

Programowanie Parami: Razem czy osobno?

Michał Jasiński*

Politechnika Poznańska, Instytut Informatyki
ul. Piotrowo 3a, 60-965 Poznań

Streszczenie. Lekkie metodyki podkreślają znaczenie komunikacji między programistami. Niektóre z nich (np. XP czy Crystal Clear) zalecają programowanie parami. Istnieją jednak dwa podejścia do programowania w parach: klasyczne, znane z XP oraz „ramię-w-ramię” (ang. *Side-by-Side*) zaproponowane w metodyce Crystal Clear. Artykuł prezentuje eksperymenty opisujące porównanie tych dwóch podejść. Uczestnikami eksperymentu było 25 studentów Informatyki IV i V roku studiów, którzy w kontrolowanym środowisku przepracowali 6 dni tworząc rozbudowane aplikacje internetowe w technologii Java z wykorzystaniem narzędzi Eclipse, MySQL i Tomcat. Uzyskane rezultaty wskazują, iż programowanie „ramię-w-ramię” jest bardzo interesującą alternatywą dla pracy parami, przede wszystkim ze względu na niższą pracochłonność (20% dla programowania SbS i 50% dla klasycznego programowania parami).

1. Wprowadzenie

W tradycyjnym podejściu do rozwoju oprogramowania, zadania programistyczne (np. stworzenie pewnego komponentu zgodnie z podaną specyfikacją) przypisywane jest pojedynczemu programiście (patrz np. [9]). Zapewnienie jakości napisanego w ten sposób kodu realizowane jest poprzez przeglądy (mogą to być inspekcje, formalne, przeglądy techniczne, etc. [16]).

Kent Beck, twórca i popularyzator Programowania Ekstremalnego (ang. *Extreme Programming, XP*), zaproponował w swojej metodyce nieco inne podejście nazwane programowanie parami. [2]. W programowaniu parami, jak sama nazwa wskazuje, zadanie przypisywane jest parze programistów wyposażonych w jeden komputer. Podczas gdy jeden z programistów pisze kod, drugi obserwuje, zadając pytania, czy proponując przypadki testowe (zapewniając w ten sposób tzw. ciągły przegląd).

Efektywność programowania parami była obiektem badań wielu studiów. Pierwsze eksperymenty dotyczące programowania parami zostały opisane przez Johna Noska [14]. Według uzyskanych przez niego wyników pary potrzebowały na ukończenie zadania około 30% mniej czasu niż pojedynczy programiści, ale pracochłonność związana z programowaniem parami była o 40% wyższa. Najbardziej optymistyczne rezultaty zostały uzyskane przez Laurie Williams [21, 22]. Według wyników

* We współpracy z Jerzym Nawrockim, Łukaszem Olkiem i Barbarą Lange

2 Programowanie Parami: Razem czy osobno?

przeprowadzonych przez nią eksperymentów przyspieszenie związane z programowaniem parami wynosiło 40%, zaś pracochłonność była zaledwie o 20% wyższa niż w przypadku programistów indywidualnych. Z kolei rezultaty eksperymentów przeprowadzonych na Politechnice Poznańskiej w 2001 roku [13] okazały się bardziej pesymistyczne. W tym przypadku przyspieszenie uzyskane dzięki pracy w parach wynosiło 20% więcej, zaś pracochłonność była o 60% wyższa niż w przypadku osób pracujących samodzielnie.

W 2005 roku Alistair Cockburn zaproponował nieco inne podejście do „programowania w parach” [7]. Podejście to nazwał programowaniem „ramię-w-ramię” (ang. *Side-by-Side, SbS*). W SbS zadanie jest przydzielane parze programistów, jednak w odróżnieniu od programowania parami – każdy programista dysponuje własnym komputerem. Takie podejście pozwala programistom dzielić zadania na mniejsze części i przydzielać je do członków dwuosobowych zespołów (podobnie, jak ma to miejsce w tradycyjnym podejściu). Główną różnicą między SbS, a podejściem klasycznym jest fizyczna odległość członków pary (poprawiająca komunikację) oraz jedność celu (programiści SbS, pracując wspólnie nad tym samym zadaniem są za nie odpowiedzialni). Niestety, jak dotąd nie przeprowadzono eksperymentów, które porównywałyby wydajność programowania parami i SbS.

Celem niniejszej pracy jest prezentacja wyników eksperymentu, który wypełnia tę lukę porównując programowanie SbS z programowaniem parami. W rozdziale 2 przedstawiono aspekty metodologiczne związane z eksperymentami programistycznymi. Kolejne rozdziały zawierają opis eksperymentu (Rozdział 3) oraz analizę jego wyników (Rozdział 4). Istotną kwestią dotyczącą eksperymentów z programowaniem parami jest zaangażowanie członków pary w proces rozwoju oprogramowania (istnieje ryzyko, że w rzeczywistości tylko jedna osoba będzie rozumiała kod, podczas gdy druga będzie tylko obserwatorem). Ten aspekt omówiono w Rozdziale 5. Ponadto, zadaliśmy uczestnikom eksperymentu kilka pytań dotyczących ich wrażeń i ocen. Wyniki ankiet omówiono w Rozdziale 6.

2. Opis eksperymentu

2.1. Uczestnicy i środowisko

Opisywany eksperyment został przeprowadzony na Politechnice Poznańskiej pomiędzy lutym i kwietniem 2005. Uczestnikami eksperymentu było 30 ochotników: studentów studiów magisterskich na specjalnościach Inżynieria Oprogramowania, Projektowanie i Eksploatacja Systemów Informatycznych (IV i V rok). Wszystkie osoby miały za sobą gruntowną wiedzę programistyczną zdobytą podczas zajęć na studiach inżynierskich i magisterskich (w tym programowanie Java i tworzenie aplikacji internetowych) obejmujących w sumie ponad 400 godzin. Uczestnicy eksperymentu mieli także za sobą 12 miesięczny projekt inżynierski, w którym pełnili rolę programistów i testerów.

Uczestnicy eksperymentu pracowali w kilku laboratoriach komputerowych pod opieką pracowników naukowych. Każdy zespół wyposażony był w taki sam komputer

PC (Pentium IV, 512 MB RAM). Środowisko rozwoju aplikacji składało się z pakietu środowiska eksperymentalnym. Nadzorujący ich opiekunowie zajmowali się także zapewnianiem jakości – poprzez testy akceptacyjne. Uczestnicy eksperymentu pracowali tak długo, dopóki zadanie nie zostało w pełni zakończone (tj. wszystkie przypadki testowe nie zostały spełnione). Żadne dodatkowe kryteria jakościowe Java J2SDK 1.4.2_06, Tomcat 5.0.18, Eclipse 3.1, serwera bazy danych MySQL 4.0.21 oraz repozytorium kodu źródłowego na serwerze CVS.

2.2. Proces

Proces eksperymentu składał się z pięciu faz: Praca Domowa, Przygotowanie, Selekcja, Rozgrzewka oraz Eksperyment. W każdej z nich uczestnicy mieli za zadanie zrealizować jedno lub więcej zadań programistycznych.

W fazie *Praca Domowa* zadaniem uczestników było wykonanie zadania programistycznego, które dotyczyło narzędzi, środowiska i technologii (Java, Tomcat, Eclipse), które były wykorzystywane w kolejnych etapach. Prace prowadzone były w warunkach domowych. Celem fazy było wyrównanie poziomu umiejętności w zakresie technologii i narzędzi wśród uczestników eksperymentu.

Podczas *Przygotowania* uczestnicy pracowali nadal indywidualnie, jednak tym razem w rzeczywistym poza testami akceptacyjnymi nie były stosowane. Głównym celem tej fazy, poza utrwalenie wiedzy z zakresu narzędzi i technologii, było zaznajomienie uczestników z procesem wykorzystywanym podczas eksperymentu (testy akceptacyjne, pomiar czasu, etc.). Przygotowanie trwało jeden dzień. Średni czas ukończenia zadania wyniósł 522 minuty.

Faza *Selekcji* zajęła kolejny dzień. Uczestnicy eksperymentu pracowali indywidualnie w laboratoriach nad zadaniami programistycznymi. Zmierzyliśmy czas, który upłynął od rozpoczęcia tej części eksperymentu do poprawnego zakończenia zadania (spełnienia wszystkich testów akceptacyjnych). W tym przypadku średni czas zakończenia wyniósł 370 minut. Na podstawie uzyskanych danych podzieliśmy uczestników na trzy grupy: Pary1, Pary2 oraz Indywidualistów. W kolejnej fazie grupy oznaczone symbolem Pary1 i Par2 miały pracować w parach (z wykorzystaniem różnych podejść do pracy w zespołach), podczas gdy grupa Indywidualistów składała się z osób pracujących samodzielnie, stanowiących grupę referencyjną. Niestety, nie od dziś wiadomo, że między programistami występują znaczące różnice w szybkości programowania (patrz np. [8, 17]). Dlatego też, aby mieć możliwość porównywania wyników dotyczących różnych stylów programowania, musieliśmy się upewnić, że wszystkie grupy będą (średnio) jednakowo szybkie. W rezultacie powstał następujący problem: jak podzielić $5n$ uczestników eksperymentu, których szybkość programowania jest znana, na podzbiór Pary1 (zawierający $2n$ elementów), Pary2 (zawierający również $2n$ elementów) oraz podzbiór Indywidualistów (n elementów) w taki sposób, aby średni czas w każdym z podzbiorów był (niemal) taki sam. Z 30 ochotników 25 osób zakończyło z sukcesem fazę *Selekcji* i zostało przydzielonych do trzech wcześniej wspomnianych grup o licznosci $n=5$ (do opracowania przydziału o żądanych własnościach wykorzystano heurystykę, która najlepszy dała wynik).

4 Programowanie Parami: Razem czy osobno?

Naszym celem było porównanie klasycznego programowania parami (znanego z XP – 1 komputer na parę) z programowaniem „ramię-w-ramię” (SbS – 2 komputery na parę). W związku z tym, *Rozgrzewkę* wykorzystaliśmy, aby nauczyć uczestników eksperymentu pracy w modelach XP oraz SbS. Faza ta trwała dwa dni. Każdy z nich rozpoczynał się 20-minutowym szkoleniem z metodyki pracy opartym o miniaturę procesu [6] (zadaniem uczestników była rozwiązanie zagadek programistycznych). Po zakończeniu sesji szkoleniowej, uczestnicy otrzymali do realizacji zadanie polegające na stworzeniu aplikacji internetowej (które zajęło pozostałą część dnia). W pierwszym dniu trwania tej fazy jedna z grup pracujących w parach stosowała model pracy XP, w drugiej zaś programiści programowali „ramię-w-ramię”. W kolejnym dniu procesy w grupach były zamieniane. Średni czas zakończenia zadań wyniósł odpowiednio 323 i 370 minut.

Jednakże najistotniejszą była faza *Eksperymentu*, która trwała ostatnie dwa dni. Celem tej fazy było zebranie danych, które pozwoliłyby porównać programowanie parami i programowanie „ramię-w-ramię” z punktu widzenia czasu realizacji prac i pracochłonności (rozumianej jako czas pracy w przypadku Indywidualistów i dwukrotność czasu prac w przypadku par). Uczestnicy pracowali w analogiczny sposób jak podczas *Rozgrzewki*, z wyjątkiem braku sesji szkoleniowych. Ponownie w pierwszym dniu Pary1 pracowały według modelu XP, zaś drugiego dnia programowały w podejściu SbS (podczas gry Pary2 – dokładnie odwrotnie). Dzięki takiemu podejściu dodatkowo zminimalizowaliśmy ryzyko wystąpienia niezbalansowanych par w grupach (ze względu na fakt, że czas realizacji jest zmienną niezależną, pozwoliło to wesprzeć wykonane w tym samym celu działania w fazie *Selekcji*). Średni czas uzyskany w tej fazie wynosił 335 minut w pierwszym i 491 minut w drugim dniu eksperymentu.

2.3. Zadania programistyczne

Coraz więcej programistów pracuje nad aplikacjami internetowymi. Dlatego też podjęliśmy decyzję, że właśnie ten najbardziej reprezentatywny typ problemu zostanie wykorzystany w naszym eksperymencie. Uczestnicy pracowali, więc nad aplikacjami internetowymi w języku Java, wykorzystującymi technologie serwletów i JSP.

W fazie *Przygotowania* uczestnicy eksperymentu mieli za zadanie zaimplementować serwis internetowy zabezpieczony mechanizmem autentykacji wykorzystującym bazę danych. Funkcje aplikacji obejmowały m.in. procedurę logowania wraz z szeregiem usług umożliwiających zarządzanie użytkownikami (wyświetlanie listy użytkowników, dodanie użytkownika, usunięcie użytkownika).

Faza *Selekcji* była oparta na dwóch zadaniach. Pierwszym z nich była prosta aplikacja do zarządzania dokumentami. Dokumenty mogły być określane jako publicznie dostępne lub prywatne. Dokumenty publiczne były dostępne dla wszystkich użytkowników aplikacji, podczas gdy dokumenty prywatne mogły być przeglądane wyłącznie przez zarejestrowanych w systemie użytkowników (po zalogowaniu się w aplikacji). Drugie zadanie polegało na implementacji systemu obsługi konferencji (zbieranie streszczeń i pełnych artykułów, recenzowanie).

Podczas *Rozgrzewki* i właściwego *Eksperymentu* (które w sumie trwały 4 dni) uczestnicy przyrostowo rozwijali system Papers-Online (rozszerzenie drugiego zadania z fazy *Selekcji*). Zadanie obejmowało następujące funkcje: obsługa ról autora, recenzenta i przewodniczącego konferencji, nadsyłanie streszczeń i pełnych artykułów przez autorów, rejestracja recenzentów i przypisanie prac do recenzji przez przewodniczącego konferencji, nadsyłanie recenzji przez recenzentów, przeglądanie statusu artykułów przez autorów, stworzenie programu konferencji przez jej przewodniczącego.

Każde zadanie składało się z przypadków użycia [5] opisujących wymagane funkcje systemu oraz z zestawu testów akceptacyjnych. Więcej szczegółów jest dostępnych w [10].

3. Analiza czasu i pracochłonności

Analiza czasu pracy

Każdemu zadaniu programistycznemu odpowiadał zestaw testów akceptacyjnych. Każdego dnia dokonywaliśmy pomiaru *czasu pracy*, to jest czasu, który upłynął od początku sesji eksperymentalnej do jej poprawnego zakończenia (poprawnie przetworzone testy akceptacyjne). Od otrzymanej wartości odejmowaliśmy czas trwania przerwy na obiad.

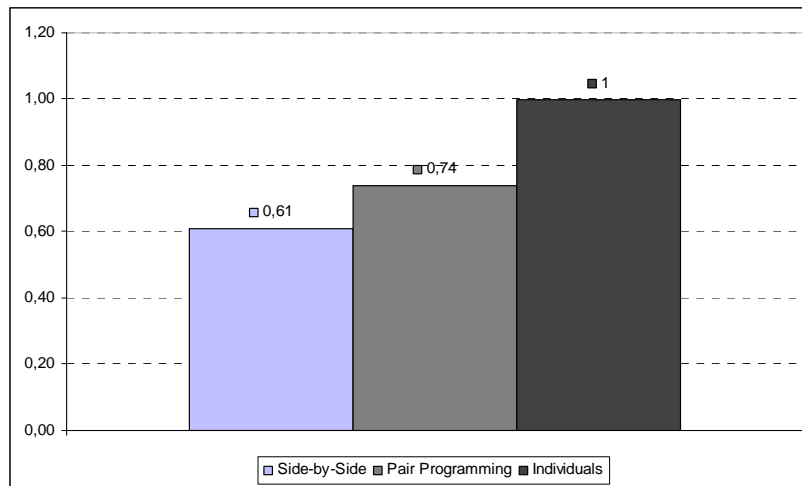
Właściwy eksperyment trwał dwa dni. Jak już wspomniano wcześniej, zarówno osoby przypisane do grupy Pary1, jak i Pary2 programowały zarówno w modelu XP, jak i w modelu SbS. Najprostszym rozwiązaniem byłoby zorganizowanie sesji XP i sesji SbS, odpowiednio pierwszego i drugiego dnia. Niestety, spowodowałoby to problem związany z przydziałem zadań programistycznych. W opisanym przypadku, jeśli zadania byłyby takie same, wówczas drugiego dnia programiści powielaliby dokładnie te same prace, które wykonali dzień wcześniej, a to prowadziłoby do błędnych rezultatów (pary SbS okazałyby się bardziej efektywne niż są w rzeczywistości). Z kolei, jeśli zadania programistyczne byłyby różne i, na przykład, pierwsze z zadań okazałyby się łatwiejsze niż drugie, wówczas pary XP wypadłyby bardziej korzystnie, co również doprowadziłoby do niewłaściwych wniosków.

Aby przezwyciężyć ten problem, zdecydowaliśmy się na rozwiązanie, w którym pierwszego dnia Pary1 pracowały w modelu XP, zaś Pary2 programowały zgodnie z podejściem SbS. Drugiego dnia obie grupy otrzymały nowe zadanie i zamieniły się rolami: Pary1 programowały zgodnie z modelem SbS, zaś Pary2 stosowały XP. Ponieważ zadanie rozwiązywane pierwszego dnia różniło się od zadania z dnia drugiego, zdecydowaliśmy się zastosować w naszych obliczeniach *relatywny czas pracy*, tzn. stosunek czasu spędzonego na pracy przez pary do średniego czasu spędzonego przez programistów pracujących samodzielnie (jeśli zadanie wykorzystane pierwszego dnia było o 30% bardziej pracochłonne niż zadanie rozwiązywane dnia drugiego, średni czas pracy programistów indywidualnych również będzie o 30% wyższy w przypadku pierwszego dnia niż drugiego, co skompensuje długie czasy zakończenia z pierwszego dnia pracy).

6 Programowanie Parami: Razem czy osobno?

Średnie względne czasy zakończenia dla par SbS i XP oraz dla programistów indywidualnych przedstawia Rysunek 1 (oczywiście, średni względny czas zakończenia dla programistów indywidualnych wynosi 1). Jak wynika z wykresu, model SbS jest szybszy niż XP i wynik ten jest istotny statystycznie na poziomie 0.15 (to znaczy, że prawdopodobieństwo przyjęcia hipotezy, że SbS jest szybsze od XP, podczas gdy tak nie jest wynosi 70%). Kolejny wniosek to przewaga SbS i XP nad programistami indywidualnymi na poziomie istotności równym 0.05.

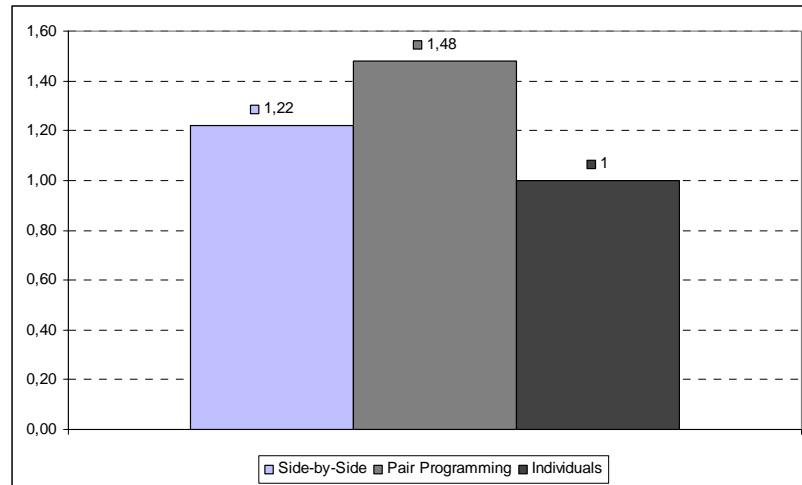
Powyższa analiza statystyczna oparta jest na założeniu, że rozkład analizowanych danych jest normalny (lub wystarczająco bliski rozkładowi normalnemu). Dlatego też konieczna jest weryfikacja, czy dane spełniają ten warunek. W tym celu wykorzystaliśmy test Shapiro-Wilka (SW) [18]. Przeprowadzony test potwierdził, że zebrane dane (zarówno zebrane bezpośrednio, jak i względne) mają rozkład normalny w przypadku XP, SbS oraz programistów indywidualnych (poziom istotności 0.05).



Rysunek 1. Średni względny czas pracy dla indywidualistów, par XP i par SbS.

Analiza pracochołności

W przypadku programistów indywidualnych *pracochołność* jest równa czasowi pracy. Natomiast w przypadku par pracochołność jest równa dwukrotności czasu pracy. Z powodów opisanych powyżej posłużyliśmy się *pracochołnością względną*, to jest stosunkiem pracochołności w danym stylu programowania do średniej pracochołności dla programistów indywidualnych. Jak przedstawia Rysunek 2, pracochołność par XP jest wyższa niż par SbS z istotnością statystyczną na poziomie 0.15.



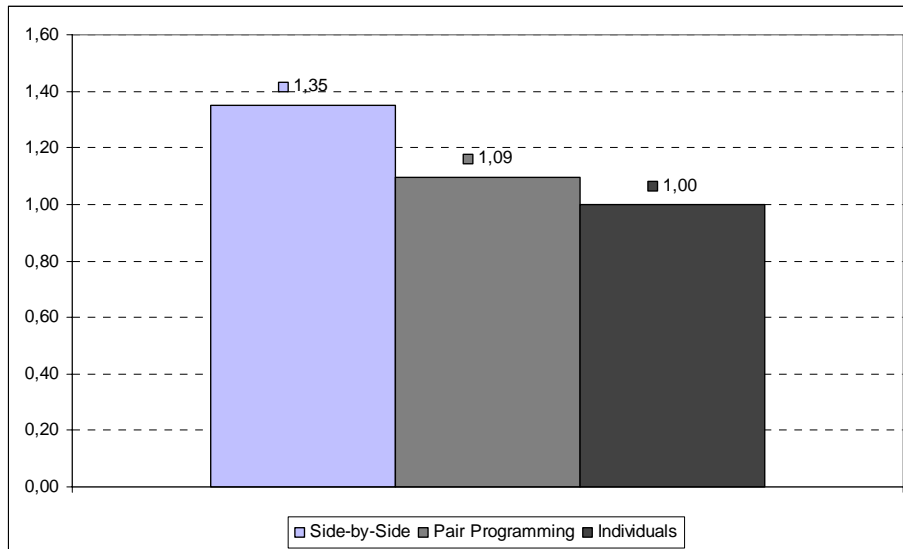
Rysunek 2. Średnia względna pracochołność dla indywidualistów oraz par SbS i XP.

4. Analiza znajomości kodu

W przypadku XP dwójka programistów cały czas pracuje razem: kiedy jeden pisze kod, drugi prowadzi tzw. ciągłą inspekcję ("w locie"). Pary SbS dysponują dwoma komputerami, dzięki czemu mogą dzielić się zadaniami w inny, wygodny dla nich sposób. Powstaje zatem pytanie, jak praca w danym modelu organizacji pary wpływa na znajomość kodu aplikacji przez partnera z zespołu. Aby zweryfikować ten aspekt współpracy wprowadziliśmy dodatkowy (*post-mortem*) krok podczas realizacji zadań. Na koniec każdego dnia wszyscy uczestnicy eksperymentu otrzymywali mikro zadanie do indywidualnej implementacji (niewielka zmiana w aplikacji, taka sama dla wszystkich uczestników eksperymentu). Następnie mierzony był czas ukończenia tego zadania oraz wyznaczany względny czas zakończenia (jako stosunek czasu zakończenia dla indywidualnych programistów – w tym członków par z grup Pary1 i Pary2, pracujących indywidualnie w fazie *post-mortem* – do średniego czasu zakończenia członków grupy programistów indywidualnych). Rysunek 3 przedstawia średni względny czas zakończenia zadania sprawdzającego znajomość kodu dla programistów pracujących w parach XP, parach SbS oraz indywidualnie (w tym przypadku „programowanie indywidualnie” odnosi się do pracy nad zadaniem głównym, nie zaś zadaniem *post-mortem*). Jak łatwo zauważyć, zrozumienie kodu partnera w przypadku grup SbS jest gorsze niż dla XP (w sensie średniego czasu wymaganego do opracowania zmiany). Jednakże poziom istotności, przy którym hipotezy tej nie można odrzucić (równy 0.25) jest dość wysoki, co czyni ją raczej słabą. Rysunek 3 wskazuje także, że czas zakończenia zadania dla XP jest wyższy niż dla programistów indywidualnych. Oznacza to, że w parach XP znajomość kodu jest niższa niż w przypadku indywidualistów. Jednakże należy pamiętać, iż wynik ten nie jest istotny statystycznie.

8 Programowanie Parami: Razem czy osobno?

Także w tym przypadku zweryfikowaliśmy, czy rozkład danych badanej próbki jest normalny (czas zakończenia pracy dla osób wykonujących zadanie *post-mortem*). Według wyniku testu Shapiro-Wilka, dane te posiadają rozkład normalny na poziomie istotności równym 0.10.



Rysunek 3. Średni względny czas zakończenia zadania *post-mortem* dla programistów indywidualnych pracujących w parach (XP, SbS) lub indywidualnie.

5. Ocena uczestników

Na zakończenie eksperymentu przeprowadziliśmy badanie ankietowe, zadając uczestnikom eksperymentu kilka pytań dotyczących ich wrażeń, co do modeli pracy, z którymi zetknęli się podczas eksperymentu. 55% uczestników preferowało programowanie w parach (podejście SbS lub XP) ponad pracę indywidualną, podczas gdy 40% wyrażało odmienną opinię (5% miało mieszane uczucia). Wśród tych 55% podejście Side-by-Side było preferowane przez 70% badanych, zaś XP przez 30%.

Komunikacja w parach SbS została oceniona pozytywnie (bardzo dobrze lub wystarczająco dobrze) przez 95% badanych.

48% badanych pracujących w parach było zadowolonych z jakości swojego własnego kodu, podczas gdy 36% osób nie była usatysfakcjonowane. 45% osób było usatysfakcjonowanych z jakości kodu stworzonego przez partnera, podczas gdy 45% było przeciwnego zdania. Ponieważ wszyscy uczestnicy eksperymentu mieli okazję pracować zarówno w modelu XP, jak i w modelu SbS, dlatego nie znamy odpowiedzi na pytanie o pewność (lub jej brak) w stosunku do XP i SbS.

6. Podsumowanie

Ważnym wnioskiem wypływającym z opisywanych badań jest fakt, że programowanie „ramię-w-ramię” (SbS) jest bardzo interesującą alternatywą dla Programowania Parami znanego z XP. Czas zakończenia dla SbS kształtował się na poziomie 60% w porównaniu z programowaniem indywidualnym, co oznacza, że różnica w pracochłonności sięga zaledwie 20% (z wcześniejszych opracowań wynikało, że dodatkowa pracochłonność sięgała nawet 60%). Jednakże należy pamiętać, że nakład pracy na rozwój istniejącego kodu wynosi również około 20% więcej niż w przypadku XP, co wskazuje, że wiedza o kodzie stopniowo się rozchodzi, lecz wolniej w przypadku SbS niż w XP. Z kolei subiektywne oceny wskazują, iż 55% badanych preferuje współpracę w parach ponad pracę indywidualną. Wśród nich, aż 70% przypadało na SbS, zaś tylko 30% dla klasycznego programowania parami.

Uzyskane wyniki wskazują, iż po raz kolejny czynnik ludzki, często niewystarczająco doceniany w organizacji przedsięwzięć programistycznych, ma istotne znaczenie na przebieg i powodzenie przedsięwzięć programistycznych. Fizyczna bliskość programistów pracujących „ramię-w-ramię” zmniejszająca nakłady na komunikację wewnątrz zespołu, co znajduje bezpośrednie odzwierciedlenie w wydajności pracy w tym właśnie modelu. Z kolei ściśle określona odpowiedzialność za realizację zadania pozwala osiągnąć efekt synergii, dzięki któremu podział prac wewnątrz pary jest dostosowany do umiejętności i doświadczenia każdego z członków zespołu. W tym świetle SbS wydaje się być dobrym rozwiązaniem dla projektów, w których konieczne jest zwiększenie tempa realizacji przy zachowaniu zadowalającej jakości wyników prac oraz bez znaczącego wzrostu kosztów przedsięwzięcia.

Literatura

1. Basili, V. E., Lanubile, F.: Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering, Volume 25, No. 4 (1999) 456–473.
2. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional (1999).
3. Brooks, R. E.: Studying programmer behavior experimentally: the problems of proper methodology. Communications of the ACM, Volume 23, No. 4 (1980) 207–213.
4. Brzeziński, J.: Metodologia badań psychologicznych. Wydawnictwo Naukowe PWN (2004).
5. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley (2000).
6. Cockburn, A.: Agile Software Development. Addison-Wesley (2002).
7. Cockburn, A.: Crystal Clear. A Human-Powered Methodology for Small Teams. Addison-Wesley (2005).
8. Dickey, T. F. Programmer variability. Proceedings of the IEEE, Volume 69, No. 7 (1981) 844–845.
9. Humphrey, W.: A Discipline for Software Engineering. Addison-Wesley, Reading MA (1995).
10. Laboratory of Software Engineering. <http://www.se.cs.put.poznan.pl/en/content/research/experiments/experiments.html> (2005).

10 **Programowanie Parami:** Razem czy osobno?

11. Lui, K. M., Chan, K. C. C.: When Does a Pair Outperform Two Individuals? Lecture Notes in Computer Science, Volume 2675 (2003) 225–233.
12. Montgomery, D. C.: Introduction to Statistical Quality Control. Third Edition. John Wiley & Sons, Inc. (1997).
13. Nawrocki, J., Wojciechowski A.: Experimental Evaluation of Pair Programming. In: Maxwell, K., Oligny, S., Kusters, R., van Venedaal E. (eds.): Project Control: Satisfying the Customer. Proceedings of the 12th European Software Control and Metrics Conference. Shaker Publishing, London (2001) 269–276.
14. Nosek J. T.: The Case for Collaborative Programming. Communications of the ACM, Volume 41, No. 3 (1998) 105–108.
15. Padberg, F., Mueller, M.: An Empirical study about the Feelgood Factor in Pair Programming. In: Proceedings of the 10th International Symposium on Software Metrics METRICS 2004, IEEE Press (2004).
16. Pressman, R. S.: Software Engineering: A Practitioner's Approach. Fifth Edition. McGraw-Hill (2001).
17. Sackman, H., Erikson, W. J., Grant, E. E.: Exploratory Experimental Studies Comparing Online and Offline Programming Performance. Communications of ACM, Volume 11, No. 1 (1968) 3–11.
18. Shapiro, S.S., Wilk, M.B. An analysis of variance test for normality (complete samples). Biometrika. 52, 3 and 4 (1965) 591– 611.
19. Sheil, B. A.: The Psychological Study of Programming. ACM Computing Surveys, Volume 13, No. 1 (1981) 101–120.
20. Tichy, W.F.: Should Computer Scientists Experiment More? IEEE Computer, Volume 31, No. 5 (1998) 32–40.
21. Williams, L.: The Collaborative Software Process. PhD Dissertation at Department of Computer Science, University of Utah, Salt Lake City (2000).
22. Williams, L. at al.: Strengthening the Case for Pair Programming. IEEE Software, Volume 17, No. 4 (2000) 19–25.