AUTOMATYKA • 2007 • Tom 11 • Zeszyt 1–2

Jacek Dajda*, Grzegorz Dobrowolski*

Współczesne metody systemowego wytwarzania oprogramowania

1. Wprowadzenie

W latach 70. i 80. XX w. rosnące zapotrzebowanie rynku na coraz większe i bardziej złożone systemy informatyczne zaowocowało opracowaniem formalnych, tzw. tradycyjnych, modeli wytwarzania oprogramowania opartych na normach ISO (np. 9000) i modelu CMM [1]. Pozwoliło to na uporządkowanie skomplikowanego procesu wytwarzania oprogramowania, kosztem jego biurokratyzacji, pogorszenia odporności na zmiany zewnętrznych wymagań oraz szybkości, z jaką dostarczany jest klientowi działający produkt [8]. Problemy te, znane jako syndrom LOOP [2], były motywacją zmian podstawowego modelu wytwarzania oprogramowania z końcem lat 90. Motorem zmian w tym zakresie stały się przeobrażenia zachodzące w przemyśle, związane z recepcją japońskiej szkoły zarządzania [3] oraz recesja w gospodarce, zmuszające podmioty do obniżania kosztów produkcji [4]. Nurt ten w zakresie inżynierii oprogramowania zaowocował m.in. powstaniem metodologii zwinnych (*agile*) takich, jak: Programowanie ekstremalne (XP) [7, 10, 11], Scrum[13] czy rodzina metodologii Crystal [12]. Jedną z najistotniejszych cech tychże metodologii jest intensywna komunikacja w obrębie zespołu programistycznego, co równoznaczne jest z wymaganiem współlokalizacji uczestników projektu. Należy zaznaczyć tu, że wymaganie to odnosi się również do klienta, którego uznano za jednego z członków zwinnego zespołu programistycznego. Metodologie zwinne, rozwiązując kilka podstawowych problemów związanych z zarządzaniem projektem, przyniosły w zamian całkiem nowe wyzwania dotyczące wytwarzania kodu [4, 14].

Przykładem może być wysoka zmienność interfejsów [14] podyktowana ciągłą modyfikacją i ulepszaniem kodu np. implementowanych bibliotek [15]. Wymusza ona intensywną komunikację pomiędzy kooperującymi instytucjami, dzięki której możliwa jest integracja nowych funkcjonalności wykorzystywanych bibliotek. Komunikacja ta realizowana jest zazwyczaj poprzez delegowanie pracowników [14], co fizycznie redukuje możliwość współpracy do granic np. miasta. Znaczne rozproszenie zespołu projektowego staje się wtedy powodem wzrostu kosztów oraz opóźnień całego przedsięwzięcia.

* Katedra Informatyki, Akademia Górniczo-Hutnicza w Krakowie

Problem komunikacji w rozproszonym zespole jest powszechnym wyzwaniem firm wytwarzających oprogramowanie, niekoniecznie wykorzystujących metodologie zwinne. Jest to wynik zewnętrznych uwarunkowań ekonomicznych, skutkujących lokalizacją ośrodków badawczych tam, gdzie koszt wytwarzania oprogramowania jest niższy. W efekcie znaczący procent przedsięwzięć informatycznych realizowany jest równolegle w kilku (zazwyczaj znacznie oddalonych) ośrodkach programistycznych. Dlatego też zaobserwować można wzrost zapotrzebowania na programistyczne i organizacyjne wsparcie dla tego typu projektów. Wsparcie to jest szczególnie istotne w przypadku zespołów pracujących za pomocą metodologii zwinnych, ze względu na wymaganą ścisłą współpracę uczestników projektu.

Celem niniejszego artykułu jest przedstawienie współczesnych trendów w procesie wytwarzania oprogramowania, opartych na dwóch, po części przeciwstawnych, kierunkach:

1) metodologiach zwinnych,
2) projektach rozproszonych.

W ramach prezentacji pierwszego z nich, autorzy odwołują się do podejścia systemowego i wskazują istniejące analogie, podparte identyfikacją kluczowych pojęć metodologii zwinnych. Omówienie drugiego z kierunków obejmuje problem przeniesienia praktyk metodologii zwinnych na grunt zespołów rozproszonych. Zawiera ono krótki przegląd obecnego stanu prac nad wspierającymi rozwiązaniami. W ramach przedstawienia dalszych kierunków badawczych autorzy zarysowują także swój wkład do omawianej dziedziny, z uwzględnieniem zakresu oraz obecnego stanu zaplanowanych prac.

Artykuł zbudowany jest następująco. W sekcji drugiej przedstawiono genezę i podstawowe założenia zwinnych metodologii. W kolejnej sekcji wskazano i wyjaśniono istotne elementy (w szczególności praktyki) metodologii zwinnych, które są przejawem podejścia systemowego. W dalszej kolejności omówiono koncepcję rozproszonego programowania zwinnego i podejmowane próby jego realizacji. Ostatnia sekcja zawiera propozycję dalszych kierunków badawczych, w tym streszczenie zakresu i obecnego stanu prac podejmowanych przez autorów niniejszego artykułu.

2. Manifest zwinnych metodologii

Krytyka klasycznych metodyk wytwarzania oprogramowania zaowocowała powstawaniem nowych koncepcji, z których najbardziej popularną stało się Programowanie ekstremalne [7]. Po raz pierwszy wykorzystana została ona przez jej twórcę, Kenta Becka, w projekcie firmy Daimler Chrysler w 1996 roku. Burzliwa reakcja środowiska, rosnące zainteresowanie nowym ruchem oraz kolejne powstające zwinne metodologie były motywacją do sformułowania i usystematyzowania jego podstawowych założeń. W efekcie, w lutym 2001, jego główni twórcy zaproponowali tzw. Manifest zwinnych metodologii [25], w którym uznali, iż cenniejsze są:

Osoby i komunikacja od procedur i narzędzi.
Działające oprogramowanie od kompleksowej dokumentacji.
Współpraca klienta od negocjacji kontraktów.
Reagowanie na zmianę od podążania za planem.

Osoby i komunikacja. Postulat ten odnosi się do sformalizowanych procedur, reguł i środowisk tradycyjnych metod, które były ważniejsze od potrzeb ludzi wchodzących w skład zespołu programistycznego. Procedury i reguły narzucane odgórnie przez ludzi nieuczestniczących w pracach zespołu, nierzadko okazywały się jedną z przeszkód, często powodując izolację poszczególnych członków zespołu. Jednym z efektów takiej sytuacji były nieporozumienia i konflikty, m.in. w czasie integracji zmian.

Działające oprogramowanie. Ten punkt manifestu podkreśla prawdziwą wartość projektu informatycznego, którą jest działające oprogramowanie, spełniające wymagania klienta. W przypadku tradycyjnych metod wytwarzania oprogramowania faza implementacji poprzedzana była wielomiesięcznymi studiami projektowymi, które z punktu widzenia klienta nie miały żadnej wartości. Także koszt uzyskania i następnie utrzymywania kompleksowej dokumentacji nierzadko przerastał wartość końcowego produktu.

Współpraca klienta. Postulat proponuje zmianę roli klienta w projekcie. Beck i inni zauważyli, iż znacznie lepsze efekty uzyskują zespoły, w których klient ma realny wpływ na pracę zespołu oraz końcowy kształt produktu. Wprowadzenie klienta do zespołu nie tylko pozytywnie wpływa na środowisko pracy, lecz także uwalnia zespół od dodatkowych prac związanych z prezentacją wyników.

Reagowanie na zmianę. Punkt ten związany jest z ryzykiem projektu informatycznego, wynikającym ze zmian wymagań klienta bądź innych zewnętrznych warunków. Natychmiastowe reagowanie na wszelkie zmiany możliwe jest dzięki realizacji trzech pierwszych punktów manifestu, w szczególności ścisłej współpracy z klientem. Odpowiednie podejście do planowania i oceny osiągniętych wyników pozwala na uniknięcie problemów związanych z syndromem LOOP [2], co pociąga za sobą zmniejszenie ryzyka całego przedsięwzięcia.

Mimo wskazanych atutów, metodologie zwinne posiadają także słabe punkty. Ich istotnym ograniczeniem, szczególnie biorąc pod uwagę obecne trendy rozproszonych projektów, jest potrzeba aktywnego udziału klienta oraz współlokalizacji wszystkich uczestników projektu. Z ostrą krytyką spotyka się również brak dokumentacji projektowej, co może mieć negatywny wpływ w przypadku długich projektów, a także wąski zakres zastosowań, obejmujący tylko niewielkie zespoły programistyczne.

Do metodologii zwinnych zalicza się, m.in. programowanie ekstremalne (XP), Scrum, rodzina metodyk Crystal, Lean Software Development, Agile Unified Process, Feature-Driven Development, Adaptive Software Development oraz Dynamic Systems Development Method.

3. Metodologie zwinne jako systemowe podejście do wytwarzania oprogramowania

Wytwarzanie oprogramowania jest procesem złożonym z elementów o różnorodnej specyfice. Aby proces ten zakończył się sukcesem, muszą zostać spełnione określone warunki dotyczące zarówno samych elementów, jak i ich wzajemnych interakcji. Trudność

polega tutaj nie tylko na spełnieniu tych warunków, ale także na samym wskazaniu elementów i interakcji, gdyż proces ten nie jest do końca określony. Co więcej, pewne projekty wymagają nieco innego podejścia lub innych narzędzi, co dodatkowo komplikuje wypracowanie uniwersalnej specyfikacji tego procesu.

3.1. Klasyczne podejście a metodologie zwinne

Klasyczne podejście do wytwarzania oprogramowania bazuje m.in. na technikach modelowania całego procesu. Przykładem może być model kaskadowy [21], który proponuje rozbicie procesu na autonomiczne etapy ze ściśle określonym sposobem ich realizacji. Jego istotną wadą jest jednak uproszczony obraz rzeczywistości oraz nieuwzględnianie wzajemnych związków pomiędzy poszczególnymi etapami i występującymi w ich ramach aktywnościami. Rozwiązanie tego problemu było punktem wyjścia dla koncepcji myślenia systemowego [20]. Wraz z narastaniem trudności i kosztów organizacji w dużych korporacjach w latach 90. XX w., nurt ten zyskał uznanie w dziedzinie ekonomii i wpłynął na proces zmian w metodach zarządzania i kierowania organizacjami. Metodologie zwinne są przejawem tego podejścia na gruncie inżynierii oprogramowania. Istotną różnicą w stosunku do tradycyjnych metod wytwarzania oprogramowania było również wskazanie ludzi i ich wzajemnych interakcji jako istotnych elementów całego procesu. Doprowadziło to do podkreślenia znaczenia komunikacji i współpracy uczestników projektu informatycznego, co jawnie sformułowano w samym manifeście.

Pojęcie **metodologii**, które oznacza naukę o **metodykach**, w przypadku metodologii zwinnych stosowane jest celowo, ze względu na często wysokopoziomowy charakter proponowanych rozwiązań. Są one podstawą do precyzowania konkretnych procesów o określonych regułach postępowania. Równie ważne tutaj pojęcia to:

- wartość,
- praktyka,
- technika,
- synergia.

Wartość oznacza tutaj element lub cechę procesu, która wyznacza podstawowe założenia danej metodologii. Zaprezentowany manifest można traktować jako syntetyzujące ujęcie wartości wszystkich metodologii zwinnych. Aby zapewnić obecność wyróżnionych wartości w ramach procesu, niezbędny jest określony zbiór *praktyk*. Pod tym pojęciem kryją się zarówno ogólne reguły, jak i specyficzne aktywności pojawiające się w pracy zespołu programistycznego, np. programowanie w parach. W ich ramach mogą być wykorzystywane konkretne **techniki**, czyli określone przepisy działania, które, w niektórych przypadkach, związane są z pewnym typem narzędzi programistycznych. Choć poszczególne praktyki i techniki stanowią wartość samą w sobie, istotne znaczenie ma również ich wzajemna interakcja, czyli **synergia**, która ma istotny wpływ na skuteczność całej metodologii.

Powyższe pojęcia odnoszą się do każdej metodologii zwinnej. Określają jej strukturę (przejście od wartości do technik), mechanizm (synergia) i dostępny zestaw narzędzi (praktyki), co pozwala na realizację podstawowych założeń manifestu. Są one wynikiem podejścia systemowego do procesu wytwarzania oprogramowania, w którym istotne znaczenie

odgrywa złożona sieć wewnętrznych zależności. Zadaniem proponowanych wartości, praktyk i technik jest podtrzymywanie tychże zależności, a także wzajemna stymulacja, a więc osiągnięcie pożądanej synergii.

Opisane powyżej elementy podejścia systemowego umożliwiły metodologiom zwinnym na praktyczne rozstrzygnięcie problemu czterech zmiennych projektu informatycznego [5]. Są to:

1) czas,
2) koszt,
3) zasoby,
4) zakres.

Zmienne te charakteryzują się zależnością, przypominającą zasadę naczyń połączonych, tzn. zmiana którejkolwiek z nich wpływa na pozostałe i na odwrót. Innymi słowy, niemożliwe jest bezpośrednie kontrolowanie wszystkich czterech na raz, jedynie maksymalnie trzech z nich. Nieznajomość tej reguły jest częstą przyczyną niepowodzeń projektów i w prosty sposób prowadzić może do syndromu LOOP, np. przez wyznaczenie zbyt szerokiego zakresu implementowanych funkcjonalności bez uwzględniania wartości pozostałych zmiennych.

3.2. Praktyki metodologii zwinnych

Ponieważ metodologie zwinne bazują na wspólnych założeniach i wartościach zawartych, w przedstawionym manifeście znaczny procent proponowanych praktyk jest realizacją tych samych koncepcji, z tym że przedstawionych lub nazwanych w odmienny sposób. W szczególności dotyczy to podstawowych praktyk, takich jak iteracyjne podejście do procesu wytwarzania oprogramowania, praca z klientem, planowanie oraz ciągła automatyczna integracja powstającego produktu. Niektóre z nich, szczególnie dotyczy to praktyk bezpośrednio związanych z produkowanym oprogramowaniem, zyskały uznanie jako praktyki niezależne i stały się podstawą nowych metodologii. Ze względu na ograniczenia niniejszego artykułu przedstawione zostaną tutaj jedynie wybrane, znaczące i specyficzne praktyki poszczególnych metodologii.

Praca z klientem i przebieg projektu. Jedną z podstawowych praktyk metodologii zwinnych jest iteracyjny przebieg projektu z niewielkimi, częstymi iteracjami (do 3 tygodni) i wydaniami. Każdorazowo organizuje się spotkania planistyczne i podsumowujące, podczas których klient ma wgląd w postęp zespołu i okazję do zaplanowania kolejnej iteracji. Celem każdej iteracji jest dostarczenie klientowi działającego produktu, dlatego wszystkie tradycyjne etapy projektu takie jak analiza, projekt, implementacja, realizowane są w obrębie jednej iteracji na niewielkim fragmencie funkcjonalności. Istotnym elementem planowania jest estymacja prędkości zespołu i ciągła jej poprawa na podstawie zebranych danych. Dzięki temu w czasie kolejnych iteracji klient i zespół mogą precyzyjniej określić jej zakres i czas zakończenia. Praca z klientem to nie tylko częste spotkania planistyczne, lecz także wizyty klienta i podejmowanie przez niego decyzji na bieżąco. Przykładem może być praktyka XP „klient na miejscu" (*On-site customer*) zakładająca, iż klient jest częścią zespołu i aktywnie uczestniczy w jego pracach.

Jakość wytwarzanego oprogramowania. Jedną z bardziej rozpoznawanych praktyk metodologii zwinnych jest „programowanie w parach" przy jednym komputerze (XP). Ciągła inspekcja i analiza tworzonego kodu przez współpartnera, dwutorowa praca nad trudnym zagadnieniem oraz wzajemna wymiana doświadczeń w obrębie całego zespołu pozwalają na uzyskanie wysokiej jakości wytworzonego oprogramowania oraz poprawę efektywności całego zespołu. XP wprowadziło także pojęcie refaktoryzacji [15], czyli udoskonalania struktury istniejącego kodu bez zmiany jego zewnętrznego zachowania (interfejsu). Praktyka ta, obok testów jednostkowych, jest jednym z najbardziej cenionych wkładów XP do inżynierii oprogramowania.

Znaczenie roli człowieka w projekcie. Jak podkreślano wcześniej, metodologie zwinne zauważyły i podkreśliły istotną rolę człowieka w procesie wytwarzania oprogramowania. Aby zapewnić wysoką wydajność zespołu programistycznego, postanowiono stworzyć komfortowe środowisko pracy. Przejawem tego podejścia są m.in. praktyka **brak nadgodzin** oraz możliwość wyboru zadań do realizacji w czasie gry planistycznej. Innym przykładem jest rola Mistrza Scruma w metodologii Scrum. Jest on odpowiedzialny za zapewnienie idealnych warunków dla zespołu programistycznego i usuwanie wszelkich przeszkód mogących zakłócić przebieg prac nad projektem.

Wymiana informacji w zespole. W ramach poprawy komunikacji w obrębie zespołu zaproponowano szereg praktyk i technik, m.in.:

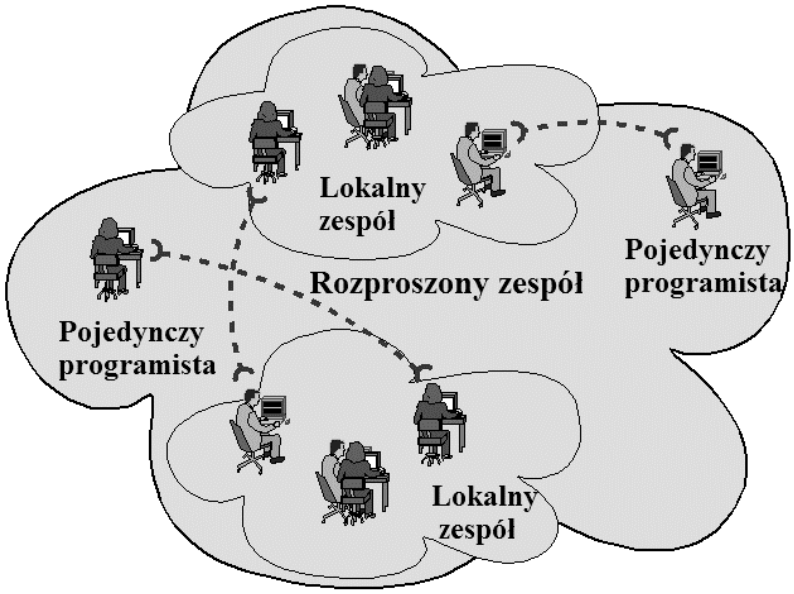
- „osmotyczną komunikację",
- „wspólny pokój",
- „radiatory informacyjne".

Dwie pierwsze są wynikiem obserwacji, iż istotne znaczenie na wydajność zespołu programistycznego ma sposób rozprzestrzeniania się wiedzy o projekcie. Według autora metodologii Crystal, zastosowanie koncepcji otwartych przestrzeni biurowych („wspólny pokój") umożliwia swobodną dyfuzję (przenikanie, osmozę) informacji. Przykładowo, programista może odpowiedzieć na zasłyszane pytanie skierowane do kogoś innego. Z drugiej strony, metodyki Crystal zauważają potrzebę izolacji programistów od zewnętrznych rozpraszających bodźców, proponując wydzielanie odpowiednich godzin konsultacji. By zachować „osmotyczną komunikację" i jednocześnie zachować niezbędną izolację, zaproponowano technikę „radiatorów informacyjnych". Polega ona na zastosowaniu prostych środków, m.in. tablic ogłoszeń z odręcznymi wykresami i notatkami w dobrze widocznych miejscach biura, które umożliwiają szybką i wygodną wymianę informacji.

Interesującym spostrzeżeniem dokonanym przez autorów niniejszego artykułu są istniejące analogie pomiędzy wybranymi praktykami metodologii zwinnych, w szczególności programowania ekstremalnego, a jedną z najbardziej sformalizowanych metod wytwarzania oprogramowania, jakim jest Cleanroom [9] (dosłownie: czysty pokój). Jednym z głównych zamierzeń tej metodyki, wykorzystywanej m.in. przez NASA, jest wytwarzanie wysokiej jakości oprogramowania, pozbawionego jakichkolwiek błędów już w pierwszej iteracji. Osiąga się to przez wprowadzenie rygorystycznego procesu kontroli, składającego się m.in. z formalnego dowodzenia implementowanych algorytmów, inspekcji kodu oraz statystycznych metod weryfikujących jakość, m.in. testów regresyjnych. Inspekcja kodu oraz testowanie znalazły swoje miejsce również w metodologiach zwinnych, co potwierdza jedno z ich głównych założeń, jakim jest dostarczanie działającego oprogramowania wysokiej jakości.

3.3. Rozproszone metodologie zwinne

Jednym ze współczesnych trendów, a jednocześnie wyzwań, jest realizacja projektów przez rozproszone zespoły programistyczne. Poważnym problemem jest tutaj zapewnienie odpowiednich kanałów komunikacji. Dla zespołów wykorzystujących metodologie zwinne ma to szczególnie istotne znaczenie ze względu na nacisk, jaki kładą one na komunikację i ścisłą współpracę uczestników projektu. Ilustracją dla omawianego typu sytuacji może być rozproszony zespół XP, którego członkowie praktykują „programowanie w parach" (rys. 1).



Rys. 1. Zdalni programiści w parach w ramach rozproszonego zespołu XP

W literaturze światowej, podejmującej omawianą dziedzinę, zaobserwować można dwa trendy badawcze, dotyczące:

1) systemów wspierających zdalną pracę w parach [16, 17],
2) systemów zorientowanych na zarządzanie projektem rozproszonym [18, 19].

Badania nad systemami wspierającymi zdalną pracę mają na celu określenie optymalnego kształtu narzędzi i praktyczną weryfikację ich przydatności. Przykładem może być projekt Sangam [17], w ramach którego tworzony jest zestaw komponentów środowiska Eclipse. Pozwala on na jednoczesną edycję wspólnego kodu oraz synchronizację zasobów danej sesji programistycznej. Do tego nurtu zaliczyć też można eksperymenty z zdalnym programowaniem w parach realizowane za pomocą oprogramowania Microsoft NetMeeting [18]. Podejście to można znacznie rozszerzyć za pomocą rozwiązań teleimersyjnych. Przykładem mogą być prace nad projektem Office of Real Soon Now [19] i pokrewne eksperymenty [16].

Reprezentantem trendu badań nad zarządzaniem projektami rozproszonymi jest system MILOS ASE [15]. W rozwiązaniu tym wprowadzono system zarządzania historiami (*Story Management System*), którego zadaniem jest koordynacja działań w geograficznie rozproszonym projekcie. Kolejnym przykładem są systemy XPlanner [22] oraz XPWeb [23], dedykowane śledzeniu postępów zespołu XP. Do grupy tej należy zaliczyć też narzędzia ogólnego przeznaczenia, m.in. listy dyskusyjne i strony wiki.

Pomimo podejmowanych prób, istniejące rozwiązania nie gwarantują wystarczającego wsparcia dla prezentowanych praktyk. Do istniejących ograniczeń należy zaliczyć, m.in. zbyt duże opóźnienia, zakłócenia i małą liczbę kanałów komunikacyjnych. Co więcej, rozwiązania te dotyczą głównie praktyk metodologii XP. Praktyki innych metodologii, m.in. „wspólny pokój", zarządzanie „dziennikiem produktu", „osmotyczna komunikacja" i „radiatory informacyjne" wymagają dalszych prac i eksperymentów.

Zarówno teoria [12], jak i praktyka sugerują jednak potrzebę testowania takich rozwiązań nie tylko w kontekście krótkich eksperymentów, ale i długotrwałej symulacji środowiska produkcyjnego, które wskazać mogą wady i zalety poszczególnych rozwiązań. Jest to szczególnie istotne w kontekście optymalizacji kosztów środowiska pracy. Ponieważ pełne oprzyrządowanie i oprogramowanie teleimersyjne [19] wymaga znacznych nakładów finansowych, minimalizacja tychże kosztów przy maksymalizacji jakości pracy wydaje się obiecującym zadaniem badawczym.

4. Podsumowanie

Celem współczesnych metod systemowego wytwarzania oprogramowania jest jednocześnie obniżenie ryzyka prowadzonych projektów oraz kosztów wytwarzania oprogramowania. Coraz częściej więc wykorzystuje się praktyki metodologii zwinnych, które stanowią kontynuację podejścia systemowego w inżynierii oprogramowania. Synergia sprawdzonych (m.in. w formalnych metodologiach takich jak Cleanroom) praktyk i technik pozwala na znaczne obniżenie ryzyka projektu oraz podniesienie jakości dostarczonego produktu. Kolejnym powszechnym trendem jest lokalizacja ośrodków badawczych w tańszych strefach ekonomicznych, co niestety skutkuje rozproszeniem zespołów projektowych i dodatkowymi kosztami związanymi z potrzebą komunikacji.

Rosnąca potrzeba na programistyczne wsparcie rozproszonych zespołów jest przesłanką do podejmowania badań, których celem jest wytworzenie odpowiednich narzędzi oraz ich eksperymentalna weryfikacja. Jest to szczególnie istotne w kontekście optymalizacji kosztów środowiska pracy. Obecnie dostępne rozwiązania nie gwarantują wysokiej wydajności pracy w środowisku rozproszonym. Co więcej, dotyczą one w głównej mierze praktyk programowania ekstremalnego.

Zamierzeniem autorów niniejszego artykułu jest tworzenie i rozwój narzędzi wypełniających tę lukę. Prowadzone prace [24] odnoszą się do metodologii XP, Crystal i Scrum. Badania obejmują rozszerzenia środowiska Eclipse o funkcjonalności wspierające zdalną pracę w parach, rozwój platformy komunikacyjnej opartej na „osmotycznej komunikacji" i „radiatorach informacyjnych" oraz realizację środowiska do przeprowadzania wirtualnych gier planistycznych w czasie rzeczywistym. Obecnie zrealizowano prototyp pierwszego z wymienionych narzędzi oraz poddano dwukrotnej eksperymentalnej weryfikacji, każdo-

razowo dokonując wskazanych ulepszeń. Przyjęty sposób rozwoju narzędzia opiera się na iteracyjnym podejściu metodologii zwinnych, a forma eksperymentów symulujących rzeczywiste środowisko pracy pozwala na uwzględnienie trudnej do mierzenia synergii charakterystycznej dla pracy w parach [6].

Literatura

[1] Humphrey W.: *Managing the Software Process.* Addison-Wesley Professional, 1989
[2] Nawrocki J., Jasiński M.: *Programowanie Ekstremalne i PRINCE 2.* Problemy i metody inżynierii oprogramowania, WNT, Warszawa-Wrocław 2003, 525–535
[3] Takeuchi N.: *The Knowledge Creating Company: How Japanese Companies Create The Dynamics of Innovation.* Oxford University Press, 1995
[4] Karlson E., Anderson L.: *XP and Large Distributed Software Projects.* Addison-Wesley, 2001
[5] Jeffries R.: *The King's Dinner.* XP Magazine, Styczeń 1999
[6] Williams L., Kessler R.: *Pair Programming Illuminated.* Addison-Wesly, Boston, MA., 2003
[7] Beck K.: *eXtreme Programming explained: embrace change.* Addison-Wesley, Boston, Ma., 1999
[8] Yourdon E.: *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*. Prentice Hall, New Jersey, 1997
[9] Mills H., Dyer M., Linger R.: *Cleanroom Software Engineering*. IEEE Software 4 (5): 19–25 września 1987
[10] Asteles D., Miller G., Nowak M.: *A Practical Guide to eXtreme Programming*. Prentice-Hall 2002
[11] Succi G. , MarchesI M.: *eXtreme Programming Examined.* Addison-Wesley, Boston, MA., 2001
[12] Cockburn A.: *Crystal Clear: A human-powered methodology for small teams*. Addison-Wesley, 2004
[13] Schwaber K., Beedle M.: *Agile Software Development with SCRUM.* Prentice Hall, 2002
[14] Lippert M., Roock S., Wolf H., Züllibhoven H.: *JVAM and XP: Using XP for Framework Development.* Addison-Wesley, 2001
[15] Fowler M.: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley 2000
[16] Baheti P., Williams L., Gehringer E., Stotts D.: *Exploring Pair Programming in Distributed Object-Oriented Team Project*. OOPSLA 2002 Educator's Symposium, Washington, 2002
[17] Ho C., Raha S., Gehringer E., Williams L.: *Sangam: A Distributed Pair Programming Plug-in for Eclipse, Eclipse Technology Exchange (Workshop)*. OOPSLA, Vancouver, 2004
[18] Maurer F., Martel S.: *Process Support for Distributed Extreme Programming Teams*. ICSE 2002 Workshop on Global Software Development, Orlando, 2002
[19] Bishop G., Welch G.: *Working in the Office of 'Real Soon Now'*. IEEE Computer Graphics and Applications, pages 76–78, 2000
[20] Weinberg G.M: *Myslenie systemowe.* PWN, 1979
[21] Royce W.: *Managing the Development of Large Software Systems*. IEEE WESCON, 1970
[22] Elliot D., Smith J.: *Manage the agile team with XPlanner*. JavaWorld.com, 15.08.2005
[23] Chirouze O., Cleary D., Mitchell G.: *A software methodology for applied research: eXtreme Researching: Research Articles.* Softw. Pract. Exper. 35, 15 (Grudzień 2005), 1441–1454
[24] Dajda J., Ciszewski S.: *Supporting extreme programming in distributed setting.* AGH Computer Science, vol. 7 (2005), 49–62
[25] Manifesto for Agile Software Development, http://www.agilemanifesto.org, 18.04.2007