Próba eksperymentalnej oceny metody PROBE*

Jerzy R. Nawrocki
Politechnika Poznańska, Instytut Informatyki
Jerzy.Nawrocki@put.poznan.pl

Maciej Szkopek
Politechnika Poznańska, Instytut Informatyki
mszkopek@man.poznan.pl

Bartosz Walter
Politechnika Poznańska, Instytut Informatyki
Bartosz.Walter@cs.put.poznan.pl

Adam Wojciechowski
Politechnika Poznańska, Instytut Informatyki
Adam.Wojciechowski@put.poznan.pl

Streszczenie

Metoda PROBE pozwala na oszacowanie pracochłonności zadania programistycznego na podstawie dotychczas zebranych danych historycznych. W artykule omówiono metodę PROBE oraz eksperyment, którego celem była ocena skuteczności metody, w zależności od liczby oraz charakteru danych historycznych.

1. Wprowadzenie

Szacowanie pracochłonności odgrywa w inżynierii oprogramowania istotną rolę. Jeśli, przystępując do przedsięwzięcia, znalibyśmy jego pracochłonność, wówczas ułożenie odpowiedniego harmonogramu i oddanie projektu na czas

* Praca wykonana w ramach projektu badawczego 43-364/01-BW i częściowo finansowana przez Accenture sp. z o.o.

byłoby znacznie prostsze. Niestety, najczęściej pracochłonność jest wielką niewiadomą i chcąc wygrać kontrakt potencjalny wykonawca, świadomie lub podświadomie, przyjmuje znacznie niedoszacowaną jej wartość, co niesie z sobą szereg negatywnych konsekwencji takich, jak opóźnienia we wdrożeniu, defekty oprogramowania, przekraczanie budżetu, praca w nadgodzinach, stres i niepotrzebne napięcia w zespole wykonawczym.

Aby szacowanie pracochłonności przebiegało sprawnie i dostarczało danych, na których można polegać, musi być ono realizowane nie tylko na poziomie zespołu (czyli na poziomie przedsięwzięcia, wydania produktu, czy jego przyrostu), ale również na poziomie poszczególnych członków zespołu (czyli na poziomie zadań). Jedną z metod szacowania pracochłonności zadań przydzielonych poszczególnym członkom zespołu jest metoda PROBE (*PROxy-Based Estimating method*). Metoda ta została zaproponowana w połowie lat 90-tych przez Wattsa Humphrey'a jako jeden z głównych elementów PSP [1] (*Personal Software Process*). Zakłada ona istnienie bazy danych historycznych, w której znajdują się dane dotyczące m.in. rozmiaru i pracochłonności oprogramowania zrealizowanego przez danego programistę. Humphrey zaproponował zestaw zadań programistycznych, który można wykorzystać do zbudowania takiej bazy [1]. Cechą charakterystyczną metody PROBE jest oparcie się na metodach statystycznych i prezentowanie programiście nie tylko wartości oczekiwanej pracochłonności (jest to tzw. estymata punktowa), ale również przedziału ufności dla przyjętego poziomu ufności p (jest to tzw. estymata przedziałowa i określa przedział pracochłonności, w którym – z prawdopodobieństwem p – znajdzie się faktyczna wartość pracochłonności).

Niestety wciąż jest brak prac dotyczących eksperymentalnej oceny metod szacowania pracochłonności, w tym metody PROBE. W artykule przedstawiono eksperyment przeprowadzony na Politechnice Poznańskiej, którego celem była ocena wybranych aspektów metody PROBE. Celem głównym było zbadanie wpływu zadań programistycznych na jakość bazy danych historycznych. W szczególności interesowało nas pytanie, czy baza danych historycznych zbudowana w oparciu o zadania Humphrey'a może służyć do szacowania pracochłonności aplikacji internetowych. Ponadto, wykorzystując dane zebrane w trakcie eksperymentu, postanowiliśmy zweryfikować, czy estymaty przedziałowe otrzymane metodą PROBE faktycznie określają przedziały, w których rzeczywiste wartości pracochłonności znajdą się z prawdopodobieństwem p odpowiadającym przyjętemu poziomowi ufności.

W następnym rozdziale krótko zaprezentowano metodę PROBE. W kolejnych rozdziałach opisano przeprowadzony eksperyment oraz wykorzystane w nim zadania programistyczne i przedstawiono uzyskane wyniki.

2. Metoda PROBE

Metoda PROBE (*PROxy-Based Estimating method*) służy do szacowania rozmiaru kodu oraz nakładu pracy. Została zaproponowana przez Wattsa Humphrey'a jako element PSP (*Personal Software Process*). Bazuje ona na metodzie standardowego składnika [2] oraz metodzie wartości rozmytych [2]. Wyróżnić można w niej następujące kroki:

1. Projekt koncepcyjny (*conceptual design*)
2. Szacowanie rozmiaru kodu
3. Szacowanie pracochłonności (wraz z przedziałem ufności).

W fazie **projektowania koncepcyjnego** powstaje projekt systemu, którego celem jest umożliwienie szacowania pracochłonności. W trakcie realizacji projekt koncepcyjny może ulec poważnym zmianom, jednak bez takiego projektu szacowanie pracochłonności jest bardzo trudne. W projekcie koncepcyjnym, z punktu widzenia szacowania pracochłonności, najistotniejsze jest wyodrębnienie obiektów i metod służących do komunikacji między nimi. Wyodrębnione obiekty dzielone są na kategorie „funkcjonalne" (przetwarzanie tekstów, graficzny interfejs użytkownika, tzw. obiekty logiczne o skomplikowanych algorytmach przetwarzania, wejście-wyjście, operacje numeryczne itp.)

W kolejnym kroku następuje **szacowanie rozmiaru kodu**. Dla każdego obiektu szacuje się liczbę metod oraz określa się jego rozmiar za pomocą wartości rozmytych (np. duży, średni, mały). Na podstawie danych historycznych, mając rozmyty rozmiar obiektu, liczbę metod i kategorię obiektu wyznacza się rozmiar obiektu w liniach kodu. Suma uzyskanych w ten sposób rozmiarów, stanowi całkowity rozmiar tworzonego oprogramowania, *X*.

Szacowanie pracochłonności obejmuje trzy przypadki dotyczące jakości bazy danych historycznych:

- Przypadek najlepszy. Dostępne są dane (pary: szacowany rozmiar i faktyczna pracochłonność) dla przynajmniej 3 dotychczas wykonanych programów. Współczynnik korelacji *r* między oszacowanymi rozmiarami a faktycznymi wartościami pracochłonności spełnia warunek $r^2 \geq 0{,}5$. Do wyznaczenia estymaty punktowej wykorzystywana jest regresja liniowa (jej współczynniki są wyznaczane na podstawie bazy danych historycznych), natomiast dla wyznaczenia estymaty przedziałowej oblicza się odchylenie standardowe od prostej wyznaczonej przez regresję liniową oraz rozkład t Studenta dla przyjętego poziomu ufności.
- Przypadek pośredni. Dla przynajmniej 3 dotychczas wykonanych programów mamy *faktyczny* rozmiar i faktyczną pracochłonność. Korelacja *r* między tymi wartościami spełnia warunek: $r^2 \geq 0{,}5$. Sposób postępowania jest analogiczny jak w przypadku najlepszym z tym, że do otrzymanych wyników musimy podchodzić z głęboką rezerwą.

- Przypadek najgorszy. Nie jest spełniony żaden z wcześniejszych warunków, ale dostępne są pary *faktyczny* rozmiar – faktyczna pracochłonność dla przynajmniej jednego dotychczas wykonanego programu. Na podstawie tych wartości wyznacza się minimalną i maksymalną prędkość kodowania i dzieląc oszacowany rozmiar przez te wartości uzyskujemy górną i dolną granicę przedziału.

3. Opis eksperymentu

W eksperymencie udział wzięli studenci Politechniki Poznańskiej studiujący na 3 roku, kierunku informatyka. Na eksperyment złożyły się cztery spotkania, w trakcie których studenci rozwiązywali jedno bądź dwa zadania. Dwa spotkania poświęcono na zbieranie danych historycznych. W tym celu wykorzystano formularze podobne do proponowanych przez Humphrey'a. Podczas kolejnych spotkań, dodatkowo wykorzystano metodę PROBE do obliczenia spodziewanej pracochłonności, na podstawie oszacowanego rozmiaru kodu źródłowego oraz dotychczas zebranych danych historycznych.

Eksperyment poprzedzony został krótkim szkoleniem dotyczącym języka Java. Przeprowadzono wykład wprowadzający oraz dwa ćwiczenia laboratoryjne, w trakcie których studenci rozwiązywali proste zadania programistyczne. Zbieranie danych historycznych rozpoczęto od drugich zajęć laboratoryjnych. Na podstawie jakości dostarczonych rozwiązań oraz obserwacji dokonanych w czasie trwania zajęć, wybrano spośród 110 osób grupę 40 najlepiej przygotowanych do eksperymentu studentów. Studentów podzielono na dwie dwudziestoosobowe grupy. Podział dotyczył rodzaju zadań służących do zbierania danych historycznych. Pierwsza grupa, WH, budowała swoje bazy danych historycznych na podstawie zadań Wattsa Humphrey'a [1], natomiast druga grupa, MS, wykorzystała w tym celu zadania zaproponowane przez jednego z autorów artykułu, specjalnie przygotowane pod kątem Internetu oraz języka Java. Chodziło o zbadanie wpływu zadań programistycznych na jakość bazy danych historycznych. W grupie WH dziesięciu programistów pracowało indywidualnie a dziesięciu innych rozwiązywało zadania parami. Podobnie było w grupie MS.

4. Zadania programistyczne

W eksperymencie wykorzystano zestaw 8 zadań programistycznych. Trzy zadania zostały zaczerpnięte ze zbioru proponowanego przez Humphrey'a [1]. Pięć pozostałych zadań zostało opracowanych na potrzeby eksperymentu. Poniżej przedstawiono skróconą wersję tych zadań. Ich pełna treść jest dostępna w [3,4]. Zadania przedstawione w rozdziałach 4.1 i 4.2 służyły do zbudowania bazy danych historycznych i miały charakter „próbki uczącej". Grupa WH budowała

swoje bazy danych historycznych w oparciu o zadania z rozdziału 4.1, natomiast grupa MS wykorzystywała do tego celu zadania z rozdziału 4.2. Zadania przedstawione w rozdziale 4.3 były wspólne dla obu grup i miały sprawdzić wpływ charakteru zadań (WH i MS) na jakość otrzymanych danych historycznych.

4.1. Zadania Humphrey'a

Jak łatwo zauważyć, wybrane zadania Humphrey'a mają charakter numeryczny i różnią się zasadniczo od pozostałych zadań rozwiązywanych w ramach eksperymentu.

Zadanie 1

Napisz w języku Java metodę obliczającą parametry β_0 oraz β_1 regresji liniowej dla szacowania rozmiaru kodu źródłowego. Parametry obliczane są na podstawie dostępnych zbiorów danych historycznych x i y, według podanych wzorów. Korzystając z obliczonych parametrów oraz podanej wartości oszacowanego rozmiaru obiektu (*object LOC*), metoda ma wyznaczyć spodziewany całkowity rozmiar programu (*actual new and changed LOC*).

Zadanie 2

Napisz w języku Java metodę obliczającą wartość współczynnika korelacji dwóch zbiorów danych o liczności n oraz wartość t (rozkład t Studenta), według podanych wzorów.

Zadanie 3

Napisz w języku Java metodę wykonującą całkowanie zgodnie z regułą Simpson'a. Program powinien być tak zaimplementowany, aby istniała możliwość całkowania różnych funkcji, bez modyfikacji kodu metody. Należy zaimplementować funkcję rozkładu normalnego. W zadaniu podana została również treść metody Simpson'a.

4.2. Zadania zorientowane na aplikacje internetowe

Zadanie 1

Napisz w języku Java dwie metody analizujące tekst. Pierwsza z implementowanych metod dzieli wejściowy łańcuch znaków na zdania (separatorem jest znak kropki). Druga metoda znajduje w pierwszym wejściowym

łańcuchu znaków słowa rozpoczynające się od liter wymienionych w drugim łańcuchu znaków.

Zadanie 2

Napisz w języku Java aplet, który pobiera z pliku HTML parametry wejściowe określające cechy figury geometrycznej. Jeśli wszystkie parametry obowiązkowe zostały poprawnie odczytane, aplet powinien narysować na środku dostępnego obszaru, zdefiniowaną parametrami wejściowymi, figurę.

Zadanie 3

Napisz w języku Java aplet (rozkład komponentów w kontenerze apletu jest dany). Aplet powinien wczytać parametry wejściowe z pliku HTML. Naciśnięcie przycisku „Zamień" powoduje zamianę, w tekście wpisanym przez użytkownika, wystąpień słowa zdefiniowanego w pierwszym parametrze HTML, słowem określonym przez drugi parametr HTML.

4.3. Zadania wspólne

Zadanie 1

Napisz w języku Java aplet (rozkład komponentów w kontenerze apletu jest dany), który po naciśnięcia przycisku „Statystyka" oblicza liczbę słów oraz znajduje słowo występujące najczęściej w polu *TextArea*. Wielkość liter nie jest istotna, wynikiem jest dowolne ze słów występujących najczęściej. Następujące znaki są traktowane jako separatory słów: znak nowej linii, tabulacji, spacji, kropka oraz przecinek.

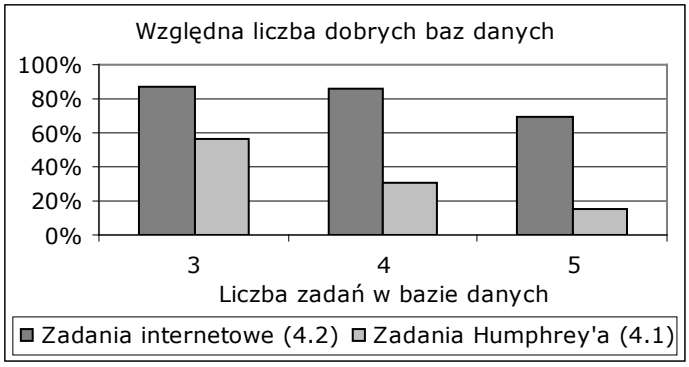
Zadanie 2

Napisz w języku Java aplet (rozkład komponentów w kontenerze apletu jest dany), który w odpowiedzi na zmianę stanu jednego z komponentów typu *Choice* bądź *Checkbox*, definiujących parametry figury geometrycznej wyświetlanej przez komponent typu *Canvas*, aktualizuje zawartość komponentu *Canvas*.

5. Wyniki eksperymentu

5.1. Wpływ charakteru zadań na jakość bazy danych historycznych

Dobra baza danych historycznych to taka, która umożliwia wykorzystanie najlepszego wariantu metody PROBE, czyli musi spełniać warunek, iż współczynnik korelacji r między szacowanym rozmiarem oprogramowania a faktyczną pracochłonnością jest nie mniejszy niż $1/\sqrt{2}$ (czyli $r^2 \geq 0{,}5$).



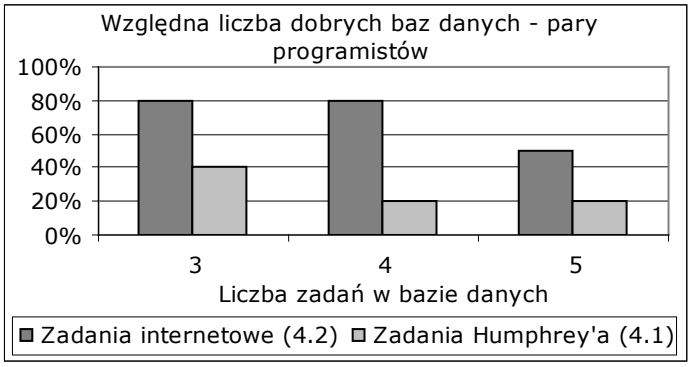
Wykres 1. Wpływ zadań na liczbę dobrych baz danych historycznych.

Wykres 1 przedstawia wpływ zadań na liczbę dobrych baz danych historycznych dla grupy WH i MS (grupa WH korzystała z zadań przedstawionych w rozdz. 4.1 i 4.3, natomiast grupa MS z zadań opisanych w rozdziałach 4.2 i 4.3). Z wykresu tego można wyciągnąć dwa wnioski:

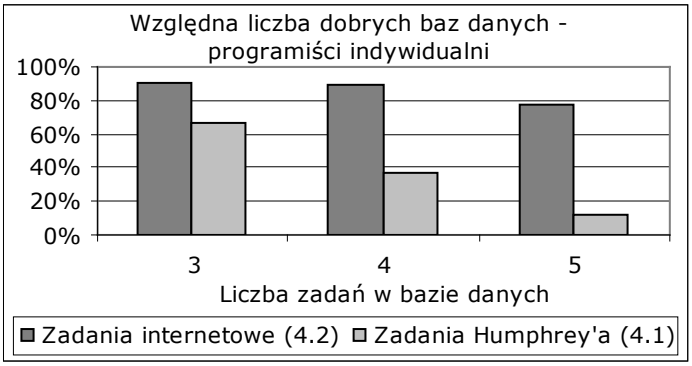
- Charakter zadań programistycznych ma istotny wpływ na jakość bazy danych historycznych (warto podkreślić, że poziom trudności zadań przedstawionych w rozdziałach 4.1, 4.2 i 4.3 jest mniej więcej ten sam).
- Im więcej zadań jest w bazie danych tym gorsza ich jakość w tym sensie, że zmniejsza się prawdopodobieństwo uzyskania dobrej bazy danych. Jest to obserwacja bardzo zaskakująca i niewątpliwie wymaga kontynuowania eksperymentu na szerszą skalę. Co ciekawe, zjawisko to dotyczy zarówno zadań Humphrey'a jak i zadań internetowych.

Na wykresach 2 i 3 przedstawiono wpływ zadań na liczbę dobrych baz danych historycznych dla par programistów oraz programistów pracujących indywidualnie. Z wykresów tych wynika, iż wcześniej zaobserwowane tendencje

są zachowane zarówno w przypadku programowania indywidualnego, jak i programowania parami, które jest jedną z głównych praktyk XP [5,6]. Dodatkowo warto zauważyć, że liczba dobrych baz danych jest, w większości przypadków, mniejsza dla par programistów niż dla programistów pracujących indywidualnie, co było dla nas kolejnym zaskoczeniem.



Wykres 2. Wpływ zadań na liczbę dobrych baz danych historycznych dla par programistów.

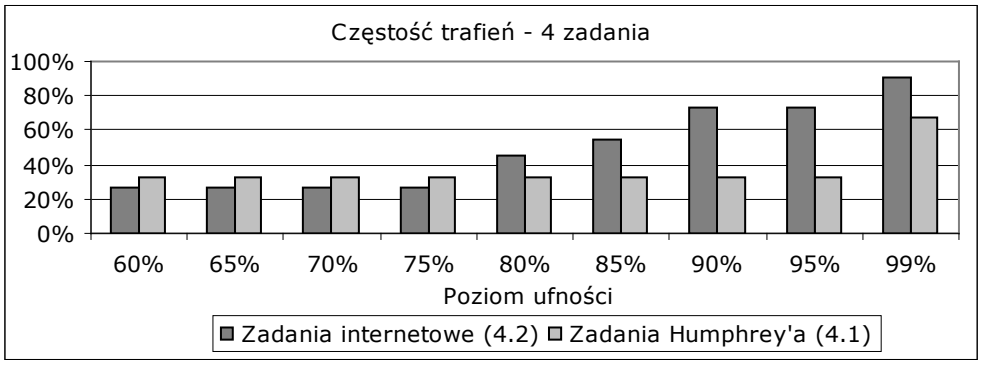


Wykres 3. Wpływ zadań na liczbę dobrych baz danych historycznych dla programistów pracujących indywidualnie.

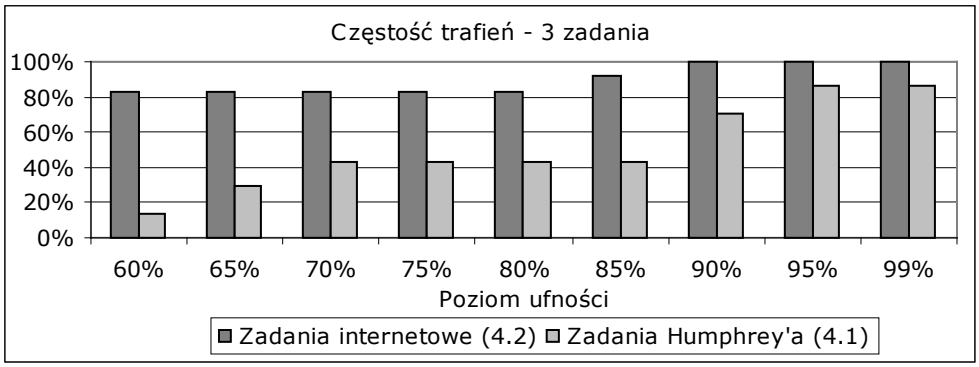
5.2. Problem poziomu ufności

Jak wcześniej wspomniano, w metodzie PROBE oprócz estymaty punktowej otrzymuje się także estymatę przedziałową, która dla założonego poziomu ufności *p*, określa przedział, w którym rzeczywista pracochłonność

znajdzie się z prawdopodobieństwem *p*. Analizując wyniki eksperymentu zauważyliśmy, że w niektórych przypadkach, częstość trafień (czyli takich zdarzeń, w których rzeczywista pracochłonność znajduje się w przedziale wyznaczonym przez estymatę przedziałową) jest istotnie niższa od założonego poziomu ufności.



Wykres 4a. Wpływ poziomu ufności *p*, na częstość trafień dla 4 zadań.



Wykres 4b. Wpływ poziomu ufności *p*, na częstość trafień dla 3 zadań.

Na wykresach, 4a i 4b przedstawiono częstość trafień dla baz danych historycznych zawierających, odpowiednio, 4 i 3 zadania. Z wykresu 4a wynika, że w przypadku 4 zadań, częstość trafień jest zawsze istotnie niższa od poziomu ufności z tym, że w miarę wzrostu poziomu ufności zadania internetowe charakteryzują się wyższą częstością trafień niż zadania Humphrey'a. W przypadku bazy danych historycznych zawierającej tylko 3 zadania, częstość trafień dla zadań internetowych jest zawsze powyżej poziomu ufności, natomiast dla zadań Humphrey'a, również i w tym przypadku, częstość trafień jest poniżej poziomu ufności.

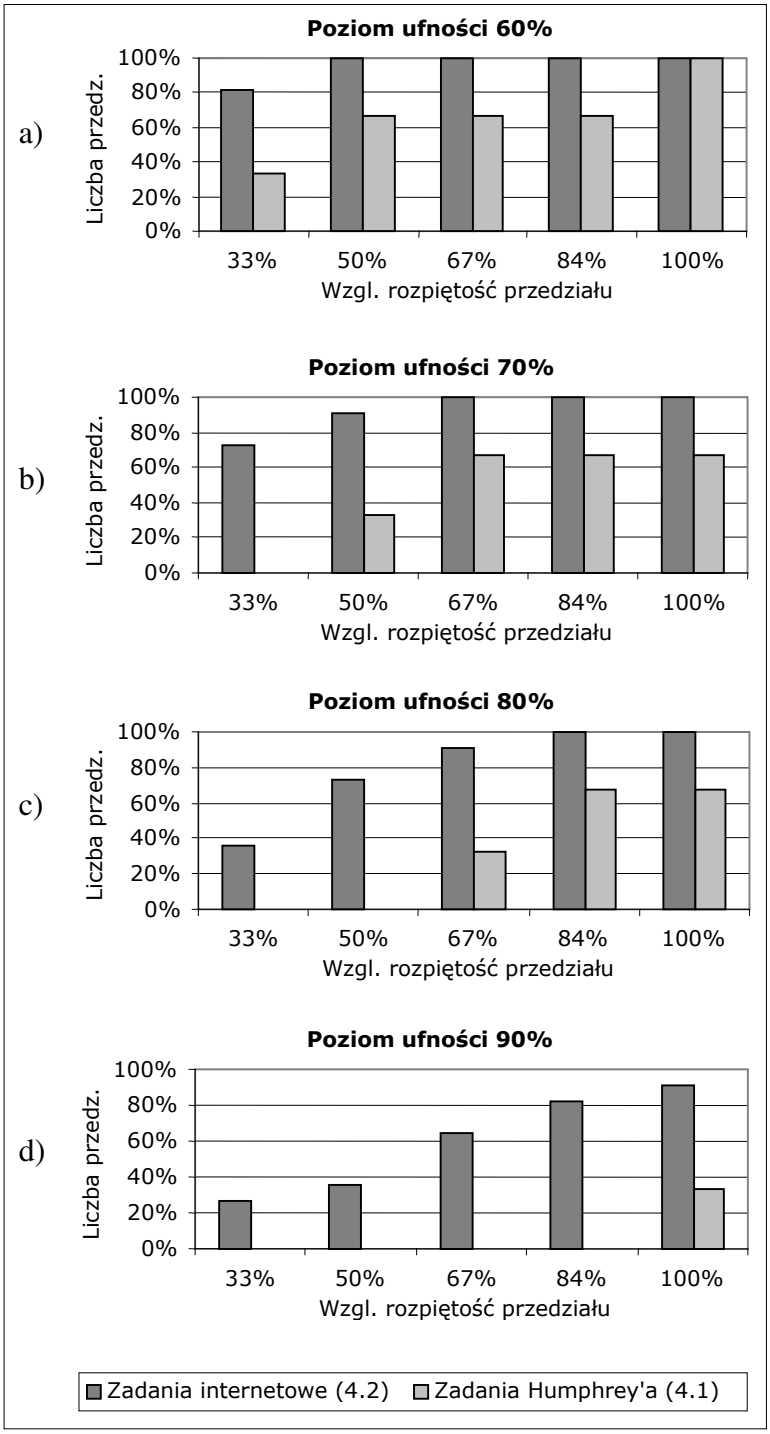
Patrząc na te wykresy można wysnuć przypuszczenie, że częstość trafień, zarówno dla zadań internetowych jak i dla zadań Humphrey'a, maleje wraz z liczbą zadań znajdujących się w bazie danych historycznych. Prawdopodobnie jest to zjawisko przejściowe i należy oczekiwać, że od pewnej liczby zadań, częstość trafień zaczyna rosnąć wraz ze wzrostem liczby zadań w bazie danych historycznych. Wymaga to jednak dalszych eksperymentów.

5.3. Dobór poziomu ufności

W metodzie PROBE istotną rolę odgrywa dobór parametru zwanego poziomem ufności. Poziom ufności przyjmuje wartości od 0 do 100%. Im większa jest jego wartość tym większe jest prawdopodobieństwo, że rzeczywista pracochłonność znajdzie się w wyznaczonym przedziale. Jednak nadmierny wzrost poziomu ufności powoduje tak duży wzrost przedziału, że „gubi się" wartość informacyjną. Można, na przykład, przyjąć poziom ufności równy 100%, ale jest to bezcelowe, gdyż otrzymalibyśmy przedział od 0 do nieskończoności, co jest równoważne stwierdzeniu, że rzeczywista pracochłonność może mieć dowolną wartość. Pojawia się więc problem takiego doboru poziomu ufności, który byłby na tyle wysoki, iż zapewniałby wysoką częstość trafień a z drugiej strony otrzymany przedział (estymata przedziałowa) byłby stosunkowo wąski i w ten sposób stanowiłby cenną informację na etapie planowania.

W metodzie PROBE przedział ufności ma postać $[E-\Delta, E+\Delta]$, gdzie E jest oczekiwaną wartością pracochłonności, natomiast Δ wyznacza szerokość przedziału i jest obliczana według odpowiednich wzorów. Przez względną rozpiętość przedziału rozumiemy wartość $\delta=100\%*\Delta/E$. Na przykład, jeśli $E=1000$, $\delta=50\%$ to mamy przedział od 500 do 1500. Dla danego poziomu ufności, bazy danych historycznych i szacowanej pracochłonności można wyznaczyć przedział ufności $[E-\Delta, E+\Delta]$. Mając bazy danych historycznych pochodzące od różnych programistów można obliczyć dla ilu spośród nich wartość $\delta=100\%*\Delta/E$ była nie większa niż założony poziom rozpiętości przedziału *d*. Wartość tę będziemy nazywać **liczbą nie za szerokich przedziałów** dla przyjętej wartości *d*.

Na wykresach 5a – 5d przedstawiono względną liczbę nie za szerokich przedziałów dla przyjętej wartości rozpiętości przedziału *d* równej 33%, 50%, 67%, 84% i 100% i dla poziomów ufności 60%, 70%, 80% i 90%, przy 4 zadaniach w bazie danych historycznych. Z wykresów tych wynika, że dobry kompromis między prawdopodobieństwem trafienia w przedział a informacyjną wartością przedziału osiąga się dla poziomu ufności równego 80%. Z wykresu 5c wynika, że dla poziomu ufności 80% aż w 73% przypadków względna rozpiętość przedziału δ była nie większa niż 50%.



Wykres 5. Względna liczba nie za szerokich przedziałów dla różnych wartości względnej rozpiętości przedziału, przy poziomie ufności 60% (a), 70% (b), 80% (c) i 90% (d).

6. Podsumowanie

W pracy przedstawiono wyniki eksperymentu stanowiącego próbę oceny skuteczności metody PROBE w szacowaniu pracochłonności. Rezultaty eksperymentu wskazują, że charakter zadań programistycznych ma istotny wpływ na jakość bazy danych historycznych oraz na częstość trafień rzeczywistej pracochłonności w przedział wyznaczony metodą PROBE. Zaobserwowano również dwa zjawiska o charakterze anomalii:

- Liczba dobrych baz danych historycznych (czyli takich, dla których korelacja między szacowanym rozmiarem a faktyczną pracochłonnością jest większa niż $1/\sqrt{2}$) była mniejsza dla 4 zadań niż dla 3. Generalnie należy się spodziewać, że w miarę wzrostu liczby zadań prawdopodobieństwo uzyskania dobrej bazy danych powinno rosnąć. Prawdopodobnie jest to zjawisko przejściowe i zanika przy odpowiednio dużej liczbie zadań.
- Dla 4 zadań częstość trafień rzeczywistej pracochłonności w przedział wyznaczony metodą PROBE była znacznie poniżej przyjętego poziomu ufności (patrz wykres 4a). Być może również i w tym przypadku zjawisko to ma charakter przejściowy i zanika przy odpowiednio dużej liczbie zadań.

Opisane eksperymenty dotyczyły również doboru poziomu ufności, który byłby odpowiednim kompromisem między prawdopodobieństwem trafienia rzeczywistej pracochłonności w obliczony przedział ufności a wartością informacyjną przedziału, która miałaby praktyczne znaczenie na etapie planowania. Z danych otrzymanych w trakcie eksperymentu wynika, że optymalna wartość poziomu ufności p jest na poziomie 80%.

Przyszłe eksperymenty powinny nie tylko dotyczyć większych baz danych historycznych (8-10 zadań), ale powinny również mieć dłuższy horyzont czasowy, obejmujący 5 do 10 dni roboczych (28 – 56 rzeczywistych godzin pracy), co bardziej by odpowiadało rzeczywistym problemom planistycznym.

Bibliografia

[1]. Humphrey W. S., *A Discipline for Software Engineering*, Addison-Wesley Publishing Company, 1995.

[2]. Putnam L. H., Myers W., *Measures for Excellence: Reliable Software on Time, within Budget*, Yourdon Press, 1992.

[3]. *http://www.man.poznan.pl/~mszkopek/exp/*, Maj 2001.

[4]. Szkopek M., *Szacowanie nakładu pracy związanej z pisaniem oprogramowania w Javie*, Politechnika Poznańska, Poznań, 2001.

[5]. Beck K., *Extreme Programming Explained*, Addison-Wesley, 2000.

[6]. Nawrocki J., Wojciechowski A., *Experimental Evaluation of Pair Programming*, Proceedings of the 12th European Software & Metrics Conference ESCOM'01, London, 2001, 269-276.