie możliwości wielokrotnego stosowania tego samego kodu źródłowego programu, ale także innych czynników kosztu [BOL90], np. w odniesieniu do takiej samej specyfikacji wymagań, takiego samego projektu bazy danych, testowania danych.

3. 2. 5. Metoda SLIM

Model metody SLIM [Software Life cycle Management] został zbudowany w 1977 roku przez Putnama w wyniku studiów nad 50 projektami tworzonymi przez wojskowy ośrodek informatyczny oraz 150 innymi (opublikowany w 1978 roku [PUT78]). Najważniejszą przesłanką tej metody jest ciągła zmienność tempa tworzenia kodu programu. Za istotne dla nakładu tworzenia uznano następujące czynniki [PUT79]:

- technologię projektu mierzoną doświadczeniem zespołu,
- poniesiony nakład pracy,
- ustalony czas tworzenia,

- dowolny punkt czasowy w cyklu życia (zmienna niezależna)

oraz

- gradient trudności mierzony złożonością problemu.

W modelu Putnama do prezentacji rozkładu nakładu pracy w czasie wykorzystano równanie krzywej Rayleigh'a

$$y = \frac{K}{t_d^2} t e^{-\frac{t^2}{2t_d^2}}, \quad (3.14)$$

gdzie:

y - wielkość nakładów pracy w czasie t ,

K - obszar pod krzywą Rayleigha dla przedziału $[0,1]$, który przedstawia nominalny nakład pracy dla danego cyklu życia mierzony w osobo - latach,

t_d - czas, w którym nakład pracy osiąga maksimum,

t - czas, zmienna niezależna.

Związek między wielkością programu, czasem jego tworzenia i potrzebnym nakładem pracy został wyrażony następująco:

$$S_s = cK^{\frac{1}{3}}T^{\frac{4}{3}}, \quad (3.15)$$

gdzie:

S_s - wielkość programu mierzona w liniach kodu źródłowego,

c - czynnik technologii, który odzwierciedla ograniczenia sprzętowe, zawilość programu, doświadczenie personelu i środowisko programowania,

K - wielkość nakładu pracy w ciągu całego cyklu życia w osobo - latach,

T - czas tworzenia w latach.

W wyniku przeprowadzonych doświadczeń [PFL91, s. 85] zaproponowano listę wartości czynnika c w przedziale od 610 do 57 314. Na przykład c jest równe 2000 dla projektu tworzonego bez określonej metodologii, ze skróconą dokumentacją i wsadowym trybie przetwarzania; c jest równe 8000, gdy istnieje metodologia, wykonano adekwatną dokumentację i występuje przetwarzanie interakcyjne. Zaś za znakomity projekt uważa się zastosowanie w procesie tworzenia zautomatyzowanych narzędzi i technik, a wartość c ustala

na równą 11 000. Alternatywnie, c może być obliczane na podstawie danych historycznych dla poszczególnych organizacji.

Jako istotny czynnik pomiaru wielkości nakładu w modelu uwzględnia się miarę poziomu złożoności problemu D , obliczaną według wzoru:

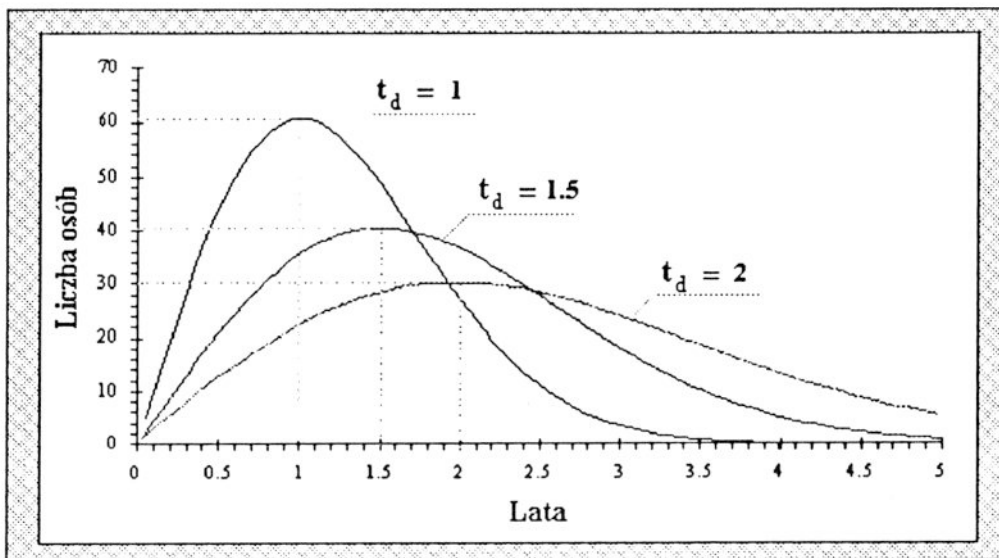
$$D = \frac{K}{T^2}. \tag{3.16}$$

Dla projektów prostych, uznanych za względnie łatwe do wykonania ustalono, że D jest małe, a dla trudnych, D zmierza do dużego. Związek między poziomem złożoności a produktywnością określono następująco

$$P = cD^{-\frac{2}{3}}, \tag{3.17}$$

gdzie P - miara produktywności.

Dynamika modelu umożliwia badanie wpływu zmieniającej się liczby osób, zaangażowanych w realizację projektu, na spodziewany czas jego zakończenia. Przedstawione na rysunku 5 krzywe ilustrują trzy możliwe rozkłady nakładu pracy w czasie dla tego samego projektu. Całkowita wielkość nakładu, reprezentowana przez obszar pod każdą krzywą, jest taka sama, ale wielkość nakładu, po upływie tego samego czasu ma dla każdej z nich inną wartość.



Rys. 5. Alternatywne nakłady projektu; źródło: [PUT79, s.138].

3. 2. 6. Metoda COPMO

Model kooperatywnego programowania COPMO (Cooperative Programming Model) został opracowany w początku lat 1980-tych przez Thebauta [PFL91, s. 88]. W odróżnieniu do innych modeli wprowadza do rozważań nad estymacją nakładu nie tylko główny czynnik - wielkość projektu ale też i wielkość zespołu projektowego, jako drugi istotny czynnik nakładu. Zgodnie z tym podejściem, nakład został zdefiniowany jako

$$E = E_1(S) + E_2(M), \quad (3.18)$$

gdzie:

E - całkowity nakład,

$E_1(S)$ - nakład wymagany przez jedną lub więcej osób pracujących niezależnie nad modułami nie wymagającymi interakcji z innymi modułami,

$E_2(M)$ - nakład wymagany dla koordynacji procesu tworzenia z innymi członkami zespołu, a

M - średnia liczba osób przydzielonych do projektu.

Pierwszy składnik ma postać

$$E_1(S) = a + bS, \quad (3.19)$$

a drugi

$$E_2(M) = cM^d. \quad (3.20)$$

Wartości a , b , c i d otrzymano w wyniku kilku testów przeprowadzonych przez grupę Purdue Software Metrics Research Group. Wielkość d jest interpretowana jako reprezentacja znaczenia koordynacji, zarządzania i komunikacji potrzebnej w zespole projektowym, a wielkość c reprezentuje słabości ścieżek komunikacji. W tabeli 17 przedstawiono empirycznie wyznaczone wartości parametrów na podstawie sześciu testów.

Tabela 17. Parametry dla modelu COPMO. Źródło: [PFL91, s. 88].

ZESTAW	LICZBA PROJEKTÓW	A	B	c	D
A	63	3,42	1,31	2,59	1,60
B	33	3,42	1,31	1,54	1,71
C	40	3,42	1,31	N/A	1,34
D	19	4,75	0,53	3,02	1,60
E	17	3,09	0,26	6,74	1,59
F	15	3,42	1,31	8,19	1,22

Metoda COPMO została uogólniona w trakcie prac zespołu Purdue Group i model COPMO przyjął postać

$$E = a \frac{d_i}{i} + b_i S + c_i M, \tag{3.21}$$

gdzie i oznacza i - ty nakład w złożonej klasie EC_i .

Dla projektów zostały ustalone określone klasy złożoności. Ich wyróżnienie może odbywać się na kilka sposobów. Sposób nie jest istotny dla metody, ale klasy muszą być tak uporządkowane, by projekt będący w klasie EC_i był mniej złożony niż projekt w klasie EC_{i+1} .

3. 2. 7. Model teoretyczny Halsteada

W roku 1977 Halsteada [HAL77] sformułował kilka zasad poznawania i mierzenia charakterystyk oprogramowania dotyczących obliczania operatorów i operandów występujących w module programowym. Zaproponował mierzenie złożoności projektu stwierdzając, że bardziej złożone projekty wymagają więcej nakładu. Przyjęto dwa główne założenia o tworzeniu systemu. Po pierwsze, wyrażenie algorytmu, który ma być implementowany w rozważanym module jest kompletne i niedwuznaczne. Po drugie, przydzielony do pisania kodu programista posiada biegłą znajomość języka implementacji, będzie pracował samodzielnie i posiada wysoki poziom koncentracji (bez dłuższych przerw). Jako wskaźnika wielkości wymaganego nakładu użyto pojęcia liczby "pamięciowych

wyróżnień" (mental discriminations) [PFL91, s. 72], której kalkulacja wynika z liczby operatorów i operandów występujących w obliczeniach. Zdefiniowano n jako liczbę unikalnych operatorów i operandów w module, a N jako całkowitą liczbę operatorów i operandów. Zatem N było obliczane jako

$$N = kS_s \quad (3.22)$$

gdzie:

k - stała, oznaczającą średnią liczbę operatorów i operandów przypadającą na linię kodu źródłowego dla konkretnego języka programowania,

S_s - liczba linii kodu źródłowego.

Ustalono, że k dla języka FORTRAN jest w przybliżeniu równe 7 i równe 5 dla języka PL/S. Następnie n obliczano według formuły

$$N = n \log_2 \frac{n}{2}. \quad (3.23)$$

W efekcie końcowym estymacja nakładu E jest obliczana jako

$$E = \frac{N^2 \log_2(n)}{4}, \quad (3.24)$$

a po aproksymacji jako

$$E = \frac{N^{2.28}}{4}. \quad (3.25)$$

Warto podkreślić, że było to jedno z pierwszych rozwiązań problemu mierzenia nakładu, choć wywołało wiele kontrowersyjnych dyskusji.

3. 2. 8. Ocena możliwości stosowania przedstawionych metod

Problem wyboru i zastosowania metody wynika zarówno ze stopnia trudności jej wykorzystania, jak i z poziomu wiarygodności otrzymanych wyników. Zarówno metoda FPA,

jak i COCOMO są efektem wielu eksperymentów, a zasady ich działania uwzględniają opinie uczestniczących w nich menedżerów projektu. Na potrzeby tych metod zostały arbitralnie ustalone charakterystyki aplikacji (metoda FPA), atrybuty sterujące kosztem (metoda COCOMO), a ich współczynniki oraz parametry występujące w równaniach znaleziono doświadczalnie. Wymienione elementy są zatem charakterystyczne dla organizacji tworzących testowane oprogramowanie lub podobnych do uczestniczących w eksperymencie. W przypadku zaś dowolnego zespołu projektującego powinny być weryfikowane. Metody te wymagają także decyzji, dla których nie ma ściśle określonych reguł, np. określenie poziomu złożoności poszczególnych elementów tworzonego oprogramowania czy dopasowanie odpowiedniego poziomu współczynników. Niewłaściwy dobór tych poziomów spowoduje otrzymanie nieprawdziwego obrazu wielkości systemu i nie może doprowadzić do wystarczającej oceny wymaganych nakładów. Zatem zaleca się odwoływanie do doświadczeń i pomocy ekspertów.

Oczekiwania zarówno użytkowników tych metod, jak i ich twórców zmierzają w kierunku poszukiwania metod niezależnych od subiektywnych decyzji oraz możliwych do zastosowania na wczesnych etapach tworzenia systemu. Tworzone obecnie systemy informatyczne zgodnie z nowoczesną technologią tworzenia podsuwają możliwości obiektywnego podejścia do ustalania wielkości systemu i wykonania estymacji wcześniej, niż jest to możliwe przy tradycyjnym podejściu. Podstawą tych technologii jest gromadzenie informacji o wymaganiach użytkownika w postaci sformalizowanych struktur już od samego początku. Informacje te, czyli wiedza o powstającym systemie, są zapamiętywane w tzw. repozytorium, czyli meta bazie systemu informacyjnego. Meta baza jest skarbnicą danych, uszczegóławianą w miarę postępu prowadzonych prac. Wykorzystanie tej skarbnicy wiedzy na potrzeby metod szacowania kosztów pozwoli (już na wczesnych etapach realizacji przedsięwzięcia informatycznego) ustalić przybliżoną wielkość planowanego systemu oraz uniezależnić metodę od subiektywnych opinii eksperta. Może okazać się realne, że przeprowadzanie wczesnej i obiektywnej estymacji nakładów bez dodatkowego nakładu pracy, związanego z gromadzeniem informacji oraz bez konieczności ich dodatkowej standaryzacji. Jednocześnie jest to szansa na znaczne zwiększenie automatyzacji omawianego procesu estymacji.

4. ZAŁOŻENIA DO BUDOWY MODELU NAKŁADÓW

W poszukiwaniu rozwiązania problemu estymacji wielkości nakładów tworzenia informatycznego produktu obiektowego, zaplanowano przeprowadzenie badań w następującej kolejności:

- próba zastosowania metody punktów funkcyjnych do mierzenia rozmiaru modelu obiektowego,
- skonstruowanie nowego modelu nakładów.

4. 1. Próba adaptacji metody *FP*

Analizę możliwości adaptacji istniejących metod do estymacji wielkości nakładów tworzenia programu przy podejściu obiektowym przeprowadzono na przykładzie metody punktów funkcyjnych (szczegółowe omówienie metody znajduje się w punkcie 3.2.1), znanej pod ogólną nazwą *FP*. Podstawowym elementem w tej metodzie jest pomysł przedstawienia rozmiaru tworzonego systemu za pomocą punktów funkcyjnych. W niniejszej próbie adaptacji odwołano się do metody *MARK II* (metoda ta została również przedstawiona w punkcie 3.2.), bliższej aktualnym technologiom i wymaganiom niż jej pierwowzór - metoda *FPA* Albrechta. Daje ona możliwość automatycznego obliczania „surowych” punktów funkcyjnych na podstawie modelu związków encji¹⁶, którego elementy również występują w modelu obiektów technologii obiektowej. Próba adaptacji została przeprowadzona w dwóch krokach. W pierwszym, przeanalizowano możliwość bezpośredniego zastosowania tej metody do produktu tworzonego w technologii obiektowej, przez poszukiwanie analogii z technologią strukturalną. W drugim kroku zastosowano postępowanie odwrotne, badając możliwość uwzględnienia w metodzie *FP* cech produktu obiektowego przez stosowane miary obiektowe. W rozważaniach niżej przedstawionych rozpatrzono również następujące elementy: możliwość przeprowadzenia estymacji na wczesnym etapie cyklu tworzenia oraz możliwość automatycznej estymacji.

¹⁶ Model związków encji w technologii strukturalnej stanowi, wraz z diagramem hierarchii funkcji oraz diagramem przepływu danych, model wymagań użytkownika. Został wprowadzony w drugiej połowie lat siedemdziesiątych przez P. Chena [CHE76]. Encja jest pojęciem odzwierciedlającym każdy, istotny dla systemu byt. Często spotykanym określeniem, zastępującym nieznane w języku polskim słowo *encja*, jest po prostu obiekt.

Krok I. Analiza możliwości zastosowania metody MARK II do produktu obiektowego

Dyskusję przeprowadzono w aspekcie wytypowanych dla metody punktów funkcyjnych elementów składowych produktu - typów funkcyjnych oraz charakterystyki złożoności przetwarzania.

Typy funkcyjne

W zmodyfikowanej metodzie FP dokonuje się podziału składników systemu na trzy podstawowe typy: dane wewnętrzne i zewnętrzne, transakcje wejściowe i wyjściowe oraz zapytania. Identyfikacja typów transakcji oraz zapytań odbywa się na podstawie specyfikacji funkcji, a danych na podstawie modelu logicznego danych. Kompletne informacje do zbudowania tych modeli są dostępne po wykonaniu projektu systemu. Przykład analizy typów funkcyjnych pokazano w załączniku 5.

Różne definicje obiektu

W technologii obiektowej podstawowym elementem, do którego odnosi się proces tworzenia produktu informatycznego jest klasa lub obiekt. Model logiczny programu, który składa się z klas lub obiektów, zarówno na poziomie konceptualnym, na poziomie specyfikacji, jak i na poziomie implementacji stanowi zintegrowany model wymaganego przetwarzania, a klasa odpowiada elementarnemu fragmentowi tego przetwarzania. Klasa i obiekt przedstawia, używając terminologii podejścia strukturalnego, zarówno dane jak i funkcje, natomiast encja, występująca w metodzie FP odnosi się tylko do danych. Zatem nie jest możliwe przeprowadzenie bezpośredniego odwzorowania pomiędzy klasą czy obiektem a encją.

Wyróżnianie typów wejściowych i wyjściowych

Funkcjonalne wymagania użytkownika wobec systemu, na poziomie logicznym, według Symonsa [SYM88, s. 4] są postrzegane jako wejściowe i wyjściowe typy transakcji.

W metodykach obiektowych funkcjonalność systemu jest przedstawiana w różnych aspektach za pomocą:

- klas i obiektów w modelu klas i obiektów,
- przypadków użycia w modelu analizy Jacobsona,
- zdarzeń i stanów w diagramie dynamicznym,

- przepływów danych i procesów w modelu funkcjonalnym.

Najbliższy koncepcji transakcji wejściowych i wyjściowych jest model wymagań Jacobsona, w którym model *przypadków użycia* bezpośrednio odnosi się do logicznych typów transakcji. Nie definiuje się w nim jednak podziału na wejścia i wyjścia, chociaż na diagramie przedstawiającym granice systemu zaznacza się strzałkami kierunki przepływu informacji. Są to przepływy pomiędzy *aktorami*, jako elementami zewnętrznymi wobec systemu a *przypadkami użycia*, jako elementami wewnętrznymi systemu. Kierunki te pokazują, które przepływy są wejściowe, a które wyjściowe, jednak nie są one zapamiętywane w repozytorium projektu. W celu ewentualnego zliczania *przypadków użycia* według kryterium podziału na wejściowe i wyjściowe, należałoby przeprowadzić czasochłonną analizę tego modelu za pomocą technik analizy obrazu. Jest to problem natury formalnej, związany z określoną notacją, który mógłby zostać rozwiązany.

Problemy z określeniem złożoności przetwarzania

Zgodnie z hipotezą Symonsa [SYM88, s. 5], miarą złożoności przetwarzania w metodzie punktów funkcyjnych jest „liczba wywoływanych¹⁷ typów encji przez określony typ transakcji”. Zgodnie z powyższym, złożoność transakcji wylicza się przez sumowanie konkretnych elementów encji, które występują w wykonywanych transakcjach, tj. atrybutów. Wykonanie, na tej podstawie, obliczeń złożoności wszystkich transakcji, decyduje też o możliwości ustalenia rozmiaru systemu ustalanego w punktach funkcyjnych. Takie postępowanie wiąże się z precyzyjnym ustaleniem:

- rodzaju encji : pierwszorzędne dla systemu, kontrolne czy tylko wyjściowe (na których nie są wykonywane żadne obliczenia),
- rodzaju elementów danych (np. atrybuty encji, atrybuty interfejsu, parametry operatorskie),
- liczby atrybutów, które uczestniczą w konkretnej transakcji.

Poszukiwanie możliwości uzyskania tych danych w technologii obiektowej prowadzi najpierw do stwierdzenia istniejącej, bezpośredniej analogii pomiędzy atrybutami encji a atrybutami klasy i obiektów. Jednak wiedza o tym, które dokładnie atrybuty są przetwarzane przez daną operację jest uzyskiwana przeważnie na etapie projektowania lub dopiero implementacji. W szczególnych przypadkach jest otrzymywana w postaci specyfikacji na

¹⁷ Sposobów wywoływania encji jest tyle, ile uznanych za standardowe rodzajów możliwości przetwarzania encji: tworzenie, aktualizowanie, usuwanie i odczytywanie. Jednak, w przytoczonej definicji nie uzależnia się złożoności od rodzaju przetwarzania.

etapie analizy. Specyfikacja operacji jest jednak wykonywana tylko dla złożonych algorytmów. Zatem otrzymanie danych na temat wykorzystywania atrybutów przez wszystkie operacje jest trudna do uzyskania na etapie analizy.

W modelach przedstawiających funkcjonalność systemu na etapie analizy (również wspomnianym wyżej modelu *przypadków użycia* Jacobsona) powstaje tekstowy opis sposobu realizacji przetwarzania, ale bez formalizacji, która umożliwiałaby jego automatyczną analizę. Zarówno typy danych jak i sposoby ich wykorzystywania są zapamiętywane pod postacią wyrażen języka naturalnego lub pseudokodu¹⁸. Możliwym, ale trudnym i bardzo czasochłonnym zadaniem byłoby automatyczne wyszukiwanie rzeczowników¹⁹ występujących w takim tekście, a następnie przeprowadzenie na tym zbiorze rzeczowników automatycznej selekcji na wyróżnione wyżej rodzaje encji i elementów danych.

Na podstawie przeprowadzonej analizy w podsumowaniu stwierdza się:

- Zasadniczą różnicę w sposobie definiowania danych i funkcji w obu metodykach.
- Brak możliwości pozyskania wszystkich szczegółowych danych potrzebnych do ewentualnego zastosowania metody FP na wczesnym etapie tworzenia systemu, tj. na etapie analizy.

Zastosowanie metody punktów funkcyjnych do programu obiektowego na etapie analizy wymagałoby istotnej modyfikacji, dotyczącej przede wszystkim sposobu wyróżniania typów funkcyjnych. Należałoby wówczas przeprowadzić ponowne badania w celu wyznaczenia nowych wartości wagowych, odpowiadających wyróżnionym w metodzie FP poziomom złożoności.

Krok II. Analiza możliwości uwzględnienia w metodzie FP cech produktu obiektowego

Analiza możliwości uwzględnienia miar obiektowych za pomocą punktów funkcyjnych została przeprowadzona względem sześciu obiektowych metryk z zestawu Chidambera i Kemerera. W celu uporządkowania obszaru własności i zachowania klas i obiektów wprowadzono klasyfikację tych własności i zachowań względem dwóch różnych poziomów abstrakcji, na których są one odkrywane²⁰: poziom pierwszy odpowiada klasie i

¹⁸ Model algorytmu może być zaprezentowany przy użyciu schematu graficznego, konkretnego języka programowania, ale też i za pomocą pseudokodu. Pseudokod jest zbiorem połączonych wyrażen języka naturalnego i wybranych struktur wyrazowych języka angielskiego.

¹⁹ Zgodnie z zasadą identyfikacji encji i ich własności, potencjalnymi elementami są właśnie rzeczowniki.

²⁰ W związku z pojęciem własności obiektu istnieje różnica w jego interpretacji. W teorii Bunga własności bytu są reprezentowane przez atrybuty a funkcjonalność związana z bytem jest odrębnym pojęciem. W technologii obiektowej atrybuty z ich metodami, reprezentującymi funkcjonalność są własnościami obiektu, metody są częścią deklaracji klasy.

obiektom, a poziom drugi odpowiada związkom występującym pomiędzy klasami i obiektami, w skład których wchodzi także związki agregacji i dziedziczenia. Pojęcie tych poziomów, zgodnie z definicją obiektu, dotyczy zarówno cech obiektu, reprezentowanych przez atrybuty, jak i zachowania obiektów, reprezentowanego przez operacje.

1. Metryka WMC

Przedstawia ona złożoność klasy, określaną przez licznosc zbioru operacji, zadeklarowanych w tej klasie. Miara tak definiowanej złożoności zaliczana jest do wewnętrznych miar modułowych i przedstawia również funkcjonalność tej klasy.

W metodzie FP funkcjonalność jest mierzona liczbą transakcji, a ich złożoność jest określana za pomocą liczby danych, wywoływanych przez transakcję. Punkty funkcyjne dla poszczególnych typów transakcji w tej metodzie są sumowane zgodnie z ustalonymi empirycznie współczynnikami równania (3.8). W celu zastosowania tej samej procedury w odniesieniu do obiektu czy klasy należałoby wprowadzić do tego równania liczbę operacji na miejsce liczby transakcji. Jednak, ze względu na to, że w tym równaniu występują współczynniki, które zostały eksperymentalnie ustalone dla środowiska strukturalnego, należałoby przeprowadzić ponowne badania w celu ustalenia odpowiednich współczynników dla środowiska obiektowego. Pojęcie złożoności w obu przypadkach odnosiłoby się do różnych wielkości. W metodzie FP określa się ją przez atrybuty, natomiast zgodnie z miarą *WMC* wystarczającą miarą złożoności jest liczba operacji bez odwoływania się do liczby atrybutów.

2. Metryka DIT

Głębokość dziedziczenia jest miarą dla klasy, która przedstawia długość ścieżki od węzła do korzenia drzewa (w przypadku wielokrotnego dziedziczenia długość maksymalną). Zgodnie z definicją *DIT* jest dalszym ciągiem opisu własności i zachowania klasy, ale dziedziczonych od klas przodków.

W metodzie FP nie ma odpowiednika dla tej miary, gdyż w metodykach strukturalnych nie występuje cecha dziedziczności. Chociaż dziedzicznie podnosi funkcjonalność klasy odkrywaną na drugim poziomie abstrakcji, to również, jako mechanizm zwiększający obszar własności, powinno znaleźć swoje odzwierciedlenie w procesie mierzenia.

3. Metryka NOC

Metryka ta jest równa liczbie najbliższych podklas podległych danej klasie w drzewie hierarchii klas. Odnosi się, podobnie jak Metryka *DIT*, do pojęcia zakresu własności i zachowania klasy. Pokazuje, jak wiele podklas może dziedziczyć metody klas rodziców.

Jest to druga metryka, która odwołuje się do cechy dziedziczności i przypisuje do klasy własności i zachowania drugiego poziomu abstrakcji. Miary tych własności i zachowań, tak jak miary własności i zachowań pierwszego poziomu, są również zaliczane do miar modułowych. Rozważania dla tej miary są identyczne, jak dla miary *DIT*, ale wskazują na dodatkowy kierunek powiększania obszaru własności i zachowań. Jest to czynnik także wymagający uwzględnienia przy ustalaniu rozmiaru programu jako czynnik, który wpływa na rozmiar tworzonego produktu, a nie posiada swojego odpowiednika w rozważanej metodzie FP.

4. Metryka CBO

Metryka *CBO* jest miarą klasy i pokazuje z jaką liczbą innych klas jest ona połączona. Odzwierciedla połączenia obiektu z jednej klasy z obiektem innej klasy. Dyskusja tej miary została zredagowana razem z miarą piątą.

5. Metryka RFC

Metryka *RFC* jest zbiorem odpowiedzi dla klasy, który stanowi suma zbioru metod tej klasy i metod wywoływanych przez te metody z innych klas i z tej samej klasy. Jest to Metryka potencjalnej komunikacji pomiędzy daną klasą a innymi klasami.

Metryka czwarta i piąta, podobnie jak metryki od pierwszej do trzeciej, odnoszą się do obszaru własności klasy i zachowania, z tym, że: czwarta metryka dotyczy własności i zachowania drugiego poziomu abstrakcji, które stanowią miarę modułową i międzymodułową, a dotyczy też własności i zachowania drugiego poziomu abstrakcji, ale tylko tych, które stanowią grupę miar międzymodułowych. Na własności drugiego poziomu abstrakcji składają się nie tylko związki dziedziczenia, ale też agregacji i asocjacji. Powiązania te wskazują na potencjalne możliwości dostępu do metod i atrybutów jednej klasy z wnętrza drugiej klasy i stąd mają wpływ na obszar dostępnych własności. Klasa może mieć związki asocjacji typu "do" (ang. fan – out) i "od" (ang. fan – in), a każde z nich, dla obu klas w tym samym związku, jest po jednej stronie "do" a po drugiej "od". Powiązania typu "do" są związane z potrzebą użycia własności tej drugiej klasy (w trakcie implementacji jest to

realizowane np. przez klasę „friend” w terminologii języka C++). Powiązanie takie zwiększa liczbę własności jednej klasy, które nie są tworzone dwa razy w każdej z nich, lecz tylko w jednej. Przez to, że zwiększa się możliwość działania jednej klasy przez udostępnianie własności drugiej, związek ten musi być odzwierciedlony w rozmiarze klasy, która jest w związku po stronie "do", to znaczy tej, która korzysta z własności. Wielkość powiązania powinna odzwierciedlać, czy dotyczy ono całej klasy czy tylko części jej własności (inaczej niż w związkach dziedziczenia, gdzie powiązanie dotyczy zawsze całej klasy). Tego typu powiązania występują również w modelu związków encji w metodykach strukturalnych. Złożoność programu wynikająca z tego typu powiązań w metodzie FP jest uwzględniona za pomocą liczby typów encji występujących na ścieżce dostępu do danych wywoływanych w transakcjach wejściowych lub wyjściowych oraz liczbą typów danych wejściowych i wyjściowych. W metrykach obiektowych jest to liczba atrybutów i/lub operacji wywoływanych przez jedną klasę z innej klasy, bez dublowania wywołań tej samej operacji przez różne operacje z tej samej klasy.

W metodykach obiektowych związki występujące pomiędzy klasami i obiektami są przedstawiane w modelu obiektów, podobnie jak w metodykach strukturalnych (oprócz mechanizmu dziedziczenia). Stąd zależności te mogłyby być przedstawione podobnie – zwiększając liczbę danych do których może odwołać się transakcja. Jednak ze względu na różne koncepcje obiektów w tych obu technologiach nie są one uwzględniane w metodzie FP.

6. Metryka LCOM

Metryka *LCOM* jest liczbą par metod, których podobieństwo jest zbiorem pustym minus liczba par metod, których podobieństwo nie jest zero. Metryka *LCOM* jest związana ze zmiennymi i metodami klasy. Jako metryka spójności, zakwalifikowana do miary modułowej, nie dotyczy określania zakresu własności klas i obiektów, lecz zwraca uwagę na jedną z cech jakościowych budowanego modelu klas. Wykryty brak spójności w klasie może sugerować, że przypisano do niej takie własności, które należą do odrębnej klasy bądź klas i wskazuje, że zachodzi potrzeba rozbicia tej klasy. Ze względu na to, że spójność ta jest mierzona przez odwoływanie się metod do atrybutów, a wiedza o tym, do których atrybutów odwołuje się dana metoda na etapie wczesnej analizy jest niedostępna, metryka ta nie może być uwzględniona w badanym podejściu.

Po przeprowadzeniu wyżej przedstawionej analizy okazało się, że metryki obiektowe postrzegają produkt jako zbiór powiązanych klas/obiektów o określonych własnościach i zachowaniach, natomiast w metodzie *FP* produkt składa się z trzech (lub pięciu) typów

elementów składowych, którym generalnie nie odpowiadają definicje tych metryk. Jedynie metryki czwarta i piąta, ale w odniesieniu do powiązań różnych niż dziedziczenie, w podobny sposób ujmują tę część rozmiaru produktu, jak w innych miarach produktu strukturalnego, np. liczba cyklomatyczna, czy metryka „fan – in”. Na koniec tej analizy stwierdzono:

- Brak możliwości uwzględnienia w metodzie FP cech obiektowych programu, które, jak wyżej opisano, w istotny sposób wpływają na jego rozmiar, wyklucza możliwość bezpośredniego zastosowania tej metody w odniesieniu do rozmiaru produktu obiektowego.
- Istnieje możliwość adaptacji tej metody do technologii obiektowej po wykonaniu jej zasadniczej modyfikacji związanej z klasyfikacją typów funkcyjnych.

4. 2. Przesłanki do ustalenia miar programu obiektowego

Przeprowadzony przegląd istniejących metod szacowania kosztów (rozdział 3.2) pozwala stwierdzić, że ich zastosowanie jest możliwe dopiero na etapie projektowania systemu, czyli późniejszego etapu cyklu życia. W praktyce wymaga się, by szacowanie nakładów i przewidywanie terminów na potrzeby budowy planu było wykonywane jak najwcześniej, a przynajmniej w wyniku prac uzyskanych na etapie analizy wymagań. Biorąc pod uwagę potrzeby dotyczące wczesnej estymacji nakładów tworzenia produktu obiektowego oraz brak możliwości bezpośredniego zastosowania istniejących dla technologii strukturalnej metod estymacji (co wykazano w rozdziale 4.1), sformułowano podstawowe założenia dla konstrukcji nowego modelu nakładów w postaci następujących przesłanek:

- 1) Estymacja nakładów na wczesnym etapie tworzenia programu.
- 2) Uwzględnienie zasadniczych cech technologii obiektowej: dziedziczenie i możliwość wielokrotnego użycia.
- 3) Konieczność sformalizowanego sposobu kolekcjonowania i automatycznej analizy dokumentacji projektowej.

4. 2. 1. Estymacja nakładów na wczesnym etapie tworzenia

Proces mierzenia tworzonego produktu i wytypowanie miar jego wielkości zależy kolejno od następujących wymiarów: celu mierzenia, etapu cyklu życia oraz poziomu abstrakcji. Oznacza to, że dla każdego z nich mogą być wytypowane inne zestawy miar.

Celem rozważanych w niniejszej pracy możliwości mierzenia produktu jest potrzeba szacowania wielkości jednego z podstawowych zasobów procesu tworzenia, jakim jest nakład

pracy, wyrażony za pomocą jednostek czasu. Potrzeba szacowania wielkości nakładów pracy wskazuje na uzależnienie wyboru miar od możliwie najwcześniejszego etapu cyklu życia.

Istnieją różne podejścia do realizacji etapów cyklu życia, ale podstawowa koncepcja w każdej metodyce jest taka sama:

- We wszystkich metodach stosuje się koncepcję budowy modelu, chociaż używa się niekiedy alternatywnych nazw wobec poszczególnych jego składników.
- Końcowy produkt informatyczny powstaje w kilku etapach i na każdym z nich istnieje wiedza o nim przedstawiona za pomocą odpowiedniego modelu.
- Modele należą do produktów kluczowych realizowanego przedsięwzięcia [BAR90], [HOL94] i stanowią podstawę w negocjacjach prowadzących do uzyskania akceptacji ze strony klienta, zaś dla menedżera przedsięwzięcia są przedmiotem kontroli realizowanych zadań [HOL98].

W analizie nakładów tworzenia programu uwzględniono cztery etapy cyklu życia, które wyróżnia się w procesie tworzenia programu: etap analizy, projektowania, implementacji i testowania.

4. 2. 2. Etap cyklu życia jako podstawowy wymiar

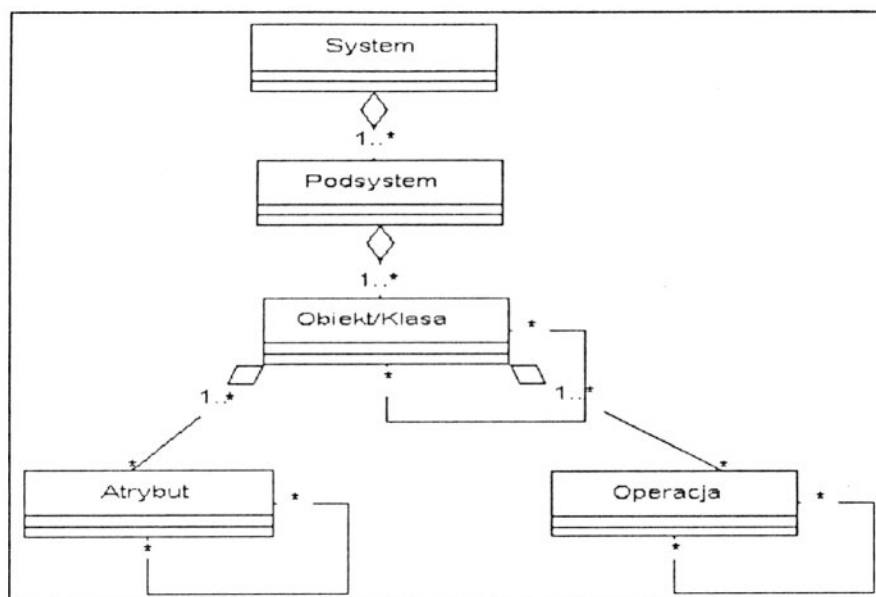
Według Hendersona - Sellersa i Fentona [HEN93 - 96], [FEN96] zajmujących się zagadnieniem mierzenia produktu informatycznego, ze względu na różny poziom szczegółowości wiedzy zdobywanej na każdym z etapów cyklu życia, produkty te charakteryzują się innymi cechami. W związku z tym, mogą być, wobec nich, stosowane różne miary. Wówczas, rozmiar programu dla każdego z etapów cyklu życia jest opisywany za pomocą jednego z różnych postaci równania (2.29). Zestaw składników występujących w tych równaniach dobierany jest zależnie od możliwości ich uzyskania na każdym z tych etapów²¹. Zgodnie z takim podejściem, etapy cyklu życia stanowią podstawowy wymiar przy określaniu sposobu mierzenia produktu informatycznego.

4. 2. 3. Poziom abstrakcji jako kolejny wymiar

W połączeniu z celem mierzenia i etapem cyklu życia, poziom abstrakcji pozwala na wytypowanie konkretnych miar tworzonego produktu informatycznego. Poziom ten dotyczy zarówno produktu końcowego, jak i jego modeli. Przy podejściu obiektowym wyróżnia się

²¹ Uważa się [HEN96], że na szczegółowym etapie projektowania początkowa miara rozmiaru i złożoności może być wyprowadzona, przynajmniej jako pierwsza estymacja na podstawie, np. liczby atrybutów i liczby metod przypadających na klasę.

następujące poziomy abstrakcji²², które można przedstawić w układzie hierarchicznym (rys. 6): system, podsystem, klasa i obiekt, atrybut i operacja.



Rysunek 6. Hierarchiczny układ poziomów abstrakcji występujących w technologii obiektowej. Źródło: [modyfikacja na podstawie HEN96].

Oznacza to, że miary mogą odwoływać się do odpowiednich poziomów abstrakcji, jak np. liczba podsystemów w systemie, liczba klas w podsystemie, czy liczba metod w klasie, ale też może istnieć możliwość ich sumowania pomiędzy tymi poziomami. Wówczas zagadnienie mierzenia jest związane z potrzebą przeprowadzenia identyfikacji w celu uzgodnienia, które z ustalonych miar przy przechodzeniu z niższego na wyższy poziom abstrakcji mogą być sumowane (np. przy odpowiedzi na pytanie: czy liczba operacji w systemie jest sumą operacji wszystkich klas tego systemu). Stwierdzenie możliwości sumowania miar przesądza o ich wykorzystaniu do obliczania całkowitej wielkości samego produktu oraz jego modelu. W odniesieniu do abstrakcji istniejących na różnych poziomach mogą być zastosowane miary dotyczące charakterystyki istniejących pomiędzy nimi układów albo też ich cech wewnętrznych, np.:

- liczba powiązań dla obiektu,
- spójność dla klasy,
- % klas z wielokrotnym użyciem,
- głębokość drzewa dziedziczenia dla klasy.

²² W różnych metodykach w odniesieniu do tych samych elementów stosowane są niekiedy inne pojęcia. W tabeli 47 (załącznik 2) zamieszczono odpowiedniki tych pojęć dla wybranych metod.

Przykłady miar na poziomie wymiaru klasy: liczba metod, liczba powiązań pomiędzy tą klasą a innymi, głębokość drzewa dziedziczenia dla klasy.

4. 2. 4. Model etapu analizy w estymacji nakładów

Ze względu na to podstawowy czynnik, od którego zależy planowana wielkość nakładów pracy - wielkość produktu oraz uzyskanie jej rzeczywistej wartości dopiero na końcu procesu wytwarzania, a nie na etapie analizy, zastąpiono go miarą wielkości modelu tego produktu. Model tworzony na etapie analizy jest z jednej strony odzwierciedleniem wymagań użytkownika wobec przyszłego systemu, a z drugiej strony ma stanowić wystarczający opis dla powstania produktu końcowego. Warunkiem podstawowym dla przyjęcia modelu analizy jako źródła wiedzy o zakresie tworzonego systemu jest wymaganie, by zakres planowego systemu w trakcie dalszych prac nie ulegał zasadniczym zmianom. Wówczas należy też oczekiwać, że miary modyfikowanego modelu na kolejnych etapach tworzenia są też kolejnymi przybliżeniami rozmiaru produktu końcowego. Problem oszacowania wielkości produktu końcowego, jakim jest program, został zatem sprowadzony do zagadnienia pomiaru wielkości jego modelu. Przy tak sformułowanym zagadnieniu zachodzi potrzeba zapewnienia:

- konstrukcji modelu według sformalizowanych zasad,
- kontrolowanej zmienności zakresu wytyczonego dla modelu etapu analizy w porównaniu do zakresu produktu końcowego,
- możliwości wykonywania porównań wielkości modyfikowanych modeli na kolejnych etapach.

Spełnienie wymienionych wymagań jest możliwe jedynie przy realizacji procesu tworzenia systemu informatycznego zgodnie z określoną metodyką i wspomaganie narzędziami typu CASE. Badania [JAC92, s. 437] przeprowadzone w roku 1990 pokazują, że w krajach Europy Zachodniej ponad połowa organizacji zajmujących się tworzeniem oprogramowania prowadzi swoje prace według formalnie zdefiniowanych metod. Brak jest natomiast danych o organizacjach, które korzystałyby z jakichkolwiek formalnych sposobów mierzenia procesu wytwarzania, czy też mierzenia samego produktu. Tak szerokie stosowanie narzędzi potwierdza słuszność przyjętych wymagań wobec stosowania formalnych metod, a brak danych na temat stosowania metod pomiaru wskazuje na potrzebę prowadzenia prac badawczych związanych z zagadnieniem mierzenia. Stosowane aktualnie w technologii obiektowej różne podejścia do realizacji etapu analizy wymagań (załącznik 7) częściowo

utrudniają wyznaczenie niezależnych od rodzaju metodyki miar modelu produktu obiektowego, ale też nie wykluczają takiej możliwości. Zamieszczone w tabeli 63 (załącznik 9) porównanie kilku metodyk obiektowych wskazuje na istnienie analogii pomiędzy większością elementów występujących w poszczególnych koncepcjach modelowania. Wobec powyższego, za decydujące o wyborze metodyki, przyjęto następujące trzy kryteria:

- 1) Konieczność budowy na etapie analizy takiego modelu wymagań, który gwarantuje uzyskanie opisu pełnego zakresu funkcjonalności planowanego systemu zarówno z punktu widzenia użytkownika jak i twórcy tego systemu.
- 2) Możliwość odwoływania się do tego modelu i jego elementów przez kolejne etapy cyklu tworzenia: analizę, projektowanie, implementację i testowanie.
- 3) Wybór takiej metodyki, która odpowiadałaby regułom zawartym w najnowszym standardzie UML.

Miary rozmiaru programu zostały wytypowane dla modelu analizy. Elementy tego modelu występują na wymienionych wyżej etapach tworzenia w postaci modeli kolejno modyfikowanych, które pozwalają na wykonywanie pomiędzy nimi istotnych analiz porównawczych.

4. 2. 5. Wpływ elementów obiektowych na rozmiar programu

Spośród elementów technologii obiektowej rozważono cechy programu oraz atrybuty procesu tworzenia tego programu.

4. 2. 5. 1. Cechy programu obiektowego i ich wpływ na jego rozmiar

Do typowych cech produktu obiektowego zalicza się: modularyzację, abstrakcję i dziedziczenie. Wysoki stopień modularyzacji programów obiektowych pozwala na:

- zmniejszenie złożoności rozwiązywanego problemu przez jego dekompozycję na części,
- bezkonfliktowe wprowadzanie zmian przez możliwość niezależnej modyfikacji jego części,
- ponowne wykorzystanie wcześniej wykonanych elementów przez możliwość tworzenia jego części w różnych okresach czasu,
- sterowanie dostępnymi zasobami przez możliwość tworzenia jego części w różnych okresach czasu oraz przez różne osoby.

Wymienione korzyści wyraźnie wskazują na istotny udział modularyzacji w kształtowaniu się kosztów tworzenia programu. Podział programu na moduły zwiększa możliwość ponownego

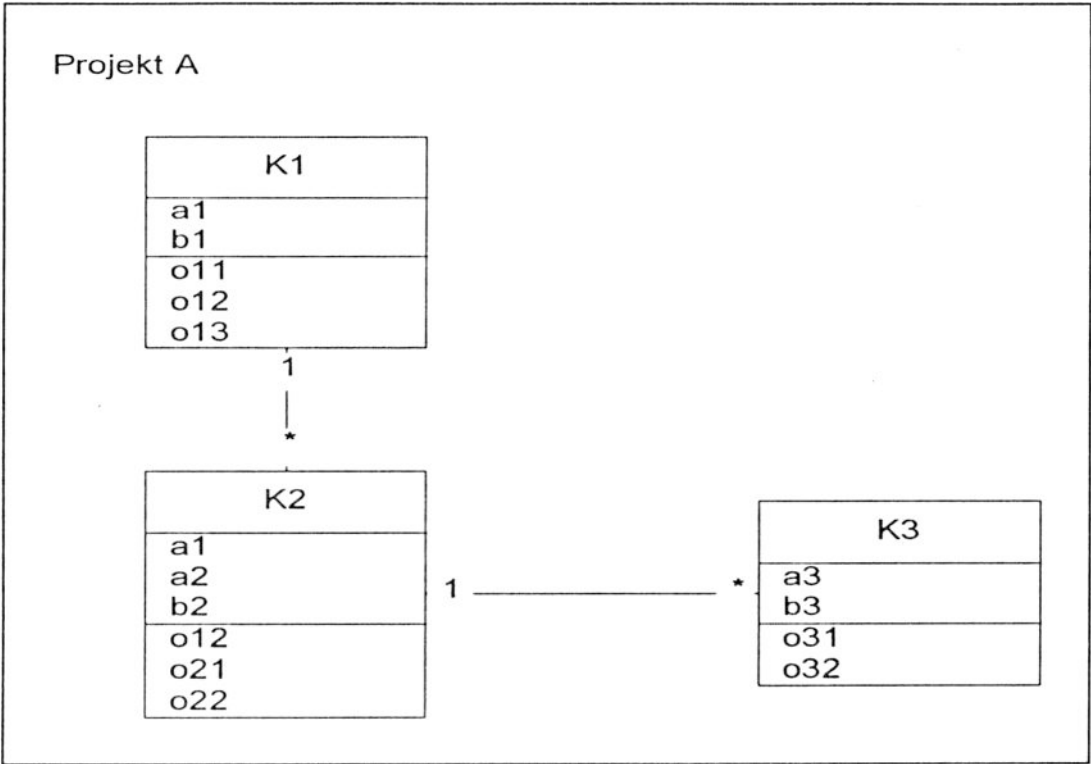
wykorzystania w tej samej aplikacji raz utworzonego kodu. Projekt tego podziału jest dokonywany jednak dopiero na etapie projektowania, ponieważ jest częścią ustalania fizycznej architektury systemu. Dlatego też na etapie analizy nie może zostać uwzględniony. Rola abstrakcji została rozważona przy wielowymiarowym podejściu do mierzenia (punkt 4.2.3 tego rozdziału). Istnieje jednak jeszcze drugi aspekt tej cechy o istotnym znaczeniu dla wielkości nakładów tworzenia programu, a związany z mechanizmem dziedziczenia. Pojęcie abstrakcji w procesie tworzenia przy podejściu obiektowym zostało, w porównaniu do podejścia strukturalnego, w istotny sposób poszerzone. Zwiększono poziom złożoności abstrakcji, który umożliwia klasyfikację obiektów świata rzeczywistego zarówno ze względu na podobieństwo cech statycznych (własności obiektu), jak i dynamicznych (zachowanie obiektu). Skutki takiego podejścia są widoczne w rozmiarze programu przez mechanizm dziedziczenia. Dziedziczenie bowiem pozwala na przedstawienie konceptualnego modelu wymagań użytkownika w postaci hierarchii obiektów/klas, utworzonych na bazie obiektów/klas wcześniej zidentyfikowanych. Taki model umożliwia, już na etapie koncepcji, przewidzieć możliwość dwukrotnego lub więcej wykorzystania (np. na etapie kodowania) tych samych deklaracji własności i zachowań. Dlatego też, wielokrotność użycia powinna znaleźć swoje odbicie również w postaci czynnika, od którego zależy wielkość nakładów.

Analiza wpływu mechanizmu dziedziczenia na rozmiar modelu programu

Mechanizm dziedziczenia, przy spojrzeniu na system z poziomu liczby deklaracji wewnątrz klas, zmniejsza jego rozmiar. Ale, patrząc na jego funkcjonalność z punktu widzenia użytkownika, program musi wykonywać tę samą funkcjonalność bez względu na występowanie dziedziczenia. Stąd, pojawia się istotny dla procesu mierzenia element punktu widzenia rozmiaru programu - funkcjonalność: oczekiwana przez zleceniodawcę i przyszłego użytkownika, wykonywana przez program i fizycznie tworzona przez twórców.

Zakłada się, że dwa pierwsze rodzaje funkcjonalności pokrywają się, gdyż program powinien działać zgodnie z wymaganiami zleceniodawcy. Pozostają zatem dwa rozmiary tego samego programu, które mogą być przedstawiać różne wartości dla ustalonych miar. Możliwość istnienia dwóch wielkości rozmiarów tego samego programu z powodu różnych technik tworzenia wskazuje na konieczność uwzględnienia tej specyfiki również w procesie badania zależności nakładów od rozmiaru tworzonego programu. Różnice pomiędzy tymi rozmiarami wynikają przede wszystkim z przyjętej technologii i metodyki realizacji. Przy podejściu obiektowym mechanizm dziedziczenia w szczególny sposób wpływa na tą różnicę, gdyż pozwala na wielokrotność użycia większą niż przy podejściu tradycyjnym. Zatem zachodzi

konieczność odzwierciedlenia wpływu tego mechanizmu na oszacowanie wielkości czasu tworzenia już na wczesnym etapie.



Rysunek 7. Projekt A bez stosowania mechanizmu dziedziczenia. Źródło: [opracowanie własne].

Mechanizm dziedziczenia w technologii obiektowej jest związany z obiektem/klasą i jego własnościami. Deklarowane własności obiektu/klasy – operacje/metody, odzwierciedlają pojęcie funkcjonalności. Cecha dziedziczenia pozwala jednej klasie korzystać ze wszystkich własności drugiej klasy na zasadzie związku pomiędzy nimi typu rodzic - potomek. Oznacza to, że raz zdefiniowane własności jednej klasy mogą być ponownie wykorzystane, bez potrzeby ich powtórznego definiowania. Możliwość ta nie może być bez wpływu na rozmiar całego programu, badanego na określonym poziomie abstrakcji. Istotą problemu, jaki tutaj powstaje, jest ustalenie zarówno poziomu jak i kierunku wpływu tej możliwości na rozmiar.

Z intuicyjnego punktu widzenia można stwierdzić, że rozmiar całego programu realizowanego z wprowadzeniem możliwości dziedziczenia będzie mniejszy niż programu bez tej możliwości. W celu teoretycznego potwierdzenia tej zależności przeprowadzono następujące porównanie dwóch projektów.

Przykład

Program realizowany w projekcie A (rys. 7) składa się z trzech klas połączonych związkami asocjacji. Przyjęto, że wielkość każdej z klas wynikająca z posiadanych atrybutów, metod i

związków wynosi c_i z odpowiednim indeksem i –tej klasy. Zatem klasa $K1$ posiada wielkość $c1$, klasa $K2$ - $c2$, klasa $K3$ - $c3$. Korzystając z założenia, że wielkość programu jest sumą wielkości wszystkich jego klas²³, wielkość programu A będzie równa:

$$WA = c1 + c2 + c3 \quad (4.1)$$

Projekt B (rys. 8) powstał z przekształcenia projektu A przez wprowadzenie możliwości dziedziczenia własności klasy. Utworzono nową klasę $K0$, która stała się superklasą o własnościach wspólnych, występujących, jak się okazało, w klasie $K1$ i klasie $K3$, a własności różne pozostały w tych obu klasach. Klasa $K2$ pozostała bez zmian, ze związkiem asocjacji z klasą $K1$ i klasą $K3$. Po dokonaniu tej operacji, zwanej generalizacją, wielkości klas można opisać następująco:

- klasa $K0$ ma wielkość $c0 = c11$,
- klasa $K1$ ma wielkość $c1' = c1 - c11$,
- klasa $K2$ ma wielkość $c2' = c2$,
- klasa $K3$ ma wielkość $c3 - c11$.

Zachodzi pytanie, jaka jest wielkość CB projektu B ? Korzystając z założenia o sumie wielkości klas otrzymujemy:

$$CB = c0 + c1' + c2' + c3 + d, \quad (4.2)$$

gdzie:

d - wielkość dodanego związku generalizacji.

Po podstawieniu odpowiednich wielkości za $c1'$ i $c2'$ otrzymuje się:

$$CB = c11 + (c1 - c11) + c2 + (c3 - c11) + d = c11 + c1 + c2 + c3 - 2*c11 + d \quad (4.3)$$

i ostatecznie:

$$CB = (c1 + c2 + c3) - c11 + d. \quad (4.4)$$

²³ Z zasady addytywnej agregacji dwu (lub więcej) bytów wynika, że kombinacja dwóch klas jest klasą, której własności są sumą logiczną własności klas składowych.

Niech $X = \langle x, p(x) \rangle$ i $Y = \langle y, p(y) \rangle$ będą dwoma klasami obiektów, to $X+Y$ jest definiowane jako $\langle z, p(z) \rangle$ gdzie z jest znakiem, za pomocą którego jest przedstawiana suma $X+Y$ oraz $p(z)$ jest dane jako:

$$p(z) = p(x) \cup p(y).$$

Przykład:

Jeśli klasa $k1$ ma własności (np. metody i wystąpienia zmiennych) a, b, c, d i klasa $k2$ ma własności a, l, m, n to klasa $k1+k2$ ma własności a, b, c, d, l, m, n .

Jeśli klasy $k1$ i $k2$ obie mają identyczne własności a, b, c, d to klasa $k1+k2$ będzie miała te same własności a, b, c, d .

Zatem

$$CB = CA - c11 + d, \quad (4.5)$$

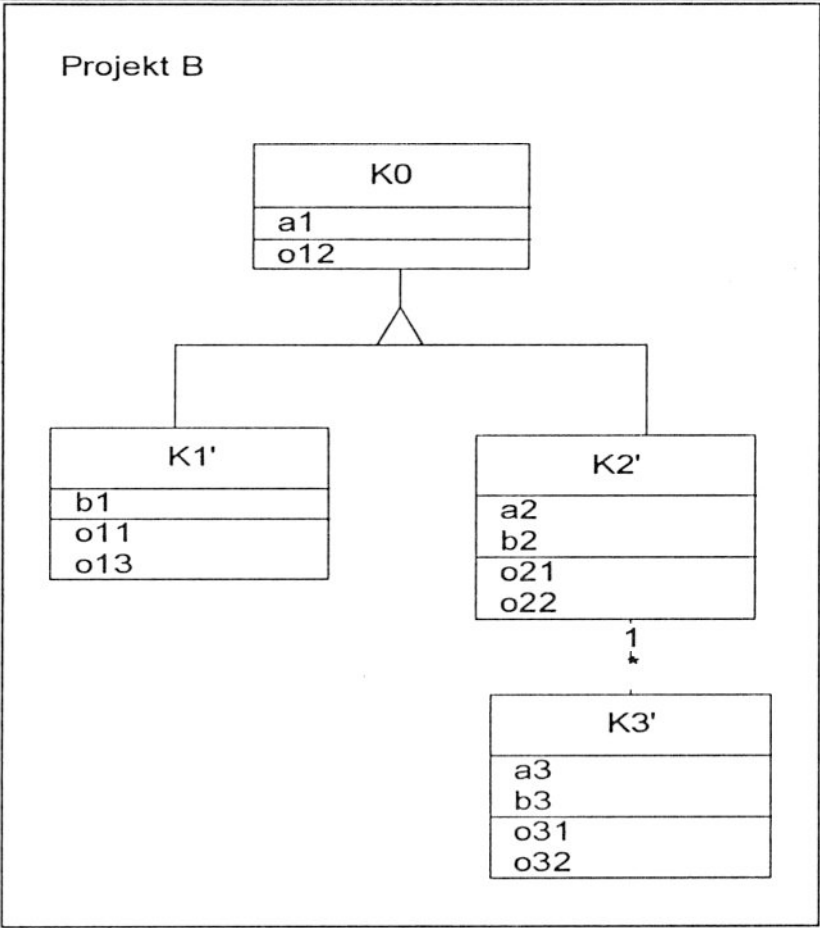
a to oznacza, że wielkość projektu B będzie mniejsza od projektu A pod warunkiem, że

$$d < c11. \quad (4.6)$$

Przykład ten potwierdza konieczność uwzględnienia efektów związku dziedziczenia przy obliczaniu wielkości programu.

Przedstawione przekształcenia i porównania wielkości projektów A i B wskazują na to, że model hierarchii klas, który naturalnie uwzględnia własności dziedziczenia, pozwala na ich odkrywanie i umożliwia badanie wpływu tego mechanizmu na rozmiar programu. Jednakże zastosowana miara wielkości powinna dotyczyć nie tylko liczby klas w programie, ale przede wszystkim własności tych klas i ich metod oraz związków pomiędzy nimi. Należy rozważyć także następne pytanie, związane z mechanizmem dziedziczenia: Czy również głębokość klasy w danej hierarchii oraz liczba bezpośrednich potomków nadklasy ma wpływ na jej rozmiar i w konsekwencji na wielkość nakładów?

Głębokość dziedziczenia w hierarchicznym modelu klas wiąże się ze zjawiskiem dziedziczenia dwupokoleniowego, a czasami więcej, nazywanym wówczas wielopoziomowym. W praktyce nie są to duże krotności, najczęściej występują w zakresie od 2 do 8. Występuje też zjawisko złożonego dziedziczenia w przypadku różnych rodziców. Dziedziczenie złożone, zarówno wielopoziomowe, jak i wielopotomne daje szansę na zmniejszenie krotności definiowania tych samych własności klasy. Jednakże ma ono dwie strony. Z jednej zmniejsza liczbę definiowanych własności, ale z drugiej wymaga poświęcenia więcej czasu na budowę najpierw złożonego modelu, a później projektu i programu. Stąd zachodzi potrzeba dokładnego zbadania stopnia i kierunku wpływu mechanizmu dziedziczenia na wielkość nakładów tworzenia. Nie może być to wielkość uzależniona tylko od liczby deklarowanych własności, lecz także od zakresu ich dostępności, wynikającym ze związków pomiędzy klasami i obiektami.



Rysunek 8. Projekt B, który przedstawia modyfikację projektu A przez wprowadzenie mechanizmu dziedziczenia. Źródło: [opracowanie własne].

4. 2. 5. 2. Cechy procesu obiektowego i ich wpływ na wielkość nakładów

Opisany w załączniku 1 iteracyjny i przyrostowy charakter procesu tworzenia programu przy podejściu obiektowym wymaga rozważenia jego wpływu na czas realizacji całego projektu. Z doświadczeń projektowych wynika, że większa liczba iteracji wydłuża czas wykonania modelu wymagań, ale też skraca czas projektowania i kodowania. Wymagania zidentyfikowane dopiero na etapie projektowania zamiast na etapie analizy mogą doprowadzić nawet do pięciokrotnego zwiększenia kosztów [RUM91, s. 456]. Druga cecha związana z przyrostowym tworzeniem również posiada dwukierunkowy wpływ na czas poświęcony wykonaniu całego projektu. Z jednej strony pozwala go skrócić ze względu na skupienie się tylko na wydzielonych obszarach funkcjonalnych. Z drugiej strony, przy zbyt wąskim podziale zakresów dla tych „przyrostów”, czas tworzenia może zostać wydłużony o czas poświęcony na uwzględnianie wszelkich powiązań pomiędzy tymi „przyrostami”. Obie te cechy, które występują przy realizacji projektu obiektowego sugerują zatem potrzebę ich uwzględnienia w analizie zależności nakładów.

4. 2. 6. Rola repozytorium CASE w estymacji nakładów

Potrzeba automatycznej analizy dokumentacji jest jednym z warunków, które mogą zapewnić obiektywny, szybki i rzetelny sposób zebrania danych, odpowiadających wytypowanym miarom. Wiele bowiem istniejących metod obliczania wielkości programu charakteryzuje się dużą uciążliwością związaną z czasochłonnym, ręcznym przeszukiwaniem dokumentacji i zliczaniem potrzebnych składników. Często dokumentacje takie są niepełne bądź nieczytelne, co znaczenie utrudnia, a czasami uniemożliwia przeprowadzenie rzetelnych pomiarów. Przy takim postępowaniu, z powodu subiektywnych interpretacji, istnieje też możliwość otrzymywania dla tego samego projektu rozbieżnych rozwiązań. Może to być wynikiem posiadanej wiedzy o danej aplikacji, ale i też doświadczenia w stosowaniu określonej metody, np. analizy punktów funkcyjnych. Dodatkowym utrudnieniem jest sytuacja, w której dokumentacja analityczna i projektowa, sporządzana ręcznie, odbiega od końcowego efektu, gdyż nie wystarczyło czasu na uzupełnianie zmian wprowadzanych w trakcie implementacji. Wówczas występuje znaczne zmniejszenie wartości zebranych danych, albo stają się one bezwartościowe z punktu widzenia szacowania wielkości produktu końcowego. Pojawiła się potrzeba nie tylko automatycznej analizy danych, ale i też automatycznego sposobu kolekcjonowania danych o powstającym produkcie.

Możliwość automatyzacji tych obu procesów [BAR90], [HOL94], [HOL95] jest oferowana przez narzędzia CASE (ang. Computer Aided Software Engineering), związane z nowoczesną technologią tworzenia systemów informatycznych zarówno przy podejściu strukturalnym, jak i obiektowym. Potrzeba automatyzacji kolekcjonowania i analizy danych o tworzonym produkcie niejako narzuca przyjęcie kolejnego założenia: uzależnienie wielkości nakładów tworzenia od liczby elementów zawartych w repozytorium⁹ systemu CASE. Taka zależność została sformułowana [BAN94] już w odniesieniu do technologii strukturalnej, zatem zachodzi potrzeba przeprowadzenia badań w celu sprawdzenia słuszności tego stwierdzenia do technologii obiektowej.

Działanie pakietu narzędzi CASE jest podporządkowane konkretnej metodyce tworzenia systemu i zgodnie z jej założeniami pozwala na komputerowe wspomaganie realizacji procesu tworzenia. Istnieją narzędzia wysokiego, średniego i niskiego poziomu [GAL93], [HOL95], a przynależność do odpowiedniej klasy zależy od zakresu ich działania: na wszystkich etapach cyklu życia czy tylko np. na etapie implementacji. Jądro narzędzi

CASE stanowi baza danych (ang. data dictionary), nazywana słownikiem danych, która, w zależności od poziomu działania tego narzędzia, zawiera różne układy informacji o tworzonym produkcie, a także wiedzę o danej organizacji (model organizacji). W tej bazie znajduje się opis realizowanego projektu, specyfikacja wymagań wobec przyszłego systemu, definicje projektowe a także implementacyjne. Jest to, z jednej strony, możliwość zapamiętywania, modyfikowania i odtwarzania wyników pracy zespołu, a z drugiej, źródło wiedzy o realizowanych produktach pośrednich (w postaci diagramów i tekstowych definicji) oraz przebiegu realizacji projektu. Stan zgromadzonej wiedzy zależy od etapu realizowanego projektu. Trzeba dodać, że praca z narzędziami mocno dyscyplinuje pracę zespołu i w wysokim stopniu formalizuje proces tworzenia. Taka formalizacja procesu zapewnia otrzymanie zestawu standardowych produktów i wykonanie tych produktów za pomocą standardowych technik.

Realizacja procesu, zgodna z metodyką i przy zastosowaniu narzędzi CASE, pozwala na wykorzystanie zawartości repozytorium tych narzędzi w procesie mierzenia, ponieważ zgodnie z przyjętymi założeniami, umożliwia:

- automatyczną identyfikację wytypowanych charakterystyk tworzonego produktu,
- automatyczne odwzorowanie mierzonego elementu na wielkość liczbową,
- automatyzację dodatkowych obliczeń,
- gromadzenie wiedzy o różnych projektach,
- wykonywania obiektywnych porównań różnych projektów.

Ze względu na przyjęte kryteria przeprowadzono analizę zawartości słownika CASE ze względu na możliwość wykorzystania go do automatycznego sposobu pozyskiwania danych, co przedstawiono w następnym punkcie tego rozdziału. Wyniki realizacji tego zadania zostały ograniczone do posiadanej przez autora wersji pakietu CASE²⁴.

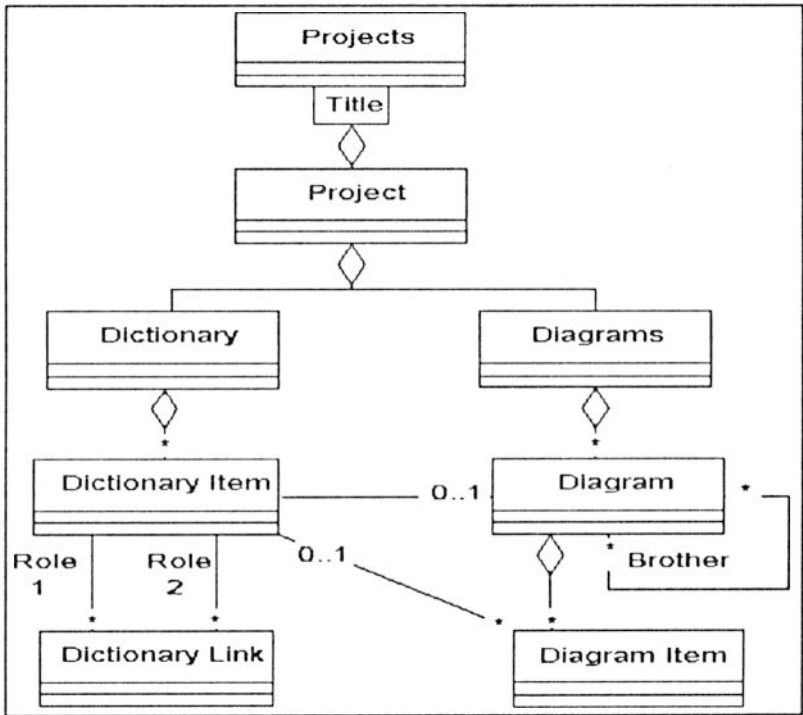
Meta model repozytorium CASE

Dla każdego etapu przedsięwzięcia informatycznego istnieje lista produktów pośrednich, które są wynikiem ukończenia określonych zadań, a także punktem wyjścia do dalszej realizacji projektu. Produkty takie są zapamiętywane w słowniku danych narzędzi

²⁴ Repozytorium jest to baza danych, która przechowuje informacje o wszystkich elementach projektu i znaczących zdarzeniach. Za pomocą odpowiedniego programu umożliwia też odczytanie interesujących informacji o danym projekcie.

²⁵ W pracy wykorzystano pakiet SELECT Enterprise w wersji 5. W załączniku 9 (tab. 63) przedstawiono odpowiedniki pojęć występujących w metodyce OOSE oraz metodyce SELECT Perspective.

CASE w postaci zbioru diagramów i zbioru zapisów tekstowych, logicznie powiązanych za pomocą nazwy projektu. Dane gromadzone w tym słowniku stanowią listę wszystkich elementów tworzonego systemu oraz występujące pomiędzy nimi powiązania. Na rysunku 9 przedstawiono ogólny model repozytorium w notacji obiektowej. W załączniku 8 (tabela 34) zamieszczono szczegółowy wykaz wszystkich obiektów, występujących w repozytorium narzędzi CASE pakietu SELECT. Dla każdego obiektu zamieszczono opis jego struktury i funkcji w systemie.



Rysunek 9. Meta model repozytorium pakietu SELECT. Źródło: [SEL97, s. 2-383].

4. 3. Opis procedury ustalania postaci modelu nakładów

Zadanie konstrukcji modelu nakładów zostało zrealizowane na bazie założenia o ekonomicznej naturze procesu tworzenia programu, na który wpływa wiele różnych czynników, między którymi mogą zachodzić wzajemne powiązania. Pytanie, od jakich elementów rozmiaru tworzonego programu zależy poziom zużycia nakładów stały się przedmiotem prowadzonych w niniejszej pracy badań analitycznych. Za podstawowe narzędzie przyjęto modele budowane przy wykorzystaniu metod ekonometrycznych. Stąd, poszukiwana zależność wielkości nakładów pracy od rozmiaru tworzonego programu ma ogólną postać równania

$$N = f(X_1, X_2, \dots, X_m, e), \quad (4.7)$$

gdzie:

N – zmienna objaśniana, która przedstawia wielkość nakładu pracy,

f - oznacza postać analityczną funkcji zmiennych objaśniających,

$X_1, X_2, \dots, X_m$ – zmienne objaśniające, czyli zbiór atrybutów określających rozmiar programu,

które zostały uznane za najważniejsze przyczyny kształtowania się zmiennej objaśnianej,

e – składnik losowy modelu.

Za zmienną objaśnianą przyjęto wielkość nakładu pracy, a za zmienne objaśniające - atrybuty charakteryzujące rozmiar programu. Przyjęto również, niezbędne w modelach ekonomicznych, założenie o niezmienności niektórych czynników: zakresu systemu informatycznego w trakcie jego tworzenia i metodyki tworzenia.

W celu zapewnienia poprawności postępowania prowadzącego do budowy modelu ekonometrycznego, konstrukcję tego modelu przeprowadzono według sformalizowanej procedury [GRA82 - 92], [NOW84 - 95], [ZEL84], [BART90], [GOL75], która składa się z następujących kroków:

- 1) Wstępne ustalenie potencjalnych zmiennych objaśniających.
- 2) Zbadanie charakteru związków występujących między nakładem – zmienną objaśnianą, a rozmiarem programu, który jest wyrażony za pomocą czynników, pełniących rolę zmiennych objaśniających.
- 3) Dobór odpowiedniego zestawu zmiennych objaśniających do odpowiedniego modelu według wybranych procedur.
- 4) Analiza zgodności i poprawności wyników doboru zmiennych.

Czynniki, które mogłyby pełnić rolę zmiennych objaśniających w budowanym modelu estymacji nakładów ograniczono do zestawu czynników spełniających merytoryczne i formalno – statystyczne kryteria optymalnego doboru zmiennych objaśniających.

Z teoretycznego punktu widzenia spełnienie merytorycznego kryterium optymalnego doboru zmiennych do modelu ekonometrycznego wymaga, by zmienne objaśniające były związane z wiedzą o analizowanym zjawisku, spełniające następujące zasady:

- Preferowane są zmienne, które pozostają w związku merytorycznym ze zmiennymi objaśnianymi.
- Preferowane są zmienne wyrażone w jednostkach naturalnych, gdyż charakteryzują się większym zasobem informacji.
- Muszą istnieć wiarygodne i łatwo osiągalne dane statystyczne dotyczące wyróżnionych w modelu zmiennych.

Na podstawie przeprowadzonego przeglądu istniejących metod szacowania nakładów (rozdział 3.2) ustalono, że przy badaniu zależności między nakładami a czynnikami były brane pod uwagę trzy grupy czynników:

- 1) Rozmiar programu, wyrażany przez wielkość realizowanego przetwarzania, albo przez jego wielkość fizyczną.
- 2) Ogólna charakterystyka programu, często nazywana czynnikiem technicznej złożoności, na którą składa się wielkość, będąca odzwierciedleniem wpływu wyróżnionych arbitralnie czynników.
- 3) Czynniki środowiska, zwykle uwzględniane w trakcie oceny ryzyka realizacji projektu, a wynikające z kwalifikacji i doświadczenia zespołu projektowego, jak też metod i narzędzi używanych przez ten zespół oraz języka implementacji.

Za grupę o zasadniczej roli uznano te czynniki, które są odpowiedzialne za odzwierciedlenie rozmiaru tworzonego produktu, ponieważ większość czynników z drugiej grupy jest przeważnie odzwierciedlana już przez czynniki z pierwszej grupy, a pozostałe na etapie analizy są jeszcze nie określone. Część czynników z grupy trzeciej została już uwzględniona w założeniach do budowy modelu, w których ogranicza się zbiór programów do tworzonych tylko w technologii obiektowej i za pomocą narzędzi CASE. Natomiast sądzi się, że aspekt kwalifikacji i doświadczenia również pośrednio jest uwzględniony w atrybutach rozmiaru programu, bowiem one to decydują o zakresie wykorzystywania w budowie programu możliwości określonej technologii. W tym przypadku głównie chodzi o paradygmaty technologii obiektowej, których wpływ na rozmiar programu, jak wcześniej pokazano, jest istotny. Za skupieniem się wyłącznie na czynnikach grupy pierwszej przemawiają także inne

elementy. Zauważa się istotną różnicę między mierzeniem czynników grupy pierwszej a pozostałymi: o ile te pierwsze można zmierzyć za pomocą określonych jednostek naturalnych, np. liczba funkcji, liczba obiektów, liczba bajtów, czy liczba linii kodu, to w przypadku tych drugich brak jest takich możliwości. W zamian za to wprowadza się przetestowane empirycznie wyznaczone wielkości wagowe, które odpowiadają określonym zestawom czynników i wyznaczonym dla nich zakresom wartości. Takie postępowaniem związane jest z ryzykiem:

- pominięcia istotnej charakterystyki dla tworzonego programu,
- nieuwzględnienia przez wielkość wagową istotnej zmiany jakościowej warunków tworzenia,
- nieodpowiedniego wyboru poziomu wpływu danego czynnika.

W celu zmniejszenia poziomu takiego ryzyka przyjęto, że poszukiwany model estymacji nakładów będzie odzwierciedleniem funkcjonalnej zależności między nakładami pracy, poświęconymi na wytworzenie produktu informatycznego i mierzonymi w naturalnych jednostkach czasu, a ilościowymi czynnikami charakteryzującymi rozmiar tego produktu, dla których istnieje obiektywny i zautomatyzowany sposób ich pozyskania. Oznacza to, że nie będą brane pod uwagę czynniki, których wielkości nie można wyrazić za pomocą odpowiedniej liczby jednostek, których ustalenie wymaga dodatkowych interpretacji. Sformułowanie analitycznej postaci dla tego modelu nakładów jest związane zatem z ustaleniem listy takich czynników i zebraniem dostatecznie dużej liczby danych historycznych, zawierających rzetelne zapisy o wytypowanych czynnikach i odpowiadających im wielkościach nakładów.

4. 4. Charakterystyki rozmiaru modelu programu

Ze względu na cel wymiarowania programu - estymacja wielkości nakładów, charakterystyki tak dobrano, by jak najlepiej przedstawiały rozmiar tworzonego produktu, co również oznaczało spełnienie kryterium merytorycznego. Następnie sprawdzono kryteria związane z wymogami poprawności formalno – statystycznej.

Rozmiar programu, jak opisano w rozdziale drugim, może być wyrażony za pomocą trzech aspektów: **dlugości**, **funkcjonalności** oraz **złożoności**. Biorąc pod uwagę zatem przyjęte założenie o mierzeniu **modelu programu**, uznano za niezbędne przeprowadzenie analizy możliwości oceny tego modelu z punktu widzenia wymienionych aspektów.

Zgodnie z przyjętymi założeniami (punkt 4.2) za czynniki charakteryzujące model programu uznano elementy modeli konstruowanych na etapie analizy. Są to następujące modele:

- model przypadków użycia, który dostarcza pełny opis tworzonego systemu z punktu widzenia funkcjonalności widzianej przez użytkownika,
- model obiektów, który przedstawia opis logicznej struktury systemu,
- model sekwencji, który przedstawia szczegółowy opis przypadków użycia w powiązaniu z obiektami.

Model przypadków użycia i model obiektów składają się na opis tworzonego systemu z dwóch różnych punktów widzenia. Model obiektów wyraża koncepcję tworzonego systemu, zatem pozwala na ocenę jego rozmiaru z poziomu twórcy, a model przypadków użycia, który wyraża oczekiwane wymagania wobec systemu pozwala ocenić jego rozmiar z punktu widzenia użytkownika. Oba modele umożliwiają oczekiwane (jak przedstawiono w punkcie 4.2) skorygowanie oceny rozmiaru systemu przez mechanizmy występujące w technologii obiektowej. Gromadzone w repozytorium CASE definicje (załącznik 8) poszczególnych elementów tych modeli pozwalają na wyodrębnienie niżej opisanych charakterystyk, nazywanych też atrybutami rozmiaru. Za miarę długości modelu tworzonego na etapie analizy przyjęto liczby stron diagramów

- przypadków użycia,
- sekwencji
- obiektów.

Przyjęto, że jednej stronie odpowiada jeden z kolekcji diagramów określonego rodzaju.

Opis funkcjonalności otrzymywany w wyniku prac etapu analizy wymagań jest odzwierciedleniem wymagań użytkownika przyszłego systemu. Miara funkcjonalności, postrzeganej zgodnie z założeniami przedstawionymi w punkcie 4.2, dotyczy zarówno funkcjonalności oczekiwanej jak i tworzonej. Funkcjonalność oczekiwana jest odzwierciedlona przez elementy diagramu przypadków użycia. Natomiast funkcjonalność tworzona jest odzwierciedlona za pomocą diagramu obiektów. Ze względu na to, że diagram ten przedstawia zarówno strukturę tworzonego programu, jak i jego funkcjonalność, został zakwalifikowany do opisu złożoności strukturalnej, a operacje - jako element składowy diagramu obiektów, również do charakterystyki funkcjonalności. Podstawowym elementem funkcjonalności systemu jest przypadek użycia przedstawiony na diagramie przypadków użycia. Opis instrukcji dotyczący rodzaju przetwarzania i wymiany informacji, jakie są z nim

związane jest umieszczony na diagramie sekwencji. Aktualnie, ze względu na brak automatycznego dostępu do tych elementów repozytorium, które umożliwiłyby identyfikację typów wyrażeń składających się na każdy z przypadków użycia, do zestawu proponowanych charakterystyk nie została dołączona identyfikacja typu przypadków użycia. Wobec tego, do charakterystyk modelu dołączono następujące atrybuty diagramu przypadków użycia:

- przypadki użycia,
- związki rozszerzenia,
- związki użycia między przypadkami użycia.

Jako atrybut diagramu sekwencji dołączono zdarzenia, które stanowią o przyczynie wykonywania danej operacji. Modelowanie związków między przypadkami użycia pozwala (na podobieństwo mechanizmu dziedziczenia) na wczesnym etapie tworzenia wskazać możliwość wielokrotnego wykorzystania w tym samym programie tych samych elementów. Jednak ze względu na to, że model przypadków użycia jest opisem tworzonego programu z punktu widzenia użytkownika, jest to miara potencjalnej możliwości, która dopiero w modelu obiektów odpowiada wykorzystanej przez projektantów możliwości.

Pojęcie złożoności, jako jednego z aspektów rozmiaru programu, w niniejszej pracy, dotyczy złożoności strukturalnej. Do ustalenia charakterystyk złożoności strukturalnej wytypowano elementy składowe budowanego na etapie analizy diagramu obiektów: obiekty, atrybuty, operacje oraz związki asocjacji, specjalizacji, generalizacji. Występujące w modelu obiektów związki specjalizacji i generalizacji są atrybutami mechanizmu dziedziczenia, a ich udział w modelu nakładów pełni rolę miary wpływu na rozmiar programu możliwości wewnętrznego wielokrotnego użycia. Jako dodatkową charakterystykę wykorzystania w projekcie mechanizmu dziedziczenia wprowadzono wskaźnik wewnętrznego ponownego użycia obliczany według wzoru (2.53)

$$WWPU = \frac{Iso}{lob}, \quad (4.8)$$

w którym:

Iso – liczba super obiektów/klas,

lob – liczba obiektów/klas.

Ze względu na brak na etapie analizy ilościowego atrybutu charakteryzującego możliwość zewnętrznego ponownego użycia, ten aspekt nie został uwzględniony budowanym modelem.

Z merytorycznego punktu widzenia aspekt długości, w przypadku rozmiaru produktów dla rozważanych w niniejszej pracy problemów, jest związany zarówno z jego

funkcjonalnością jak i złożonością strukturalną. Dlatego też, oczekiwano wykazania wysokiej korelacji charakterystyk długości z charakterystykami funkcjonalności i złożoności. Sprawdzenie istnienia bądź braku tych zależności zostało przeprowadzone za pomocą wybranych metod statystycznych.

4. 5. Zestaw potencjalnych zmiennych objaśniających

W tabeli 18 przedstawiono opisane wyżej charakterystyki dotyczące aspektu długości, funkcjonalności oraz strukturalnej złożoności modelu programu. Zostały one zakwalifikowane do listy potencjalnych zmiennych objaśniających, przedstawiających rozmiar tworzonego programu, traktowane jako podstawowe czynniki kształtowania się wielkości nakładów.

Tabela 18. Charakterystyki rozmiaru modelu analizy wytypowane do zestawu potencjalnych zmiennych objaśniających w modelu nakładów. Znak (*) wskazuje czynniki wyeliminowane z powodu braku danych. Źródło: [opracowanie własne].

Lp.	Nazwa atrybutu	Aspekt rozmiaru
1	Przypadki użycia	Funkcjonalność
2	Związki użycia między przypadkami użycia *	Funkcjonalność, ponowne użycie
3	Związki rozszerzeń między przypadkami użycia *	Funkcjonalność, ponowne użycie
4	Obiekty	Złożoność strukturalna
5	Związki asocjacji	Złożoność strukturalna
6	Związki specjalizacji	Złożoność strukturalna, ponowne użycie
7	Atrybuty	Złożoność strukturalna
8	Operacje	Złożoność strukturalna, funkcjonalność
9	Zdarzenia *	Złożoność strukturalna
10	Diagramy przypadków użycia	Długość
11	Diagramy sekwencji	Długość
12	Diagramy obiektów	Długość
13	Wskaźnik ponownego użycia	Ponowne użycie
14	Przyrosty *	Proces

5. CHARAKTERYSTYKA KOLEKCJI PROJEKTÓW

5. 1. Zbiór danych z obserwacji

W celu optymalnego doboru zmiennych objaśniających oraz ustalenia postaci analitycznej modelu nakładów przeprowadzono badania empiryczne. Po wykonaniu penetracji polskiego rynku firm tworzących oprogramowanie udało się skompletować kolekcję czternastu zestawów danych. Każdy zestaw odpowiada jednemu projektowi. Nie jest to duża liczba, ale ze względu na niewielki procent projektów realizowanych w naszym kraju w technologii obiektowej i jeszcze przy dodatkowym wymaganiu - korzystaniu z narzędzi CASE, konieczne było ograniczenie się do uzyskanej próbki danych.

Tabela 19. Wykaz projektów. Źródło: [opracowanie własne].

Nr projektu	Zespół	Nazwa projektu
1	A	FOPO
2	A	SZKO
3	A	MAPO
4	A	EXPO
5	A	EDU
6	A	KURSY
7	A	PLUTO
8	B	AGENCJA
9	B	BANK
10	B	MPK
11	C	TWORK
12	C	REGON
13	C	SERWUS
14	C	REJROL

W wyniku wstępnej analizy tych danych, z powodu niewykorzystywania przez projektantów wszystkich cech konstruowanych modeli, spośród wytypowanej listy potencjalnych zmiennych objaśniających, wystąpiła konieczność wyeliminowania czterech czynników z powodu braku odpowiadających im danych. Dwa z nich były przeznaczone do charakterystyki poziomu ponownego użycia w modelu przypadków użycia:

- związki rozszerzeń,
- związki użycia,

trzeci czynnik był atrybutem diagramu sekwencji, a czwarty dotyczył charakterystyki procesu tworzenia. Zatem lista potencjalnych zmiennych została zmniejszona do dziesięciu. W tabeli 18 wyeliminowane czynniki zostały wskazane za pomocą znaku (*). W tabeli 20 zamieszczono wykaz skróconych nazw zmiennych używanych w dalszym ciągu pracy.

Kolekcja projektów powstała w wyniku poszukiwań przy wykorzystaniu różnych środków: ankiet, wywiadów i referencji. W ten sposób uzyskano dostęp do prac kilku firm, ale jedynie projekty z trzech zespołów odpowiadały postawionym wymaganiom. W tabeli 19 pokazano, które projekty były wykonywane przez poszczególne zespoły. W skład tych zespołów wchodziły osoby o podobnym doświadczeniu w tworzeniu programów w technologii obiektowej. Wszystkie projekty były wykonane przy wykorzystaniu narzędzia SELECT.

Tabela 20. Wykaz zmiennych i ich skrótów.
Źródło: [opracowanie własne].

Skrót nazwy zmiennej	Nazwa zmiennej
NAKLAD	Nakład pracy
OBJ	Obiekt
ATT	Atrybuty
SPEC	Specjalizacje
ASSO	Asocjacje
OPER	Operacje
USCA	Przypadki użycia
DOBJ	Diagramy obiektów
DUSE	Diagramy przypadków użycia
DSEQ	Diagramy sekwencji
WWPU	Wskaźnik ponownego użycia

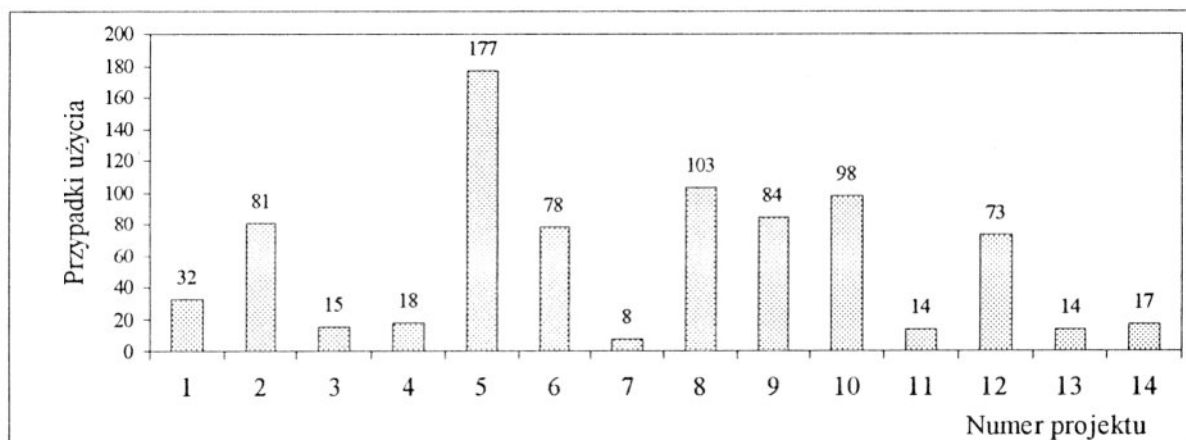
W tabeli 21 przedstawiono kolekcję danych rzeczywistych, pochodzących z obserwacji dla trzech zespołów projektowych: A, B i C.

Tabela 21. Dane z obserwacji dotyczące czternastu projektów wykonanych przez zespoły A, B i C. Źródło: [opracowanie własne].

Nr	Nakład	OBJ	ATT	SPEC	ASSO	OPER	USCA	DOBJ	DUSE	DSEQ	WWPU
	Y [om]	X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	X ₉	X ₁₀
1	9,00	16	135	2	21	19	32	1	1	3	0,0625
2	9,00	92	295	68	48	205	81	10	1	8	0,0870
3	5,00	15	45	11	9	180	15	1	1	1	0,2667
4	7,50	45	79	6	21	38	18	4	1	0	0,0667
5	8,30	38	275	6	47	456	177	5	1	0	0,1053
6	5,00	60	502	35	29	238	78	7	4	17	0,0667
7	9,60	25	104	6	17	300	8	5	1	0	0,0800
8	45,50	101	980	17	90	488	103	11	13	92	0,0198
9	19,00	77	860	2	64	174	84	8	10	38	0,0130
10	23,50	78	161	6	46	267	98	8	13	92	0,0385
11	16,75	174	67	145	129	805	14	13	1	19	0,1379
12	113,0	121	615	19	148	65	73	11	12	11	0,0826
13	10,00	58	51	50	22	302	14	7	1	12	0,1897
14	12,00	14	18	2	17	73	17	3	3	12	0,0714

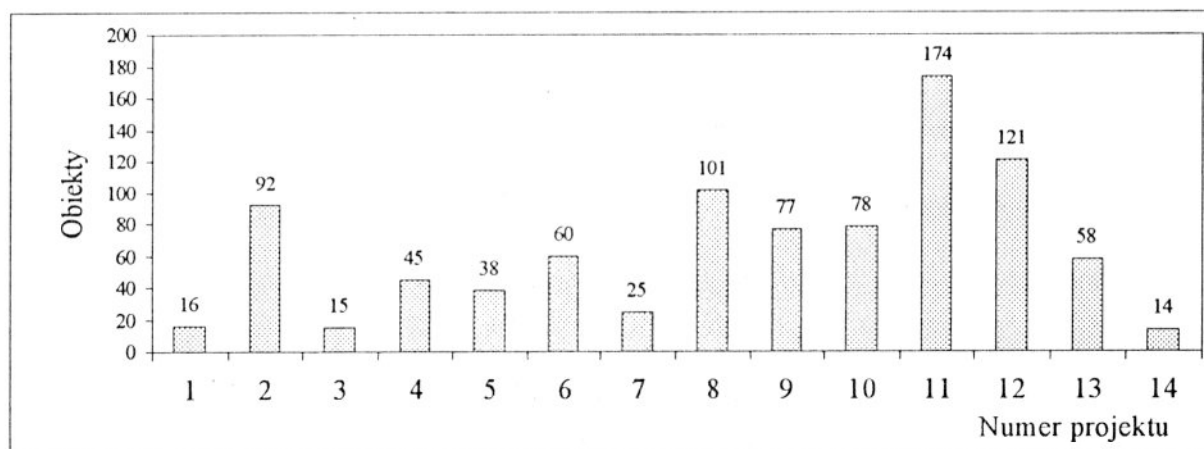
5. 2. Analiza graficzna danych

Na histogramach przedstawiono graficzną prezentację danych, odpowiadających wybranym czynnikom w odniesieniu do wszystkich projektów. Rysunek 10 jest ilustracją liczby *przypadków użycia* przypadających na każdy projekt. Przedstawiony zakres wielkości od 8 do 177 wskazuje na dużą rozpiętość między funkcjonalnością poszczególnych projektów.



Rysunek 10. Histogram przypadków użycia, Źródło: [opracowanie własne].

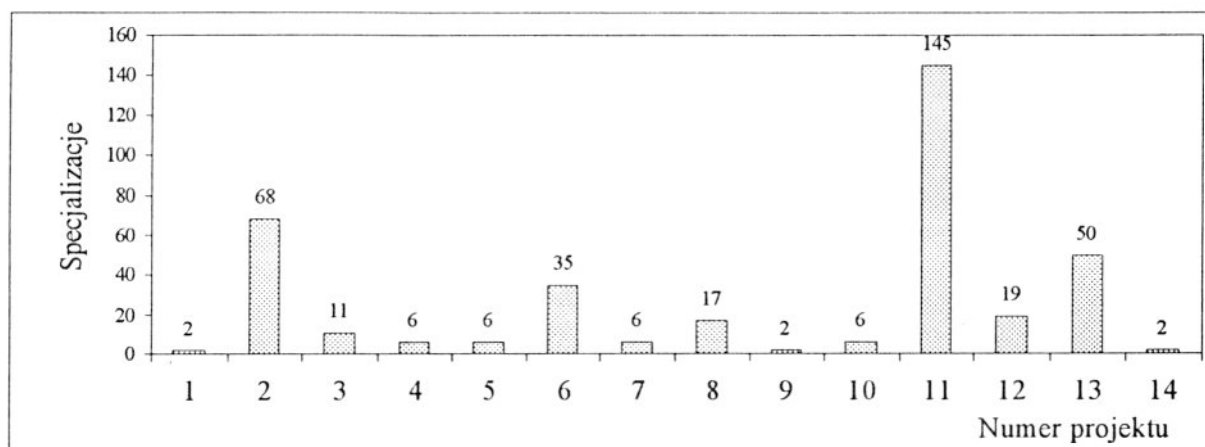
Kolejne wartości atrybutów rozmiaru przedstawiono w postaci wykresów odpowiadających wybranym elementom modelu, które ilustrują histogramy na rysunkach 11, 12 oraz 15 i 16.



Rysunek 11. Histogram obiektów. Źródło: [opracowanie własne].

Maksymalna liczba *przypadków użycia* wystąpiła w projekcie nr 5, natomiast maksymalna liczba zdefiniowanych *obiektów* w projekcie nr 11, które w poszczególnych projektach występują w zakresie od 14 do 174. Również w projekcie nr 11 wystąpiła maksymalna liczba związków *specjalizacji* oraz *asocjacji* zachodzących między obiektami, co jest zgodne z rolą

tych elementów w modelu, gdyż wszystkie dotyczą obiektów. Zatem sądzi się, że analiza statystyczna pozwoli wykazać istniejącą zależność między tymi atrybutami.



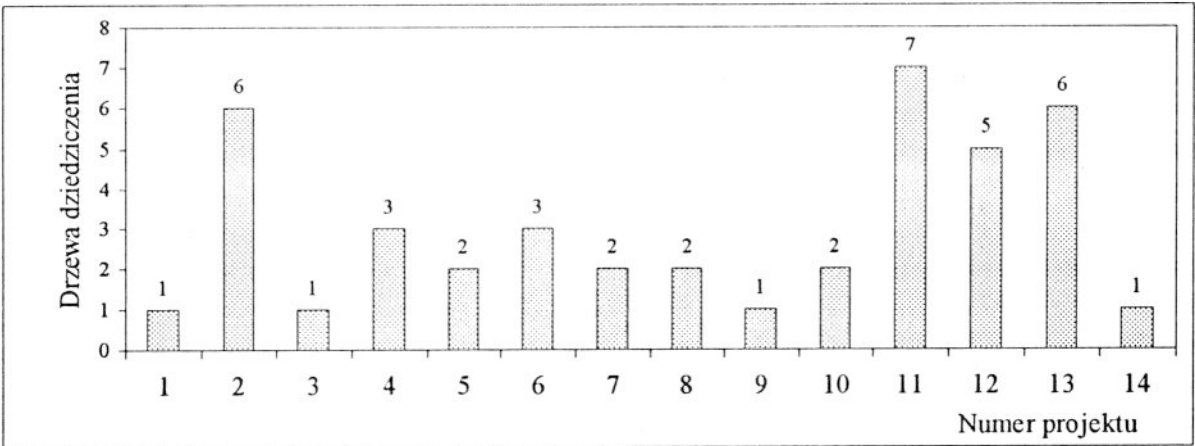
Rysunek 12. Histogram specjalizacji. Źródło: [opracowanie własne].

Liczba związków generalizacji i *specjalizacji* wskazuje na wykorzystanie mechanizmu dziedziczenia już na etapie analizy. W przypadku projektów nr 2 i 11 widać wyraźną różnicę w liczbie tych typów związków w porównaniu do pozostałych projektów. Związki generalizacji i specjalizacji tworzą struktury hierarchiczne, dla których stosuje się metrykę głębokości. W celu zbadania wykorzystania możliwości wewnętrznego ponownego użycia z tytułu dziedziczenia dla poszczególnych projektów przeprowadzono analizę głębokości występujących w nich drzew dziedziczenia (tab. 22). Na histogramie (rys. 13) pokazano ogólną liczbę drzew dla każdego z projektów. Ich liczba waha się w zakresie od 1 do 7. W projekcie nr 11 występuje maksymalna liczba drzew, a w projektach nr 2 i 13 po 6 drzew. Liczby te są bardzo małe wobec ogólnej liczby obiektów w tych projektach (rys. 11). W wyniku dalszej analizy ustalono dla każdego z tych drzew ich głębokości. Okazało się, że we wszystkich projektach występują one w zakresie od 1 do 3. Są to również małe wielkości, które, podobnie jak suma drzew, wskazują, że na tym etapie tworzenia, realizowanym przez dane zespoły, występuje niewielki poziom wykorzystania możliwości ponownego użycia. Potwierdzeniem płaskiej hierarchii modeli programów jest również bardzo niska wartość *wskaźnika ponownego użycia*, która przebiega w granicach od 0,2667 do 0,013.

Tabela 22. Analiza głębokości drzew dziedziczenia.
Źródło: [opracowanie własne].

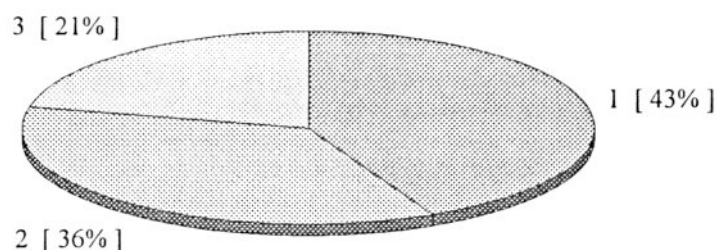
Nr projektu	Liczba drzew	Głębokość drzewa	
		Max	Min
1	1	1	1
2	6	3	1
3	1	2	1
4	3	1	1
5	2	3	1
6	3	2	1
7	2	1	1
8	2	1	1
9	1	1	1
10	2	2	1
11	7	3	1
12	5	2	1
13	6	2	1
14	1	1	1

Następnie przedstawiono wyniki podsumowania w postaci wykresu kołowego (rys. 14), który pokazuje procentowy udział liczby projektów w występujących głębokościach (1, 2 i 3), ustalonych dla tych projektów i pokazanych w tabeli 22 w postaci maksymalnych i minimalnych głębokości drzew dziedziczenia.



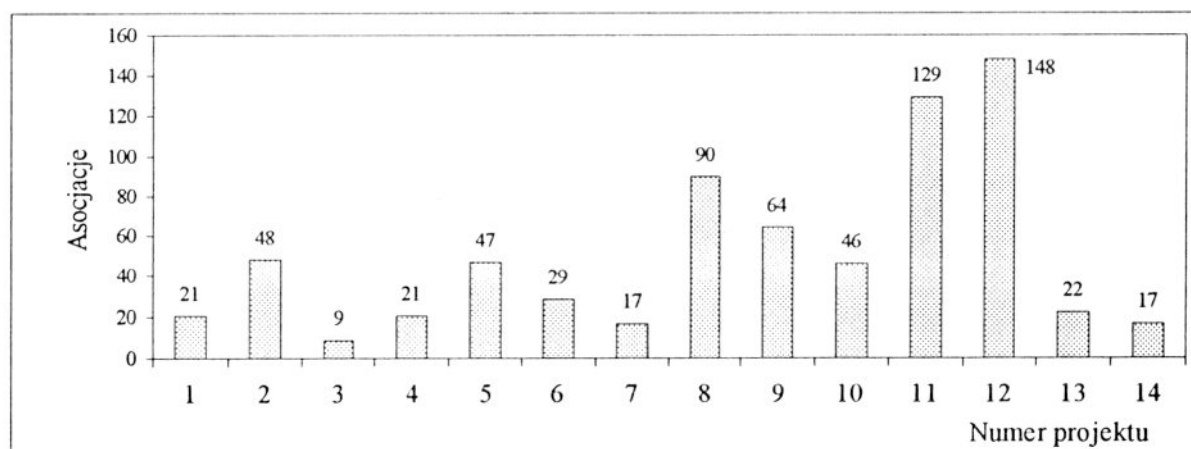
Rysunek 13. Histogram drzew dziedziczenia. Źródło: [opracowanie własne].

Największy odsetek projektów, bo aż 43%, charakteryzuje się minimalną głębokością drzew. Najmniejszy odsetek, bo tylko w 21% projektów występują maksymalne głębokości, które i tak są bardzo małe w porównaniu do wielkości średnich, czyli około 10, występujących w modelach obiektów/klas na etapie projektowania.



Rysunek 14. Procentowy udział projektów dla ustalonych głębokości drzew dziedziczenia. Źródło: [opracowanie własne].

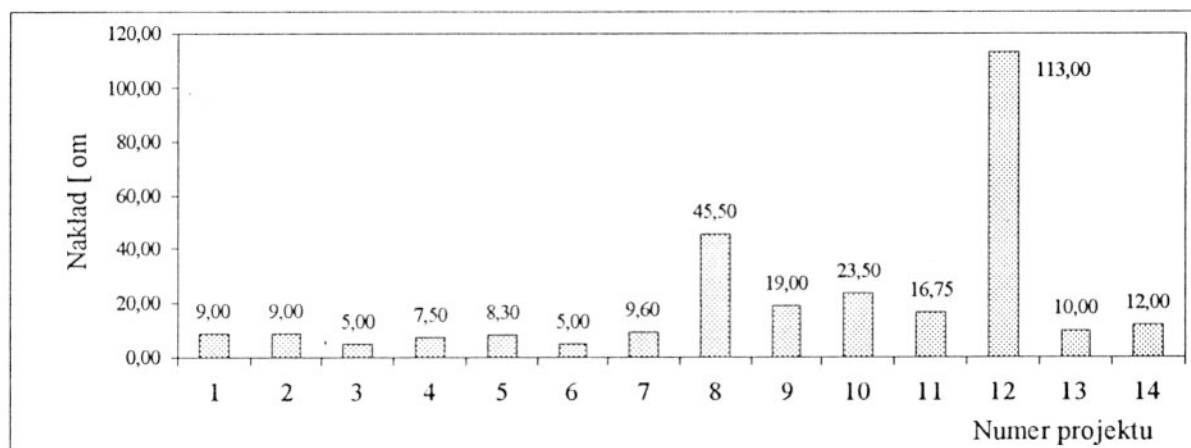
Przeprowadzono analizę związków asocjacji, które zwiększają istotnie zakres własności obiektów i klas, jak to opisano w rozdziale 5. Na histogramie (rys. 15) przedstawiono liczbę asocjacji występujących w poszczególnych projektach. Projekty nr 11 i 12 znacznie wyróżniają się spośród pozostałych liczbą tego typu powiązań.



Rysunek 15. Histogram asocjacji. Źródło: [opracowanie własne].

Przedstawione na rys. 16 wartości *nakładów*, które występują w zakresie od 5 do 113 osobo - miesięcy, wskazują na dużą rozpiętość tej wielkości. W przypadku projektu trzeciego, czwartego i szóstego jest to najmniejszy przedział wartości: od 5 do 7.50 osobo - miesięcy. Następna grupa to projekty pierwszy, drugi, piąty i siódmy, których nakłady występują w przedziale od 7.50 do 9.60, a dla kolejnych dwóch projektów: trzynastego i czternastego w

zakresie od 10 do 12, projekty od dziewięć do jedenaście w zakresie od 16.75 do 23.50. Z ostatnich dwóch projektów jeden (dwunasty) znajduje się w maksymalnym przedziale z wielkością nakładów 113 osobo - miesięcy, a ósmy projekt z wielkością 43.50 w przedziale wielkości średnich. Duża różnorodność w zakresie wartości nakładów pozwala przypuszczać o rozkładzie normalnym tej wielkości. Wyniki badania rozkładu tej wielkości i innych parametrów statystycznych zostały przedstawione w następnym rozdziale.



Rysunek 16. Histogram wielkości nakładów dla wszystkich projektów. Na osi X umieszczono numery projektów według kolejności podanej w tabeli 21. Źródło: [opracowanie własne].

6. KONSTRUKCJA MODELU NAKŁADÓW

6. 1. Badanie charakteru związków między atrybutami rozmiaru programu

Do analizy siły ewentualnych związków między potencjalnymi zmiennymi objaśniającymi, dla których dane zostały przedstawione w zestawieniu zbiorczym w tabeli 38, wykorzystano metody [BART74 - 90], [FAR67], [GAR82], [GRA82 - 89], [NOW84 - 94], [PAW76] przeznaczone do badania współczynnika korelacji liniowej przez badanie współczynnika zmienności i wyznacznika macierzy współczynników korelacji liniowej oraz weryfikację hipotezy o wyznaczniku macierzy współczynników korelacji liniowej zmiennych objaśniających.

6. 1. 1. Współczynnik zmienności

Wstępnym warunkiem dla uznania różnych czynników za zmienne objaśniające jest ich dostatecznie duża zmienność. Miarą poziomu zmienności jest współczynnik zmienności

$$v_i = \frac{s_i}{\overline{x_i}}, \quad (6.1)$$

gdzie:

s_i - odchylenie standardowe zmiennej X_i ,

$\overline{x_i}$ - wartość średnia zmiennej X_i .

W tabeli 23 przedstawiono wartość współczynnika zmienności v_i dla każdej zmiennej X_i . Te zmienne, których współczynniki spełniają następującą nierówność

$$v_i \leq v^* \quad (i=1,2, \dots) \quad (6.2)$$

powinny zostać wyeliminowane. Przyjęto wartość $v^* = 0,10$, dla której odpowiednie zmienne z tabeli 21 stają się potencjalnymi zmiennymi objaśniającymi, gdyż wszystkie odpowiadające im współczynniki nie spełniają nierówności (6.2).

Tabela 23. Wartości odchylenia standardowego s_i , średniej $\bar{x}_i$ zmiennej objaśniającej X_i oraz współczynnika zmienności v_i . Źródło: [opracowanie własne].

Zmienna objaśniająca X_i	Odchylenie standardowe s_i	Średnia $\bar{x}_i$	Współczynnik zmienności v_i
OBJ	46,1260	65,2857	0,7065
ATT	317,6002	299,0714	1,0620
SPEC	39,4738	26,7857	1,4737
ASSO	43,3514	50,5714	0,8572
OPER	212,7544	257,8571	0,8251
USCA	49,4539	58,0000	0,8527
DOBJ	3,7505	6,7143	0,5586
DUSE	5,0498	4,5000	1,1222
DSEQ	31,4525	21,7857	1,4437
WWPU	0,0677	0,0920	0,7355

6. 1. 2. Badanie wyznacznika macierzy współczynników korelacji liniowej

W pierwszym kroku zbudowano macierz X o wymiarach $n \times k$, zawierającą wartości zmiennych objaśniających. Elementom tej macierzy odpowiada tablica danych nr 21, gdzie $n = 14$, jako liczba obserwacji, a $k = 10$, jako liczba zmiennych objaśniających w budowanym modelu nakładów. Następnie utworzono macierz Z , również o wymiarach $n \times k$, która składa się z wartości standaryzowanych poszczególnych zmiennych. Standaryzację tych zmiennych przeprowadzono według odpowiednich wzorów [BRA82, s. 47]. Macierz R współczynników korelacji liniowej wyznaczono na podstawie równania macierzowego

$$R = Z^T Z = \begin{bmatrix} 1 & r_{12} & \dots & r_{1k} \\ r_{21} & 1 & \dots & r_{2k} \\ \dots & \dots & \dots & \dots \\ r_{k1} & r_{k2} & \dots & 1 \end{bmatrix}, \quad (6.3)$$

gdzie:

R – macierz korelacji o wymiarach $k \times k$,

Z – macierz standaryzowanych wartości zmiennych o wymiarach $n \times k$,

Z^T – macierz transponowana w stosunku do macierzy Z , o wymiarach $k \times n$,

r_{jk} - współczynnik korelacji liniowej między j – tą i k – tą zmienną.

W tabeli 24 w pierwszym rzędzie każdego wiersza przedstawiono²⁶ elementy macierzy R , a w drugim rzędzie poziomy istotności (w nawiasach). Kolumny i wiersze tabeli odpowiadają kolumnom i wierszom macierzy R . Za pomocą znaku (*) oznaczono istotne współczynniki korelacji.

Tabela 24. Macierz R współczynników korelacji liniowej Pearsona między nakładem i atrybutami rozmiaru oraz odpowiadających im poziomów istotności (w nawiasach). Źródło: [opracowanie własne].

	NAKLAD	ASSO	ATT	DOBJ	DSEQ	DUSE	OBJ	OPER	SPEC	USCA	WWPU
NAKLAD	1,000										
ASSO	0,772* (0,001)	1,000									
ATT	0,500 (0,069)	0,493 (0,073)	1,000								
DOBJ	0,502 (0,068)	0,825* (0,000)	0,500 (0,069)	1,000							
DSEQ	0,231 (0,426)	0,283 (0,327)	0,513 (0,061)	0,453 (0,104)	1,000						
DUSE	0,670* (0,009)	0,520 (0,057)	0,723* (0,003)	0,483 (0,080)	0,813* (0,000)	1,000					
OBJ	0,487 (0,077)	0,886* (0,000)	0,370 (0,193)	0,943* (0,000)	0,361 (0,205)	0,390 (0,168)	1,000				
OPER	-0,102 (0,729)	0,435 (0,120)	0,059 (0,840)	0,580* (0,030)	0,274 (0,343)	-0,031 (0,916)	0,603* (0,022)	1,000			
SPEC	-0,054 (0,855)	0,474 (0,087)	-0,170 (0,562)	0,626* (0,017)	-0,076 (0,795)	-0,270 (0,350)	0,740* (0,002)	0,694* (0,006)	1,000		
USCA	0,195 (0,504)	0,254 (0,380)	0,548* (0,042)	0,278 (0,335)	0,372 (0,190)	0,430 (0,125)	0,149 (0,610)	0,186 (0,525)	-0,213 (0,465)	1,000	
WWPU	-0,214 (0,464)	-0,205 (0,483)	-0,556* (0,039)	-0,277 (0,338)	-0,479 (0,083)	-0,549* (0,042)	-0,149 (0,610)	0,142 (0,629)	0,291 (0,312)	-0,396 (0,161)	1,000

Teoretycznie, możliwe jest wystąpienie jednego z dwóch skrajnych przypadków, kiedy wyznacznik macierzy R

a)

$$\det R = 1, \quad (6.4)$$

wówczas zmienne objaśniające tworzą zbiór ortogonalny, co oznacza, że są nie skorelowane;

b)

$$\det R = 0, \quad (6.5)$$

wówczas istnieje dokładna liniowa zależność między zmiennym objaśniającymi;

lub trzeci przypadek, kiedy wyznacznik macierzy R spełnia nierówność

c)

$$0 < \det R < 1. \quad (6.6)$$

²⁶ Przedstawione w pracy wyniki obliczeń analiz statystycznych zostały otrzymane za pomocą specjalistycznego pakietu statystycznego SPSS w wersji 7.0.

Praktycznie, najczęściej występuje przypadek *c*. Wówczas o silnej współliniowości świadczy wartość $\det R$ bliska 0, a o liniowej niezależności świadczy wartość $\det R$ bliska 1. Obliczony dla macierzy R , przedstawionej w tabeli 24, wyznacznik wynosi

$$\det R = 1,669E - 06. \quad (6.7)$$

Otrzymana wartość wyznacznika macierzy R wskazuje na istnienie zależności liniowej między zmiennymi objaśniającymi.

6. 1. 3. Weryfikacja hipotezy o wyznaczniku macierzy współczynników

Metoda weryfikacji hipotezy o wyznaczniku macierzy współczynników korelacji liniowej zmiennych polega na sprawdzeniu hipotezy H_0 , która zakłada, że zmienne objaśniające tworzą zbiór ortogonalny. Według wzoru (6.4) hipoteza H_0 brzmi:

$$H_0 : \det R = 1 , \quad (6.8)$$

a hipoteza alternatywna H_1 o dokładnej zależności liniowej między zmiennymi objaśniającymi ma postać

$$H_0 : \det R < 1 . \quad (6.9)$$

Zakłada się, że zmienne standaryzowane z macierzy Z mają k – wymiarowy sferyczny rozkład normalny, gdzie k - liczba zmiennych. W celu sprawdzenia tej hipotezy przyjmuje się statystykę chi – kwadrat postaci

$$\chi^2_{\det R} = -[n - 1 - \frac{1}{6}(2k + 5)] \ln \det R , \quad (6.10)$$

uwarunkowaną przez $n = 14$, jako liczbę obserwacji dla każdej zmiennej oraz $k = 10$, jako liczbę zmiennych objaśniających. Przy założeniu prawdziwości hipotezy H_0 statystyka chi – kwadrat ma asymptotyczny rozkład o liczbie stopni swobody

$$v = \frac{1}{2} k(k - 1) . \quad (6.11)$$

Obszar krytyczny prawostronny dla tego testu określa nierówność

$$\chi^2_{\det R} \geq \chi^2(\alpha; \nu), \quad (6.12)$$

gdzie:

$\chi^2(\alpha; \nu)$ - wartość krytyczna odczytana z tablicy rozkładu²⁷ chi-kwadrat wynosi 69,957 dla ustalonego poziomu istotności $\alpha = 0.01$ i obliczonej ze wzoru (6.11) liczby $\nu = 45$ stopni swobody. Natomiast wyliczona wartość statystyki $\chi^2_{\det R} = 117,5123$.

Zatem, hipotezę H_0 należy odrzucić, ponieważ nierówność dana wzorem (6.12) jest spełniona. Oznacza to, że w zbiorze zmiennych objaśniających (tab. 21) zachodzi zjawisko współliniowości.

Analiza macierzy korelacji przedstawionej w tabeli 24 pozwala również na przeprowadzenie testu o istnieniu liniowej zależności między każdymi dwoma zmiennymi, ale z uwzględnieniem wpływu pozostałych zmiennych. W pierwszym rzędzie każdego wiersza tabeli zamieszczono wartość współczynnika korelacji a w drugim poziom istotności. Wyliczone wartości poziomów istotności pozwalają na weryfikację hipotezy H_0 na ustalonym poziomie $\alpha = 0,05$ o braku korelacji między poszczególnymi zmiennymi. Na przykład wartości poziomów istotności dla atrybutów: *asocjacje* (ASSO) oraz *diagramy przypadków użycia* (DUSE), nie dają podstaw do odrzucenia hipotezy H_0 o braku ich skorelowania z *nakładami* (NAKŁAD). Analiza innych współczynników prowadzi też do stwierdzenia braku podstaw do odrzucenia hipotezy H_0 , a przyjęcia hipotezy H_1 np. dla *diagramów obiektów i asocjacji*, *specjalizacji*, *obiektów i operacji*.

Zależność liniowa między tymi atrybutami była wcześniej przewidywana jako, że są to atrybuty dotyczące długości i złożoności strukturalnej tego samego modelu, modelu obiektów. Nie wystąpiła jednak, podobnie przewidywana dla modelu przypadków użycia, zależność liniowa między *diagramami przypadków użycia* i *przypadkami użycia*.

Zmniejszenie wielkości *nakładów* powodują tylko trzy atrybuty: *operacje*, *specjalizacje* oraz *wskaźnik ponownego użycia* (WWPU), których współczynniki korelacji mają wartość ujemną. Jednak żaden z tych atrybutów, jak wynika z poziomów istotności, nie wyróżnia się istotnym wpływem na wielkość tych *nakładów*.

²⁷ Występujące w tym rozdziale wartości statystyk zostały odczytane z tablic statystycznych [ZIE72].

6. 2. Test zgodności rozkładu zmiennych z rozkładem normalnym

Do weryfikacji hipotezy o określonym typie rozkładu zmiennej zastosowano test zgodności Kołmogorowa – Smirnowa. Przeprowadzono test o normalności rozkładu zmiennej objaśnianej i kolejnych zmiennych objaśniających, stawiając następującą hipotezę zerową H_0 : dystrybuanta zmiennej losowej X jest równa wartości dystrybuanty $F(x)$, gdzie $F(x)$ należy do rodziny dystrybuant zmiennych losowych o rozkładzie normalnym wobec hipotezy alternatywnej

H_1 : dystrybuanta zmiennej losowej X jest znacząco różna od wartości dystrybuanty $F(x)$.

Sprawdzianem dla hipotezy H_0 jest statystyka

$$D_n = \sup_x |F_n(x) - F(x)|,$$
 (6.13)

gdzie:

$F_n(x)$ - empiryczna dystrybuanta zmiennej losowej X .

Z tablic odczytano, że dla $n = 14$ oraz $\alpha = 0,05$ prawdopodobieństwo $P\{D_n \geq 0,348\} = 0,05$.

Oznacza to, że dla wszystkich zmiennych, których wartości statystyki dla rozkładu empirycznego, obliczone ze wzoru (6.13), nie przekraczają wartości krytycznej 0,348, nie ma podstaw do odrzucenia hipotezy zerowej.

Tabela 25. Wyniki testu Kołmogorowa – Smirnowa (K – S) dla rozkładu zmiennej objaśnianej i zmiennych objaśniających, część I. Źródło: [opracowanie własne].

Statystyki		Zmienne						
		NAKLAD	LNNAK	ASSO	ATT	DOBJ	DSEQ	DUSE
Liczba obserwacji N		14	14	14	14	14	14	14
Parametry	Średnia	20,9393	2,5899	50,57	299,07	6,71	21,79	4,50
Normalne	Standardowe	28,5118	0,857685	43,35	317,60	3,75	31,45	5,05
	Odchylenie							
Największe	Absolutna	0,321	0,203	0,238	0,240	0,105	0,321	0,327
Odległości	Dodatnia	0,321	0,203	0,238	0,240	0,105	0,321	0,327
	Ujemna	-0,288	-0,126	-0,169	-0,188	-0,102	-0,244	-0,244
Statystyka K-S		1,202	0,758	0,890	0,896	0,392	1,201	1,225
Poziom istotności		0,111	0,614	0,406	0,398	0,998	0,112	0,100

W wyniku przeprowadzonego testu (tab. 25 i tab. 26) dla wszystkich zmiennych otrzymano największe wartości różnic, co do modułu, $D_n \leq 0.327$. Oznacza to, że wartość

krytyczna nie została przekroczona dla żadnej z dziesięciu zmiennych objaśniających oraz zmiennej objaśnianej. Zatem nie ma podstaw do odrzucenia H_0 . Jednak, w przypadku zmiennej objaśniającej *diagramy przypadków użycia (DUSE)* wartość $D_n = 0,327$ jest bliska wartości krytycznej, co wskazuje na słabe potwierdzenie hipotezy o normalności rozkładu dla tej zmiennej. Podobnie dla zmiennej objaśniającej *diagramy sekwencji* oraz zmiennej objaśnianej *nakład* wartość $D_n = 0,321$ wskazuje na słabe potwierdzenie tej hipotezy. Z tego względu dla zmiennej objaśnianej przeprowadzono sprawdzenie tego testu na jej logarytmicznej transformacji (LNNAK). Dla logarytmu *nakładów* wartość $D_n = 0,203$ daje podstawy do przyjęcia hipotezy o normalności rozkładu tej zmiennej.

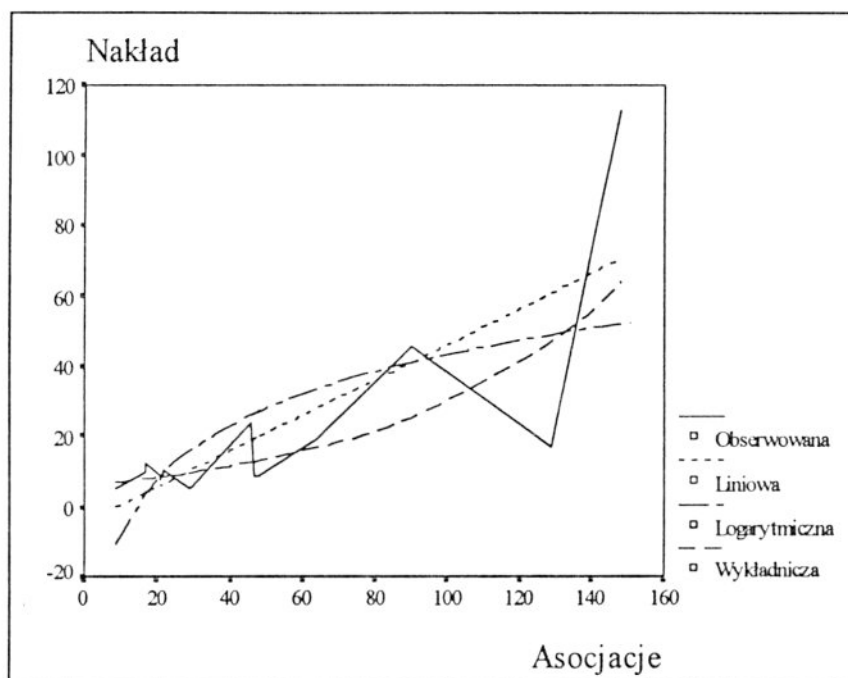
Tabela 26. Wyniki testu Kolmogorowa – Smirnowa dla rozkładu zmiennej objaśnianej i zmiennych objaśniających, część II. Źródło: [opracowanie własne].

Statystyki		Zmienne				
		OBJ	OPER	SPEC	USCA	WWPU
Liczba obserwacji N		14	14	14	14	14
Parametry	Średnia	65,29	257,86	26,79	58,00	9,2E-02
Normalne	Standardowe	46,13	212,75	39,47	49,45	6,8E-02
	Odchylenie					
Największe	Absolutna	0,133	0,204	0,292	0,219	0,244
Odległości	Dodatnia	0,117	0,204	0,292	0,219	0,244
	Ujemna	-0,133	-0,131	-0,265	-0,156	-0,121
Statystyka K-S		0,498	0,762	1,094	0,820	0,912
Poziom istotności		0,965	0,608	0,182	0,511	0,376

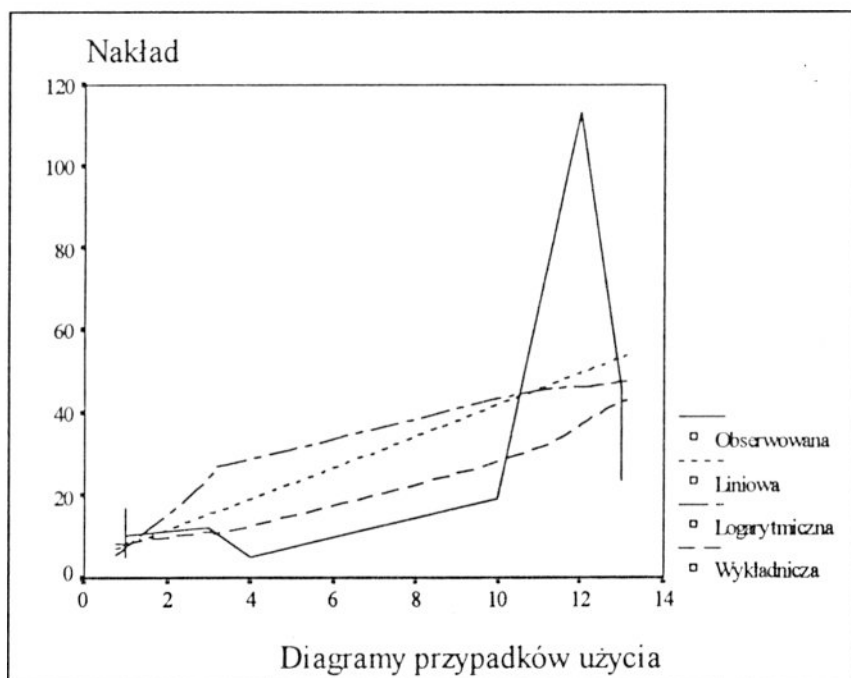
6. 3. Proces doboru optymalnych zmiennych objaśniających

Według wzoru (4.7) ustalono postać analityczną modelu *nakładów*. W celu znalezienia właściwej postaci analitycznej dla tego modelu ogólnego, najpierw zbadano charakter zależności między zmienną objaśnianą, a zmiennymi objaśniającymi za pomocą metody graficznej, a następnie wykonano estymację modelu *nakładów* za pomocą metody najmniejszych kwadratów. Wobec stwierdzenia (punkt 6.1.3) istnienia współliniowości między zmiennymi objaśniającymi przystąpiono do ustalenia, które zmienne z listy potencjalnych zmiennych objaśniających należy wyeliminować, by uniknąć powielania informacji. Przed wyborem metody prowadzącej do redukcji zmiennych przeanalizowano rozrzut punktów empirycznych na graficznej prezentacji związków *nakładów* z poszczególnymi atrybutami rozmiaru modelu programu. Na rysunkach od 18 do 25 pokazano porównanie przybliżeń wartości obserwowanych za pomocą funkcji liniowej, logarytmicznej i wykładniczej.

W ocenie wzrokowej wykres zależności między *nakładem* i związkami *asocjacji* (rys. 18) najlepiej może być przybliżony za pomocą funkcji wykładniczej, która swoim kształtem jest najbardziej zbliżona do wykresu danych z obserwacji. Wykres tej zależności wykazuje wyraźnie rosnący kierunek wpływu liczby asocjacji na wielkość nakładów.

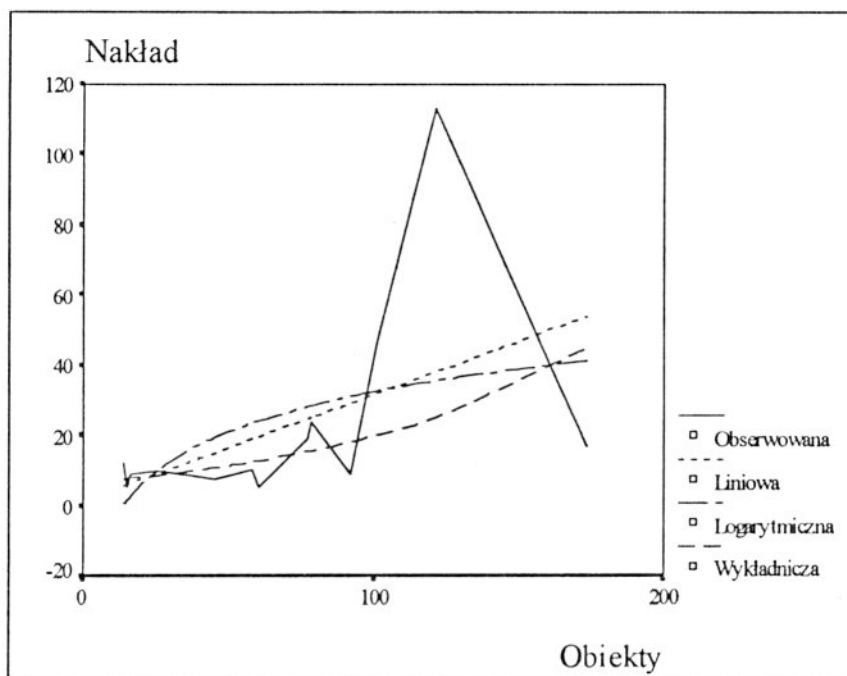


Rysunek 18. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i asocjacje. Źródło: [opracowanie własne].



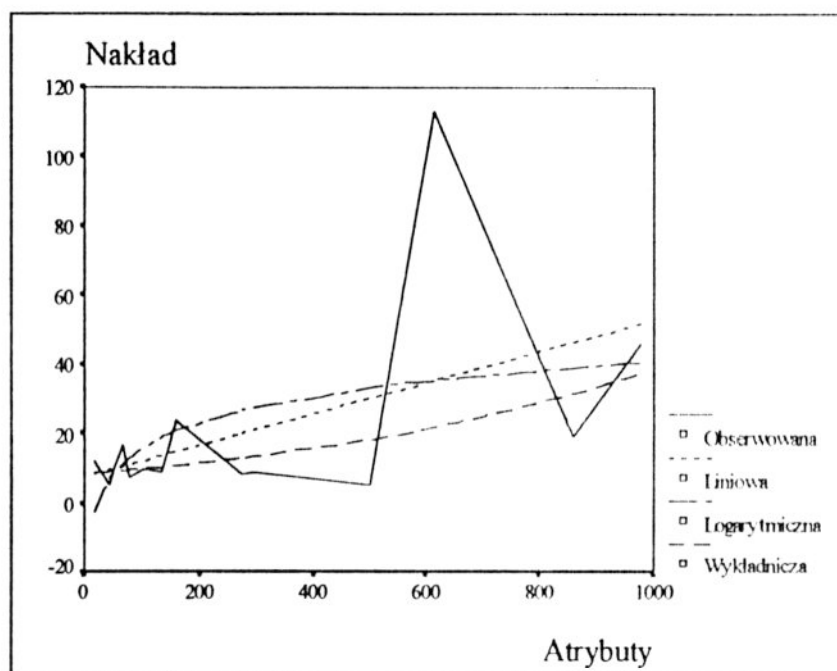
Rysunek 19. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i diagramy przypadków użycia. Źródło: [opracowanie własne].

Na wykresie zależności między *nakładem* i liczbą *diagramów przypadków użycia* (rys. 19) również zauważa się wyraźnie rosnący kierunek wpływu liczby *diagramów przypadków użycia* na wielkość *nakładów*. Ma on jeden wyraźny pik, który powoduje, że żadna z funkcji go nie obejmuje, ale dla pozostałych obserwacji najbliższy jest wykres funkcji wykładniczej.



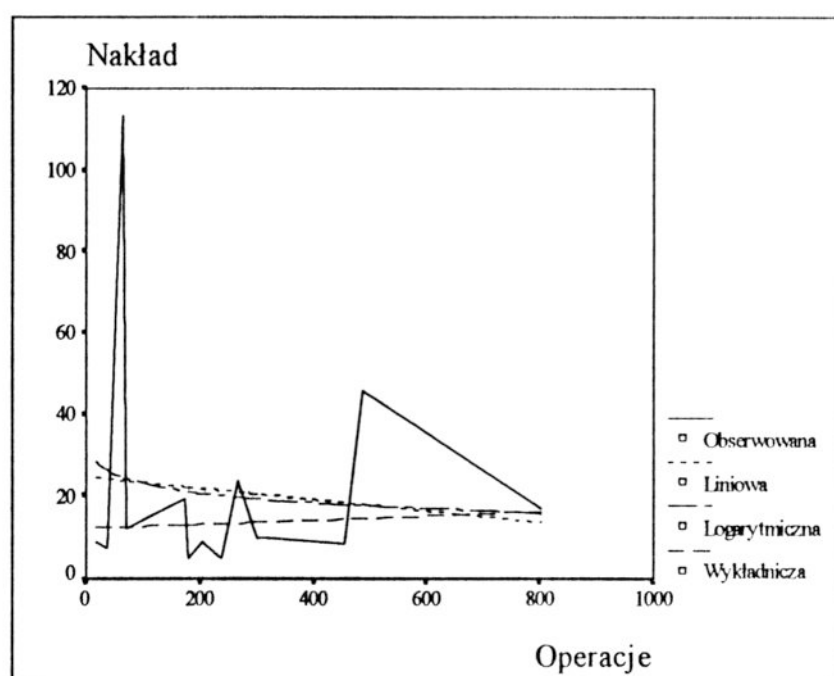
Rysunek 20. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą – nakłady i obiekty. Źródło: [opracowanie własne].

Wykres zależności między *nakładem* i liczbą *obiektów* (rys. 20) ma jeden wyraźny pik, który powoduje, że żadna z funkcji go nie obejmuje, ale dla pozostałych obserwacji również najbliższy jest wykres funkcji wykładniczej. Zauważa się wyraźnie rosnący kierunek wpływu liczby *obiektów* na wielkość *nakładów*.



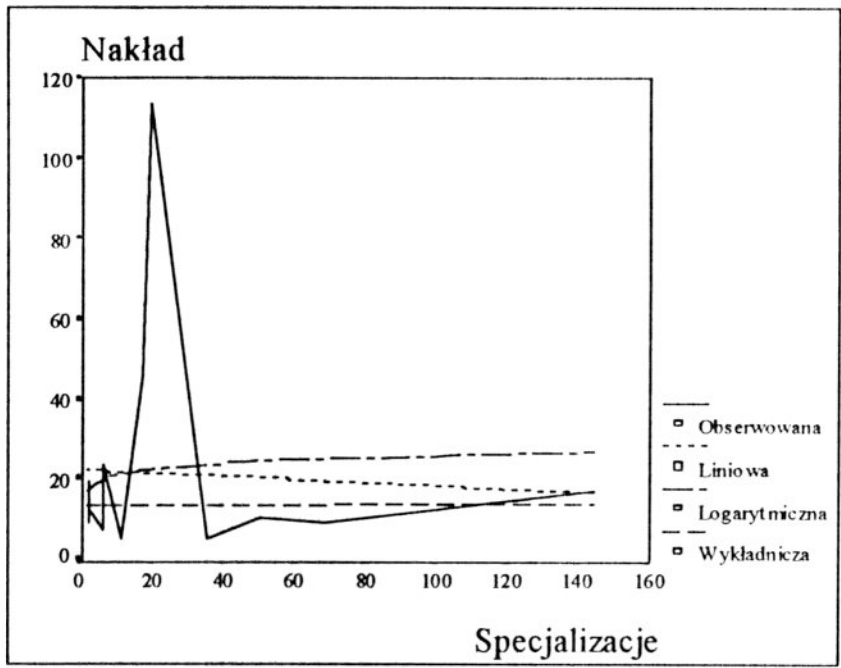
Rysunek 21. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i atrybuty. Źródło: [opracowanie własne].

Wykres zależności między *nakładem* i *atrybutami* (rys. 21) jest bardzo nieregularny, ma jeden wyraźny pik, ale wykazuje także tendencję wzrostową. Nieregularność tego wykresu utrudnia wybór najlepszego przybliżenia, ale wydaje się, że jest to wykres funkcji wykładniczej



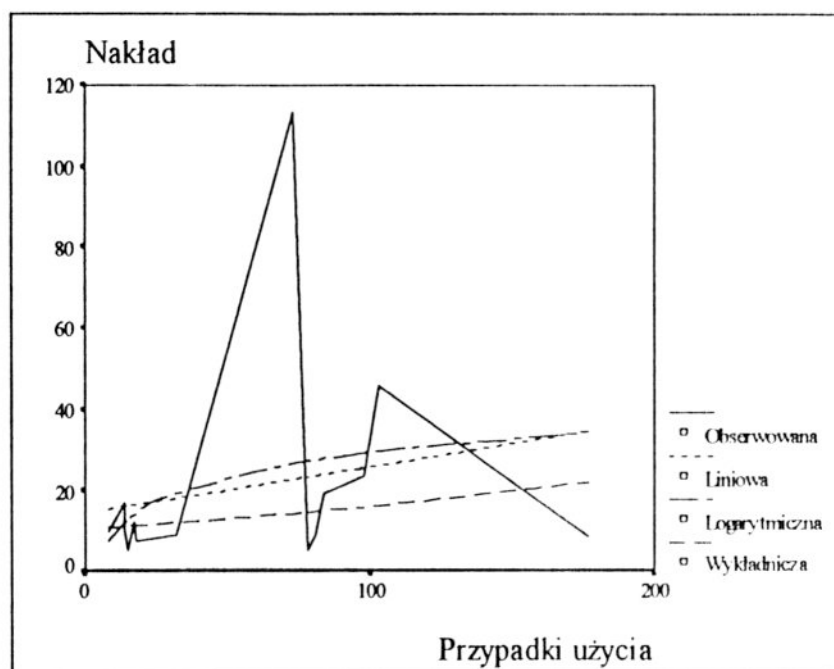
Rysunek 22. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i operacje. Źródło: [opracowanie własne].

Wykres zależności między *nakładem* i *operacjami* (rys. 22) podobnie jak wykres dla atrybutów jest bardzo nieregularny, ma dwa wyraźne piki, ale można też zauważyć jego tendencję wzrostową. Jako najlepsze przybliżenie wybrano też wykres funkcji wykładniczej.



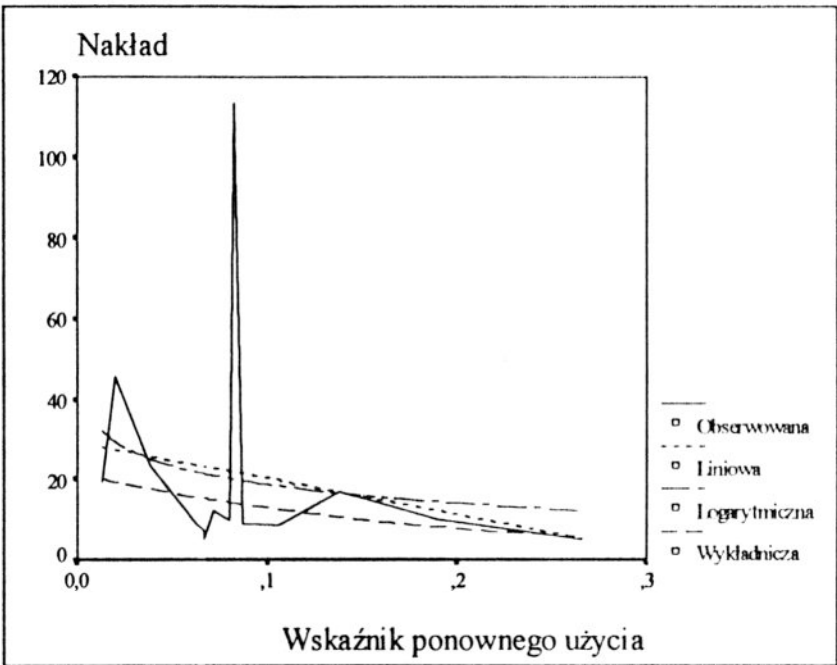
Rysunek 23. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i specjalizacje. Źródło: [opracowanie własne].

Wykres zależności między *nakładem* i liczbą związków *specjalizacji* (rys. 23) podobnie jak wykres dla zależności obiektów ma jeden wyraźny pik, a za najlepsze przybliżenie, wybrano również wykres funkcji wykładniczej. W przypadku porównywania projektów o takim samym zakresie, ale różnej liczbie związków *specjalizacji*, zależność ta powinna przedstawiać kierunek malejącego wpływu wzrostu liczby *specjalizacji* na wielkość *nakładów*. Jednak w przypadku różnych projektów, a takie znajdują się w badanej kolekcji, trudno jest ustalić kierunek tego wpływu. Wykres pokazuje wiele skoków, lecz na końcu tendencję wzrostową.



Rysunek 24. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i przypadki użycia. Źródło: [opracowanie własne].

Wykres zależności między *nakładem* i *przypadkami użycia* (rys. 24), podobnie jak wykres dla *operacji* jest bardzo nieregularny i ma dwa wyraźne piki. Jako najlepsze przybliżenie dla tej zależności wybrano wykres funkcji wykładniczej, który ma najwięcej punktów przecięcia z wykresem danych z obserwacji. Ostatni z wykresów zależności - *nakładem* i *wskaźnik ponownego użycia* (rys. 25), podobnie jak wykres dla *operacji* i *przypadków użycia*, charakteryzuje się nieregularnością i ma trzy piki. Mimo tego, ten wykres, jako jedyny z przedstawionych zależności, ma wyraźnie malejący kierunek wpływu na wielkość *nakładów*. Jako najlepsze przybliżenie dla tej zależności wybrano również wykres funkcji wykładniczej.



Rysunek 25. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i wskaźnik wewnętrznego ponownego użycia. Źródło: [opracowanie własne].

Analiza graficzna wszystkich wytypowanych zmiennych prowadzi do stwierdzenia nieliniowej postaci analitycznej związku między *nakładami* i atrybutami rozmiaru programu. Spośród wykreślonych postaci funkcji: liniowej, wykładniczej, logarytmicznej i wykładniczej wykresy o postaci funkcji wykładniczej, na podstawie analizy wzrokowej, przyjęto jako najlepsze przybliżenia krzywej obrazującej zależność wielkości *nakładów* od wytypowanych atrybutów rozmiaru. Kierunki wpływu wzrostu liczby tych atrybutów na wielkość *nakładów* pokrywa się z ich merytorycznym uzasadnieniem i przewidywaniami.

Tabela 27. Przekształcenie logarytmiczne wartości nakładów dla czternastu projektów. Źródło: [opracowanie własne].

Nr projektu	1	2	3	4	5	6	7
ln <i>Nakład</i>	2,20	2,20	1,61	2,01	2,12	1,61	2,26
Nr projektu	8	9	10	11	12	13	14
ln <i>Nakład</i>	3,82	2,94	3,16	2,82	4,73	2,30	2,48

Stwierdzona zatem, nieliniowa zależność *nakładów* względem wytypowanych zmiennych objaśniających jest opisana za pomocą funkcji wykładniczej w postaci ogólnego wzoru

$$N = \alpha_0 e^{\alpha_1 X_1} \varepsilon, \quad (6.14)$$

gdzie:

α_0, α_1 – parametry strukturalne,

ε – składnik losowy.

Nieliniowy charakter związków między *nakładem* a wytypowanymi atrybutami rozmiaru modelu programu został sprowadzony do postaci liniowej przez zlogarytmowanie obu stron równania (6.14). Otrzymano wówczas równanie liniowe postaci

$$V = \beta + \alpha_1 X_1 + \xi, \quad (6.15)$$

gdzie:

$$V = \ln N,$$

$$\beta = \ln \alpha_0,$$

$$\xi = \ln \varepsilon.$$

Funkcja wykładnicza po zlogarytmowaniu spełnia postulat liniowości względem parametrów strukturalnych. Stąd, dalsze postępowanie w procesie doboru zmiennych do modelu nieliniowego pozwoliło na wykorzystanie metod stosowanych do modeli liniowych, co wiązało się ze sprowadzeniem pierwotnych wartości zmiennej objaśnianej do postaci logarytmów, które nadano skróconą nazwę LNNAK (tab. 27).

Do ustalenia, od których atrybutów rozmiaru programu istotnie zależy wielkość, już teraz, logarytmu *nakładów* zastosowano metody taksonomiczne. Bazują one na progowej wartości współczynnika korelacji oraz odpowiednio zdefiniowanej progowej odległości. Jako progową wartość współczynnika korelacji przyjęto statystykę wyznaczoną wzorem

$$r^* = \sqrt{\frac{t_\alpha^2}{t_\alpha^2 + n - 2}}, \quad (6.16)$$

gdzie:

t_α - wartość odczytana z tablic dystrybucyj rozkładu Studenta dla $\nu = n - 2$ stopni swobody oraz ustalonego poziomu istotności α .

Jeżeli wartość bezwzględna współczynnika korelacji spełnia następującą nierówność

$$|r_{jk}| \leq r^*, \quad (6.17)$$

to brak jest podstaw do odrzucenia hipotezy o zerowej wartości danego współczynnika, a w przypadku odwrotnym, gdy

$$|r_{jk}| > r^*, \quad (6.18)$$

przyjęcia hipotezy o wartości równej jeden dla danego współczynnika.

Jedna grupa tych procedur opiera się na macierzy R (6.3) współczynników korelacji między zmiennymi objaśniającymi oraz na wektorze współczynników korelacji między zmiennymi objaśniającymi X_i a zmienną objaśnianą Y

$$r_0 = [r_{0j}] \quad (j = 1, \dots, m). \quad (6.19)$$

W metodach, w których stosuje się odpowiednio zdefiniowaną progową wartość odległości najczęściej stosowaną metryką odległości jest odległość Euklidesa, zdefiniowana następującym wzorem

$$d_{jk}^2 = \frac{\sum_{i=1}^n (x'_{ij} - x'_{ik})^2}{n} \quad (j, k = 1, \dots, m), \quad (6.20)$$

gdzie:

d_{jk} - odległość między j – tą oraz k – tą zmienną,

x'_{ij} oraz x'_{ik} - standaryzowane (na 0, 1) wartości zmiennych dla i – tej obserwacji.

Proces eliminacji skorelowanych zmiennych objaśniających przeprowadzono za pomocą dwóch metod. Pierwszej z nich podstawą jest macierz współczynników korelacji, a drugiej macierz odległości.

6. 3. 1. Metoda analizy macierzy współczynników korelacji

Zgodnie z zasadą metody pierwszej przeprowadzono weryfikację hipotezy o braku skorelowania między zmiennymi. W tabeli 28 przedstawiono współczynniki korelacji liniowej między wartościami logarytmu *nakładów* (zmienna o nazwie LNNAK) i atrybutami rozmiaru modelu programu.

Tabela 28. Współczynniki korelacji liniowej oraz poziomy istotności (w nawiasach) między atrybutami modelu programu oraz nakładem, którego wartości zostały zlogarytmowane. Znak (*) wskazuje wartości świadczące o istnieniu korelacji między danymi znajdującymi się na przecięciu wiersza i kolumny. Źródło: [opracowanie własne].

	LNNAK	ASSO	ATT	DOBJ	DSEQ	DUSE	OBJ	OPER	SPEC	USCA	WWPU
LNNAK	1,000										
ASSO	0,812* (0,000)	1,000									
ATT	0,564* (0,036)	0,493 (0,073)	1,000								
DOBJ	0,635* (0,015)	0,825* (0,000)	0,500 (0,069)	1,000							
DSEQ	0,513 (0,061)	0,283 (0,327)	0,513 (0,061)	0,453 (0,104)	1,000						
DUSE	0,801* (0,001)	0,520 (0,057)	0,723* (0,003)	0,483 (0,080)	0,813* (0,000)	1,000					
OBJ	0,598* (0,024)	0,886* (0,000)	0,370 (0,193)	0,943* (0,000)	0,361 (0,205)	0,390 (0,168)	1,000				
OPER	0,087 (0,767)	0,435 (0,120)	0,059 (0,840)	0,580* (0,030)	0,274 (0,343)	-0,031 (0,916)	0,603* (0,022)	1,000			
SPEC	0,008 (0,980)	0,474 (0,087)	-0,170 (0,562)	0,626* (0,017)	-0,076 (0,795)	-0,270 (0,350)	0,740* (0,002)	0,694* (0,006)	1,000		
USCA	0,241 (0,407)	0,254 (0,380)	0,548* (0,042)	0,278 (0,335)	0,372 (0,190)	0,430 (0,125)	0,149 (0,610)	0,186 (0,525)	-0,213 (0,465)	1,000	
WWPU	-0,402 (0,154)	-0,205 (0,483)	-0,556* (0,039)	-0,277 (0,338)	-0,479 (0,083)	-0,549 (0,042)	-0,149 (0,610)	0,142 (0,629)	0,291 (0,312)	-0,396 (0,161)	1,000

Analiza macierzy korelacji przedstawionej w tabeli 28 pozwala na przeprowadzenie testu o istnieniu liniowej zależności między każdymi dwoma zmiennymi, podobnie jak dla macierzy przedstawionej w tabeli 24. W pierwszym rzędzie każdego wiersza tabeli zamieszczono wartości współczynników korelacji, a w drugim poziomy istotności. Za pomocą znaku (*) zaznaczono wszystkie istotne korelacje między badanymi zmiennymi. W porównaniu do macierzy współczynników z tabeli 24 wystąpiła zmiana jedynie w pierwszej kolumnie, która zawiera współczynniki korelacji logarytmicznej transformacji *nakładów* (LNNAK) z atrybutami rozmiaru. Wartości współczynników korelacji wzrosły, a współczynniki atrybutów *operacje* oraz *specjalizacje* zmieniły znak na przeciwny, chociaż ich wartości bezwzględne zmalały. Oznacza to, że wpływają one na zwiększenie wielkości logarytmu *nakładów*, a nie na zmniejszenie, jak to było w przypadku wielkości *nakładów*. Z malejącym kierunkiem wpływu pozostał atrybut ponownego użycia (WWPU), chociaż wartość bezwzględna jego współczynnika korelacji z logarytmem *nakładów* wzrosła prawie dwukrotnie. Natomiast poziomy istotności dla większości zmiennych odpowiednio zmalały, a wzrosły tylko dla *operacji* i *specjalizacji*. Efektem tego jest wzrost liczby atrybutów z dwóch do pięciu, z którymi logarytm *nakładów* jest istotnie skorelowany w porównaniu do *nakładów*. Analiza korelacji między samymi atrybutami rozmiaru prowadzi do otrzymania takich samych wyników, jak w przypadku macierzy współczynników zawartej w tabeli 24.

Ponownie, zatem, zweryfikowano hipotezę H_0 o braku skorelowania między wielkościami logarytmicznymi *nakładów* a atrybutami rozmiaru na ustalonym poziomie $\alpha = 0,05$. Wartości poziomów istotności dla następujących atrybutów: *asocjacje* (ASSO), *atrybuty* (ATT), *diagramy obiektów* (DOBJ), *diagramy przypadków użycia* (DUSE), *obiekty* (OBJ) nie dają podstaw do odrzucenia hipotezy H_0 o braku skorelowania z *nakładami* w postaci zlogarytmowanej (LNNAK). Na tej postawie sformułowano wniosek o istotnym wpływie tych atrybutów na wielkość logarytmu *nakładów*. Jednak, ze względu na istotne skorelowanie, jakie występuje również wśród powstałej, ze względu na istnienie skorelowania, grupy atrybutów: *diagramy obiektów*, *asocjacje* i *obiekty* zachodzi potrzeba wprowadzenia do modelu *nakładów* tylko reprezentanta tej grupy.

Druga metoda analizy macierzy współczynników korelacji wymagała konstrukcji grafu [PLU69], [GRA82], którego wierzchołkami są zmienne, a łukami współczynniki korelacji. Rozwiązanie zależy od wstępnego uporządkowania zmiennych objaśniających, gdyż w procedurze najpierw uwzględnia się kryterium skorelowania zmiennych objaśniających ze zmienną objaśnianą, a dopiero potem stosuje się kryterium wzajemnego nieskorelowania tych zmiennych.

W pierwszym kroku uporządkowano atrybuty rozmiaru według malejących wartości bezwzględnych ich współczynnika korelacji z logarytmem nakładów (tab. 29).

Tabela 29. Wykaz zmiennych objaśniających, uporządkowany według wartości bezwzględnej malejącego współczynnika korelacji liniowej ze zmienną objaśnianą. Źródło: [opracowanie własne].

Kolejność	Zmienna X_i	Współczynnik r_{ij}
1	ASSO	0,812
2	DUSE	0,801
3	DOBJ	0,635
4	OBJ	0,598
5	ATT	0,564
6	DSEQ	0,513
7	WWPU	-0,402
8	USCA	0,241
9	OPER	0,087
10	SPEC	0,008

Następnie ustalono wartość krytyczną $r^* = 0.5324$, korzystając z tablic dystrybucyj rozkładu Studenta, gdzie odczytano wartość $t_\alpha = 2.179$ dla poziomu istotności $\alpha = 0.05$ oraz $\nu = n - 2 = 12$. Zgodnie ze wzorem (6.17), przystąpiono do weryfikacji hipotezy H_0 o braku skorelowania między zmienną objaśnianą a kolejno wszystkimi zmiennymi objaśniającymi w postaci

$$H_0 : |r_{jk}| \leq r^* \quad (6.21)$$

i hipotezy alternatywnej (według wzoru 6.18) o istnieniu skorelowania między zmienną objaśnianą a kolejnymi zmiennymi objaśniającymi w postaci

$$H_1 : |r_{jk}| > r^*, \quad (6.22)$$

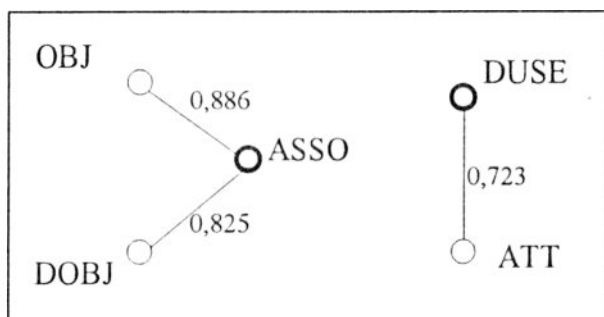
przy następujących warunkach: $\alpha = 0,05$, $\nu = 12$.

Wobec braku podstaw do odrzucenia hipotezy H_0 w odniesieniu do pięciu zmiennych (od numeru 6 do 10) spośród wyjściowego zbioru dziesięciu zmiennych (tab. 29) zostały one z niego wyeliminowane, ponieważ ich współczynniki korelacji ze zmienną objaśnianą spełniają nierówność (6.21). W odniesieniu do pierwszych pięciu hipoteza H_0 została odrzucona, a przyjęta hipoteza alternatywna H_1 , ponieważ ich współczynniki spełniają nierówność (6.22).

Oznacza to, że otrzymano zredukowany zbiór zmiennych, utworzony przez pięć zmiennych: *asocjacje* (ASSO), *diagramy przypadków użycia* (DUSE), *diagramy obiektów* (DOBJ), *obiekty* (OBJ) oraz *atrybuty* (ATT), które są istotnie skorelowane ze zmienną objaśnianą (rys.26). Wynik ten jest potwierdzeniem wniosków z analizy macierzy współczynników korelacji.

W drugim kroku metody ze zbioru zredukowanych zmiennych wybrano zmienną maksymalnie skorelowaną ze zmienną objaśnianą, którą są *asocjacje*.

W trzecim kroku do nowego zbioru zmiennych dołączono również zmienne, które nie są istotnie skorelowane z *asocjacjami*: *diagramy przypadków użycia* (DUSE) i *atrybuty* (ATT). Weryfikacja hipotezy H_0 wobec tych zmiennych (6.21) w wyniku odpowiadających im nierówności $0,520 < 0,5324$ oraz $0,493 < 0,5324$ prowadzi do stwierdzenia braku podstaw do jej odrzucenia.



Rysunek 26. Graf połączeń zredukowanego zbioru zmiennych objaśniających otrzymany przy zastosowaniu metody graficznej analizy współczynników korelacji. Źródło: [opracowanie własne].

Jednakże zmienne *diagramy przypadków użycia* oraz *atrybuty* są wzajemnie istotnie skorelowane, gdyż ich współczynnik korelacji jest $0,723 > 0,5324$. Zatem istnieją podstawy do odrzucenia hipotezy o braku ich skorelowania. Stąd, w skład nowego zbioru zmiennych objaśniających weszły tylko dwie zmienne: *asocjacje* oraz *diagramy przypadków użycia* (na rysunku 26 zaznaczone linią pogrubioną).

6. 3. 2. Metoda taksonomii wrocławskiej

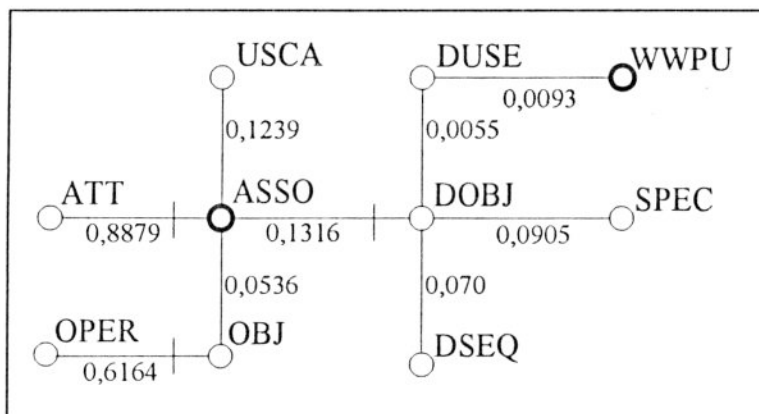
Podstawą konstrukcji grafu w metodzie taksonomii wrocławskiej [FLO51] jest macierz odległości między klasyfikowanymi zmiennymi. W tabeli 30 zamieszczono macierz odległości klasyfikowanych zmiennych skonstruowaną według metryki określonej wzorem (6.18) przy wykorzystaniu metody różnic. Wartości metryk odległości zostały przeskalowane w zakresie 0 – 1. Proces doboru zmiennych dla obu macierzy przeprowadzono w następujących etapach.

Tabela 30. Macierz odległości klasyfikowanych atrybutów rozmiaru i logarytmu nakładów, zbudowana według metryki Euklidesa. Źródło: [opracowanie własne].

	LNNAK	ASSO	ATT	DOBJ	DSEQ	DUSE	OBJ	OPER	SPEC	USCA
ASSO	0,1424									
ATT	0,9951	0,8879								
DOBJ	0,0060	0,1316	0,9861							
DSEQ	0,0773	0,1177	0,9402	0,0703						
DUSE	0,0047	0,1365	0,9868	0,0055	0,0682					
OBJ	0,1739	0,0536	0,8745	0,1620	0,1390	0,1686				
OPER	0,7636	0,6560	0,8416	0,7531	0,7197	0,7605	0,6164			
SPEC	0,0998	0,1054	0,9725	0,0905	0,1129	0,1007	0,1091	0,6836		
USCA	0,1654	0,1239	0,8676	0,1570	0,1315	0,1594	0,1362	0,6615	0,1676	
WWPU	0,0000	0,1479	1,0000	0,0116	0,0815	0,0093	0,1794	0,7683	0,1030	0,1702

Etap I.

Z każdego wiersza odległości wybrano najmniejszy element wskazujący parę zmiennych leżących najbliżej siebie. Otrzymane połączenia przedstawiono w postaci grafu nieorientowanego (rys. 27), w którym na łukach opisano miary przeskalowanych odległości.



Rysunek 27. Graf połączeń zmiennych objaśniających przy zastosowaniu metody taksonomii wrocławskiej.
Źródło: [opracowanie własne].

Etap II.

Etap ten polega na sprawdzeniu spójności otrzymanego grafu odległości. Graf przedstawiony na rys. 27 jest spójny, ponieważ każde dwa różne wierzchołki połączone są nieprzerwanym ciągiem krawędzi.

Etap III.

Przeprowadzono podział zbioru zmiennych na k skupisk. Liczba k została wyznaczona na podstawie określonego algorytmu [GRA82, s. 139] i obliczono, że $k = 4$. W celu uzyskania podziału zbioru klasyfikowanych zmiennych na cztery skupiska odrzucono 4 – 1 najdłuższych krawędzi (na rys. 27 krawędzie odrzucone zaznaczono znakiem „|”). Otrzymano następujące cztery skupiska:

- z trzema zmiennymi: *asocjacje*, *obiekty*, *przypadki użycia*,
- z pięcioma zmiennymi: *diagramy obiektów*, *diagramy przypadków użycia*, *diagramów sekwencji*, *specjalizacje*, *wskaźnik ponownego użycia*,
- dwa jednoelementowe: *operacje* oraz *atrybuty*.

Etap IV.

Z każdego skupiska wybrano reprezentantów na podstawie analizy odległości tych zmiennych od logarytmu *nakładów*. Zostały wybrane te atrybuty rozmiaru, które mają najmniejsze odległości. Z dwóch skupisk jednoelementowych do zbioru optymalnych zmiennych objaśniających powinny być automatycznie zakwalifikowano: *operacje* oraz *atrybuty*. Jednak, ze względu na duże ich odległości od zmiennej objaśnianej nie zostały one dołączone do nowego zbioru zmiennych (tab. 30). Ze skupiska pięcioelementowego wybrano *wskaźnik ponownego użycia*, jako zmienną o najmniejszej odległości od zmiennej objaśnianej. Ze skupiska trzejelementowego wybrano, jako jego reprezentanta, *asocjacje*, również ze

względem na najmniejszą odległość od logarytmu nakładów. Zatem zbiór optymalnych zmiennych objaśniających zawiera:

- *wskaźnik ponownego użycia*,
- *asocjacje*.

Z powodu uzyskania dwóch różnych rozwiązań zastosowano do wyboru optymalnego zbioru zmiennych objaśniających jeszcze metodę trzecią, nazywaną metodą selekcji stopniowej (Stepwise selection), odpowiedniej w przypadku występowania istotnej korelacji między zmiennymi objaśniającymi.

6. 3. 3. Metoda selekcji stopniowej

W wyniku działania metody selekcji stopniowej do zbioru optymalnych zmiennych, spośród dziesięciu potencjalnych zmiennych zostały włączone dwie zmienne: *asocjacje* i *diagramy przypadków użycia*. Zmienna *asocjacje* została wprowadzona w pierwszym kroku ze względu na najsilniejszą korelację z logarytmem *nakładów* (tab. 28). W dalszych krokach metody zmienne są dołączane lub usuwane ze względu na krytyczne wartości statystyki t . Dlatego też ustalono, korzystając z tablic dystrybucji rozkładu Studenta dla poziomu istotności $\alpha = 0.05$ oraz stopni swobody $\nu = n - 2 = 12$ wartość krytyczną $t_{\alpha 1} = 2,179$ do wprowadzania zmiennych do zbioru zmiennych optymalnych. W ten sam sposób dla poziomu istotności $\alpha = 0,10$ ustalono wartość krytyczną $t_{\alpha 2} = 1,782$ do usuwania ze zbioru zmiennych już dołączonych. Zmienne dołączone mogą zostać usunięte, podobnie jak w metodzie analizy współczynników korelacji, ze względu na istnienie korelacji między nimi. W tabeli 31 zamieszczono wartości statystyk t dla poszczególnych zmiennych w kolejnych krokach metody już po dołączeniu *asocjacji* do zbioru zmiennych optymalnych.

Tabela 31. Wartości statystyk *t* dla zmiennych objaśniających w trzech krokach metody selekcji stopniowej.
Źródło: [opracowanie własne].

Krok	Zmienna	Statystyka <i>t</i>	Poziom istotności	Współczynnik korelacji częściowej
1	ATT	1,128	0,283	0,322
	DOBJ	-0,353	0,731	-0,106
	DSEQ	1,944	0,078	0,506
	DUSE	3,880	0,003	0,760
	OBJ	-1,670	0,123	-0,450
	OPER	-1,957	0,076	-0,508
	SPEC	-3,608	0,004	-0,736
	USCA	0,203	0,843	0,061
	WWPU	-1,504	0,161	-0,413
2	ATT	-1,024	0,330	-0,308
	DOBJ	-0,985	0,348	-0,297
	DSEQ	-0,979	0,351	-0,296
	OBJ	-1,786	0,104	-0,492
	OPER	-1,441	0,180	-0,415
	SPEC	-1,607	0,139	-0,453
	USCA	-1,189	0,262	-0,352
	WWPU	-0,060	0,954	-0,019
3	ATT	-1,024	0,330	-0,308
	DOBJ	-0,985	0,348	-0,297
	DSEQ	-0,979	0,351	-0,296
	OBJ	-1,786	0,104	-0,492
	OPER	-1,441	0,180	-0,415
	SPEC	-1,607	0,139	-0,453
	USCA	-1,189	0,262	-0,352
	WWPU	-0,060	0,954	-0,019

W drugim kroku metody została dołączona zmienna *diagramy przypadków użycia* ze względu na największą wartość statystyki *t*. W trzecim kroku nie dołączono już żadnej zmiennej, ponieważ wszystkie wartości statystyk były mniejsze od wartości krytycznej, co zakończyło proces dołączania zmiennych. Również nie usunięto żadnej z wprowadzonych zmiennych, gdyż wartość statystyki *t* dla obu zmiennych nie jest mniejsza niż 1,782.

6. 4. Estymacja parametrów modelu nakładów

W wyniku przeprowadzenia procesu doboru zmiennych na podstawie zbioru potencjalnych zmiennych objaśniających, korzystając z algorytmów wybranych trzech metod, otrzymano dwa różne rozwiązania z dwoma zmiennymi (tab. 32): rozwiązanie *A* i rozwiązanie *B*, z których pierwsze zostało otrzymane w wyniku zastosowania dwóch różnych metod.

Tabela 32. Podzbiory zmiennych objaśniających otrzymane metodą analizy współczynników korelacji, taksonomii wrocławskiej oraz selekcji stopniowej. Źródło: [opracowanie własne].

L.P.	Metoda	Wykaz zmiennych	Rozwiązanie
1	Współczynników korelacji	ASSO, DUSE	A
2	Taksonomii wrocławskiej	ASSO, WWPU	B
3	Selekcji stopniowej	ASSO, DUSE	A

Rozwiązanie *A* otrzymane na podstawie analizy macierzy współczynników korelacji pozwala na wprowadzenie do modelu nakładów dwóch zmiennych, a rozwiązanie *B*, otrzymane na podstawie analizy macierzy odległości również dwie zmienne, z których jedna pokrywa się z zestawem zmiennych z rozwiązania *A*. Zakwalifikowane do zbioru rozwiązania *A* i *B* *asocjacje* stanowią atrybut złożoności strukturalnej rozmiaru modelu programu i są elementem składowym modelu obiektów. *Diagramy przypadków użycia* są natomiast atrybutem długości. Zmienna, zakwalifikowana do rozwiązania *B*, *wskaźnik ponownego użycia* należy do atrybutów ponownego użycia i charakteryzuje poziom dziedziczenia wykorzystany w budowie danego programu.

Proces estymacji modelu nakładów przeprowadzono dla obu otrzymanych wariantów rozwiązań ze względu na przypuszczenie o istotnym wpływie na kształtowanie się wielkości nakładów *wskaźnika ponownego użycia*, który wystąpił jedynie w rozwiązaniu *B*. Sądzi się, że dalsze badania prowadzone w tym kierunku, z większą liczbą obserwacji i projektów bardziej zaawansowanych w stosowaniu technologii obiektowej niż projekty badane, pozwolą na otrzymanie takiego rozwiązania z lepszymi statystykami.

W celu potwierdzenia słuszności wyboru nieliniowej postaci funkcji *nakładów* proces ustalania analitycznej postaci ich zależności od otrzymanych wariantów rozwiązań przeprowadzono za pomocą dwóch funkcji: liniowej i nieliniowej. Wyniki estymacji parametrów i otrzymane statystyki zamieszczono w sumarycznych tabelach.

Tabela 33. Współczynniki modeli estymacji nieliniowej otrzymanych metodą najmniejszych kwadratów dla rozwiązań A i B. Źródło: [opracowanie własne].

Wariant modelu		Współczynniki niestandardyzowane		Współczynniki standaryzowane	Statystyka t	Poziom istotności
		B	Błąd standardowy	Beta		
A	(Stała)	1,650	0,152		10,873	0,000
	ASSO	1,1E-02	0,003	0,543	4,061	0,002
	DUSE	8,8E-02	0,023	0,519	3,880	0,003
B	(Stała)	2,114	0,306		6,917	0,000
	ASSO	1,5E-02	0,003	0,762	4,660	0,001
	WWPU	-3,118	2,074	-0,246	-1,507	0,161

Estymację parametrów poszukiwanych wariantów modelu przeprowadzono za pomocą metody najmniejszych kwadratów, a obliczenia wykonano z wykorzystaniem programu SPSS. Na potrzeby stwierdzenia możliwości zastosowania tej metody do estymacji modelu nakładów, po wykonaniu procedury oceny parametrów, przeprowadzono weryfikację następujących założeń:

- rozkład normalny składnika losowego modelu,
- jednorodność wariancji składnika losowego,
- niezależność składnika losowego (brak autokorelacji),

za pomocą testów, których opis zamieszczono w punkcie 6.5.3.

Wykonano szacowanie parametrów następujących postaci modelu:

a) liniowej z dwoma zmiennymi

$$N = \alpha_0 + \alpha_1 X_1 + \alpha_2 X_2 + \varepsilon, \tag{6.23}$$

b) nieliniowej z dwoma zmiennymi

$$N = \alpha_0 e^{\alpha_1 X_1} e^{\alpha_2 X_2} \varepsilon. \tag{6.24}$$

Nieliniowy charakter związków między *nakładem* a wytypowanymi czynnikami rozmiaru modelu, jak to wcześniej zdecydowano, został sprowadzony do postaci liniowej. W tym celu funkcję nakładów z dwoma zmiennymi objaśniającymi X_1, X_2 o postaci wykładniczej przekształcono do postaci liniowej przez zlogarytmowanie obu stron równania (6.24). Stąd, dalsze postępowanie polegało na ocenie modelu nieliniowego *nakładów* za pomocą liniowego modelu *logarytmu nakładów*.

Otrzymano równanie liniowe postaci

$$V = \beta + \alpha_1 X_1 + \alpha_2 X_2 + \xi, \quad (6.25)$$

gdzie: $V = \ln N$, $\beta = \ln \alpha_0$, $\xi = \ln \varepsilon$.

Transformacja równania (6.24) pociąga za sobą transformację danych rzeczywistych z tabeli 21, odpowiadających zmiennej objaśnianej, czyli nakładowi (NAKŁAD) przez zlogarytmowanie zbioru wartości nakładów (tab. 27).

Model nakładów na podstawie wyników obliczeń procedury metody najmniejszych kwadratów (tab. 33) i równania (6.23) dla wariantu A ma postać

$$V = 1,650 + 0,011 * asocjacje + 0,088 * diagramy przypadków użycia. \quad (6.26)$$

Równanie tego modelu, po wykonaniu retransformacji do postaci pierwotnej wykładniczej, czyli sprowadzeniu równania liniowego (6.26) do postaci nieliniowej zależności *nakładów* od ustalonych atrybutów rozmiaru modelu programu ma postać

$$N = 5,207e^{0,011*asocjacje + 0,088*diagramyprzypadkówużycia}, \quad (6.27)$$

gdzie N – *nakład*.

W przypadku wariantu B model estymacji ma postać

$$V = 2,114 + 0,015 * asocjacje - 3,118 * wskaźnik ponownego użycia. \quad (6.28)$$

Po wykonaniu retransformacji do postaci nieliniowej równanie (6.28) przybiera postać

$$N = 8,28e^{0,015*asocjacje - 3,118*wskaźnikponownegoużycia}, \quad (6.29)$$

w którym N – *nakład*.

6. 5. Weryfikacja modelu nakładów

W tabeli 34 zawarto statystyki opisowe modelu liniowego i nieliniowego *nakładów* (ale liniowego logarytmu *nakładów*) w dwóch wariantach A i B. W dalszym ciągu pracy przyjęto stosowanie powyższego nazewnictwa dla odróżnienia odwoływania się do tych dwóch postaci modeli.

Tabela 34. Statystyki opisowe dla modelu liniowego i nieliniowego rozwiązań *A* i *B*. W przypadku modelu liniowego zmienną objaśnianą jest wielkość nakładu z obserwacji (NAKLAD), a w przypadku nieliniowego zlogarytmowana wielkość nakładu pod nazwą LNNAK. Źródło: [opracowanie własne].

Wariant	Zmienne	Średnia		Standardowe odchylenie		Liczba obserwacji
		Model liniowy	Model nieliniowy	Model liniowy	Model nieliniowy	
<i>A</i>	NAKLAD	20,9393		28,5118		14
	LNNAK		2,5899		0,8577	14
	ASSO	50,5700	50,5700	43,3500	43,3500	14
	DUSE	4,5000	4,5000	5,0500	5,0500	14
<i>B</i>	NAKLAD	20,9393		28,5118		14
	LNNAK		2,5899		0,8577	14
	ASSO	50,5700	50,5700	43,3500	43,3500	14
	WWPU	9,2E-02	9,2E-02	6,8E-2	6,8E-02	14

Wszystkie czternaście wartości obserwacji z próby odpowiadają występującym w obu wariantach modelu zmiennym objaśniającym. W modelu nieliniowym dla wariantu *A* wartość odchylenia standardowego logarytmu *nakładów* jest mniejsza od ich wartości średniej, natomiast w modelu liniowym wartość standardowego odchylenia *nakładów* przekracza wartość średnią, co wskazuje na duży przedział zmienności w obserwacjach *nakładów*. Podobnie sytuacja wystąpiła w obu postaciach modeli dla wariantu *B*. Współczynniki korelacji liniowej między zmiennymi objaśniającymi i *nakładem* są zawarte w tabeli 24, a między zmiennymi objaśniającymi i logarytmem *nakładów* w tabeli 28. Zmienna, która występuje w modelu zarówno w wariancie *A* i *B*, *asocjacje* oraz *diagramy przypadków użycia*, dołączona tylko w wariancie *A*, są istotnie skorelowane zarówno z *nakładem* jak i logarytmem *nakładów*. Obie są znaku dodatniego, zatem wpływają rosnąco na obie wielkości. W wariancie *B*, otrzymanym w wyniku analizy macierzy odległości, *wskaźnik ponownego użycia* charakteryzuje się najmniejszą odległością od logarytmu *nakładów* (w tabeli miara odległości = 0,0000, co oznacza, że jest mniejsza niż 0,0005), ale wartość jego współczynnika korelacji nie daje podstaw do odrzucenia hipotezy o braku jego korelacji ze zmienną objaśnianą (wartość współczynnika korelacji wynosi - 0,402). Wartość ujemna tego współczynnika wskazuje na to, że jest to atrybut, który powoduje zmniejszanie wielkości zarówno *nakładów*, jak i logarytmu *nakładów*, co jest zgodne z wcześniejszymi przewidywaniami. Natomiast *asocjacje* oraz *diagramy przypadków użycia*, jako zmienne istotnie skorelowane z *nakładem* i logarytmem *nakładów* również posiadają wartości miar wskazujące na ich małą odległość od

logarytmu *nakładów*, a przez dodatni kierunek wpływu powodują wzrost jego wielkości, co też pokrywa się z uzasadnieniem merytorycznym takiego wpływu na wielkość *nakładów*.

6. 5. 1. Analiza dopasowania modelu do danych z obserwacji

Wyniki procedury analizy regresji dla postaci nieliniowej modelu *nakładów*, ale liniowej względem logarytmu *nakładów* (w tabeli nazywanym modelem nieliniowym) oraz, dla porównania, postaci liniowej modelu *nakładów* (w tabeli nazywanym modelem liniowym) zamieszczono w tabeli 35. Wartość *R* odpowiada za korelację między obserwowaną i przewidywaną wartością zmiennej objaśnianej. Wartość *R*² jest kwadratem tej korelacji i stanowi miarę dopasowania liniowego modelu, często nazywana „współczynnikiem determinacji”.

W modelu nieliniowym, odpowiadającym wariantowi *A* wartość *R* wskazuje na wyższy stopień korelacji niż dla wariantu *B*. W wariancie pierwszym modelu obie zmienne wyjaśniają w 86% zmienność logarytmu *nakładów*, a w drugim tylko 72%. Wyniki odpowiadające liniowej postaci modelu są ponad 0,1 gorsze od wyników dla modelu nieliniowego, co tłumaczy się tym, że model liniowy w wariancie *A* wyjaśnia zmienność *nakładu* przez zmienne tylko w 70%, a w wariancie *B* tylko w 60%.

Tabela 35. Wartości parametrów oceny modeli w wariancie *A* i *B*. Źródło: [opracowanie własne].

Wariant	Zmienne	<i>R</i>		<i>R</i> ²		$\overline{R^2}$		Standardowy błąd estymacji	
		Liniowy	Nieliniowy	Liniowy	Nieliniowy	Liniowy	Nieliniowy	Liniowy	Nieliniowy
<i>A</i>	DUSE ASSO	0,834	0,925	0,695	0,857	0,639	0,830	17,1232	0,3532
<i>B</i>	WWPU ASSO	0,774	0,847	0,599	0,718	0,526	0,667	19,6392	0,4951

Występujący w ocenach modeli skorygowany współczynnik $\overline{R^2}$ odzwierciedla, jak dobrze model jest dopasowany do danych z obserwacji. Jego wartość oblicza się na podstawie równania, które jest funkcją *R*² oraz liczby zmiennych w modelu i liczby obserwacji

$$\overline{R^2} = R^2 - \frac{p(1 - R^2)}{N - p - 1},$$

(6.30)

gdzie:

N - liczba obserwacji,

p - liczba zmiennych objaśniających w modelu.

Również wartość tej statystyki wskazuje na najlepsze dopasowanie do danych empirycznych modelu postaci nieliniowej, zarówno odpowiadającemu wariantowi *A*, jak i *B*.

Podobna ocena wynika też z analizy wartości standardowych błędów estymacji. Wskaźnik ten jest miarą rozpiętości residuów (błędów) wokół krzywej estymacji. Jego jednostką miary jest jednostka odpowiadająca zmiennej objaśnianej. Jeśli ta wartość nie jest mniejsza niż standardowe odchylenie tej zmiennej, to model otrzymany w wyniku analizy regresji nie jest lepszy niż wartość średnia zastosowana do przewidywania wartości zmiennej objaśnianej. Wartość standardowego odchylenia logarytmu *nakładów* wynosi 0.8577 (tab. 34), a wartości standardowych błędów estymacji dla nieliniowego modelu są znacznie mniejsze niż wartość standardowego odchylenia zarówno w wariancie *A* i *B*. Procentowo, standardowy błąd estymacji wynosi odpowiednio 41% i 58% wartości odchylenia standardowego. W przypadku modelu liniowego również wartości standardowych błędów są mniejsze od odchylenia standardowego nakładów, które wynosi 28,5118, ale ta różnica jest znacznie mniejsza, bo procentowo dla rozwiązania *A* i *B* wynosi odpowiednio 60% i 69%.

Wartości statystyk przewidywania wielkości logarytmu *nakładów*, jak to wynika z dotychczas przeprowadzonej analizy, są lepsze niż z przewidywania wielkości *nakładów*, dlatego ze względu na słabszą ocenę statystyk modelu liniowego dalsze postępowanie dotyczyło już tylko modelu nieliniowego.

Weryfikację modelu nieliniowego spróbowano też przeprowadzić na podstawie analizy wariancji, ale najpierw sprawdzono wymagane dla tej analizy założenia. Dla wytypowanych zmiennych w wariancie *A* i *B* nie powiodło się sprawdzenie obu warunków. Pierwsze założenie o normalności rozkładu zmiennych zostało pozytywnie zweryfikowane za pomocą testu *Kolmogorowa – Smirnowa*, wyniki przedstawiono w tabelach 25 i 26. Natomiast drugie wymagane założenie o równości wariancji było możliwe do sprawdzenia tylko wobec zmiennej *diagramy przypadków użycia*. Do weryfikacji hipotezy o równości wariancji w grupie przeprowadzono test Lewena, w którym statystyka osiągnęła wartość $F = 0,643$ na poziomie istotności $\alpha = 0,675$. Na tej podstawie hipoteza o równości wariancji na poziomie istotności $\alpha = 0,05$ została przyjęta. Dla *asocjacji* oraz *wskaźnika ponownego użycia* hipoteza ta nie została zweryfikowana z powodu zbyt małej liczby obserwacji przypadających na ustalone grupy. Ponieważ nie otrzymano oceny negatywnej ani pozytywnej wykonano analizę wariancji, ale bez wykorzystania jej wyników do zmiany modelu.

Tabela 36. Wyniki analizy wariancji dla modelu nieliniowego w wariancie *A* i *B*. Źródło: [opracowanie własne].

Wariant		Suma kwadratów	Df	Kwadrat średniej	Statystyka F	Poziom istotności
<i>A</i>	Regresja	8,191	2	4,095	32,829	0,000
	Residuum	1,372	11	0,125		
	Ogółem	9,563	13			
<i>B</i>	Regresja	6,867	2	3,434	14,010	0,001
	Residuum	2,696	11	0,245		
	Ogółem	9,563	13			

W wyniku analizy wariancji (tab. 36) otrzymano potwierdzenie testu na dopasowanie modelu do zmienności zmiennej objaśnianej. W przypadku modelu nieliniowego wytypowane zmienne wyjaśniają zmienność logarytmu nakładów odpowiednio w 86% i 72%. Pozostała część wielkości jest zależna od innych zmiennych, które nie zostały wprowadzone do tego modelu. Dla modelu liniowego są to wartości 69% i 60%. Procentowe wskaźniki dopasowania zostały wyliczone na podstawie wartości zamieszczonych w kolumnie pierwszej tabeli 36: wyniki równań regresji i wartości ogółem. Wartości statystyki *F* dla obu modeli z prawdopodobieństwem bliskim zera (<0.0005) lub znacznie mniejszym od przyjętego poziomu istotności $\alpha = 0.05$ także przekracza wartość krytyczną, która wynosi 4,75. Pozwala to na odrzucenie hipotezy o jednoczesnym testowaniu każdego współczynnika modelu, a tak mały poziom istotności wskazuje, że każda ze znajdujących się w modelach zmiennych posiada pełny udział w wyjaśnianiu zmienności logarytmu *nakładów*.

6. 5. 2. Badanie wpływu atrybutów rozmiaru na wielkość nakładów

Otrzymane w wyniku zastosowania metody najmniejszych kwadratów poszukiwane współczynniki modelu zamieszczono w tabeli 33. W kolumnie *B* występują estymatory parametrów, w kolumnie *Beta* standaryzowane estymatory parametrów, które umożliwiają proste porównanie parametrów modelu, ale ich zastosowanie wymaga transformacji danych do standardu *z*. Statystyka *t*, obliczana jako iloraz współczynnika *B* i błędu standardowego pozwala na porównywanie siły udziału poszczególnych współczynników modelu w kształtowaniu wielkości zmiennej objaśnianej. Jednak jej wartość nie może być traktowana jako kryterium dołączania lub usuwania zmiennych z modelu. W przypadku modelu w wariancie *A* wartości statystyki *t* istotnie przekraczają krytyczną wartość statystyki, którą jest 2 lub -2. Na tej podstawie można wnioskować o podobnej sile wpływu zarówno *asocjacji* jak

i *diagramów przypadków użycia* na wielkość logarytmu *nakładów*. Oba czynniki wchodzą do modelu z dodatnimi wartościami parametrów, zatem powodują wzrost logarytmu *nakładów*. W przypadku modelu z rozwiązaniem *B* siła wpływu *asocjacji* jest istotnie większa niż *wskaźnika ponownego użycia*, dla którego statystyka *t* osiąga wartość mniejszą od krytycznej ($t = -1,507$), co jednak nie oznacza, że tą zmienną należy wyeliminować z modelu, ale może spowodować brak homoscedastyczności estymatora błędów. Dodatnia wartość parametru *asocjacji* oznacza rosnący wpływ tego czynnika na wielkość logarytmu *nakładów*, natomiast ujemna wartość parametru *wskaźnika ponownego użycia* wskazuje na jego malejący udział w kształtowaniu się wielkości logarytmu *nakładów*.

Z powodu otrzymania słabszych statystyk dla wariantu *B* oraz niekorzystnych testów dla *wskaźnika ponownego użycia* zdecydowano się na zbadanie modelu składającego się jednocześnie z trzech wytypowanych zmiennych dla obu wariantów. Zbudowano wariant *C* modelu logarytmu *nakładów* przez dołączenie do wariantu *A* *wskaźnika ponownego użycia* i wykonano dla tego wariantu estymację parametrów wraz z analizą regresji.

Tabela 37. Współczynniki modelu estymacji nieliniowej otrzymanych metodą najmniejszych kwadratów dla wariantu *C*. Źródło: [opracowanie własne].

Wariant modelu	Współczynniki niestandardyzowane		Współczynniki standaryzowane	Statystyka <i>t</i>	Poziom istotności
	B	Błąd standardowy	Beta		
<i>C</i> (Stała)	1,663	0,271		6,142	0,000
ASSO	1,1E-02	0,003	0,544	3,855	0,003
DUSE	8,7E-02	0,028	0,513	3,107	0,011
WWPU	-0,109	1,829	-0,009	-0,060	0,954

Model nakładów na podstawie wyników obliczeń procedury analizy regresji (tab. 37) i równania (6.23) dla wariantu *A* ma postać

$$V = 1,663 + 0,011 * asocjacje + 0,087 * diagramy\przypadkow\uzycia - 0,109 * wskaźnik\ponownego\uzycia. \tag{6.31}$$

Równanie tego modelu, po wykonaniu retransformacji do postaci pierwotnej wykładniczej zależności *nakładów* od ustalonych atrybutów rozmiaru modelu programu ma postać

$$N = 5,275e^{0,011*asocjacje+0,087*diagramy\przypadkow\uzycia-0,109*wskaźnik\ponownego\uzycia}, \tag{6.32}$$

gdzie *N* – *nakład*.

W modelu odpowiadającym wariantowi *C* wartość *R* wskazuje na wyższy stopień korelacji niż dla wariantu *B*, bo aż 0,92, a trzy dołączone zmienne wyjaśniają zmienność logarytmu *nakładów* w blisko 86%, podobnie, jak dla wariantu *A*. Wskaźnik dopasowania modelu do danych z obserwacji jest również bliski wskaźnikowi wariantu *A* (tab. 38). Wartość standardowego błędu estymacji modelu w tym wariancie jest również znacznie mniejsze niż wartość standardowego odchylenia logarytmu *nakładów*, która wynosi 0.8577 (tab. 34). Model w wariancie *C* osiąga podobne wartości ocen, jak model w wariancie *A*, oprócz oceny wpływu *wskaźnika ponownego użycia* na wielkość logarytmu nakładów, która w porównaniu do *asocjacji* i *diagramów przypadków użycia* jest mało istotna. Nie została jednak wyeliminowana z powodów wcześniej określonych.

Tabela 38. Wartości parametrów oceny modelu w wariancie *C*. Źródło: [opracowanie własne].

Wariant	Zmienne	<i>R</i>	<i>R</i> ²	$\overline{R^2}$	Standardowy błąd estymacji
<i>C</i>	DUSE ASSO WWPU	0,920	0,857	0,814	0,370375

6. 5. 3. Analiza składnika losowego modelu nakładów

Następnie przeprowadzono sprawdzenie założeń związanych z możliwością szacowania parametrów modelu za pomocą analizy regresji. W tabeli 39 zawarto statystyki dotyczące składnika losowego wariantów modelu danych równaniami (6.26), (6.28) i (6.31). Residua, czyli różnice między wartościami obserwowanymi zmiennej objaśnianej i wartościami przewidywanymi przez modele są to błędy, które odpowiadają pionowym liniom poprowadzonym od każdego punktu wartości obserwowanej do punktu na krzywej estymacji. Ponieważ estymacja jest prowadzona za pomocą funkcji liniowej, wobec tego jest to długość odcinka od punktu wartości obserwowanej do punktu na linii. Długości tych odcinków, nazywane residuami odpowiadają błędom wartości estymowanych za pomocą ustalonego modelu. Zatem residua są estymatorami błędów ϵ zbudowanego modelu. Ze względu na to, że modele (6.26), (6.28) i (6.31) są transformacją logarytmiczną modelu pierwotnego, to residua również przedstawiają wartość zlogarytmowaną $\xi = \ln \epsilon$.

Miarą wpływu estymatora błędów na wynik szacowanej wielkości za pomocą zbudowanego modelu jest ocena jego wariancji, rozkładu oraz korelacji. Zakłada się, że wartość średnia estymatora ma wartość równą 0.

Tabela 39. Statystyki estymatora błędów dla modelu odpowiadającego wariantom A, B i C. Źródło: [opracowanie własne].

Wariant modelu	Statystyka	Minimum	Maximum	Średnia	Standardowe odchylenie
A	Wartość przewidywana	1,835024	4,297170	2,58991	0,793766
	Stand. wartość przewidywana	-0,951	2,151	0,000	1,000
	Residuum	-0,704706	0,430218	3,0E-16	0,324898
	Stand. residuum	-1,995	1,218	0,000	0,920
	Student. odrzuc. residuum	-2,577	1,849	-0,032	1,165
B	Wartość przewidywana	1,418317	4,088188	2,58991	0,726803
	Stand. wartość przewidywana	-1,612	2,061	0,000	1,000
	Residuum	-0,810889	0,639200	4,9E-16	0,455390
	Stand. residuum	-1,638	1,291	0,000	0,920
	Student. odrzuc. residuum	-2,661	1,997	-0,003	1,218
C	Wartość przewidywana	1,818321	4,293383	2,58991	0,793789
	Stand. wartość przewidywana	-0,972	2,146	0,000	1,000
	Residuum	-0,707497	0,434005	5,2E-16	0,455390
	Stand. residuum	-1,910	1,172	0,000	0,877
	Student. odrzuc. residuum	-2,499	1,846	-0,053	1,161

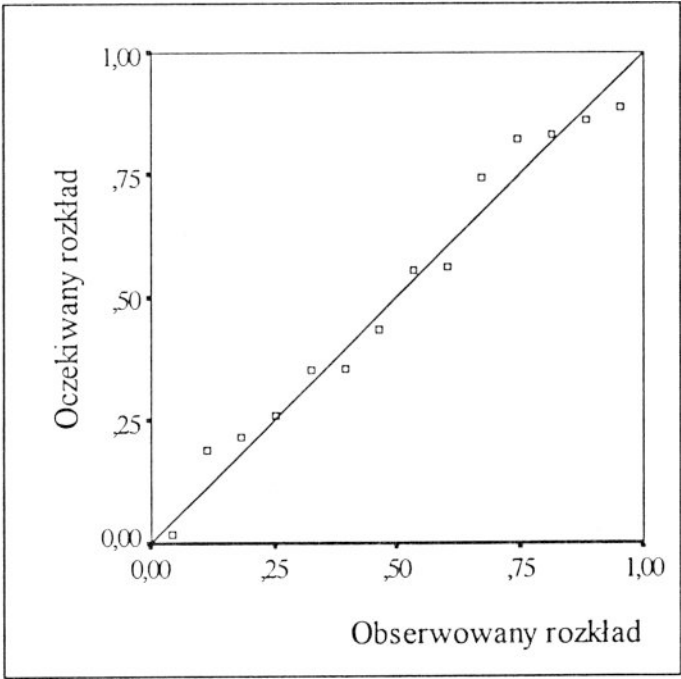
Weryfikacja hipotezy o normalności rozkładu składnika losowego

Standaryzowane wartości średnie błędów zbudowanych modeli mają wartość 0,000, a standardowe odchylenia wartość bliską 1, skąd można sądzić o ich rozkładzie normalnym, co zostało sprawdzone dla zbudowanych modeli w wyniku dalszej analizy. Wartości zamieszczone w tabeli 40, 41 i 42 są wynikiem obliczeń wielkości logarytmu *nakładów* za pomocą otrzymanych wariantów modelu dla każdej obserwacji danych przedstawionych w tabeli 21. Zawierają one wartości rzeczywiste i standaryzowane. Oprócz wielkości przewidywanych zawierają również wartości błędów. Spośród 14 obserwacji tylko jedna wartość residuum standaryzowanego równa -1,99519 (numer 6) dla przewidywanego logarytmu *nakładów* za pomocą modelu w wariancie A, przekracza ponad dwukrotnie wartość standaryzowaną odchylenia standardowego równą 0,920.

Tabela 40. Wartości obserwowane i przewidywane logarytmu nakładów na podstawie modelu w wariancie A oraz wartości residuów i standaryzowanych residuów. Źródło: [opracowanie własne].

Numer obserwacji	Logarytm wartości nakładów		Residuum		
	obserwowany	przewidywany	rzeczywiste	standaryzowane	student. odrzuc.
1	2,1972	1,96392	0,23330	0,66053	0,68674
2	2,1972	2,25395	-0,05673	-0,16061	-0,16326
3	1,6094	1,83502	-0,22559	-0,63869	-0,67383
4	2,0149	1,96392	0,05098	0,14433	0,14680
5	2,1163	2,24321	-0,12695	-0,35944	-0,36706
6	1,6094	2,31414	-0,70471	-1,99519	-2,57698
7	2,2618	1,92096	0,34081	0,96490	1,03580
8	3,8177	3,76224	0,05547	0,15704	0,17792
9	2,9444	3,21867	-0,27424	-0,77643	-0,84060
10	3,1570	3,28961	-0,13261	-0,37545	-0,46348
11	2,8184	3,12403	-0,30563	-0,86532	-1,44150
12	4,7274	4,29717	0,43022	1,21805	1,84891
13	2,3026	1,97467	0,32792	0,92842	0,98733
14	2,4849	2,09715	0,38776	1,09784	1,19157

Ocena poprawności modelu wymaga, by wykres estymatora błędów charakteryzował się rozkładem normalnym. Na rysunku 28 przedstawiono wykres zgodności rozkładu estymatora błędów z rozkładem normalnym dla modelu w wariancie A. Większość wartości jest skupiona wzdłuż linii prostej, co jest wystarczającym potwierdzeniem tej zgodności. Cztery obserwacje są nieco oddalone od linii prostej, a najbardziej obserwacja odpowiadająca numerowi 6. Taki sam wniosek wynika z analizy wartości residuów zawartych w tabeli 40.



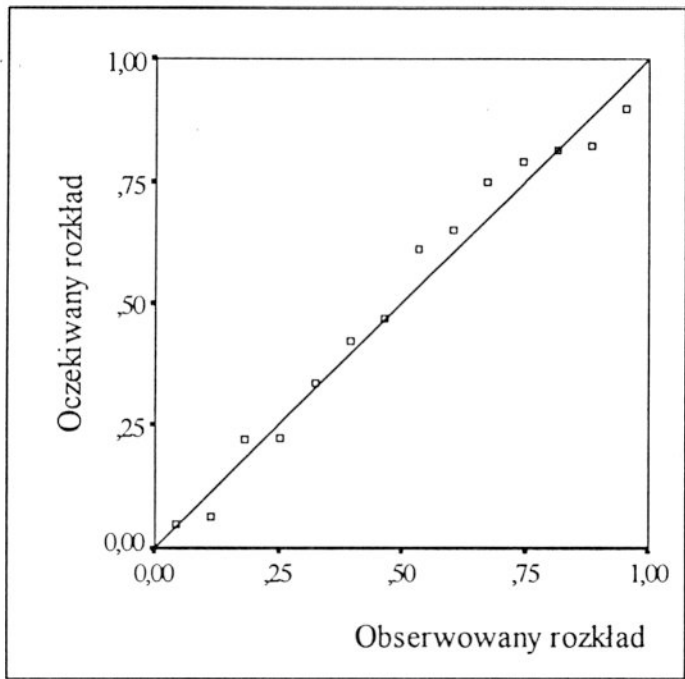
Rysunek 28. Wykres zgodności rozkładu standaryzowanego estymatora błędów z rozkładem normalnym modelu w wariancie A. Źródło: [opracowanie własne].

Wartości zamieszczone w tabeli 41 są wynikiem obliczeń wielkości logarytmu *nakładów* za pomocą modelu w wariancie *B*. Dla tego modelu spośród 14 obserwacji dwie wartości residuum standaryzowanego równe $-1,63796$ (numer 11) oraz $-1,48282$ (numer 6) przekraczają blisko dwukrotnie wartość standaryzowaną odchylenia standardowego równą $0,920$.

Tabela 41. Wartości obserwowane i przewidywane logarytmu nakładów na podstawie modelu w wariancie *B* oraz wartości residuów i standaryzowanych residuów. Źródło: [opracowanie własne].

Numer obserwacji	Logarytm wartości nakładów		Residuum		
	obserwowany	przewidywany	rzeczywiste	standaryzowane	student.odrzuc.
1	2,1972	2,23588	-0,03865	-0,07808	-0,08001
2	2,1972	2,56677	-0,36955	-0,74647	-0,75998
3	1,6094	1,41832	0,19112	0,38606	0,59162
4	2,0149	2,22289	-0,20798	-0,42011	-0,43306
5	2,1163	2,49461	-0,37836	-0,76426	-0,78018
6	1,6094	2,34352	-0,73409	-1,48282	-1,70023
7	2,2618	2,12099	0,14077	0,28435	0,29133
8	3,8177	3,40951	0,40820	0,82454	0,91339
9	2,9444	3,03869	-0,09425	-0,19038	-0,20046
10	3,1570	2,68782	0,46918	0,94772	1,01469
11	2,8184	3,62929	-0,81089	-1,63796	-2,66079
12	4,7274	4,08819	0,63920	1,29115	1,99658
13	2,3026	1,85448	0,44811	0,90515	1,04422
14	2,4849	2,14772	0,33719	0,68110	0,71579

Również wykres estymatora błędów (rys. 29) pokazuje, że większość wartości jest skupiona wzdłuż linii prostej, co jest wystarczającym potwierdzeniem zgodności rozkładu tego estymatora błędów z rozkładem normalnym. Dwie obserwacje (6 i 11) są najbardziej oddalone od linii prostej.



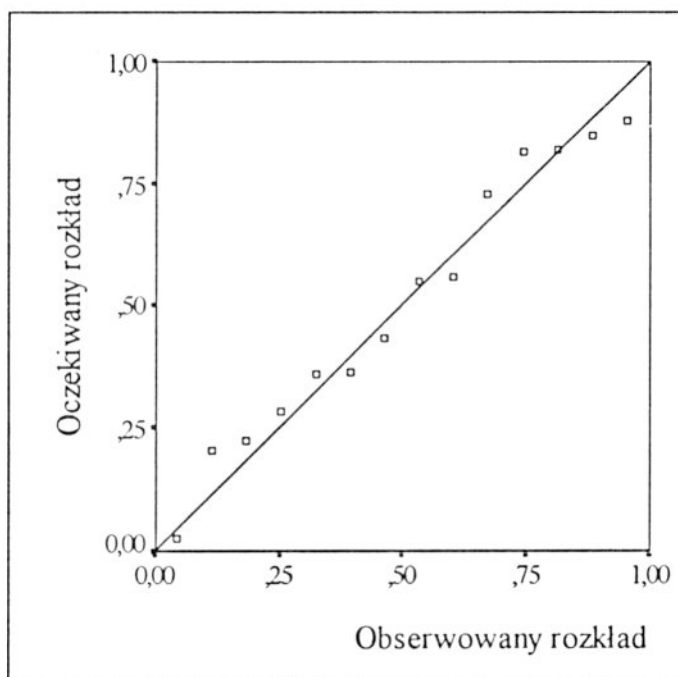
Rysunek 29. Wykres zgodności rozkładu standaryzowanego estymatora błędów z rozkładem normalnym modelu w wariancie B. Źródło: [opracowanie własne].

W tabeli 42 zawarto wartości przewidywane logarytmu *nakładów* oraz wartości błędów dla modelu w wariancie C. Są to wyniki bardzo zbliżone do zawartości tabeli 40 dla wariantu A, zatem ocena dotycząca rozkładu estymatora błędów z wariantu A przenosi się również na wariant C (rys. 30).

Tabela 42. Wartości przewidywane rzeczywiste i standaryzowane logarytmu nakładów na podstawie modelu w wariancie C oraz wartości residuów i standaryzowanych residuów. Źródło: [opracowanie własne].

Numer obserwacji	Logarytm wartości nakładów		Residuum		
	obserwowany	przewidywany	rzeczywiste	standaryzowane	student.odrzuc.
1	2,1972	1,96967	0,22755	0,61438	0,66210
2	2,1972	2,25755	-0,06032	-0,16286	-0,16731
3	1,6094	1,81832	-0,20888	-0,56398	-1,07435
4	2,0149	1,96922	0,04568	0,12334	0,12910
5	2,1163	2,24479	-0,12854	-0,34704	-0,35373
6	1,6094	2,31693	-0,70750	-1,91022	-2,49888
7	2,2618	1,92473	0,33704	0,90999	0,98928
8	3,8177	3,76332	0,05439	0,14685	0,16581
9	2,9444	3,22266	-0,27822	-0,75118	-0,82770
10	3,1570	3,28783	-0,13083	-0,35323	-0,43613
11	2,8184	3,12360	-0,30520	-0,82404	-1,36629
12	4,7274	4,29338	0,43400	1,17180	1,84606
13	2,3026	1,96659	0,33599	0,90717	1,05475
14	2,4849	2,10008	0,38483	1,03902	1,13394

Przeprowadzono weryfikację hipotezy H_0 o rozkładzie normalnym estymatora błędów za pomocą statystyki t . Wartość krytyczna na poziomie ufności $\alpha = 0,05$ i stopni swobody $v = n - k - 1 = 11$ statystyki $t = 2,201$ pozwala przyjąć hipotezę o rozkładzie normalnym estymatora błędów dla większości obserwacji zarówno w przypadku modelu A i C , jak też i modelu B . Tylko jedna wartość różnicy między przewidywanym za pomocą tych modeli logarytmem *nakładów* a pochodzącym z obserwacji przekracza wartość krytyczną, co świadczy o ich niezbyt dobrym dopasowaniu otrzymanych postaci modelu do tej obserwacji.



Rysunek 30. Wykres zgodności rozkładu standaryzowanego estymatora błędów z rozkładem normalnym modelu w wariancie C . Źródło: [opracowanie własne].

Weryfikacja hipotezy o równości wariancji estymatora błędów

W celu sprawdzenia wariancji składnika losowego modelu wykonano weryfikację następującej hipotezy:

H_0 : składnik losowy modelu jest heteroscedastyczny wobec hipotezy alternatywnej

H_1 : składnik losowy modelu jest homoscedastyczny.

Sprawdzenie tej hipotezy wykonano za pomocą testu parametrycznego Goldfelda i Quandta przeznaczonego do małych próbek. Przyjmuje tutaj się założenie, że wariancja reszt rośnie wraz ze wzrostem kwadratu jednej ze zmiennych objaśniających.

Sprawdzianem jest statystyka

$$R = \frac{S_2}{S_1}, \quad (6.33)$$

gdzie:

S_1 – suma kwadratów reszt funkcji regresji dla pierwszej grupy danych,

S_2 – suma kwadratów reszt funkcji regresji dla ostatniej grupy danych.

W przypadku prawdziwości hipotezy zerowej wartość ilorazu R rośnie dla dwóch skrajnych grup. W przypadku prawdziwości hipotezy alternatywnej iloraz R ma rozkład Fishera – Snedecora

$$F\left(\frac{n-c}{2}-k, \frac{n-c}{2}-k\right), \quad (6.34)$$

gdzie:

n – liczba obserwacji z próby,

k – liczba zmiennych objaśniających w modelu regresji,

c – liczba wyeliminowanych obserwacji.

Przeprowadzono dwie serie obliczeń. W pierwszej serii wyeliminowano $c = 2$ obserwacje środkowe ze zbioru danych z próby, który został uporządkowany według rosnących wartości *asocjacji*. W drugiej serii wyeliminowano $c = 4$ obserwacje środkowe. W pierwszej serii otrzymano dwa podzbiory obserwacji po sześć, a w drugiej po cztery obserwacje. Najpierw wykonano obliczenia dla wariantu A . Wartość statystyki z pierwszej serii wynosi $R = 2,2747$, a z drugiej $R = 2,1654$. Ze względu na to, że wartość statystyki R maleje hipoteza zerowa o heteroscedastyczności składnika losowego modelu w wariancie A zostaje odrzucona.

W celu przyjęcia hipotezy alternatywnej zbadano jeszcze, czy iloraz R ma rozkład Fishera – Snedecora. Statystyka F na poziomie istotności $\alpha = 0.05$ wynosi $F(4, 4) = 6.39$, a $F(3, 3) = 9.28$. Otrzymane wartości R są znacznie mniejsze od odpowiadających im wartości krytycznych, zatem należy przyjąć hipotezę o homoscedastyczności składnika losowego modelu w wariancie A . Następnie przeprowadzono sprawdzenie hipotezy dla wariantu B . Wartość statystyki z pierwszej serii wynosi $R = 2,5921$, a z drugiej $R = 19,1872$. Wartość statystyki R rośnie, zatem nie ma podstaw do odrzucenia hipotezy zerowej o heteroscedastyczności składnika losowego modelu w wariancie A .

Wobec stwierdzonego istnienia heteroscedastyczności estymatora błędów modelu w wariancie B , co już sugerowano przy analizie wpływu poszczególnych jego zmiennych na

dopasowanie modelu, przeprowadzono ponowną estymację parametrów tego modelu za pomocą ważonej metody najmniejszych kwadratów (tab. 43). Wówczas otrzymano model w postaci równania

$$V = 2,1105 + 0,016 * asocjacje - 3,7213 * wskaźnik ponownego użycia. \tag{6.35}$$

Po wykonaniu retransformacji do postaci nieliniowej równanie (6.35) przybiera postać

$$N = 8,25e^{0,016*asocjacje - 3,7213*wskaźnik ponownego użycia}, \tag{6.36}$$

gdzie N – nakład.

Tabela 43. Współczynniki modelu estymacji nieliniowej otrzymane ważoną metodą najmniejszych kwadratów dla modelu w wariancie B. Źródło: [opracowanie własne].

Wariant modelu	Współczynniki niestandardyzowane		Współczynniki standaryzowane	Statystyka t	Poziom istotności
	B	Błąd standardowy	Beta		
B (Stała)	2,1105	0,2887		7,310	0,0000
ASSO	1,6E-02	0,0033	0,774	5,000	0,0004
WWPU	-3,7213	2,2518	-0,256	-1,653	0,1270

Parametry modelu danego równaniem (6.35) w niewielkim stopniu różnią się od parametrów równania (6.28) otrzymanego za pomocą metody najmniejszych kwadratów, jednakże wzrost wartości statystyk t wskazuje na większy stopień wpływu obu zmiennych na wyjaśnianie logarytmu *nakładów*.

Kolejne obliczenia dotyczyły wariantu C. Wartość statystyki z pierwszej serii wynosi $R = 0,8181$, a z drugiej $R = 0,1350$. Zatem, dla dwóch kolejnych serii danych wartość statystyki R maleje, a hipoteza zerowa o heteroscedastyczności składnika losowego modelu w wariancie C zostaje odrzucona. W celu przyjęcia hipotezy alternatywnej zbadano, czy iloraz R ma rozkład Fishera – Snedecora. Otrzymane wartości R są znacznie mniejsze od odpowiadających im wartości krytycznych, zatem należy przyjąć hipotezę o homoscedastyczności składnika losowego modelu również w wariancie C.

Badanie autokorelacji estymatora błędów

Hipoteza zerowa dla współczynnika autokorelacji rzędu pierwszego mówi o braku autokorelacji

$$H_0 : \rho = 0, \quad (6.37)$$

a hipoteza alternatywna

$$H_1 : \rho \neq 0. \quad (6.38)$$

Dla sprawdzenia hipotezy zerowej przeprowadzono dwa testy: za pomocą testu Studenta oraz Durбина - Watsona. Dla testu Studenta sprawdzianem jest statystyka

$$t = \frac{|r|\sqrt{n-3}}{\sqrt{1-r^2}}, \quad (6.39)$$

gdzie:

n – liczba obserwacji z próby,

$$r = \frac{\frac{1}{n-1} \sum_2^n e_i e_{i-1}}{\frac{1}{n-k-1} \sum_1^n e_i^2}, \quad (6.40)$$

k – liczba zmiennych.

Wartość krytyczna dla $n - 3 = 11$ stopni swobody i poziomu istotności $\alpha = 0,05$ wynosi $t^* = 2,2010$. Obliczone wartości statystyki t (tab. 44) dla modeli we wszystkich trzech wariantach są mniejsze od wartości krytycznej t^* . Zatem nie ma podstaw do odrzucenia hipotezy o braku autokorelacji estymatora błędów zarówno dla modelu A , B , jak i C , co również potwierdzają bardzo małe wartości współczynnika r , obliczonego według wzoru (6.40). Dla testu Durбина – Watsona sprawdzeniem hipotezy zerowej jest statystyka

$$d = \frac{\sum_2^n (e_i - e_{i-1})^2}{\sum_1^n e_i^2}, \quad (6.41)$$

gdzie e_i – reszta wyznaczona za pomocą funkcji regresji.

Tabela 44. Wartości statystyki t oraz d wariantu A , B i C . Źródło: [opracowanie własne].

Wariant modelu	Statystyka t	R	Statystyka d
A	0,0015	0,0005	1,7805
B	0,0206	-0,0062	2,2017
C	0,0018	0,0005	1,7695

Wartości krytyczne, odczytane z tablic [DOM90], dla ustalonego poziomu istotności $\alpha = 0.05$, liczby obserwacji $n = 15$ i liczby zmiennych objaśniających $k = 2$ wynoszą $d_L = 1,077$, $d_U = 1,361$, a dla $k = 3$ odpowiednio $d_L = 0,946$, $d_U = 1,543$. Ponieważ wielkości d_L i d_U dla danych α i k oraz zmniejszającej się wartości n o wielkość 1 zmieniają się nie więcej niż o $\pm 0,01$, to obliczone wartości statystyk dla liczby obserwacji $n = 14$ porównano ze statystykami dla $n = 15$ z uwzględnieniem tej poprawki.

Wartości statystyki d obliczone ze wzoru (6.41) dla testu Durbina – Watsona w przypadku modeli w wariancie A i C (tab. 44) są większe od wartości krytycznej d_U . Nie ma zatem podstaw do odrzucenia hipotezy o braku autokorelacji składnika losowego tego modelu.

W przypadku modelu w wariancie B , którego współczynnik $r < 0$ zastosowano statystykę $d' = 4 - d$. Obliczona wartość wynosi $d' = 1,7983$, również przekracza wartość krytyczną d_U .

Potwierdzony brak autokorelacji estymatora błędów dla wariantów A i C modelu logarytmu *nakładów* wskazuje na istotne wyjaśnienie kształtowania się logarytmu *nakładów* przez wprowadzone zmienne objaśniające.

W wyniku przeprowadzonej weryfikacji trzech założeń, które warunkują zastosowanie metody analizy regresji, otrzymano potwierdzenie słuszności jej przyjęcia do budowy modelu estymacji logarytmu *nakładów*.

7. ZAKOŃCZENIE

Celem niniejszej pracy było ustalenie modelu nakładów pracy związanych z tworzeniem programu przy podejściu obiektowym. Zadanie to zostało określone jako poszukiwanie postaci funkcji tych nakładów, które stanowią jeden z podstawowych elementów składających się na koszt realizacji takiego przedsięwzięcia. Wykonanie tak sformułowanego zadania zostało poprzedzone niezbędnymi rozważaniami. W ich wyniku określono wymagania, które zadecydowały o wyborze postępowania prowadzącego do osiągnięcia zamierzonego celu.

Omówiono zagadnienia dotyczące sposobu mierzenia rozmiaru programu oraz czynniki występujące w modelach nakładów pracy potrzebnych do zaplanowania i kontroli realizacji wykonania programu. Ponieważ badania dotyczyły tworzenia programu przy podejściu obiektowym, dlatego scharakteryzowano także podstawowe elementy technologii obiektowej.

W rozdziale wstępnym pracy przedstawiono przyczyny niepowodzeń przedsięwzięć informatycznych. Pozwoliły one sformułować najważniejsze założenie na potrzeby estymacji nakładów - możliwość jej wykonania już na podstawie wyników analizy wymagań użytkownika wobec przyszłego systemu informatycznego. Wymagania te zaś są przedstawiane w postaci sformalizowanego modelu. Wtedy to wielkość planowanych zasobów dla takiego przedsięwzięcia będzie mogła być wynikiem szacunków bazujących na wymiernych danych liczbowych.

Przegląd metod stosowanych do szacowania wielkości nakładów oraz dokonana próba adaptacji metody *FP* do warunków technologii obiektowej pozwoliły na stwierdzenie braku możliwości bezpośredniego odzwierciedlenia w nich cech obiektowych programu. Cechy te w istotny sposób wpływają na rozmiar programu i nie mogą być pomijane bez wpływu na końcową ocenę szacowanej wielkości nakładów. Istotny udział cech technologii obiektowej w sposobie powstawania programu jest szczególnie widoczny w jego wielkości, a to stwarza potrzebę ich uwzględniania również w konstrukcji modelu nakładów.

Zadanie konstrukcji modelu nakładów zostało zrealizowane na bazie założenia o ekonomicznej naturze procesu tworzenia programu. Na taki proces wpływa wiele różnych czynników, między którymi mogą zachodzić wzajemne powiązania. Do podstawowych czynników zaliczono rozmiar programu, ponieważ zmiany tego czynnika znajdują w wielkości nakładów swoje bezpośrednie odzwierciedlenie.

Zbadano, od których atrybutów rozmiaru tworzonego programu zależy poziom zużycia nakładów. W analizie tej za podstawowe narzędzie przyjęto modele budowane przy wykorzystaniu metod ekonometrycznych. Zasadniczymi składnikami modelu ekonometrycznego są dwie grupy zmiennych oraz parametry strukturalne. Pierwsza grupa zmiennych to zmienne objaśniane, a druga to zmienne objaśniające. Nakład pracy należy do pierwszej grupy, a atrybuty rozmiaru do drugiej. Przez zastosowanie odpowiednich metod oczekiwano wyeliminowania z zestawu potencjalnych zmiennych tych atrybutów, których zmiana jest nieistotna z punktu widzenia zmienności nakładów. Zagadnienie doboru odpowiednich zmiennych do modelu stanowi bowiem jeden z podstawowych problemów w poszukiwaniu postaci modelu opisującego zjawiska natury ekonomicznej. Do takiego rodzaju zjawisk przypisano również proces tworzenia programu informatycznego.

Jednym z kluczowych zadań w prowadzonych badaniach było ustalenie, jakie czynniki mają istotny wpływ na wielkość nakładów tworzenia. Jak opisano w rozdziale trzecim, czynniki te mogą pochodzić z trzech następujących grup:

- rozmiar programu,
- techniczna złożoność programu,
- środowisko tworzenia programu.

Przeprowadzono rozważania, w wyniku których zdecydowano, że badanie zależności nakładów pracy zostanie zrealizowane w odniesieniu do podstawowej grupy czynników, jaką stanowi rozmiar programu. Taka droga postępowania jest konsekwencją wcześniej sformułowanych wniosków:

- Czynniki należące do grupy technicznej złożoności nie są jasno zdefiniowane i w opisywanych metodach są różnie interpretowane, a zatem są obarczone subiektywnym spojrzeniem na ocenę tych cech programu. Często też pokrywają się z niektórymi aspektami rozmiaru programu.
- Warunki tworzenia programu są określane przede wszystkim za pomocą metodyki tworzenia, stosowanych narzędzi oraz kwalifikacji zespołu. Natomiast w ramach

określonej metodyki i narzędzi również atrybuty rozmiaru programu czy też jego modelu są odzwierciedleniem zarówno kwalifikacji jak i doświadczenia. W mniejszym stopniu różnice w wielkości ponoszonych nakładów pracy wynikają z socjologicznych i psychologicznych aspektów pracy zespołu. Jednak miary atrybutów takiej natury wymagają oddzielnego procesu badania sposobów ich mierzenia, co wiąże się z zagadnieniami wykraczającymi poza zakres niniejszej pracy.

Wobec powyższych argumentów zadanie poszukiwania zależności nakładów polegało na badaniu związku nakładów tylko z jedną grupą czynników, a mianowicie z rozmiarem programu. Ze względu na trzy aspekty tego rozmiaru, jak długość, funkcjonalność i złożoność oraz strukturę modelu programu (składającą się z wielu elementów) okazało się, że istnieje wiele atrybutów rozmiaru, które należałoby uwzględnić. Wytypowano zatem trzynaście atrybutów rozmiaru oraz dodatkowo - jeden atrybut procesu. Stanowiły one zestaw potencjalnych zmiennych objaśniających do modelu nakładów. W wyniku wstępnej analizy kolekcji danych wystąpiła konieczność wyeliminowania czterech atrybutów z powodu nie wykorzystywania tych elementów w realizacji badanych projektów. Ze względu na nowatorską, jaką jest jeszcze w naszym kraju obiektowa technologia tworzenia programów, udało się skompletować niezbyt liczną kolekcję projektów. Z tego też powodu zaakceptowano wariantowe rozwiązanie, którego modele mogą zostać zweryfikowane w ciągu kontynuowanych w przyszłości badań.

Dane, które były podstawą do wyprowadzenia postaci analitycznej modelu, zostały zebrane w kilku firmach wytwarzających oprogramowanie w technologii obiektowej i przy zastosowaniu w procesie tworzenia narzędzi CASE. Do utworzenia kolekcji danych wykorzystano program „Analizator repozytorium”, opracowany za pomocą Delphi, narzędzia do szybkiego tworzenia aplikacji. Program ten umożliwił szybkie i obiektywne skompletowanie danych dla wytypowanych zmiennych na podstawie zawartości repozytorium, w którym pamiętane są modele wykonanych programów. Taka automatyczna analiza zawartości repozytorium całkowicie eliminuje element subiektywnej interpretacji oceny rozmiaru programu i pozwala na porównywanie różnych projektów.

Dla zebranej kolekcji danych, odpowiadających wytypowanym do modelu nakładów zmiennym, przeprowadzono odpowiednio dobrane procedury statystyczne. Na podstawie analizy graficznie przedstawionych zależności (rysunki od 18 do 25) między wielkością nakładów i wielkością poszczególnych zmiennych zdecydowano o nieliniowej postaci modelu

nakładów. Po zlogarytmowaniu obu stron równania (6.24) otrzymano nieliniowy charakter związków między nakładem i atrybutami rozmiaru z jednocześnie liniowym związkiem względem parametrów strukturalnych tego modelu. Takie przekształcenie umożliwiło zastosowanie metod statystycznych odpowiednich dla związków liniowych, zarówno do wyboru optymalnych zmiennych objaśniających, jak i do estymacji tych parametrów.

Analizując wyznacznik macierzy współczynników korelacji liniowej, stwierdzono istnienie współliniowości między zmiennymi objaśniającymi. Następnie ustalono, które zmienne z listy potencjalnych zmiennych objaśniających należy wyeliminować, by uniknąć wykazanego powielania informacji. Na podstawie analizy współczynników korelacji liniowej między logarytmem *nakładów* i wytypowanymi atrybutami rozmiaru przeprowadzono weryfikację hipotezy H_0 o braku skorelowania między wielkościami logarytmicznymi *nakładów* a atrybutami rozmiaru na ustalonym poziomie $\alpha = 0,05$. Wartości poziomów istotności dla pięciu następujących atrybutów: *asocjacje*, *atrybuty*, *diagramy obiektów*, *diagramy przypadków użycia* i *obiekty* pozwoliły na odrzucenie hipotezy H_0 o braku skorelowania. Na tej postawie potwierdzono wniosek o istotnym wpływie tych atrybutów na wielkość logarytmu *nakładów*. Idąc dalej, pokazano istotne skorelowanie, jakie występuje wśród przykładowej grupy atrybutów: *diagramy obiektów*, *asocjacje* i *obiekty*, ponieważ charakteryzują one rozmiar od strony dwóch aspektów: długości i złożoności strukturalnej, wobec których przypuszczano istnienie liniowej zależności. Do modelu *nakładów* wprowadzono zatem tylko reprezentanta tej grupy. Alternatywny zestaw zmiennych objaśniających, składający się z *asocjacji* i *wskaźnika ponownego użycia* dobrano na podstawie analizy macierzy odległości za pomocą metody taksonomii wrocławskiej.

Model *nakładów* wyprowadzono w trzech wariantach *A*, *B* i *C*. Dwa warianty *A* i *B* powstały w wyniku działania procedur doboru zmiennych objaśniających, a trzeci *C* na podstawie ich analizy. W wariacie *A* uwzględniono *asocjacje* i *diagramy przypadków użycia*, w *B* *asocjacje* i *wskaźnik ponownego użycia*, a w wariacie *C* *asocjacje*, *diagramy przypadków użycia* i *wskaźnik ponownego użycia*.

W wyniku weryfikacji tych trzech wariantów modelu *nakładów* dla wariantu *B* w porównaniu do *A* i *C* stwierdzono:

- mniejszy wskaźnik korelacji między obserwowaną i przewidywaną wartością logarytmu *nakładów*,
- mniejszy wskaźnik procentowy dla wyjaśniania zmienności logarytmu *nakładów* przez ustalone zmienne objaśniające,

- słabsze dopasowanie do danych empirycznych,
- większy standardowy błąd estymacji,
- brak homoscedastyczności składnika losowego.

Z wymienionych powodów za ostateczne rozwiązanie przyjęto wariant *A* i *C*, które są opisane w postaci liniowej równaniami (6.26) i (6.31) oraz nieliniowej w postaci wykładniczej (6.27) i (6.32). Po przeprowadzeniu testów statystycznych najlepszą ocenę otrzymano dla wariantu *A* ze związkami *asocjacji* oraz *diagramami przypadków użycia*. Jednakże, ze względu na przewidywany istotny wpływ na wielkość *nakładów* również *wskaźnika ponownego użycia*, wariant *C* ze wszystkimi trzema zmiennymi przyjmuje się za ważny dla wyjaśnienia zależności *nakładów* od atrybutów rozmiaru modelu programu. Zmienna objaśniająca *wskaźnik ponownego użycia* charakteryzuje aspekt podstawowy dla technologii obiektowej, a którego brak zauważa się w wariancie *A*. Wzrost roli tego atrybutu w kształtowaniu wielkości *nakładów* oczekuje się dla projektów, w których paradygmat dziedziczenia będzie z większą siłą wykorzystywany przez projektantów. W tabeli 45 zawarto porównanie otrzymanych wielkości standardowych błędów estymacji dla trzech opisywanych wariantów modelu. Najmniejszą wartością tego błędu charakteryzuje się model logarytmu *nakładów* w wariancie *A* i *C*.

Tabela 45. Porównanie wartości standardowych błędów estymacji.
Źródło: [opracowanie własne].

Wariant modelu	Zmienne	Standardowy błąd estymacji
A	ASSO DUSE	0,35320
B	ASSO WWPU	0,49506
C	ASSO DUSE WWPU	0,37038

Założenia przyjęte do konstrukcji modelu *nakładów* umożliwiły otrzymanie równania estymacji opartego na atrybutach modelu programu, które są znane na wczesnym etapie tworzenia – na etapie analizy wymagań użytkownika. Spełnia on oczekiwania menedżerów projektów informatycznych, by wykonywane szacowania w odniesieniu do *nakładów* pracy mogły być prowadzone bez subiektywnych ocen i na bazie posiadanej wiedzy o dziedzinie tworzonego programu. Model etapu analizy jest z jednej strony odzwierciedleniem wymagań użytkownika wobec przyszłego systemu, a z drugiej bazą dla powstania produktu końcowego. Przy konstrukcji modelu *nakładów* kierowano się tym, że model analizy dostarcza rzetelnych informacji do obliczeń szacunkowych o procesie tworzenia tego systemu.

Warunkiem obiektywnego, szybkiego i rzetelnego sposobu zebrania danych, odpowiadających wytypowanym miarom była automatyczna analiza dokumentacji modelu programu i automatyczny sposób kolekcjonowania tych danych. Spełnienie tego warunku osiągnięto za pomocą repozytorium narzędzi CASE, korzystanie z którego wpłynęło na uzależnienie wielkości *nakładów* tworzenia od liczby elementów zawartych w repozytorium systemu CASE. Praca z narzędziami mocno dyscyplinuje pracę zespołu i w wysokim stopniu formalizuje proces tworzenia. Taka formalizacja procesu zapewniła otrzymanie zestawu modeli analizy wykonanych według obowiązujących standardów. Jednakże nieduże doświadczenie zespołów w bogatym stosowaniu technik obiektowych wyeliminowało z badań niektóre charakterystyczne dla nich atrybuty. Badania prowadzone w odniesieniu do programów tworzonych za pomocą narzędzi CASE zapewniło automatyczną identyfikację wytypowanych charakterystyk tworzonego produktu i automatyczne zliczanie mierzonych atrybutów. Dzięki standaryzacji prowadzonych prac, przez zastosowanie narzędzi CASE, otrzymany model nakładów jest możliwie niezależny od indywidualnych modyfikacji ustalonych w metodyce procedur działań.

Wytypowane atrybuty rozmiaru programu dla modelu analizy występują także na dalszych etapach tworzenia (projektowania, kodowania, testowania i pielęgnacji) w postaci modeli kolejno modyfikowanych, które pozwalają na wykonywanie między nimi istotnych analiz porównawczych. Oszacowane wielkości nakładów na podstawie danych z analizy mogą być poprawiane za pomocą ponownego wykonania obliczeń dla danych z etapu np. projektowania. Umożliwia to dokonywanie na bieżąco zarówno kontroli zaplanowanych terminów wykonania poszczególnych zadań projektowych, jak i ich ewentualnej korekty. Zastosowanie tego samego modelu zależności nakładów od jego atrybutów na kolejnych etapach powinno być jednak poprzedzone dalszymi badaniami w celu weryfikacji takiej możliwości. Za istotne uważa się zatem prowadzenie dalszych badań w kierunku budowy modelu ogólnego dla tych etapów cyklu życia systemu informatycznego, dla których podstawą może być skonstruowany model nakładów w wyniku zrealizowanej pracy.

8. BIBLIOGRAFIA

- [AHO74] Aho A.V., Hopcroft J. E., Ullman J. D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974
- [ALB79] Albrecht A.J., Measuring application development productivity, [w:] Proc. IBM Applications Development Symp., GUIDE Int. and SHARE Inc., IBM Corp., Monterey, CA, October 14-17, 1979, s. 83
- [ALB83] Albrecht A.J., Gaffney J.E., Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, November 1983
- [ALL91] McAllister D.F., Scott R.K., Cost modelling of fault-tolerant software, Information and Software Technology, October 1991
- [BAL81] Balzer R., Transformational Implementation: An Example, IEEE Transactions on Software Engineering, January 1981
- [BAN87] Banerjee i inni, Data model issues for object oriented applications, ACM Trans. Office Information Systems, vol. 5, 1987
- [BAR90] Barker R., CASE*Method: Tasks and Deliverables, Addison-Wesley Publishing Company, 1990
- [BAR92] Barker R., Longman C., CASE*Method: Function ana Process Modelling, Addison-Wesley Publishing Company, 1990
- [BART74] Bartosiewicz S., Metody ekonometryczne, PWN, Warszawa, 1974
- [BART90] Praca zbiorowa pod redakcją S. Bartosiewicz, Estymacja modeli ekonometrycznych, PWE, Warszawa, 1990
- [BAS88] Basili V.R., Rombach H.D., The TAME project: towards improvement – oriented software environments, IEEE Transactions on Software Engineering, June 1988
- [BEG96] Begg D., Fischer S., Dornbush, R., Ekonomia. Mikroekonomia, PWE, Warszawa, 1996
- [BEH83] Behrens C.A., Measuring the Productivity of Computer Systems Development Activities with Function Points, IEEE Transactions on Software Engineering, November 1983
- [BER92] Berson A., Client/Server Architecture, McGraw Hill, New York, 1992
- [BOE81] Boehm B.,W., Software Engineering Economics, Prentice Hall, Englewood Cliffs, New Jersey, 1981
- [BOE84] Boehm B.,W., Software Engineering Economics, IEEE Transactions on Software Engineering, January 1984

- [BOE88] Boehm B., W., Papaccio P.N., Understanding and Controlling Software Costs, IEEE Transactions on Software Engineering, vol.14, no 10, October 1988
- [BOL90] Bollinger T., Pfleeger S.L., The Economics of Software Reuse: Issues and Alternatives, Information and Software Technology, November 1990
- [BOO91] Booch G., Object-Oriented Analysis and Design with Application, Addison-Wesley Publishing Company, 1994
- [BUN77] Bunge M., Treatise in Basic Philosophy: Ontology I: The Furniture of the World, Boston Riedel, 1977
- [BUN79] Bunge M., Treatise in Basic Philosophy: Ontology II: The World of Systems, Boston Riedel, 1979
- [CAB76] Cabe Mc T., A software complexity measure, IEEE Transaction on Software Engineering, 2(4), 1976
- [CAR90] Card D.N., Glass R. L., Measuring Software Design Quality, Prentice Hall, Englewood Cliffs, NJ, 1990
- [CHE76] Chen P.P., The Entity-Relationship Model: Toward a Unified View of Data, ACM Transaction on Database Systems", 1976
- [CIE78] Cieślak M., Dobór cech prognostycznych metodą heurystyczną, Prace naukowe Akademii Ekonomicznej we Wrocławiu, 1978
- [COA94a] Coad P., Yourdon E., Analiza obiektowa, Oficyna wydawnicza Read Me, Warszawa 1994
- [COA94b] Coad P., Yourdon E., Projektowanie obiektowe, Oficyna wydawnicza Read Me, Warszawa 1994
- [CHI91] Chidamber S.R., Kemerer C.F., Towards a Metrics Suite for Object Oriented Design, Proceedings of OOPSLA'91, Sigplan Notices, 1991
- [CHI94] Chidamber S.R., Kemerer C.F., A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, June 1994
- [CZA94] Czarnacka-Chrobot B., Metoda Function Points, Informatyka, 5, 1994
- [CHE91] Cherniavsky J.C., Smith C.H., On Weyuker's Axioms for Software Complexity Measures, IEEE Transactions on Software Engineering, June 1991
- [DAT81] Date C., An Introduction to Database Systems, Addison-Wesley, Reading, MA, 1981
- [DEM82] Demshki D., Ligett B., Linn G., McCluskey, Miller R., Wang Institute cost model (WICOMO) tool user's manual, Wang Inst. Graduate Studies, Tyngsboro, Ma, June 1982
- [DOM90] Domański Cz., Testy statystyczne, PWE, Warszawa, 1990

- [FAR67] Farrar D. E., Glauber R. R., Multicollinearity in Regression Analysis: The problem Revisited, *The Review of Economics and Statistics*, 1967, vol. 49
- [FEG94] Feghali I., Watson A., H., Clarification concerning modularization and McCabe's cyclomatic complexity, *Comm. ACM*, 1994
- [FEN91] Fenton N., E., *Software Metrics - A rigorous approach*, Chapman & Hall, London 1991
- [FEN94] Fenton N., E., *Software Measurement: A Necessary Scientific Basis*, *IEEE Transactions on Software Engineering*, 3, 1994
- [FEN96] Fenton N., E., Pfleeger S. L., *Software Metrics*, PWS Publishing Company, 1996
- [FLO51] Florek K., Łukaszewicz J., Perkal J., Steinhaus H., Zubrzycki S., *Taksonomia wroclawska, Przegląd Antropologiczny*, 1951, t. 17
- [FRO95] Frost S., *The SELECT Perspective: Developing Enterprise Systems Using Object Technology*, SELECT Software Tools, Inc., 1995
- [GAL93] Galant-Pater M., Hołodnik-Janczura G., Piekarski L., Płonka-Szydłak E., Zabawska E., *Analiza możliwości wykorzystania narzędzi CASE w modelowaniu systemów informacyjnych zarządzania, Raport Inst. Org. i Zarządz. Politechniki Wrocławskiej, SPR nr 15, Wrocław 1993*
- [GAR79] Garey M., R., Johnson D., S., *Computers and Intractability*, W. H. Freeman, San Francisco, 1979
- [GAR96] Garmus D., Herron D., *Measuring the Software Process, A Practical Guide to Functional Measurements*, Yourdon Press, Prentice Hall, 1996
- [GOL75] Goldberger S., *Teoria ekonometrii*, PWE, Warszawa, 1975
- [GRA82] Grabiński T., Wydymus S., Zeliaś A., *Metody doboru zmiennych w modelach ekonometrycznych*, PWN, Warszawa, 1982
- [GRA89] Grabiński T., Wydymus S., Zeliaś A., *Metody taksonomii numerycznej w modelowaniu zjawisk społeczno - gospodarczych*, PWN, Warszawa, 1989
- [GOR90] Gorla N., Benander A., C., Benander B., A., *Debugging Effort Estimation Using Software Metrics*, *IEEE Transactions on Software Engineering*, February 1990
- [HAL77] Halstead M., *Elements of Software Science*, Elsevier, New York, 1977
- [HAR92] Harel D., *Rzecz o istocie informatyki - Algorytmika*, Wydawnictwa Naukowo – Techniczne, Warszawa, 1992
- [HEN93] Henderson-Sellers B., *The economics of reusing library classes*, *Journal of the Object – Oriented Programming*, 6(4), s. 43-50, 1993
- [HEN96] Henderson-Sellers B., *Object-Oriented Metrics*, Prentice Hall PTR, New Jersey, 1996

- [HOL94] Hołodnik-Janczura G., Płonka-Szydłak E., Zabawska E., Realizacja przedsięwzięć informatycznych z wykorzystaniem metodyki CASE firmy Oracle, *Informatyka*, 2, 1994
- [HOL95] Hołodnik-Janczura G., Płonka-Szydłak E., Zabawska E., CASE firmy Oracle do analizy organizacji oraz definiowania strategii informatyzacji, *Informatyka*, 2, 1995
- [HOL96] Hołodnik-Janczura G., Jak płacić programistom, *Informatyka*, 9, 1996
- [HOL97] Hołodnik-Janczura G., Miary produktu informatycznego tworzonego w technologii obiektowej, Raport Inst. Org. i Zarządz. Politechniki Wrocławskiej, SPR nr 9, Wrocław, 1997
- [HOL98] Hołodnik-Janczura, G., Zasady tworzenia systemów informatycznych, *Informatyka*, 3, 1998
- [HOO89a] HOOD User Manual, Issue 3.0., WME/89-352/JB. HOOD Working Group, European Space Agency, December, 1989
- [HOO89b] HOOD Reference Manual, Issue 3.0., WME/89-173/JB. HOOD Working Group, European Space Agency, September, 1989
- [HOP94] Hopcroft J.E., Ullman J.D., Wprowadzenie do teorii automatów, języków i obliczeń, PWN, Warszawa, 1994
- [HUT94] HUTT A., Object Analysis and Design. Comparison of Methods, Wiley&Sons, Inc., 1994
- [IIV90] Iivari J., Hierarchical spiral model for information systems and software development. Part 2: design process, *Inf. Software Technology*, 32(7), 1990
- [JAC92] Jacobson I., Christerson M., Jonsson P., Overgaard G., Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Reading, MA, 1992
- [JAG94] Jagas, J., Produktywność pracy, Uniwersytet Opolski, Opole, 1994
- [JON88] Jones A., A short history of function and feature points, Int. Function Point Users Group Conf., 1988
- [KAN95] Kan S., H., Metrics and Models in Software Quality Engineering, Addison-Wesley Publishing Company, Massachusetts 1995
- [KLI95] Klimczak B., Mikroekonomia, Akademia Ekonomiczna, Wrocław, 1995
- [LIH93] Li H., Henry S., Object – oriented metrics that predict maintainability, *J. Sys. Software*, 23, 1993
- [LOR93] Lorenz M., Object-Oriented Software Development: A Practical Guide, Prentice Hall, NJ, 1993
- [MAR82] DeMarco T., Controlling Software Projects, Yourdon Press, New York, 1982

- [MAT94] Matson J.E., Bruce E., Barrett J., Mellichamp, M., Software Development Cost Estimation Using Function Points, IEEE Transactions on Software Engineering, April 1994
- [MEY88] Meyer B., Object-Oriented Software Construction, Prentice Hall, Hemel Hempstead, UK, s. 534, 1988
- [MEY92] Meyer B., Software engineering: reflections on a scientific discipline and its educational requirements, seminar presented at University of Technology, October 12, Sydney, Australia, 1992
- [MIK86] Mikołajczak B., Stokłosa J., Złożoność obliczeniowa algorytmów, Wydawnictwo Politechniki Poznańskiej, Poznań, 1986
- [MUK92] Mukhopadhyay T., Kekre S., Software Effort Models for Early Estimation of Process Control Applications, IEEE Transaction on Software Engineering, October 1992
- [NOW84] Nowak E., Problemy doboru zmiennych do modelu ekonometrycznego, PWN, Warszawa, 1984
- [NOW94] Nowak E., Analiza kosztów, Akademia Ekonomiczna, Wrocław, 1994
- [NOW95] Nowak E., Metody badania kosztów, Fundacja rozwoju rachunkowości w Polsce, Warszawa, 1995
- [NOW96] Nowak E., Teoria kosztów w zarządzaniu przedsiębiorstwem, PWN, Warszawa, 1996
- [OMG89] Object Management Architecture Guide, OMG, 1989
- [OMG91] Common Object Request Broker, OMG, 1991
- [PAR80] Parr F.,N., An Alternative to the Rayleigh Curve Model for Software Development Effort, IEEE Transactions on Software Engineering, vol. Se-6, No 3, 1980
- [PAW76] Pawłowski Z., Statystyka matematyczna, PWN, Warszawa, 1976
- [PFL91] Pfleeger L., Software Engineering: The Production of Quality Software, McMillan Publishing Company, NY, Toronto 1991
- [PLU69] Pluta W., Grafowa metoda klasyfikacji cech, Prace Naukowe WSE, Wrocław, 1969
- [PUT78] Putnam L.H., A General Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transaction on Software Engineering, July 1978
- [PUT79] Putnam, L.,H., A. Fitzsimmons: Estimating software costs, Datamation, No. 9,10,11, 1979
- [REK96] Rekowski M., Wprowadzenie do mikroekonomii, Poznań, 1996
- [ROB79] Roberts F., Encyclopedia of Mathematics and its Applications, Reading, MA, Addison – Wesley, 1979

- [RUB83] Rubin A., Macro-estimation of software development parameters: The ESTIMACS system, SOFTFAIR Conf. Software Dev. Tools, Techniques and Alternatives, 1983, s. 109-118
- [RUB92] Rubin K., Goldberg, K., Object Behavior Analysis., Communications of ACM, vol. 35(9), 1992
- [RUM91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., Object-Oriented Modeling and Design, Englewood Cliffs, NJ: Prentice-Hall, 1991
- [RUM92] Rumbaugh J., An object or not an object?, Journal Object-Oriented Programming, 5(3), 1992
- [SAN95] Sanders J., Curran E., Software Quality, Addison-Wesley Publishing Company, 1995
- [SCH92] Schneidewind N.F., Methodology for Validating Software Metrics, IEEE Transaction on Software Engineering., May 1992
- [SEL96] SELECT Enterprise, SELECT Software Tools, 1996
- [SHE89] Shepperd M., Ince D., Metrics, outlier analysis and the software design process, Information and Software Technology, March 1989
- [SHE95] Shepperd M., Foundation of Software Measurement, Prentice Hall, 1995
- [SHL88] Shlaer S., Mellor S., Object-Oriented Systems Analysis: Modelling the World in Data, Englewood Cliffs, NJ, Yourdon Press, 1988
- [SŁO78] Słownik języka polskiego, PWN, Warszawa, t. I - III, 1978
- [SOM95] Sommerville I., Software Engineering, Addison-Wesley Pub. Comp., New York 1995
- [SPE86] Spector A., Gifford, D., A Computer Science Perspective of Bridge Design, Communications of the ACM, 29, s. 267-283
- [SUB97] Subieta K., Obiektość w bazach danych – koncepcje, nadzieje i fakty, III Konferencja użytkowników i developerów ORACLE, Zakopane, 1997
- [SUB97a] Subieta K., Słownik często spotykanych terminów dotyczących obiektości, Instytut Podstaw Informatyki PAN, Warszawa, 1997
- [SYM88] Symons C.,R., Function Point Analysis: Difficulties and Improvements, IEEE Transactions on Software Engineering, January 1988
- [TAT91] Tate G., Verner J. M., Approaches to measuring size of application products with CASE tools, Information and Software Technology, November 1991
- [THO89] Thomas D., In search of an object – oriented development process, J. Obj. – Oriented Programming, 2, 1989

- [TIA95] Tian J., Zelkowitz M., V., Complexity Measure Evaluation and Selection, IEEE Transactions on Software Engineering, August 1995
- [UML97] Booch G., Jacobson I., Rumbaugh J., UML Notation Guide, Rational Software Corporation, 1997, <http://www.rational.com>
- [VER88] Verner J., Tate, G., Estimation of size and effort for fourth - generation development, IEEE Software, pp.15-22, July 1988
- [VER92] Verner J., Tate G., A Software Size Model, IEEE Transactions on Software Engineering, April 1992
- [VES84] Vessey I., Weber R., Research on structured programming: An empiricist's evaluation, IEEE Transactions on Software Engineering, 1984
- [WAL77] Walston C., Felix C., A Method of Programming Measurement and Estimation, IBM Systems Journal, 16(1), 1977
- [WAN90] Wand Y., Weber R., An Ontological Model of an Information System, IEEE Transactions on Software Engineering, November 1990
- [WEY88] Weyuker E., J., Evaluating Software Complexity Measures, IEEE Transactions on Software Engineering, September 1988
- [YOU79] Yourdon E., Constantine L., L., Structured Design, Prentice Hall, Englewood Cliffs, NJ, 1979
- [ZAK90] Zakrzewski M., Wprowadzenie w teorię złożoności obliczeniowej. W kręgu zagadnienia P – NP, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 1990
- [ZIE72] Zieliński R., Tablice statystyczne, PWN, Warszawa, 1972
- [ZUS90] Zuse H., Software Complexity: Measures and Methods, Walter de Gruyter, Berlin, 1990
- [ZWE95] Zweben S., H., Edwards S., H., Weide B., W., Hollingsworth J., E., The Effects of Layering and Encapsulation on Software Development Cost and Quality, IEEE Transactions on Software Engineering, March 1995

9. Załączniki

9. 1. Opis koncepcji podejścia obiektowego

Zrozumienie technologii obiektowej w kontekście mierzenia tworzonych produktów wymaga omówienia charakteryzujących ją szczegółów oraz podkreślenia różnic w porównaniu do technologii strukturalnej. Różnice stanowią bowiem o potrzebie wprowadzenia nowych miar dla tej nowej technologii w inżynierii systemów informatycznych.

Technologia obiektowa powstaje²⁸ z udziałem informatyków znanych również jako współtwórców technologii strukturalnej, np. Peter Coad i Edward Yourdon. Wiele elementów tej technologii wprowadzono w wyniku rozwiązywania problemów związanych z tworzeniem coraz to większych i bardziej złożonych systemów w technologii strukturalnej, a technologia obiektowa ma swoje źródło zarówno w technologii baz danych, jak i w technologii programowania.

Początki programowania obiektowego datuje się na koniec lat sześćdziesiątych [COA94] - język Simula 67, a następnie Smalltalk 72 oraz późniejsze wersje tych języków. W późniejszych latach powstały obiektowe wersje języka Pascal – Object Pascal oraz języka C – C++, a także Ada, Eiffel i CLOS (ang. Common Lisp Object System). Modelowanie bazy danych w terminologii obiektów po raz pierwszy zaproponował Chena [CHE76] w drugiej połowie lat siedemdziesiątych. Przy tym podejściu świat jest opisywany za pomocą modelu związków encji (ang. Entity Relationship), a jego elementami są encje (byty, często nazywane obiektami), ich atrybuty oraz związki między tymi encjami. Tak budowany model jest opisem struktury informacyjnej. Nie znano wówczas pojęcia „operacji” i „metody” w obecnym znaczeniu. Działania realizowane przez system nazywa się funkcjami, a ich opis przedstawia model struktury funkcjonalnej.

Potrzeba jednolitej koncepcji obiektowej zarówno do tworzenia baz danych, jak i oprogramowania, obejmującej proces tworzenia systemu informatycznego od momentu jego rozpoczęcia przez analizę, projektowanie do implementacji doprowadziła do powstania pod koniec lat osiemdziesiątych metodologii obiektowej, której dojrzewanie nie zostało jeszcze zakończone i trwa w dalszym ciągu. Ze względu na różnorodne środowiska, w których się ona

²⁸ Użyto formę czasownika niedokonanego ze względu na to, że proces tworzenia podstaw teoretycznych dla technologii obiektowej nie został zakończony, gdyż wprowadza się w dalszym ciągu nowe elementy lub uszczegółowienia już istniejących, np. coraz częściej w odniesieniu do zaimplementowanej klasy używa się pojęcia „komponent” (ang. component).

rozwiija, istnieje wiele metodyk zgodnych z podstawowymi zasadami tej metodologii, ale r33niacych si33 niektórymi koncepcjami i sposobami ich realizacji. W tabeli 46 przedstawiono metodyki najcz33ściej omawiane w literaturze, a w tabeli 47 (zał. 2) por33wnano koncepcje w kilku metodykach z OOSE.

Tabela 46. Metodyki obiektowe. Źródło: [opracowanie własne].

Nazwa		Autor
OOA/OOD	Object-Oriented Analysis/Object-Oriented Design	P. Coad i E. Yourdon [COA94a,b]
OOD	Object-Oriented Design	G. Booch [BOO94]
OMT	Object Modeling Technique	J.Rumbaugh [RUM91]
OOSA	Object-Oriented Systems Analysis	S. Shlaer i S.J. Mellor [SHL88]
OOSE	Object-Oriented Software Engineering	I. Jacobson [JAC92]
OOIE	Object-Oriented Information Engineering	J. Martin [OMG89], [OMG91]
HOOD	Hierarchical Object-Oriented Design	Konsorcjum CISI Ingenieere, CRI A/S, Matra Space [HOO89a,b]
UML	Unified Modelling Language	G. Booch, I. Jacobson, J. Rumbaugh [UML97]

Główni twórcy metodyk OOD, OMT i OOSE opracowali wspólny zbiór pojęć i zasad pt. nazwą UML - „Ujednolicony J33zyk Modelowania” (ang. Unified Modelling Language) [UML97], aby uporządkować nazewnictwo i koncepcje zawarte w tych trzech metodykach. Praca ta została opublikowana w styczniu 1997 roku i jest proponowana jako standard ANSI (ang. American National Standards Institute) oraz ISO (ang.), dlatego zamieszczono ją również w wykazie metod.

W odr33żnieniu od wymienionych metodyk w systemach obiektowych baz danych panuje r33norodnořć zarówno w założeniach, architekturze jak i j33zykach. Jedną z waŹniejszych inicjatyw w tej dziedzinie było powstanie grupy OMG (ang. Object Management Group), zrzeszającej ponad 700 firm, której prace doprowadziły do powstania standardu ODMG-93 dla obiektowych baz danych, obejmującego nast33pujące elementy [SUB97]:

- Model obiektowy (precyzuje si33 w nim podstawowe pojęcia zwi33zane z technologią obiektową oraz ustala podstawowe założenia dotycz33ce struktur danych przechowywanych przez system).
- J33zyk opisu danych ODL (ang. Object Definition Language).
- J33zyk zapytań OQL (ang. Object Query Language).

- Łącznik do języka C++ (połączenie języka ODL i OQL z językiem C++).
- Łącznik do języków Smalltalk i Java (połączenie języka ODL i OQL z językiem Smalltalk i Java).

Po wprowadzeniu krótkiego zarysu historycznego, poniżej zostanie przedstawiony opis podstawowych definicji i reguł obowiązujących w technologii obiektowej, które pozwolą w dalszym ciągu niniejszej pracy na odwoływanie się do nich w sposób nie wymagający dalszych wyjaśnień. Zamieszczono objaśnienia wyprowadzone na podstawie przeglądu następującej literatury [RUM91], [JAC92], [BOO94], [COA94a,b], [MAR96], [UML97] z wyraźnym wyodrębnieniem definicji sformułowanych przez Grady Boocha.

9. 1. 1. Paradygmaty technologii obiektowej

Twórca metody OOD, Booch [BOO94], sformułował cztery paradygmaty koncepcji obiektowej: abstrakcja, hermetyzacja, modułowość i hierarchia, które muszą być spełnione przez model obiektowy niezależnie od metodyki i stylu pracy. Niespełnienie jednego z nich wyklucza obiektowy charakter danego modelu.

Abstrakcja

Istnieje wiele objaśnień pojęcia abstrakcji²⁹. Najczęściej oznacza ona coś nierzeczywistego, co istnieje jako pewien pomysł³⁰. W informatyce przyjmuje się, że jest to podstawowy sposób opisywania fragmentu świata rzeczywistego, możliwie najprostszy i wolny od nadmiernej szczegółowości. Szczególnym przypadkiem abstrakcji jest klasyfikacja, przez którą indywidualności (obiekty) występujące w świecie rzeczywistym, które posiadają podobne lub identyczne charakterystyki, są grupowane we wspólną klasę. Notacje abstrakcji i klasyfikacji prowadzą do utworzenia pojęć, które np. dla analityka systemu informacyjnego są ekwiwalentem zbiorów rzeczywistych obiektów, ich zachowań i związków między nimi, a dla programisty są ekwiwalentem przetwarzanego typu danych (ang. user-defined type).

²⁹ Zamiast abstrakcji często używanym słowem w teorii modelowania systemów informacyjnych jest granularność.

³⁰ Słownikowa definicja abstrakcji [SŁO78]:

- „pogląd lub twierdzenie nie oparte na faktach, nie odpowiadające rzeczywistości, wyprowadzone z pewnych z góry przyjętych założeń; ... „
- „wyodrębnienie w przedmiocie lub zdarzeniu konkretnym cech bądź stosunków; tworzenie idei ogólnych; rozumowanie w sposób formalny; abstrahowanie”
- „obraz, rzeźba, grafika itp. skomponowane w myśl zasad abstrakcjonizmu...”

Ze względu na to, że w strukturze systemów informacyjnych wyróżnia się kilka poziomów, to również pojęcie abstrakcji, jako sposób opisu każdego z poziomów jest wielopoziomowe, np. system, podsystem, klasa i metoda. Oznacza to, że abstrakcja na coraz to niższych poziomach, stanowi opis danej rzeczywistości o coraz to wyższym stopniu szczegółowości.

Hermetyzacja

Hermetyzacja (ang. encapsulation), interpretując wprost, oznacza ukrywanie informacji. Jej rola w technologii obiektowej jest szczególnie wyraźna na etapie implementacji klas lub na etapie pielęgnacji programu. Daje ona możliwość ochrony programu przed wzajemnym uzależnianiem się jego elementów składowych, co pozwala na przeprowadzenie zmian w wybranym obiekcie bez wpływu na działanie innych części aplikacji. Szczegółowo zaś można określić hermetyzację jako mechanizm, który pozwala na oddzielenie implementacji aspektów zewnętrznych obiektu, które są też dostępne innym obiektom, od implementacji szczegółów wewnętrznych obiektu, które są ukryte przed innymi obiektami. (Zbiór publicznie dostępnych elementów obiektu danej klasy nazywa się publicznym interfejsem obiektu.) Decyzje o ukrywaniu i udostępnianiu informacji podejmowane są na etapie projektowania. Zgodnie z definicją Boocha hermetyzacja jest procesem „szufladkowania” elementów abstrakcji, co pozwala na ukrywanie ich struktury oraz sposobu zachowania³¹ (ang. behavior).

Modułowość

Podział programu na moduły jest dokonywany na etapie projektowania systemu. Tradycyjnie, w technologii strukturalnej modułowość polega na grupowaniu podprogramów przy zachowaniu kryterium spójności i powiązań. W projektowaniu obiektowym modułowość wymaga podjęcia decyzji o fizycznym umiejscowieniu klasy i obiektu w logicznej strukturze projektu oraz jest częścią projektu struktury fizycznej systemu. Decyzje te nie należą do łatwych i powinny uwzględniać z jednej strony granice wyznaczone przez pojedyncze abstrakcje, czyli mechanizm hermetyzacji, a z drugiej potrzebę podziału na moduły. Jest to element ważny również ze względu na czas trwania budowy systemu, gdyż modularyzacja przeprowadzona przy zapewnieniu niewielkiej liczby połączeń między modułami prowadzi do

³¹ W literaturze polskiej występuje tłumaczenie angielskiego terminu *behavior* w odniesieniu do obiektu, jako zachowanie obiektu, które reprezentuje kolekcję abstrakcji proceduralnych przypisanych do obiektów (klasy) [SUB97a].

zredukowania złożoności tego systemu. Modułowość jest traktowana jako własność systemu informacyjnego, która przedstawia jego dekompozycję na spójne i luźno powiązane zbiory modułów. Ta definicja wskazuje na istotną różnicę w kryteriach obowiązujących przy tworzeniu modułów w technologii obiektowej w porównaniu do strukturalnej, gdzie wymaga się, by moduły były spójne ale też i ściśle powiązane.

Hierarchia

Hierarchia wprowadza uporządkowanie zbioru elementów według stopniowania zależności istniejących między nimi. Podobnie definiuje się pojęcie hierarchii występującej w systemach informacyjnych, tworzenie hierarchii polega na przydzielaniu odpowiedniej rangi wyróżnionym abstrakcjom oraz ich uporządkowaniu. Pojęciem bezpośrednio związanym z hierarchią jest dziedziczenie (ang. inheritance), które stanowi zasadniczy element systemów obiektowych. Stosowane są tutaj dwa podejścia do klasyfikacji dziedziczenia: dziedziczenie jednokrotne i wielokrotne oraz generalizacja i specjalizacja. Mechanizm dziedziczenia odpowiada zatem hierarchii abstrakcji, w której podklasa dziedziczy własności od jednej lub więcej superklas. Jednokrotne dziedziczenie wskazuje, że klasa może mieć tylko jedną klasę przodka, a wielokrotne dziedziczenie występuje wtedy, gdy podklasa może dziedziczyć własności od więcej niż jednej klasy przodka. Superklasa przedstawia generalizację abstrakcji, a podklasa specjalizację.

Spośród opisanych czterech paradygmatów technologii obiektowej dwa z nich: abstrakcja i modułowość występują z podobną siłą również w metodykach strukturalnych, z tą różnicą, że w technologii obiektowej występują nowe poziomy abstrakcji: klasa, metoda a podział systemu na moduły odbywa się przy zachowaniu zmienionych kryteriów. O hermetyzacji trudno jest mówić, gdyż pojęcie to nie występuje w technologii strukturalnej. Paradygmatu struktur hierarchicznych można doszukać się w strukturalnych modelach związków encji. Występuje w nich pojęcie nadtypu i podtypu, ale wykorzystywane jedynie w celu zasygnalizowania projektantowi wspólnych szczegółów między tymi typami obiektów. Problem jest rozwiązywany za pomocą decyzji projektowych, w wyniku których w otrzymanym projekcie tabel nie występuje już pojęcie hierarchii. Mechanizm dziedziczenia, który jest konsekwencją istniejących powiązań hierarchicznych nie ma w tej technologii możliwości implementacyjnych.

Pojęcia hermetyzacji i mechanizmu dziedziczenia, występujące w technologii obiektowej są podstawowym elementem, który różnią podejście obiektowe od podejścia strukturalnego. Ich istotny wpływ na czas realizacji systemu zauważa się szczególnie w procesie kodowania programu.

9. 1. 2. Definicje podstawowych elementów

Obiekt

Obiekt (ang. object) jest podstawowym elementem modelu systemu, który przedstawia występujące w świecie realnym, np. osoby, przedmioty, zdarzenia oraz wykonywane w nim działania, na przykład.:

Oblicz saldo na koniec miesiąca.

Wyszukaj pracownika o najkrótszym stażu pracy.

W procesie tworzenia systemów używa się też pojęcia *wystąpienie* (ang. instance), które w odniesieniu do zbioru obiektów oznacza pojedynczy obiekt i jest stosowane zamiennie z pojęciem *obiekt*. Obiekty, dla których tworzony jest model, ze względu na posiadane różne własności (ang. properties) oraz zachowania są klasyfikowane na różne podzbiory obiektów. Przykładem własności obiektu jest np. waga, długość, kategoria. Zachowanie obiektów może dotyczyć ich własności lub czynności wykonywanych przez ten obiekt. Niektóre obiekty nie wykonują żadnych operacji, ale istnieje potrzeba udostępniania informacji o ich stanie, np. dla obiektu *AKUMULATOR* istotne są pytania:

Do jakiej kategorii należy ?,

Jaka jest data jego ważności?

Zachowanie obiektu *JAN KOWALSKI* może dotyczyć:

- operacji wykonywanej na jego własnościach, np.: *Pokaż wiek, Zapamiętaj datę urodzenia,*
- operacji wykonywanej przez obiekt, np.: *Maluje, Gra, Uczy się.*

Według Boocha, do cech charakteryzujących obiekt należą: tożsamość, zachowanie i stan. Według definicji Jacobsona obiekt jest charakteryzowany przez zestaw operacji oraz stan, a operacje odpowiadają zachowaniu obiektu z definicji Boocha. Dla każdego wyróżnionego podzbioru obiektów definiuje się na etapie projektowania klasę.

Tożsamość

Tożsamość jest własnością obiektu, która umożliwia wyróżnienie go spośród wszystkich innych obiektów. W większości systemów baz danych w tym celu używa się specjalnie skonstruowanych kluczy, których rolą jest jednoznaczna identyfikacja pojedynczego obiektu. Własnością wyróżniającą konkretny obiekt nie powinna być jego nazwa, gdyż obiektów z tą samą nazwą może być wiele, co wówczas prowadzi do niemożliwości rozpoznania jednego konkretnego obiektu.

Stan obiektu

Obiekt jest opisywany przez odpowiednio dobrany zestaw własności, które charakteryzują jego naturę, np. numer, długość, data urodzenia. Dla różnych obiektów mogą one przyjmować inne wartości. Wartości wszystkich własności obiektu określają jego stan. Niektóre z własności obiektu, w trakcie okresu³² jego życia mogą się zmienić (np. nazwisko pracownika) i wówczas są nazywane własnościami dynamicznymi, a niektóre pozostają niezmienione, np. numer seryjny samochodu, które nazywane są własnościami statycznymi.

Zachowanie

Uzupełnieniem charakterystyki obiektu jest opis jego zachowania, które przedstawia z jednej strony, jak obiekt działa, a z drugiej, jak reaguje na zdarzenia wynikające z działania innych obiektów. Wynikiem tych działań jest odpowiedni stan obiektu. W przypadku działania jednego obiektu na drugi stosuje się pojęcie *przekazywanie komunikatów*³³. Ze względu na dwukierunkowy charakter zachowania obiektu (działanie i oddziaływanie) w definicji zwraca się uwagę na to, że „stan obiektu przedstawia skumulowane wyniki jego zachowania” [BOO94, s. 87].

³² Okres życia obiektu (ang. object life span) trwa od czasu utworzenia go po raz pierwszy (czyli zajęcia miejsca w przestrzeni pamięci) aż do zwolnienia tej części pamięci.

³³ W obiektowej koncepcji Boocha pojęcie operacji i komunikatu jest traktowane zamiennie. W językach obiektowych, takich jak Smalltalk, mówi się, że jeden obiekt przekazuje komunikat innemu obiektowi, a w językach takich, jak C++ mówi się, że jeden obiekt wywołuje funkcję członka innego obiektu. Ogólnie, komunikat jest prostą operacją.

Operacje

Pojęcie operacji³⁴ w technologii obiektowej jest związane z etapem modelowania konceptualnego i przedstawia zachowanie obiektu. W niektórych metodykach zachowanie obiektu nazywane jest operacją [RUM91, s. 3] bądź usługą [COA94a, s. 151], natomiast implementacja operacji i usługi nazywana jest metodą. W praktyce wykonywanych jest pięć rodzajów operacji:

- modyfikację – zmienia stan obiektu,
- selekcję – ma dostęp do stanu obiektu, ale go nie zmienia,
- iterację – udostępnia wszystkie części obiektu w dobrze zdefiniowanym porządku,
- konstrukcję – tworzy obiekt i/lub inicjuje jego stan,
- destrukcję – zwalnia stan obiektu i/lub usuwa obiekt.

Połączenia i agregacja obiektów

Pojedyncze obiekty przedstawiają ubogi fragment rzeczywistości, w której występuje wiele powiązań umożliwiających współpracę między obiektami. Stąd przedmiotem analizy i projektowania obiektowego są, występujące w hierarchii obiektów, dwa podstawowe rodzaje połączeń: łącznik (ang. link), agregacja (ang. aggregation). Według Rumbaugh łącznik jest „fizycznym lub konceptualnym związkiem między obiektami” [RUM91, s. 27], a według Boocha łącznik „wskazuje specyficzne związki, przez które jeden obiekt (klient³⁵) stosuje usługi innego obiektu (dostawcy) lub przez które jeden obiekt może sterować innym” [BOO94, s. 98]. Łącznik między dwoma obiektami przedstawiany jest na diagramie za pomocą linii łączącej oba obiekty, która wyznacza ścieżkę przekazywania komunikatów (udostępniania operacji). Agregacja, jako szczególny rodzaj związku, według Rumbaugh może być dwojakiego typu „jest częścią całości” lub „składa się z”, w którym obiekty przedstawiające składniki jakiejś całości są powiązane z obiektem przedstawiającym tę całość. Podstawowymi własnościami agregacji są: przechodniość, jeśli *A* jest częścią *B* i *B* jest częścią *C*, to *A* jest

³⁴ Działanie i oddziaływanie (reakcja) jest nazywane operacją, np. dodaj obiekt, oblicz. W większości języków programowania obiektowego operacje są opisywane jako metody i definiowane są w części deklaracyjnej klasy. W języku C++ zamiast metody używana się również pojęcia funkcji członka.

Jednak nie wszystkie operacje stają się metodami, gdyż w językach takich, jak Object Pascal, C++, Ada występują operacje nazywane „wolnymi podprogramami”. W języku C++ nazywane są funkcjami nie będącymi członkami. Natomiast wszystkie metody są operacjami.

W obiektowej koncepcji Boocha pojęcia operacji i metody są używane zamiennie.

³⁵ Klient jest obiektem, który używa zasobów innego obiektu, nazywanego wówczas serwerem.

częścią B , antysymetria, jeśli A jest częścią B , to B nie jest częścią A . Związek agregacji może, ale nie musi odpowiadać fizycznemu zawieraniu się części w całości, np. związek „akcjonariusz jest właścicielem akcji” przedstawia jeden ze sposobów logicznego agregowania, chociaż akcje nie są fizyczną częścią akcjonariusza. Przykładowy związek odzwierciedla konceptualną agregację i nie stanowi fizycznej agregacji części w całość.

Klasa

Zbiór obiektów, które cechują wspólne własności i wspólne zachowanie w technologii obiektowej jest przedstawiany za pomocą abstrakcji, nazywanej klasą (rys. 31). Pojedynczy obiekt jest nazywany wystąpieniem klasy i samodzielnie nie może stanowić klasy, ale klasa może być obiektem. Istnieje również ograniczenie co do złożoności klasy: abstrakcja wysokiego poziomu, taka jak baza danych czy cały system nie powinny być wyrażone jako pojedyncze klasy. Takie abstrakcje nazywane są klastrami klas lub komponentami³⁶.

Osoba
data urodzenia
nazwisko
oblicz wiek

Rysunek 31. Przykład modelu klasy, nazwa klasy – *Osoba*, atrybuty – *data urodzenia*, *nazwisko*, operacja – *oblicz wiek*. Źródło: [opracowanie własne].

Klasy, podobnie jak obiekty, nie istnieją w odosobnieniu, lecz dla poszczególnych dziedzin problemu tworzą struktury klas przez zachodzące między nimi powiązania.

Każdy związek opisuje grupę powiązań ze wspólną strukturą i wspólną semantyką. Związek opisuje zbiór potencjalnych powiązań w taki sam sposób, w jaki klasa opisuje zbiór potencjalnych obiektów.

³⁶ Pojęcie komponentu występuje w języku polskim jako „część składowa czegoś” [SŁO78]. W technologii obiektowej odnosi się przeważnie do małego fragmentu programu, który wykonuje specyficzną funkcję [SUB97].

Rodzaje związków między klasami

Koncepcja modelowania rzeczywistości przy podejściu obiektowym umożliwia wyróżnienie sześciu rodzajów związków, z których tylko trzy pierwsze występują we wszystkich metodykach obiektowych:

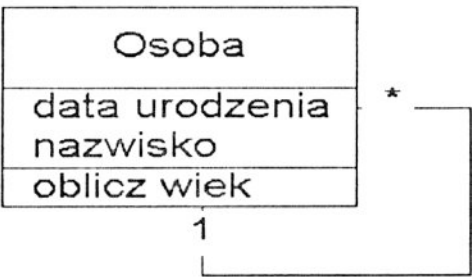
- asocjacja (ang. association)
- dziedziczenie
- agregacja (ang. aggregation)
- użycie (ang. using)
- metaklasa (ang. metaclass).

Asocjacja

Związek typu asocjacji jest najczęściej identyfikowanym na etapie analizy związkiem występującym między klasami, który na etapie projektu niekiedy zmienia swoją kwalifikację. Przedstawia on semantyczną zależność między modelowanymi zbiorami informacji i ich zachowaniem. Krotność dla wystąpień poszczególnych związków jest określana za pomocą trzech podstawowych typów licznosci:

- jeden – do – jeden
- jeden – do – wiele
- wiele – do – wiele.

W praktyce występują różne kombinacje i uszczegółowienia licznosci, np. „jeden lub więcej” lub „zero lub więcej”, a także konkretne zbiory liczb, np. „2, 6, 9”. Licznosc precyzuje, jak wiele wystąpień jednej klasy może łączyć pojedyncze wystąpienie drugiej klasy, będącej w danym związku. Związki asocjacji są dwukierunkowe. Kierunek wynikający z nazwy związku jest kierunkiem "przednim" (forward), a przeciwny „odwrotnym” (inverse). W rzeczywistości oba kierunki są jednakowo znaczące. W jednym związku mogą wystąpić jedna, dwie lub więcej klas, stąd wynikają następujące typy związków: unarny, binarny, n -arny. Związki unarne opisują zależności występujące w tym samym zbiorze obiektów. Na rysunku 32 pokazano sposób modelowania związków unarnych, których przykładem może być związek między szefem a podwładnymi, gdzie szef i podwładni stanowią zbiór obiektów o podobnych własnościach i tworzą jedną klasę *Pracownik*.

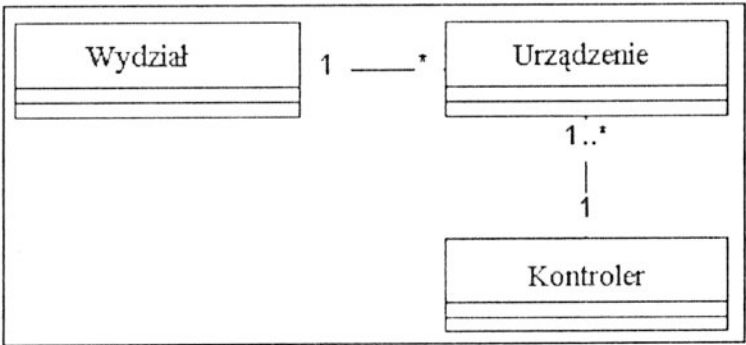


Rysunek 32. Związek unarny, który przedstawia powiązania między osobami. Źródło: [opracowanie własne].

Związki wyższego typu są skomplikowane i utrudniają implementację, z tego powodu najczęściej stosowane są związki unarne i binarne. Na rysunku 32 przedstawiono dwa związki binarne: „Urządzenie znajduje się tylko na jednym Wydziale” oraz „Kontroler odpowiada za jedno lub więcej Urządzeń”, które czytane w kierunku odwrotnym będą brzmiały: „Wydział posiada zero lub więcej Urządzeń” oraz „Urządzenie ma odpowiedzialnego tylko jednego Kontrolera”.

W modelu tym (rys. 33) znajdują się następujące elementy:

- nazwy klas – Urządzenie, Wydział, Kontroler,
- nazwy ról (końców) – znajduje się na, odpowiada za,
- liczność związków – „jeden – do- wiele” i „jeden - do – wiele”.

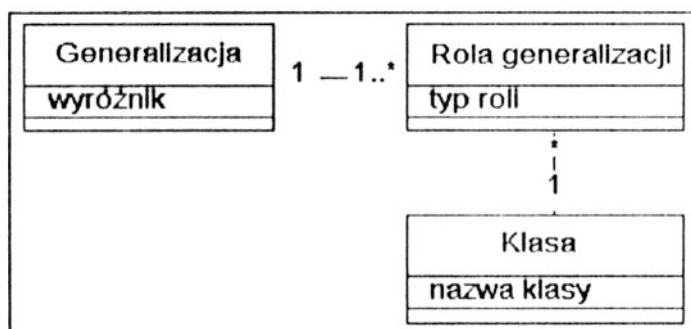


Rysunek 33. Binarne związki według notacji UML. Źródło: [opracowanie własne].

Każdy związek asocjacji ma dwa końce, nazwę roli opisuje się na diagramie tylko dla jednego końca, a odczytuje w dwie strony i stąd w interpretacji słownej znalazły się dla dwóch związków cztery nazwy ról. Natomiast liczność związku opisuje się dla każdego końca oddzielnie. Oba związki różni koniec po stronie końca „wiele”. Zamieszczony na rysunku znak „*” (rys. 33) oznacza, że Wydział może posiadać dowolną liczbę urządzeń lub nie posiadać żadnego, natomiast „1..*” oznacza, że Kontroler musi odpowiadać przynajmniej za jedno urządzenie.

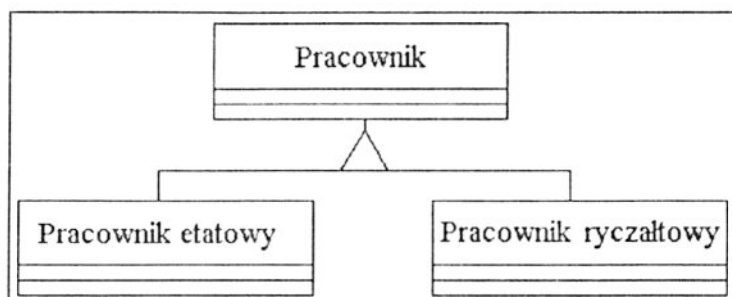
Dziedziczenie

Z pojęciem dziedziczenia związane są generalizacja i specjalizacja, które odpowiadają tej samej idei i często są używane zamiennie. Według Rumbaugh generalizacja jest używana w odniesieniu do związku między klasami, dziedziczenie odpowiada mechanizmowi współdzielenia atrybutów (własności) i operacji używanych przez związek generalizacji. Według Boocha pojęcie dziedziczenie również stosuje się do określenia rodzaju związku.



Rysunek 34. Meta model związku generalizacji. Źródło: na podstawie [RUM91, s. 82]

Generalizacja i specjalizacja (rys. 34) są dwoma różnymi punktami widzenia tego samego związku. Jeżeli wychodzi się od superklasy, to mówi się o specjalizacji, jeżeli od podklasy, to mówi się o generalizacji. Generalizacja jest relacją między klasą i jedną lub więcej wyspecjalizowanymi wersjami tej klasy. Klasa wyspecjalizowana jest nazywana podklasą i każda wyspecjalizowana wersja jest podklasą. Klasa specjalizowana jest nazywana superklasą. Używając terminu dziedziczenie, klasa, od której dziedziczą inne klasy nazywana jest superklasą, natomiast klasa, która dziedziczy od jednej lub więcej klas nazywana jest podklasą (rys. 35).



Rysunek 35. Przykład związku dziedziczenia, klasa *Pracownik* jest superklasą, klasy *Pracownik etatowy* i *Pracownik ryczałtowy* są podklasami. Źródło: [opracowanie własne].

Oczekuje się od najbardziej wyspecjalizowanej klasy, tzn. znajdującej się na dole hierarchii, posiadania wystąpień i wówczas taka klasa nazywana jest konkretną. Wobec klas pośrednich, bardziej uogólnionych, nie występuje takie żądanie i klasa, która nie posiada wystąpień nazywana jest klasą abstrakcyjną. Każda podklasa nie tylko dziedziczy wszystkie cechy swoich przodków³⁷, ale dodaje swoje własne atrybuty i operacje. Odziedziczone przez podklasę cechy i zachowania mogą zostać, jeżeli zachodzi taka potrzeba, zmodyfikowane przez wykorzystanie występującej wraz z mechanizmem dziedziczenia, zasady polimorfizmu³⁸. Zjawisko to jest nazywane nadpisywaniem (ang. overriding).

Nadpisywanie własności i operacji

Podklasa może zawierać nadpisywane cechy superklasy przez zdefiniowane cechy o tej samej nazwie. Własność nadpisana (własność podklasy) specjalizuje i zastępuje nadpisaną własność (własność superklasy). Większość konfliktów pojawiających się w trakcie implementacji jest związana właśnie z nadpisywaniem operacji - metod. Trudności występują szczególnie wtedy, kiedy nadpisywana metoda poważnie różni się od nadpisywanej. Wówczas, zamiast oczekiwanego zmniejszenia czasu pracy, wynikającego z możliwości ponownego użycia, może wystąpić poważne utrudnienie pracy, równoznaczne z jej wydłużeniem. Polimorfizm pozwala zaimplementować operację zależną od wystąpienia klasy - obiektu. Na przykład implementacja operacji "oblicz-płatę" może zależeć od różnych obiektów, reprezentujących pracowników przedsiębiorstwa. Z jednej strony polimorfizm redukuje

³⁷ W analogii do drzewa genealogicznego używane są w modelowaniu klas i obiektów pojęcia przodek (rodzic), który jest odpowiednikiem superklasy oraz potomek (dziecko) – podklasy.

³⁸ Polimorfizm oznacza, że operacja o tej samej nazwie może mieć różne zachowanie dla różnych klas [RUM91, s. 2].

złożoność systemu obiektowego, ale może też powiększyć jego złożoność. Zatem, przy sterowaniu złożonością systemów obiektowych przez użycie polimorfizmu istotne jest zapewnienie, że wszystkie operacje z tą samą nazwą są semantycznie zgodne. Nadpisywanie najczęściej jest stosowane z powodu:

- rozszerzenia odziedziczonych zachowań,
- restrykcji, zawężenia zachowania do określonych typów argumentów,
- optymalizacji algorytmu (wewnętrzna reprezentacja nadpisywanej metody i algorytm mogą być całkowicie różne, ten sam musi być zewnętrzny protokół³⁹ oraz wyniki otrzymane z wykonania tej metody),
- dla udogodnienia, wówczas poszukuje się klasy podobnej do żądanej i stanowią nową klasę jako podklasę istniejącej, a nadpisuje się te metody, które są nieodpowiednie. Takie użycie dziedziczenia jest semantycznie niepoprawne i prowadzi do problemów pielęgnacyjnych tego programu.

Dziedziczenie jest użyteczną konstrukcją podczas conceptualnego modelowania i implementacji. W trakcie modelowania pozwala na odkrycie tego, co jest w klasach podobne i co jest różne, a na etapie implementacji pozwala na skrócenie czasu tworzenia programu przez korzystanie z pokazanych możliwości ponownego użycia części napisanego kodu programu.

Wielokrotne dziedziczenie

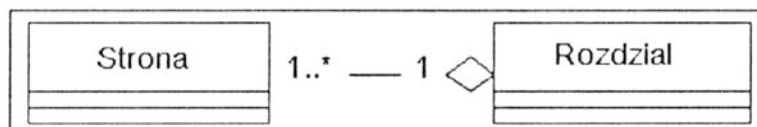
Przy pojedynczym dziedziczeniu każda klasa ma dokładnie jedną superklasę. Wprowadzono znaczne rozszerzenie możliwości mechanizmu dziedziczenia, zwiększając funkcjonalność klasy przez dziedziczenie od więcej niż jednej superklasy. Klasa z więcej niż jedną superklasą jest nazywana klasą łącznikiem (join class). Przy takim sposobie modelowania może wystąpić niebezpieczeństwo dublowania się własności i operacji, np. dwie superklasy *A* i *B* posiadają operacje o takiej samej nazwie *wyświetl*. Powstaje wówczas konflikt: którą operację powinna dziedziczyć podklasa *C*? W celu wyeliminowania takich dwuznaczności wprowadzono zastrzeżenie dotyczące cech klasy przodka, które mogą być znalezione po więcej niż jednej ścieżce – mogą być dziedziczone tylko raz, gdyż jest to ta sama cecha [RUM91, s. 65]. Konkretny sposób rozwiązania tego problemu zależy od języka implementacji. Wielokrotne dziedziczenie z rozwiązaniem problemem dublowania pozwala, by klasa miała więcej niż jedną superklasę i korzystała z możliwości wszystkich rodziców. Istotne

³⁹ Protokół obiektu zawiera definicję dozwolonego zachowania obiektu widzianego z zewnątrz [BOO94, s.517].

korzyści, jakie wówczas wynikają, pojawiają się przy szczegółowej specyfikacji klas oraz zwiększają możliwość ponownego użycia w trakcie implementacji.

Agregacja

Związek agregacji między klasami bezpośrednio odpowiada związkowi typu „część – całość” między obiektami (rys. 36).



Rysunek 36. Związek agregacji „obiekt *Rozdział* składa się z jednego lub więcej obiektów *Strona*”; znak rombu wskazuje na związek typu agregacja, znak „1..*” określa stopień agregacji. Źródło: [opracowanie własne].

Użycie

Związek „użycia” służy do wskazania postawionego wymagania dla konkretnego związku między klasami: wyznaczenie roli każdej z klas w tym związku, która z nich pełni rolę klienta a która - dostawcy.

Meta klasa

Koncepcja meta klasy powstała na potrzeby modelowania związków między klasami, z których jedna przedstawia wystąpienia drugiej klasy. Zgodnie z zasadą metodyki obiektowej wystąpieniami klasy są obiekty. Jeżeli jednak wystąpi sytuacja, w której wystąpieniami danej klasy są również klasy, to ta klasa stanowi meta klasę w związku między klasami takiego typu.

9. 1. 3. Charakterystyka procesu tworzenia programu

W procesie tworzenia systemu informatycznego przy podejściu obiektowym wyróżnia się podobne etapy, jak przy podejściu strukturalnym, ale sposób ich realizacji charakteryzuje się wyraźną przewagą znaczenia iteracji. Proces o charakterze iteracyjnym wykonywany jest w kilku krokach, np. po przeprowadzeniu w pierwszym kroku etapów analizy i projektowania, otrzymane wyniki są przenoszone do następnego kroku analizy i projektowania. Oznacza to sukcesywne doskonalenie budowanej architektury obiektowej przez iteracyjne kroki tych etapów. Produkty otrzymywane na etapie analizy i projektowania, modyfikowane przez iterację, prowadzą do otrzymania systemu spełniającego zmieniające się potrzeby klienta. Takie postępowanie jest korzystne również dla drugiej strony, ponieważ powiększają

stopniowo wiedzę twórców o tworzonej systemie. Model takiego procesu tworzenia znacznie odbiega do tradycyjnego modelu procesu kaskadowego i jest zbliżony do modelu spiralnego. Zastosowanie w tym modelu techniki prototypowania⁴⁰ jest dodatkowym elementem, który zbliża go do modelu spiralnego.

Analiza ogólna	Analiza szczegółowa	Projektowanie	Implementacja	Testowanie

	Cały system
	I część
	II część
	III część

Rysunek 37. Przyrostowy proces powstawania kolejnych części produktu. Źródło: [opracowanie własne].

Drugą cechą, odróżniającą proces tworzenia przy obu podejściach jest budowa przyrostowa (rys. 37). Wówczas etapy analizy ogólnej wymagań dotyczy całego systemu, natomiast przez następne etapy realizowane są kolejne części systemu. Każde przejście przez cykl „analiza – projektowanie – implementacja – testowanie” prowadzi również do stopniowego poprawiania decyzji strategicznych i taktycznych, ostatecznie skupionych na rozwiązaniu spełniającym rzeczywiste wymagania użytkownika końcowego.

Podobnie, jak przy podejściu strukturalnym, proces tworzenia małych aplikacji jest mniej skomplikowany od procesu tworzenia złożonych systemów, zarówno przez skalę rozwiązywanego problemu i wielkość zaangażowanego do tej pracy zespołu. Tradycyjne etapy analizy i projektowania nie są tak wyraźnie oddzielone, a często doświadczenie wraz z inteligencją twórców zastępują reguły metodyczne. Dlatego przy badaniu wielkości nakładów potrzebnych do tworzenia, konieczne jest uwzględnienie takiej cechy produktu informatycznego, która pozwoli wyróżnić proces tworzenia w skali mikro i makro (szczególnie potrzeba zarządzania wersjami).

⁴⁰ Prototyp jest częściową implementacją produktu, wyrażonego za pomocą logicznego lub fizycznego modelu wraz ze wszystkimi zewnętrznymi interfejsami. Jest on tworzony z myślą o użytkowniku, który po

Model cyklu życia

W technologii obiektowej zauważa się brak zdecydowania o odpowiednim modelu cyklu życia. Najczęściej wymienia się podobieństwo do modelu spiralnego lub sugeruje zupełnie nowy model, przypominający działanie fontanny [HEN96]. W różnych metodykach obiektowych występują etapy znane z technologii strukturalnej: analiza, projektowanie, implementacja, wdrożenie i pielęgnacja, czasami niektóre zagregowane bądź zdekomponowane. Zaś we wszystkich metodykach występuje nowy proces: tworzenie i pielęgnacja komponentów, które mogą być wykorzystywane w kilku różnych aplikacjach.

Model procesu tworzenia w metodyce Boocha

Organizacja procesu tworzenia systemu w metodyce Boocha [BOO94] zależy od jego wielkości. W przypadku małych systemów, model cyklu życia jest bliski modelowi spiralnemu, a proces tworzenia nazywany jest mikro procesem. W dużych systemach proces tworzenia jest bardziej związany z tradycyjnym modelem kaskadowym i nazywany jest makro procesem. Działania, składające się na mikro proces stanowią także jądro makro procesu, który dodatkowo zawiera ramy organizacyjne i reguły zarządzania. Podstawową ideą makro procesu jest charakterystyczny dla technologii obiektowej mechanizm tworzenia przyrostowego. Makro proces składa się z tradycyjnych etapów analizy, projektowania oraz implementacji. Zawiera dodatkowy etap konceptualizacji, który ma na celu ustalenie, za pomocą budowy prototypu, pomysłu opracowania nowego systemu bądź też adaptacji istniejącego.

W mikro procesie wyróżnia się cztery główne kroki postępowania [BOO94, s. 235], w których realizowane są działania przechodzące przez kolejne iteracje podstawowych etapów cyklu tworzenia: analizy, projektowania i implementacji (rysunek).

- 1) **Identyfikacja klas i obiektów** na ustalonym poziomie abstrakcji. Przy zastosowaniu trzech wymienionych typów klasyfikacji (klasycznego, analizy zachowań i analizy przypadków użycia) odkrywane są kluczowe abstrakcje w granicach badanego problemu, które tworzą słownik dziedziny problemu i stanowią potencjalne klasy i obiekty; są to czynności należące do etapu analizy. Wymyślane są nowe abstrakcje, które stanowią elementy przyszłego rozwiązania – jest to część etapu projektowania. Tworzone są abstrakcje

przeanalizowaniu jego działania, pozwala ostatecznie wyjaśnić rzeczywiste wymagania wobec przyszłego systemu.

niższego poziomu, uwzględniające potrzeby implementacyjne. Wszystkie zidentyfikowane abstrakcje składają się na słownik danych (ang. data dictionary) realizowanego projektu, który stanowi centralne miejsce gromadzenia elementów powstającego systemu i jest aktualizowany w miarę postępującego procesu tworzenia.

- 2) Identyfikacja semantyczna **klas i obiektów**. Bada się rolę klas i obiektów zidentyfikowanych w poprzednim kroku. Analiza – zlokalizowanie i specyfikacja dla każdej abstrakcji odpowiedzialności za zachowanie systemu na podstawie wygenerowanego scenariusza, odkrycie wzorców zachowań dla klas i obiektów, które mogą stanowić o możliwości ich wielokrotnego użycia, identyfikacja atrybutów niezbędnych do budowy scenariusza. Projektowanie – umiejscowienie operacji na odpowiednim poziomie abstrakcji, ewentualne wprowadzenie klas abstrakcyjnych. W tym kroku klasy stanowią odizolowane abstrakcje z nazwanymi operacjami i ważniejszymi atrybutami, a projekt klas przedstawia ich semantykę, która odpowiada przede wszystkim zachowaniu obiektów. Projekt ten nie uwzględnia jeszcze projektu architektury. Implementacja – specyfikacja konkretnego protokołu dla każdej abstrakcji z użyciem narzędzi do testowania i potwierdzenia decyzji projektowych. Wynikiem realizacji tego kroku jest zaktualizowany słownik danych, zawierający definicje z diagramu obiektów i diagramu interakcji.
- 3) Identyfikacja relacji między **klasami i obiektami**. Analiza - identyfikowane są interakcje klas i obiektów wraz ze związkami dziedziczenia i agregacji, definiowane są pozostałe atrybuty i operacje na podstawie kompletnego scenariusza. Projektowanie – przedstawienie projektu architektury, z uwzględnieniem mechanizmów metodycznych: tworzenie kategorii klas i modułów za pomocą klastrowania. Implementacja – przeddefiniowanie niektórych związków asocjacji ze względu na potrzeby implementacyjne oraz wprowadzenie związków inicjacji oraz „użycia”. Wynikiem realizacji tego kroku jest zaktualizowany słownik danych, zawierający definicje z diagramów klas, obiektów i modułów.
- 4) Implementacja **klas i obiektów**. Czynności wykonywane w trzech poprzednich krokach skupiają się wokół zewnętrznej strony wszystkich abstrakcji, natomiast w kroku czwartym dotyczą ich strony wewnętrznej. Następuje udoskonalenie istniejących abstrakcji przez kolejną iterację etapu analizy, której efektem może być, np. ujawnienie nowych abstrakcji, do których mogą być delegowane⁴¹ zachowania. Iteracyjne projektowanie sukcesywnie

⁴¹ Delegacja jest alternatywnym mechanizmem dziedziczenia, występującym w językach obiektowych. Obiekty są wówczas traktowane jako prototypy (egzemplarze), które przekazują swoje zachowanie do związanych z nimi innych obiektów, eliminując w ten sposób pośrednictwo przez klasę.

wspomaga dochodzenie do powstania adekwatnego modelu abstrakcji i podjęcia decyzji o odwzorowaniu do modelu fizycznego w danym języku implementacji. Implementacja polega na wygenerowaniu kodu za pomocą odpowiednich narzędzi programowych.

9. 2. Porównanie pojęć w metodykach

Tabela 47. Porównanie pojęć występujących w metodyce OOSE z innymi metodykami technologii obiektowej. Nawiasy wskazują na możliwość takiej analogii, ale bez formalnej jednoznaczności. Źródło: [JAC92, s. 501].

OOSE	OOA	OOD	HOOD	OMT
Klasa	klasa	klasa	(klasa obiekt)	klasa
Obiekt	obiekt	obiekt	obiekt	obiekt
Dziedziczenie	struktura gen-spec	dziedziczenia		generalizacja
Znajomość	połączenie wystąpienia /całość-część	(relacje)	(relacje)	połączenie
Komunikacja	powiązanie typu komunikat	(użycie/relacja inicjacji)	(relacje)	(przepływ danych)
Bodziec	komunikat	komunikat	(bodziec)	zdarzenie
Operacja	usługa	operacja	operacja	operacja
Atrybut	atrybut	pole	(typ)	atrybut
Aktor	(użytkownik)		(obiekt środowiskowy)	
Przypadek użycia	wątki wykonania	(=mechanizmy)	(przepływ sterujący)	scenariusz
Podsystem	(tematy)	kategorie klas/podsystemy	(Root) obiekt	podsystem
Pakiet usług				
Blok		moduł/kategoria klas	obiekt	moduł
Moduł obiektowy – moduł na poziomie języka		klasy	obiekt końcowy	klasy
Publiczny moduł obiektowy		(kategorie widzialności klas)	dostarczony interfejs	(usługa)

9. 3. Metryki Lorenza i Jacobsona

Tabela 48. Zestaw metryk Lorenza proponowanych dla projektu programu. Źródło: [LOR93].

Lp.	Nazwa metryki	Klasa mierzonego obiektu
1	Średni rozmiar metody	Rozmiar
2	Średnia liczba metod dla klasy	Rozmiar
3	Średnia liczba zmiennych dla klasy	Powiązanie
4	Poziom głębokości hierarchii klas	Rozmiar
5	Liczba relacji podsystem/podsystem	Powiązanie
6	Liczba relacji klasa/klasa wewnątrz każdego podsystemu	Spójność i powiązanie
7	Użycie zmiennych	Złożoność semantyczna
8	Średnia liczba linii komentarza	Złożoność pojęciowa
9	Liczba raportów dla klasy	Proces
10	Krotność ponownego użycia dla klasy	Proces
11	Liczba klas i metod wyrzuconych	Proces

Tabela 49. Metryki procesu i programu Jacobsona. Źródło: [JAC92].

Nazwa metryki
Proces Całkowity czas tworzenia; Czas tworzenia w każdym procesie i podprocesie; Czas poświęcony modyfikowaniu modeli z wcześniejszych procesów; Czas poświęcony na podprocesy takie, jak: specyfikacja przypadków użycia obiektów, projektowanie i testowanie bloków i przypadków użycia; Liczba różnego rodzaju znalezionych błędów; Liczba propozycji zmian dla wcześniejszych modeli; Koszt zapewnienia jakości; Koszt wprowadzenia nowego procesu i narzędzi tworzenia.
Program <i>Metryki ogólne</i> Całkowita liczba klas; Liczba klas ponownie użytych i nowo utworzonych; Całkowita liczba operacji; Liczba operacji ponownie użytych i nowo utworzonych; Całkowita liczba przesyłanych komunikatów; <i>Metryki specyficzne</i> Liczba, szerokość i wysokość w hierarchii dziedziczenia; Liczba klas dziedziczących specyficzne operacje; Liczba klas specyficznych od których zależą inne klasy; Liczba klas zależnych od specyficznych klas; Liczba bezpośrednich użytkowników klas lub operacji

9. 4. Ontologiczna koncepcja systemu informacyjnego

W ontologicznej teorii systemów Bunga [BUN77], [BUN79] przedstawiono model systemu, w którym elementarnym pojęciem w opisie świata rzeczywistego jest byt. Jest to teoria, która według Wanda i Webera [WAN90, s. 1283] pozwala skonstruować teoretyczne podstawy dla opisu struktury wewnętrznej systemu w przeciwieństwie do modelu "czarnej skrzynki", który przedstawia system jako pewną transformację zachodzącą między stanami wejścia i wyjścia bez wnikania w jego strukturę wewnętrzną.

Ontologia Bunga podaje matematyczne definicje obrazu świata, a występujące w nim obiekty są definiowane niezależnie od środowiska implementacji. Świat jest postrzegany w kategorii bytów, opisywanych jako istotne indywidualności. Poniżej zostaną przedstawione definicje podstawowych elementów związanych z systemem informacyjnym tworzonym w technologii obiektowej.

Definicja systemu

Niech T będzie niepustym zbiorem, to uporządkowana trójka $\sigma = \langle C, E, S \rangle$ jest systemem nad T wtedy i tylko wtedy, jeśli C i E są wzajemnie rozłącznymi podzbiórmi zbioru T , a S jest niepustym zbiorem relacji na związku C i E . System jest konceptualny, jeśli T jest zbiorem konceptualnych pozycji, jest konkretny, jeśli $T \subseteq \Theta$ jest zbiorem konkretnych encji, np. bytów.

Wand i Weber zaadoptowali ontologiczną koncepcję modelu systemu do obiektowego modelu systemu informacyjnego, budując teoretyczne podstawy formalnego opisu za pomocą standardowych pojęć matematycznych. Częściowo uzupełnili definicje Bunga, gdyż nie odnoszą się one wprost do elementów obiektowej koncepcji systemu.

Kluczowym pojęciem w ontologii są posiadane przez *istotną indywidualność własności*. Własność jest cechą nieodłączną indywidualności. Obserwator może przydzielić cechy do indywidualności, które stają się atrybutami. Wszystkie indywidualności mają skończony zbiór własności. Atrybuty indywidualności odzwierciedlają jej własności. Faktycznie, własności są rozpoznawane przez atrybuty. Indywidualność i jej własności stanowią wspólny *obiekt*. Zgodnie z powyższą definicją własności obiektu, obiekt nie jest wprost powiązany z operacjami. Dopiero w trakcie procesu przedstawiania dziedziny

aplikacji, w której identyfikowane są wystąpienia zmiennych i zachowania obiektów, następuje przypisanie operacji do obiektów, które wtedy stają się też jego własnościami.

Definicja obiektu

Obiekt jest definiowany w następujący sposób [BUN79]:

$$X = \langle x, p(x) \rangle \quad (9.4.1)$$

gdzie:

x - istotna indywidualność, może to być nazwa obiektu,

$p(x)$ - skończona kolekcja własności x .

W technologii obiektowej definicja obiektów [BOO94, s. 83] łączy obiekt wprost z jego własnościami i operacjami.

Definicje powiązania, spójności i złożoności

Intuicyjnie, powiązanie odpowiada stopniu wewnętrznej zależności między częściami projektu, a spójność wewnętrznej zgodności między częściami projektu. Z praktyki wiadomo, że projekt programu obiektowego wymaga minimalizacji powiązań i maksymalizacji spójności.

Powiązanie

Dwa obiekty są powiązane wtedy i tylko wtedy, jeśli przynajmniej jeden z nich działa na drugi. Zatem X działa na Y , jeśli historia Y jest pod wpływem X . Historia jest definiowana jako chronologicznie uporządkowane stany, które byty przechodzą w czasie [VES84].

Niech $X = \langle x, p(x) \rangle$ i $Y = \langle y, p(y) \rangle$ będą dwoma obiektami.

$$\begin{aligned} p(x) &= \{M_x\} \cup \{I_x\} \\ p(y) &= \{M_y\} \cup \{I_y\} \end{aligned} \quad (9.4.2)$$

gdzie $\{M_i\}$ jest zbiorem metod⁴², $\{I_i\}$ jest zbiorem wystąpień⁴³ zmiennych i – tego obiektu. Używając powyższej definicji powiązania, działanie przez $\{M_x\}$ na $\{M_y\}$ lub $\{I_y\}$ tworzy powiązanie takie, jak działanie przez $\{M_y\}$ na $\{M_x\}$ lub $\{I_x\}$. Zatem jawność metody jednego obiektu wywołującej metody lub wystąpienia zmiennych innego obiektu tworzy

⁴²Metoda jest operacją na obiekcie, która jest zdefiniowana jako część deklaracji klasy.

⁴³Wystąpienie zmiennej przechowuje unikalną wartość w każdym wystąpieniu klasy.

powiązanie. Skoro obiekty tej samej klasy mają takie same własności, dwie klasy są powiązane, kiedy zadeklarowane metody w jednej klasie używają metod lub wystąpień zmiennych drugiej klasy.

Spójność

Spójność to podobieństwo $\sigma()$ dwóch bytów, które jest częścią wspólną zbioru własności tych dwóch bytów

$$\sigma(X, Y) = P(x) \cap P(y) \quad (9.4.3)$$

Zgodnie z tą ogólną zasadą definicji podobieństwa w terminologii zbiorów, poziom podobieństwa metod wewnątrz obiektu może być definiowany jako część wspólna zbiorów wystąpień zmiennych, które są wykorzystywane w tych metodach. To jest rozszerzenie definicji Bunga od podobieństwa bytów do podobieństwa metod. Trzeba zaznaczyć, że wystąpienia zmiennych nie są własnościami metod, ale jest to zgodne ze stwierdzeniem, że metody obiektu są powiązane z wystąpieniami jego zmiennych. Wówczas

$$\sigma(M_1, M_2) = \{I_1\} \cap \{I_2\} \quad (9.4.4)$$

gdzie $\sigma(M_1, M_2)$ jest poziomem podobieństwa metod M_1 i M_2 , a I_i jest zbiorem wystąpień zmiennych używanych przez metodę M_i .

Poziom podobieństwa metod odnosi się zarówno do konwencjonalnego pojęcia spójności w inżynierii oprogramowania (utrzymywanie powiązanych bytów), jak i do hermetyzacji, to jest ukrycia metod i wystąpień zmiennych w klasie obiektu. Poziom podobieństwa metod może być postrzegany jako główny aspekt spójności obiektu danej klasy. Jeśli obiekt danej klasy ma różne metody wykonujące różne operacje na tym samym zbiorze wystąpień zmiennych, to klasa jest spójna. Takie widzenie spójności jest skoncentrowane na danych ukrytych wewnątrz obiektu, którego metody działają na tych danych.

Złożoność

Złożoność indywidualności to liczność układu, przy założeniu, że złożona indywidualność ma dużą liczbę własności. Używając tej definicji jako podstawy, złożoność klasy i obiektu może być definiowana jako kardynalność jego zbioru własności. Złożoność obiektu zatem przedstawia się następująco:

$$\langle x, p(x) \rangle = |p(x)| \quad (9.4.5)$$

gdzie $|p(x)|$ jest kardynalnością $p(x)$.

9. 5. Przykład obliczania punktów funkcyjnych

Analizowany przykład dotyczy badania punktów funkcyjnych przeprowadzonego dla wybranej aplikacji, wchodzącej w skład projektu systemu „Zarządzanie inwestycjami”. Do obliczeń wykorzystano znowelizowany przez członków Międzynarodowej Grupy Użytkowników Punktów Funkcyjnych (IFPUG) opis [GAR96] metody Function Point Albrechta. W tej metodzie punkty funkcyjne są używane jako znormalizowany czynnik, który umożliwia porównanie programów w taki sam sposób, jak jednostki rozmiaru innych produktów przemysłowych, np. metr, litr, kilogram. Określają one rozmiar programu, który może być wykorzystany jako czynnik wpływający między innymi na wielkość kosztów tworzenia programu. Metoda może być stosowana w odniesieniu do: tworzenia nowej aplikacji, modyfikacji istniejącej aplikacji, zainstalowanej aplikacji.

Przykład obliczeniowy przeprowadzono na poziomie tworzenia nowej aplikacji. Wykorzystano dokumentację etapu strategii, analizy oraz etapu projektowania systemu, której zawartość odpowiadała założeniom i wymaganiom metodyki strukturalnej. Obliczanie punktów funkcyjnych przeprowadzono zgodnie z zaleceniami, przechodząc przez kolejne siedem kroków metody:

- Określenie poziomu aplikacji.
- Identyfikacja granicy aplikacji.
- Identyfikacja typów danych związanych z funkcjami oraz określenie ich złożoności.
- Identyfikacja funkcji transakcyjnych i ich złożoności.
- Określenie liczby „niedopasowanych” punktów funkcyjnych.
- Określenie wielkości czynnika dopasowania na podstawie czternastu charakterystyk systemu.
- Obliczenie końcowej liczby dopasowanych punktów funkcyjnych.

9. 5. 1. Określenie poziomu aplikacji

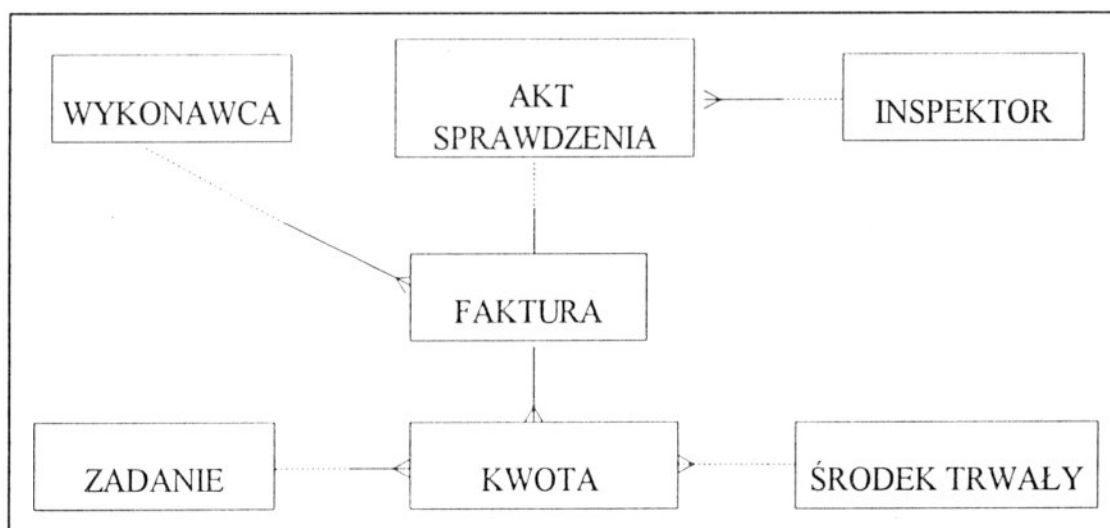
Spśród wymienionych trzech rodzajów punktów:

- Liczba punktów funkcyjnych dotyczących projektu tworzenia systemu, która może być mierzona podczas kolejnych etapów cyklu życia.
- Liczba punktów funkcyjnych dotyczących projektu modyfikacji istniejącego systemu.
- Liczba punktów funkcyjnych dotyczących zainstalowanej aplikacji.

ustalono, że badanie dotyczy pierwszego rodzaju procesu, gdyż projekt systemu „Zarządzanie inwestycjami” był realizowany w całości jako nowy system poczynszyszy od opracowania strategii dla tego przedsięwzięcia, przez analizę wymagań aż do otrzymania działającej aplikacji.

9. 5. 2. Identyfikacja granicy aplikacji

Określenie granicy aplikacji polega na ustaleniu zakresu tworzonego systemu, który wyznacza się przez identyfikację tych obszarów funkcjonalnych, które mają być przedmiotem realizowanego projektu. Polega to na identyfikacji funkcji należących do danej aplikacji i należących do jej otoczenia, w którym może też działać inna aplikacja. Wyznaczenie granic aplikacji jest konieczne dla rozróżnienia wewnętrznych i zewnętrznych typów danych, jakie występuje w tej metodzie. Prace te prowadzą do wyznaczenia systemów użytkowych i określenia kolejności ich realizacji, ale przede wszystkim pozwalają na ustalenie granic dla prowadzonego badania, którego celem jest obliczenie rozmiaru tworzonej aplikacji.



Rysunek 38. Fragment Diagramu Obiekt – Związek dla aplikacji „Realizacja inwestycji”. Źródło: [opracowanie własne].

W badanym projekcie systemu „Zarządzanie inwestycjami” wyodrębniono pięć samodzielnie działających aplikacji:

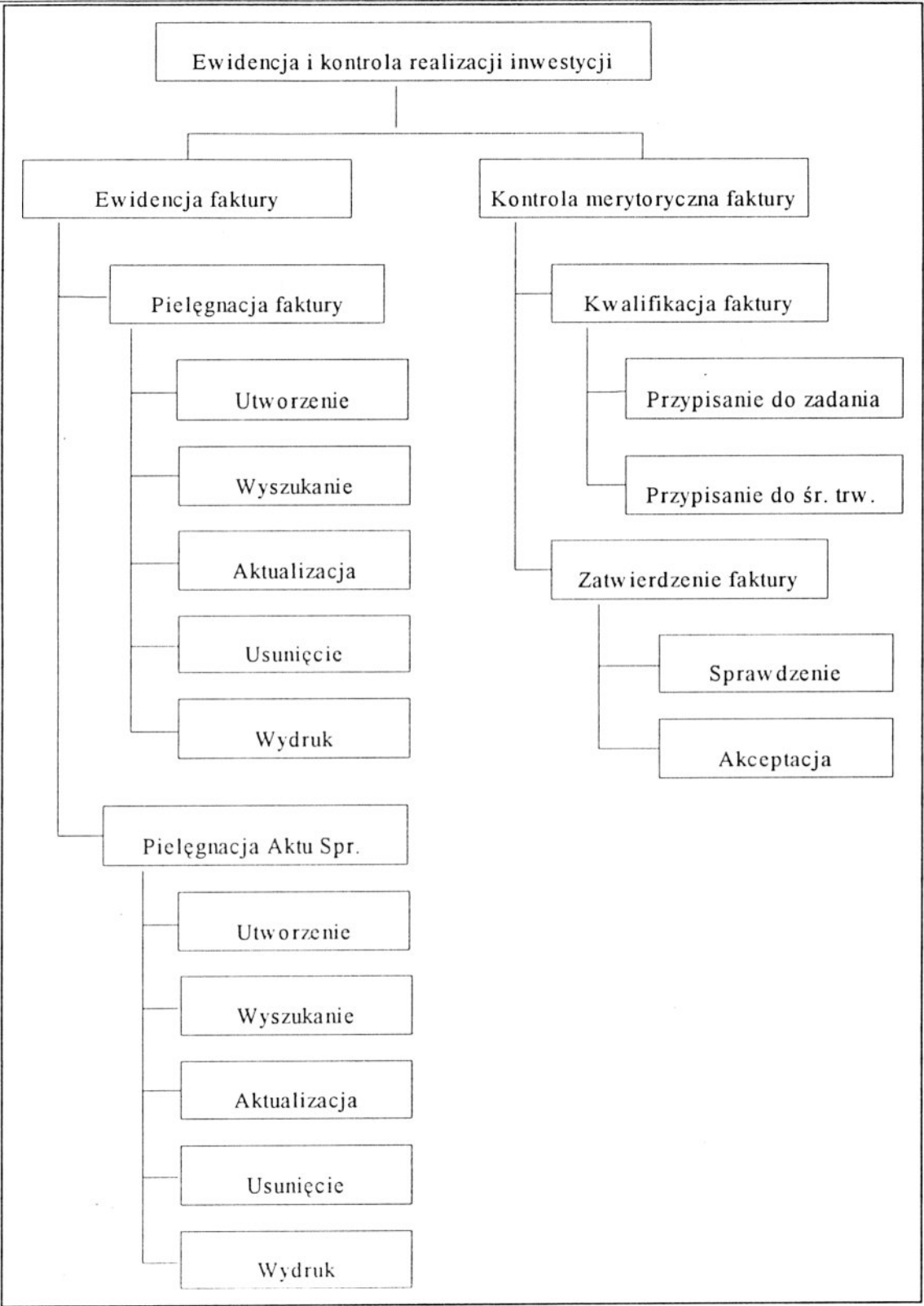
- Plan inwestycyjny
- Umowy
- Realizacja inwestycji

- Nadzór inwestycyjny

Dla każdej z aplikacji ustalono granice systemu wraz z jego otoczeniem i przedstawiono za pomocą skonstruowanego diagramu kontekstowego.

9. 5. 3. Identyfikacja typów danych oraz ustalanie ich złożoności

Spośród pięciu wyróżnionych w metodzie typów funkcyjnych dwa z nich zalicza się do typów odpowiadających danym powiązanym z tworzoną aplikacją, do których należą zarówno dane pielęgnowane za pomocą elementarnych funkcji w ustalonych granicach tej aplikacji oraz dane pielęgnowane w granicach innych aplikacji. W metodzie FP są one nazywane: wewnętrznymi logicznymi plikami (ILF), zewnętrznymi plikami interfejsu (EIF). Model procesów zbudowany na podstawie przeprowadzonej analizy wymagań dla wybranej aplikacji jest bardzo złożony, zatem na potrzeby przedstawienia sposobu obliczania punktów funkcyjnych ograniczono się do dwóch wybranych modułów: „Ewidencja faktur” i „Kontrola merytoryczna faktur”. Wyróżniono w nim funkcje, które przedstawiono na diagramie hierarchii funkcji (rys. 38).



Rysunek 39. Diagram hierarchii funkcji dla modułów „Ewidencja faktur” i „Kontrola merytoryczna faktur”.
Źródło: opracowanie własne.

Identyfikację typów danych związanych z wyżej wymienionymi funkcjami przeprowadzono na podstawie analizy Diagramu Obiekt-Związek, który przedstawia strukturę przetwarzanych informacji. Na rysunku 38 przedstawiono tylko ten fragment diagramu, który jest związany z wymienionymi funkcjami.

Tabela 50. Tablica punktów funkcyjnych dla trzech poziomów złożoności wszystkich typów składników.
Źródło: [GAR96, s. 53]

Typ składnika	Poziomy funkcji		
	Niski (L)	Średni (A)	Wysoki (H)
Wewnętrzny logiczny plik (ILF)	7	10	15
Zewnętrzny plik interfejsu (EIF)	5	7	10
Zewnętrzne wejście (EI)	3	4	6
Zewnętrzne wyjście (EO)	4	5	7
Zewnętrzne zapytanie (EQ)	3	4	6

Opis danych (tab. 51) zawiera również informację o przynależności do odpowiedniego typu danych: ILF lub EIF. Dane te pochodzą z modelu przedstawionego na rysunku 39. W tabeli tej również zamieszczono przydzieloną każdemu typowi encji odpowiednią liczbę punktów, charakteryzującą stopień jego złożoności. Została ona ustalona na podstawie matrycy złożoności (tabela 52) oraz liczby punktów, odpowiadających określonymu poziomowi złożoności (tabela 50). Zatem każdemu zidentyfikowanemu typowi ILF oraz EIF przydziela się liczbę określającą jego złożoność funkcjonalną na podstawie liczby typów elementów

- danych (z dołączeniem kluczy obcych) nazywanej DET (pole\atrybut),
- rekordów, nazywanej RET (podtyp).

Tabela 51. Wykaz danych według typów, występujących w wyróżnionym module. Źródło: [opracowanie własne].

Nazwa typu encji i jej atrybutów	Typ danych	DET	RET	Złożoność
FAKTURA Numer Data	ILF	2+1	3	L
KWOTA FAKTURY Wartość netto Wartość brutto VAT	ILF	3+3	3	L
WYKONAWCA NIP Nazwa Miejscowość Kod pocztowy Ulica Telefon Fax	EIF	7	2	L
AKT SPRAWDZENIA Numer Data wystawienia Wartość netto Wartość brutto VAT Kaucja Data sprawdzenia Data zatwierdzenia Termin zwrotu	ILF	9+5	5	L
INSPEKTOR Symbol Tytuł Nazwisko Imię	EIF	4	1	L
ZADANIE Numer ZTE Nazwa	EIF	2	3	L
ŚRODEK TRWAŁY Numer fabryczny Nazwa Typ Rok budowy Lokalizacja Symbol SW Symbol P Grupa rodzajowa Data dopuszczenia do eksploatacji	EIF	9	4	L

Dla wszystkich typów danych (dwa typy ILF, cztery typy EIF) otrzymano: $RET < 2$, a $DET < 19$. Na podstawie matrycy złożoności ustalono, że złożoność wszystkich typów jest na poziomie *L*, czyli niska.

9. 5. 4. Identyfikacja typów funkcyjnych oraz ustalanie ich złożoności

Funkcje transakcyjne mogą być typu:

Przykład obliczania punktów funkcyjnych

- Zewnętrzne wejście (EI) - dopisywanie, zmiana, usuwanie,
- Zewnętrzne wyjście (EO) – raport,
- Zewnętrzne zapytanie (EQ) – odzyskanie aktualnej wartości.

Tabela 52. Macierz do określania złożoności wewnętrznych plików logicznych ILF i zewnętrznych plików interfejsu EIF. Źródło: [GAR96, s. 51].

Liczba typów elementów rekordów	Liczba pól danych		
	1-19	20-50	51+
<2	L	L	A
2-5	L	A	H
>5	A	H	H

Złożoność funkcji ustala się na podstawie związanych z funkcją liczby typów

- elementów danych (DET),
- plików referencyjnych (FTR),

oraz macierzy złożoności dla EI.

Tabela 53. Macierz złożoności dla zewnętrznych wejść EI. Źródło: [GAR96, s. 62].

Liczba typów plików referencyjnych	Liczba pól danych		
	1-4	5-15	16+
<2	L	L	A
2	L	A	H
>2	A	H	H

Złożoność danych zaliczonych do typu zewnętrzne wejście EI oraz zewnętrzne wyjście EO jest obliczana według przedziałów liczebności przedstawionych w tabeli 53 i 54.

Tabela 54. Macierz złożoności dla zewnętrznych wyjść EO. Źródło: [GAR96, s. 69].

Typ pliku	Pola danych		
	1-5	6-19	20+
<2	L	L	A
2-3	L	A	H
>3	A	H	H

Dla funkcji zaliczonych do zewnętrznych zapytań wprowadzono dodatkowe rozróżnienie, jakiego nie było w pierwotnej wersji metody FP:

- Zapytanie wejścia, które zawiera dane sterujące wyszukiwaniem.
- Zapytanie wyjścia, które nie zawiera żadnych danych sterujących wyszukiwaniem.

Tabela 55. Macierz złożoności dla zewnętrznego zapytania wejścia EQ. Źródło: [GAR96, s. 77].

Liczba plików	Liczba pól danych		
	1-4	5-15	16+
<2	L	L	A
2	L	A	H
>2	A	H	H

W konsekwencji, oba typy zapytań posiadają odrębne macierze dla odczytu poziomu złożoności, przedstawione w tabeli 55 i 56 .

Tabela 56. Macierz złożoności dla zewnętrznego zapytania wyjścia EQ. Źródło: [GAR96, s. 77].

Typ pliku	Pola danych		
	1-5	6-19	20+
<2	L	L	A
2-3	L	A	H
>3	A	H	H

Sumaryczne zestawienie złożoności wyróżnionych typów funkcji, przeprowadzone na podstawie poszczególnych macierzy złożoności, zamieszczono w tabeli 57.

Tabela 57. Specyfikacja występujących w module typów funkcji transakcyjnych i ich złożoności [źródło: opracowanie własne].

Nazwa funkcji	Typ	DET	FTR	Złożoność
Pielęgnacja AS (Akt Sprawdzenia Inspektor)	ILF EIF EIF			
Dodaj	EI	14	5	H
Zmień	EI	14	7	H
Usuń	EI	14	1	L
Wyszukaj	EQ	14	3	H
Wydrukuj	EO	14	3	A
Pielęgnacja faktury (Faktura Kwota faktury Wykonawca Środek trwały)	ILF ILF EIF EIF			
Dodaj	EI	8	4	H
Zmień	EI	8	4	H
Usuń	EI	8	2	A
Wyszukaj	EQ	7	3	H
Wydrukuj	EO	7	3	A
Kwalifikacja faktury Do zadania	EI	1	5	A
Do środka trwałego	EI	1	5	A
Zatwierdzenie faktury Sprawdzenie	EI	1	5	A
Ostateczna akceptacja	EI	1	5	A

Funkcje typu zapytanie: „Wyszukaj fakturę” i „Wyszukaj Akt Sprawdzienia” wymagają podania danych sterujących do wyboru obu rodzajów dokumentów, zatem ich poziom złożoności został ustalony na podstawie macierzy dla zewnętrznego zapytania wejścia.

9. 5. 5. Obliczanie „niedopasowanych” punktów funkcyjnych

Na podstawie analizy złożoności wszystkich pięciu typów funkcyjnych przedstawionej w tabeli 51 i 57 oraz wzoru (3.4) obliczono liczbę niedopasowanych punktów funkcyjnych *UFP*:

$$UFP=2*7+4*5+3+5*4+4*6+2*5+2*6=103 \tag{9.5.1}$$

Ustalone poziomy złożoności dla wszystkich występujących w przykładzie typów danych i typów funkcyjnych prowadzą do ostatecznego wyboru poszukiwanych wielkości wagowych. Podstawę do wyliczenie niedopasowanych punktów funkcyjnych, wynikających z liczby elementów danego typu danych i ustalonego poziomu ich złożoności pokazano w tabeli 58.

Tabela 58. Tabela niedopasowanych punktów funkcyjnych dla modułu „Ewidencja dowodów realizacji inwestycji za pomocą faktur” (znak * wskazuje na wystąpienie danego stopnia złożoności o podanej krotności).
Źródło: [opracowanie własne].

Typ składnika	Poziomy funkcji		
	Niski (L)	Średni (A)	Wysoki (H)
Wewnętrzny logiczny plik (ILF)	2*7	10	15
Zewnętrzny plik interfejsu (EIF)	4*5	7	10
Zewnętrzne wejście (EI)	1*3	5*4	4*6
Zewnętrzne wyjście (EO)	4	2*5	7
Zewnętrzne zapytanie (EQ)	3	4	2*6

9. 5. 6. Obliczanie czynnika dopasowania

Lista (tab. 15) czternastu czynników, zwanych "głównymi charakterystykami aplikacji" (General Application Characteristics), reprezentuje grupę elementów o drugorzędym znaczeniu dla szacowanego rozmiaru systemu. Odzwierciedlają one parametry techniczne, związane z realizacją przetwarzania danych. Poziom wpływu poszczególnych elementów z tej listy jest ustalany w skali od 0 do 5 (tab. 12). Suma punktów dla wszystkich 14 charakterystyk jest określona jako całkowity stopień wpływu *TDI*.

W celu ułatwienia i ujednolicenia ustalania stopni wpływu opracowano dodatkowe tabele, które szczegółowo interpretują znaczenie skali poziomu wpływu. Ze względu na podobny sposób definiowania każdej z piętnastu charakterystyk, przykładowo, w tabeli 59 pokazano opis poszczególnych stopni wpływu tylko dla piątej charakterystyki: „tempo

przyrostu transakcji”. Pozostałe tabele można znaleźć w opracowaniu Garmusa i Herrona [GAR96, s.81-90].

Tabela 59. Opis charakterystyki „tempo przyrostu transakcji”. Źródło: [GAR96, s. 83].

Liczba stopni wpływu	Opis
0	Brak okresowego szczytu liczby transakcji.
1	Okresowy szczyt liczby transakcji.
2	Tygodniowy szczyt liczby transakcji.
3	Dzienny szczyt liczby transakcji.
4	Wysokie tempo transakcji stwierdzone przez użytkownika i wymaga analizy wykonywania zadań w fazie projektowania.
5	To samo co dla „4”, dodatkowo wymagane narzędzia analizy wykonywania zadań we wszystkich fazach.

Wartość współczynnika *DI* jest sumą wszystkich stopni wpływu dla każdej charakterystyki. W analizowanym przykładzie dla jego charakterystyki (tab. 60), po przeprowadzeniu wyliczeń, otrzymano:

$$DI=38 \tag{9.5.2}$$

Tabela 60. Charakterystyka systemu „Zarządzanie inwestycjami” na podstawie 14 czynników [źródło: opracowanie własne].

Lp	Nazwa charakterystyki	Poziom wpływu	Lp	Nazwa charakterystyki	Poziom wpływu
1	Transmisja danych	4	8	Aktualizacja on-line	5
2	Rozproszone przetwarzanie	3	9	Złożoność przetwarzania	2
3	Wydajność	2	10	Ponowne użycie	0
4	Obciążenie konfiguracji	0	11	Łatwość instalacji	1
5	Tempo przyrostu transakcji	2	12	Łatwość eksploatacji	3
6	Wprowadzenie danych on-line	5	13	Wielokrotność instalacji	1
7	Sprawność końcowego użytkownika	5	14	Łatwość wprowadzania zmian	5

Wobec powyższego oraz na podstawie wzoru (3.5) otrzymano następującą wartość czynnika dopasowania *VAF* (ang. Value Adjustment Factor), który w metodzie FP Albrechta nazywanego współczynnikiem technicznej złożoności systemu *TCF*:

$$VAF = 0,65 + 0,01 * 38 = 1,03 \tag{9.5.3}$$

9. 5. 7. Obliczanie ostatecznej liczby punktów funkcyjnych

Na podstawie końcowego wzoru (3.6) otrzymano:

$$FP = 103 * 1,03 \quad (9.5.4)$$

$$FP = 106,09 \quad (9.5.5)$$

Wielkość FP określa rozmiar przykładowej aplikacji w punktach funkcyjnych. Stanowi ona podstawę do estymacji wielkości nakładów potrzebnych do jej wykonania, a także podstawę do sterowania przebiegiem jego realizacji.

9. 6. Przykład zastosowania metody COCOMO

Oszacowanie kosztu tworzenia oprogramowania, według metody COCOMO poziomu pośredniego, wymaga przeprowadzenia wyżej opisanej analizy i obliczeń kolejno w krokach od 1 do 4 (rozdział 3.2.3). Aby zilustrować postępowanie przedstawiono obliczenia na potrzeby tworzenia oprogramowania działającego w środowisku mikroprocesorowym, z wymaganiami przetwarzania komunikacyjnego, realizowanego za pomocą nowych urządzeń transmisyjnych. Ponadto określono wysokie wymagania dotyczące wydajności i niezawodności pracy systemu, a także oczekiwania planowej realizacji tego przedsięwzięcia. Rozmiar szacowanego produktu w przybliżeniu oceniono na 10 0000 instrukcji źródłowych.

Kroki postępowania

- 1) Zgodnie z zarysowanym profilem oprogramowania ustalono, na podstawie charakterystyki (tab. 11), że należy zaliczyć program do trybu "wbudowanego". Zatem nominalny nakład tworzenia (w osobomiesiącach), po podstawieniu podanej wielkości do trzeciego równania (tab. 14) będzie wynosił:

$$2,8(10)^{1,20} = 44MM. \quad (9.6.1)$$

- 2) Na wielkość szacowanego nakładu istotny wpływ mają czynniki nie tylko związane z samym produktem, ale także ze środowiskiem jego powstawania. Według wyspecyfikowanych wymagań przykładowego oprogramowania i danych z tabel 15 i 16 obliczono sumaryczny czynnik nominalnego nakładu (tab. 61).

Przykład zastosowania metody COCOMO

Tabela 61. Obliczone sterowniki kosztów według metody COCOMO dla przykładowego oprogramowania.
Źródło: [BOE84, s.13].

STEROWNIK KOSZTU	OPIS	KATEGORIA	WSPÓŁCZYNNIK NAKŁADU
RELY	Finansowe konsekwencje awarii oprogramowania	Wysoki	1,15
DATA	20 000 bajtów	Niski	0,94
CPLX	Przetwarzanie komunikacyjne	Bardzo wysoki	1,30
TIME	Potrzeba korzystania z 70% dostępnego czasu	Wysoki	1,11
STOR	45K z 64K pamięci (70%)	Wysoki	1,06
VIRT	Na bazie komercyjnego sprzętu mikroprocesorowego	Nominalny	1,00
TURN	Przeciętnie dwie godziny	Nominalny	1,00
ACAP	Dobry i doświadczony	Wysoki	0,86
AEXP	Trzy lata	Nominalny	1,00
PCAP	Dobry i doświadczony	Wysoki	0,86
VEXP	Sześć miesięcy	Niski	1,10
LEXP	Dwanaście miesięcy	Nominalny	1,00
MODP	Większość technik stosowanych ponad rok	Wysoki	0,91
TOOL	Poziom narzędzi podstawowych	Niski	1,10
SCED	Dziewięć miesięcy	Nominalny	1,00
Sumaryczny czynnik wnioskowania			1,35

3) Przybliżony nakład tworzenia oblicza się jako iloczyn nominalnego nakładu tworzenia (wynik kroku 1) i sumarycznego czynnika wnioskowania (tab. 61). Zatem przybliżony nakład dla przykładowego projektu wynosi (w osobomiesiącach):

$$(44\text{ MM}) * (1,35) = 59\text{ MM}. \tag{9.6.2}$$

4) Korzystając z trzeciego równania dla planów (tabela 14), obliczono na potrzeby budowy harmonogramu przewidywany czas tworzenia projektu (tryb "wbudowany"):

$$T_{DEV} = 2,5(59)^{0,32} = 9\text{ MM}. \tag{9.6.3}$$

Sterowanie współczynnikami nakładów pozwala także na przeprowadzanie analiz wrażliwości i analiz porównawczych. Można prześledzić wpływ tych współczynników na estymowany nakład, dokonując na przykład redukcji. I tak, korzystając z danych tabeli 16, redukcja atrybutu *TURN* z wartości wskaźnika 1,00 do 0,87 spowoduje:

$$(59\text{ MM}) * (0,87) = 51\text{ MM} \tag{9.6.4}$$

zmniejszenie oszacowanego wcześniej nakładu (9.6.2).

9. 7. Podejścia do wykonania analizy obiektowej

Spośród wielu metod tworzenia systemów informatycznych w technologii obiektowej można wyróżnić dwa zasadnicze nurty, jeden to podejście ukierunkowane na obiekty, z różniącymi się nieco notacjami, a drugi to podejście również ukierunkowane na obiekty, ale z podkreśleniem znaczenia analizy procesów, szczególnie na etapie analizy wymagań. Niezależnie od podejścia, po przeprowadzeniu studium wykonalności, następuje etap analizy wymagań użytkownika wobec przyszłego systemu. Z punktu widzenia potrzeby oszacowania kosztów planowanego przedsięwzięcia zachodzi konieczność skupienia się nad tym etapem, który w odróżnieniu od późniejszych prac projektowych, jest nazywany wczesnym etapem tworzenia oprogramowania. Od etapu analizy wymaga się określenia możliwości wykonania produktu informatycznego, tzn. oszacowania zarówno kosztów jak i terminów realizacji tego produktu. na podstawie zdecydowanego zakresu systemu i analizy wymagań użytkownika. Etap ten stanowi ważny i czasochłonny element całego cyklu, decydujący w dużej mierze o powodzeniu tego przedsięwzięcia. Ustalenie wielkości zasobów finansowych i czasowych nie jest możliwe bez zapoznania się z oczekiwaniami zleceniodawcy, a także ogólnego zrozumienia złożoności dziedziny problemu. Ta faza prac nazywana jest opisem wymagań użytkownika, otrzymywanym w wyniku szczegółowej analizy⁴⁴. W metodykach obiektowych istnieje kilka podejść do przeprowadzenia etapu analizy wymagań. Różnią się one przede wszystkim stosowanym podejściem do identyfikacji klas i obiektów, jako kluczowych elementów występujących w technologii obiektowej.

Przy podejściu, nazywanym klasycznym, wyprowadza się kandydujące⁴⁵ klasy i obiekty z analizy zgodnie z zasadami klasycznej klasyfikacji⁴⁶, a źródłem zdobywania wiedzy o badanej rzeczywistości [SHL88] są : rzeczywiste przedmioty, pełnione role, zdarzenia i interakcje.

W metodyce Coad i Yourdona [COA91] źródłem potencjalnych klas i obiektów są struktury, inne systemy, urządzenia, historyczne zdarzenia, pełnione role, lokalizacje i jednostki organizacyjne.

⁴⁴ Etap szczegółowej analizy wymagań jest najbardziej pracochłonny i nie przystępuje się do niego bez ustalenia wstępnych warunków realizacji przedsięwzięcia, które są formułowane w sposób ogólny i bardzo przybliżony.

⁴⁵ Pojęcie kandydujących klas i obiektów oznacza, że są to potencjalne elementy modelu obiektowego w pierwszej koncepcji dokonanej klasyfikacji.

⁴⁶ W klasyfikacji klasycznej za kryterium wyróżniającym zbiory obiektów używa się sprawdzenia polegającego na zadawaniu pytania: czy poszczególne własności są obecne czy nieobecne?

Podejście ukierunkowane na badanie dynamiki obiektów, nazywane analizą zachowań, w którym obiekty są grupowane ze względu na podobieństwo zachowań znacznie się różni od podejścia klasycznego. Jednym z przykładów takiego podejścia jest identyfikacja klas i obiektów na podstawie funkcji, które powinien realizować system. Wymaga się najpierw zrozumienia działania poszczególnych części systemu, a następnie dochodzenia kto i co inicjuje te zachowania oraz kto i co w nich uczestniczy [RUB92].

Trzecim podejściem jest analiza dziedziny, stosowana przeważnie w tworzeniu pojedynczej, specyficznej aplikacji. Polega ono na identyfikowaniu obiektów, operacji i relacji istotnych dla rozważanej dziedziny, ale dotyczy również klas i obiektów, które są wspólne dla wszystkich aplikacji związanych z tą dziedziną. Takie podejście jest stosowane w metodyce OMT i metodyce Boocha..

Kolejnym podejściem, które łączy trzy poprzednie, jest analiza przypadków użycia po raz pierwszy sformalizowana przez Jacobsona w postaci metodyki Object-Oriented Software Engineering [JAC92]. Jako podstawowy element w podejściu Jacobsona pojawia się, nie występujący w żadnym innym, model „przypadków użycia” (ang. Model Use Case). Stanowi on podstawowe źródło wiedzy o zakresie funkcjonalnym mającego powstać systemu, a identyfikacja klas i obiektów, która odbywa się w pierwszym kroku badań przy innych metodach, następuje dopiero po ukończeniu budowy tego modelu. Szczegółowy opis tego podejścia zamieszczono poniżej.

9. 7. 1. Etap analizy w metodyce Jacobsona

W metodyce Jacobsona OOSE (Object-Oriented Software Engineering [JAC92]) z etapu analizy wyraźnie wyróżnia się fazę ogólną⁴⁷, podczas której powstaje model wymagań oraz fazę analizy szczegółowej, której wynikiem jest model analizy. Celem budowy modelu wymagań jest ustalenie zakresu systemu i definicja funkcjonalności, która ma być przez ten system oferowana. Należy podkreślić, że jest to model budowany z punktu widzenia użytkownika. Natomiast celem budowy modelu analizy jest otrzymanie specyfikacji wszystkich logicznych obiektów systemu i przedstawienie wszystkich możliwych powiązań istniejących między tymi obiektami.

Model wymagań jest centralnym miejscem dla całego procesu tworzenia systemu. Powstaje on w wyniku rozmów i wywiadów z użytkownikiem, jest łatwy do zrozumienia i

⁴⁷ Przymiotnik „ogólny” w odniesieniu do fazy etapu analizy został wprowadzony przez autora pracy przez analogię do metodyki strukturalnej w celu podkreślenia roli tych dwóch faz analizy, gdyż obie składają się na

sformułowania i według metodyki Jacobsona jest pierwszym z tworzonych modeli. Wymagania w nim zawarte są podstawą do otrzymania najpierw sformalizowanej postaci modelu analizy, który z kolei jest podstawą do realizacji modelu projektowego. Z projektu powstaje model implementacji, który po przetestowaniu, staje się pierwszą wersją produktu.

Dla organizacji procesu tworzenia w metodyce Jacobsona podstawową zasadą jest przyjęcie podejścia przyrostowego (rys. 37) z jak najmniejszą liczbą iteracji [JAC92, s.457]. Jednak nie oznacza to podziału systemu na części od samego początku tego procesu. W pierwszej fazie analizy powstaje model wymagań dla całego systemu, w celu zidentyfikowania wszystkich przypadków użycia dla tej aplikacji. Dopiero po otrzymaniu tego modelu wymagań podejmowane są decyzje o realizacji kolejnych przyrostów aplikacji. Przyrosty są wówczas częściami tworzonej aplikacji, które powstają w wyniku klasyfikacji przypadków użycia w zależności od ustalonych priorytetów. Zaś końcowe testowanie całego systemu może być przeprowadzone, jeżeli zakończone będzie testowanie działania wszystkich przypadków użycia, zawartych w zbudowanym modelu wymagań. Model wymagań składa się z trzech części: modelu przypadków użycia (rys. 40), model dziedziny problemu i opisu interfejsu użytkownika. Każdy przypadek użycia jest kompletnym przebiegiem zdarzeń w systemie, widzianym z perspektywy użytkownika i przez niego inicjowanym, który odzwierciedla interakcje między tym użytkownikiem a systemem. Ze względu na możliwość różnych wyników opisywanych zachowań, ciąg zdarzeń zawarty w przypadku użycia może mieć kilka alternatywnych dróg (wyrażonej w języku specyfikacji za pomocą bloku warunkowego If ... Then... Else...). Model przypadków użycia pozwala skupić się na analizie wybranych obszarów funkcjonalnych systemu, co jest wykorzystywane w przyrostowym podejściu do dalszej budowy całego systemu. Zatem wybrane grupy przypadków użycia mogą być niezależnie budowane, a następnie połączone w kompletny produkt.

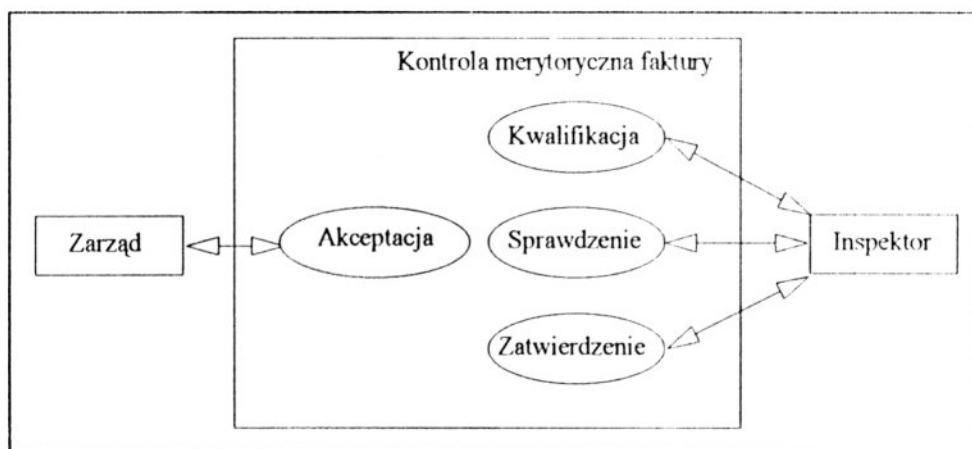
9. 7. 2. Model przypadków użycia

W modelu wymagań wprowadzono trzy podstawowe pojęcia: aktor, granica systemu, przypadek użycia. W reakcji systemu z otoczeniem użytkownik reprezentuje jedną z możliwych ról, jakie mogą wystąpić. Role, w konwencji tego diagramu, są przedstawiane za

pomocą aktora⁴⁸, który stanowi klasę, a użytkownik jest wystąpieniem tej klasy. W rzeczywistości ta sama osoba może występować jako wystąpienia różnych aktorów. Spośród wielu różnych ról, aktor może przedstawiać również inny system komunikujący się z danym systemem, albo jakieś urządzenie. Najistotniejszą cechą dla aktora jest fakt bycia elementem zewnętrznym wobec systemu, który wynika z wprowadzonego pojęcia granicy systemu, czyli linii oddzielającej aktorów od przypadków użycia.

Przypadki użycia (rys. 40) są identyfikowane przez aktora, na rysunku aktorami są: zarząd i inspektor. Każdy, zakończony bieg zdarzeń, zainicjowanych przez aktora nazywany jest przypadkiem użycia, np. potwierdzenie lotu wykonane przez pilota. Każde wykonanie przypadku użycia może być postrzegane w terminologii obiektowej jako wystąpienie przypadku użycia. Kiedy użytkownik wprowadza bodziec, to wystąpienie przypadku użycia wykonuje rozpoczęcie transakcji należącej do tego przypadku użycia. Transakcja może składać się z różnych czynności i jest zakończona wówczas, kiedy wystąpienie przypadku użycia znów otrzyma bodziec od wystąpienia aktora. Wystąpienie przypadku użycia istnieje tak długo, jak długo działa przypadek użycia.

Wystąpienia przypadku użycia, podobnie jak inne wystąpienia w systemach obiektowych należą do specyficznej klasy i stanowią klasy przypadków użycia. Zbiór wszystkich klas (opisów) przypadków użycia specyfikuje kompletną funkcjonalność systemu.



Rysunek 40. Przykład modelu przypadków użycia. Źródło: [opracowanie własne].

Jeżeli zachowanie systemu się zmienia, to przypadek użycia musi być przemodelowany, gdyż cała architektura systemu powinna być podporządkowana aktualnym potrzebom użytkownika.

⁴⁸ Wprowadzono podział na aktorów pierwszorzędnych, którzy korzystają bezpośrednio z systemu i drugorzędnych, którzy działają poprzez pierwszorzędnych aktorów. Identyfikacja przypadków użycia zaczyna

Związki między przypadkami użycia

Między przypadkami użycia, podobnie jak i obiektami, mogą występować pewne powiązania. Są one klasyfikowane jako:

- związki rozszerzenia,
- związki użycia.

Związek rozszerzenia występuje wtedy, gdy istnieje możliwość wstawienia jednego opisu przypadku użycia do innego. Pierwszy z nich wówczas jest nazywany przypadkiem użycia rozszerzającym a drugi rozszerzonym. Związek ten został wprowadzony na podobieństwo związków asocjacji między klasami obiektów. Wymaga się by rozszerzony przypadek użycia był całkowicie niezależny od wstawionego opisu, dlatego też, przypadek użycia, do którego nowa część funkcjonalności jest wstawiana, powinien być zakończonym ciągiem zdarzeń. Związek rozszerzenia jest podstawowym sposobem na rozszerzanie funkcjonalności przypadków użycia.

Drugi typ związku między przypadkami użycia wyróżnia się przez identyfikację podobnych części w samym modelu. Dla odkrytych podobnych części tworzy się abstrakcyjne przypadki użycia. Wówczas opis jest wykonywany tylko dla tej abstrakcyjnej części, a nie dla każdej z osobna. Każda z części podobnych, w odróżnieniu od abstrakcyjnego przypadku użycia, jest nazywana konkretnym przypadkiem użycia. Może się zdarzyć, że konkretny przypadek użycia będzie miał swój własny opis, a dalszym jego ciągiem będzie opis zawarty w abstrakcyjnym przypadku użycia. Relacja między abstrakcyjnym przypadkiem użycia a konkretnym przypadkiem użycia przypomina mechanizm dziedziczenia między klasami obiektów. Doszukując się dalszych analogii związku użycia z dziedziczeniem można stwierdzić też istnienie wielokrotnego związku użycia, gdyż okazuje się, że jeden konkretny przypadek użycia może korzystać z kilku abstrakcyjnych przypadków użycia.

Technika odkrywania abstrakcyjnych przypadków użycia pociąga za sobą również identyfikację abstrakcyjnych aktorów. Dzieje się tak wówczas, kiedy różni aktorzy spełniają podobne role, wobec tego można utworzyć dla nich wspólnego - abstrakcyjnego aktora.

Rola modelu przypadków użycia

- Wspomaga identyfikację obiektów przez bezpośrednie odwoływanie się do dziedziny problemu.

- Wspomaga tworzenie opisu interfejsu użytkownika przez symulację zachowań systemu.
- Stanowi formę kontroli dla wszystkich innych modeli.
- Końcowy model stanowi narzędzie do testowania integracji systemu.
- Wspomaga tworzenie instrukcji obsługi systemu oraz dokumentacji technicznej.

9. 7. 3. Model obiektów dziedzicznych

Model obiektów, jako część składowa modelu wymagań, wspomaga tworzenie listy rzeczowników, które wystąpią w opisie przypadków użycia i stanowi formę podręcznego słownika przy specyfikacji funkcjonalności systemu. W wielu innych metodach technologii obiektowej, np. Coad i Yourdona, Boocha, OMT na podstawie tego modelu przeprowadza się identyfikację obiektów, a następnie konstruuje się model klas. W metodyce OOSE w tym celu jest tworzony drugi model szczegółowy, nazywany modelem analizy, a model dziedziny problemu jest traktowany jako pomoc w zrozumieniu zachowania systemu związanego z występującymi w nim obiektami oraz jako źródło wspólnej terminologii. Okazuje się, że wiele obiektów z dziedzicznego modelu staje się też obiektami w modelu analizy [JAC92, s. 168]. Za wystarczający poziom do budowy tego modelu uważa się: identyfikację obiektów, atrybutów i statycznych powiązań (statyczne odwołania między tymi obiektami). Identyfikację dziedziczenia, dynamicznych powiązań i operacji zaleca się odłożyć do fazy szczegółowej analizy. Autor proponuje również dziedziczny model obiektów jako łącznik między modelem organizacji a procesem tworzenia systemu. Model organizacji przedstawia jej potrzeby informacyjne i funkcjonalne z punktu widzenia danej organizacji, a jednym z diagramów przedstawiających te potrzeby jest właśnie diagram obiektów sporządzany zgodnie z takimi samymi zasadami, jak dziedziczny model obiektów.

9. 7. 4. Opis interfejsów użytkownika

Opis interfejsu specyfikuje szczegółowo wymagania użytkownika dotyczące wykonywania przypadków użycia (co użytkownik będzie widział na ekranie podczas jego wykonywania) i stanowi integralną część opisów tych przypadków użycia. Opisy te odzwierciedlają logiczny sposób widzenia systemu przez użytkownika, podobnie jak i dziedziczny model obiektów. Przykładem interfejsu jest postać komunikatów, alarmów, przycisków, a także protokół komunikacyjny z innym systemem.

9. 8. Obiekty repozytorium SELECT'a

Tabela 62. Opis obiektów występujących w repozytorium CASE pakietu SELECT Perspectiv. Źródło: [SEL96]

Nazwa obiektu	Opis obiektu w repozytorium
Projects	Obiekt „root” dla systemu SELECT, który stanowi kolekcję wszystkich projektów w systemie.
Project	Obiekt, który stanowi zbiornik dla słownika i kolekcji diagramów.
Dictionary	Obiekt, który stanowi zbiornik dla wszystkich pozycji słownika., jest obiektem iteracyjnym, może być przeglądany za pomocą filtrowania poszczególnych podzbiorów pozycji poprzez ich nazwę lub nazwę i typ, np. wszystkie elementy o nazwie "Oblicz" lub wszystkie operacje o nazwie "Oblicz".
Actor	Aktywny obiekt, który steruje grafem przepływu danych poprzez wytwarzanie lub pobieranie wartości..
Association	Opisuje grupy połączeń o wspólnej strukturze i semantyce pomiędzy wystąpieniami dwóch Class.
Attribute	Nazwana własność klasy, opisująca wartość danej utrzymywanej przez każdy obiekt klasy.
Class	Opis grupy obiektów z podobnymi własnościami, wspólnym zachowaniem, wspólnymi relacjami i wspólną semantyką. Podzbiory obiektu klasa stanowią np. Association, Attribute, które są iteracyjnymi obiektami.
Column	Atrybut tabeli, którego dane mają identyczny format i są zapamiętane dla każdego rekordu tej tabeli.
Corporate View	Przedstawia perspektywę zdefiniowaną na klasie organizacji, która występuje też w innych modelach aplikacji.
Data Store	Obiekt pasywny, który przechowuje dane do późniejszego wykorzystania.
Dependency	Przedstawia zależność przepływu pomiędzy dwoma podsystemami (który z nich przekazuje, a który otrzymuje).
Event	Zdarzenie – musi być rozpoznane i wymaga podjęcia reakcji przez system.
Generalization	Relacja, która wskazuje, że charakterystyki klasy będącej początkiem w takim związku są częścią definicji klasy końca.
Increment	Część projektu, która może stanowić samodzielnie działający moduł, nazywany przyrostem. Jego struktura zwykle odpowiada jednemu lub więcej „przypadkom użycia”.
Index	Kolekcja kolumn należących do tabeli, używany do promowania efektywnych ścieżek dostępu do wybranych rekordów z tabeli.
Interface Class	Opis grupy obiektów z podobnymi własnościami, wspólnymi zachowaniami, relacjami i semantyką, składających się na model klas interfejsu.
Operation	Przedstawia funkcję/transformację która może być zastosowana do obiektów danej klasy.
Process	Działanie, które przekształca dane.
Relationship	Powiązanie pomiędzy dwoma tabelami w modelu klas pamięci.
Role	Jest opisem roli klasy w związku.

Specialization	Relacja, która wskazuje, że charakterystyki klasy końca są częścią definicji klasy początku.
State	Indywidualny stan klasy lub operacji.
Subsystem	Logiczna część modelowanego systemu, obejmująca dalsze podsystemy definiujące ten system.
Table	Opis grupy obiektów z podobnymi własnościami, wspólnymi zachowaniami, relacjami i semantyką, składających się na model klas pamięci.
Ternary Association	Związek pomiędzy trzema lub więcej wystąpieniami klas.
Ternary Role	Mechanizm dla związków ternarnych, które mają specyficzną nazwę roli dla każdej związanej klasy.
Use Case	Przypadek użycia, inaczej scenariusz pokazujący jeden ze sposobów użycia systemu.
Diagrams	Obiekt - zbiornik dla diagramów definiowanych w projekcie. Jest iteracyjny, może być filtrowany względem podzbiorów kolekcji poszczególnych typów, np. diagram klas, diagram klas interfejsu.

9. 9. Terminy stosowane w metodyce OOSE i SELECT

Tabela 63. Porównanie pojęć występujących w metodyce OOSE z metodyką SELECT Perspective. Źródło: [opracowanie własne] .

OOSE	SELECT Perspectiv
Klasa	Klasa
Obiekt	Obiekt
Dziedziczenie	Struktura gen-spec
Znajomość	Połączenie
Komunikacja	Połączenie typu komunikat
Bodziec	Komunikat
Operacja	Operacja
Atrybut	Atrybut
Aktor	Aktor
Przypadek użycia	Przypadek użycia
Podsystem	Podsystem
Blok	Blok

9. 10. Czynniki produktywności w modelu Walstona - Felixa

Tabela 64. Wykaz czynników produktywności występujących w modelu Walstona – Felixa. Źródło: [WAL77]

Lp.	Nazwa czynnika produktywności
1	Złożoność interfejsu klienta
2	Udział użytkownika w definiowaniu wymagań
3	Zmiany projektu przez klienta
4	Doświadczenie klienta w obszarze zastosowania projektu
5	Doświadczenie i kwalifikacje całego personelu
6	Procent programistów uczestniczących w projektowaniu
7	Znajomość działania komputerów
8	Doświadczenie w programowaniu
9	Doświadczenie w tworzeniu aplikacji o podobnych lub większym rozmiarze i złożoności
10	Stosunek liczbowy średniego rozmiaru zespołu do czasu trwania projektu (liczba osób na miesiąc)
11	Sprzęt do tworzenia
12	Dostęp do komputera na życzenie
13	Dostęp do komputera zamknięty
14	Zagwarantowane środowisko dla komputera, przynajmniej 25 % programów i danych
15	Stosowanie programowania strukturalnego
16	Stosowanie inspekcji projektu i kodu
17	Stosowanie podejścia top – down
18	Utrzymywanie roli szefa programistów
19	Całkowita złożoność tworzonego kodu
20	Złożoność przetwarzania w aplikacji
21	Złożoność przepływów
22	Ogólne ograniczenia dla projektu programu
23	Projektowe ograniczenia pamięci głównej
24	Projektowe ograniczenia czasu
25	Kod działający w czasie rzeczywistym, w trybie interakcyjnym lub działający pod czasowym ograniczeniem
26	Procent kodu wytworzonego
27	Kod zaklasyfikowany jako aplikacja niematematyczna, program formatowania wejście/wyjście
28	Liczba klas pozycji w bazie danych przypadających na 1000 linii kodu
29	Liczba stron wykonanej dokumentacji przypadającej na 1000 linii kodu

9. 11. Atrybuty nakładów w modelu Bailey' a – Basili'a

Tabela 65. Wykaz atrybutów nakładu występujących w modelu Bailey'a – Basili'a. Źródło: [PFL91, s. 75]

Lp.	Nazwa atrybutu nakładów
1	Metodologia (METH)
2	Schematy drzewa
3	Podejście top-down
4	Dokumentacja formalna
5	Szef zespołu programistów
6	Szkolenie formalne
7	Plany testu formalnego
8	Formalizmy projektowe
9	Czytanie kodu
10	Prospekty tworzenia jednostki
11	Złożoność (CPLX)
12	Złożoność interfejsu klienta
13	Złożoność aplikacji
14	Złożoność przepływu
15	Złożoność wewnętrznej komunikacji
16	Złożoność bazy danych
17	Złożoność zewnętrznej komunikacji
18	Zmiany projektu wnoszone przez klienta
19	Doświadczenie (EXP)
20	Kwalifikacje programisty
21	Doświadczenie maszynowe programisty
22	Doświadczenie programowe programisty
23	Doświadczenie aplikacyjne programisty
24	Doświadczenie zespołu

10. SPIS TABEL

1. Empiryczny system relacyjny i odpowiadający mu formalny system relacyjny	11
2. Wartości miar stopnia trudności w skali porządkowej	17
3. Stopnie trudności poszczególnych modułów w systemie X	18
4. Stopnie trudności poszczególnych modułów w systemie Y	18
5. Typy skal, dopuszczalne przekształcenia i statystyki	18
6. Przykłady atrybutów dla wyróżnionych klas obiektów	20
7. Przykłady miar bezpośrednich i pośrednich stosowanych w inżynierii oprogramowania	21
8. Wyniki porównania liczby linii kodu	33
9. Klasyfikacja metod szacowania kosztów	52
10. Wagi typów składników według poziomów złożoności	55
11. Lista charakterystycznych czynników aplikacji	56
12. Skala wpływu czynników	56
13. Tryby tworzenia oprogramowania w metodzie COCOMO	61
14. Równania nakładu i planowanego czasu w metodzie COCOMO	62
15. Sterowniki kosztu w metodzie COCOMO	63
16. Współczynniki nakładu tworzenia oprogramowania dla odpowiednich sterowników w pośredniej metodzie COCOCMO	64
17. Parametry dla modelu COPMO	70
18. Charakterystyki rozmiaru modelu analizy wytypowane do zestawu potencjalnych zmiennych objaśniających w modelu nakładów	98
19. Wykaz projektów	99
20. Wykaz zmiennych i ich skrótów	100
21. Dane z obserwacji dotyczące czternastu projektów wykonanych przez zespoły A, B i C	100
22. Analiza głębokości drzew dziedziczenia	103
23. Wartości odchylenia standardowego i średniej zmiennej X_i oraz współczynnika zmienności v_i	107
24. Macierz R współczynników korelacji liniowej Pearsona między nakładem i atrybutami rozmiaru oraz odpowiadających im poziomów istotności	108

25. Wyniki testu Kołmogorowa – Smirnowa ($K - S$) dla rozkładu zmiennej objaśnianej i zmiennych objaśniających, część I	111
26. Wyniki testu Kołmogorowa – Smirnowa ($K - S$) dla rozkładu zmiennej objaśnianej i zmiennych objaśniających, część II	112
27. Przekształcenie logarytmiczne wartości nakładów dla czternastu projektów	118
28. Współczynniki korelacji liniowej oraz poziomy istotności między atrybutami modelu programu oraz nakładem, którego wartości zostały zlogarytmowane	121
29. Wykaz zmiennych objaśniających, uporządkowany według wartości bezwzględnej malejącego współczynnika korelacji liniowej ze zmienną objaśnianą ..	122
30. Macierz odległości klasyfikowanych atrybutów rozmiaru i logarytmu nakładów, zbudowana według metryki Euklidesa	124
31. Wartości statystyk t dla zmiennych objaśniających w trzech krokach metody selekcji stopniowej	127
32. Podzbiory zmiennych objaśniających otrzymane metodą analizy współczynników korelacji, taksonomii wrocławskiej oraz selekcji stopniowej	128
33. Współczynniki modeli estymacji nieliniowej otrzymanych metodą najmniejszych kwadratów dla rozwiązań A i B	129
34. Statystyki opisowe dla modelu liniowego i nieliniowego rozwiązań A i B	131
35. Wartości parametrów oceny modeli w wariancie A i B	132
36. Wyniki analizy wariancji dla modelu nieliniowego w wariancie A i B	134
37. Współczynniki modelu estymacji nieliniowej otrzymanych metodą najmniejszych kwadratów dla wariantu C	135
38. Wartości parametrów oceny modelu w wariancie C	136
39. Statystyki estymatora błędów dla modelu odpowiadającego wariantom A , B i C	137
40. Wartości obserwowane i przewidywane logarytmu nakładów na podstawie modelu w wariancie A oraz wartości residuów i standaryzowanych residuów	138
41. Wartości obserwowane i przewidywane logarytmu nakładów na podstawie modelu w wariancie B oraz wartości residuów i standaryzowanych residuów	139
42. Wartości obserwowane i przewidywane logarytmu nakładów na podstawie modelu w wariancie C oraz wartości residuów i standaryzowanych residuów	140
43. Współczynniki modelu estymacji nieliniowej otrzymane ważoną metodą najmniejszych kwadratów dla modelu w wariancie B	143
44. Wartości statystyki t oraz d wariantów A , B i C	145
45. Porównanie wartości standardowych błędów estymacji	150

46. Metodyki obiektowe	160
47. Porównanie pojęć występujących w metodyce OOSE z innymi metodykami technologii obiektowej	178
48. Zestaw metryk Lorenza proponowanych dla projektu programu	179
49. Metryki procesu i programu Jacobsona	179
50. Tablica punktów funkcyjnych dla trzech poziomów złożoności wszystkich typów składników	188
51. Wykaz danych według typów, występujących w wyróżnionym module	189
52. Macierz do określania złożoności wewnętrznych plików logicznych ILF i zewnętrznych plików interfejsu EIF	190
53. Macierz złożoności dla zewnętrznych wejść EI	190
54. Macierz złożoności dla zewnętrznych wyjść EO	190
55. Macierz złożoności dla zewnętrznego zapytania wejścia EQ	191
56. Macierz złożoności dla zewnętrznego zapytania wyjścia EQ	191
57. Specyfikacja występujących w module typów funkcji transakcyjnych i ich złożoności	191
58. Tabela niedopasowanych punktów funkcyjnych dla modułu „Ewidencja dowodów realizacji inwestycji za pomocą faktur”	192
59. Opis charakterystyki „tempo przyrostu transakcji”	193
60. Charakterystyka systemu „Zarządzanie inwestycjami” na podstawie 14 czynników	193
61. Obliczone sterowniki kosztów według metody COCOMO dla przykładowego oprogramowania	196
62. Opis obiektów występujących w repozytorium CASE pakietu SELECT Perspectiv ...	203
63. Porównanie pojęć występujących w metodyce OOSE z metodyką SELECT Perspective	205
64. Wykaz czynników produktywności występujących w modelu Walstona – Felixa	206
65. Wykaz atrybutów nakładu występujących w modelu Bailey’a – Basili’a	207

11. SPIS RYSUNKÓW

1. Graf przepływu sterowania dla procedury z przykładu	27
2. Przykład modelu klas i wartości metryk <i>RFC</i> i <i>CBO</i>	35
3. Hierarchia klas w diagramie klas	36
4. Porównanie produktywności wydziału W1 i W2 przy wykonywaniu programu bez ponownego użycia - przypadek <i>A</i> oraz z ponownym użyciem – przypadek <i>B</i>	40
5. Alternatywne nakłady projektu	68
6. Hierarchiczny układ poziomów abstrakcji występujących w technologii obiektowej ...	82
7. Projekt <i>A</i> bez stosowania mechanizmu dziedziczenia	86
8. Projekt <i>B</i> , który przedstawia modyfikację projektu <i>A</i> poprzez wprowadzenie mechanizmu dziedziczenia	89
9. Meta model repozytorium pakietu SELECT	92
10. Histogram przypadków użycia	101
11. Histogram obiektów	101
12. Histogram specjalizacji	102
13. Histogram drzew dziedziczenia	103
14. Procentowy udział projektów dla ustalonych głębokości drzew dziedziczenia	104
15. Histogram asocjacji	104
16. Histogram	
17. Histogram wielkości nakładów dla wszystkich projektów	105
18. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i asocjacje	113
19. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i diagramy przypadków użycia	113
20. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i obiekty	114
21. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i atrybuty	115
22. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i operacje	115
23. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami:	

liniową, logarytmiczną i wykładniczą - nakłady i specjalizacje	116
24. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i przypadki użycia	117
25. Porównanie graficznych przybliżeń wykresu danych z obserwacji funkcjami: liniową, logarytmiczną i wykładniczą - nakłady i wskaźnik wewnętrznego ponownego użycia	118
26. Graf połączeń zredukowanego zbioru zmiennych objaśniających otrzymany przy zastosowaniu metody analizy współczynników korelacji	123
27. Graf połączeń zmiennych objaśniających przy zastosowaniu metody taksonomii wrocławskiej	125
28. Wykres zgodności rozkładu standaryzowanego estymatora błędów z rozkładem normalnym modelu w wariancie A	138
29. Wykres zgodności rozkładu standaryzowanego estymatora błędów z rozkładem normalnym modelu w wariancie B	140
30. Wykres zgodności rozkładu standaryzowanego estymatora błędów z rozkładem normalnym modelu w wariancie C	141
31. Przykład modelu klasy	167
32. Związek unarny, który przedstawia powiązania pomiędzy osobami	169
33. Binarne związki według notacji UML	169
34. Meta model związku generalizacji	170
35. Przykład związku dziedziczenia	171
36. Związek agregacji	173
37. Przyrostowy proces powstawania kolejnych części produktu	174
38. Fragment diagramu Obiekt – Związek dla aplikacji „Realizacja inwestycji”	185
39. Diagram hierarchii funkcji dla modułów „Ewidencja faktur” i „Kontrola merytoryczna faktur”	187
40. Przykład modelu przypadków użycia	200

