Zarządzanie 3.0

Kierowanie zespołami z wykorzystaniem metodyk agile

Jurgen Appelo

Helion

Tytuł oryginału: Management 3.0: Leading Agile Developers, Developing Agile Leaders

Tłumaczenie: Ireneusz Jakóbik (wstęp, rozdz. 1 – 13, 16);
Joanna Zatorska (rozdz. 14 – 15)

ISBN: 978-83-283-1801-4

Authorized translation from the English language edition, entitled: MANAGEMENT 3.0: LEADING AGILE DEVELOPERS, DEVELOPING AGILE LEADERS; ISBN 0321712471; by Jurgen Appelo; published by Pearson Education, Inc, publishing as Addison Wesley.

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/zarz30
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 1

Dlaczego rzeczy nie są łatwe

Każdy złożony problem ma rozwiązanie, które jest jasne, proste i niewłaściwe.

— H.L. Mecken, dziennikarz i pisarz (1880 – 1956)

Na papierze byłem milionerem. Nieformalni inwestorzy wycenili mój internetowy start-up na 10 milionów euro. Byłem właścicielem 70% finansowej fikcji, którą wokół mnie utworzyli. Zostałem nawet uhonorowany tytułem *przedsiębiorcy roku*[1], ponieważ bardzo dobrze potrafiłem przekazywać swoją wizję. Moje kolorowe wykresy spodziewanych przychodów i zysków wyglądały rewelacyjnie na papierze.

Pieniądze wpłacone przez inwestorów i przeze mnie nie przyczyniły się jednak do osiągnięcia większego zysku. Dodatkowe treści, które utworzyliśmy, nie przyciągnęły do naszej strony większej liczby odwiedzających. Wynajęci przez nas programiści nie przyspieszyli tempa, w jakim tworzyliśmy programy. Interesy, które prowadziliśmy z innymi stronami, nie przełożyły się na większe dochody. W rzeczywistości zarabialiśmy *mniej* niż przed pojawieniem się pierwszych inwestorów. Jestem pewien, że gdybym podał Ci nazwę naszej niechlubnej strony, nie potrafiłbyś jej skojarzyć. Tworzyliśmy tyle hałasu co mucha owocówka w huraganie. A kiedy internetowa bańka dot-comów pękła, zmiotła nasze małe przedsiębiorstwo, łącznie ze wszystkimi innymi start-upami wokół nas.

Mieliśmy jednak niezłą zabawę. I uczyliśmy się. Ach, ile się nauczyliśmy! Jeśli prawdą jest, że ludzie uczą się na własnych błędach, dzisiaj niewiele brakowałoby mi do zyskania statusu Istoty Wszechwiedzącej. Jako szef programistów, lider zespołu, menedżer projektu i twórca oprogramowania popełniałem tyle błędów, że aż dziwię się, iż nie pociągnąłem za sobą całego internetu. Ale czego się nauczyliśmy, to nasze.

Mam nadzieję, że podczas czytania tej książki Ty też sporo się dowiesz. Że będziesz uczyć się na moich błędach i na błędach wielu innych osób przede mną. Jedną z najcenniejszych rzeczy, których nauczyłem się w minionej dekadzie, jest to, że **programowanie zwinne**[2] (patrz rozdział 2., „Programowanie zwinne") stanowi najlepszą metodę tworzenia oprogramowania.

[1] Komunikat prasowy o firmie Millidian jest dostępny, w języku holenderskim, pod adresem *http://www.mgt30.com/millidian/*.

[2] *http://www.mgt30.com/agile/*.

Dowiedziałem się również, że zarządzanie w starym stylu jest na całym świecie największą przeszkodą w przyjęciu programowania zwinnego. Zakładam, że jesteś menedżerem albo kimś zainteresowanym zarządzaniem. Być może jesteś twórcą oprogramowania, szefem działu informatyki, liderem zespołu lub testerem ze zdolnościami przywódczymi. W tej chwili nie ma to znaczenia. Ważne jest, że chcesz się nauczyć zarządzania — zarządzania **zwinnego**. Obiecuję, że tak się stanie. Książka ta nauczy Cię, jak być dobrym menedżerem zwinnym i jak utworzyć zwinną organizację. Już wkrótce się tym zajmiemy, ale *nie* bez solidnych podstaw, co oznacza, że w pierwszej kolejności musisz poznać ludzi oraz systemy, a także dowiedzieć się, co ludzie *myślą* o systemach. Być może zastanawiasz się dlaczego. Otóż dlatego, że lekarze uczą się, jak działa ludzkie ciało. Ponieważ piloci uczą się, jak działa samolot. I ponieważ inżynierowie oprogramowania uczą się, jak działa komputer. To właśnie dlatego menedżerowie muszą się dowiedzieć, jak działają systemy społeczne.

Jedną z rzeczy, których boleśnie doświadczyłem na *sobie samym*, jest fakt, że niezależnie od tego, co zaplanujesz dla systemu, tak się nie stanie. Świat nie działa w taki sposób. System, w którym żyjesz, nie dba o Twoje plany. Możesz myśleć, że A prowadzi do B, i teoretycznie możesz nawet mieć rację, ale teoria rzadko sprawdza się w praktyce, a *przewidywalność* ma podstępną siostrę o imieniu *złożoność*.

Ale nieco się zagalopowałem. Jak wyjaśnię później, ludzie wolą rozumieć rzeczy liniowo, co oznacza, że do naszej historii najlepiej podejść w sposób liniowy. A historia w naszej książce rozpoczyna się od przyczynowości. Rozdział ten zaczyna się od przestudiowania przyczynowości i nieliniowości, a kończy się wprowadzeniem modelu zarządzania 3.0.

Przyczynowość

Idea mówiąca o tym, że coś zdarza się zgodnie z naszym planem (jak miałem nadzieję wtedy, gdy byłem milionerem na papierze), ma swoje korzenie w naszej wrodzonej skłonności do **determinizmu przyczynowego**. Jest to teza, że „każde zdarzenie i stan są zdeterminowane przez swoje uprzednio istniejące przyczyny”[3]. Determinizm przyczynowy mówi nam, że wszystko, co się zdarza, jest spowodowane przez coś, co zdarzyło się uprzednio. Z punktu widzenia logiki oznacza to, że gdybyśmy wiedzieli wszystko o naszej obecnej sytuacji i znali wszystkie możliwe przejścia prowadzące z jednego zdarzenia do innego, moglibyśmy przewidzieć przyszłe wydarzenia, obliczając je na podstawie wydarzeń minionych i praw natury. Potrafisz złapać rzuconą w Twoim kierunku piłkę, ponieważ możesz przewidzieć kierunek, w którym ona poleci. Wiesz również, jak mało pozostanie Ci z wypłaty po imprezie z przyjaciółmi i jaki jest najlepszy sposób na bezkarne rozwścieczenie Twojego brata lub siostry.

W świecie naukowym determinizm przyczynowy odniósł ogromny sukces, umożliwiając naukowcom dokładne przewidywanie wielu różnych zdarzeń i zjawisk. Na przykład korzystając z praw fizyki newtonowskiej, przewidują oni, że kometa Halleya powróci do naszego systemu słonecznego w 2061 roku[4]. Tego rodzaju naukowe przepowiednie są pod względem astronomicznym bardziej wiarygodne niż przepowiednie o końcu świata, którego data jest przekładana za każdym razem, gdy okazuje się, że wcześniejsze proroctwo się nie sprawdziło. Naukowa metoda obliczania przyszłych wydarzeń na podstawie zdarzeń minionych i praw natury

[3] *http://www.mgt30.com/determinism/*.

[4] *http://www.mgt30.com/halley/*.

odniosła taki sukces, że filozof Immanuel Kant promował uniwersalny determinizm przyczynowy jako niezbędny warunek całej wiedzy naukowej [Prigogine, Stengers 1997:4].

Determinizm przyczynowy umożliwia twórcom oprogramowania projektowanie, planowanie i przewidywanie tego, co ich oprogramowanie będzie robić w swoim środowisku roboczym. Piszą oni albo zmieniają kod w celu zdefiniowania lub zmodyfikowania przyszłego zachowania systemu po jego skompilowaniu i wdrożeniu. Jeśli tylko pominiemy błędy, awarie systemu operacyjnego, zaniki zasilania, kierowników księgowości i pozostałe zagrożenia środowiskowe, przekonamy się, że przewidywania programistów często są dość dokładne. Przyczynowość pozwoliła mi w miarę dokładnie przewidzieć, że mój start-up pogrąży się, jeżeli nie znajdę więcej klientów.

Może się to wydawać dziwne, ale sama przyczynowość nie wystarczy. Chociaż możemy przewidzieć powrót komety Halleya i zachowanie programu w produkcji, nie potrafimy dokładnie przepowiedzieć pogody w przyszłym miesiącu. Nie potrafimy też przewidzieć pełnej kombinacji funkcji, walorów, czasu i zasobów projektu programistycznego ani (o ja biedny!) czasu, w którym pojawią się nowi klienci.

W czym tkwi różnica?

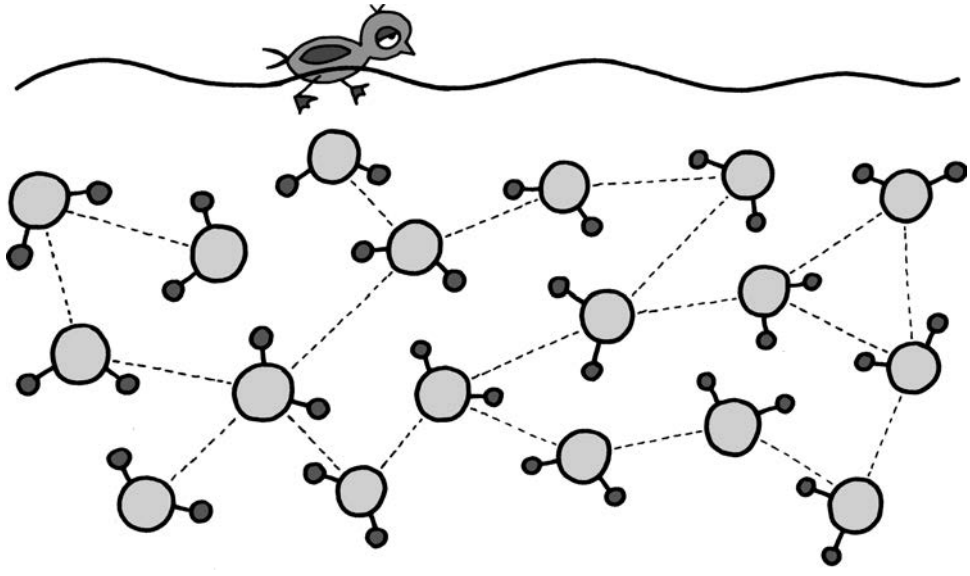
Złożoność

Jeśli przewidywalność jest miłym i solidnym synem naszych sąsiadów, **złożoność** będzie ich tajemniczą i niesforną młodszą siostrą. Przewidywalność pozwala Ci chodzić do pracy, umawiać się na spotkania, uprawiać sport i oglądać telewizję, natomiast złożoność często zamienia interakcję między Tobą a światem w nieprzewidywalny i niemożliwy do opanowania bałagan, pełen zaskakujących problemów i niespodzianek.

Ludzie często mylą złożoność z dużymi liczbami (jak na przykład wtedy, gdy wiele zdarzeń zachodzi w tym samym czasie), ale złożone rzeczy nie zawsze są duże. Weźmy choćby cząstkę wody (rzecz jasna w przenośni, gdyż w przeciwnym razie potrzebowalibyśmy do tego sporej praktyki). Cząstka wody składa się tylko z dwóch atomów wodoru i jednego atomu tlenu. Nic wielkiego, prawda? Mimo to połączenie tych trzech atomów prowadzi do nieprzewidywalnego zachowania się cząstek wody, wywołującego dziwne efekty płynności, gęstości oraz inne zjawiska fizyczne i chemiczne [Solé 2000:13], których nie da się (łatwo) wyjaśnić w kategoriach pojedynczych atomów (patrz rysunek 1.1). Złożoność niekoniecznie wynika z dużych liczb. Wystarczą trzy cząstki wody, aby uzyskać złożone zachowanie, ujęte w słynnym **problemie trzech ciał**[5].

Na szczęście od czasów entuzjastycznego poparcia determinizmu przyczynowego przez Kanta nauka nie stoi w miejscu. Teoria systemów dynamicznych, teoria chaosu, teoria sieci, teoria gier, a także kilka innych dziedzin nauki, poczyniły ogromne postępy w kierunku wyjaśnienia, *dlaczego* niektóre zjawiska są nieprzewidywalne i dlaczego niektórych zdarzeń po prostu nie da się zaplanować ani wyliczyć — można ich tylko doświadczać i obserwować je. Całość korpusu nauki w zakresie systemów złożonych jest czasami zbiorczo nazywana **nauką o złożoności** (patrz rozdział 3., „Teoria systemów złożonych").

[5] *http://www.mgt30.com/euler/*.



Rysunek 1.1. *Co tak naprawdę dzieje się w wodzie?*

Chociaż przyczynowość z powodzeniem rządziła nauką już od XVII wieku, złożoność jest tworem XX wieku; twór ten nabrał rozpędu, od kiedy **teoria złożoności**[6] stała się pod koniec wieku samodzielną dyscypliną naukową. Cytowane jest często stwierdzenie fizyka teoretycznego Stephena Hawkinga, że XXI wiek jest wiekiem złożoności [Chui 2000].

Powstanie teorii złożoności to dobra wiadomość dla menedżerów, liderów zespołów i menedżerów projektów (a także wszelkich innych rodzajów „liderów" i „menedżerów") w organizacjach tworzących oprogramowanie. Oznacza ona, że istnieje nowy, *naukowy* sposób postrzegania złożonych systemów, który uwzględnia problem tworzenia oprogramowania i zarządzania organizacjami. Chociaż obawiam się, że objawienie to nastąpiło dla mnie o 10 milionów euro za późno, zgadzam się ze Stephenem Hawkingiem, iż złożoność to jedno z najważniejszych pojęć XXI wieku.

Nasze liniowe umysły

Niestety, stosując teorię złożoności do rozwiązywania problemów, stajemy w obliczu pewnej niedogodności; nasze umysły wolą przyczynowość od złożoności. W artykule *Born Believers: How your brain creates God* [Brooks 2009] opisano, jak ludzki umysł rozwinął w sobie nadmierne poczucie przyczyny i skutku, co skłania nas do dostrzegania celu i zamiaru wszędzie, nawet gdy ich tam nie ma. W artykule napisano też, że dzieci są przekonane, iż ostro zakończone skałki istnieją po to, aby zwierzęta miały się o co drapać, a rzeki po to, aby łodzie miały po czym pływać. Wydaje się, że ludzki mózg jest skonstruowany w taki sposób, by wszędzie doszukiwać się celowości i przyczynowości. Wszystkiemu wokół nas przypisujemy przyczynę i skutek, nawet kiedy nie ma ku temu żadnych powodów.

[6] *http://www.mgt30.com/complex-systems/*.

> „Widzisz poruszające się krzaki i zakładasz, że ktoś lub coś tam jest". (...) Takie nadinterpretowanie przyczyny i skutku prawdopodobnie wyewoluowało po to, abyśmy mogli przetrwać. Jeżeli wokół są drapieżniki, nie będzie dobrze, jeśli spostrzeżesz je w 9 przypadkach na 10. Uciekanie wtedy, gdy nie musisz, jest niewielką ceną za uniknięcie ryzyka w porównaniu z sytuacją, gdy zagrożenie jest realne[7].

Nasze mózgi są stworzone do przedkładania tego, co nazywam „myśleniem liniowym" (przy założeniu przewidywalności przyczyny i skutku), nad „myślenie nieliniowe" (przy założeniu, że sprawy są o wiele bardziej skomplikowane). Jesteśmy przyzwyczajeni do historii opowiadanych liniowo, od początku do końca. W szkole nauczono nas równań liniowych i w znacznej mierze pominięto o wiele powszechniejsze równania nieliniowe, po prostu dlatego, że są zbyt trudne do rozwiązania. Łatwiej przyjmujemy stwierdzenie „on to zrobił" niż „coś się wydarzyło". Kiedy tylko pojawia się problem B, zakładamy, że wywołało go zdarzenie A. Kryzys finansowy wywołali bankierzy, utratę pracy spowodowali imigranci, zła atmosfera w biurze jest „zasługą" szefa, za topniejącą polarną czapę lodową odpowiada emisja CO_2, a zespół nie zdążył w terminie, bo ktoś zawalił sprawy. Nasze liniowo myślące umysły postrzegają świat jako miejsce wypełnione łatwymi do wyjaśnienia zdarzeniami z prostymi przyczynami i prostymi skutkami. Gerald Weinberg nazywa to **logicznym błędem przyczynowości** [Weinberg 1992:90].

Mentalne uzależnienie od determinizmu przyczynowego spowodowało, że ludzie zaczęli korzystać ze *sterowania* w swoich próbach upewnienia się, iż *właściwe* zdarzenia są odseparowane od niewłaściwych. W końcu jeśli wiemy, że sytuacja A prowadzi do zdarzenia B, podczas gdy sytuacja A' prowadzi do zdarzenia C, a C jest lepsze niż B, musimy tylko przekształcić A na A' i sprawy przybiorą lepszy obrót. A przynajmniej na to wygląda.

Inżynierowie i osoby o technicznych umysłach cechują się szczególną podatnością na taką koncepcję sterowania. To właśnie inżynierowie opracowali koncepcję **zarządzania naukowego**[8], stylu zarządzania typu „rozkazuj i kontroluj", który zdobywa szczyty popularności od wczesnych lat XX wieku. Również inżynierowie opracowali systemy kontroli, które nadal możemy znaleźć w wielu organizacjach [Stacey 2000, a:7]. Obecnie wszyscy już wiemy, że systemy te działają poprawnie tylko przy powtarzalnych zadaniach, które nie wymagają zbyt wiele myślenia. *Nie* sprawdzają się natomiast w przypadku kreatywnych procesów produkcyjnych! Wydaje się, iż możemy oczekiwać od inżynierów, że spróbują wyciągnąć ludzi z bagna zarządzania, w które wszystkich wepchnęli.

Przyczynowość w zarządzaniu sprawia, że menedżerowie szukają przyczyn, które dzięki dokładnemu projektowi i drobiazgowemu planowaniu z góry do dołu zapewnią im uzyskanie potrzebnych wyników. Im większa organizacja, tym większy wysiłek należy włożyć w dekonstrukcję i ponowną konstrukcję całego systemu w celu osiągnięcia pożądanych wyników.

W przeszłości uparcie tworzyłem własne iluzje szczegółowych projektów oraz planowania z góry do dołu. Mój zdobywający nagrody plan liczył sobie ponad 30 stron starannie opracowanego nonsensu. Szczegółowo opisałem, w jaki sposób stanę się bogaty. Wtedy w to wierzyłem. Sam tak napisałem, więc musiała to być prawda.

---

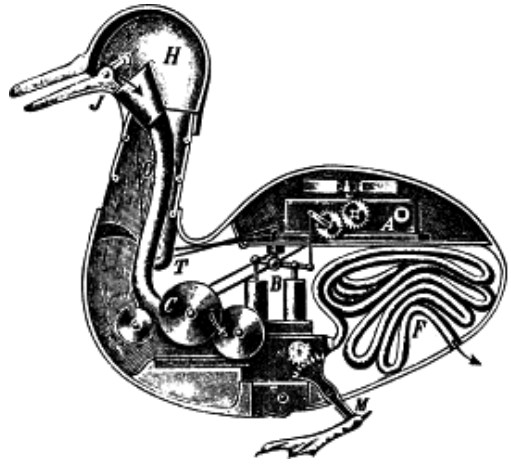
[7] Brooks M., *Born believers: How your brain creates God*, „New Scientist", 4 lutego 2009; *http://www.mgt30.com/believers/*, [Brooks 2009:32].

[8] *http://www.mgt30.com/scientific-management/*.

Redukcjonizm

Podejście polegające na rozkładaniu systemów na części i analizowaniu sposobu, w jaki części te współpracują ze sobą jako całość, nazywane jest **redukcjonizmem**[9]. Idea ta bazuje na przekonaniu, że „zjawisko można w zupełności wyjaśnić w kategoriach innych, bardziej fundamentalnych zjawisk". Rozkładamy samolot i rozumiemy, jak działa, dzięki przestudiowaniu wszystkich jego części; możemy zrozumieć system komputerowy, analizując jego kod; współcześni naukowcy próbują zrozumieć choroby oraz schorzenia, analizując ludzki genom, i mają nadzieję znaleźć poszczególne geny „odpowiedzialne" za wszelkiego rodzaju „problemy".

Podejście redukcjonistyczne sprawdza się dobrze, ale tylko do pewnego stopnia (patrz rysunek 1.2). Po wielu dekadach badań naukowcy nadal nie rozumieją, jak działa ludzka świadomość. Chociaż teorie ekonomiczne istnieją już od ponad stu lat, ekonomiści w dalszym ciągu nie dysponują modelami, które dokładnie przewidywałyby kryzysy finansowe. Wiele teorii stosowanych do modelowania zmian klimatycznych przewiduje diametralnie różne konsekwencje globalnego ocieplenia. Mimo że mamy wiele modeli tworzenia oprogramowania, realizowane na całym świecie projekty nadal przynoszą nieprzewidywalne wyniki. Organizmy, ludzka świadomość, ekonomia, klimat i projekty programistyczne zachowują się w sposób, którego nie można przewidzieć poprzez ich dekonstrukcję i studiowanie ich elementów składowych.



Rysunek 1.2. *Redukcjonizm posunięty nieco za daleko*[10]

Ludzie także źle interpretują

Kilku recenzentów mojej książki zauważyło, że ludzie notorycznie źle interpretują swoje środowisko. My, istoty ludzkie, mamy tendencję do ignorowania rzeczy, w które nie wierzymy, i odrzucamy wszystko to, co nie odpowiada naszym mentalnym modelom. Zjawisko to również ma swój wkład w to, że niedokładnie przewidujemy, co się stanie.

[9] *http://www.mgt30.com/reductionism/*.

[10] Rysunek z Wikipedii; w domenie publicznej: *http://www.mgt30.com/duck/*.

Holizm

Holizm[11] to idea zakładająca, że zachowanie systemu nie może być w pełni determinowane tylko przez jego części składowe. To system jako całość w istotny sposób determinuje swoje zachowanie. Holizm często jest postrzegany jako przeciwieństwo redukcjonizmu, chociaż naukowcy zajmujący się złożonością uważają, że te dwa podejścia łączy właśnie złożoność i że oba są potrzebne, choć niewystarczające [Corning 2002:69].

Nawet niektórzy z najbardziej zagorzałych redukcjonistów odrzucają ideę, że wszystkie zjawiska można wyjaśnić w kategoriach ich elementów składowych. Filozof Daniel Dennett ukuł termin **chciwy redukcjonizm**[12] [Dennett 1995], mając na myśli formy myślenia redukcjonistycznego, w którym dane zjawisko jest wyjaśnianie z perspektywy jego elementów składowych. Na przykład stwierdzenie, że hiperłącza „nie są niczym więcej jak tylko elektronami i tak naprawdę nie istnieją", byłoby formą chciwego redukcjonizmu. Mój kontrargument na chciwy redukcjonizm jest taki, że jeśli chciwi redukcjoniści mają rację, to tak naprawdę nie istnieją, co obala ich niedorzeczne argumenty. Ale to tylko dygresja.

Celem **redukcjonizmu hierarchicznego**, pojęcia zasugerowanego przez biologa ewolucyjnego Richarda Dawkinsa [Dawkins 1996], jest osiągnięcie kompromisu za pomocą poglądu głoszącego, że system złożony można zdefiniować jako hierarchię, której każdy poziom może być opisany w kategoriach elementów położonych w hierarchii o jeden stopień w dół, ale nie niżej. Dzięki temu skutecznie zostaniesz pozbawiony możliwości tłumaczenia się, że Twój projekt poniósł porażkę, bo szyki popsuła Ci banda kwarków i leptonów.

Wiele osób błędnie uważa, że hipoteza redukcjonistyczna implikuje istnienie hipotezy „konstrukcjonistycznej", z której wynika, iż dowolny system można skonstruować, gdy zrozumiemy jego elementy składowe. To założenie jest fałszywe, ponieważ jeśli nawet w pełni zrozumiemy wszystkie elementy systemu, nie oznacza to, że całość jest prostą sumą części [Miller, Page 2007:41]. Znajomość elementów niższego poziomu nie implikuje naszej umiejętności skonstruowania systemu wyższego poziomu. Nawet jeżeli zaprzęgniemy redukcjonizm do prześledzenia problemu do jego źródeł (czego dobrym przykładem może być *technika analizy przyczyn źródłowych*[13]), to, co ciekawe, nie będziemy mogli zastosować podejścia konstrukcjonistycznego w celu zbudowania systemu, który w pierwszym rzędzie będzie *zapobiegać* powstawaniu takich problemów. Możemy się na przykład dowiedzieć, dlaczego ludzkie serce zawodzi (redukcjonizm), ale nie potrafimy stworzyć serca, które nie zawiedzie (konstrukcjonizm).

Czy analiza przyczyn źródłowych jest pozbawiona wartości?

Analiza przyczyn źródłowych ma *ogromną* wartość. Mam na myśli to, że dzięki analizie przyczyn źródłowych można spojrzeć wyłącznie w przeszłość. Analiza ta pomaga rozwiązywać problemy, które już wystąpiły, dzięki czemu nie wystąpią one ponownie, ale nie pomoże Ci przewidzieć, co *pójdzie* źle w przyszłości.

[11] *http://www.mgt30.com/holism/*.

[12] *http://www.mgt30.com/greedy-reductionism/*.

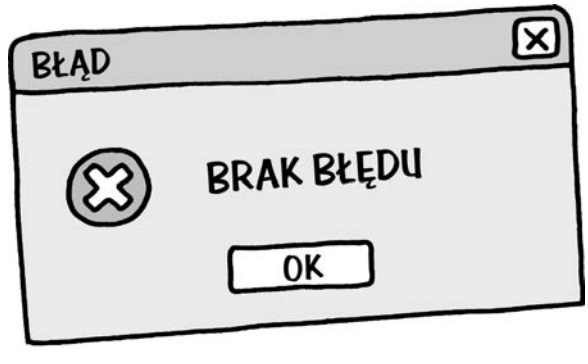
[13] *http://www.mgt30.com/root-cause/*.

Zarządzanie hierarchiczne

Zarówno spojrzenie holistyczne, jak i hierarchiczne spojrzenie redukcjonistyczne są zgodne co do tego, że nie wszystko w systemie złożonym można wyjaśnić poprzez szukanie przyczyn w niższych poziomach danego systemu. Oba spojrzenia dopuszczają, aby na każdym poziomie istniały nowe i *nieredukowalne* właściwości. Na przykład niezależnie od tego, jak mocno byś się przypatrywał, nie znajdziesz łatwych do zidentyfikowania dźwigni, pokręteł i przekładni służących do chodzenia, pływania i kwakania w zdekonstruowanej kaczce (patrz rysunek 1.2). Mimo to gdy spotkasz takie zwierzę w parku, rozpoznasz je jako kaczkę.

Fakt ten rodzi daleko idące konsekwencje dla menedżerów złożonych systemów, takich jak Ty i ja, a także wielu innych kierowników produkcji, menedżerów projektu i liderów zespołów. Wynika z niego, że osoby, które wiedzą wszystko o jednym poziomie systemu hierarchicznego, mogą nie mieć kwalifikacji do zarządzania niższymi lub wyższymi poziomami tego systemu, gdyż poziomy te wymagają innego rodzaju wiedzy. Biolog molekularny może nie mieć „kwalifikacji" ogrodnika, ponieważ ze zrozumienia działania biologii na poziomie komórek eukariotycznych, genów i RNA nie wynika wiedza na temat pielęgnacji ogrodu. Z kolei ogrodnik nie musi wiedzieć nic o chromosomach i genomach, aby być dobrym ogrodnikiem. Podobnie dyrektor generalny organizacji musi dużo wiedzieć o zarządzaniu biznesem, ale może być kompletnym ignorantem w zakresie coachingu oraz pozostałych umiejętności związanych z zarządzaniem ludźmi (jestem pewien, że wielu czytelników mogłoby przedstawić na ten temat relację z pierwszej ręki).

Zarządzanie organizacjami wymaga innego rodzaju wiedzy i doświadczeń niż zarządzanie ludźmi, chociaż ***pewna*** znajomość niższych poziomów *mogłaby* być przydatna. Inżynier oprogramowania Joel Spolsky zaproponował **prawo przeciekających abstrakcji** [Spolsky 2002], wyjaśniające, jak elementy systemu mogą manifestować swoją obecność na wyższych poziomach w sposób sprzeczny z intuicją, który powinien ukrywać szczegóły implementacji niższego poziomu. Warstwy oprogramowania wyższego poziomu, które niedomagają na skutek zdarzeń zachodzących w leżących niżej implementacjach, są uważane za przeciekające. Wyświetlane użytkownikom nieczytelne komunikaty o błędach są kolejnym, często spotykanym efektem przeciekających abstrakcji w oprogramowaniu (patrz rysunek 1.3).



Rysunek 1.3. *Wynik przeciekającej abstrakcji?*

Podobne problemy można zauważyć w innych systemach złożonych. Mój świadomy umysł cierpi czasami na zamroczenia, *déjà vu*, roztargnienie, chaotyczne wspomnienia oraz inne dziwne zjawiska, które można wyjaśnić tylko jako występujące w mojej sieci neuronowej zakłócenia niższego poziomu, przeciekające na wyższy poziom, nazywany przeze mnie umysłem. Nie muszę jednak analizować własnych ścieżek neuronowych, aby dobrze korzystać ze świadomości, chociaż miło jest dowiedzieć się od neurologów, że kłopotliwe przypadki mojego umysłu są dość powszechne. Podobnie nie musisz w pełni rozumieć programowania w asemblerze, żeby pisać dobre, wysokopoziomowe programy, choć *pewna* niskopoziomowa wiedza mogłaby ułatwić Ci czasami życie. Tak samo jest z zarządzaniem. Dyrektor generalny nie musi być dobrym szefem, by kierować organizacją, gdy wszystkie „ludzkie sprawy" zostały przydzielone zaufanemu zespołowi menedżerskiemu (w przeciwieństwie do szefów programistów, menedżerów projektu i liderów zespołów, którzy na co dzień muszą zarządzać ludźmi). Jednakże przynajmniej *kilka* umiejętności interpersonalnych mogłoby się przydać w przypadku, gdy problemy niższego poziomu przedostają się na powierzchnię wyższych poziomów (innymi słowy, gdy sprawy zaczynają przeciekać).

Zarządzanie zwinne

Kiedy zarządzanie hierarchiczne zaczyna obejmować złożoność i myślenie nieliniowe, pojawia się coś, co nazywam **zarządzaniem zwinnym**. Jest ono logicznym uzupełnieniem **programowania zwinnego** — to podejście do tworzenia oprogramowania zostało wypracowane niezależnie przez klika grup oraz pojedynczych osób na przestrzeni lat dziewięćdziesiątych XX wieku (patrz rozdział 2.). Powstało na skutek niezadowolenia z wielu niepowodzeń deterministycznego podejścia do produkcji oprogramowania, w którym ścisła kontrola, szczegółowy projekt oraz planowanie z góry do dołu dawały wynik w postaci wielu intensywnie zarządzanych, ale katastrofalnie realizowanych projektów programistycznych.

Programowanie tkwi (częściowo) swoimi korzeniami w teorii złożoności, gdyż uznaje, że determinizm przyczynowy nie wystarcza do uzyskania udanych projektów. Dobrze znane koncepcje zwinne, takie jak *samoorganizacja* i *emergencja*, zostały skopiowane bezpośrednio z literatury poświęconej nauce o złożoności [Schwaber, Beedle 2002], a praktycy Agile rozumieją obecnie, że przy korzystaniu z podejścia kontrukcjonistycznego nie ma możliwości uniknięcia niepowodzeń. Tylko dzięki wielokrotnemu ponoszeniu porażek i oczyszczaniu systemu z ich przyczyn można miarowo rozwijać system i sprawić, że będzie on działać prawidłowo. To prawie jak wychowywanie dzieci.

Wbrew ogromnemu sukcesowi w postaci wielkości zwrotu z inwestycji w programistycznych projektach zwinnych [Rico 2009] wielu menedżerów na całym świecie ponosi odpowiedzialność za utrudnianie wprowadzania zwinnego zarządzania oraz zwinnego programowania w swoich organizacjach. Badania nad przyjmowaniem praktyk zwinnych wskazują, że zarządzanie zmianami, kultura organizacyjna, wsparcie kierownicze, wiedza zespołów i presja zewnętrzna stanowią największe przeszkody w dalszym wprowadzaniu metod Agile i powodują niepowodzenia projektów programistycznych [Version-One 2009]. W większości za porażki te odpowiada jednak kierownictwo. Zakładając, że raporty z badań są wiarygodne (a nie mam podstaw, aby sądzić, że jest inaczej), wydaje się, iż menedżerowie na całym świecie stwarzają problemy, zamiast przyczyniać się do ich rozwiązywania. Niestety, reguła ta dotyczy nie tylko programowania zwinnego. Tak samo dzieje się w przypadku niemal każdej poważnej zmiany organizacyjnej.

W książce tej stoję na stanowisku, że w dowolnej sytuacji zarządzania zmianami tradycyjne kierownictwo zwykle nie jest rozwiązaniem, lecz stanowi problem, co jest poglądem wyrażonym już wiele lat temu przez W. Edwardsa Deminga. To właśnie dlatego potrzebujemy teorii zarządzania zwinnego, tj. teorii zarządczej, która jest blisko spokrewniona z programowaniem zwinnym.

Moja teoria wszystkiego

Czy istnieje jakaś teoria, która mogłaby pomóc menedżerom, podpowiadając im, co robić w środowisku zwinnym? W ciągu kilku dekad zaproponowano wiele teorii zarządzania, chociaż większość z nich nie stanowi teorii w sensie naukowym [Lewin, Regine 2005:5]. Prawdziwa teoria naukowa nie tylko identyfikuje pewne zjawisko naturalne, ale też formułuje teorie dotyczące obserwacji poczynionych w rzeczywistym świecie, wyjaśniając, czego można oczekiwać, jeszcze *zanim* coś się wydarzy. Pod tym względem większość „teorii" zarządzania zawodzi. Często nie są to teorie, lecz techniki. Zamiast przedstawiać opis funkcjonowania świata, oferują one (przydatne) wskazówki dotyczące radzenia sobie z pewnymi problemami oraz sytuacjami. Dobrym przykładem może tu być teoria ograniczeń (ang. *Theory of Constraints*; TOC). Nie jest to teoria naukowa, ale filozofia zarządzania, oferująca technikę ulepszania procesów, która służy osiąganiu celów przez ciągłe skupianie się na ograniczeniach.

Czy oznacza to, że mogę teraz zaproponować własną „teorię" zarządzania zwinnego, mając skrytą nadzieję na zajęcie miejsca obok takich osobistości jak Porter, Deming i Drucker? Obawiam się, że nie.

Kiedyś miałem nadzieję, że znajdę *teorię wszystkiego*, która dotyczyłaby zarządzania zespołami deweloperskimi. Teoria ta opisywałaby reguły wszystkich zespołów deweloperskich i pomagałaby ludziom za pomocą pełnego, jednolitego modelu zarządzania zespołami. Z perspektywy czasu myślę, że mój umysł cierpiał wówczas z powodu ogromnego wycieku abstrakcji.

Na szczęście wkrótce odkryłem, że ów cel pozostawał poza moim zasięgiem z dwóch powodów. Po pierwsze, istniało już mnóstwo teorii opisujących współpracę ludzi w zespołach. Dziedzina ta znana jest pod nazwą **złożoności społecznej** — to badanie grup społecznych jako systemów złożonych [polecane publikacje w tym zakresie to książka *Small Groups as Complex Systems* (Arrow 2000) oraz magazyn „Emergence: Complexity & Organization"[14]]. Po drugie, sama teoria złożoności mówi nam, że nie da się zbudować jednolitego modelu systemu złożonego. Każda próba utworzenia jednego modelu w pełni opisującego pewną klasę złożonego systemu zawsze będzie skazana na porażkę. Zagadnienie to poruszam w rozdziale 16., „Wszystko jest błędne, chociaż coś z tego jest przydatne". Kiedy na nie natrafiłem, odetchnąłem z ulgą. *Nie da się. Świetnie! To znaczy, że mogę pracować nad czymś innym!* Trudno mi podać lepszy przykład wczesnego niepowodzenia (**twierdzenia Gödla**[15] udowadniają, że ta sama niemożliwość odnosi się do *wszystkich* teorii jednolitych; być może powinniśmy się cieszyć, że naukowcy nie poddają się tak szybko jak ja).

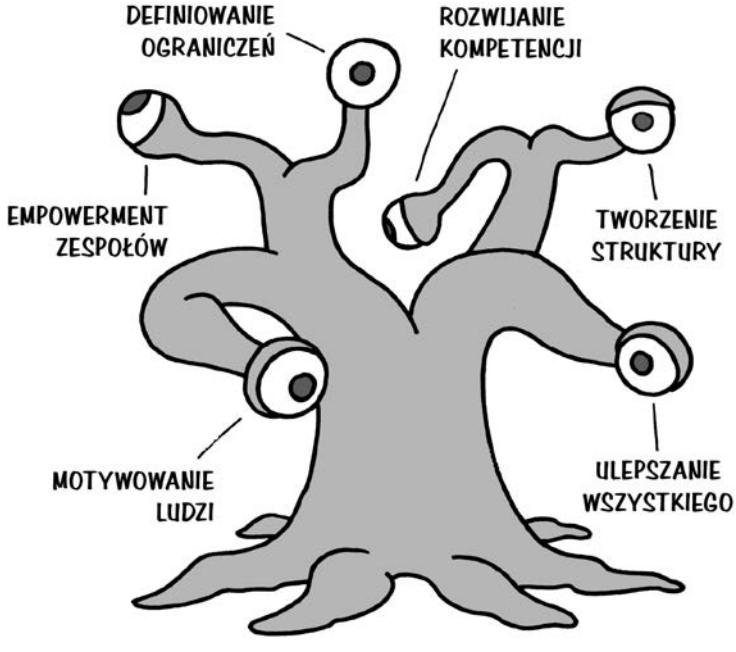
[14] Magazyn „E:CO" jest wydawany przez Emergent Publications; patrz *http://www.mgt30.com/eco/*.

[15] *http://www.mgt30.com/godel/*.

Książka i model

Książka ta może Ci pomóc w staniu się lepszym menedżerem. W szczególności dowiesz się, jaki jest zakres Twojej odpowiedzialności jako *zwinnego menedżera* w *zwinnej* organizacji, która realizuje projekty zwinnego programowania. Poznasz również liczne techniki służące do przekładania teorii na codzienną praktykę. Książka pokaże Ci, jak zarządzać zespołami wiedzącymi, że systemy zazwyczaj nie są liniowe, lecz złożone, a także jak skupiać się na dostosowalności zamiast na przewidywalności. Nie ma większego znaczenia, czy jesteś menedżerem programistów, liderem zespołu, dyrektorem ds. technologii, czy programistą. W końcu wszyscy zarządzamy środowiskiem, które nas otacza. Postarajmy się zrozumieć, jak robić to dobrze.

Model wykorzystany w tej książce zobrazowano na rysunku 1.4. Nadałem mu nazwę „Martie, model zarządzania 3.0". Martie spogląda na organizacje z sześciu różnych perspektyw. Każda z nich została opisana oddzielnie w dwóch rozdziałach, zarówno od strony teoretycznej, jak i praktycznej. Model zarządzania 3.0 jest moją reprezentacją różnych aspektów zarządzania zwinnego. Zanim jednak omówię go w szczegółach, pozwolę sobie przedstawić podstawy jego dwóch składników, *zwinności* i *złożoności*, a także pokrótce opiszę ich dzieje. Rozdział 2. prezentuje w skrócie programowanie zwinne, podczas gdy w rozdziale 3. opisuję podstawy teorii systemów złożonych. Najważniejsze zagadnienie, tj. sposób kierowania zespołami deweloperskimi przy wykorzystaniu sześciu perspektyw modelu zarządzania 3.0, przedstawiam w zasadniczej części książki, która rozpoczyna się rozdziałem 4., „System informacyjno-innowacyjny", a kończy rozdziałem 15., „Jak ulepszać wszystko". Na koniec, w rozdziale 16., zamieściłem krótką konkluzję.



Rysunek 1.4. *Martie, model zarządzania 3.0*

Chciałbym tylko, aby książka taka jak ta była mi dostępna (albo znana), kiedy dziesięć lat temu tworzyłem swój internetowy start-up. Wtedy jednak *mógłbym* zostać milionerem i prawdopodobnie nie zawracałbym sobie głowy pisaniem tej książki, co zdaje się dowodzić, że planowanie kariery często jest zbyteczne, a niepowodzenie może być zamaskowanym szczęściem.

Podsumowanie

Mózg ludzki zaprogramowany jest w taki sposób, aby przyjmował, że każde zdarzenie ma swoją identyfikowalną przyczynę. Tę cechę nazywamy przyczynowością. Jest ona przydatna w przewidywaniu i planowaniu. Bardzo często rzeczy są jednak bardziej złożone, niż się to wydaje. Nauka o złożoności uczy nas, że przykładanie liniowego myślenia do skomplikowanych problemów może prowadzić do bolesnych pomyłek.

Chociaż redukcjonizm (poznawanie systemu poprzez poznawanie jego części) odnosi sukcesy w nauce, obecnie przyjmuje się powszechnie, że z redukcjonizmem można zapędzić się za daleko.

Aby zrozumieć wiele ze złożonych problemów, potrzebna jest perspektywa bardziej holistyczna, co jest celem nauki zajmującej się poznawaniem złożoności społecznej. Oferuje ona holistyczne spojrzenie na zjawiska zachodzące w grupach ludzkich.

Zarządzanie 3.0 to model zarządzania zwinnego, w którym złożonościowe myślenie jest stosowane przez zwinne zespoły deweloperskie.

Refleksje i działania

Zobaczmy, czy uda Ci się zrealizować w swojej organizacji niektóre z idei przedstawionych w tym rozdziale.

- Przyjrzyj się jednemu z problemów ze swojej listy zagadnień do rozwiązania. Spróbuj wyobrazić sobie przyczynę problemu. Czy masz pewność, że jest to jedyny powód? Skąd to wiesz? Czy omówiłeś ten problem ze wszystkimi interesariuszami? Czy każdy z nich zgadza się co do tej jednej jedynej przyczyny? Zastosuj to proste ćwiczenie umysłowe do każdego ze swoich najważniejszych problemów. Upewnij się, że nie upraszczasz nadmiernie ich złożoności i że nie skupiasz się na niewłaściwej przyczynie.
- Jeśli osoby w Twojej organizacji stosują technikę analizy przyczyn źródłowych (taką jak „5 dlaczego”[16]), wciągnij je w dyskusję o skłonnościach tych technik do upraszczania związków przyczynowo-skutkowych. Liczne efekty występujące w złożonych systemach mają *wielorakie* przyczyny oraz charakteryzują się cyklicznymi związkami między przyczynami a skutkami. Żadna z przyczyn nie jest rzeczywiście źródłowa, w związku z czym techniki analizy przyczyn źródłowych mogą nie objąć w pełni złożoności świata, w którym żyjesz. Podczas wymiany zdań z kompetentnym kolegą możecie jednak tę złożoność dostrzec. Zorganizuj taką dyskusję.

[16] *http://www.mgt30.com/5-whys/*.

Rozdział 2

Programowanie zwinne

Każdego ranka budzę się z postanowieniem, że zmienię świat i nieźle będę się bawić. Czasami w związku z tym zaplanowanie mojego dnia jest trudne.

— E.B. White, amerykański pisarz (1899 – 1985)

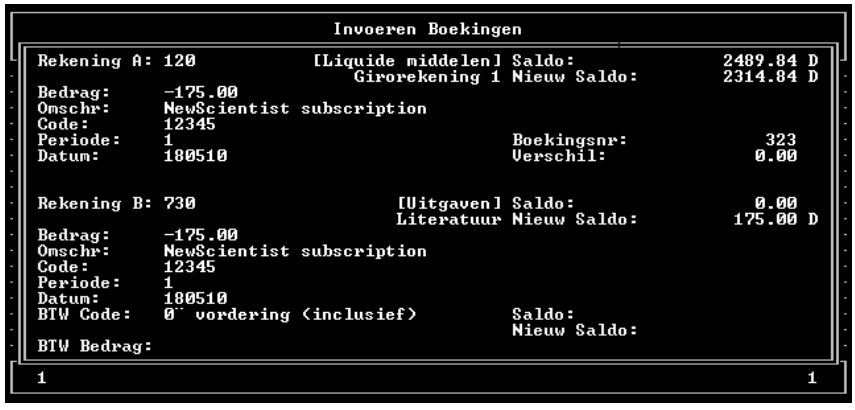
Lektura tego rozdziału nie jest obowiązkowa. Jeśli znasz programowanie zwinne, wiesz już dużo (a może nawet wszystko) o tym, co rozdział ten ma do zaoferowania. Jego celem jest przedstawienie zwięzłego wprowadzenia do programowania zwinnego, szczególnie tym czytelnikom, którzy chcieliby wiedzieć nieco więcej na temat podstaw i kontekstu metod Agile, *zanim* zaczniemy poznawać rolę menedżera w organizacjach zwinnych (co nastąpi w rozdziale 4., „System informacyjno-innowacyjny").

W książce tej zakładam, że już trochę *znasz* podstawy programowania zwinnego. Na razie jednak możesz udawać, że według Ciebie XP to stary system operacyjny, i kontynuować czytanie.

Preludium do programowania zwinnego

Liczenie pieniędzy sprawia mi prawie tyle samo radości co ich wydawanie. We wczesnych latach 90. XX wieku, gdy studiowałem na Uniwersytecie Technicznym w Delft, napisałem w wolnym czasie program księgowy. Zrobiłem to, bo miałem z tego frajdę, chociaż pewną niedogodnością była okoliczność, że nie miałem pieniędzy, które mógłbym liczyć. Być może w jakimś mrocznym zakątku swojego umysłu skrywałem nadzieję, że miliony napłyną do mnie automatycznie, gdy już będę gotowy do ich liczenia. Niestety, nigdy to nie nastąpiło.

Cały program (około 30 000 wierszy kodu) napisałem sam. Nie znałem żadnej formalnej metodologii, brakowało mi doświadczenia w tworzeniu oprogramowania, nie miałem menedżera, nauczyciela ani mentora. Miałem za to czas, komputer, wizję oraz silną motywację do stworzenia wspaniałego produktu (patrz rysunek 2.1).



Rysunek 2.1. *JEBS 2.0, mój 20-letni program księgowy (w języku holenderskim)*

O dziwo sprzedałem swój program kilkudziesięciu klientom. Niektórzy z nich byli zdumieni, że oprogramowanie księgowe może być proste i przyjazne, a zarazem może dobrze wyglądać (jak na program z 1990 roku). Obecnie, 20 lat później, *nadal* wykorzystuję do własnych potrzeb ten sam stary program księgowy. Przez te 20 lat znalazłem w nim tylko trzy niewielkie błędy.

Jak to jest możliwe? W jaki sposób niedoświadczony programista może zbudować coś tak wysokiej jakości, co przez 20 lat działa niemal bezbłędnie?

Nie mam zielonego pojęcia.

Chociaż... mogę wymienić kilka okoliczności, które zapewne rozpoznają praktycy programowania zwinnego:

- **Budowałem swój produkt z pasją.** Miałem trochę doświadczenia z aplikacjami księgowymi i byłem przekonany, że są to pochodzące z piekła pachołki, które przy każdym naciśnięciu klawiszy komputera próbują wyssać dusze użytkowników. Miałem wizję, że mój program będzie *inny*. W przeciwieństwie do pozostałych programów z tej dziedziny mój miał być przyjemny w użyciu.
- **Sam byłem swoim krytycznym klientem.** Program tworzyłem dla *samego siebie*, a nie dla innych. Oczywiście byłem szczęśliwy, że znalazłem klientów, nawet jeśli nie przynieśli mi oni milionów, na które miała nadzieję moja mroczna część. Robiłem jednak wszystko, aby produkt działał w taki sposób, w jaki to *ja* chciałem, żeby działał.
- **Nie miałem planu, a tylko listę funkcji.** Zacząłem od funkcji, które były mi potrzebne na co dzień, takich jak wprowadzanie nowych transakcji. Następnie przeszedłem do mniej krytycznych zadań, takich jak tworzenie bilansów i korekt. Pracę kończyłem, pisząc funkcje, które dobrze byłoby mieć, takie jak strony z pomocą i eksport danych, aż się tym wszystkim zmęczyłem i oznajmiłem, że produkt jest gotowy.
- **Proces tworzyłem podczas budowania produktu.** Budując program, kierowałem się prostą listą kontrolną dla każdej procedury, którą pisałem, a lista ta stopniowo wydłużała się w miarę upływu czasu. W ogóle nie słyszałem o testach jednostkowych, ale moje kontrole i powtórne kontrole, moja codzienna dyscyplina mogłyby rywalizować z dyscypliną pilota samolotu.

To wszystko. Miałem motywację, krytycznego klienta, dyscyplinę, samoorganizujący się proces i żadnego z góry opracowanego planu. Nie miało znaczenia, że nigdy wcześniej nie robiłem czegoś podobnego. Liczyło się to, że chciałem się nauczyć.

Dziesięć lat po utworzeniu własnego programu księgowego dowiedziałem się, że część procesu, z którego wówczas korzystałem, została znienacka nazwana „programowaniem zwinnym". A teraz, dziesięć lat po tym, jak to odkryłem, piszę książkę o brakującym składniku metody Agile. Zakres tej książki jest mniej więcej taki sam jak zakres mojego starego programu księgowego. Podobnie jak wtedy, mroczna część mojego umysłu jest już gotowa na rozpoczęcie liczenia.

Księga programowania zwinnego

Na początku inżynierowie stworzyli komputery i oprogramowanie. Oprogramowanie było bezkształtne i złe, a twarze użytkowników spowijała ciemność. I powiedzieli inżynierowie: „Niech stanie się struktura" — i stała się struktura.

Prawdę mówiąc, całkiem sporo struktury.

W ciągu ostatnich pięćdziesięciu czy też sześćdziesięciu lat wielu inżynierów oprogramowania zajmowało się ogromnymi różnicami w jakości programów, które były tworzone rozmaitymi podejściami *ad hoc* do produkcji oprogramowania. Zaczęli więc tworzyć i wymyślili podejścia *formalne*. Narodziła się dziedzina zwana **inżynierią oprogramowania**[1]. Przyjęto w niej założenie, że produkcja oprogramowania jest przedsięwzięciem *inżynieryjnym*, i zaproponowano wiele modeli, metod, platform, języków, wzorców i technik, które miały pomagać programistom w budowaniu lepszego oprogramowania. O dziwo w przypadku wielu projektów programy wcale nie były lepsze. Formalne podejścia *przyczyniły* się za to do wprowadzenia biurokracji. Tworzenie oprogramowania trwało tak długo i wymagało przekazywania takich ilości dokumentów, że „formalne" wymagania zmieniały się na długo przed tym, zanim dostarczano gotowe systemy. Tymczasem małe zespoły zapalonych i zdyscyplinowanych programistów, z wdrażanymi ad hoc procesami i elastycznymi wymaganiami, za ułamek kosztów i w krótszym czasie tworzyły produkty wyższej jakości. W akcie tworzenia powstały słonie, ale to mrówki uciekały z pożywieniem.

We wczesnych latach 90. ubiegłego wieku wynaleziono nowe podejście, któremu nadano nazwę ***Rapid Application Development*** **(RAD)**[2]. Łączyło ono w sobie niektóre z formalnych technik „wagi ciężkiej" inżynierii oprogramowania (takie jak tablice zmian, inspekcje i metryki) z działaniami praktycznymi (jak prototypowanie, ewolucyjne tworzenie oprogramowania oraz intensywna współpraca z klientem), stosowanymi przez wiele odnoszących sukcesy zespołów projektowych pracujących ad hoc [McConnell 1996]. Mieszanie podejść formalnych i ad hoc doprowadziło ostatecznie do powstania pierwszych nazwanych metod tworzenia oprogramowania „wagi lekkiej", takich jak **Evo**[3] (1988), **Scrum**[4] (1995), **DSDM**[5] (1995),

[1] *http://www.mgt30.com/software-engineering/*.

[2] *http://www.mgt30.com/rad/*.

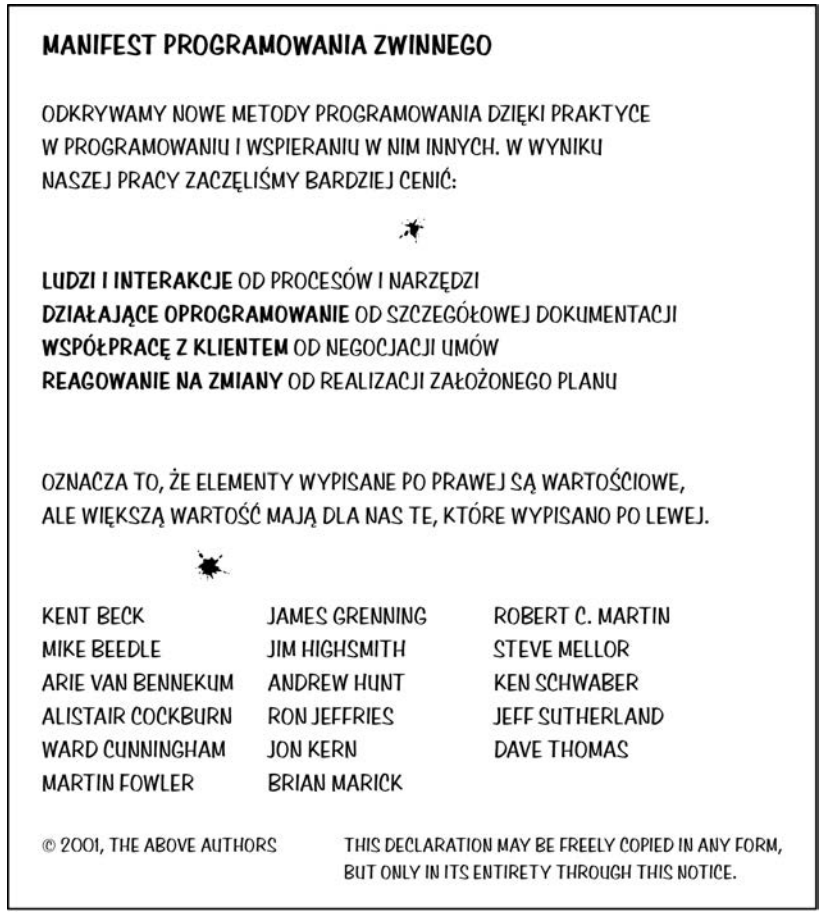
[3] Manuskrypt EVO, datowany na dzień 21 sierpnia 1997 roku, jest dostępny pod adresem *http://www.mgt30.com/evo/*.

[4] *http://www.mgt30.com/scrum/*.

[5] *http://www.mgt30.com/dsdm/*.

Crystal[6] (1997), **programowanie ekstremalne** (***Extreme Programming***; **XP**)[7] (1999), ***Feature Driven Development*** (**FDD**)[8] (1999), **programowanie pragmatyczne** (***Pragmatic Programming***)[9] (1999) i **programowanie adaptacyjne** (***Adaptative Software Development***)[10] (2000).

Kambryjska eksplozja metod, artykułów, książek i seminariów na temat lekkiego programowania podsunęła niektórym ekspertom pomysł zorganizowania spotkania z najważniejszymi postaciami tego ruchu. W 2001 roku zebrali się oni w jednym z kurortów narciarskich w stanie Utah. Zdecydowali o zastąpieniu słowa „lekki" słowem „zwinny", po czym narodził się manifest programowania zwinnego[11] (patrz rysunek 2.2).



MANIFEST PROGRAMOWANIA ZWINNEGO

ODKRYWAMY NOWE METODY PROGRAMOWANIA DZIĘKI PRAKTYCE
W PROGRAMOWANIU I WSPIERANIU W NIM INNYCH. W WYNIKU
NASZEJ PRACY ZACZĘLIŚMY BARDZIEJ CENIĆ:

LUDZI I INTERAKCJE OD PROCESÓW I NARZĘDZI
DZIAŁAJĄCE OPROGRAMOWANIE OD SZCZEGÓŁOWEJ DOKUMENTACJI
WSPÓŁPRACĘ Z KLIENTEM OD NEGOCJACJI UMÓW
REAGOWANIE NA ZMIANY OD REALIZACJI ZAŁOŻONEGO PLANU

OZNACZA TO, ŻE ELEMENTY WYPISANE PO PRAWEJ SĄ WARTOŚCIOWE,
ALE WIĘKSZĄ WARTOŚĆ MAJĄ DLA NAS TE, KTÓRE WYPISANO PO LEWEJ.

KENT BECK, MIKE BEEDLE, ARIE VAN BENNEKUM, ALISTAIR COCKBURN, WARD CUNNINGHAM, MARTIN FOWLER, JAMES GRENNING, JIM HIGHSMITH, ANDREW HUNT, RON JEFFRIES, JON KERN, BRIAN MARICK, ROBERT C. MARTIN, STEVE MELLOR, KEN SCHWABER, JEFF SUTHERLAND, DAVE THOMAS

Rysunek 2.2. *Manifest programowania zwinnego*

[6] *http://www.mgt30.com/crystal/*.

[7] *http://www.mgt30.com/xp/*.

[8] *http://www.mgt30.com/fdd/*.

[9] *http://www.mgt30.com/prag/*.

[10] *http://www.mgt30.com/asd/*.

[11] Manifest programowania zwinnego można znaleźć pod adresem *http://www.mgt30.com/manifesto/*.

Manifest programowania zwinnego był przez wielu postrzegany przede wszystkim jako reakcja przeciw biurokracji podejść formalnych, które wyraźnie były zbyt „uporządkowane". Jednak niewiele osób zdawało sobie sprawę, że manifest wypowiadał się także przeciw niezdyscyplinowanym programistom, „chaotycznym procesom" i niskiej jakości produktom, które dominowały po stronie *ad hoc* świata twórców oprogramowania. Przywódcy nowego ruchu rozumieli, że istnieje ścieżka wiodąca środkiem między strukturą a brakiem struktury, między porządkiem a chaosem. W pewnym sensie była to heroiczna próba powrotu do wczesnych dni pełnego pasji przecierania szlaków, jednak bez koszmarów, które tak często przybywały z anarchią.

Niektórzy guru zwinności sformowali następnie Agile Alliance[12], organizację typu non profit, której celem jest promowanie programowania zwinnego na całym świecie. Narodził się nowy ekosystem konferencji, konsultantów, książek i czasopism. W języku angielskim nazwę programowania, **Agile**, zaczęto pisać dużą literą „A", co znaczy, że jest ono czymś więcej niż tylko zbiorem praktyk dotyczących tworzenia oprogramowania. Dzięki odkryciu i uznaniu, że projekty programistyczne, podobnie jak istoty żywe, istnieją między uporządkowaniem a chaosem, programowanie zwinne stało się stylem życia.

Podstawy programowania zwinnego

Obecnie liczba **agilistów** (osób, które próbują postępować zgodnie z wartościami i regułami programowania zwinnego) sięga milionów. Badania potwierdzają, że większość światowych producentów oprogramowania praktykuje przynajmniej niektóre z „kluczowych praktyk zwinnych" [VersionOne 2009].

Podstawy programowania zwinnego opisywano już wiele razy, istnieje też wielu autorów, którzy potrafią je wyjaśnić lepiej ode mnie. Mimo to uważam, że w książce tej należy zamieścić krótki przegląd tych podstaw. Sam będąc agilistą, wolę postępować we własny sposób, w związku z czym podstawy programowania zwinnego opiszę za pomocą swoich „siedmiu wymiarów projektów programistycznych", zagadnienia, do którego powrócę w rozdziale 11., „Jak rozwijać kompetencje".

Ludzie

Przede wszystkim w programowaniu zwinnym uznano, że ludzie nie są możliwymi do zastąpienia zasobami, tylko niepowtarzalnymi jednostkami, a ich największa wartość nie leży w ich głowach, lecz w interakcjach i współpracy między nimi. Programowanie zwinne opowiada się za niewielkimi zespołami, w których różne role (programiści, projektanci, testerzy itd.) tworzą wielofunkcyjne jednostki, zebrane w miarę możliwości w jednym miejscu (w tym samym pokoju). Zespoły te następnie samoorganizują się, co znaczy, że nie jest im narzucana żadna metoda ani proces. Zakładając, że wiedzą, jak wykonać swoją pracę, ufa się, że zrobią to jak najlepiej i że przyjmą odpowiedzialność za uzyskane rezultaty.

[12] Agile Alliance ma swoją stronę internetową: *http://www.mgt30.com/agilealliance/*.

Funkcje

W programowaniu zwinnym zrozumiano, że najlepsze produkty powstają, gdy klienci bezpośrednio uczestniczą w pracach zespołu tworzącego dany produkt. Zespół współpracuje z klientem (albo przedstawicielem klienta) w celu utrzymywania i ciągłego aktualizowania priorytetów na stale zmieniającej się liście funkcji do zrealizowania. Funkcje te opisane są w zwięzły, „płytki" sposób, a dokładniejsze ich rozpoznawanie i dokumentowanie rozpoczyna się, jak tylko zespół wybierze je do bezzwłocznej implementacji. Kluczem do poprawnego zaprojektowania każdej funkcji jest prostota, a po zaimplementowaniu danego rozwiązania jego przydatność jest natychmiast weryfikowana przez klienta.

Jakość

W celu realizacji udanych produktów niezbędne jest skupienie się na jakości, dlatego najważniejszym elementem programowania zwinnego jest doskonałość techniczna. Jest ona osiągana za pomocą programowania sterowanego testami[13] (pisanie kodu testowego *przed* kodem produkcyjnym), definicji ukończenia (listy kontrolne), programowania iteracyjnego (dostosowywanie kodu do zmian i nowych przemyśleń) oraz refaktoryzacji (ulepszanie kodu, nawet jeśli nie zmieniono żadnych funkcji). Agiliści uznają potrzebę istnienia emergentnego projektu, co oznacza, że najlepsza architektura nie jest definiowana z góry (lub jest określana w podstawowym zakresie) i może wyłaniać się w miarę postępu projektowania produktu.

Narzędzia

Chociaż agiliści uważają, że narzędzia należą do najmniej ważnych elementów przyczyniających się do powstania udanego produktu, w literaturze zwinnej opisywane są i promowane liczne narzędzia. Doświadczone zespoły zwinne preferują narzędzia do dziennych kompilacji, ciągłej integracji oraz automatycznego testowania. Programowanie zwinne *wymaga* motywowania zespołów. Powtarzalne zadania są jednak nużące i nie motywują, w związku z czym powinny być automatyzowane. Wielu agilistów opowiada się także za przyjaznymi środowiskami pracy, takimi jak otwarte przestrzenie biurowe i narzędzia „promieniujące" informacje, jak na przykład duże tablice zadaniowe i wykresy spalania. W kontekście zwinnym narzędzia mają wzmacniać motywację, komunikację i współpracę w ramach zespołu.

Czas

Programowanie zwinne łączy z czasem specjalna relacja. W projektach zwinnych terminy przekazywania produktu, a także budżety, można wyznaczać niemal dowolnie. Oprogramowanie jest wytwarzane w krótkich etapach, często ograniczonych czasowo, albo w „sprintach" i oddawane w wielu przyrostowych wydaniach, przy czym każde z tych wydań stanowi potencjalnie gotowy produkt. Dzięki temu właściciele biznesu mogą sprawować kontrolę nad synchronizacją prac, przenosząc daty kolejnych wydań wstecz lub do przodu w zależności od tego, jakie funkcje i kiedy chcą udostępniać. W tym czasie zespoły dążą do zachowania stałego tempa pracy, przez co niemal w nieskończoność mogą utrzymywać niezmienny rytm produkcji.

[13] *http://www.mgt30.com/tdd/*.

Wartość

Jednym z głównych powodów, dla których ogłoszono manifest programowania zwinnego, była konieczność zareagowania na zmiany. Środowisko nigdy nie jest statyczne. Funkcje, które były cenne wczoraj, mogą być bezwartościowe jutro, włączając w to *także* funkcje przekazane już z powodzeniem użytkownikom. Agiliści próbują sprostać temu wyzwaniu, dbając o regularne pozyskiwanie opinii zwrotnych oraz stosując krótkie cykle wydawnicze. Celem częstego wydawania produktów jest nie tylko zachęcenie środowiska do dzielenia się swoim zdaniem i uwzględnianie zdobytych opinii w procesie produkcyjnym, ale też przekazywanie użytkownikom nowych i zaktualizowanych funkcji, gdy tylko zostanie wykryta potrzeba, a co za tym idzie, optymalizacja wartości biznesowej produktu.

Proces

Chociaż programowanie zwinne sugeruje przedkładanie człowieka nad proces, nie oznacza to, że proces jest nieważny. Nic bardziej mylnego. Niektóre z kluczowych procesów w kontekście programowania zwinnego to minimum planowania (lub „planowanie kroczące"), codzienna komunikacja twarzą w twarz (często w postaci zebrań na stojąco) oraz mierzenie postępu przez ocenianie funkcjonującego oprogramowania (funkcji przyjętych przez klienta). Agiliści uznają także potrzebę ciągłego ulepszania, zgodnie z którym procesy są regularnie oceniane i poprawiane na podstawie refleksji albo retrospektyw.

Konflikt

Takie są moim zdaniem fundamenty programowania zwinnego. Jest to rzecz jasna wyłącznie moja opinia. Niektórzy agiliści mogą nie zgadzać się z krótkimi opisami, które tu przedstawiłem. Taki brak zgody także jest częścią programowania zwinnego. „Konflikt" mógłbym nawet uznać za ósmy wymiar Agile. Jak sam zobaczysz później, konflikt wewnętrzny jest naturalnym aspektem systemów złożonych i warunkiem koniecznym kreatywności oraz innowacji. To wielki przywilej być wśród ludzi, których cieszy jeszcze coś innego niż tylko próby wzajemnego udoskonalania się.

Konkurencja programowania zwinnego

Istnieje zaledwie kilka gier pozbawionych współzawodnictwa i niewiele systemów bez konfliktów. Nasz świat nie byłby interesujący bez odmiennych poglądów. Na szczęście w świecie programowania zwinnego istnieje zdrowa konkurencja, jak na przykład Scrum kontra programowanie ekstremalne, Scrum kontra Kanban, a nawet Scrum[14] kontra Scrum[15]! Rozmaite metody Agile nie są jednak jedynymi graczami na rynku. Istnieje jeszcze kilku innych silnych i obiecujących zawodników, oferujących idee, które czasami są podobne, czasami komplementarne, a czasami diametralnie różne.

[14] Scrum Alliance ma swoją stronę internetową: *http://www.mgt30.com/scrumalliance/*.

[15] Organizacja Scrum.org została założona przez twórcę metodyki Scrum, Kena Schwabera: *http://www.mgt30.com/scrumorg/*.

Jednym z większych graczy jest **programowanie odchudzone**[16] (*lean software development*), które stanowi odpowiednik zarządzania odchudzonego w dziedzinie programowania. Siedem zasad zarządzania odchudzonego [Poppendieck 2009:193] bazuje na 14 zasadach drogi Toyoty[17] (filozofii zarządzania korporacji Toyota) i 14 zasadach zarządzania W. Edwardsa Deminga[18]. Światy programowania odchudzonego i zwinnego mają sporą część wspólną i to właśnie dlatego często grają one po tej samej stronie, mają tych samych ekspertów, te same grupy fanów i są omawiane na tych samych blogach, w tych samych magazynach i programach telewizyjnych. Programowanie odchudzone, ze swoim skupianiem się na redukcji marnotrawstwa i optymalizacją większej całości, wniosło ogromny wkład do świata programowania zwinnego z perspektywy zarządczej. Chociaż metody odchudzone dołączyły do ligi programistycznej kilka lat później niż metody zwinne, ruch z nimi związany nadrobił ten poślizg, dorabiając się własnych konferencji, konsultantów, szkoleniowców i konsorcjów[19].

Mniejszym, ale równie zdolnym graczem jest ruch **mistrzów sztuki programowania** (*Software Craftsmanship*), działający na podstawie manifestu mistrzów sztuki programowania[20] (patrz rysunek 2.3), który ma stanowić zarówno rozwinięcie, jak i wyzwanie dla oryginalnego manifestu programowania zwinnego. Zwolennicy sztuki programowania stoją na stanowisku, że programiści nie są inżynierami, lecz rzemieślnikami (niektóre osoby jako trafną metaforę przywołują model nauki rzemiosła w średniowiecznej Europie). Ruch sztuki programowania jest zręcznym i nieustraszonym nowym współgraczem, mającym własne (mniejsze) imprezy, książki i fora. Razem trójka ta stanowi zgrany zespół występujący w wadze lekkiej programowania, i to pomimo okazjonalnych bójek na pięści, które czasami wybuchają w szatniach.

Metody i platformy wagi ciężkiej również nie pozostawały bezczynne. Prawdopodobnie najsłynniejszym z graczy tej kategorii, a zarazem najbardziej kontrowersyjnym, jest **zintegrowany model dojrzałości organizacyjnej**[21] (*Capability Maturity Model Integration*; CMMI). Począwszy od roku 1987, jest on rozwijany i pielęgnowany przez Software Engineering Institute, centrum badawczo-rozwojowe z siedzibą na Uniwersytecie Carnegie Mellon. Model ten powstał jako opis ulepszania procesu inżynierii oprogramowania, ale rozrósł się do postaci bardziej abstrakcyjnej platformy, która oprócz programowania obejmuje obecnie także inne dziedziny. CMMI to podejście, którego celem jest służenie poradą w postaci opisu pięciu poziomów dojrzałości oraz 22 obszarów procesowych. CMMI wskazuje tylko, do *których* obszarów procesów możesz się odnieść w swoich staraniach o ulepszenie procesów. Z tego powodu niektórzy agiliści uważają, że CMMI, chociaż jego pełny opis liczy setki stron, nadal jest zgodne z programowaniem zwinnym, ponieważ metody zwinne uzupełniają CMMI, opisując, w jaki sposób ulepszać procesy. Agiliści nie byliby jednak agilistami, gdyby byli ze sobą zgodni. Niektórzy z nich twierdzą zatem, że ciężar CMMI, wbrew jego dobrym intencjom, spycha organizacje w kierunku biurokracji oraz ułomnych zespołów z dużymi ambicjami odnośnie do wyglądu i sprzętu, ale z niskimi wynikami osiąganymi w rzeczywistej rozgrywce.

---

[16] *http://www.mgt30.com/lean/*.

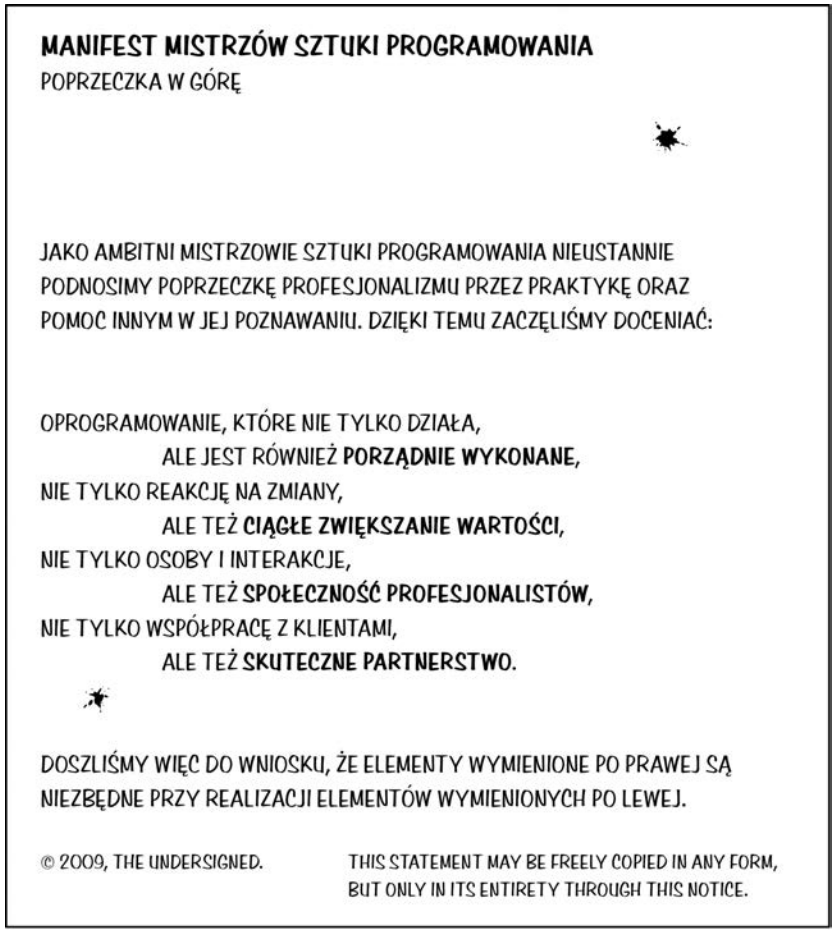
[17] *http://www.mgt30.com/toyota/*.

[18] *http://www.mgt30.com/deming/*.

[19] Lean Software and Systems Consortium można znaleźć pod adresem *http://www.mgt30.com/leanssc/*.

[20] Manifest mistrzów sztuki programowania można znaleźć pod adresem *http://www.mgt30.com/craftsmanship/*.

[21] *http://www.mgt30.com/cmmi/*.



MANIFEST MISTRZÓW SZTUKI PROGRAMOWANIA

POPRZECZKA W GÓRĘ

JAKO AMBITNI MISTRZOWIE SZTUKI PROGRAMOWANIA NIEUSTANNIE PODNOSIMY POPRZECZKĘ PROFESJONALIZMU PRZEZ PRAKTYKĘ ORAZ POMOC INNYM W JEJ POZNAWANIU. DZIĘKI TEMU ZACZĘLIŚMY DOCENIAĆ:

OPROGRAMOWANIE, KTÓRE NIE TYLKO DZIAŁA,
ALE JEST RÓWNIEŻ **PORZĄDNIE WYKONANE**,
NIE TYLKO REAKCJĘ NA ZMIANY,
ALE TEŻ **CIĄGŁE ZWIĘKSZANIE WARTOŚCI**,
NIE TYLKO OSOBY I INTERAKCJE,
ALE TEŻ **SPOŁECZNOŚĆ PROFESJONALISTÓW**,
NIE TYLKO WSPÓŁPRACĘ Z KLIENTAMI,
ALE TEŻ **SKUTECZNE PARTNERSTWO**.

DOSZLIŚMY WIĘC DO WNIOSKU, ŻE ELEMENTY WYMIENIONE PO PRAWEJ SĄ NIEZBĘDNE PRZY REALIZACJI ELEMENTÓW WYMIENIONYCH PO LEWEJ.

Rysunek 2.3. *Manifest mistrzów sztuki programowania*

Podobnie sprzeczne sygnały można było usłyszeć na temat zbioru ***Guide to Project Management Body of Knowledge*** (**PMBOK**)[22], publikowanego i aktualizowanego przez Project Management Institute. Co ciekawe, zbiór ten powstał jako opis najlepszych praktyk zarządzania projektami w ogólności. Od chwili pierwszego wydania w 1987 roku był jednak wielokrotnie aktualizowany i w odpowiedzi na sukcesy osiągane przez menedżerów projektów zwinnych sam stał się bardziej „zwinny". W przeciwieństwie do CMMI PMBOK wyraźnie sugeruje za pomocą licznych procesów, *jak* menedżerowie projektów mogą wykonywać swoją pracę. Chociaż sugerowane praktyki nie zawsze wpasowują się w reguły zarządzania zwinnego, wielu menedżerów projektów aktywnie stara się niwelować owe różnice. Należy nadmienić, że większość praktyk PMBOK zajmuje się innymi rewirami niż zarządzanie zwinne. To samo można powiedzieć o **PRINCE2**[23], podobnej metodzie zarządzania projektami, publikowanej i aktualizowanej przez Office of Government Commerce w Wielkiej Brytanii i praktykowanej przede wszystkim w Europie.

[22] *http://www.mgt30.com/pmbok/*.

[23] *http://www.mgt30.com/prince2/*.

Na koniec mamy ***Unified Process***[24] oraz jego lepiej znaną odmianę ***Rational Unified Process*** (**RUP**)[25], opracowaną w 1997 roku przez Rational Software (obecnie IBM). RUP jest dla deweloperów oprogramowania tym, czym PMBOK jest dla menedżerów projektów. Definiuje pokaźną strukturę procesów, którą można (a nawet trzeba) dostosować do określonych sytuacji, ale dokumentacja RUP jest przygotowana w taki sposób, że cała ta struktura często jest postrzegana jako biurokratyczna. Agiliści uważają, że proces powinien być rozwijany w ramach projektu, począwszy od absolutnego minimum kilku praktyk. W RUP przyjęto odwrotne podejście, definiując wiele praktyk i sugerując, że te, które są niepotrzebne, można wykreślić (podejście to często porównywałem do zakupu i stopniowego demontowania boeinga 747 w celu uzyskania roweru miejskiego; w przypadku wielu projektów mądrzejsze wydaje mi się po prostu sięgnięcie od razu po rower). Nie powinno dziwić, że w odpowiedzi na liczne sukcesy metod zwinnych zaproponowano kolejne, bardziej zwinne alternatywy wobec RUP, w tym ***Agile Unified Process*** (**AUP**)[26], ***Open Unified Process*** (**OpenUP**)[27] oraz ***Essential Unified Process*** (**EssUP**)[28]. Niemniej wydaje się, że żaden z tych graczy nie zagościł na dobre w światowej lidze metod zwinnych.

Przeszkoda w przyjęciu programowania zwinnego

Raz po raz empiryczne doświadczenia udowadniają, że programowanie zwinne, gdy jest dobrze realizowane, przynosi ogromne zwroty z inwestycji [Rico 2009]. Skoro jednak metody zwinne dają tak pozytywne efekty, to dlaczego nie korzystają z nich wszyscy? Dlaczego tak wiele projektów programistycznych na całym świecie kończy się niepowodzeniem[29]?

Raport *State of Agile Development Survey 2009*, przygotowany przez VersionOne, jako największe przeszkody w przyjęciu praktyk zwinnych wymienia: „kierownictwo przeciwne zmianie", „utratę kontroli przez kierownictwo". „zespół przeciwny zmianie" oraz „jakość zdolności inżynierskich", łącznie z wieloma „potrzebami" organizacji dotyczącymi planowania, przewidywalności i dokumentacji [VersionOne 2009].

Chileczkę... spójrzmy jeszcze raz na te przeszkody. Mówimy o kontroli kierowniczej, zarządzaniu zmianami organizacyjnymi i zdolnościach inżynierskich...

Wybaczcie mi, jeśli nie mam racji, ale... czy to wszystko nie leży... hmm... w zakresie odpowiedzialności *kierownictwa*? Czy nie oznacza to, że największą przeszkodą w programowaniu zwinnym na całym świecie są *menedżerowie*?

Jako menedżera wniosek taki wcale mnie nie cieszy.

Jako pisarza — tak.

[24] *http://www.mgt30.com/up/*.

[25] *http://www.mgt30.com/rup/*.

[26] *http://www.mgt30.com/aup/*.

[27] *http://www.mgt30.com/openup/*.

[28] *http://www.mgt30.com/essup/*.

[29] Notatka prasowa z raportu CHAOS Summary za rok 2009 jest dostępna pod adresem *http://www.mgt30.com/chaosreport/*.

Uważam, że w programowaniu zwinnym przeoczono wagę (bezpośredniego) zarządzania. Jeśli menedżerowie nie wiedzą, co robić i czego oczekiwać po organizacji zwinnej, jak możemy oczekiwać ich zaangażowania w przestawienie się na programowanie zwinne? Jaki komunikat przekazują nam w tej sytuacji metody zwinne? Jeśli brzmi on tylko: „Nie potrzebujemy menedżerów", nie ma się co dziwić, że przejście do metod zwinnych spotyka się na całym świecie z oporem.

Aby więc organizacje mogły korzystać z zalet przyjęcia metod zwinnych, powinny znać odpowiedź na ważne pytanie: jaka *jest* przyszłość menedżera w świecie zwinnym?

Zarządzanie bezpośrednie a zarządzanie projektem

Moje imię jest rzadko spotykane w moim kraju. Jakimś cudem miałem jednak do czynienia w swojej karierze zawodowej z kilkoma wcieleniami Jurgenów, Jurjenów i Jörgenów, co prowadziło do wielu nieporozumień. Kiedy imiona są do siebie podobne, ludzie mają skłonność do pomijania wszelkich pozostałych różnic. Założę się, że gdyby Ella Fitzgerald miała na imię Jurgen, moi koledzy prosiliby mnie, abym dla nich zaśpiewał.

Ten sam problem dostrzegam w przypadku osób, które są nazywane „menedżerami".

W 2005 roku kilkanaście osób specjalizujących się w pracy zarządczej (menedżerowie projektu, bezpośredni przełożeni itd.) spotkało się ze sobą i ogłosiło deklarację wzajemnej zależności (***Declaration of Interdependence***; **DOI**)[30] (patrz rysunek 2.4).

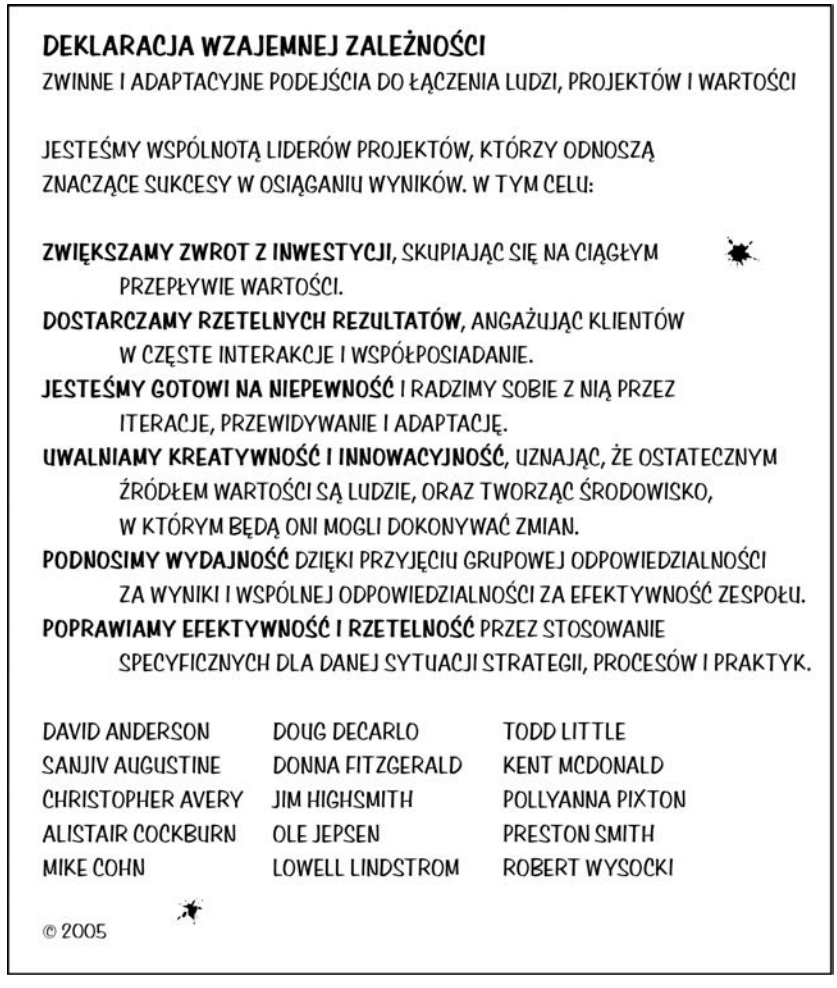
W swoim pierwszym wcieleniu deklaracja była skierowana przede wszystkim do menedżerów projektu. Później zdano sobie sprawę, że jej zasady można interpretować szerzej i odnosić je do „zarządzania w ogólności". Deklaracja ta dotyczy jednak głównie *zarządzania projektami programistycznymi*, a nie *zarządzania zespołami ludzkimi*. Należy to podkreślić, gdyż autorzy DOI założyli również organizację Agile *Project* Leadership Network[31].

Niestety, zarządzanie projektami i zarządzanie funkcjonalne (bądź bezpośrednie) są często mylone. W znakomitych książkach autorstwa czołowych ekspertów, w tym *Agile Management* [Anderson 2004], *Managing Agile Projects* [Augustine 2005] i *Agile Project Management* [Highsmith 2009], omówiono kwestie dotyczące zarówno zarządzania projektami, jak i zarządzania bezpośredniego. Podobną sytuację można zastać na wielu forach, blogach i w czasopismach. Wolałbym, żeby było inaczej, ponieważ zarządzanie projektami i zarządzanie bezpośrednie nie są tym samym. To jak mylenie programistów z administratorami systemu. Mogą oni wyznawać te same poglądy, opowiadać te same dowcipy, mieć takie same fryzury i te same ubrania (rzecz jasna w przenośni), ale *nie* należy ich traktować, jakby byli tymi samymi osobami (poważnie — spróbuj tylko poprosić jakiegokolwiek programistę, żeby naprawił Twój komputer... albo lepiej nie proś!).

Bez wyraźnego odróżnienia zarządzania bezpośredniego od zarządzania projektem utrudniamy zarówno bezpośrednim przełożonym, jak i menedżerom projektów zrozumienie ich ról w organizacji zwinnej. Na szczęście nie tylko ja jeden zdaję sobie z tego sprawę. Przed moją książką wydano kilka innych, w tym *Behind Closed Doors* [Rothman, Derby 2005] i *Leading Lean Software Development* [Poppendieck 2009], w których lepiej zarysowano zakres odpowiedzialności bezpośrednich przełożonych w organizacjach tworzących oprogramowanie.

[30] Deklaracja wzajemnej zależności jest dostępna pod adresem *http://www.mgt30.com/doi/*.

[31] *http://www.mgt30.com/apln/*.



Rysunek 2.4. *Deklaracja wzajemnej zależności*

W swojej książce oddzielam zarządzanie bezpośrednie od zarządzania projektami. Moim głównym celem jest pomaganie *bezpośrednim przełożonym* (w tym menedżerom zespołów deweloperskich i liderom zespołów) w zrozumieniu ról, jakie odgrywają w swoich organizacjach. Jestem jednak pewien, że menedżerowie projektów, menedżerowie systemów, menedżerowie serwisu, menedżerowie biur i menedżerowie ekspresów do kawy także stwierdzą, że moja książka jest ciekawa.

Natomiast tym z Was, którzy myśleli, że DJ Jurgen to ja, chciałbym powiedzieć... „przykro mi!".

Podsumowanie

Programowanie zwinne to podejście do tworzenia oprogramowania, które powstało w latach 90. XX wieku. Było odpowiedzią zarówno na metody biurokratyczne, jak i *ad-hoc*, które nie były w stanie z powodzeniem i konsekwentnie wytwarzać produktów programistycznych.

Programowanie zwinne, z wartościami i zasadami wyrażonymi w manifeście programowania zwinnego, skupiło się na ludziach i zespołach, częstym oddawaniu wysokiej jakości produktów, intensywnej współpracy z klientem i reagowaniu na zmiany, przy ograniczonym do minimum planowaniu z góry.

Wartości i zasady programowania zwinnego zostały zaimplementowane poprzez rozmaite metody zwinne, takie jak Scrum i programowanie ekstremalne. Żadna z metod zwinnych nie uwzględnia jednak roli bezpośredniego kierownictwa (którego nie należy mylić z kierownictwem projektu) w organizacjach zwinnych. Z tego powodu kierownictwo bezpośrednie często jest identyfikowane jako największa przeszkoda w przyjęciu praktyk zwinnych.

Refleksje i działania

Zobaczmy, czy uda Ci się zrealizować w swojej organizacji niektóre z idei przedstawionych w tym rozdziale.

- Przyjrzyj się siedmiu wymiarom projektów programistycznych (ludzie, funkcje, jakość, narzędzia, czas, wartość i proces). Czy w Twoich projektach programistycznych uwzględniono wszystkie te wymiary? Czy w każdym z tych wymiarów Twoje zespoły działają zwinnie? Co zamierzasz zrobić, jeśli tak nie jest?
- Pomyśl o menedżerach w swojej organizacji. Którzy z nich mogą przeszkadzać w przyjęciu programowania zwinnego? Czy jest coś, co mógłbyś w związku z tym zrobić? Upewnij się, że wiesz, czego od *nich* oczekujesz, aby *Twoje* podejście do zarządzania zwinnego odniosło sukces.
- Czy każdy wie, kto jest jego bezpośrednim przełożonym, a kto nie? Czy istnieje brak pewności albo zgody co do ról bezpośredniego przełożonego i menedżera projektu? Jeśli tak, co masz zamiar z tym zrobić?
- Rozwiń swoje umiejętności zarządzania zwinnego, subskrybując blogi i grupy o zwinnych zespołach i organizacjach. Aktualną ich listę możesz znaleźć na stronie Management 3.0, dostępnej pod adresem *http://www.management30.com/*.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ż

ZARZĄDZANIE 3.0
TO DROGA DO PRAWDZIWEGO SUKCESU!

Wykonanie produktu o dużej wartości rynkowej i osiągnięcie prawdziwego sukcesu często wymaga od menedżerów zmiany dotychczasowego podejścia. Współczesne firmy są połączonymi systemami, a samo zarządzanie dotyczy głównie ludzi i relacji. Podejście zwane zarządzaniem zwinnym czy też programowaniem zwinnym (*agile*) ma szczególne miejsce w nowoczesnej teorii systemów złożonych i procesach wytwarzania oprogramowania. Co istotne, wdrożenie metodologii zwinnych ułatwia realistyczne podejście do kierowania projektami czy doskonalenia zespołów i zarządzania nimi.

Książka adresowana jest przede wszystkim do kierowników zespołów, umożliwia dogłębne zrozumienie reguł rządzących pracą zespołu. Poruszono w niej takie tematy jak: teoria systemów złożonych, teoria gier, samoorganizacja i zasada ciemności. Zebrano i usystematyzowano znane od wielu lat klasyczne idee i techniki zarządzania, a następnie połączono je z ideą programowania zwinnego. Powstał w ten sposób spójny system idei, który powinien sobie przyswoić każdy adept zarządzania, mający pasję, ambicję i odznaczający się dążeniem do zarządzania doskonałego, wyzwalającego kreatywność zespołu i prowadzącego wprost do celu.

W książce omówiono:

- kluczowe cechy modelu zarządzania 3.0
- podstawy teorii systemów złożonych
- wpływ złożoności systemów na organizację
- utrzymywanie aktywności, kreatywności, innowacyjności i motywacji pracowników
- ideę kultury rzemiosła programistycznego
- ciągłe doskonalenie się w ramach organizacji
- ideę przywództwa ukierunkowanego na cel

Jurgen Appelo — pisarz, mówca, szkoleniowiec, programista, przedsiębiorca, menedżer, bloger, czytelnik, marzyciel, lider i wolnomyśliciel. Założył kilka firm i pełnił w nich funkcję lidera zespołu, menedżera lub kierownika. Był głównym specjalistą do spraw informatyki w ISM eCompany, jednym z największych dostawców rozwiązań e-biznesowych w Holandii. Jest niekwestionowanym autorytetem w dziedzinie metodyki agile i popularyzatorem modelu zarządzania 3.0.

Addison-Wesley
Pearson Education

Helion

42157 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-1801-4

Informatyka w najlepszym wydaniu

cena: 59,00 zł