

Programowanie Ekstremalne: dyscyplina inaczej

Bartosz Walter

Instytut Informatyki Politechniki Poznańskiej

1 Wprowadzenie

W porównaniu do historii społeczeństwa, historia inżynierii oprogramowania jest wprawdzie znacznie krótsza, bo sięga co najwyżej lat 60-tych ub. stulecia, jednak następujące po sobie okresy prosperity, zastoju i rewolucji wydają się znajome. Już pierwszy kryzys, który pojawił się przy tworzeniu pierwszych dużych systemów, takich jak OS/360, dowiódł, że metody inżynierii oprogramowania starzeją się niewiarygodnie szybko i często już po dekadzie okazują się bezradne wobec nowych potrzeb rynku. W kolejnych latach starano się definiować procesy, wprowadzono m.in. metodę punktów funkcyjnych, analizę strukturalną czy inspekcje artefaktów, jednak często nie wystarczało to na długo Syndrom LOOP, nazwany tak od akronimu zbudowanego z angielskich określeń *Late, Over time, Over budget, Poor quality*, od lat wisi nad głowami dostawców software'u niczym miecz Damoklesa.

W latach 70-tych i 80-tych zaczęło upowszechniać się przekonanie, że źródłem zła jest samowola programistów i bałagan organizacyjny, w tym brak jasno zdefiniowanych reguł współpracy, rozmyta odpowiedzialność, brak właściwych metod kontroli i zapewniania jakości. Na gruncie takich poglądów wyrosły standardy ISO 9000 i model CMM (*Capability Maturity Model*) opracowany przez SEI (*Software Engineering Institute*), kładące nacisk na konieczność posługiwania się udokumentowanymi procedurami. Jednak z czasem i one stały się przedmiotem krytyki. Z czasem proces tworzenia oprogramowania został sprowadzony do wytwarzania dokumentacji, której towarzyszył program. Ponadto, w czasach boomu internetowego, który zmienił model biznesowy wielu przedsiębiorstw, tworzenie oprogramowania za pomocą procedur i procesów okazywało się zbyt czasochłonne, nieefektywne, nieelastyczne, powodujące frustrację pracujących w nadgodzinach programistów. Dlatego rewolucja dająca upust frustracjom była nieunikniona...

Rewolucja, jaką było pojawienie się pod koniec lat dziewięćdziesiątych ubiegłego stulecia, nowych, lekkich metodyk, przywracała znaczenie kodowania, a procesy, dokumentację, hierarchie stawiała na dalszym planie. Celem stało się bardzo szybkie dostarczanie klientowi funkcjonalności potrzebnej do realizacji jego celów biznesowych, a nie ściśle podążanie za planem. Rola programisty, dotychczas zwykłego robotnika tłumaczącego formalną specyfikację na język programowania, została znów doceniona.

Tak zwany Manifest Zwinności (ang. Agile Manifesto), ogłoszony w 2001 roku, stanowił podsumowanie dotychczasowych osiągnięć w zakresie lekkich (odtąd nazywanych zwinnymi). Stwierdzał on, że dotychczasowe wartości: procesy, narzędzia, wyczerpująca dokumentacja, kontrakty oraz plany są ważne, ale ważniejsze są osowości, interakcje, działające oprogramowanie, współudział klienta przy tworzeniu systemu oraz podążanie za zmianą. Zwinne metodyki zatem starały się nie tyle zburzyć istniejący świat, co raczej zaproponować inne jego urządzenie.

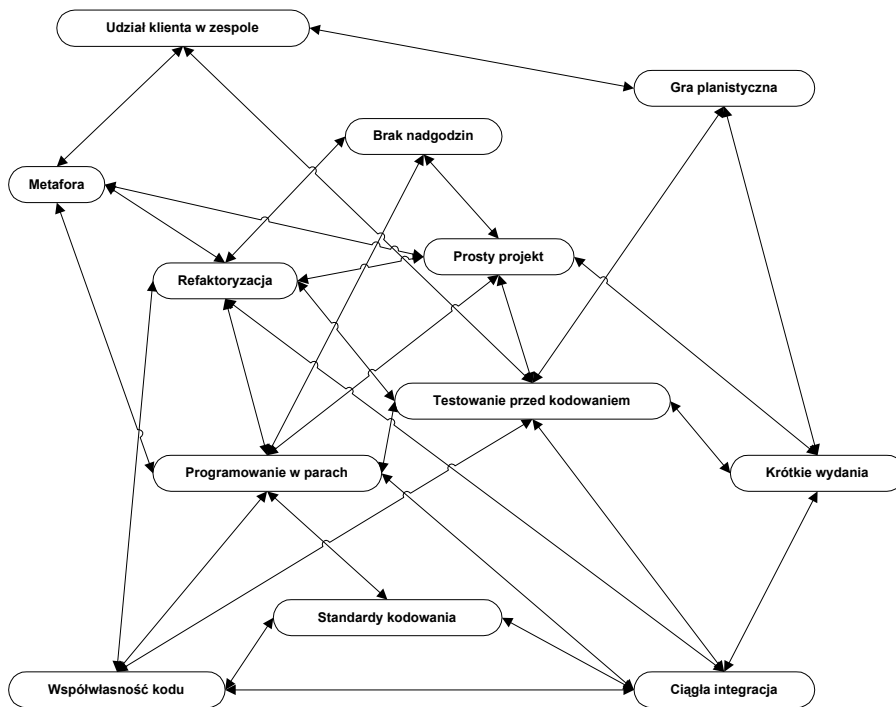
Spośród zwinnych metodyk największą popularność zdobyło Programowanie Ekstremalne (XP). Prowokująca nazwa sugeruje wprawdzie zupełnie nową jakość, jednak XP jest właściwie kompilacją pomysłów, które są doskonale znane, a jedynie nie były stosowane razem. Jednak to właśnie XP stało się synonimem nowych metodyk i wzbudziło – i nadal wzbudza – spore kontrowersje. Dlatego w dalszej części spróbujemy przyjrzeć się nieco dokładniej Programowaniu Ekstremalnemu, jego praktykom i ocenić zmiany, jakie wprowadza.

2 Przegląd XP

Programowanie Ekstremalne, wbrew nazwie, jest metodyką zdyscyplinowaną. Zdaniem Kenta Becka, twórcy XP, u jej podstaw leżą cztery podstawowe wartości:

- uczciwa komunikacja wewnątrz zespołu (którego członkiem jest klient),
- prostota (rzeczy proste łatwo zrozumieć)
- informacja zwrotna od klienta,
- i wreszcie odwaga, aby umieć zmieniać system i dostosowywać do potrzeb klienta.

Tym czterem wartościom odpowiada dwanaście podstawowych, mocno ze sobą powiązanych praktyk XP (rys. 1). Przyjrzyjmy się najważniejszym i najbardziej egzotycznym z nich, budzącym największe spory. Zostały one pogrupowane na obszary, na które zwykle dzieli się proces tworzenia oprogramowania.



Rysunek 1. Praktyki XP i ich powiązania (wg K. Becka)

2.1 Obszar planowania i definiowania wymagań

W XP, inaczej niż w metodach klasycznych, horyzont planowania nigdy nie wykracza poza jedno wydanie, czyli okres dostarczenia klientowi nowej wersji systemu (kilka tygodni do kilku miesięcy). Wynika to z przekonania, że klient może na każdym etapie zmienić zakres prac, a więc wysiłek włożony w przygotowanie daleko siężnego planu może okazać się daremny.

Przedstawiciel klienta jest w XP postacią kluczową. Nie jest jedynie źródłem informacji potrzebnych do opracowania specyfikacji: to on planuje pracę, zamawia

funkcjonalność i decyduje o bieżących priorytetach prac. Zakres prac, jakie mają być wykonane, jest ustalany podczas tzw. **gry planistycznej**, która decyduje o zakresie prac w najbliższym wydaniu bądź iteracji, a także określane są *priorytety* poszczególnych funkcji. Efektem gry jest *lista funkcji* (opisanych za pomocą opowieści użytkownika), które są najbardziej wartościowe dla klienta, z oszacowaniem czasu realizacji i przydzielonymi wykonawcami.

Klient przedstawia programistom opis funkcji systemu w postaci tzw. opowieści użytkownika (ang. *user stories*). Są to nieformalne, opisane w języku naturalnym wymagania, jakie ma spełnić system. Na tej podstawie programiści szacują pracochłonność każdej z funkcji, a klient, biorąc pod uwagę ich priorytety oraz dostępny "budżet czasowy", decyduje o ich realizacji. Decyzja ta może w trakcie realizacji ulec zmianie, dlatego gra planistyczna czasem jest powtarzana wielokrotnie.

Ponieważ decyzja o realizacji poszczególnych funkcji spoczywa wyłącznie w rękach klienta, gra planistyczna nie wprowadza typowego zagrożenia niespełnienia celu biznesowego klienta, a przede wszystkim znacznie zmniejsza ryzyko nieporozumień i tarć między klientem a programistami. Z drugiej jednak strony fakt, że planowanie odbywa się w bardzo krótkiej, kilkutygodniowej perspektywie, często jest nie do przyjęcia w przypadku dużych przedsięwzięć.

Nazwa 'gra planistyczna' dobrze oddaje charakter tej praktyki XP: zespół wraz z klientem 'rozgrywa' między sobą rozmiar funkcjonalności, którą należy zaimplementować, i możliwości jej realizacji w założonym terminie. Decyzję o aprobującej efekty gry podejmuje cały zespół (wraz z przedstawicielem klienta) przez konsensus.

Jak można zauważyć, Programowanie Ekstremalne zakłada znacznie ściślejszą współpracę klienta z zespołem wykonawców. Przedstawiciel klienta jest członkiem zespołu nie tylko nominalnie, ale realnie: na bieżąco i stale uczestniczy w przedsięwzięciu, posiada pełną wiedzę dziedzinową i jest w stanie podjąć wszelkie decyzje projektowe, w tym planistyczne. Oznacza to również jego ciągłą obecność w miejscu pracy zespołu wykonawców i gotowość do odpowiadania na pytania programistów. To ostatnie założenie jest często krytykowane, gdyż oznacza wyłączenie tej osoby z rutynowej pracy dla organizacji klienta, co dla klienta wiąże się z dodatkowym kosztem. K. Beck argumentuje jednak, że jeżeli koszt w postaci oddelegowania osoby do

pracy w zespole jest wyższy od spodziewanego zysku z szybszego wdrożenia lepszego systemu, to może budowa systemu jest w ogóle nieopłacalna. Ponadto przedstawiciel klienta w rzeczywistości nie odpowiada na pytania programistów przez cały czas, a więc wolne chwile może częściowo poświęcić na zwykłe obowiązki.

2.2 Obszar projektowania i kodowania

W Programowaniu Ekstremalnym nie istnieje wydzielona faza projektowania. Jest ona powiązana z implementacją i fazą definicji wymagań tworzących razem przyrost funkcjonalności. Nie ma również dokumentu opisującego projekt. Jedynym artefaktem, z którego wynika architektura i model systemu, jest kod i komentarze w nim zawarte. Takie rozwiązanie, wprowadzające potencjalny element krytyczny w postaci kodu, wymaga zdyscyplinowania procesu implementacji, a więc wprowadzenia standardu kodowania i komentowania, zasad zarządzania zmianami i przyrostowego rozwijania funkcjonalności.

Dlatego obie tytułowe czynności zostały w XP silnie ze sobą związane i tworzą w rzeczywistości jeden proces. Każda para programistów, podejmując się realizacji danej opowieści użytkownika, odpowiada jednocześnie za jej zaprojektowanie, implementację i przetestowanie. Oznacza to rezygnację ze szczegółowego projektu całego systemu na rzecz **metafory** – opisu architektury w języku zrozumiałym dla wszystkich udziałowców systemu. Metaforę tworzy wspólnie cały zespół. Stanowi ona pomost między klientem i programistami, dlatego powinna posługiwać się językiem zrozumiałym dla obu stron. Celem metafory nie jest pełne opisanie struktury i zachowania systemu. Ma ona dać jedynie bardzo ogólne wyobrażenie systemu oraz porównanie jego działania do wzorców znanych z rzeczywistości. Język metafory to zatem niekoniecznie wyłącznie język dziedzinowy, ale właśnie język przenośni, zrozumiały dla obu stron, w którym nawet za cenę pewnych nieścisłości, najłatwiej wyrazić ogólny zarys systemu.

We innych zwinnych metodykach rolę metafory odgrywa modelowanie architektoniczne. W przypadku XP zrezygnowano ze ścisłego definiowania architektury na rzecz komunikatywności i zrozumienia roli, jaką odgrywają poszczególne elementy.

Ponieważ kształt całego systemu jest nieznany, dlatego nie warto również spekulować i projektować struktur pod funkcje, które być może nigdy nie zostaną zaimplementowane (tzw. *up-front design*). **Prosty projekt** wg XP to taki, który:

- przechodzi pomyślnie wszystkie testy,
- nie ma redundancji logicznych,
- potrafi czytelnie przekazać odbiorcy zamysł projektanta,
- zawiera najmniejszą możliwą liczbę niezbędnych komponentów.

Aby projekt, modyfikowany poprzez dodawanie nowych funkcji, pozostał otwarty na zmiany, stosuje się **refaktoryzację**. Polega ona na zmianie wewnętrznej struktury programu, usuwaniu redundancji, czyszczeniu kodu - ale bez zmiany jego funkcjonalności. System po refaktoryzacji nadal musi przejść pomyślnie wszystkie testy jednostkowe. Dlatego refaktoryzacja w XP jest ściśle związana z innymi praktykami dotyczącymi programowania, przede wszystkim programowaniem parami, współwłasnością kodu oraz ciągłą integracją systemu.

Aby ją ułatwić i obniżyć jej koszty (klient płaci przecież za funkcjonalność, a nie projekt), wiele narzędzi CASE posiada wsparcie dla refaktoryzacji i testowania.

Szybka reakcja klienta na kolejną wersję programu jest jedną z podstawowych wartości uznawanych przez XP, dlatego tak ważne jest szybkie wdrożenie kolejnych wersji. Beck sugeruje, aby wydania były tworzone co kilka tygodni, a najdalej co kilka miesięcy, po zaimplementowaniu kilku-kilkunastu nowych opowieści użytkownika.

Programiści XP **pracują w parach**. Koncepcja jest bardzo prosta: skoro inspekcje są wydajnym sposobem podnoszenia jakości oprogramowania, najlepiej, aby trwały one cały czas. Jeden programista pisze kod, a drugi na bieżąco go weryfikuje, przy czym partnerzy zmieniają się zarówno wewnątrz pary, jak i na zewnątrz. Programowanie parami w znaczny sposób ułatwia komunikację pomiędzy członkami zespołu. Problemy przedstawiane za pomocą języka naturalnego w otwartej dyskusji są bardzo często różnie rozumiane przez osoby uczestniczące w dyskusji. Jednak po przełożeniu tej samej myśli na kod programu, czyli do postaci sformalizowanej, pozostali programiści śledząc zawartą w nim logikę, są w stanie dokładnie odtworzyć tok myślenia autora. Mechanizm komunikacji poprzez kod programu jest zatem efektywny i

skuteczny. Należy również zwrócić uwagę, że taki sposób komunikacji ułatwia naukę i przyspiesza proces rozwiązywania problemów. Obserwator analizuje podejście swojego partnera do implementacji, co może mu podsuwać na myśl inne, jeszcze lepsze rozwiązania. Wiedza na temat różnych technik programistycznych oraz inne umiejętności wykorzystywane podczas pracy przez jednego programistę, w naturalny sposób są przekazywane drugiemu programiście. Mając na uwadze fakt, że skład par często się zmienia, w efekcie czego każdy programuje z każdym, można stwierdzić, że wiedza i umiejętności jednostek rozpraszają się wśród wszystkich pracowników firmy.

Programowanie parami wymaga od programistów szczególnego podejścia do pracy oraz kilku ważnych umiejętności. Programowania parami można się nauczyć, ale w wielu przypadkach wymaga to zerwania z dotychczasowym rozumieniem istoty pracy w grupie. Podstawową wartością programisty jest komunikacja z innymi członkami zespołu. Indywidualista niechętnie dzielący się swoją wiedzą i obserwacjami nigdy nie stworzy skutecznej pary.

Zmiany wprowadzane w kodzie powodują, że staje się on **wspólną własnością** całego zespołu – każdy może go uzupełniać i poprawiać. Żaden fragment kodu nie posiada swojego właściciela – jedyne uprawnione do wykonywania zmian. Wręcz przeciwnie, programiści są pod tym względem niemal anonimowi, a kod jest efektem pracy całego zespołu. W ten sposób unika się zamykania szczegółów implementacyjnych w głowie jednej osoby, która może przecież zachorować lub zmienić pracę. Konieczny jest także **standard kodowania**.

W celu obniżenia koszt integracji nowych funkcji, cały system powinien być budowany ze źródeł **przynajmniej raz dziennie**. Żeby nie wprowadzać bałaganu, w danej chwili tylko jedna para może dokonywać integracji swoich funkcji. Czynność tę można zautomatyzować i połączyć z wykonaniem testów, np. przy użyciu narzędzi typu Gump czy Maven. W ten sposób system pod koniec każdego dnia tworzy spójną całość, którą można skompilować i która nie zawiera błędów. Przy integracji okazjonalnej jej koszt często bywa wypadkową jakości interfejsów modułów, wiedzy i zdolności poszczególnych programistów oraz ich umiejętności komunikowania się. W przypadku XP integracja następuje na bieżąco i błędy wynikające z niedopasowania

interfejsów są usuwane w miarę ich pojawiania się. Całkowity koszt takiej integracji, mimo że zdarza się ona znacznie częściej, jest niższy, a ponadto w każdej chwili można przygotować wydanie systemu dla klienta.

Częste wydawanie kolejnych wersji programu i przekazywanie ich klientowi ma dwie podstawowe zalety. Przede wszystkim, system jest często poddawany walidacji przez klienta, który następnie decyduje o kierunku dalszych prac. Ponadto daje zespołowi poczucie kontroli nad przedsięwzięciem i redukuje ryzyko błędu podczas integracji, ponieważ następuje ona w sposób ciągły.

Obowiązująca w XP zasada braku nadgodzin jest praktyką o raczej socjalnym charakterze, ale ponieważ dotyczy programistów, została zakwalifikowana jako praktyka programistyczna. Jest ona często postrzegana jednoznacznie jako przyjazna wyłącznie programistom, natomiast klient nie odnosi z niej żadnej korzyści. Jednak nie zawsze jest tak w istocie. Wiadomo, że przeciętny człowiek (czyli także programista) nie może efektywnie pracować dłużej niż ok. 40 godzin tygodniowo przez dłuższy czas. Dlatego w przedsięwzięciu XP, w którym kolejny tydzień z rzędu programiści pracują w nadgodzinach, istnieje znacznie poważniejszy problem, który nie może być rozwiązany przez wydłużenie czasu pracy. Takie przedsięwzięcie, jak każde przekraczające założony harmonogram, wymaga szybkiej i zdecydowanej interwencji osób zarządzających. Programowanie Ekstremalne, wbrew swej nazwie, wymaga powtarzalności i regularności, czego nie można zapewnić przy zaburzeniach w harmonogramie.

2.3 Obszar testowania

Testowanie w XP jest wykonywane w dwojaki sposób: na poziomie testów jednostkowych, przez każdego programistę, oraz poprzez testy akceptacyjne, przez przedstawiciela klienta. Oba rodzaje testów powstają **przed implementacją**.

Każda para programistów przed rozpoczęciem implementacji, a już po zaprojektowaniu nowej funkcji powinna zaprojektować przypadki testowe sprawdzające poprawność poszczególnych klas i metod. Przypadki te następnie są implementowane równoległe z samą funkcją, a na końcu – wszystkie muszą zostać pomyślnie zweryfikowane. Aby uznać napisany fragment kodu za poprawny, musi on przejść pomyślnie **wszystkie** testy jednostkowe. Jeżeli którykolwiek test zawiedzie,

wszystkie testy jednostkowe. Jeżeli którykolwiek test zawiedzie, naprawienie kodu otrzymuje najwyższy priorytet spośród wszystkich realizowanych zadań programistycznych. Zdarza się jednak, że błąd jest wykrywany później: wówczas koniecznie należy stworzyć przypadek testowy wykrywający ten błąd. Przypadki testowe wraz z kodem stanowią jedyny zapisany artefakt stosowany w XP.

3 XP w praktyce

Jak łatwo się domyślić, programiści przyjęli nowe metodyki z ciepłym zainteresowaniem, menedżerowie projektów – z chłodną rezerwą, a naukowcy – z zainteresowaniem. Co ciekawe, sytuacja ta właściwie nie zmieniła się do dzisiaj: trudno o jednoznaczną i miarodajną ocenę XP. Dlaczego? Bo XP nie jest równe XP.

Pierwszy raport z wdrożenia XP, słynny już projekt 3C z 1998 roku – pole ćwiczebne Kenta Becka, pozostał jednak jednym z niewielu „czysto” ekstremalnych. Zestaw praktyk XP, choć gorąco broniony przez Becka, okazuje się trudny do wdrożenia w całości, dlatego wiele kolejnych projektów jest ekstremalnych zaledwie w większym lub mniejszym stopniu, co znacznie utrudnia jednoznaczną ocenę.

Poza implementacją całej metodyki poddawano badaniom pojedyncze praktyki, a przede wszystkim programowanie parami. Podstawowy zarzut to konieczność poświęcenia pracy dwóch osób na zadanie, które mógłby wykonać jeden programista. Wprawdzie opublikowane wyniki badań pokazują, że narzut dodatkowej pracy waha się od ok. 50 do 80%, jednak nie uwzględniały one spodziewanej poprawy jakości. Badania te były prowadzone na studentach i przy wykorzystaniu niewielkich zadań, dlatego trudno mówić o wyczerpującym i kompletnym eksperymencie. W praktyce programowanie parami jest stosowane przy implementacji trudniejszych fragmentów systemu, natomiast większość systemu jest tworzona w tradycyjny sposób. Programowanie parami pozostało dla wielu osób papierkiem lakmusowym XP, jego sztandarową cechą, co często przysłańiało pozostałe praktyki.

Kolejna wątpliwość związana z XP to niemal całkowity brak dokumentacji (jedynie kod źródłowy i przypadki testowe), a co za tym idzie – potencjalne problemy w fazie po wdrożeniu oprogramowania. Faktycznie, problemy te mogą się pojawić –

jeżeli XP jest stosowane niewłaściwie. Cykl życia XP to właściwie niekończący się okres utrzymywania systemu – tworzenie nowych funkcji biegnie równoległe z poprawianiem błędów w istniejącej wersji. Projekt XP żyje tak długo, jak długo są w nim wprowadzane zmiany, bo tylko w ten sposób dokumentacja utrzyma się w pamięci (niestety, często ulotnej) programistów. Rotacja programistów i przejrzystość kodu w parach ma temu zapobiec – i to wystarcza w przypadku niewielkich systemów i budujących je zespołów. Nie są znane przypadki zastosowania XP do realizacji większych projektów.

Pytanie o jakość właściwie nasuwa się samo. Czy taki anarchizujący proces może dać w wyniku dobre jakościowo oprogramowanie, spełniające wymagania klienta? Paradoksalnie, prowadzone badania dowodzą, że XP wcale nie jest tak chaotycznym procesem, jak wydaje się na pierwszy rzut oka. Mark C. Paulk, współtwórca modelu CMM ocenił, że XP implementuje praktyki 2 poziomu tego modelu, spełniając także kilka wymagań poziomów wyższych. Ponadto uzupełnienie XP o kilka dodatkowych praktyk pozwala znacznie ustabilizować proces zarządzania wymaganiami wg modelu Sommerville’a–Sawyer’a.

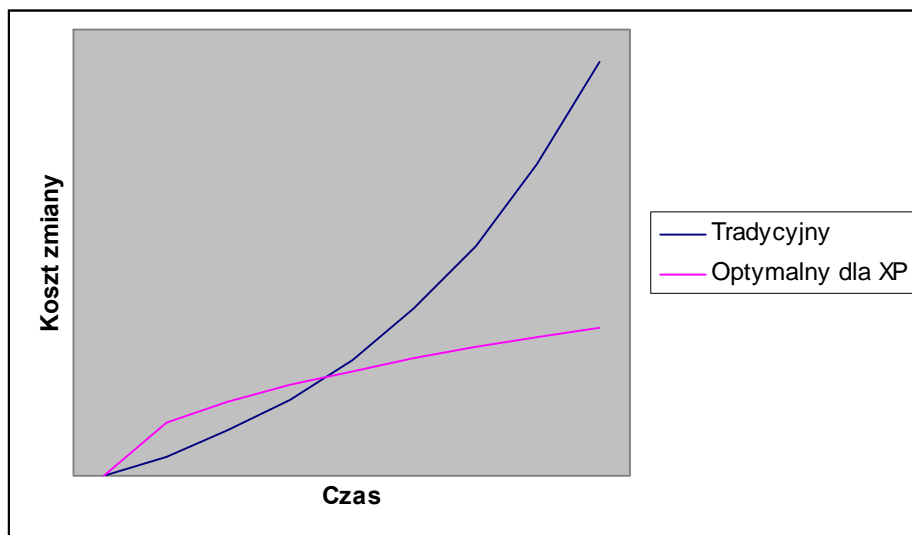
Z drugiej jednak strony jakość produktu – to, za co płaci klient – opiera się wyłącznie na testach jednostkowych. Jeżeli one zawiodą (będzie ich zbyt mało lub będą niedokładne), nawet dojrzały proces nie spowoduje, że oprogramowanie będzie działało poprawnie.

Jedno z najpoważniejszych nieporozumień dotyczących XP ma źródła w prowokującej nazwie: programowanie ekstremalne. Wiele organizacji, które mówiąc eufemistycznie, zostałyby sklasyfikowane na poziomie 1 modelu CMM, ogłasza z dumą, że stosują XP. XP to nie jest brak procesu; przeciwnie – oznacza zdyscyplinowaną metodykę, która ekstremalnie poważnie podchodzi do zagadnień jakości oprogramowania, i która wymaga stosowania określonych praktyk. Stąd biorą się lekceważące uśmiešky na ustach niektórych krytyków XP, dla których oznacza ono tylko bałagan i brak jakiegokolwiek procesu.

XP z pewnością nie nadaje się do stosowania w wielu projektach, dlatego warto określić granice jego stosowania. Pierwsze ograniczenie to zapewne rozmiar zespołu: Beck wspomina o kilkunastu osobach, inni – dwudziestu. Nie wydaje się, aby XP

działało skutecznie przy większych zespołach. Podobnie wygląda sprawa z czasem trwania projekty: raporty wspominają o przedsięwzięciach co najwyżej kilkunastomiesięcznych (choć wypada wspomnieć, że w tym czasie wypuszczanych jest kilkanaście wersji systemu).

XP zakłada bowiem zupełnie inny model współpracy między zamawiającym a wykonawcą. Nie jest to kontrakt o ustalonej cenie, terminie i zakresie – bo wówczas XP nie może pokazać swoich zalet. Najlepszym środowiskiem jest "abonament" na rozwijanie systemu, usługa wykonywania zleceń klienta na rozbudowę systemu. Wymaga to jednak znacznie większego zaufania po obu stronach i delegowania przedstawiciela klienta do pracy w zespole.



Rysunek 2. Tradycyjna i optymalna dla XP zależność kosztu utrzymania od czasu (wg K. Becka)

XP, zdaniem Becka nie nadaje się także do realizacji systemów w technologiach o wykładniczym koszcie wprowadzenia zmiany (rys. 2), natomiast wydaje się szczególnie uzasadnione w sytuacji, gdy zespół dobrze zna technologię, a koszt zmiany w czasie jest spłaszczony.

4 Podsumowanie

Skuteczność XP zależy od tak wielu czynników, że nie podano jeszcze uniwersalnej recepty na jego stosowanie. Pozostaje ono domeną rosnącego grona entuzjastów, czego dowodem jest choćby organizacja Agile Alliance i jej Manifest Zwinności. Jednak coraz większa liczba raportów, konferencji i książek świadczy, że temat ten budzi zainteresowanie, także ze strony wielkich postaci w inżynierii oprogramowania (wystarczy wspomnieć T. DeMarco, M. Paulka czy T. Gilba).

XP wymaga dokładności i konsekwencji w stosowaniu poszczególnych praktyk z jednej strony, a z drugiej – elastyczności w dostosowywaniu procesu do lokalnych warunków. Utrudnia to wprowadzić ocenę metodyki, ale pozwala budować oprogramowanie. Dlatego prawdopodobnie nie uda się właściwie i ostatecznie zdefiniować relacji pomiędzy klasycznymi, ciężkimi metodykami budowy oprogramowania, a zwinnymi z XP na czele.

Rewolucja w oprogramowaniu, jaką wywołały, nieco przygasła razem z internetowym boomem, jednak nadal budzą one spory i kontrowersje. Czy i kiedy ta rewolucja spod znaku Agile Manifesto się zakończy – nie wiadomo...