Planowanie pracy zespołu programistów z użyciem metody PROBE

Jerzy R. Nawrocki <Jerzy.Nawrocki@put.poznan.pl>*
Bartosz Walter <Bartosz.Walter@cs.put.poznan.pl>
Adam Wojciechowski <Adam.Wojciechowski@put.poznan.pl>

Instytut Informatyki Politechniki Poznańskiej
60-965 Poznań, ul. Piotrowo 3A

Streszczenie

Metoda PROBE, będąca elementem *Personal Software Process*^{SM}, służy do szacowania rozmiaru i czasu budowy oprogramowania w przypadku realizacji systemu przez indywidualnego programistę. Metoda ta nie zawsze znajduje pełne zastosowanie przy planowaniu projektów realizowanych przez zespoły. Dlatego zaproponowano sposób wspomagania przydziału osób do zadań oparty o analizę danych historycznych zaczerpniętych z modelu PSP^{SM}, opisujących kompetencje poszczególnych członków zespołu.

1. Wprowadzenie

Planowanie jest tą częścią fazy strategicznej, która decyduje o powodzeniu projektu informatycznego zanim on się jeszcze rozpocznie. Rezultat planowania wpływa na cały przebieg projektu, jego koszt, ryzyko z nim związane, a przede wszystkim na sposób realizacji.

W każdym procesie programowym, zarówno klasycznym, opartym o obieg dokumentów, jak CMM (*Capability Maturity Model*) [6] czy lekkim, który

* Praca wykonana w ramach projektu badawczego 43-364/01-BW

rezygnuje z większości artefaktów, jakim jest XP (*eXtreme Programming*) [1], planowanie pozostaje istotnym czynnikiem powodzenia projektu.

Model CMM[SM], zaproponowany przez Software Engineering Institute, przewiduje 5 poziomów dojrzałości procesu programowego. Na drugim poziomie (zarządzanym – *managed*; dawniej ten poziom nosił nazwę powtarzalnego – *repeatable*) jednym z kluczowych obszarów procesu (*Key Process Area*) jest właśnie planowanie oparte na dokumentach i ich wzorcach.

Idea programowania ekstremalnego (XP) stanowi odpowiedź na problemy związane z brakiem elastyczności dobrze zdefiniowanego, klasycznego procesu budowy oprogramowania. Producenci oprogramowania nie nadążali za gwałtownie zmieniającymi się potrzebami klientów, a obszerne dokumenty wprowadzały istotny narzut na pracę włożoną w produkcję oprogramowania. Mimo tego, że XP odrzuca większość klasycznych technik związanych z klasycznym procesem programowym, to jeśli chodzi o planowanie pozostaje ono jedną z najważniejszych praktyk.

Modele procesu programowego dla pracy indywidualnej (*Personal Software Process*[SM]) i zespołowej (*Team Software Process*[SM]) zaproponował W. Humphrey [4, 5]. Oba opierają się na uproszczonych formularzach, które stanowią jednocześnie źródło informacji o procesie i prostą metodę jego oceny. W zakresie planowania TSP[SM] opiera się na PSP[SM] i obejmuje następujące zagadnienia:

- szacowanie pracochłonności oprogramowania,
- układanie harmonogramu prac,
- opracowanie struktury projektu (zespołu realizacyjnego),
- przydział zadań do osób.

TSP[SM], przeznaczony dla małych zespołów, jest rozwinięciem PSP[SM]. Zakłada on, że każdy z członków zespołu stosuje PSP[SM], czyli planuje pracę we własnym zakresie. Oba modele wymagają kolekcjonowania danych dotyczących oszacowań i faktycznych wielkości osiągniętych w trakcie dotychczasowej pracy.

Humphrey do analizy statystycznej zebranych danych zaproponował metodę PROBE, która szacuje wysiłek indywidualnego programisty wymagany przy tworzeniu przez niego oprogramowania o przewidywanym rozmiarze. Z punktu widzenia TSP[SM] metoda PROBE ma następujące wady:

- nie uwzględnia sytuacji, gdy baza danych historycznych dotyczy zróżnicowanych zadań programistycznych, nawet jeśli zostały one zimplementowane w tym samym języku,
- brak jest mechanizmów, które umożliwiłyby transfer wiedzy między członkami zespołu nt. pracochłonności programowania. Może się na przykład zdarzyć, że mamy w zespole osobę, która zebrała swoje dane historyczne dotyczące kilkudziesięciu programów języka Java i mamy też nowicjusza. W

klasycznej wersji metody PROBE dane zgromadzone przez „weterana" Javy są bezużyteczne z punktu widzenia nowicjusza,

- metoda PROBE jest zorientowana na pojedynczego programistę i nie obejmuje możliwości zmniejszenia pracochłonności projektu poprzez odpowiednie przypisanie zadań do członków zespołu.

Celem artykułu jest zaproponowanie modyfikacji metody PROBE, która ułatwiłaby jej stosowanie do planowania przedsięwzięć zespołowych. Proponowane modyfikacje obejmują:

- wprowadzenie metody pomiaru kompetencji poszczególnych osób w zakresie stosowania pewnej technologii, obejmujące doświadczenie, wiedzę i umiejętność radzenia sobie z podobnymi problemami,
- zastosowanie metody przydziału osób do zadań, opartej na analizie zgromadzonych przez członków zespołu danych historycznych.

Drugi rozdział jest poświęcony wykorzystanym w zaproponowanej metodzie przydziału metodom szacowania: metodzie PROBE i jej zastosowaniu w PSP^{SM} oraz Wideband-Delphi. Kolejny rozdział przedstawia proponowaną metodę opisu kompetencji programistów, profilu zadań oraz przydziału zadań do osób w przedsięwzięciach zespołowych z uwzględnieniem danych historycznych. Rozdział czwarty stanowi podsumowanie przedstawionych rozwiązań, formułuje wnioski oraz kierunki dalszych prac.

2. Proste metody szacowania

2.1. Metoda Probe

Metoda PROBE została zaproponowana przez W. Humphreya jako część indywidualnego procesu budowy oprogramowania PSP^{SM} [4].

PROBE (*PROxy Based Estimation*) służy do szacowania wysiłku wymaganego przy tworzeniu wybranych komponentów (*proxies*) programowych na podstawie zebranych danych historycznych. Idea oparcia estymacji o znormalizowany komponent (którym w przypadku PROBE jest obiekt) umożliwia stworzenie intuicyjnego elementu systemu, którego rozmiar można stosunkowo łatwo oszacować metodami statystycznymi.

Metoda PROBE składa się z dwóch zasadniczych kroków:

- znalezienia współczynników regresji liniowej B_0 i B_1 pomiędzy szacowanym i faktycznym rozmiarem na zbiorze programów w danych historycznych;

dzięki temu jest możliwe znalezienie zależności między tymi dwiema wielkościami w dotychczas wytworzonym oprogramowaniu,

- oszacowania pracochłonności (czasu) potrzebnego do stworzenia oprogramowania o szacowanym rozmiarze.

Oba kroki zakładają wykorzystanie regresji liniowej; jednak jej stosowanie jest obwarowane warunkami: zbiór danych historycznych musi być odpowiednio liczny (zawierać co najmniej 3 punkty) oraz dane te muszą spełniać zależność

$$r^2 \geq 0.5 \tag{2.1}$$

Niestety, nie zawsze te warunki są spełnione. Ogólnie możliwe są trzy warianty:

- **A** (najlepszy): zbiór danych historycznych musi zawierać co najmniej 3 pary *szacowany rozmiar – faktyczny czas*, dla których spełniony jest warunek (2.1) ,
- **B**: relacja (2.1) zachodzi nie między szacowanym rozmiarem i faktycznym czasem, ale między *faktycznym rozmiarem* i *faktycznym czasem realizacji,*
- **C** (najgorszy): nie jest spełniony żaden z powyższych warunków; w tym przypadku nie stosuje się regresji liniowej, tylko oblicza szacowaną pracochłonność na podstawie średniego rozmiaru oprogramowania tworzonego w jednostce czasu.

Oszacowanie uzyskane w każdym z przypadków jest estymatą punktową, która nie zawiera informacji o jakości tego oszacowania. Aby to osiągnąć, oblicza się za pomocą dwustronnego rozkładu *t* Studenta przedział ufności, w którym z zadanym prawdopodobieństwem znajduje się estymowana wielkość.

Metoda PROBE dzięki prostemu aparatowi matematycznemu i stosunkowo niewielkiej ilości koniecznych danych historycznych stanowi rozwiązanie dla pojedynczych programistów. W przypadku zespołów pojawia się problem szacowania pracochłonności dla całego zespołu oraz podziału pracy między członków zespołu. Proponowane rozwiązanie zostanie zaprezentowane w rozdziale 3.

2.2. Metoda Wideband-Delphi

Metoda Wideband-Delphi (delficka), zaproponowana przez B. Boehma, jest zupełnie innym podejściem do szacowania rozmiaru oprogramowania niż PROBE. Jest ona szczególnie przydatna w sytuacjach, gdy posiadane doświadczenie nie jest usystematyzowane w postaci zbioru danych historycznych.

Metoda delficka opiera się na szacowaniu rozmiaru lub pracochłonności przez ekspertów. Istotą metody są spotkania, w trakcie których eksperci dyskutują o systemie (ale nie o estymatach!), zgłaszają uwagi, a następnie przekazują swoje oszacowania prowadzącemu. Ten je zbiera i zestawia, jednak bez identyfikowania autorów poszczególnych wyników. Następuje dyskusja, a potem po raz kolejny każdy ekspert dokonuje oceny rozmiaru lub pracochłonności. Ta procedura jest powtarzana do czasu osiągnięcia założonego przedziału rozbieżności lub wykorzystania przeznaczonego na szacowanie czasu.

Elementem, który decyduje o skuteczności metody delfickiej, jest anonimowość oszacowań: dzięki temu nie istnieje psychologiczny przymus obrony własnego zdania i eksperci łatwiej dokonują rewizji swoich estymat. Współdzielenie wielu ocen daje efekt synergii, która często przynosi dobre rezultaty.

Wadą metody delfickiej jest duża zależność od indywidualnych cech osób szacujących, a także prowadzącego dyskusję. Umiejętność kierowania uwagi ekspertów na system i jego cechy, a nie konkretne oceny, decyduje o wzajemnym zrównoważeniu tendencji w indywidualnych wynikach i – w efekcie – skuteczności szacowania.

3. Metoda przydziału zadań do osób

Obie przedstawione w rozdziale 2 mają na celu uzyskanie lepszych oszacowań rozmiaru i pracochłonności. Jednak planowanie wymaga również m.in. wykonania harmonogramu prac oraz przydzielenia zadań do osób.

TSP^{SM}, proces zespołowej budowy oprogramowania, zakłada dokonywanie podziału zadań w trakcie negocjacji i uzgodnień wewnątrz zespołu. Podejście takie, mimo swej prostoty, może przynieść w efekcie nierównomierne obciążenie członków zespołu pracą oraz przydział zadań do osób daleki od optymalnego.

W praktyce członkowie zespołu mają często przydzielone na stałe role, które pełnią w projektach. W przypadku osób nie zajmujących się produkcją oprogramowania takie przypisanie jest uzasadnione: zarządzanie konfiguracją czy nadzór jakości jest zwykle pomocniczą rolą w projekcie i nie narusza istotnie podziału pracy. Programiści jednak często pracują zamiennie z wykorzystaniem różnych technologii, co powoduje, że nie mogą specjalizować się w jednej technologii czy wykonywać wyłącznie zadania jednego typu.

Powstaje potrzeba liczbowego, opartego na statystyce, sposobu szacowania kosztu i czasu realizacji systemu w zależności od przydziału poszczególnych zadań do osób. Potrzebna jest też metoda znajdowania przydziału dającego w najlepsze (najtańsze) rozwiązanie.

3.1. Ocena kompetencji

Szacowanie w metodzie PROBE jest oparte na założeniu, że programista ma do wykonania jedno zadanie, którego pracochłonność jest jednorodna: niezależnie od istoty zadania prędkość jego rozwiązywania jest zawsze taka sama.

Takie założenie jest prawdziwe jedynie gdy programista stosujący PSP^{SM} pracuje wciąż nad bardzo podobnymi problemami. W praktyce zadania programistyczne są bardziej złożone, wymagają nie tylko umiejętności programistycznych (o co zapewne chodziło twórcy PSP^{SM}), ale także m.in. wiedzy dziedzinowej problemu, doświadczenia w projektowaniu czy znajomości technologii. Dlatego przyjmowanie, że każdy programista tworzy oprogramowanie z tą samą prędkością, nie zawsze jest uzasadnione.

Pierwszym elementem proponowanej metody jest ocena kompetencji programistów na podstawie danych historycznych. Programowanie obecnie nie oznacza wyłącznie umiejętności tworzenia kodu w danym języku, ale przede wszystkim znajomość technologii, algorytmiki czy istoty problemu, który należy rozwiązać. W związku z tym wyznaczanie średniej produktywności programisty na podstawie wszystkich zrealizowanych projektów może być niewystarczające; konieczne staje się stworzenie modelu kompetencji, który będzie uwzględniał stopień znajomości poszczególnych technologii, doświadczenia, wiedzy dziedzinowej.

Prostym modelem spełniającym to kryterium jest *profil kompetencji*, czyli wektor czynników, które decydują o wiedzy i doświadczeniu programisty. Każdemu z tych czynników można przypisać wartość liczbową oznaczającą poziom kompetencji w danej dziedzinie: im wyższa wartość tego atrybutu, tym poziom kompetencji wyższy; nowicjusz otrzymuje wartość 0 w odpowiednim polu wektora.

Otrzymany w rezultacie wektor liczb naturalnych o długości i stanowi prosty opis poziomu wiedzy i doświadczenia programisty. Znacznie trudniejsze jest jednak jego stworzenie w praktyce: należy zdefiniować zbiór czynników istotnych przy ocenie kompetencji, a następnie procedurę oceny i zakres możliwych wartości oceny. Przy tworzeniu profili programistów pomocne są przedstawione w rozdziale 2 metody PROBE (w przypadku posiadania dostatecznej ilości danych liczbowych) lub Wideband-Delphi (gdy danych brak i konieczna jest ocena ekspercka). Ponieważ zdobycie wiarygodnych danych liczbowych wymaga olbrzymiej ilości pomiarów, niezbędne może okazać się przeprowadzenie sesji testowych, w trakcie których programiści rozwiązują zadania wymagające kompetencji w danej dziedzinie; następnie dane te są przetwarzane zgodnie z metodą PROBE.

Profil kompetencji będzie ulegał modyfikacjom w czasie: zmieniać się będzie liczba czynników istotnych dla skutecznego tworzenia oprogramowania oraz ich wartości – badana osoba będzie nabierała doświadczenia, uczyła się,

zwiększała swoje możliwości. Procedura włączania nowych danych do profilu zależy od potrzeb zespołu lub organizacji.

3.2. Przydział zadań i wyznaczanie czasochłonności projektu

Profil kompetencji programisty opisuje średnią prędkość, z jaką pracuje on nad poszczególnymi artefaktami programowymi, np. czyta i przyswaja dokumentację techniczną, projektuje hierarchię obiektów czy pisze procedurę numeryczną w Pascalu. Do wyznaczenia czasu wymaganego do realizacji zadania przez konkretnego programistę potrzebna jest jeszcze metoda opisu *profilu zadania*.

Profil zadania jest wektorem tych samych czynników (a więc także wektorem o tej samej długości), co w przypadku profilu kompetencji. Tutaj jednak wartości liczbowe określają nie *średnią prędkość tworzenia* artefaktów związanych z daną technologią, ale szacowany *rozmiar* tych artekfaktów. Tworzenie profilu zadania jest równie trudne, co w przypadku kompetencji programistów: należy dla każdego zadania określić z jakich elementów (artefaktów) się ono składa oraz oszacować ich rozmiar za pomocą metod PROBE lub Wideband-Delphi, w zależności od ilości i jakości dostępnych danych historycznych.

Najistotniejszym krokiem w procedurze przydziału zadań jest dopasowanie profilu kompetencji programisty do profilu zadania. Ponieważ kolejne elementy profili odpowiadają sobie, można zdefiniować wiele metod sprawdzających ich dopasowanie.

Prostym rozwiązaniem jest obliczenie wektora ilorazów odpowiadających sobie elementów profilu zadania Z przez elementy wektora profilu kompetencji P, czyli:

$$\forall k : T_k = \frac{Z_k}{P_k} \tag{3.2.1}$$

Suma ilorazów wszystkich elementów stanowi *szacowany czas* t_i realizacji zadania Z przez programistę P, czyli wielkość decydującą o przydziale osób do zadań w przedsięwzięciu..

Po obliczeniu szacowanego czasu realizacji wszystkich zadań Z wchodzących w zakres projektu przez wszystkich programistów P powstaje macierz o wymiarach $p \times z$, gdzie p oznacza liczbę programistów, natomiast z – liczbę zadań do wykonania.

Problem przydziału zadań może zostać zdefiniowany na różne sposoby, w zależności od przyjętego celu. W dalszej części przyjęto, że jest nim

minimalizacja sumarycznego czasu realizacji projektu przy założeniu braku ograniczeń kolejnościowych.

Wówczas można go opisać następująco:

$$cel: \min T = \sum_{i=1}^{p} \sum_{j=1}^{z} T_{ij} x_{ij}$$

$$p.o.$$

$$\sum_{i=`}^{p} x_{ij} = 1 \qquad (3.2.2)$$

$$\forall i: \sum_{j=1}^{z} T_{ij} x_{ij} \leq M_i$$

Pierwsze ograniczenie oznacza, że każde zadanie musi zostać wykonane w całości, natomiast drugie, że każdy programista dysponuje określonym *budżetem czasowym* M_i – liczbą godzin, które może poświęcić na realizację projektu. Chroni to osoby o wysokich kompetencjach w wielu dziedzinach przed przydziałem większości zadań (choć byłoby to rozwiązanie optymalne).

W zależności od dopuszczalnych wartości zmiennych x_{ij} (ciągłych lub binarnych) problem jest zwykłym problemem programowania liniowego (rozwiązywalnym w czasie wielomianowym) lub problemem programowania zero-jedynkowego (obliczalnym w czasie wykładniczym). Wartości przyjmowane przez zmienne x_{ij} oznaczają część zadania j, które powinien wykonać programista i (dla zmiennych ciągłych) lub przydział zadania j albo brak przydziału (dla zmiennych binarnych) do programisty i. T_{ij} jest szacowanym na podstawie profili kompetencji i zadania czasem realizacji zadania j przez programistę i (por. wzór 3.2.1).

Problem można rozwiązać w zależoności od wartości przyjmowanych przez zmienne x_{ij} metodą simpleks (w przypadku zmiennych ciągłych) lub heurystyką (gdy x_{ij} przyjmują wartości binarne).

Otrzymana w rozwiązaniu (macierz X zmiennych x_{ij} opisuje optymalny przydział pracowników do zadań, to znaczy taki, aby suma czasów realizacji wszystkich zadań w projekcie była minimalna. Jednak ponieważ w definicji zadania nie uwzględniono wielu istotnych czynników wpływających na przydział (np. harmonogramu realizacji czy ograniczeń kolejnościowych między zadaniami), otrzymane rozwiązanie można traktować wyłącznie jako metodę wspomagającą podjęcie decyzji, a nie ją zastępującą.

W efekcie cała zaproponowana metoda szacowania składa się z 4 kroków:

1. podziału całego systemu na zadania programowe o zdefiniowanych profilach i oceny rozmiaru artefaktów związanych z danym zadaniem,

2. wyznaczenia za pomocą ilorazów czynników profilu zadania i profilu kompetencji szacowanego czasu realizacji każdego zadania przez każdego z programistów oraz zdefiniowanie zadania programowania matematycznego,
3. rozwiązania problemu programowania liniowego lub binarnego,
4. ewaluacji otrzymanego rozwiązania, traktując je jako punkt wyjścia do ostatecznego przydziału.

Opisana procedura wymaga wielu danych liczbowych zgromadzonych w poprzednich systemach oraz starannej pielęgnacji danych o profilach, jednak pozwala ona na oparte na statystyce znalezienie optymalnego przydziału zadań.

4. Podsumowanie

Przedstawiona metoda może znaleźć zastosowanie w zespołach stosujących procesy PSP^{SM} i TSP^{SM}. Opiera się ona o proste i łatwe (choć bardzo liczne) w uzyskaniu dane oraz wspomaga planowanie pracy zespołu. Pomaga w opracowaniu optymalnego przydziału zadań opartego o statystyczną analizę danych oraz liczbowy opis kompetencji członków zespołu.

Wadą jest przede wszystkich ilość niezbędnych do skutecznego szacowania danych. W przypadku ich braku (lub danych o niskiej jakości, nie spełniających warunki 2.1) zastosowanie metody delfickiej do oszacowań znacznie obniża weryfikowalną wartość oszacowania. Dlatego metoda ta powinna być stosowana przede wszystkim jako wsparcie decyzji do ostatecznego przydziału zadań, natomiast finalną decyzję musi podjąć kierownik zespołu.

Doskonalenie metody wymaga podjęcia prac w kilku kierunkach.. Przede wszystkim należy uogólnić problem przydziału zadań do programistów, ujmując sytuacje często spotykane w praktyce, a nie opisane tutaj, np. konieczność koordynacji wielu przedsięwzięć czy uwzględnienie czynnika edukacyjnego przy realizacji projektu. Ponadto można zmodyfikować cel przydziału zadań z minimalizacji sumarycznego czasu pracy do minimalizacji czasu trwania projektu (czyli z uwzględnieniem zrównoleglenia prac).

Interesujące byłyby także badania nad możliwością skalowania metody PROBE (stanowiącej w przedstawionej metodzie podstawowe źródło informacji o profilach kompetencji i zadania), czyli szacowania pracochłonności oprogramowania o znacząco różnym rozmiarze niż systemy, dla których są dostępne dane historyczne. Ważna jest także odpowiedź na pytanie, czy jest możliwe odtwarzanie profilu kompetencji na podstawie niepełnych danych, co znacznie zmniejszyłoby uzależnienie od kompletności bazy danych historycznych.

Liczbowe określenie podstawowych parametrów przedsięwzięcia i możliwości jego realizacji ma duże znaczenie dla skutecznego planowania. Zaproponowana metoda stanowi proste rozwinięcie PROBE dla systemów realizowanych przez niewielkie zespoły zbierające i wykorzystujące dane historyczne.

Bibliografia

[1]. K. Beck, M. Fowler "Planning Extreme Programming", Addison-Wesley, 2000

[2]. J. Błażewicz, W. Cellary, R. Słowiński, J. Węglarz "Badania operacyjne dla informatyków", PWN, Warszawa XXX

[3]. Z. Hellwig "Elementy rachunku prawdopodobieństwa i statystyki matematycznej", PWN, Warszawa 1995

[4]. W. S. Humphrey "Discipline for Software Engineering", Addison-Wesley, 1995

[5]. W. S. Humphrey "Introduction to the Team Software Process^{SM}", Addison-Wesley, 1999

[6]. M. C. Paulk et al. „The Capability Maturity Model: Guidelines for Improving the Software Process", Addison-Wesley, 1999