Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 22

XVI Seminarium
ZASTOSOWANIE KOMPUTERÓW W NAUCE I TECHNICE' 2006
Oddział Gdański PTETiS
Referat nr 15

POMIAR ROZMIARU I PRACOCHŁONNOŚCI OPROGRAMOWANIA KOMPUTEROWEGO

Maciej KUCHARSKI

Politechnika Gdańska, Wydział ETI
tel: 58 3471037 fax: 58 3472727 e-mail: mkuchar@eti.pg.gda.pl

Ocena rozmiaru oprogramowania oraz pracochłonności związanej z jego wytworzeniem są niezwykle istotne z punktu widzenia przemysłowego podejścia do jego wytwarzania. Wiedza ta jest niezbędna dla porównywania między sobą różnych projektów informatycznych. Zależność pomiędzy rozmiarem wytworzonego oprogramowania a pracochłonnością jest istotną wskazówką dla skutecznego planowania procesu wytwórczego oraz konstruowania harmonogramów. W referacie skupiono się na fragmencie tego problemu dotyczącą miar oraz technik pomiaru rozmiaru i pracochłonności związanych z realizacją przedsięwzięcia informatycznego. Przedstawiono główne miary rozmiaru i pracochłonności oraz zaprezentowano narzędzie ułatwiające pomiar rozmiaru kodu źródłowego aplikacji napisanych w języku Java.

1. POTRZEBA OCENY ROZMIARU I PRACOCHŁONNOŚCI

Podstawowym problemem stojącym przed każdym projektem wytwórczym jest wiedza dotycząca rozmiaru oprogramowania oraz nakładu pracy ludzkiej potrzebnej do ukończenia projektu. Wiedza ta jest niezbędna dla porównywania między sobą różnych projektów informatycznych. Zależność pomiędzy rozmiarem wytworzonego oprogramowania a pracochłonnością jest istotną wskazówką dla skutecznego planowania procesu wytwórczego oraz konstruowania harmonogramów. Stanowi to podstawę skutecznego zarządzania projektem informatycznym.

Z punktu widzenia zarządzania projektem informatycznym istotne są dwa aspekty określania rozmiaru i pracochłonności:

- Bezpośredni lub pośredni pomiar
- Oszacowanie, czyli przewidywanie przyszłej miary rozmiaru lub nakładu dla wytwarzanego oprogramowania

Pomiar dotyczy istniejącego oprogramowania. Oszacowanie zwykle dotyczy oprogramowania, które dopiero ma być wytworzone. Kluczowe znaczenie dla powodzenia projektu informatycznego stanowi trafna ocena nakładu pracy ludzkiej potrzebnej do wytworzenia oprogramowania przed rozpoczęciem projektu.

Recenzent: Prof. dr hab. inż. Bogdan Wiszniewski - Wydział Elektroniki, Telekomunikacji Informatyki Politechniki Gdańskiej

2. ISTOTA MIAR ROZMIARU ORAZ PRACOCHŁONNOŚCI

Oprogramowanie jest niematerialne, przez co nie ma wymiarów fizycznych. Rozmiar oprogramowania można więc rozumieć w różny sposób. Przy czym N. E. Fenton wskazuje na to, że w zależności od celu, do którego używamy tego pojęcia, możemy je różnie interpretować [1].

Najczęściej miary rozmiaru wykorzystywane są do przewidywania lub oceny pracochłonności oraz efektywności i jakości procesu wytwórczego, czyli inaczej pracy ludzkiej. Prowadzi to do ujmowania rozmiaru oprogramowania jako rozmiaru intelektualnego wkładu człowieka w bezpośrednie wytworzenie oprogramowania, czyli inaczej wolumenu informacji dostarczonej przez człowieka w celu wytworzenia oprogramowania. Takie ujmowanie rozmiaru oprogramowania prowadzi do silnej zależności pomiędzy rozmiarem a pracochłonnością.

Pracochłonność postrzegana jest jako ilość czasu zużytego na wytworzenie oprogramowania. Nakład pracy ludzkiej jest najistotniejszym czynnikiem kosztowym związanym z procesem wytwórczym oprogramowania. Z tego też powodu pracochłonność w ogólnym rozumieniu traktowana jest jako suma czasu pracy wszystkich osób uczestniczących w pracach nad systemem. Odpowiada to definicji nakładu pracy ludzkiej zwanego w skrócie nakładem (ang. effort).

3. ZALEŻNOŚĆ POMIĘDZY ROZMIAREM A NAKŁADEM

Wytworzenie większego oprogramowania wymaga więcej pracy ludzkiej. Oprócz rozmiaru są również inne czynniki wpływające na ilość pracy ludzkiej potrzebnej do wytworzenia oprogramowania. Zależność pomiędzy rozmiarem a nakładem nie jest liniowa. Zgodnie ze stworzonym przez B. Bohema modelem COCOMO zależność pomiędzy rozmiarem oprogramowania a nakładem pracy potrzebnej do jego wytworzenia jest wykładnicza zgodnie ze wzorem [2]:

$$NAKLAD = A \cdot (ROZMIAR)^{k} \cdot f \tag{1}$$

gdzie*: NAKLAD* - liczba osobomiesięcy potrzebna do wytworzenia systemu, *ROZMIAR* – liczba logicznych linii kodu źródłowego, *A* – stała zależna od kalibracji modelu, *k* i *f* - stałe zależne od typu projektu.

W modelu COCOMO oprócz rozmiaru i nakładu pojawia się również wiele innych czynników mających istotny wpływ na nakład pracy ludzkiej, takich jak na przykład poziom umiejętności zespołu wytwórczego.

4. MIARY PRACOCHŁONNOŚCI

Najczęściej stosowaną jednostką nakładu jest osobomiesiąc rozumiany jako wolumen pracy wykonanej przez jedną osobę w ciągu jednego miesiąca. Pewien problem stanowi różna liczba dni roboczych w zależności od miesiąca, a także urlopy oraz inne fluktuacje. Z tego powodu przez osobomiesiąc należy rozumieć jedną dwunastą średniej liczby przepracowanych dni w roku. Liczba ta zależy od uwarunkowań danego kraju.

W przypadku Stanów Zjednoczonych przyjmuje się, że osobomiesiąc składa się z 20 osobodni, czyli 20 dni pracy jednej osoby [3]. Kontynuując, nasuwa się pytanie o definicję osobodnia.

Teoretycznie osobodzień oznacza rozmiar pracy, jaką jedna osoba może wykonać w ciągu jednego dnia. Zwykle dzień pracy trwa osiem godzin. Nie oznacza to jednakże ośmiu

godzin przeznaczonych na wykonywanie zaplanowanych zadań. Na podstawie przeprowadzonych badań dotyczących efektywnej liczby godzin w ciągu ośmiogodzinnego dnia pracy stwierdzono, że pracownik na efektywną pracę poświęca średnio około 6 godzin [3,4].

5. MIARY ROZMIARU OPROGRAMOWANIA

Pojęcie rozmiaru aplikacji można rozumieć w różny sposób. Przy czym w zależności od celu, do którego używamy tego pojęcia, można je różnie interpretować. Rozmiar jest zazwyczaj używany jako kluczowy czynnik niezbędny do określenia nakładu, stosuje się go również w kontekście funkcjonalności, złożoności, redundancji i powtórnego użycia [1]. W zależności od kontekstu użycia inna miara może okazać się bardziej odpowiednia. Stało się to przyczyną powstania wielu miar rozmiaru istotnych z punktu widzenia różnych atrybutów, takich jak: długość kodu źródłowego, długość dokumentów wytwarzanych w różnych fazach rozwoju systemu, funkcjonalność systemu, złożoność kodu źródłowego, rozmiar kodu wynikowego systemu.

Przykładowe miary i metryki rozmiaru aplikacji przedstawiono w tablicy 1.

Tablica 1. Przykładowe metryki rozmiaru.

Miara	Metryka	Przykłady zastosowań
Długość kodu źródłowego	Liczba znaków kodu źródłowego Liczba linii kodu Liczba logicznych linii kodu Liczba klas Liczba metod	Szacowanie nakładu za pomocą metod: COCOMO, SLIM, PSP. Kontrola jakości oraz określanie intensywności zjawisk w kodzie źródłowym, takich jak np. liczba defektów
Długość dokumentu	Liczba stron, liczba znaków, liczba linii	Kontrola jakości dokumentacji np. liczba defektów na stronę.
Funkcjonalność systemu	Liczba Punktów Funkcyjnych Liczba Punktów Obiektowych Liczba przypadków użycia	Szacowanie nakładu na podstawie funkcjonalności przy wykorzystaniu modelu FPA i COCOMO.
Złożoność kodu źródłowego	Złożoność metody Złożoność klasy Miary spójności klasy Miary złożoności powiązań pomiędzy klasami Metryki McCabe'a	Kontrola poprawności modelu obiektowego z punktu widzenia właściwej hierarchii dziedziczenia. Kontrola jakości wytwarzanego kodu.
Rozmiar kodu wynikowego	Rozmiar pliku kodu wynikowego	Określenie rozmiaru nośników potrzebnych do dystrybucji oprogramowania.

W większości przypadków gotowy program powstaje w wyniku transformacji wytworzonego przez człowieka w języku zbliżony do naturalnego kodu źródłowego. Z tego też powodu większość metryk rozmiaru oprogramowania dotyczy właśnie kodu źródłowego.

Liczba linii kodu stanowi najbardziej intuicyjny i zarazem bardzo popularny sposób określania długości kodu źródłowego aplikacji. Stanowi podstawę dla wielu technik szacowania nakładu oraz pozwala na określanie intensywności występowania pewnych zjawisk

w kodzie źródłowym, takich jak np. gęstość defektów, definiowana jako liczba defektów przypadająca na tysiąc linii kodu.

Liczbę linii kodu źródłowego można liczyć na różne sposoby. Najprostszy i najstarszy sposób to zliczanie liczby znaków końca linii. Cechują go jednak pewne niedoskonałości, za sprawą których powstało wiele różnych alternatywnych sposobów określania liczby linii kodu źródłowego. Do najważniejszych problemów związanych ze zliczaniem linii kodu należą:

- Liczba zdań w linii kodu. W większości języków programowania można w jednej linii zawrzeć wiele konstrukcji języka programowania. Z punktu widzenia kompilatora znaki końca linii mogą być zbyteczne. W takiej sytuacji można zawrzeć nawet całą aplikację w jednej bardzo długiej linii. Eliminacji tego problemu służy definicja linii kodu jako zdania w rozumieniu kompilatora. W efekcie powstały dwa sposoby rozróżniania tego elementu: bazujące na znakach końca linii - fizyczne linie kodu, oraz bazujące na elementach składniowych języka programowania - logiczne linie kodu.
- Komentarze, deklaracje, dyrektywy kompilatora etc. Z punktu widzenia pewnych zastosowań ich zliczanie wraz z liniami kodu może być niekorzystne. W innym przypadku zaś jest korzystne.
- Puste linie kodu. Podobnie jak w przypadku komentarzy uwzględnianie pustych linii może być niekorzystne. Dotyczy to zarówno logicznych, jak i fizycznych linii kodu.
- Automatyczne generatory kodu. Wraz z rozwojem narzędzi programistycznych generatory kodu zdobywają sobie szczególną popularność. Metody szacowania nakładu zakładają, że kod wygenerowany automatycznie jest pomijany. Dla innych potrzeb kod ten może być uwzględniany.
- Kwestia ponownego wykorzystania wcześniej napisanego kodu w aplikacji. Kod ten może zostać skopiowany z wcześniejszego projektu. Może zostać również wykorzystany jako biblioteka czy komponent.
- Modyfikacja kodu źródłowego. W trakcie procesu wytwórczego linie kodu mogą być dodawane, usuwane bądź modyfikowane.

W literaturze można znaleźć liczne dyskusje na temat wad i zalet poszczególnych metod pomiarów. W celu uporządkowania i ujednolicenia zasad pomiaru Software Engineering Institute stworzył standard dla liczenia linii kodu [5]. Dokument ten nie precyzuje jednak jednego sposobu pomiaru linii kodu, a raczej sankcjonuje istnienie wielu różnych sposobów pomiaru. Według SEI każda definicja jest dobra pod warunkiem czytelności i użyteczności z punktu widzenia celu, dla którego została stworzona. Standard SEI określa natomiast wszystkie warunki, jakie muszą być spełnione, aby w sposób czytelny i jednoznaczny zdefiniować metodę pomiaru liczby linii kodu.

Wyróżniono dwa typy definicji SLOC:

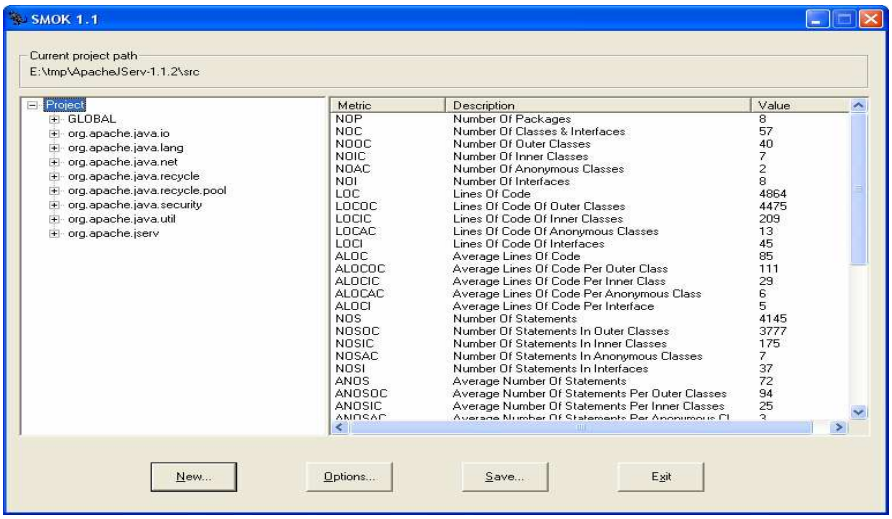
- Fizyczne linie kodu (physical SLOC) bazujące na znakach końca linii.
- Logiczne linie kodu (logical statments lub logical SLOC) bazujące na zdaniach danego języka programowania.

SEI stworzył szablon definicji liczby linii kodu. Za pomocą tego szablonu można jednoznacznie zdefiniować własną miarę rozmiaru. Bazuje on w dużej mierze na listach kontrolnych (ang. checklist), za pomocą których w przejrzysty i czytelny sposób można określić główne atrybuty metody pomiarowej. Lista ta jest listą otwartą i w miarę potrzeby można ją rozszerzyć o dodatkowe elementy.

6. NARZĘDZIE WSPOMAGAJĄCE POMIAR ROZMIARU

Pomiar nakładu pracy ludzkiej wykonywany jest w większości przedsiębiorstw albo ręcznie albo za pomocą systemu do rejestracji czasu pracy.

Pomiar rozmiaru kodu źródłowego jest o wiele bardziej złożony z uwagi na różne sposoby ujmowania rozmiaru. Dodatkowo ręczny pomiar rozmiaru jest czasochłonny i mało dokładny. Jednakże proces ten można w istotnym stopniu zautomatyzować. W tym celu powstał system SMOK mające na celu umożliwienie automatycznego pomiaru różnych metryk rozmiaru. Przykładową formatkę, wytworzonego w ramach prowadzonych przez autora prac badawczych, programu SMOK przedstawiono na rysunku 1.



Rys. 1. System pomiaru metryk kodu źródłowego Javy SMOK.

Program ten aktualnie umożliwia pomiar następujących metryk rozmiaru oraz struktury kodu źródłowego:

- Liczba pakietów
- Liczba klas w rozbiciu na zewnętrzne, wewnętrzne, anonimowe oraz interfejsy.
- Całkowita liczba linii kodu źródłowego
- Liczba linii kodu źródłowego w rozbiciu na kod zwarty w klasach: zewnętrznych, wewnętrznych, anonimowych oraz interfejsach.
- Całkowita liczba logicznych zdań
- Liczba logicznych zdań w rozbiciu na kod zwarty w klasach: zewnętrznych, wewnętrznych, anonimowych oraz interfejsach.
- Głębokość w drzewie dziedziczenia
- Liczba potomków

Wymienione metryki mogą być użyteczne z punktu widzenia budowania modeli przedstawiających zależność nakładu od rozmiaru.

SMOK działa w oparciu o parser kodu źródłowego Javy. Rozpoznaje konstrukcje składniowe języka i na tej podstawie tworzy statystyki. Taka budowa oprogramowania pomiarowego umożliwia łatwe rozszerzanie systemu o nowe metryki. Jest to szczególnie uży-

teczne w przypadku badań naukowych, kiedy na ogół nie ma z góry ustalanego zestawu metryk.

7. PODSUMOWANIE

Podstawą większości badań naukowy są pomiary. Tak samo jest w przypadku oprogramowania. Przedstawiony w referacie program SMOK jest przydatnym i elastycznym narzędziem wspomagającym badania z zakresu rozmiaru i nakładu oraz jakości oprogramowania komputerowego. Program ten wykazał swoją użyteczność przy pracach badawczych dotyczących jakości [6] oprogramowania oraz zależności nakładu pracy ludzkiej od rozmiaru oprogramowania [7]. Można oczekiwać, że narzędzie to wykaże swoją użyteczność również w środowisku przemysłowym.

8. BIBLIOGRAFIA

1. Fenton N. E.: Software Metrics. A Rigorous Approach, International Thomson Computer Press, 1991, ISBN: 1850322422
2 Boehm. B.: Software Engineering Economics. Prentice Hall, Englewood Cliffs, New Jersey, 1981, ISBN: 0138221227
3 Yeates D. Project Management for Information Systems. Pitman Publishing, London 2004, ISBN: 0273685805
4 Gilb T. Software Metrics, Winthrop Publishers, 1977, ISBN: 0876268556
5 Park R.E. Software Size Measurement: A Framework for Counting Source Statements. Software Engineering Institute, 1992, tech. Report SEI-92-TR-20
6 Kaczmarek J., Kucharski M.: Application of Object-Oriented Metrics for Java Programs, Project Control for Software Quality. Proceedings of ESCOM-SCOPE99, Herstmonceux Castle, Shaker Publishing, England 1999
7 Kaczmarek J., Kucharski M.: Size and Effort Estimation for Application Written in Java, Information and Software Technology, Elsevier, 2004,Vol. 46, s. 589-601, ISSN: 0950-5849

SOFTWARE SIZE AND EFFORT MEASUREMENT

Software size and effort measurement is an important matter in software project management. It is used to compare different projects. The knowledge of software size and effort is essential for project planning and scheduling. The paper concentrates on just one aspect of software size and effort: size and effort measurement. It describes size and effort metrics and presents a tool useful for size measurements of Java source code.