

POZNAJ
SCRUM
I BĄDŹ
EFEKTYWNY!



MARIUSZ CHRAPKO

SCRUM

O zwinnym **ZARZĄDZANIU PROJEKTAMI**

one
press

Melion



Autorstwo: Mariusz Chrapko (rozdziały 1 – 12), Mike Cohn (Wprowadzenie)

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Ewelina Burska

Ilustracje: Aleksandra Bułhak

Projekt okładki: Magdalena Stasik

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie?scrumo>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-3828-4

Copyright © Helion 2013

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

SPIS TREŚCI

O AUTORZE	7
PODZIĘKOWANIA	9
WPROWADZENIE	13
ZAMIAST WSTĘPU	15
1. MYŚLENIE ODWROTNE. Czym jest agile?	17
Szkola przetrwania	17
Wodospad	18
Pan Samochodzik i tworzenie oprogramowania	23
Ludzie z kryjówek	24
Wychodzenie z kryjówki	26
Myślenie odwrotne	27
Zdrowie szaleńców	28
Manifest agile	29
2. LEKCJE PŁYWANIA. O zwinnym zarządzaniu projektami	33
Klasyka i jazz	33
Piankowe wyzwanie	40
Dama z łasiczką	42
Agile i pornografia	44
Scrum	48
Lekcje pływania	50

3. KONTRABASISTA. Nowe role i obowiązki	53
Lekcja z <i>Kontrabasisty</i>	53
ScrumMaster	54
Cechy dobrego ScrumMastera	58
Wybór ScrumMastera	66
W dużej organizacji	68
Właściciel Produktu	69
Czy kierownik projektu jest potrzebny w Scrumie?	82
4. ŁAWA PRZYSIĘGLYCH. O hodowaniu zwinnych zespołów projektowych	97
Ugotowani	97
Zespół = nirwana projektu	99
Szlachetny cel	103
Uprawa zwinnych zespołów	105
Wielofunkcyjne zespoły	120
Zespoły komponentowe	121
Ława przysięgłych	124
5. PLATON, IDEE I RYBY. Jak zwinnie zarządzać wymaganiami?	131
Jaszczurka czy dinozaur?	131
Diabeł tkwi w... komunikacji	133
Historijki użytkownika	140
ZaINWESTuj w dobre historijki	147
Rejestr produktu i ocean plazmy	158
Historijki, epiki, tematy... thriller	160
Wymagania jak góra lodowa	161
Pielęgnacja	162
Praca z dużym rejestrem	163
Tylko 150, reszta nie ma znaczenia	165
6. LIZANIE ZNACZKÓW I CHIRURGIA MÓZGU. Szacowanie projektów w Scrumie	167
Kadzidłowy dym, ekstatyczny trans	167
O istocie szacowania	168
W punktach czy w osobodniach?	190
Trochę techniki i człowiek... się nie gubi	195
7. O ŻEGLOWANIU I OBIERANIU CEBULI. Planowanie projektów w Scrumie	203
Obsesja planowania	203
Zwinne planowanie	204
Żeglowanie i obieranie cebuli	206
Jak się do tego zabrać?	208

Plan wydania	227
Planowanie na dużą skalę	229
Jak śledzić postęp prac?	233
8. BIEGNIJ, FORREST, BIEGNIJ. Sprint	239
Jak na nartach po Barcelonie	239
Planowanie sprintu	240
Plan sprintu	253
Gotowi!... Do startu!... Gotowi?!	254
W dużych projektach	256
Zrobione czy niezrobione?	260
Puste taczki	262
Komunikacja	265
Śledzenie postępu prac	267
9. W POKOJU SZTABOWYM. Codzienne scrumy	273
Wniosek o zakończenie wojny	273
Codzienny scrum	278
Jak się komunikować?	290
Zespoły rozproszone	293
Scrum of Scrums	297
10. FURTKA DO OGRODU. O tym, jak zorganizować przegląd sprintu	301
W sklepie z zabawkami	301
Przegląd sprintu	302
Metoda Kawasakiego	304
Zarażać dobrą energią	305
Furtka do ogrodu	306
Atak na Ziemię	307
11. MYŚŁODSIEWNIA. Retrospektywa na koniec sprintu	309
Myśłodsiewnia	309
Oczyszczenie	310
Najczęściej pomijana praktyka	311
Lekcja anatomii	312
Próba ogarnięcia kuwety	321
12. WSPÓLNY REJS. Historia z morza wzięta	331
SKOROWIDZ	335

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 parolotni, 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 huśtawki¹.

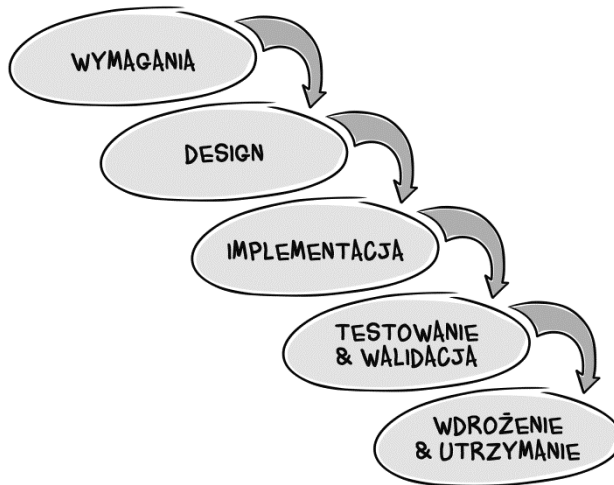
Tak sobie myślę, że gdybym ja, podobnie jak Bear Grylls, został rzucony na pastwę losu i musiał przetrwać w najdzikszych miejscach na świecie, na pewno nie schodziłbym w dół wodospadem. Znając moje fizyczne możliwości, dodatkowo biorąc pod uwagę fakt, że nie służyłem w SAS 21, z pewnością poszukałbym 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 linowo (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 ciężenia, a czasoprzestrzeń zakrzywia się w nieprawdopodobny wręcz sposób. Scena żywcem wyjęta z *Incepcji* Christophera Nolana². Chciałoby się włamać przez sen do świadomości klienta i zaszcześcić 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).

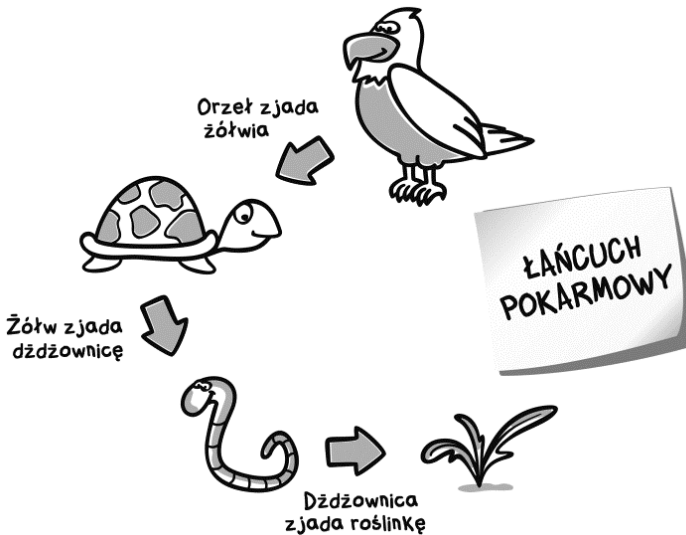
przeplanować i starać się jakoś podnieść morale sfrustrowanego zespołu. Nie trzeba już chyba dodawać, jak bardzo takie „wrzutki” zwiększają koszty prowadzonego projektu.

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, zapalki, 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 wchodzi 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 poprzepłatanane — 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 wyzwalala 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 przrzucany 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 okna³.

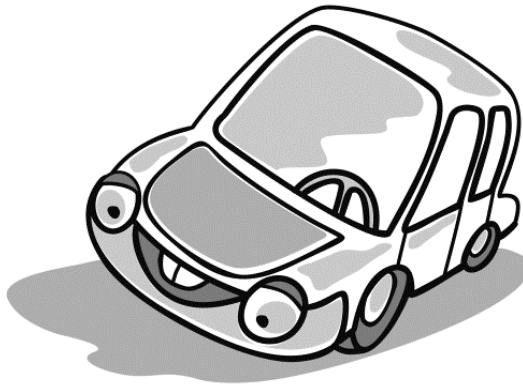
Winston Royce, który jako pierwszy w artykule *Managing the Development of Large Software System* (1970) opisał model kaskadowy, powiedział bardzo ciekawą rzecz: „Podoba mi się ten pomysł, ale jego implementacja jest ryzykowna i ma tendencję do błędów”⁴. Jest swoistym paradoksem, że wiele osób uważa 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 *Myslenia 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, *Myslenie 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 wycucia 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 „gagałków” do pracy w naszej firmie. Dla kierownika, który swoją inspirację czerpie z linii produkcyjnej, programista-indywidualista jest tylko źródłem problemów. Tymczasem to często *wyjatkowość* decyduje o sukcesie danego przedsięwzięcia. Ten, kto oglądał film *Lśnienie* Stanleya Kubricka, z rewelacyjną rolą Jacka Nicholsona, z pewnością wie, o czym mówię⁷.

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 prawdę⁸.

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 natrafiłicie (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 oglądać, 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 rolam 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 równoległe 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 najśłynniejsze 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 zmieniają¹¹.

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 równoległe 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 biznesu¹². Metody agile znalazły się na 18. miejscu.

Manifest agile

Równoległe 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 Programming (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. *Manifesto 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. Czynnikiem 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 hasle, 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/>.

SKOROWIDZ

A

Albrecht Allan, 187
analityk biznesowy, 74, 80, 120, 134, 135, 138
analityk systemowy, 120
Atlassian, 263
atrybut mówcy, 281

B

Beck Kent, 30, 141
Bezos Jeff, 115
blokada, 49
błędy, 248, 249
Boehm Barry, 174
Brass Dick, 66
Brown Tim, 113
burndown chart, *Patrz:* wykres spalania

C

Cannon-Brokkes Mike, 263, 264
Chambers Harry, 86
Chief Architect, *Patrz:* Główny Architekt
Chief Engineer, *Patrz:* Główny Inżynier
Chief ScrumMaster, *Patrz:* Główny ScrumMaster
ciąg
 Fibonacci, 198
 geometryczny, 198

Class Owner, *Patrz:* Właściciel Klasy
Cockburn Alistar, 30
Codziennik, 294
Cohn Mike, 147, 157, 180, 205, 260, 285
cross-functional teams, *Patrz:* zespół wielofunkcyjny
Cskiszentmihalyi Mihaly, 130
cykl
 wytwórczy, 47
 życia projektu, *Patrz:* projekt cykl życia
czas trwania, 176, 177, 183

Ć

ćwiczenie Piankowe wyzwanie, 40

D

daily scrum, *Patrz:* scrum codzienny
Dama z łasiczką, 43
definicja ukończenia, 261
definition of done, *Patrz:* definicja ukończenia
DeLuca Jeff, 30
DeMarco Tom, 105
Derby Esther, 312
dni idealne, 178, 191, 193, 195, 218, 240
dokumentacja, 31
Drucker Peter, 86
DSDM, 30, 47

Dunbar Robin, 165
duration, *Patrz:* czas trwania
Dynamic Systems Development
Methodology, *Patrz:* DSDM

E

effort, *Patrz:* pracochłonność
elicitation, 138
Entergy, 103
entroskop, 37
epik, 79, 160, 161, 166, 185, 213, 214, 232
estymacja, 26, 169, 171, 189, 190, 193,
196, 197, 201, 222
Extreme Programming, 30, 45, 46, 141,
155, 160

F

Farquhar Scott, 264
FDD, 30, 46
Feature Team, *Patrz:* Zespół
Funkcjonalny
Feature-Driven Development, *Patrz:*
FDD
feng shui, 106
Feynman Richard, 96
Fforde Jasper, 37
Fisher Darin, 70
Fosbury Dick, 27
Fowler Martin, 141
frequent delivery, 48

G

Główny Architekt, 46
Główny Inżynier, 78
Główny ScrumMaster, 69
godziny idealne, 245, 254
Goodger Ben, 70
Google Chrome, 70
Gran Torino, 91
grooming, 162, 255, 285

H

hand-over, 139
Heller Michał, 77
hermeneutyka, 132

historyjka badawcza, 155
historyjka użytkownika, 81, 141, 147,
149, 150, 152, 153, 154, 155, 156, 158,
160, 161, 166, 179, 188, 198, 200, 201,
209, 229, 232, 240, 243, 253, 255, 256, 268
INVEST, 148
karta, 144
konfirmacja, 146
konwersacja, 145
Hohmann Luk, 317

I

ideal days, *Patrz:* dni idealne
idealne dni, *Patrz:* dni idealne
idealne godziny, *Patrz:* godziny idealne
impediment, *Patrz:* blokada
impediment backlog, *Patrz:* przeszkody
rejestr
implementacja, 19, 20, 34, 120, 261
inkrement, 44
Innovation Games, 317
inspekcja kodu, 56, 261
integrowanie, 19
interesariusz, 36, 38, 74, 169, 265, 307
INVEST, 148
inżynier wymagań, 134, 135, 138, 139
inżynieria oprogramowania, 18
iteracja, 44

J

Jacobellis Nick, 44
Jaspers Karl, 45, 90
Jeffries Ron, 144
JIRA, 71, 270
Jobs Steve, 28, 72, 301, 305

K

kampania Think Different, 28
Kanban, 47
Kawasaki Guy, 304
Kennedy John Fitzgerald, 71
kierownik projektu, 69, 80, 81, 82, 84, 86,
87, 95, 112, 114, 169, 268, 278
kierownik projeku, 196
klika, 102, 105
kod flagowy MKS, 260

komunikacja, 265, 290
 osmotyczna, 48
 komunikator internetowy, 293
 konwersacja, 141, 145, 147, 154, 158, 163
 kryterium akceptacyjne, 147, 200, 256, 304
 Kubrick Stanley, 25

L

Larsen Diana, 312
 lean, 47
 lean management, 47
 Lean Manufacturing, 47
 lejek niepewności, 174, 220, 221
 Lencioni Patrick, 277
 Leonard da Vinci, 43
 linie kodu, 178
 lista kontrolna, 146
 Lister Timothy, 105
 Low Level Design, *Patrz:* projekt
 niskopoziomowy
 Lowndes Leil, 74
 ludzie na kształt litery T, 113, 119, 128

Ł

łańcuch projektowy, 21
 łącznik, 295

M

maintenance, 19
 Malle Louis, 44
 Manifest zwinnego tworzenia
 oprogramowania, 30, 45
 Manifesto for Agile Software
 Development, 30
 mapa drogowa, 31
 McConnell Steve, 174
 McLuhann Marshall, 293
 menedżer produktu, 69, 74
 metoda
 Crystal, 30, 48
 iteracyjna, 26
 kaskadowa, *Patrz:* model
 kaskadowy
 Scrum, 30, 46, 48
 metodyka kaskadowa, 17
 mikrozarządzanie, 85

model
 CMMI, 31
 kaskadowe, 175
 kaskadowy, 19, 20, 21, 22, 26, 31, 33,
 46, 120, 133, 137, 190, 215
 sekwencyjny, *Patrz:* model
 kaskadowy
 Model Kano, 214
 MoSCoW, 47

N

Na białce, *Patrz:* Triangulacja
 Nonaka Ikujiro, 128, 129

O

osmotic communication, *Patrz:*
 komunikacja osmotyczna
 osmotyczna komunikacja, *Patrz:*
 komunikacja osmotyczna
 osobodzień, *Patrz:* dni idealne

P

Page Scott, 127, 128
 Penderecki Krzysztof, 39
 Pichler Roman, 255, 307
 Planning Poker, 189, 197, 198, 209
 planowanie, 32, *Patrz też:* projekt plan,
 sprint planowanie, wydanie
 planowanie
 do przodu, 257
 Polanyi Michael, 108
 połączenie online, 57
 Poppendieck Mary, 47
 Poppendieck Tom, 47
 Popper Karl, 173
 pracochłonność, 116, 176, 183, 187, 191,
 193, 214
 priorytet, 212, 213, 217, 241, 243, 253, 267
 priorytetyzacja wymagań, 47
 proces, 55
 empiryczny, 36, 39, 40, 42, 44
 powtarzalny, 34, 36, 42
 product backlog, *Patrz:* rejestr produktu
 Product Backlog, *Patrz:* rejestr produktu
 Product Owner, *Patrz:* Właściciel
 Produktu

programista, 135, 137, 138, 139, 140, 145,
147, 151, 158, 214

projekt

cykl życia, 18
interesariusz, *Patrz:* interesariusz
niskopoziomowy, 19
plan, 34, 204, 205
ryzyko
ryzyko, 216
systemu, *Patrz:* system projekt
złożoność, 36, 37
zorientowany na datę, 225
zorientowany na funkcjonalności, 227

projektowanie

liniowe, 18
sekwencyjne, *Patrz:* projektowanie
liniowe

próżniactwo społeczne, 117

przekazanie, *Patrz:* hand-over

przeszkody, 289

przeszkoga, *Patrz:* blokada

punkty, 178, 179, 182, 183, 188, 190, 192,
193, 214, 218, 240, 251

funkcyjne, 178

Putnam Doug, 116

Q

QSM, 115

Quantitative Software Management,
Patrz: QSM

R

rejestr produktu, 47, 49, 70, 72, 120, 158,
162, 166, 208, 209, 232

relatywne ważenie, 214

Release Kick-Off, 79

retrospektywa, 47, 309, 311, 312, 316,
317, 319, 321

Ringelmann Max, 118

Robertson Suzanne i James, 138

rola projektowa, 46, 49, 53

Royce Winston, 22

Rubinstein Artur, 34

ryzyko, 216, 230

projektowe, 49

S

samoorganizacja, 86, 87, 88, 95, 130,
268, 290

samotranscendencja, 129

Schwaber Ken, 54, 253

Schwarz Roger, 322

scrum

analiza, 319

codzienny, 278, 279, 280, 281, 282,
285, 290, 291, 294, 295, 296, 297

Scrum of Scrums, 297, 298

ScrumMaster, 49, 53, 54, 55, 57, 58, 60, 61,
63, 65, 66, 76, 78, 198, 199, 202, 214, 271,
279, 282, 287, 289, 295, 311, 323, 327

sesja pielęgnacyjna, *Patrz:* grooming

Skillman Peter, 40

Sorensen Ted, 71

specjalista, 114, 119

specyfikacja

techniczna, 18

wymagań, 158

spike, 155, 186

spotkanie, 312

codzienne, *Patrz:* scrum codzienny
projektowe, 278

sprint, 48, 56, 77, 140, 146, 154, 156, 163,
209, 210, 218, 241, 259, 309

cel, 242, 254, 303

planowanie, 207, 240, 244, 246, 250,
253, 256

przegląd, 302, 306

synchronizacja, 258

tablica, 267, 279, 285

stakeholder, *Patrz:* interesariusz

Stańko Tomasz, 39

Stewart Potter, 44

story points, *Patrz:* punkty

strefy czasowe, 295

Streisand Barbra, 35

Süskind Patrik, 53

system

architektura, 20

implementacja, *Patrz:*

implementacja

kolejkowania pracy, 47

projekt, 19, 21

szacowanie, 169, 171, 173, 174, 176, 177,
178, 179, 182, 187, 188, 190, 192, 197,
201, 208, 209, 219, 245

Ś

środowisko pracy, 106

T

tablica sprintu, *Patrz:* sprint tablica
tacit knowledge, *Patrz:* wiedza ukryta
Takeuchi Hirotaka, 128, 129
telekonferencja, 291
temat, 79, 161, 166, 212, 214, 232
test
 akceptacyjny, 146, 157, 256, 261
 jednostkowy, 261
 regresyjny, 212, 261
tester, 135, 137, 139, 140, 145, 214, 269
testowanie, 19, 20, 120, 122, 157, 212, 261
Think Different, 28
Tischner Józef, 24, 105, 213
Toyota Production System, 47
Triangulacja, 197, 201, 209
Truman Harry, 64
T-shaped People, *Patrz:* ludzie
 na kształt litery T

U

ujawnianie, *Patrz:* elicitation
user story, *Patrz:* historyjka
 użytkownika
utrzymywanie, *Patrz:* maintenance

W

Wake Bill, 154
Wake William, 147
warunki środowiskowe, 32
waterfall model, *Patrz:* metodyka
 kaskadowa
WBS, 26
Welch Jack, 136
wideokonferencja, 57, 202, 292
wiedza ukryta, 108
wielozadaniowość, 110
Witkacy, 42

Właściciel

Klasy, 46
Produktu, 49, 53, 60, 61, 69, 71, 72, 73,
74, 75, 77, 78, 80, 120, 121, 140, 145,
147, 148, 151, 153, 154, 163, 166,
190, 198, 200, 202, 207, 208, 212,
213, 214, 216, 229, 242, 253, 285, 306
Work Breakdown Structure, *Patrz:* WBS
Wujec Tom, 40
wydanie, 209, 210, 229
 planowanie, 207, 212, 227, 229, 230,
 231, 232
wydobywanie, *Patrz:* elicitation
wykres spalania, 233, 234, 236, 270, 271,
287, 294
wymagania, 133, 134, 135, 138, 139, 140,
141, 144, 146, 150, 153, 158, 162, 169,
171, 172, 190, 193, 196
 biznesowe, 18
 funkcjonalne, 18

Y

Young Jeffrey, 72

Z

zadanie, 241, 245, 246, 254, 268, 285
 wielkość, 247
zasada ograniczonego zaufania, 146, 169
zespół, 99, 102, 105, 106, 108, 130, 145,
147, 155, 156, 163, 169, 198, 210, 214,
229, 253, 278
 funkcjonalny, 120, 121
 komponentowy, 122, 123
 prędkość, 149, 160, 191, 199, 210, 218,
 219, 221, 229, 240, 243, 251, 254
 rozproszony, 293, 296
 wielkość, 115, 116
 wielofunkcyjny, 120, 129, 190
 wirtualny, *Patrz:* zespół rozproszony
Zespół Funkcjonalny, 46

Ż

żargon, 136, 138

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

Zwonne metody projektowania, takie jak Scrum, święcą dziś triumfy jako alternatywne podejście do przeprowadzania skomplikowanych, wieloaspektowych projektów, zwłaszcza z dziedziny IT. W przeciwieństwie do tradycyjnych systemów Scrum zakłada podział projektu na krótkie iteracje (sprinty), z których każda kończy się przedstawieniem fragmentu działającego produktu. Niezwykle ważną częścią projektu jest działanie w pełnej zgodzie z osiągnięciami psychologii — nic nie dzieje się tu bez uwzględnienia potrzeb członków zespołu projektowego oraz docelowego klienta.

Jeśli chcesz dowiedzieć się więcej o zwinnych sposobach prowadzenia projektów i zastosować je we własnej pracy, Mariusz Chrapko podpowie Ci, jak planować takie procesy, rozdzielać role, zbierać informacje i unikać poważniejszych błędów. Wskaże, jak szacować projekty i określać postęp prac. Wszystko to, okraszone mnóstwem konkretnych przykładów, barwnych analogii i dowcipnych anegdot, sprawi, że książka nie tylko bardzo Ci pomoże, ale także dostarczy wiele przyjemności podczas lektury!

Myślenie odwrotne w trudnym procesie omijania raf

Witaj, Zmiano! — szybka reakcja na zmienne priorytety

Scrum na przekór utartym sposobom prowadzenia projektów

ScrumMaster i Właściciel Produktu — trudna kooperacja?

Hodowanie i pielęgnacja zwinnych zespołów projektowych

Zdobywanie potrzebnych wiadomości w procesie... mądrej komunikacji

Rewolucyjne narzędzia szacowania projektów

Planowanie projektów w Scrumie i uczciwe śledzenie ich postępów

Planowanie sprintu i planowanie w dłuższej perspektywie

Przegląd i retrospektywa na koniec sprintu

Autor doskonale łączy teorię z praktyką.

Książkę wyróżnia styl, który sprawia, że czyta się ją niezwykle przyjemnie.

Łukasz Łopuszański

redaktor naczelny magazynu „Programista”

**SCRUM —
WYKORZYSTAJ
JEGO SIŁĘ!**

one
D R E S S

Ni katalogowy: 3 24 3



Księgarnia internetowa:

<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• [Książki w promocji](#)

• [Książki najchętniej czytane](#)

• [Książki w promocji](#)

Zamów informacje o nowościach:

• [Książki](#)

• [Magazyny](#)

• [Książki w promocji](#)

• [Magazyny](#)

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

helion.pl
księgarnia
internetowa

Cena 59,00 zł

ISBN 978-83-246-3828-4



9 788324 638284

Informatyka w najlepszym wydaniu