POLSKA AKADEMIA NAUK

Instytut Badań Systemowych

ROZWÓJ I ZASTOSOWANIA METOD ILOŚCIOWYCH I TECHNIK INFORMATYCZNYCH WSPOMAGAJĄCYCH PROCESY DECYZYJNE

Redakcja:

Jan Studziński
Ludosław Drelichowski
Olgierd Hryniewicz

ROZWÓJ I ZASTOSOWANIA METOD ILOŚCIOWYCH I TECHNIK INFORMATYCZNYCH WSPOMAGAJĄCYCH PROCESY DECYZYJNE

Redakcja:

Jan Studziński

Ludosław Drelichowski

Olgierd Hryniewicz

Wydanie tej publikacji było możliwe dzięki pomocy finansowej MINISTERSTWA NAUKI I SZKOLNICTWA WYŻSZEGO.

Książka zawiera wybór artykułów poświęconych omówieniu aktualnego stanu badań w kraju w zakresie rozwoju i zastosowań metod, modeli, technik i systemów informatycznych w procesach podejmowania decyzji. Kilka artykułów przedstawia rezultaty projektów badawczych finansowanych przez Ministerstwo Nauki i Szkolnictwa Wyższego i realizowanych przez polskie instytucje badawcze.

Recenzenci:
Prof. Olgierd Hryniewicz
Prof. Andrzej Straszak
Dr hab. Jan Studziński

Komputerowa edycja tekstu: Anna Gostyńska

Wydawca: Instytut Badań Systemowych PAN
Newelska 6, PL 01-447 Warszawa

Sekcja Informacji Naukowej i Wydawnictw
e-mail: biblioteka@ibspan.waw.pl

ISBN 83-894-7506-5
9788389475060
ISSN 0208-8029

ROZWÓJ I ZASTOSOWANIA METOD ILOŚCIOWYCH I TECHNIK INFORMATYCZNYCH WSPOMAGAJĄCYCH PROCESY DECYZYJNE

Instytut Badań Systemowych • Polska Akademia Nauk
Seria: Badania Systemowe
Tom 49

Redaktor Naukowy:

Prof. Jakub Gutenbaum

Warszawa 2006

METODA ESTYMACJI DŁUGOŚCI KODU ŹRÓDŁOWEGO NA PODSTAWIE DIAGRAMÓW UML

Maciej STACHURSKI
Wydział Informatyki, Politechnika Szczecińska
<*mstachurski@wi.ps.pl*>

Streszczenie: *Artykuł ten prezentuje nową metodę estymacji rozmiaru realizowanego systemu informatycznego w aspekcie długości jego kodu źródłowego. Jako źródło danych wykorzystywane są diagramy statyczne (klas) UML, a estymaty przedstawiane są w ilości tokenów – najmniejszych jednostek leksykalnych posiadających określone znaczenie.*

Słowa kluczowe: Estymacja rozmiaru systemu informatycznego, estymacja długości kodu źródłowego, diagramy UML, diagramy klas.

1. Wprowadzenie

Estymacja parametrów realizowanego systemu informatycznego jest jednym z najistotniejszych elementów procesu planowania w ramach cyklu życia produktu informatycznego. Stanowi ona bazę do wcześniejszego określania podstawowych cech wytwarzanego systemu, takich jak, zwykle najistotniejszy, koszt przedsięwzięcia, czy też czas realizacji lub wielkość i charakterystyka czasowa wykorzystania potrzebnych zasobów. Jednocześnie estymacja jest procesem trudnym, gdyż jest ona próbą przewidywania przyszłości na podstawie zwykle stosunkowo skąpych informacji, próbą wnioskowania przy ograniczonych danych wejściowych.

Szacowanie parametrów projektów informatycznych nieodłącznie wpisało się już w cykl życia produktów informatycznych, stając się jego nieusuwalnym elementem. Wszystkie zdefiniowane metody zarządzania procesem tworzenia oprogramowania wymagają stosowania metod estymacji dla sprawnego działania, włączając w to najbardziej znane metody, takie jak CMMI-SW (2002). Metoda ta wymusza stosowanie szacowania różnych aspektów realizowanego projektu informatycznego już w najprostszych wersjach modelu zarządzającego.

2. Estymacja rozmiaru systemu informatycznego

W ramach planowania realizacji systemu informatycznego dokonuje się oszacowań wielu różnych jego parametrów. Najczęściej są to rozmiar, nakład pracy potrzebnej na realizację, czas realizacji i całkowity koszt realizacji. Listę tę nie bez

powodu otwiera rozmiar, który jest czynnikiem kluczowym bezpośrednio wpływającym na wszelkie pozostałe parametry procesu realizacji systemu informatycznego: system informatyczny cechujący się dużym rozmiarem niemal na pewno będzie wymagał większych nakładów pracy na jego realizację, co zapewne wydłuży czas jego przygotowania i spowoduje wzrost kosztów.

2.1. Pojęcie rozmiaru systemu informatycznego

Samo pojęcie rozmiaru systemu informatycznego może być rozumiane na różne sposoby. W literaturze zwykle wyróżnia się trzy podstawowe aspekty rozmiaru:

- zakres funkcjonalności
- stopień złożoności
- długość kodu źródłowego

Zakres funkcjonalności określa zbiór funkcji, jakie realizowany system informatyczny będzie oferował użytkownikowi końcowemu. Jest to sposób opisywania rozmiaru systemu informatycznego widzianego od strony użytkownika. Można go określać za pomocą różnych miar bezpośrednich, np. ilości okien interakcji z użytkownikiem, ilości kontrolek na formularzach, liczby możliwych scenariuszy wykorzystania systemu informatycznego. Oprócz miar bezpośrednich istnieje również pewna grupa miar bardziej abstrakcyjnych, związanych zwykle bezpośrednio z metodą definiującą sposób wyznaczania rozmiaru w aspekcie zakresu funkcjonalności. Takimi miarami są na przykład liczba punktów funkcyjnych (używana w metodzie punktów funkcyjnych – Longstreet D. (2004)), liczba Cfsu (*COSMIC functional size unit*; używana w metodzie COSMIC-FFP – COSMIC-FFP (2003)), liczba przypadków użycia i aktorów (metody bazujące na diagramach przypadków użycia UML).

Stopień złożoności jest miarą rozmiaru systemu informatycznego z punktu widzenia twórców systemu. Obrazuje on charakter zarówno strukturalnego rozplanowania komponentów realizowanego systemu informatycznego, jak i mechanizmów przetwarzania danych. Opisując strukturę systemu najczęściej wyróżnia się jego moduły, definiując zależności między nimi. Z kolei mechanizmy przetwarzania danych opisują schematy komunikacji pomiędzy poszczególnymi modułami, a także definiują specyficzne algorytmy realizujące przetwarzanie danych. Przykładowe miary stopnia złożoności struktury to liczba modułów, liczba klas (paradygmat obiektowy), głębokość drzewa dziedziczenia (paradygmat obiektowy), liczba związków między klasami (paradygmat obiektowy). Przykładowe miary złożoności mechanizmów przetwarzania danych to liczba cyklomatyczna, opis złożoności asymptotycznej (np. notacja wielkie O), liczba komunikatów na diagramach sekwencji.

Długość kodu źródłowego jest najprostszą miarą rozmiaru oprogramowania widzianego z punktu widzenia jego twórców. Jest to po prostu całkowita ilość kodu

źródłowego napisanego w językach programowania implementująca funkcjonalność realizowanego systemu informatycznego. Najczęściej stosowane miary tego aspektu rozmiaru systemu informatycznego to ilość linii kodu, metryki Halsteada, liczba tokenów (najmniejszych jednoznacznych elementów języka).

2.2. Metody estymacji rozmiaru

Metody estymacji rozmiaru systemu informatycznego najogólniej można podzielić na dwie podstawowe grupy:

- metody szacujące w ujęciu deklaratywnym
- metody szacujące w ujęciu imperatywnym

Pierwsza grupa metod estymacji traktuje system informatyczny jako czarną skrzynkę – za znaczące uważa tylko te jego elementy, które są widoczne z zewnątrz, czyli z punktu widzenia docelowego użytkownika systemu. Jest to równoznaczne ze stwierdzeniem, że w zakresie zainteresowań metod deklaratywnych jest tylko i wyłącznie jego funkcjonalność, czyli odpowiedź na pytanie *CO* system informatyczny powinien robić. Najpopularniejsze deklaratywne metody estymacji rozmiaru to rodzina metod punktów funkcyjnych (np. metoda IFPUG – Longstreet (2004), metoda pełnych punktów funkcyjnych COSMIC – COSMIC-FFP (2003)) oraz metoda punktów przypadków użycia – Karner (1993).

Przeciwieństwem metod deklaratywnych są metody imperatywne. Ich spektrum zainteresowań dotyczy tego, *JAK* system informatyczny ma działać. Z punktu widzenia metod imperatywnych najważniejszymi aspektami systemu informatycznego są sposoby organizacji struktury wewnętrznej oraz mechanizmy przetwarzania danych. Przykładem imperatywnej metody estymacji jest metoda Fast&Serious – Carbone, Santucci (2002).

Podział metod estymacji rozmiaru na deklaratywne i imperatywne wiąże się z ich dostępnością w trakcie cyklu życia systemu informatycznego oraz ze skutecznością uzyskiwanych przy ich pomocy oszacowań. Zakładając wykorzystanie kaskadowego modelu cyklu życia systemu informatycznego składającego się z etapów specyfikowania wymagań, analizy, projektowania, implementacji i testowania (Górski (2000)), metody deklaratywne, a więc bazujące na opisie funkcjonalności, można wykorzystywać począwszy od etapu analizy. Wówczas to dokonuje się szczegółowego opisu funkcjonalnego systemu informatycznego dostarczając podstaw do oszacowania jego rozmiaru. Przykładowo w metodzie punktów funkcyjnych (Longstreet (2004)) rozmiar systemu informatycznego szacuje się na podstawie ilości i charakteru wejść i wyjść danych, ilości i charakteru użytych wewnętrznych i zewnętrznych plików danych, oceny czynników technicznych.

W tym miejscu warto również wspomnieć o metodzie punktów przypadków użycia – Karner (1993). Metoda ta jako dane wejściowe wykorzystuje diagramy

przypadków użycia – Fowler, Scott (2000), które opisują scenariusze wykorzystania systemu informatycznego przez aktorów (użytkowników systemu).

Podsumowując, podstawową zaletą deklaratywnych metod estymacji rozmiaru jest to, że mogą być wykorzystane bardzo wcześnie w trakcie cyklu życia systemu informatycznego. Jest to o tyle istotne, że skuteczne oszacowania są jednymi z najważniejszych i najbardziej znaczących danych wejściowych dla procesu planowania. Jednocześnie stosowanie metod deklaratywnych jest potencjalnie ryzykowne, bo ich rezultaty bazują na niepełnych danych o systemie informatycznym. Sam opis funkcjonalności na pewno stanowi dobrą podstawę do oszacowań, ale jest niewystarczający, aby uzyskiwać dokładne oszacowania. Lukę tę uzupełniają metody szacujące w ujęciu imperatywnym.

Zakres wykorzystania metod imperatywnych, a więc bazujących na opisie wewnętrznej struktury systemu informatycznego oraz stosowanych w jego ramach mechanizmów przetwarzania danych jest węższy niż jest to w przypadku metod deklaratywnych. Szacowanie rozmiaru na podstawie wewnętrznej budowy systemu możliwe jest dopiero na etapie jej planowania, czyli podczas etapów zaawansowanej analizy oraz projektowania. Wynika z tego to, że oszacowania uzyskiwane w wyniku zastosowania metod imperatywnych mogą być nieco spóźnione w stosunku do typowych oczekiwań kierownictwa dotyczących fundamentalnych kwestii oceny globalnych zasobów potrzebnych do realizacji całego systemu. Z drugiej jednak strony wyniki działania metod imperatywnych mogą stanowić znaczące źródło danych dla estymacji zasobów w sensie lokalnym (np. do oszacowania nakładu pracy potrzebnej na implementację określonego modułu systemu czy też jego cechy).

Typowa metoda imperatywna operuje na definicji bądź planie struktury wewnętrznej systemu informatycznego. W przypadku metody Carbone, Santucci (2002) (Fast&&Serious) struktura wewnętrzna musi być opisana przy pomocy diagramów klas, przypadków użycia, współdziałania, stanów UML. Diagramy te są przetwarzane w celu uzyskania danych wejściowych wykorzystywanych następnie do wyznaczania estymat rozmiaru w półautomatycznym procesie obliczeniowym. Dane na podstawie, których dokonuje się wyznaczania oszacowań rozmiaru to między innymi (wszystkie pojęcia dotyczą paradygmatu obiektowego) liczba i złożoność atrybutów klas, liczba i złożoność metod klas, głębokość w drzewie hierarchii dziedziczenia.

W praktyce stosuje się również metody imperatywne bazujące na ocenie eksperta, jednak ze względu na fakt, że metody te nie są zwykle sformalizowane nie stanowią one przedmiotu zainteresowania w tym artykule.

Podsumowując, można przyjąć, że metody imperatywne stanowią uszczegółowienie metod deklaratywnych i mogą być rozumiane jako kolejny krok procesu estymacji (po zastosowaniu metod deklaratywnych). Jest to szczególnie prawdziwe, jeżeli spojrzy się na realizację systemu informatycznego jak na proces stopniowego uszczegóławiania – McConnell (1996).

3. Założenia nowej metody estymacji długości kodu źródłowego

Nowa metoda szacowania rozmiaru systemu informatycznego w aspekcie długości kodu (metoda szacująca w ujęciu imperatywnym) powinna spełniać następujące założenia:

- szacowaniu mogą podlegać tylko systemy informatyczne zrealizowane z wykorzystaniem paradygmatu obiektowego
- diagramy UML powinny być podstawowym źródłem danych na podstawie których dokonywane będą oszacowania
- długość kodu źródłowego powinna być wyrażana w obiektywnej jednostce miary
- proces dokonywania oszacowań powinien być pozbawiony elementów subiektywnych
- uzyskiwane oszacowania powinny cechować się względnym błędem nie większym niż ±20%

Warunek wykorzystania metody tylko do szacowania długości kodu źródłowego systemów informatycznych realizowanych z wykorzystaniem paradygmatu obiektowego jest powodowany jego rosnącą popularnością. Sposób modelowania bytów, w których następuje połączenie cech i operacji w jednym obiekcie okazał się być rewolucyjny z powodu swej prostoty i naturalności, pozwalającej jednocześnie tworzyć także duże systemy w sposób spójny i sprawny.

Wymaganie zastosowania diagramów UML jako źródła danych opisujących strukturę systemu informatycznego jest, podobnie jak to było w przypadku paradygmatu obiektowości, ich duża popularność. W chwili obecnej UML to podstawowy język modelowania używany w inżynierii oprogramowania. Jest to język graficzny pozwalający na modelowanie różnych aspektów realizowanego systemu informatycznego – Fowler, Scott (2000):

- statycznego (diagramy komponentów, klas, wdrożenia)
- dynamicznego (diagramy przypadków użycia, sekwencji, współdziałania, czynności, stanów)
- zarządzania (diagramy pakietów, diagramy klas opisujące podsystemy)

Jako język modelowania UML opisuje tylko pewne pojęcia kluczowe oraz składnię, nie definiuje natomiast reguł opisujących, kiedy należy go stosować. W związku z tym twórcy oprogramowania dość dowolnie podchodzą do kwestii wykorzystywania języka UML w procesie analizy i projektowania, przygotowując tylko te diagramy, które uznają za konieczne. Wiąże się z tym pewna trudność dla twórców metod szacowania wykorzystujących diagramy UML jako podstawowe źródło danych o projekcie: nie można zakładać, że w procesie analizy i projektowania powstają wszystkie diagramy. W rzeczywistości najczęściej wykorzystuje się diagramy przypadków użycia oraz diagramy klas.

Jednostka miary długości kodu źródłowego powinna mieć przede wszystkim jasną i spójną definicję. Nie można tego powiedzieć np. o ilości linii kodu (LOC – *Lines of Code*), najpopularniejszej mierze długości kodu źródłowego, która w rzeczywistości mierzy ilość znaków końca linii w zbiorze znaków (pliku), która może być zupełnie niezależna od długości kodu źródłowego. Dlatego też powstały liczne dodatkowe zasady pozwalające zaklasyfikować dany wiersz znaków jako linię kodu źródłowego (np. wymaganie *non-commented, non-blank* – niepusta linia niebędąca komentarzem). Dobrym przykładem miary długości kodu źródłowego o ścisłej definicji może być token. Token to pojęcie wywodzące się z analizy leksykalnej i oznaczające najmniejszą jednostkę leksykalną posiadającą określone znaczenie. Przykładem tokena jest w językach programowania np. operator przypisania, nawias, identyfikator zmiennej, stała liczbowa, słowo kluczowe. Token jest jednostką ściśle zdefiniowaną, nie ma tu swobody interpretacji jak to było w przypadku linii kodu źródłowego LOC. Jednak token jest do pewnego stopnia zależny od języka programowania, np. połączony operator przypisania i dodawania '+=' jest zdefiniowany w językach C i Java, nie ma go natomiast w języku Pascal. Z drugiej jednak strony większość współcześnie stosowanych języków programowania ma bardzo podobny zestaw podstawowych tokenów, przynajmniej w sensie znaczeniowym, zatem zależność tokena jako miary rozmiaru systemu informatycznego od języka programowania nie jest bardzo duża.

Wymaganie dotyczące wyeliminowania elementów subiektywnych jest podyktowana zamierzeniem wykorzystania metody do automatycznej estymacji długości kodu źródłowego w pakietach modelowania UML.

Graniczna wartość błędu względnego na poziomie ±20% została dobrana zgodnie z panującymi obecnie standardami w ocenie metod estymacji rozmiaru systemów informatycznych.

4. Propozycja metody estymacji długości kodu źródłowego

Na proponowaną metodę estymacji długości kodu źródłowego systemu Indormatycznego składa się model szacujący oraz zbiór reguł i procedur postępowania wymaganych w celu sprawnego wyznaczania dokładnych oszacowań. Sednem metody jest model, który operuje na danych wejściowych podzielonych na dwa poziomy ziarnistości, odpowiadających dwóm poziomom agregacji wydzielonym w diagramach statycznych UML (a co za tym idzie również w obiektowych językach programowania):

- poziom funkcji (metody)
- poziom klasy

Poziomy te są w sobie zagnieżdżone – metoda jest elementem klasy. Relacja zawierania się implikuje sposób dokonywania estymacji rozmiaru tych bytów: roz-

miar klasy będzie szacowany na podstawie oszacowanego rozmiaru wchodzących w jego skład funkcji i ewentualnie dodatkowych, własnych cech.

4.1. Estymacja długości kodu źródłowego implementującego metodę

Definicja metody w większości języków programowania składa się z następujących elementów:

- nazwy (identyfikatora) metody
- listy typów opisujących rezultaty działania (wyjścia) metody
- listy typów opisujących parametry działania (wejścia) metody

W proponowanym rozwiązaniu przyjęto założenie, że nazwa metody niesie informację dotyczącą sposobu przetwarzania danych. W ogólnym przypadku nie musi to być prawdą (zasady tworzenia identyfikatorów funkcji ograniczają tylko zestaw znaków, jakie mogą być użyte), jednak w praktyce stosuje się nazewnictwo odpowiadające funkcjonalności. Istnieje pewna grupa kategorii metod, które są stosunkowo proste w identyfikacji i przez to są powszechnie uznawane – Riehle D. (2000). Niektóre z nich (np. metody ustawiające – ang. *setter* – i pobierające dane – ang. *getter*) stanowią znaczącą część całkowitej populacji metod (patrz Tabela 1).

Tabela 1. Udział niektórych kategorii metod w populacji generalnej

Kategoria metody	Udział w populacji [%]
konstruktor	10,51%
getter	28,16%
setter	11,09%

Źródło: opracowanie własne

Każdej metodzie w zależności od identyfikatora (nazwy) została przypisana jedna z następujących kategorii (opracowanie własne):

- konstruktor (ang. *constructor*)
- metoda ustawiająca dane (ang. *setter*)
- metoda pobierająca dane (ang. *getter*)
- metoda tworząca obiekty (ang. *factory*)
- metoda wejścia/wyjścia (ang. *I/O*)
- metoda przetwarzająca (ang. *processor*)
- metoda obsługi zdarzeń (ang. *handler*)
- metoda obsługująca kolekcje (ang. *collection*)
- metoda wyszukująca (ang. *finder*)

- metoda konwertująca (ang. *converter*)
- metoda testująca (ang. *tester*)
- inna metoda (ang. *other*)

W proponowanym modelu dla każdej kategorii funkcji utworzono podmodel na bazie regresji wielorakiej, przyjmujący jako zmienne niezależne liczbę rezultatów funkcji, liczbę parametrów funkcji i liczbę atrybutów klasy, na których funkcja operuje, a jako zmienną zależną długość kodu źródłowego implementującego daną funkcję (wyrażony w liczbie tokenów). Jest on odzwierciedleniem następującej zależności:

$$S_m = f\left(kategoria_m, |parametry_m|, |rezultaty_m|\right)$$

gdzie S_m to długość kodu implementującego metodę, $kategoria_m$ to kategoria metody m, $parametry_m$ to zbiór parametrów metody m, a $rezultaty_m$ to zbiór rezultatów metody m.

4.2. Szacowanie długości kodu źródłowego implementującego klasę

Klasy są schematem obiektu i zawierają definicje pól (atrybutów) oraz funkcji (metod) a także powiązania z innymi klasami (w postaci dziedziczenia i asocjacji). Proponowany model do wyznaczenia oszacowania rozmiaru klasy (wyrażanego w ilości tokenów) wykorzystuje estymaty rozmiarów metod i ilość atrybutów zawartych w klasie. Model jest wyznaczony z użyciem regresji wielorakiej i ma postać odpowiadającą zależności:

$$S_c = f\left(\sum_m S_{m,c}, \sum_d S_{d,c}, |atrybuty_c|\right)$$

gdzie S_c to długość kodu implementującego klasę c, $S_{m,c}$ to długość m-tej metody klasy c, $S_{d,c}$ to długość d-tej metoda klasy c, a $atrybuty_c$ to zbiór atrybutów klasy c.

4.3. Wyniki badań sprawności estymacji

Prototyp modelu poddano badaniom sprawności estymacji. W tym celu zgromadzono kody źródłowe 21 systemów informatycznych zaimplementowanych w języku Java (projekty te to aplikacje i biblioteki open source dostępne w portalu http://www.sourceforge.net/. Następnie każdy z projektów poddano procesowi reinżynierii w celu pozyskania danych wymaganych przez model, a następnie oszacowano długość kodu źródłowego przy pomocy opracowanego wcześniej modelu. Uzyskane oszacowania każdego z projektów zostały zamieszczone w Tabeli 2.

Estymatę każdego projektu oceniono w dwóch kategoriach:

- średniego błędu względnego (MRE) – obrazująca o ile procent różni się oszacowanie od rzeczywistej wartości rozmiaru,

- bezwzględnego średniego błędu względnego (MMRE) – wartość bezwzględna z MRE.

Tabela 2. Wyniki estymacji długości kodu źródłowego

Projekt	MRE	Projekt	MRE	Projekt	MRE
azureus_2302	-17,48%	jboss_402	2,84%	rtext_0923	-2,23%
columba_1	-26,74%	jetty_514	16,50%	saxon_84	22,02%
gantt_20	1,93%	jtds_11	10,63%	spring_121	32,48%
hibernate_305	32,85%	liferay_361	-18,15%	track_310	-27,92%
hsqldb_1733	28,32%	mantaray_18	-4,89%	weka_350	-4,10%
jameleon_302	59,88%	phex_26489	14,09%	wicket_102	14,81%
jasperreports_100	-2,83%	rssowl_113	-21,66%	xdoclet_123	-15,99%

Źródło: opracowanie własne

Definicje poszczególnych pojęć wskaźników oceny znajdują się poniżej:

$$RE(P_i) = \frac{estymata_i - rozmiar_i}{rozmiar_i}, i = 1,...,N$$

$$MRE(P_i) = |RE(P_i)|, i = 1,...,N$$

$$MMRE(P) = \frac{1}{N}\sum_{i=1}^{N} MRE(P_i)$$

gdzie $RE(P_i)$ to błąd względny oceny projektu P_i (ang. *relative error*), $MRE(P_i)$ to średni błąd względny oszacowania P_i (ang. *mean relative error*), *MMRE(P)* to średnia wartość bezwzględna błędu względnego (ang. *magnitude of mean relative error*).

Ponadto całościowo model poddano ocenie w dwóch kategoriach:

- średniej bezwzględnego średniego błędu względnego
- współczynnika predykcji PRED(25%) (ilorazu ilości przypadków w których MMRE jest mniejsze niż 25% do całkowitej liczby przypadków)

$$PRED(p,N) = \frac{|\{P_i : MRE(P_i) \le p\}|}{N}, i = 1,...,N$$

gdzie p to graniczna średnia bezwzględnego średniego błędu względnego, N to moc zbioru populacji badanych projektów, P_i to i-ty projekt.

Wyniki skuteczności działania modelu są zadowalające: przeciętna wartość błędu wynosi około 18% (w maksymalnym przypadku około 60%). Wartość współczynnika predykcji jest bliska przedziałowi optymalnemu – 71,43% (uznaje się, że wartość PRED(25%) powinna być większa niż 75%).

5. Wnioski

Rezultaty przeprowadzonego eksperymentu potwierdzają skuteczność zaproponowanej metody. Jako przedstawiciel imperatywnych metod estymacji długości kodu źródłowego może stanowić skuteczne uzupełnienie metod deklaratywnych (funkcjonalnych). Jednocześnie warte podkreślenia jest to, że potencjał zaprezentowanych tutaj badań nie został całkowicie wyczerpany. Kwestie, które należałoby dodatkowo rozważyć to, między innymi, uwzględnienie typów parametrów i rezultatów metod oraz atrybutów klas, kalibracja metody do specyficznych warunków konkretnego systemu informatycznego. Wprowadzenie tych cech do modelu może mieć pozytywny wpływ na dokładność uzyskiwanych oszacowań długości kodu źródłowego.

Literatura

Carbone M., Santucci G. (2002) Fast&Serious: a UML Based Metric for Effort Estimation, materiały 6 warsztatów Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE).

CMMI-SW (2002) Capability Maturity Model Integration for Software Engineering, http://www.sei.cmu.edu/cmmi/

COSMIC-FFP (2003) COSMIC Full Function Point, http://www.lrgl.uqam.ca/cosmic-ffp/

Fowler M., Scott K. (2000) *UML w kropelce.* Oficyna Wydawnicza LTP, Warszawa

Górski J. (2000) *Inżynieria oprogramowania w projekcie informatycznym.* Wydawnictwo Informatyki MIKOM, Warszawa.

Karner G. (1993) Resource Estimation for Objectory Projects, Objective Systems SF AB.

Longstreet D. (2004) *Function Points Analysis Training Course.* Longstreet Consulting Inc.

McConnell S. (1996) *Rapid Development. Taming Wild Software Schedules.* Microsoft Press, Redmond.

Riehle D. (2000) Metod Types In Java. *Java Report 2/2000.*

SOURCE CODE ESTIMATION METHOD WITH THE USE OF UML DIAGRAMS MACIEJ STACHURSKI

Abstract: *This article gives an overview of a new software size estimation method, while software size is interpreted as source code length. The main source of estimation data are statical (class) UML diagrams. The estimates are expressed in number of tokens – the smallest lexical units with assigned meaning.*

Keywords: Computer aided support of process management, human resource management.

Jan Studziński, Ludosław Drelichowski, Olgierd Hryniewicz
(Redakcja)

ROZWÓJ I ZASTOSOWANIA METOD ILOŚCIOWYCH I TECHNIK INFORMATYCZNYCH WSPOMAGAJĄCYCH PROCESY DECYZYJNE

Monografia zawiera wybór artykułów dotyczących informatyzacji procesów zarządzania, prezentując aktualny stan rozwoju informatyki stosowanej w Polsce i na świecie. Zamieszczone artykuły opisują metody, modele, techniki i systemy informatyczne stosowane do wspomagania procesów podejmowania decyzji, a także omawiają zastosowania narzędzi informatycznych w różnych sektorach gospodarki. Kilka prac przedstawia wyniki projektów badawczych Ministerstwa Nauki i Szkolnictwa Wyższego, dotyczących rozwoju metod informatycznych i ich zastosowań.

ISBN 83-894-7506-5
9788389475060
ISSN 0208-8029

W celu uzyskania bliższych informacji i zakupu dodatkowych egzemplarzy prosimy o kontakt z Instytutem Badań Systemowych PAN
ul. Newelska 6, 01-447 Warszawa
tel. 837-35-78 w. 241 e-mail: biblioteka@ibspan.waw.pl