1

MYŚLENIE ODWROTNE

Czym jest agile?

Większość naszych wielkich wynalazków i genialnych osiągnięć zawdzięczamy lenistwu,
czy to narzuconemu, czy dobrowolnemu. Umysł nasz lubi być karmiony, jak łyżeczką,
pomysłami innych ludzi, jeśli się go jednak pozbawi tej pożywki, zaczyna,
zrazu niechętnie, myśleć samodzielnie, a tego rodzaju myślenie, proszę pamiętać,
jest myśleniem oryginalnym i może przynieść bardzo cenne rezultaty.

Agatha Christie, *Zatrute pióro*

Szkoła przetrwania

Jestem wielkim fanem Beara Gryllsa, bohatera programu telewizyjnego emitowanego przez Discovery Channel — *Ultimate Survival* (polski tytuł: *Szkoła przetrwania*). Dzielny Bear, podróżnik i były żołnierz służb specjalnych SAS 21 (*Special Air Service*), uzbrojony jedynie w nóż, manierkę, krzesiwo i ubranie, pokazuje, jak przetrwać w absolutnie ekstremalnych warunkach. Pamiętam jeden z odcinków (właściwie chyba był to pilot 1. sezonu), który był kręcony w Górach Skalistych, w USA. Duże wrażenie zrobiła na mnie scena, w której Bear schodził w dół rzeki (było może z 9 metrów) samym środkiem wodospadu. Jeśli ktoś z Was widział ten odcinek, to wie, że Bear pokonywał go trochę „na raty". Najpierw pierwszy etap — dojście do wodospadu. Potem drugi — zdobycie małej półki skalnej, oczywiście niewidocznej ze względu na ogromną masę lodowatej wody, wpadającej wprost na Beara. Kolejny moment to zejście na półkę skalną, ale jak? — za pomocą drabinki zrobionej z linek paralotni, na której nasz bohater wylądował na początku programu. I wreszcie ostatni etap — skok z kilku metrów do ujścia rzeki. Myślę, że opisana scena „pokonywania wodospadu" bardzo dobrze pokazuje pułapkę, w jaką wpada większość z nas, stosując dobrze znany wszystkim tzw. *waterfall model* (z ang. *metodyka kaskadowa*). Na pierwszy rzut oka wydaje nam się, że nie ma innej drogi. Kiedy się pokonuje wodospad, można sobie przecież znacznie

skrócić drogę, tym samym szybciej dotrzeć do zamierzonego celu. Do kogoś, kto nie ma dużego doświadczenia w rozwijaniu oprogramowania, taki argument może faktycznie trafiać. Opiera się przecież na bardzo logicznych przesłankach — najpierw musimy wiedzieć, czego chce od nas klient (wymagania biznesowe), żebyśmy mogli myśleć o specyfikacji technicznej. Dopiero kiedy już mamy te dwa etapy za sobą, możemy rozpocząć prace architektoniczne i zająć się dokumentacją wybranych rozwiązań. Gdy już nam się to uda, wreszcie mekka! — upragnione pisanie kodu. Potem już tylko testy, retesty i — uwaga! — Wielki Finał, czyli demonstracja efektów naszej pracy klientowi, który dostaje oponę zawieszoną na linie zamiast wymarzonej huśtawki[1].

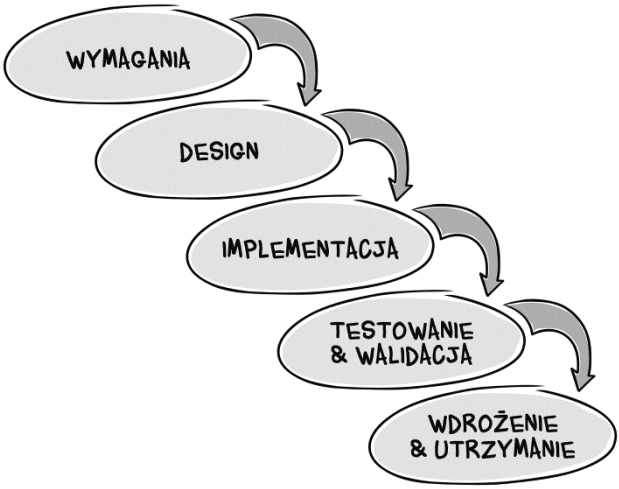
Tak sobie myślę, że gdybym ja, podobnie jak Bear Grylls, został rzucony na pastwę losu i musiał przetrwać w najdzikszych miejscach na świecie, na pewno nie schodziłbym w dół wodospadem. Znając moje fizyczne możliwości, dodatkowo biorąc pod uwagę fakt, że nie służyłem w SAS 21, z pewnością poszukałbym jakiejś innej drogi w dół — niekoniecznie takiej, która skazałaby mnie na połamanie sobie rąk i nóg na śliskich kamieniach lub na nabawienie się hipotermii od lodowatej wody. Metody agile (z ang. *zwinny*) to szybka droga do celu, która omija wodospad, bystrza i wszystko, co może się zdarzyć po drodze.

Wodospad

Z powstaniem zwinnych metod tworzenia oprogramowania było trochę tak, jak z powstaniem życia na Ziemi. Ich podstawowe idee, wyznawane wartości i zasady ewoluowały wraz z dyscypliną inżynierii oprogramowania, a więc od początku jej istnienia (przełom lat 50. i 60. XX w.). Niewątpliwym katalizatorem, który przyczynił się do ich rozwoju, był świat tradycyjnych metod wytwórczych, które najpełniej definiuje podejście zwane kaskadowym (od ang. *waterfall*). Polega na tym, że aktywności projektowe realizowane są liniowo (sekwencyjnie) — płyną niczym Nil, tworząc imponujące kaskady wodne (dwie z nich mają postać potężnych wodospadów — nazywanych Ripon i Owena). Cykl życia projektu dzielony jest na określone fazy, które wzajemnie od siebie zależą. Najpierw ustalane są potrzeby klienta (wymagania biznesowe), potem tłumaczy się je na język zrozumiały dla programistów (wymagania funkcjonalne), żeby z kolei, na ich podstawie, można

[1] Zob. *http://www.projectcartoon.com/cartoon/32* (dostęp: 3 maja 2012).

było zaprojektować określone rozwiązania techniczne (projekt systemu). Kolejna faza to implementacja wybranych rozwiązań, które są: integrowane, testowane i utrzymywane (ang. *maintenance*) w wyniku wdrożenia. Zwróćcie uwagę na to, że każda faza w tym podejściu stanowi domkniętą całość. Jej produkty wyjściowe (*outputs*) stanowią wejścia (*inputs*) do fazy następnej, co pokazuje rysunek 1.1.



Rysunek 1.1. Cykl kaskadowy projektu

Bardzo dobrze pamiętam problemy wynikające ze stosowania metody kaskadowej, z którymi sam, jako początkujący kierownik projektów, musiałem się kiedyś zmierzyć. Przede wszystkim dość szybko doszedłem do wniosku, że stałe wymagania w projekcie są tak rzadkie jak oscarowe role u Arnolda Schwarzeneggera. Wyobraźcie sobie sytuację, że skończyliście prace związane z przygotowaniem niskopoziomowego projektu systemu (ang. *Low Level Design*). Nagle dzwoni klient i mówi, że chciałby trochę rozbudować dwie ostatnie funkcjonalności, o których rozmawialiście pięć dni temu, i dodatkowo dorzucić jedną nową. Do dzisiaj myśleliście, że wszystko jest pod kontrolą. Nagle cały świat wywraca się do góry nogami, grawitacja nie działa zgodnie z powszechnym prawem ciążenia, a czasoprzestrzeń zakrzywia się w nieprawdopodobny wręcz sposób. Scena żywcem wyjęta z *Incepcji* Christophera Nolana[2]. Chciałoby się włamać przez sen do świadomości klienta i zaszczepić mu ideę cierpliwego czekania i „niewrzucania" nam dodatkowej roboty. Ale… nie ma sprawiedliwości na tym świecie. Musimy kilka rzeczy przeprojektować,

[2] *http://www.imdb.com/title/tt1375666/* (dostęp: 3 maja 2012).

przeplanować i starać się jakoś podnieść morale sfrustrowanego zespołu. Nie trzeba już chyba dodawać, jak bardzo takie „wrzutki" zwiększają koszty prowadzonego projektu.

Kolejnym problemem, który napotkałem przy okazji stosowania modelu kaskadowego, jest formalne pozbawienie prawa głosu naszego klienta w trakcie prowadzenia prac rozwojowych. Celowo napisałem „formalne", gdyż klient i tak kontaktował się z nami na wiele różnych sposobów i demonstrował swoje widzimisię. I nic nie mogliśmy zrobić. Nie pomagało alarmowanie o tym problemie wyższego kierownictwa, które, jak jedna z pluszowych zabawek stojących przy wejściu do Smyka, natrętnie powtarzało: „Takie jest życie! Dobrze wiesz, że jest to nasz strategiczny partner (czyt. dojna krowa) i musimy to jakoś znieść. Głowa do góry! Następnym razem będzie lepiej". Nie było.

Model kaskadowy umożliwia różnego rodzaju „ucieczki" błędów z jednej fazy do drugiej. Wyobraźcie sobie np., że na etapie testów systemowych nagle dowiadujecie się, że określone rozwiązanie zostało źle zaprojektowane i musicie wrócić do wcześniejszej fazy, rozgrzebać architekturę systemu, zaimplementować odpowiednie poprawki, zrobić testy i wdrożyć zmiany na środowisko produkcyjne klienta. W tym momencie Wasze plany biorą w łeb. To jest trochę tak, jak z wyjazdem na weekendowy biwak, na Mazury. Pakujemy się: śpiwór, karimata, ubrania, ręczniki (oczywiście wszystko razy kilka, chyba że nie bierzemy żony i dzieci), buty, kosmetyczka, latarka, zapałki, saperka, ładowarka do telefonu, konserwy… Wreszcie wyjeżdżamy z Krakowa. Nawet nam sprawnie poszło, mimo że jest piątek i 17.00. Jedziemy już około dwie godziny, nagle żona, patrząc błędnym wzrokiem na przejeżdżające sąsiednim pasem auta, pyta: „Krzysiek, a namiot?". Możecie sobie wyobrazić, jaki jest dalszy ciąg tej historii. Z modelem kaskadowym jest podobnie. Im później się zorientujemy, że nie mamy namiotu, tym więcej nas kosztuje powrót po niego.

W modelu kaskadowym nad sukcesem każdej fazy czuwa inny zespół, który skupia ludzi o bardzo zbliżonych kompetencjach. Na przykład za fazę wymagań odpowiadają analitycy biznesowi, analitycy systemowi oraz inżynierowie wymagań. Na etapie projektowania pierwsze skrzypce grają projektanci i architekci. Później do gry wchodzą programiści, którzy implementują przyjęte wcześniej rozwiązania, a wyniki swoich prac przekazują dalej testerom, którzy z kolei sprawdzają, czy wszystko działa zgodnie z wymaganiami, a więc z tym, czym zajmował się pierwszy zespół. Jeżeli wszystko jest przetestowane, a znalezione błędy poprawione, produkt trafia w ręce wdrożeniowców, a w następnej kolejności zespołu zajmującego się

pracami utrzymaniowymi. Proces ten przypomina trochę łańcuch pokarmowy, w którym mamy do czynienia z szeregiem różnych organizmów ustawionych w takiej kolejności, że każdy z nich jest źródłem pożywienia dla kolejnego. Na rysunku 1.2 bliżej niezidentyfikowana roślina jest pokarmem dla dżdżownicy, którą zjada ze smakiem na śniadanie mały żółwik, będący niezłym obiadowym kąskiem dla zmęczonego lataniem orła, wprost uwielbiającego te opancerzone stwory. Mamy tu więc i „zjadających", i „zjadanych".

Rysunek 1.2. Łańcuch pokarmowy

W modelu kaskadowym zespół, który zbiera i analizuje wymagania, jest pierwszym ogniwem łańcucha projektowego. Wytwory jego prac (lista wymagań klienta) są „zjadane" przez zespół projektantów („zjadających"), które z kolei dostarczają cennego pożywienia (projekt systemu) zespołom programistów. Łańcuszek ten zamyka się na etapie, kiedy klient konsumuje „gotowy produkt". W przyrodzie łańcuchy pokarmowe są długie i wzajemnie poprzeplatane — tworzą sieci różnego rodzaju zależności pokarmowych. I to jest coś, co nie jest brane pod uwagę w podejściu kaskadowym. Tam bowiem zakłada się płynne przejście pomiędzy jedną fazą a drugą.

Model kaskadowy, przez surowe rozgraniczenie prac wykonywanych przez różne zespoły, powoduje rozmycie odpowiedzialności za rozwój produktu. Każdy zespół skupia się tylko na swojej działce i w żaden sposób nie czuje się odpowiedzialny za to, co robi zespół kolejny. Każdy zamyka się w swoim silosie. Przypomina mi to pewną historię. Kilka lat temu byłem na

weselu u mojego znajomego. Nie wiem jak Wy, ale ja, delikatnie mówiąc, nie najlepiej znoszę wszelkiej maści zabawy weselne, które się przy tej okazji odbywają. No ale przecież nie wypadało odmówić. Gra była bardzo prosta. Polegała na tym, że goście weselni — zarówno ci, którzy zgłosili się dobrowolnie, jak i ci, tacy jak ja, dostali się do zabawy z „łapanki" — mieli utworzyć koło i kolejno podawać sobie małą łyżeczkę do herbaty. W tym czasie zespół pastwił się nad wesołymi weselnikami, grając skoczne „umpa... umpa...", od czasu do czasu robiąc niespodziewane przerwy. I właśnie te „przerwy", ni z gruchy ni z pietruchy, powodowały, że człowiek chciał jak najszybciej pozbyć się tej okropnej łyżeczki. I czym prędzej wcisnąć ją w ręce sąsiada. Jest to postawa, którą wyzwalała zasada tej zabawy, że gdy tylko muzyka przestaje grać, a ktoś zostanie z „problemem" w ręku, wypada z gry. Projekty w modelu kaskadowym przypominają bardzo zabawę z łyżeczką. Każdy zespół chce jak najszybciej „wypchnąć" to, co zrobił, do sąsiedniego zespołu — „Niech oni się tym martwią". „To nie jest już nasz problem". Ile razy słyszeliśmy takie hasła w swoim projekcie? Pamiętacie, jak nieraz dany problem (błąd, poprawka) był przerzucany między programistami, testerami, utrzymaniowcami lub osobami z obsługi klienta? „Przecież my zrobiliśmy to tak, jak było w wymaganiach, to ONI zawalili, nie MY!". Albo: „MY tego nie będziemy naprawiać, bo myśmy tego nie robili, to ich robota". W modelu kaskadowym poszczególne zespoły przypominają trochę monady, o których mówił Leibniz. Monady nie mają drzwi ani okien. Są światem samym w sobie. Żyją w radykalnej separacji. Nie komunikują się, bo, używając metafory, którą posłużył się niemiecki filozof — do tego potrzebne są drzwi i okna[3].

Winston Royce, który jako pierwszy w artykule *Managing the Development of Large Software System* (1970) opisał model kaskadowy, powiedział bardzo ciekawą rzecz: „Podoba mi się ten pomysł, ale jego implementacja jest ryzykowna i ma tendencję do błędów"[4]. Jest swoistym paradoksem, że wiele osób uważa tego człowieka za twórcę metodyki kaskadowej — człowieka, który widział w niej duże zagrożenie.

[3] Zob. F. Copleston, *Historia filozofii*, tłum. J. Marzęcki, t. IV, Instytut Wydawniczy PAX, Warszawa 1995, s. 299 – 300.

[4] W. Royce, *Managing the Development of Large Software System*, Proceedings of IEEE WESCON 26 (August): 1 – 9, *http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf* (dostęp: 3 maja 2012).

Pan Samochodzik i tworzenie oprogramowania

Metody agile powstały jako reakcja na założenie, które od samego początku było głęboko zakorzenione w inżynierii oprogramowania: „proces tworzenia oprogramowania jest procesem, który niczym nie różni się od procesu produkcyjnego". Jest linia produkcyjna, są stanowiska robocze (maszynowe, ręczne lub mieszane), pogrupowane według kolejnych operacji procesu technicznego. Idea „linii produkcyjnej", w momencie powstania, była alternatywnym rozwiązaniem wobec dotychczasowej produkcji rzemieślniczej i jej utworzenie stanowiło niekwestionowaną zasługę amerykańskiego koncernu Ford. Inżynierowie oprogramowania popełnili jednak zasadniczy błąd, próbując przenieść ten pomysł na grunt projektów software'owych. Co gorsza, na tym założeniu osadzona została cała dotychczasowa filozofia zarządzania ludźmi. Na czym ten błąd polegał? Wyobraź sobie, Drogi Czytelniku, że awansowałeś i zostałeś kierownikiem projektu w zakładzie Tomasza N. N. (tytułowy bohater książki *Pan Samochodzik*), któremu znudziła się już praca historyka sztuki, postanowił zacząć masową produkcję pokracznego wehikułu, zbudowanego na bazie rozbitego *Ferrari 410 Superamerica*. Jako menedżer odpowiadasz za linię produkcyjną. Natychmiast eliminujesz wszystkie pojawiające się defekty; jesteś wściekły i nie tolerujesz błędów popełnianych przez pracowników, którzy obsługują linię; traktujesz ich jak kolejny trybik maszyny, który w razie potrzeby zawsze można wymienić na inny; no i jesteś mistrzem wszelkich instrukcji — wszystko musi „tykać" jak w szwajcarskim zegarku i na wszystko jest standardowa procedura. Aha, jeszcze jedno: nie znosisz eksperymentów — nie ma czasu sprawdzać, czy coś się da wykonywać lepiej, czy nie, to nie jest przecież Twoja rola.

Takie podejście faktycznie ma sens i sprawdza się podczas pracy przy linii produkcyjnej, np. samochodów. Ogromnym błędem jest jednak próba zaaplikowania tego modelu do projektów software'owych, w których kluczową rolę odgrywa tzw. „czynnik ludzki". To od stopnia zaangażowania ludzi, ich kreatywności, umiejętności akceptowania problemów, radzenia sobie z konfliktami, zdolności komunikacji, pracy w grupie zależy sukces danego projektu. Stosowanie mechanizmów zaczerpniętych ze świata produkcji może prowadzić do postaw zupełnie odwrotnych — stłumienia inicjatywy, braku odwagi w wyrażaniu swoich opinii i pomysłów, chowania się do „kryjówek", z których nie trzeba się zbytnio wychylać, żeby przeżyć kolejny dzień w pracy.

Ludzie z kryjówek

Ponieważ prywatnie zawsze byłem i dalej jestem „fanatykiem" filozofii w każdym wydaniu, przytoczę krótki fragment *Myślenia według wartości* ks. Józefa Tischnera (na pewno kojarzycie jego kultową już dzisiaj *Filozofię po góralsku*[5], jeśli nie — polecam!):

> *Człowiek w kryjówce chroni się przed światem i przed innymi. Przyszłość nie obiecuje człowiekowi nic wielkiego, pamięć przeszłości podsuwa mu przed oczy same doznane porażki, przestrzeń nie zaprasza do żadnego ruchu. Wprawdzie w kryjówce nadzieja nie znika bez reszty, tylko maleje, ale maleje do tego stopnia, że staje się jedynie nadzieją przetrwania*[6].

Styl zarządzania, który próbuje odzwierciedlać świat produkcji, „spycha" członków zespołów projektowych właśnie do „kryjówek". Zauważcie, w ilu projektach, w których sami uczestniczyliście, mieliście do czynienia z sytuacją, gdy kierownik zawsze brał sprawy w swoje ręce w obawie o Wasze kompetencje. Współpracowałem kiedyś z firmą, w której spotkałem się z dość komiczną sytuacją, a przypominała mi ona dawne działania Głównego Urzędu Kontroli Prasy, Publikacji i Widowisk cenzurującego publikacje prasowe, radiowe i telewizyjne w PRL-u. Każdy e-mail, który ScrumMaster lub lider techniczny wysyłał do klienta, musiał być wcześniej wnikliwie przeanalizowany i autoryzowany przez kierownika projektu. Z aptekarską wręcz „precyzją"

[5] J. Tischner, *Historia filozofii po góralsku*, Znak, Kraków 1997.

[6] J. Tischner, *Myślenie według wartości*, Znak, Kraków 2000, s. 412 – 413.

kierownik studiował każdą wiadomość, która wychodziła poza firmę, nanosił kluczowe — jak twierdził — poprawki. Kiedyś zapytałem go, dlaczego to robi. Szybko dostałem odpowiedź, że jego ludzie nie mają wyczucia tzw. „kwestii politycznych", więc nie będzie z tego względu ryzykował kontraktu z klientem, który jest jego jedyną „dojną krową". Niestety nie byłem w stanie go przekonać, że w ten sposób zabija inicjatywę w zespole i — w efekcie — kształtuje mentalność „ludzi z kryjówek", którzy wcześniej czy później przejdą do defensywy i przestaną wykazywać jakąkolwiek wolę twórczego działania. Bo, koniec końców, po co podejmować takie działanie, skoro zawsze istnieje ryzyko, że może się to źle skończyć dla projektu?

Innym poważnym błędem menedżerów wychowanych w świecie produkcji jest zabijanie indywidualności. Przez „indywidualistę" rozumiem osobę mającą swoje zdanie, kreatywną, patrzącą na problemy, które wcześniej czy później pojawią się w projekcie, zawsze w sposób nowatorski i inny od wszystkich proponowanych. Nie chodzi mi jednak o „artystę", który pojawia się w pracy, kiedy chce, a kiedy już przyjdzie, to wszystko, co robi, traktuje jako środek do samopodniety. Obecność takich ludzi w zespole projektowym jest bardzo destruktywna i trzeba dobrze się zastanowić, zanim zatrudnimy takich „gagatków" do pracy w naszej firmie. Dla kierownika, który swoją inspirację czerpie z linii produkcyjnej, programista-indywidualista jest tylko źródłem problemów. Tymczasem to często *wyjątkowość* decyduje o sukcesie danego przedsięwzięcia. Ten, kto oglądał film *Lśnienie* Stanleya Kubricka, z rewelacyjną rolą Jacka Nicholsona, z pewnością wie, o czym mówię[7].

Ostatnią rzeczą, o której chciałbym wspomnieć w kontekście tradycyjnego stylu zarządzania, jest koncentracja na realizacji zadań. W zarządzaniu wzorującym się na produkcji samochodów nie ma czasu na myślenie o tym, jak coś zrobić. Zadania muszą być wykonywane mechanicznie, żeby ze wszystkim zdążyć na czas. Tymczasem w projektach software'owych tak się nie da. Oczywiście wszyscy o tym wiemy, niemniej często jest tak, że kiedy już dostaniemy projekt do ręki, rzucamy się w wir pracy, najczęściej nie mając odpowiedniej ilości czasu na planowanie, analizę nowych metod i technologii, na szkolenia, czytanie fachowych książek, na wspólne dyskusje o problemach. W ubiegłym roku miałem przyjemność współpracować z firmą, która „programowała na odległość" dla niemieckiego zleceniodawcy — dużej korporacji z, wydawałoby się, uporządkowaną strukturą i tzw. dojrzałym ładem organizacyjnym. Jednym z zadań, do których zostałem zatrudniony,

[7] *http://www.imdb.com/title/tt0081505/* (dostęp: 3 maja 2012).

było usprawnienie procesów szacowania parametrów projektu. Chodziło o to, żeby nauczyć ludzi przygotowywać WBS-a (ang. *Work Breakdown Structure*), odpowiednio definiować zadania projektowe, a także wdrożyć jedną z technik estymacyjnych. Początkowo wszystko szło jak po maśle. Zespół z Polski bardzo szybko się wdrożył. Wspólnie przygotowaliśmy kilka arkuszy estymacyjnych do wcześniej złożonych zamówień. I tu zaczęły się schody. Nasi niemieccy koledzy nie mogli zrozumieć, dlaczego w wycenie uwzględniliśmy analizę rozwiązań architektonicznych, czas spędzony na telekonferencjach, a także zrobienie prototypu sprawdzającego jedno z możliwych rozwiązań. Ta firma w ogóle nie brała pod uwagę, że zanim coś zrobimy, musimy najpierw się zastanowić, jak to zrobić. Spotkać się, pogadać, pomyśleć nad rozwiązaniem. Nie da się ludzi podpiąć do klawiatury i kazać im pisać kod.

Wychodzenie z kryjówki

Agile to rodzina metod, które pomagają ludziom zarządzanym według reguł linii produkcyjnej wyjść z kryjówek.

> *Kryjówka ma jakiś próg, który trzeba przekroczyć. Próg znaczy koniec kryjówki i początek nowej przestrzeni. Jeśli się widzi koniec, a nie widzi się początku, będzie się mimo wszystko więcej kochać złudzenie niż prawdę*[8].

Jaki początek proponują metody agile? Przede wszystkim dzięki iteracyjnej metodzie pracy, w której projekt podzielony jest na kilka mniejszych kawałków (iteracji), akceptują „prawo" członków zespołów do robienia błędów. Zastanówcie się przez chwilę, na ile ślepych zaułków w trakcie realizacji Waszych projektów natrafiliście (bez względu na ich rozmiar i złożoność). Ile było meandrów? Ile razy waliliście głową w mur, nie wiedząc, co robić dalej — w którą stronę iść? Agile zakłada, że projekty są podatne na błędy. Jest rzeczą zupełnie normalną, że często zmienia się kierunek prac rozwojowych. Klient co kilka tygodni dostaje fragmenty działającego produktu, może go sobie oglądnąć, nacieszyć się nim — zgłosić swoje zastrzeżenia oraz zaproponować nowe pomysły. To jest duża siła tego podejścia. Pamiętam, że jako początkujący kierownik projektu, przygotowując harmonogram prac w oparciu o metodykę kaskadową, zawsze miałem problem ze zmiennością wymagań. Zbierałem nawet metrykę śledzącą ich fluktuację (ile pojawiło się nowych?

[8] J. Tischner, *Myślenie według wartości*, op. cit., s. 427.

ile zmieniono starych? ile te zmiany nas kosztowały?), a wszystko po to, żeby mieć „haka" na klienta na wypadek, gdyby przyszło mu do głowy czepiać się nas, bo po raz kolejny nie zdążyliśmy z osiągnięciem zaplanowanego kamienia milowego, bo w trakcie implementacji kilka razy zmieniły się wymagania i pierwotny zakres prac poszerzył się o trzy nowe funkcjonalności. Muszę przyznać, że zawsze byłem bezsilny. Kiedy jednak zacząłem stosować zwinne praktyki projektowe, wszystko się zmieniło. Agile „oswoił" zmianę. Pokazał, że „nie taki diabeł straszny...".

Metody zwinne promują opisany wcześniej indywidualizm. W tym sensie są drogą pod prąd tradycyjnego stylu zarządzania. Dzięki stosowanym metodom i narzędziom tworzą coś, co moglibyśmy nazwać demokratycznym stylem zarządzania. Kierownik projektu nie jest już „królem", który panuje nad ludem, niezależnie od układów politycznych i systemów. Jest bardziej sługą i mentorem osób, które angażują się w projekt. Jego rola musi więc zostać odpowiednio przedefiniowana. Dodatkowo o „demokracji" świadczy fakt, że „władza została oddana w ręce ludu". Odtąd to zespół, a nie kierownik projektu, decyduje o tym, jak zdefiniować poszczególne zadania projektowe oraz kto się nimi zajmie. Rola kierownika musi więc zostać zdefiniowana na nowo, w kontekście wartości i zasad, które przynosi ze sobą idea zwinności. Będzie o tym więcej w rozdziale 3., poświęconym rolom w Scrumie.

Myślenie odwrotne

Metody agile powstały w wyniku próby spojrzenia na proces tworzenia oprogramowania w nieco inny — *odwrotny* sposób. Na myśl przychodzi mi tutaj historia, którą przeczytałem w książce Paula Ardena *Cokolwiek myślisz, pomyśl odwrotnie*[9]. Rzecz zdarzyła się przed olimpiadą w Meksyku, która odbyła się w 1986 r. Był to czas, kiedy technika skoków wzwyż polegała na tym, że sportowcy w trakcie skoku „przerzucali" swoje ciało równolegle do poprzeczki. Na wspomnianej olimpiadzie pojawił się, wtedy jeszcze mało znany zawodnik, Dick Fosbury, który podbiegł do poprzeczki, ustawionej na rekordowej wysokości 2 m 24 cm. Wybił się i zamiast skoczyć jak wszyscy, ustawił swoje ciało w kierunku poprzeczki, obracając się do niej plecami. Unosząc nogi, pokonał ją tyłem. Jego styl skakania obowiązuje do dzisiaj

[9] P. Arden, *Cokolwiek myślisz, pomyśl odwrotnie*, tłum. O. Siara, Insignis, Kraków 2008, s. 7.

i zasłynął jako „flop Fosbury'ego". Dick skoczył wyżej niż ktokolwiek przedtem, bo odważył się *myśleć inaczej*. Metody agile to właśnie przykład „myślenia odwrotnego" względem pokonywania poprzeczki w tradycyjny sposób, czyli w naszym przypadku — stosowania podejścia kaskadowego. Kiedy po raz pierwszy oglądałem w internecie zdjęcia skoczków, którzy oddawali swoje skoki przed 1986 r., pomyślałem: „Jak oni mogli tak nienaturalnie skakać? Przecież to zupełnie cudaczne. Aż wierzyć się nie chce, że ludzie nie skręcali sobie karków przy lądowaniu". A jednak.

Zdrowie szaleńców

Ludziom, którzy *myśleli odwrotnie*, była w całości poświęcona jedna z najbardziej innowacyjnych kampanii reklamowych wszechczasów — *Think Different* firmy Apple. Steve Jobs w jednym z wywiadów opowiadał, jak doszło do jej powstania:

> *Kampania Think Different wynikała stąd, że ludzie, w tym nasi pracownicy, zapomnieli, do jakich wartości odwołuje się Apple. Długo zastanawialiśmy się, jak powiedzieć komuś, za czym się opowiadamy, jakie wartości wyznajemy, aż uświadomiliśmy sobie, że kiedy nie znasz kogoś dobrze, możesz go zapytać: „Kim są twoi bohaterowie?". Można się wiele dowiedzieć o ludziach na tej podstawie. Stwierdziliśmy więc: „Dobra, powiemy im, kim są nasi bohaterowie"*[10].

O kampanii *Think Different* mówi się często, że przyczyniła się do odbudowania wizerunku firmy Apple po fiasku sprzed poprzednich lat. Nie jestem specem od marketingu, ale dla mnie była to najbardziej innowacyjna seria reklam, jakie kiedykolwiek widziałem. Na ekranie pojawiały się, sprytnie zmontowane, czarno-białe migawki bohaterów, wynalazców, myślicieli i buntowników. Mogliśmy tam zobaczyć Alberta Einsteina, który pali fajkę, Boba Dylana grającego na harmonijce, Martina Luthera Kinga wygłaszającego swoje najsłynniejsze kazanie: *I Have a Dream* (*Mam marzenie*), Richarda Bransona potrząsającego butelką szampana, tańczącą Marthę Graham, pełną pokoju twarz Mahatmy Ghandiego czy malującego Picassa. Tym wszystkim obrazom towarzyszył znakomity tekst czytany przez aktora Richarda Dreyfussa:

[10] S. Levy, *The Perfect Thing. How the iPod Shuffles Commerce, Culture and Coolness*, Simon & Schuster, New York 2006, s. 118 [tłum. fragmentu: M. Chrapko].

Zdrowie szaleńców. Odmieńców. Buntowników. Wichrzycieli. Okrągłych kołków w kwadratowych dziurach. Tych, którzy patrzą na wszystko inaczej. Nie lubią reguł. Nie mają szacunku dla status quo. Możesz ich cytować, nie zgadzać się z nimi, gloryfikować ich albo szkalować. Tylko zignorować ich nie możesz. Bo oni zmieniają świat. Popychają ludzkość do przodu. I chociaż niektórzy widzą w nich szaleńców, my widzimy ich geniusz. Bo ludzie na tyle szaleni, aby myśleć, że mogą zmienić świat, faktycznie go zmieniają[11].

Powstanie zwinnych metod tworzenia oprogramowania od samego początku było pewnego rodzaju *myśleniem odwrotnym* do tego wszystkiego, co działo się w inżynierii oprogramowania od jej powstania. Idee zwinności kiełkowały w głowach takich „odmieńców", jak: Gerald M. Weinberg, Frederick P. Brooks, Tom De Marco, Timothy Lister. Oni już w latach 70. podkreślali znaczenie „myślenia odwrotnego" w tworzeniu oprogramowania i odejścia od mentalności zaczerpniętej ze świata produkcji, która to mentalność miałaby sens, gdybyśmy pracowali w barach szybkiej obsługi (lub innym środowisku produkcyjnym), ale na pewno nie przy projektach software'owych, gdzie ludzie bardziej pracują głową niż rękami.

Świat metod zwinnych, który powstawał niemalże równolegle do świata kaskadowego, był światem ludzi patrzących na wszystko inaczej. Światem, który powywracał do góry nogami wszystkie dotychczasowe reguły prowadzenia projektów. W lipcu 2007 r. portal CNN Money przeprowadził ankietę, na podstawie której wyłonił listę pięćdziesięciu najbardziej wpływowych ludzi, idei, trendów, produktów — tego, co zmieniło bieg świata biznesu[12]. Metody agile znalazły się na 18. miejscu.

Manifest agile

Równolegle z kształtowaniem się nowej fali „myślenia odwrotnego" jako opozycji do tego wszystkiego, co wiązało się z tradycyjnym rozwojem oprogramowania, rodziły się konkretne praktyki stosowane przez ambasadorów zmiany. Lata 80. i 90. to czas stosowania alternatyw, w pewnym sensie: uczenia się na błędach wynikających z próby zaszczepienia idei linii produkcyjnej

[11] YouTube, *Apple — Crazy Ones*, *http://www.youtube.com/watch?v=4oAB83Z1ydE* (dostęp: 3 maja 2012) [tłum. fragmentu: M. Chrapko].

[12] *The 50 Who Matter Now*, CNN Money, *http://money.cnn.com/galleries/2007/biz2/0706/gallery.50whomatter.biz2/33.html* (dostęp: 3 maja 2012).

do świata tworzenia oprogramowania. Jest to okres, w którym różne grupy praktyków, na swój własny i niczym nieskrępowany sposób, zaczynają tworzyć nową rzeczywistość — ta rzeczywistość później zostanie nazwana słowem „agile". Lata 80. i 90. to również czas, kiedy próbuje się nazywać i systematyzować metody agile. To wtedy, w 1986 r., zaczyna być głośno o metodzie Scrum; zostaje opracowana Dynamic Systems Development Methodology (1994 r.); Kent Beck nazywa praktyki Extreme Progamming (1996 r., choć prawdziwy boom tej metody to 2000 r.); Alistar Cockburn mówi o metodach Crystal, a Jeff DeLuca publikuje *Feature-Driven Development* (1998 r.). Nowa fala praktyk nabiera rozmachu.

W 2001 r. w ośrodku wypoczynkowym Snowbird, w USA (stan Utah), zbiera się grupa zwolenników nowego podejścia celem nazwania tego, co tak naprawdę charakteryzuje powstające metody. W efekcie zostaje opracowany tzw. *Manifesto for Agile Software Development* (z ang. *Manifest zwinnego tworzenia oprogramowania*), który stanowi deklarację podstawowych wartości i zasad agile (w całości do przeczytania na stronie: *http://agilemanifesto.org/*). Pierwsza część manifestu to cztery krótkie stwierdzenia, które w sposób prosty i jasny oddają filozofię zwinności. Mówią o tym, jakie wartości zwolennicy zwinnych metod cenią najbardziej. Zostały one przedstawione na rysunku 1.3.

Rysunek 1.3. Manifest agile

Ludzie i interakcje ponad procesami i narzędziami

Można mieć świetnie zdefiniowane procesy, kupić bardzo drogie narzędzia, ale i tak, koniec końców, największy wpływ na powodzenie naszych projektów mają ludzie zaangażowani w ich rozwój. Czynnik ludzki jest elementem,

którego bardzo brakuje we wszystkich modelach i standardach zaawansowanych procesów wytwórczych, np. w modelu CMMI®. Oczywiście można odpierać ten atak, mówiąc, że model ten ma trochę inne zastosowanie — jego zadaniem jest nakreślić mapę drogową, która pomoże zorganizować świat procesów w firmie. Niemniej prawda jest taka, że w praktyce model CMMI® nie zawiera żadnych konkretnych mechanizmów, które by uwydatniały wpływ tego czynnika — czy to na poziomie życia organizacji, czy w porządkowaniu różnego rodzaju działań projektowych. Oczywiście umożliwia wprowadzenie określonych zwinnych praktyk, które promują pracę zespołową, wzmacniają komunikację interpersonalną, jednak w żaden sposób nie zapewnia, że te praktyki zostaną wprowadzone.

Działające produkty ponad złożoną dokumentację

Czytając to stwierdzenie manifestu, przypominam sobie rozmowy prowadzone przy okazji różnego rodzaju wdrożeń — rozmowy z programistami, którzy narzekali na prowadzenie i utrzymywanie dokumentacji w ich projektach. Często mieli wrażenie, że poruszają się między jedną a drugą ryzą papieru; że w efekcie produkują stosy dokumentów, które są generowane, bo taka jest polityka firmy i tego wymaga od nich kierownictwo. A tymczasem klienta i tak interesuje działający software, który jest wolny od defektów i dostarczony na czas. Dlatego dokumentacja powinna raczej wspierać rozwój produktów, niż go utrudniać.

Współpraca z klientem ponad negocjacją kontraktu

Przy tym haśle, ale i przy wszystkich pozostałych wartościach i zasadach manifestu, należy pamiętać, że jest jeszcze realny świat naszych projektów. I wszystko to, co tworzy ich niepowtarzalną rzeczywistość. Dlatego nawet jeżeli Wasi klienci „kupią" idee filozofii zwinności, to mimo to zawsze będą się chcieli jakoś zabezpieczyć (a na pewno będzie tego chciał ich dział finansowy czy prawny) na wypadek, gdyby coś poszło nie tak. Ciągła współpraca z klientem na każdym etapie prac rozwojowych jest jedną z podstawowych charakterystyk projektów zwinnych, która stanowi odpowiedź na metody kaskadowe, gdzie podpisanie kontraktu z klientem stanowiło o tym, czy dany projekt wystartuje, czy nie. Filozofia agile to obecność klienta na każdym etapie prac rozwojowych. Praca programistów zależy bardzo mocno od informacji zwrotnej, którą dostarcza klient.

Reagowanie na zmiany ponad trzymaniem się planu

Osoby, które kiedykolwiek miały do czynienia z tradycyjnymi metodami tworzenia oprogramowania, bardzo dobrze pamiętają te chwile, kiedy w trakcie implementacji pojawiały się nowe wymagania lub klient nagle coś zmieniał. Zmiana w trakcie kolejnych faz rozwojowych była wtedy najmniej pożądaną rzeczą. Mogliśmy przeżyć brak kawy w firmie, ale nie kolejne „wrzutki" od klienta. Zmiana była czymś obcym, niechcianym. Baliśmy się jej jak ognia. Dużą zasługą metod agile jest „oswojenie" zmiennych życzeń klienta w trakcie prac rozwojowych. Zmiana jest dobra, jest niezmiennym elementem naszych prac wytwórczych. Oczywiście to nie oznacza, że nasze projekty nie powinny mieć planu i że planowanie jest niepotrzebne. Przeciwnie! Plan musi być. Jednak każdorazowo jest on dopasowywany do zmieniających się warunków środowiskowych.

Podstawowe wartości manifestu zostały dodatkowo wzbogacone o 12 zasad, które można potraktować, jako swoistą listę kontrolną zwinnego projektu. Można się z nimi zapoznać na wspominanej już stronie: *http://agilemanifesto.org/*.