POLSKA AKADEMIA NAUK

Instytut Badań Systemowych

ROZWÓJ I ZASTOSOWANIA TECHNOLOGII I SYSTEMÓW INFORMATYCZNYCH

pod redakcją:
Jana Studzińskiego
Ludosława Drelichowskiego
Olgierda Hryniewicza

ROZWÓJ I ZASTOSOWANIA TECHNOLOGII
I SYSTEMÓW INFORMATYCZNYCH

Polska Akademia Nauk • Instytut Badań Systemowych

Seria: BADANIA SYSTEMOWE
tom 28

Redaktor naukowy:

Prof. dr hab. Jakub Gutenbaum

Warszawa 2001

ROZWÓJ I ZASTOSOWANIA TECHNOLOGII I SYSTEMÓW INFORMATYCZNYCH

pod redakcją
Jana Studzińskiego, Ludosława Drelichowskiego
i Olgierda Hryniewicza

Wydano z wykorzystaniem dotacji KOMITETU BADAŃ NAUKOWYCH

Książka zawiera wybór artykułów poświęconych omówieniu aktualnego stanu badań w kraju w zakresie rozwoju technologii, modeli i systemów informatycznych oraz ich zastosowań w różnych dziedzinach gospodarki narodowej. Wyodrębnioną grupę stanowią artykuły aplikacyjne omawiające wyniki projektów badawczych i celowych KBN.

Recenzenci artykułów:

Dr hab. inż. Ryszard Budziński, prof. US
Prof. dr hab. inż. Janusz Kacprzyk
Dr hab. Adam Kopiński, prof. AE we Wrocławiu
Doc dr hab. inż. Marek Libura
Prof. dr hab. inż. Andrzej Straszak

ISBN 83-85847-59-6
ISSN 0208-8028

Rozdział 2

Metodologia i rozwój systemów informatycznych

WYMIAROWANIE PROJEKTÓW INFORMATYCZNYCH

Zdzisław Szyjewski
Uniwersytet Szczeciński, Instytut Informatyki w Zarządzaniu
zszyjew@uoo.univ.szczecin.pl

1. Wprowadzenie

Systemy informatyczne stanowią istotną gałąź działalności gospodarczej zarówno w wymiarze ponoszonych nakładów, jak i obszaru oddziaływania. Znaczenie systemów informatycznych można rozpatrywać w kontekście ponoszonych nakładów inwestycyjnych lub potencjalnych zysków uzyskiwanych z zastosowań tych systemów w przypadku ich pomyślnego ukończenia i wdrożenia do działalności gospodarczej. Statystyki[1] pokazują, że wiele projektów informatycznych kończy się niepowodzeniem lub zakładane efekty osiągane są w części lub w dłuższym niż przewidywano, okresie czasu. Przyczyny takiego stanu są bardzo złożone i brak jednoznacznych odpowiedzi jakie działania mogą temu przeciwdziałać.

Inżynieria oprogramowania, jest dziedziną badań naukowych, która stawia sobie za cel zasadniczy wypracowanie takich metod wytwarzania systemów informatycznych aby minimalizować te negatywne skutki nieudanych inwestycji informatycznych. Jednym z obszarów aktywności naukowej jest stworzenie wiarygodnych metod wymiarowania systemów informatycznych. Wymiarowanie systemów informatycznych w początkowych fazach cyklu życia systemu jest szczególnie trudne ale bardzo istotne z punktu widzenia efektywności podejmowanych inwestycji informatycznych. Szacunki dokonane we wstępnych fazach są bardzo często obarczone dużym błędem, który przenoszony jest na ocenę efektywności przedsięwzięcia informatycznego. Inżynieria oprogramowania dostarcza metod pozwalających na wymiarowanie systemów informatycznych.

2. Miary stosowane w projektach informatycznych

Szacowaniu podlegają różne elementy projektu, takie jak czas, pracochłonność, koszty, wydajność, zużycie materiałów i inne. W przypadku projektów informatycznych oprócz oczywistych trudności wynikających z samej procedury szacowania dochodzi dodatkowa trudność związana z przedmiotem estymacji. Jak mierzyć projekty informatyczne, jakie jednostki miary należy stosować w przypadku projektów informatycznych?

Wymiarowanie systemów informatycznych jest potrzebne, ponieważ umożliwia korzystanie z doświadczeń zebranych w trakcie realizacji poprzednich

[1] Bogate statystyki można znaleźć na stronie domowej Software Productivity Research www.spr.org

projektów. Jeśli na przykład szacujemy koszt nowej budowy, odwołujemy się do powierzchni budowli. Wyliczony średni koszt metra kwadratowego pozwala łatwo oszacować koszt budowy innej budowli z prostego wzoru, gdzie szacowana wielkość to iloczyn budowanej powierzchni przez koszt jednego metra. W przypadku systemu informatycznego nie ma jednoznacznej miary, która pozwalałaby przenieść doświadczenia z jednego systemu na inny. Abstrahując od różnych uwarunkowań realizacyjnych, nie istnieje prosty sposób porównania dwóch systemów informatycznych. W jaki sposób zatem można określić wielkość czy złożoność systemu informatycznego?

Wiele miar dotyczących programów odnosi się do kodu źródłowego programu. Jeśli jednak oceniamy wielkość programu według liczby linii kodu, nie uzyskujemy informacji na temat złożoności programu. Porównując dwa programy, możemy tylko powiedzieć, że jeden jest większy od drugiego. Programy mające rozwiązywać różne problemy są natomiast nieporównywalne z punktu widzenia złożoności, która stanowi o pracochłonności wykonania każdego z nich. Porównanie takie jest więc ułomne, nie daje informacji wystarczających do dokonania dalszych szacunków. Problem komplikuje się dodatkowo, jeśli programy pisane są w różnych językach programowania.

Pierwsze wzmianki o próbach mierzenia oprogramowania można znaleźć w pracy [24]. Inne prace naukowe dotyczące złożoności oprogramowania to praca [2] i przywołana w niej praca doktorska Van Emdena *An Analysis of Complexity* [29]. Dwie inne miary złożoności oprogramowania zaproponował Hecht [11]. Warto jeszcze wspomnieć o pracach [17] i [10] powstałych w latach siedemdziesiątych. W latach osiemdziesiątych kontynuowano prace nad problematyką miar oprogramowania. Wyniki tych badań można znaleźć w pracach Cote [6], w publikacji Conte, gdzie przedstawiona jest lista miar różnego rodzaju, w pracy Waguespacka [25] czy w raporcie [24]. Proponowane miary za podstawę przyjmują linie kodu oprogramowania.

Z nowszych prac warto wymienić prace Horsta Zuse, który proponuje w pracy [30] kilkadziesiąt metod wymiarowania złożoności programów komputerowych. Złożoność programu jest wyliczana na podstawie zamiany schematu opisującego program na odpowiednie wagi numeryczne. Tak zmierzone programy dają się porównywać, czyli możemy określić, że jeden program jest bardziej lub mniej złożony niż inny. Procedura "mierzenia" programu jest dość skomplikowana i bez komputerowych narzędzi wspomagających bardzo trudna do przeprowadzenia. Przy użyciu podanych metod możemy wymiarować programy dobrze wyspecyfikowane.

Podstawowym problemem jest odpowiedź na pytanie, *co mierzyć*? W naukach wymiernych, takich jak fizyka, sprawa jest dość oczywista i problemem pozostaje jedynie jednostka miary i metoda jej wyznaczenia. W naukach takich jak psychologia, socjologia czy informatyka mamy do czynienia z problemem wyznaczenia jednostki miary, która byłaby odpowiednia do oceny istotnych cech rozważanego produktu. W przypadku informatyki problem ten w szczególności dotyczy złożoności oprogramowania. W wielu przypadkach nie chodzi o wyznaczenie numerycznej miary jakiejś cechy oprogramowania, ale o określenie relacji, np.:

- program A jest dwa razy bardziej złożony niż program B,
- program B jest dwa razy trudniejszy do pielęgnacji niż program A,
- program A jest bardziej złożony niż program B.

Według Robertsa [22], w celu udzielenia poprawnych odpowiedzi na postawione pytania należy rozważyć następujące problemy:

- Co wyznacza miara "bardziej lub mniej złożony"?
- Kiedy ma sens stwierdzenie, że program A jest dwa razy bardziej złożony niż program B?
- Jaki sens ma stwierdzenie, że średnia złożoność programów w systemie A jest dwukrotnie większa niż średnia złożoność programów w systemie B?
- Czy jest możliwe mierzenie złożoności programu według jednej miary, czy też należałoby wyliczać różne rodzaje złożoności?
- Jaki sens ma stwierdzenie, że złożoność kolejnej wersji programu wzrasta o 25%?

Zestawione powyżej problemy to jedynie kilka przykładów trudności, z jakimi spotykamy się w próbach określenia relacji lub wymiarowania złożoności oprogramowania. W związku z tymi problemami dość oczywiste wydaje się stwierdzenie autora pracy [24], że trudno jest znaleźć w literaturze odpowiednią definicję miary złożoności oprogramowania. Podejmowane są jednak próby, z których warto przytoczyć definicję zawartą w pracy [19]. Według niej miara oprogramowania to możliwość określenia czynników wspomagających ilościowe porównanie i oszacowanie oprogramowania w procesach towarzyszących jego projektowaniu, rozwojowi, użytkowaniu, pielęgnacji i modyfikacji.

Według Rombacha i Bradforda [23] miary oprogramowania obejmują proces od planowania miar (identyfikacja celów mierzenia), poprzez wykonanie pomiaru (zebranie danych i ich wartościowanie), aż do czerpania wiedzy z analizowanych danych. Najważniejsza jest odpowiedź na pytanie, jaki jest cel mierzenia. Po zdefiniowaniu celu mierzenia powinniśmy określić, jakich miar będziemy używać.

W pracy [24] zdefiniowano sześć podstawowych obszarów mierzenia oprogramowania:

a) koszty,

b) pracochłonność,

c) jakość,

d) niezawodność,

e) złożoność,

f) złożoność obliczeniowa algorytmu.

Ad. a) Najlepszym ze znanych modeli szacowania kosztów jest Model COCOMO opisany przez Boehma [3] i Putnama [20]. Model ten wspomaga szaco-

wanie całkowitych kosztów wytworzenia oprogramowania całego projektu lub jedynie określonego etapu.

Ad. b) i c) Nie tylko koszt wytworzenia oprogramowania jest ważny, ale również pracochłonność i jakość wytworzonego oprogramowania. Rozważania na ten temat można znaleźć w pracach [3], [15] i [18].

Ad. d) Modele niezawodności to modele statystyczne służące określeniu średniego czasu bezawaryjnej pracy oprogramowania. Rozważania na ten temat można znaleźć w pracy [21].

Ad. e) Złożoność oprogramowania jest złym sformułowaniem. Termin ten oznacza, jak trudno jest pielęgnować, zmieniać i zrozumieć program. Mamy tutaj do czynienia ze złożonością w rozumieniu psychologicznym. W szczególności można mierzyć przepływy danych czy przepływy informacji [14] lub związki międzymodułowe [30].

Ad. f) Złożoność obliczeniowa jest związana ze złożonością rozwiązywanego problemu i określeniem efektywności algorytmu zastosowanego rozwiązania [9].

W literaturze występują dwa różne rodzaje złożoności oprogramowania:

- złożoność obliczeniowa,
- złożoność psychologiczna.

Złożoność obliczeniowa to złożoność algorytmu, na przykład może być to liczba operacji potrzebnych dla uporządkowania danych. Złożoność psychologiczna to intuicyjne rozumienie tego pojęcia przez kierownika projektu lub programistę w kontekście prac nad programem. W naszych rozważaniach skoncentrujemy się na złożoności psychologicznej oprogramowania.

Basili [1] dzieli złożoność oprogramowania na dwie klasy: miary statyczne i historyczne.

Statyczne miary złożoności produktu w określonym czasie dzielą się na trzy grupy:

miara objętości:

- linie kodu (LOC),
- liczba zdań źródłowych,
- liczba operacji i operandów,
- liczba procedur,
- średnia długość procedury,
- liczba zmiennych;

miara organizacji sterowania:

- złożoność cykliczna zdefiniowana przez McCabe [17],
- miara pętli zdefiniowana przez Woodwarda i Hennela [26].

– minimalna liczba przecięć zdefiniowana przez Chena [4],

– średni poziom zanurzenia zdefiniowany przez Dunsmore [7];

miara organizacji danych:

– miara powiązania danych.

Historyczne miary to jakość produktu w określonym czasie.

W pracy [8] można znaleźć wiele innych podziałów i charakterystyk miar złożoności oprogramowania. Niezależnie od tego, jak sformułujemy problem w sensie przedmiotu szacowania i jednostek miary, zawsze problemem będzie metoda pomiaru i jej wiarygodność.

3. Metoda linii kodu

Historycznie pierwszą próbą wprowadzenia jednorodnej miary dla oceny systemów informatycznych była metoda linii kodu, opracowana przez firmę IBM na podstawie doświadczeń zdobytych przy opracowaniu systemów informatycznych[2]. Metoda ta ma wiele wad i stosowanie jej obecnie jest bardzo ograniczone, ale warto zapoznać się z jej ideą i sposobem rozumowania. Metoda linii kodu może być stosowana w przypadku oceny projektów powstających w takim samym języku programowania, chociaż w tym przypadku też powstają wątpliwości, jak należy rozumieć pojęcie "linia kodu". Czy linią kodu jest komentarz, deklaracja itp. Komplikacje zwiększają się, jeśli mamy do czynienia z różnymi językami programowania. Jeśli w jednym języku można dokonać zapisu jakiejś operacji za pomocą kilkunastu instrukcji, a w innym wystarczy jedna instrukcja, porównywalność wyników szacowania jest bardzo wątpliwa. Niektórzy autorzy wprowadzają definicję operacji wykonywanych na LOC *(Lines Of Codes)*, co pozwala oszacować duże programy na podstawie podziału na mniejsze jednostki programowe [30].

Metoda linii kodu była wykorzystywana do szacowania pracochłonności opracowania programów komputerowych. Metoda ma charakter parametryczny i opiera się na następującej regule:

$$P = \frac{N}{170} * K$$

gdzie:

P – pracochłonność w osobomiesiącach,

N – liczba linii kodu,

K – współczynnik korygujący, zawierający się w przedziale (1 – 3,1).

Wielkość 170 jest wartością stałą, wyliczoną na podstawie opracowania danych statystycznych z programów wykonanych dotychczas w firmie IBM. Pozo-

[2] Linia kodu jako miara programu była stosowana przez wielu autorów zajmujących się miarami w informatyce (por. [5],[16],[30]).

stałe wielkości wykorzystywane w formule są szacowane. Reguła pozwala wyliczyć pracochłonność w osobomiesiącach na podstawie określonego rozmiaru programu, wyrażonego w liniach kodu[3], oraz oszacowanej trudności programu.

Współczynnik korygujący jest obliczany według formuły:

$$K = 1 + a + b + c + d + e$$

gdzie:

a = (0 – 0,5) – doświadczenie zespołu
b = (0 – 0,7) – zmienność wymagań
c = (0 – 0,3) – ograniczenia sprzętowe
d = (0 – 0,2) – wymagania kompletności rozwiązania
e = (0 – 0, 4) – ograniczenia zewnętrzne.

Formuła wyliczania wartości współczynnika korygującego uwzględnia najistotniejsze uwarunkowania procesu wytwarzania programu. Z podanych wartości widać, że największą trudność realizacyjną mogą sprawiać zmieniające się wymagania użytkownika. Kompletność rozwiązań jest oczywistym utrudnieniem realizacji, ale nie stanowi zbyt ważnego ograniczenia. Wartość każdego składnika formuły jest szacowana w zależności od konkretnych warunków realizacji programu. W przypadku gdy wszystkie składniki przyjmują wartości maksymalne, wyliczona wartość pracochłonności programu podlega korekcie o 3,1. Oznacza to, że taki sam program, w sensie rozmiaru liczonego w liniach kodu, może być oszacowany jako trzykrotnie trudniejszy w zależności od warunków realizacji. Metoda linii kodu odnosi się oczywiście do pełnego cyklu wytwarzania programu, a nie tylko do fazy kodowania.

Na podstawie wyliczonej pracochłonności można oszacować liczebność zespołu wykonawczego oraz czas realizacji programu. Liczebność zespołu jest szacowana, jako pierwiastek z pracochłonności, na podstawie wzoru:

$$N = \sqrt{P},$$

gdzie N oznacza liczebność zespołu.

Natomiast czas realizacji zadania wyliczany jest według formuły:

$$T = \frac{P}{N},$$

gdzie T oznacza czas realizacji zadania wyrażony w miesiącach.

Mimo, że metoda linii kodu ma kilka istotnych wad, należy odnotować, że była pierwszą próbą sparametryzowania metod szacowania podstawowych wskaźników projektu informatycznego. Wykorzystanie doświadczeń zebranych przy opra-

[3] Powstaje pytanie, jak na wstępnym etapie prac programistycznych oszacować liczbę linii kodu. Metoda ta nie rozwiązuje tego problemu. Pozostaje nam wnioskowanie na podstawie doświadczeń, czyli stosowanie metody analogii.

cowaniu podobnych programów pozwala wyciągać wnioski przy szacowaniu parametrów dla kolejnego programu. Mimo niewielkich walorów stosowania tej metody w warunkach programowania w różnych językach i przy innych zmiennych uwarunkowaniach technologii informatycznych, może być ona przydatna do opracowania własnych wielkości parametrycznych służących szacowaniu pracochłonności, czasu realizacji projektu i optymalnej liczebności zespołu wykonawczego.

Podstawowym problemem jest jednak sposób określenia wielkości programu w liniach kodu na wstępnym etapie prac nad projektem. W sytuacji gdy wiele istotnych elementów jest nieznanych, szacunek taki jest bardzo wątpliwy, a zatem wnioski sformułowane na podstawie oszacowanej liczby linii kodu nie są wiarygodne. Rozwiązaniem jest podział programu na mniejsze elementy i szacowanie mniejszych partii, ale podział taki nie jest możliwy na wstępnym etapie prac.

4. Metoda punktów funkcyjnych

We wstępnym etapie prac nad projektem informatycznym znacznie lepszą miarą niż liczba linii kodu jest określenie struktury systemu informatycznego z uwzględnieniem elementów systemu przetwarzania. Takie podejście do problemu mierzenia projektów informatycznych zaproponował Allan J. Albrecht i nazwał je metodą punktów funkcyjnych [3]. Metoda punktów funkcyjnych została zdefiniowana przez Albrechta na podstawie prac zespołu pracowników IBM. Powstała w wyniku wieloletnich doświadczeń przy opracowywaniu systemów informatycznych z zakresu przetwarzania danych. Metoda ta zdobyła szybko dużą popularność i nadal jest doskonalona przez organizację IFPUG (*International Function Point User Group*) [12,13,27].

Metoda szacowania wielkości systemu informatycznego w punktach funkcyjnych pozwala na jednorodną ocenę różnych systemów informatycznych, czyli stanowi wspólną miarę dla dokonywanych ocen. Jeśli wielkość systemu informatycznego została wyliczona w punktach funkcyjnych, można porównywać złożoność różnych systemów informatycznych. Brak jednorodnej miary nie pozwalał jednoznacznie ocenić, który z systemów jest bardziej złożony, jeśli systemy dotyczyły różnych klas zastosowań. Metoda ta pozwala również ocenić wydajność zespołów wykonawczych pracujących nad różnymi systemami informatycznymi. Zespół, który w takim samym czasie wykonał system o większej liczbie punktów funkcyjnych, ma zapewne większą wydajność niż zespół realizujący system o mniejszej liczbie punktów funkcyjnych.

Jeśli dysponujemy już jednorodną miarą, możemy przeliczać różne inne wielkości na punkt funkcyjny. W szczególności:

- liczbę wymaganych testów oprogramowania,
- koszt globalny wykonania jednego punktu funkcyjnego,
- koszt pielęgnacji punktu funkcyjnego,
- koszt eksploatacji punktu funkcyjnego,
- stopień zmian na punkt funkcyjny,

- wydajność programisty,
- inne.

Wartości różnych wielkości odnoszących się do punktu funkcyjnego mogą być uniwersalne lub specyficzne dla konkretnej firmy lub zespołu wykonawców. Wartości o charakterze uniwersalnym można znaleźć na stronie internetowej *Software Productivity Research* [28]. Znacznie ważniejsze, dla potrzeb szacowania, są wartości wyliczone na podstawie danych dotyczących wykonanych już systemów własnych. Nawet jeśli liczba zrealizowanych systemów nie jest zbyt duża, obliczenia wykonane dla konkretnych warunków są dokładniejsze niż miary uniwersalne.

5. Podsumowanie

Problem wymiarowania systemu informatycznego nie jest łatwy i niezależnie od przyjętej metody szacowania pozostają wątpliwości o wiarygodność dokonanych szacunków. Alternatywą dla metod szacowania jest jednak pozostawianie sytuacji nieokreślonej, gdzie brak jakichkolwiek liczbowych wielkości stanowi poważne zagrożenie błędnych decyzji. Ważnym problemem pozostaje problem danych historycznych adekwatnych dla konkretnych warunków realizacji systemu informatycznego. Korzystanie z danych publikowanych dla innych warunków realizacji wprowadza dodatkowe źródło błędu i właśnie dlatego tak ważne jest zbieranie własnych danych historycznych. Nawet stosunkowo nieliczne własne dane historyczne mogą stanowić lepszą podstawę szacunków niż bardzo dokładne dane uniwersalne.

Zbieranie, gromadzenie i opracowywanie danych historycznych z realizowanych projektów informatycznych jest kosztowne. Brak tradycji i mechanizmów prowadzenia takich statystyk jest dodatkowym utrudnieniem wdrożenia systemu tworzenia własnych miar dla inwestycji informatycznych. Biorąc jednak pod uwagę potencjalne zyski z posiadania takich miar adekwatnych do konkretnych warunków realizacji systemów informatycznych należy liczyć się z coraz większym zainteresowaniem stosowania metod szacowania projektów informatycznych.

Bibliografia

[1] V. Basili, Tutorial on Models and Metrics for Software Management and Engineering, Computer Society Press, EHO-167-7, COMPSAC80

[2] L. A. Belady, *On Software Complexity* w Workshop on Quantitative, Software Models for reliability, 1979

[3] B. W. Bochm, *Software Engineering Economics*, Prentice Hall, 1981

[4] E. Chen, *Program Complexity and Programmer Productivity*, IEEE Transactions on Software Engineering, Vol. SE-4, May 1978

[5] S. D. Conte, H. E. Dunsmore, V. Y. Shen, *Software Engineering Metrics and Model*, Benjamin/Cummings Poblishing Company, Menlo Park, 1984

[6] V. Cote, P. Bourque, S. Oligny, N. Rivard, *Software Metrics: An Overview About Recent Results*, The Journal of Systems and Software, No. 8, 1988

[7] H. E. Dunsmore, J. D. Gannon, *Programming Factors; Language Features That Help Explain Programming Complexity*, Proc. ACM Annual Conference, Washington, D.C., Dec. 1978

[8] J. J. Elliott, N. E. Fenton, S. Linkman, G. Markham, R. Whitty, *Structure-Based Software Measurement*, Alvey Project SE/069, Department of Electrical Engineering, South Bank, Polytechnic, Borough Road, London, SE1 OAA, Uk, 1988

[9] M. R. Garey, D. S. Johnson, *Computers and intractability*, W. H. Freeman & Co., 1979

[10] M. H. Halstead, *Elements of Software Science*, New York, Elsevier North-Holland, 1977

[11]M. S. Hecht, *Flow Analysis of Computer Programs*, Elsevier, New York, 1977

[12] IFPUG, *International Function Point Users Group, Function Point Counting Practices Manual,* Release 3.0, IFPUG, Westerville, Ohio, 1990

[13] IFPUG, *International Function Point Users Group, Function Point Counting Practices Manual,* Release 4.0, IFPUG, Westerville, Ohio, 1994

[14] Dennis Kafura, S. Henry, *Software Quality Metrics Based on Interconnectivity*, Journal onSystems and Software, Vol. 2, 1982

[15] B. Kitchenham, *Towards a Constructive Quality Model*, IEEE Transactions of Software Engineering, Vol. 2 No.4, 1987

[16] V. A. Levitin, *How to Measure Software Size, And How To Do Not* , COMPSAC 86

[17] M. Maya, A. Abran, P. Bourque, *Measuring the size of small functional enhancements to software*, 6th International Workshop on Software Metrics, Germany, Sep. 1996

[18] J. A. McCall, P. K. Richards, G. F. Walters, *Factors in Software Quality*, Vols I, II and III, US Rome Air Developmemnt Center, Reports NTIS AD/A-049014, 015, 055, 1977

[19] Perlis, Alan, Sayward, Frederick, Shaw, Mary, *Software Metrics, An Analysis and Evaluation*, The MIT Press, 1981

[20] L. H. Putnam, *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*, IEEE Transactions of Software Engineering, SE-4, July 1978

[21] C. V. Ramamoorthy, F. B. Bastani, *Software Reliability – Status and Perspectives*, IEEE Transactions of Software Engineering, SE-8, July 1982

[22] Roberts, S. Fred, *Measurement Theory with Applications to Decisionmaking, Utility and the Social Sciences,* Encyclopedia of Mathematics and its Applications, Addison Wesley Publishing Company, 1979

[23] H. D. Rombach, T. U. Bradford, *Improving Software Maintenance Through Measurement,* Proc. Of the IEEE, Vol. 77, No. 4, April 1989

[24] R. J. Rubey, R. D. Hartwick, *Quantative Measurement Program Quality*, ACM, National Computer Conference, 1968

[25] Waguespack, J. Leslie, Badlani, Sunil, *Software Complexity Assessment: An Introduction and Annoted Bibliography*, ACM SIGSOFT, Vol 12, No. 4, 1987

[26] M. R. Woodward, M. A. Hennell, Hedley, *A Measure of Control Flow Complexity in Program Text,* IEEE Transactions on Software Engineering, Vol. 5, 1979

[28] [29] M. H. Van Emden, *An Analysis of Complexity*, Mathematisches Zentrum, Amsterdam, 1971

[30] Horst Zuse, *Software Complexity. Measures and Methods*, Walter de Gruyter, Berlin, New York, 1991

ISSN 0208-8028
ISBN 83-85847-59-6

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: bibliote@ibspan.waw.pl