Magazine

Strategie testowania i jakości Agile: dyscyplina ponad retoryką

Cześć I z IV: Zwinne tworzenie oprogramowania

Autor: Scott Ambler

O autorze: W lipcu 2006 r. Scott dołączył do zespołu IBM w Kanadzie jako Główny Metodolog Agile i Lean dla IBM Rational. Scott pracuje dla klientów IBM, w szczególności w obszarach coachingu i usprawniania procesów wytwórczych w IT. Scott pracuje w przemyśle IT od połowy lat 80-tych, a z technologią obiektową od wczesnych lat 90-tych. Napisał kilka książek i *white papers* traktujących o obiektowym wytwarzaniu oprogramowania, procesie oprogramowania, Modelu Skalowania Agile (ASM), Wytwarzaniu Agile Kierowanym Modelem (AMDD), Technikach Baz Danych Agile, Agile UP (ang. *Unified Process*). Przemawiał na wielu konferencjach na całym świecie.

Kontakt: **Scott_Ambler@ca.ibm.com** twitter: **http://twitter.com/scottwambler**

Wprowadzenie

Lubimy mówić, że developerzy Agile są „zarażeni jakością” i w wielu aspektach jest to prawdą. Praktycy Agile, przynajmniej ci zdyscyplinowani, mają skłonność do walidowania swojej pracy na miarę swoich możliwości. W wyniku tego znajdują sposoby na włączenie tak wielu technik testowania i zapewnienia jakości do swoich praktyk zawodowych, jak to możliwe. Niniejszy artykuł opisuje techniki i filozofię, które zdyscyplinowani developerzy Agile stosują w praktyce, umieszczając je w kontekście zwinnego cyklu życia wytwarzania oprogramowania.

Zwinne tworzenie oprogramowania

Niniejszy rozdział stanowi krótkie wprowadzenie do zwinnego tworzenia oprogramowania. Jest podzielony na następujące sekcje:

- Definiowanie Agile
- Cykl życia wytwarzania systemu wg Agile
- Tradycyjny cykl życia wytwarzania systemu
- W jaki sposób Agile jest inne?
- Porównanie podejść Agile i tradycyjnego

Definiowanie Agile

Jedną z frustracji, jakie dręczą wielu nowych ludzi w Agile, jest to, że nie ma żadnej oficjalnej definicji zwinnego wytwarzania oprogramowania, mimo, że wielu wskaże na wartości i zasady Manifestu Agile. Powiedziawszy to, moją definicją zdyscyplinowanego wytwarzania oprogramowania Agile jest:

Iteracyjne i przyrostowe (ewolucyjne) podejście do wytwarzania oprogramowania, wykonywane w ścisłej współpracy przez samoorganizujące się zespoły wewnątrz efektywnego środowiska zarządczego, z tylko taką ilością „ceremonii”, by wytworzyć wysokiej jakości oprogramowanie w efektywny pod względem kosztu i czasu sposób, które spełnia zmieniające się potrzeby udziałowców.

Oto kryteria, których szukam by określić, czy zespół realizuje podejście zdyscyplinowane do wytwarzania Agile. Po pierwsze, czy zespół wykonuje developerskie testy regresji, lub nawet lepiej – stosuje w wytwarzaniu podejście kierowane testami? Po drugie, czy udziałowcy są czynnymi uczestnikami w

wytwarzaniu? Po trzecie, czy zespół produkuje wysokiej jakości, działające oprogramowanie w regularnych odstępach czasu? Po czwarte, czy zespół ściśle ze sobą współpracuje, w samoorganizujący się sposób w ramach efektywnego środowiska zarządczego? Po piąte, czy doskonali on swoje podejście podczas trwania projektu?

Cykl życia wytwarzania systemu Agile

Aby naprawdę zrozumieć strategie testowania i jakości Agile, musimy zrozumieć, jak wpasowują się one do całego cyklu życia wytwarzania systemu (SDLC). Rysunek 1 prezentuje wysokopoziomowy widok na cykl życia wytwarzania Agile, pokazując, że projekty Agile są organizowane w serie bloków czasowych (ang. *time boxes*) zwanych iteracjami (w metodologii Scrum nazywają się one „Sprinty" i niektórzy traktują je jak cykle). Mimo, że wielu ludzi powie , że cykl życia Agile jest iteracyjny, to nie jest do końca prawdą – możemy zobaczyć, że tak naprawdę jest seryjny w dużych, a iteracyjnych w małych projektach. Aspekt seryjności bierze się z faktu, że istnieją co najmniej 3 różne rodzaje iteracji: iteracje inicjacyjne (jasny zielony), iteracje konstrukcyjne (jasny niebieski) i iteracje wydań (niebieski) – gdzie natura pracy, którą wykonujesz jest inna. Implikacją tego jest, że nasze podejście do testowania/walidacji również różni się w zależności od tego, w którym miejscu cyklu życia się znajdujemy. Skutkiem jest to, iż ważne jest, by zrozumieć każdą z wysokopoziomowych czynności wskazanych w tym cyklu życia:

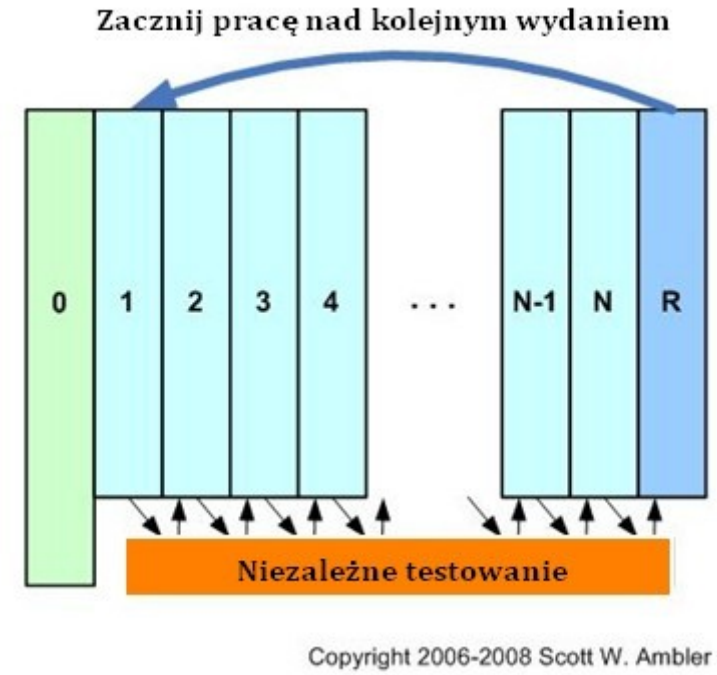
Iteracja 0. Celem tej iteracji, lub tych iteracji w złożonych projektach, jest zainicjowanie projektu. Będziemy wykonywać określanie wstępnej wizji wymagań, wstępnej wizji architektury, rozpoczniemy identyfikację i organizowanie zespołu developerskiego, dojdziemy do pewnego rodzaju zbieżności udziałowców zgodnie z celem (celami) projektu, będziemy zdobywać fundusze i wsparcie dla projektu. Czynności testowania/walidacji obejmują rozpoczęcie określania środowiska testowego i narzędzi oraz potencjalne recenzowanie dokumentacji wstępnych modeli, planów, wizji czy celów udziałowców.

Iteracje konstrukcyjne. Podczas każdej iteracji konstrukcyjnej (iteracje od 1 do N na Rysunku 1), celem jest wytworzenie oprogramowania potencjalnie gotowego (ang. *potentially shippable software*). Zespoły Agile przestrzegają praktyki priorytetyzowania wymagań – w każdej iteracji biorą najważniejsze wymagania z pozostającego stosu zadań (co zespoły Scrum nazywają *product backlog*) i je implementują. Zespoły Agile będą przyjmować podejście „zespołu kompletnego", w którym testerzy należą do zespołu developerskiego, pracując ramię w ramię z developerami, by zbudować system. W swoich pracach testowych skupiają się często na testowaniu potwierdzającym i testach regresji wykonywanych przez developera lub – co wydaje się lepsze – na Wytwarzaniu Kierowanym Testami (ang. *Test-Driven Development* (TDD)).

Równoległe testowanie niezależne. Zdyscyplinowane zespoły Agile w trakcie całego cyklu życia wykonują ciągłe, równoległe do iteracji konstrukcyjnych, testowanie niezależnie. Celem tej pracy jest znalezienie defektów, które przeoczył zespół developerski; często wykonywane są tu wyższe formy testowania, takie jak testowanie eksploracyjne, testy integracyjne systemu, testy bezpieczeństwa, użyteczności i tak dalej – które wymagają znaczących umiejętności testerskich, złożonych narzędzi do testowania i często złożonych środowisk przedprodukcyjnych. Proporcje ludzi z zespołu developerskiego i niezależnych,

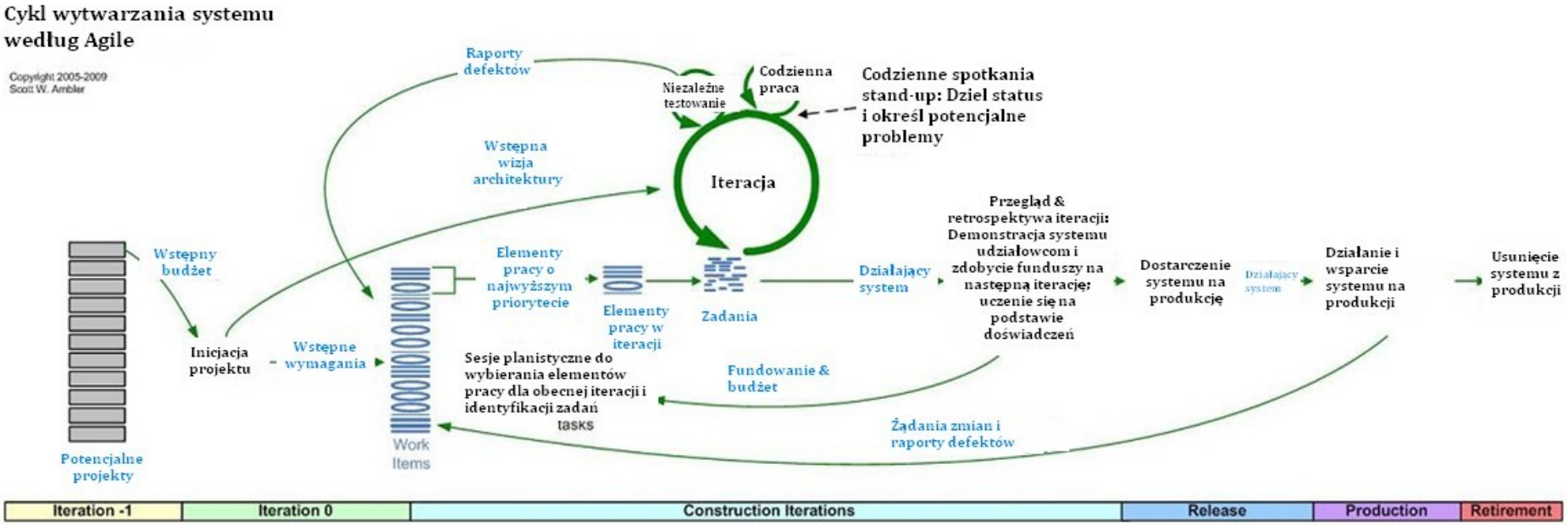
wspierających ich testerów, często wynoszą od 10: 1 do 15:1. W większych organizacjach te niezależne zespoły testowe zwykle wspierają kilka zespołów developerskich - tym samym umożliwiając wykonanie bardziej wyszukanych testów integracji systemu, ponieważ łatwiej mogą pracować z wersjami wielu wytwarzanych systemów.

Iteracje wydań. Celem iteracji wydań (ciemnoniebieskich iteracji "R" na Rysunku 1) jest udane zainstalowanie twojego systemu na produkcji. W praktyce może to być dość złożone, włączając w to szkolenia użytkowników końcowych i ludzi „operacyjnych", komunikację/marketing związany z wydaniem produktu; backup i potencjalne poprawki (jeśli coś pójdzie źle); wdrożenie pilotowe/stabilizacja systemu; ostateczne tłumaczenie UI oraz dokumentacji; finalizowanie dokumentacji systemowej i użytkowej; itd. Podczas iteracji wydania nadal jest nieco testowania na końcu cyklu życia, by upewnić się, że system jest gotowy na produkcję.



Rysunek 1 Wysoko-poziomowy cykl życia wytwarzania Agile

Rysunek 2 prezentuje szczegółową wersję SDLC, rozwijając detale z Rysunku 1. Rysunek 2 dodaje również nowe fazy, aby zobrazować pełen cykl życia, od początku do końca, włącznie z „iteracją 1”, gdzie identyfikujemy potencjalny projekt, fazą produkcji – w której działamy i wspieramy system po tym, jak już został wydany oraz fazą spoczynku, w której całkowicie usuwamy zbędny system z produkcji.



Rysunek 2 Szczegółowy SDLC Agile

Tradycyjny cykl życia wytwarzania systemu

Rysunek 3 przedstawia Model V wytwarzania oprogramowania, a w zasadzie bardziej złożoną formę tradycyjnego modelu wodospadowego. W modelu V, czynności po lewej stronie diagramu są walidowane później w cyklu życia poprzez odpowiadające im, zachodzące później w cyklu życia, aktywności (np. wymagania są walidowane poprzez testy akceptacyjne, architektura poprzez testy integracyjne itd.). Mimo, że to podejście jest lepsze, niż brak testowania w ogóle, okazało się bardzo drogie w użyciu z powodu kilku problemów systemowych:

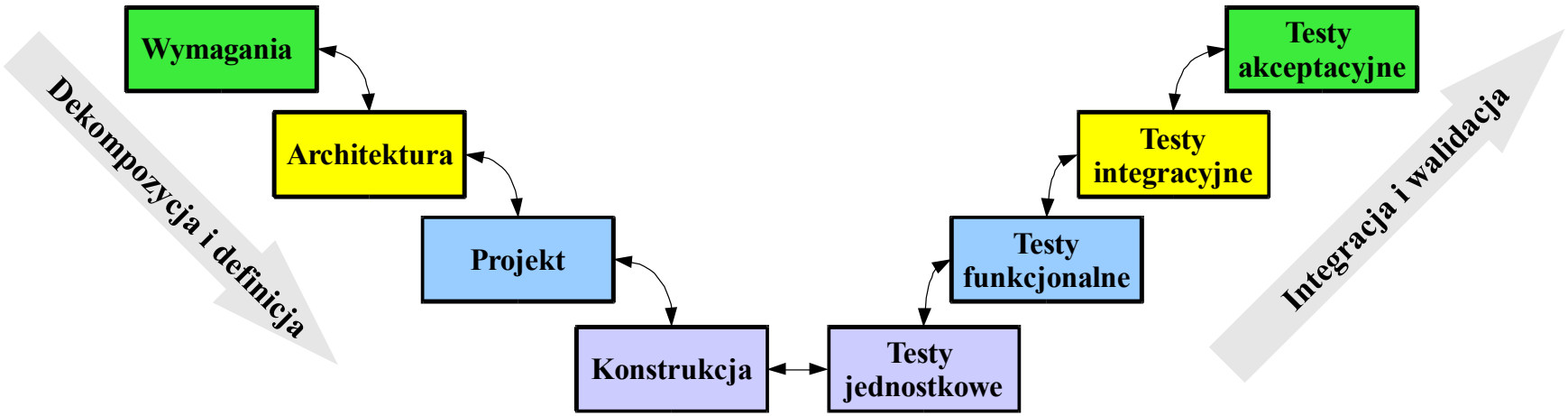
Dostarczanie złej funkcjonalności. Model V promuje podejście, w którym wymagania są definiowane szczegółowo we wczesnej fazie projektu. Mimo, że może to być dobrą teorią (zwróćcie uwagę na użycie słowa „może"), w praktyce okazuje się bardzo kiepskim sposobem pracy. Takie podejścia: „duże wymagania od początku" (ang. *big requirements up front*, BRUF), powodują znaczące straty, ponieważ w najlepszym wypadku zespół projektowy będzie tworzył coś pod specyfikację, zamiast czegoś, czego naprawdę potrzebują udziałowcy.

Tworzenie kruchego projektu. Podobnie – mimo, że w teorii dobrym pomysłem może być obmyślanie szczegółów architektury/projektu na początku, w rzeczywistości kończymy na podejmowaniu, a następnie zatwierdzaniu technicznych decyzji za wcześnie – kiedy jest dostępna najmniejsza ilość informacji (podczas faz architektury/projektu podejmujemy decyzje oparte na tym, co jak masz nadzieję będzie działać, w przeciwieństwie do tego, o czym wiemy, że będzie działać – w oparciu o aktualnie działające oprogramowanie).

Przekazywanie [produktów] powoduje defekty. Za każdym razem, kiedy musisz przekazać coś pomiędzy dwoma grupami ludzi, z powodu nieporozumień do produktu będą wkradać się defekty. Mimo, że ten problem może być częściowo złagodzony przez przeglądy, okazuje się to być kosztowne porównując do bardziej zwinnych podejść, takich jak *non-solo development* (od redakcji: podejścia non-solo to np. programowanie parami w XP, czy Zwinne modelowanie z innymi (ang. ***A****gile Modeling Model With Others*)).

Drogie naprawianie defektów. Im dłuższej trwa cykl informacji zwrotnej, tym wyższy średni koszt naprawy znalezionego błędu. Podejście V modelu promuje bardzo długie cykle informacji zwrotnej.

Zwiększony czas do dostarczenia wartości. Model V wydłuża ilość czasu, jaki zabiera dostarczenie funkcjonalności na produkcję, poprzez zwiększoną biurokrację i czas oczekiwania. To z kolei zmniejsza możliwe korzyści i wartość zysku (ang. *net present value* (NPV)) uzyskaną przez dostawę.



Rysunek 3 Seryjny cykl SDLC/Model V Model

W jaki sposób Agile jest inne?

Tradycyjny profesjonaliści testowania, którzy robią ruch w kierunku wytwarzania Agile, mogą uznać następujące aspekty zwinnego rozwijania za zdecydowanie różne od tego, co zwykli robić:

Lepsza współpraca. Developerzy Agile pracują blisko razem, preferując bezpośrednią komunikację, niż wzajemne przekazywanie sobie dokumentacji. Oni wiedzą, że dokumentacja jest najmniej efektywnym sposobem komunikacji pomiędzy ludźmi.

Krótszy cykl pracy. Czas pomiędzy szczegółowym określeniem wymagań, a walidowaniem tych wymagań jest teraz rzędu minut, nie miesięcy czy lat – z powodu zastosowania podejść wytwarzania kierowanego testami (ang. *test-driven development*), lepszej współpracy i mniejszego zaufania do tymczasowej dokumentacji.

Praktycy Agile przyjmują zmianę. Developerzy Agile wolą traktować wymagania jako spriorytetyzowany stos, który może się zmieniać podczas cyklu życia. Zmienione wymaganie jest korzyścią konkurencyjną, jeśli jesteś w stanie je zaimplementować.

Większa elastyczność wymagana od testerów. Minęły dni, w którym zespół developerski przekazywał „kompletną specyfikację", którą testerzy mogą znów testować. Wymagania ewoluują podczas projektu. W sytuacji idealnej przed wyprodukowaniem kodu, który je spełnia, pisane są „*story test*" poziomu akceptacyjnego, co implikuje, że testy stają się szczegółowymi specyfikacjami wymagań.

Większa dyscyplina wymagana od IT. Jest bardzo łatwo powiedzieć, że mamy zamiar pracować blisko ze swoimi udziałowcami, respektować ich decyzje, produkować potencjalnie gotowe oprogramowanie w regularnych odstępach czasu i pisać pojedynczy test przez napisaniem takiej ilości kodu produkcyjnego, który spełni ten test – ale znacznie trudnej rzeczywiście to robić. Wytwarzanie Agile wymaga o wiele więcej dyscypliny, niż tradycyjny development.

Większa odpowiedzialność wymagana od udziałowców. Jedną z implikacji zastosowania praktyk aktywnego udziału udziałowców, priorytetyzowanych wymagań i produkowania działającego oprogramowania w regularnych odstępach jest to, że udziałowcy są teraz rozliczani za decyzje, które podejmują.

Wymagany większy zakres umiejętności. Bycie tylko testerem, czy tylko programistą, czy analitykiem, czy tylko… kimkolwiek, to za mało. Praktycy Agile odchodzą od Taylorystycznych podejść tradycyjnego developmentu, które motywują ludzi do stania się specjalistami, a zamiast tego skłaniają się do wysoce iteracyjnego i kolaboracyjnego podejścia, które wymaga specjalistów uogólniających (ang. *generalizing specjalist*) (NIE ogólnych!)

Porównanie podejść Agile i tradycyjnego

Podejście Agile oferuje wiele korzyści w stosunku do tradycyjnego modelu V:

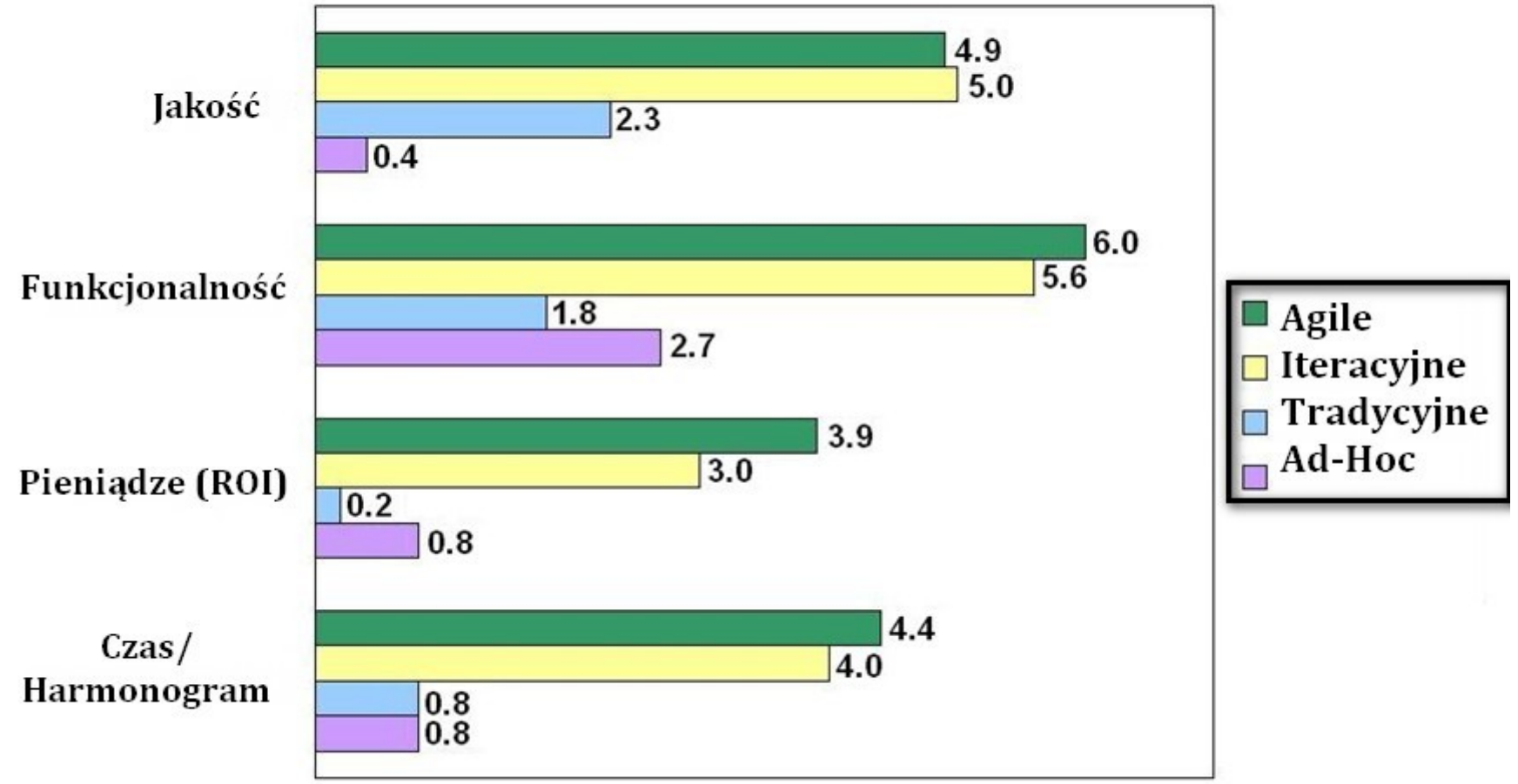
Większa zdolność dostarczenia wymaganej funkcjonalności. Zespół Agile pracuje ściśle ze swoimi udziałowcami, idealnie przestrzegając praktyki aktywnego udziału udziałowców. Ta praktyka, w połączeniu z podejściem ewolucyjnym, daje w wyniku większą zdolność zespołu Agile do zrozumienia a następnie zaimplementowania rzeczywistych potrzeb ich udziałowców. Rysunek 4 podsumowuje wyniki z Badania Sukcesu Projektu przeprowadzonego w 2008 r. przez Dr. Dobb Journal, pokazując, że zespoły zwinne są bardziej efektywne w dostarczaniu pożądanej funkcjonalności, niż zespoły tradycyjne.

Wyższa jakość. Rysunek 4 pokazuje też, że podejścia zwinne dają w wyniku wyższą jakość, niż podejścia tradycyjne, najprawdopodobniej z powodu zwiększonej współpracy wewnątrz zespołu oraz wcześniejszego i intensywniejszego testowania podczas cyklu życia.

Ulepszone projekty. Strategie architektury i projektowania Agile są z natury ewolucyjne – co, w połączeniu z wyższymi poziomami współpracy wykazywanymi przez zespoły zwinne, daje lepsze rezultaty, w porównaniu do podejść bardziej tradycyjnych. Architektura i projektowanie są tak ważne dla zespołów Agile, że wykonują te czynności podczas całego cyklu życia, nie tylko podczas wczesnych faz cyklu.

Lepsze wskaźniki ekonomiczne. Rysunek 4 pokazuje, że zespoły Agile dają większy zwrot z inwestycji, niż zespoły tradycyjne. Wynika to z krótszego cyklu informacji zwrotnej w podejściach zwinnych, co obniża średni koszt rozwiązywania defektów. Co więcej, ponieważ zespoły zwinne pracują mądrzej, nie ciężej, często dostarczają funkcjonalność wcześniej (co Rysunek 4 również obrazuje), tym samym dając krótszy czas do dostarczenia wartości i większy zysk.

Efektywność Paradygmatów Developerskich
(Skala od -10 do +10)



Source: DDJ 2008 Project Success Survey
www.ambysoft.com/surveys/

Rysunek 4 Czynniki sukcesu wg paradygmatów (skala od -10 do +10).

W końcu – chciałem jeszcze zaakcentować to, że wyniki pokazane na Rysunku 4 nie są anomalią. Badanie Zastosowania Agile z roku 2008 wykonane przez DDJ odkryło również, że ludzie wierzą w to, iż zespoły Agile produkowały wyższą jakość, niż zespoły tradycyjne, powodując większą satysfakcję udziałowców i dostarczając wyższe poziomy produktywności.

c.d.n.

W następnym numerze: Strategie testowania i jakości Agile: dyscyplina ponad retoryką. Cześć I z IV: Strategie wymagań Agile.