O'REILLY®

Agile

Przewodnik po zwinnych metodykach programowania

POZNAJ NOWOCZESNE PODEJŚCIE DO WYTWARZANIA OPROGRAMOWANIA!

Helion

Andrew Stellman, Jennifer Greene

Tytuł oryginału: Learning Agile: Understanding Scrum, XP, Lean, and Kanban

Tłumaczenie: Tomasz Walczak (wstęp, rozdz. 2 – 10), Joanna Zatorska (rozdz. 1)

ISBN: 978-83-283-0940-1

© 2015 Helion S.A.

Authorized Polish translation of the English edition of Learning Agile, ISBN 9781449331924 © 2015 O'Reilly Media.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek 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.

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

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Przedmowa **11**

Wprowadzenie **13**

1. Poznaj Agile **15**

Czym jest Agile? 16
Kto powinien przeczytać tę książkę? 20
Cele do osiągnięcia 21
Wpakujemy Ci Agile do głowy wszelkimi możliwymi sposobami 21
Struktura książki 24

2. Wartości Agile **27**

Lider zespołu, architekt i kierownik projektu wchodzą do baru… 28
Uniwersalne rozwiązania nie istnieją 31
Podejście zwinne nas uratuje! Prawda? 33
Niespójna perspektywa 37
Manifest Agile pomaga zespołom zrozumieć cel stosowania poszczególnych technik 42
Jak zrozumieć słonia? 46
Od czego zacząć wprowadzanie nowej metodyki? 52

3. Zasady Agile **57**

Dwanaście zasad podejścia zwinnego 58
Klient ma zawsze rację, prawda? 58
Dostarczanie projektu 61
Komunikacja i współpraca 68
Przebieg projektu — posuwanie się do przodu 76
Nieustanne ulepszanie projektów i zespołu 80
Projekt w podejściu zwinnym — łączenie wszystkich zasad 83

Kup ksi k Pole ksi k

4. Scrum i samoorganizujące się zespoły 87
Zasady podejścia Scrum 89
Akt I. Ja móc Scrum? 91
W zespole stosującym podejście Scrum wszyscy są właścicielami projektu 93
Akt II. Aktualizacje stanu są dobre w sieciach społecznościowych! 106
Codzienne spotkania są dla całego zespołu 107
Akt III. Sprintem prosto w mur 115
Sprinty, plany i retrospekcje 116
Akt IV. Pies goniący samochód 124

5. Planowanie w Scrumie i wspólne zobowiązanie 131
Akt V. Nie do końca przygotowani na nieoczekiwane 132
Historie użytkowników, szybkość i ogólnie przyjęte praktyki w podejściu Scrum 134
Akt VI. Runda honorowa 149
Jeszcze o wartościach w Scrumie 150

6. XP i otwartość na zmiany 165
Akt I. Nadgodziny 166
Podstawowe techniki XP 167
Akt II. Zmieniliśmy strategię, ale ciągle przegrywamy 176
Wartości XP pomagają zespołom zmienić nastawienie 178
Budowanie właściwego nastawienia zaczyna się od wartości XP 183
Akt III. Zmiana sytuacji 187
Zrozumienie zasad XP pomaga otworzyć się na zmiany 188

7. Prostota i projektowanie przyrostowe w XP 203
Akt IV. Nadgodziny, część II — znów to samo 204
Kod i projekt 205
Decyzje związane z kodem i projektem podejmuj w ostatnim sensownym momencie 217
Projektowanie przyrostowe i holistyczne techniki XP 224
Akt V. Ostateczny wynik 235

8. Lean, unikanie marnotrawstwa i spojrzenie na całość 245
Myślenie odchudzone 246
Akt I. I jeszcze jedna sprawa… 254
Kreowanie herosów i myślenie magiczne 255
Eliminowanie marnotrawstwa 257
Lepsze zrozumienie produktu 262
Dostarczanie tak wcześnie, jak to możliwe 269

Kup ksi k Pole ksi k

9. **Kanban, przepływ i nieustanne doskonalenie ... 285**
Akt II. Ciągły wyścig 287
Zasady podejścia Kanban 288
Doskonalenie procesu za pomocą podejścia Kanban 294
Pomiar przepływu i zarządzanie nim 306
Zachowania emergentne w Kanbanie 322

10. **Coach metodyk zwinnych ... 333**
Akt III. I jeszcze jedna sprawa (znów?)… 334
Coachowie rozumieją, dlaczego ludzie nie zawsze chcą się zmieniać 335
Coachowie rozumieją proces uczenia się 339
Coach rozumie, dzięki czemu metodyka działa 343
Zasady coachingu 345

Skorowidz ... 348

Kup ksi k Pole ksi k

Kup ksi k Pole ksi k

ROZDZIAŁ 3.

Zasady Agile

Gdybym zapytał ludzi, czego oczekują, powiedzieliby, że szybszych koni.

— Henry Ford[1]

Nie istnieje jeden przepis pozwalający za każdym razem otrzymać doskonałe oprogramowanie. Zespoły stosujące podejście zwinne zdają sobie z tego sprawę. Dlatego posługują się ideami i podstawowymi regułami, które pomagają dokonywać właściwych wyborów i unikać problemów, a także radzić sobie z nieuniknionymi kłopotami, gdy się pojawią.

Poznałeś już cztery wartości z manifestu Agile. Oprócz nich istnieje dwanaście zasad, które każdy użytkownik podejścia zwinnego powinien stosować w czasie pracy w zespole programistów. Gdy siedemnastu pierwszych sygnatariuszy manifestu Agile spotkało się w Snowbird w stanie Utah, szybko wymyślili oni cztery wspomniane wartości. Więcej czasu zajęło im opracowanie dołączonych do manifestu dwunastu dodatkowych zasad. Jeden z sygnatariuszy, Alistair Cockburn, wspomina[2]:

> Nasza siedemnastoosobowa grupa szybko uzgodniła wartości. Opracowanie deklaracji z następnego poziomu okazało się zbyt poważnym zadaniem, abyśmy mogli ukończyć je w czasie zaplanowanym na spotkanie. Wartości wymienione w tym punkcie stanowią obecny zestaw roboczy.
>
> Przedstawione deklaracje będą ewoluować, gdy poznamy reakcje ludzi na nasze słowa i wymyślimy bardziej precyzyjne sformułowania. Będę zaskoczony, jeśli się okaże, że ta wersja nie stanie się przestarzała wkrótce po opublikowaniu tej książki. Najnowszą wersję znajdziesz w witrynie Agile Alliance (*http://www.agilealliance.org/*).

Alistair miał rację. Sformułowanie zasad ze wspomnianej witryny różni się nieco od wersji z tej książki. Jednak choć język się zmienia, idee i reguły pozostają niezmienne.

W tym rozdziale poznasz 12 zasad podejścia zwinnego. Dowiesz się, czym są, dlaczego są potrzebne i jak wpływają na projekty. Zobaczysz na praktycznym przykładzie, jak zastosować te zasady w rzeczywistym projekcie. Aby ułatwić naukę, przypisaliśmy reguły do czterech kategorii:

---

[1] Toczą się spory na temat tego, czy Henry Ford rzeczywiście to powiedział, jednak prawie wszyscy są zgodni co do tego, że to stwierdzenie z pewnością by mu się spodobało.

[2] Alistair Cockburn, *Agile Software Development: The Cooperative Game*, wydanie drugie (Boston: Addison Wesley, 2006).

Kup ksi k Pole ksi k

dostarczanie, komunikacja, wykonanie i rozwój. Te kategorie dotyczą kwestii powtarzających się w zasadach i ogólnie w podejściu zwinnym. Jednak choć taki podział pozwala na skuteczną naukę, pamiętaj, że każda reguła jest niezależna.

Dwanaście zasad podejścia zwinnego

1. Naszym priorytetem jest zapewnianie satysfakcji klientów dzięki szybkiemu i ciągłemu dostarczaniu wartościowego oprogramowania.
2. Jesteśmy otwarci na zmiany wymagań nawet na późnych etapach prac. Procesy zwinne pozwalają poradzić sobie ze zmianami w celu zapewnienia klientom przewagi konkurencyjnej.
3. Często dostarczamy działające oprogramowanie; zajmuje to od kilku tygodni do kilku miesięcy, przy czym preferowane są krótsze okresy.
4. Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji zespołowi programistów i komunikowania się w jego ramach jest bezpośrednia rozmowa.
5. Pracownicy biznesowi i programiści muszą codziennie wspólnie pracować nad projektem.
6. Opieraj projekty na zmotywowanych osobach. Zapewnij im potrzebne środowisko i wsparcie oraz uwierz, że wykonają zadanie.
7. Główną miarą postępów jest działające oprogramowanie.
8. W procesach zwinnych ważna jest możliwość utrzymania tempa programowania. Sponsorzy, programiści i użytkownicy powinni móc w nieskończoność utrzymywać stałe tempo pracy.
9. Ciągła troska o techniczną doskonałość i dobry projekt są zgodne z podejściem zwinnym.
10. Konieczna jest prostota rozumiana jako sztuka maksymalizowania niewykonywanej pracy.
11. Najlepsze architektury, wymagania i projekty są owocem pracy samoorganizujących się zespołów.
12. Zespół regularnie zastanawia się nad tym, jak zwiększyć swoją efektywność, a następnie odpowiednio usprawnia i dostosowuje swoje postępowanie[3].

Klient ma zawsze rację, prawda?

Wróć do początku rozdziału i ponownie przeczytaj cytat. Co Henry Ford chciał tak naprawdę powiedzieć? Mówi o dawaniu ludziom tego, czego potrzebują, a nie tego, o co proszą. Klient ma określone potrzeby. Jeśli budujesz oprogramowanie, które ma je zaspokajać, musisz je zrozumieć — i to niezależnie od tego, czy klient potrafi Ci je zakomunikować. Jak należy współpracować z klientem, który nie potrafi na początku projektu wytłumaczyć, że potrzebuje samochodu, a nie tylko szybszych koni?

Właśnie ta kwestia jest podstawą dwunastu zasad; chodzi o to, aby zespół mógł zbudować oprogramowanie, którego użytkownik naprawdę potrzebuje. Przedstawione reguły są oparte na idei budowania projektów w celu zapewnienia *wartości*. Jednak słowo „wartość” jest w tym kontekście kłopotliwe, ponieważ każdy może uznawać za wartościowe w oprogramowaniu coś innego. Różni ludzie mają odmienne potrzeby.

---

[3] Źródło: *http://agilemanifesto.org/principles.html* (wersja z czerwca 2014 roku).

Kup ksi k Pole ksi k

Możliwe, że trzymasz w ręku dobry przykład ilustrujący to zagadnienie. Jeśli czytasz tę książkę na przenośnym czytniku e-booków, używasz oprogramowania napisanego w celu wyświetlania e-booków w tym urządzeniu. Poświęć minutę na zastanowienie się nad wszystkimi interesariuszami (osobami, które mogą czegoś oczekiwać od danego projektu) takiego oprogramowania:

- Dla czytelnika ważne jest, aby oprogramowanie umożliwiało łatwe czytanie książek. Istotna jest możliwość przechodzenia między stronami, zaznaczania fragmentów, wykonywania notatek, wyszukiwania tekstu i zapamiętywania ostatniej czytanej strony.
- Dla autorów bardzo istotne jest to, by napisane słowa były poprawnie wyświetlane, wypunktowania były wyróżnione wcięciem (co ułatwia lekturę) oraz by możliwe było łatwe poruszanie się do przodu i wstecz po tekście i przypisach. Ważny jest też ogólny odbiór książki, żeby czytelnicy mogli czerpać przyjemność z czytania i czegoś się nauczyć.
- Dla redaktorów z wydawnictwa Helion ważna jest łatwość dystrybucji książki, a także — jeśli dana pozycja Ci się podoba — możliwość dodania pozytywnej recenzji i zakupu innych książek tego wydawnictwa.
- Dla księgarni lub sprzedawcy, od którego kupiłeś tę książkę, liczy się to, abyś mógł bardzo łatwo przeglądać i kupować dostępne pozycje oraz szybko i bez trudu pobierać je do czytnika.

Prawdopodobnie uda Ci się wskazać również innych interesariuszy i ważne dla nich kwestie. Każdy z wymienionych czynników reprezentuje zapewnianą interesariuszom wartość.

Pierwsze dostępne na rynku czytniki nie miały wszystkich wymienionych funkcji. Potrzeba było dużo czasu, aby oprogramowanie działające w czytnikach wyewoluowało do obecnej postaci. Ponadto jest prawie pewne, że to oprogramowanie będzie coraz lepsze, ponieważ pracujące nad nim zespoły odkryją nowe sposoby zapewniania wartości.

Łatwo jest dostrzec wartość, jaką dają czytniki, gdyż każdy jest mądry po fakcie. Znacznie trudniej jest zauważyć wartość na początku projektu. Przeprowadźmy krótki eksperyment mentalny, by się o tym przekonać. Jak mógłby wyglądać czytnik, gdyby opracować go w modelu kaskadowym?

„Rób tak, jak mówię, a nie tak, jak powiedziałem"

Wyobraź sobie, że pracujesz w zespole rozwijającym pierwszy przenośny czytnik e-booków. Grupa odpowiedzialna za sprzęt dostarczyła prototyp z portem USB, przez który można wczytywać e-booki, i z małą klawiaturą służącą do interakcji z urządzeniem. Ty wraz z zespołem masz przygotować oprogramowanie wyświetlające e-booki użytkownikom.

Niestety, firma ma długą historię tworzenia oprogramowania za pomocą wyjątkowo nieefektywnego modelu kaskadowego, w którym najpierw zajmuje się ważnymi wymaganiami. Dlatego kierownik projektu zaczyna od zwołania wielkiego spotkania z udziałem wszystkich osób, do których udało mu się dotrzeć. Cały zespół spędza kilka następnych tygodni w sali obrad, gdzie w różnym czasie pojawiają się: starsi menedżerowie z Twojej firmy, przedstawiciele zaprzyjaźnionego wydawnictwa, które chce publikować e-booki obsługiwane przez czytnik, starszy sprzedawca z księgarni internetowej chcącej sprzedawać te książki, a także inni interesariusze, których kierownikowi projektu udało się znaleźć i nakłonić do udziału w spotkaniu.

Kup ksi k Pole ksi k

Po wielu dniach intensywnych spotkań i gorących dyskusji analitycy biznesowi zdołali opracować rozbudowaną specyfikację z wymaganiami wszystkich interesariuszy, z którymi udało się porozmawiać. Wymagało to dużo pracy, jednak w efekcie powstała specyfikacja, która wszystkim się podoba. Opisane są w niej rozbudowane funkcje dla użytkowników, dzięki którym będzie to najbardziej zaawansowane oprogramowanie dla przenośnych czytników. Uwzględnione są też: funkcje rejestrowania statystyk marketingowych przeznaczone dla wydawnictwa, sklep internetowy umożliwiający łatwy zakup książek, a nawet dostosowane do autorów innowacyjne mechanizmy podglądu i edycji książek w trakcie ich pisania, usprawniające proces publikacji. Zapowiada się naprawdę rewolucyjne oprogramowanie. Razem z zespołem przystępujecie do szacowania czasu potrzebnego do realizacji projektu i ustalacie, że zajmie to piętnaście miesięcy. Wydaje się, że to długo, jednak wszyscy są podekscytowani i masz pewność, że zdołacie dostarczyć oprogramowanie na czas.

Mija półtora roku. Zespół pracował bardzo ciężko. Programiści siedzieli nad kodem wieczorami i w weekendy, co negatywnie odbiło się na jakości niejednego małżeństwa. W projekt włożono ogrom pracy, ale produkt jest gotowy i udało się go dostarczyć dokładnie według planu, niemal co do dnia (tak, wiemy, że to prawie niemożliwe, lecz na potrzeby tego eksperymentu odrzuć wątpliwości i wyobraź sobie, że ten jeden raz ta sztuka się udała). Kod dla każdego wymagania ze specyfikacji został zaimplementowany, przetestowany i uznany za zakończony. Zespół jest bardzo dumny, a wszyscy interesariusze, którzy widzieli oprogramowanie, zgadzają się, że dostali dokładnie to, o co prosili.

Produkt trafia na rynek, jednak okazuje się niewypałem. Nikt go nie kupuje, a interesariusze nie są zadowoleni. Co się stało?

Okazuje się, że oprogramowanie potrzebne półtora roku wcześniej obecnie nie jest już przydatne. W czasie trwania prac nad projektem w branży pojawił się nowy standardowy format e-booków. Ponieważ nie uwzględniono go w specyfikacji, nie jest on obsługiwany. Żaden ze sprzedawców internetowych nie chce publikować książek w niestandardowym formacie używanym w czytniku. Ponadto choć zespół zbudował świetny sklep internetowy, jest on znacznie mniej zaawansowany niż sklepy obecnie wykorzystywane przez sprzedawców, dlatego nie jest dla nich atrakcyjny. Poza tym opracowana dużym nakładem pracy specjalna funkcja podglądu dla autorów okazuje się mniej przydatna niż oferowana przez konkurencję możliwość przesyłania e-mailem dokumentów Worda bezpośrednio do czytnika i ich wyświetlania.

Co za porażka! Początkowa specyfikacja była bardzo wartościowa dla wszystkich klientów — zarówno wewnętrznych, jak i zewnętrznych. Jednak obecnie oprogramowanie, które zespół półtora roku temu zgodził się zbudować, jest znacznie mniej wartościowe. Niektóre z potrzebnych zmian można było dostrzec na wczesnym etapie prac nad projektem, ale inne na początku nie były oczywiste. Zespół musiałby bardzo szybko i często zmieniać kierunek prac, aby móc uwzględnić modyfikacje. Model kaskadowy z początkowym określaniem rozbudowanych wymagań nie daje zespołowi swobody niezbędnej do reagowania na zmiany.

Jak więc znaleźć lepszy sposób na zaspokojenie potrzeb interesariuszy i klientów w ramach projektu, który pozwala dostarczyć działające oprogramowanie?

Kup ksi k Pole ksi k

Dostarczanie projektu

Zespoły stosujące podejście zwinne wiedzą, że najważniejsze jest dostarczenie działającego oprogramowania klientom. Z rozdziału 2. wiesz już, jak osiągają ten cel — dzięki pracy zespołowej, komunikacji z klientami i reagowaniu na zmiany. Jak przekłada się to na codzienną pracę zespołu?

Jeśli za priorytet uznasz częste dostarczanie wartości, to dzięki traktowaniu każdej zmiany jako czegoś korzystnego dla projektu i częstemu udostępnianiu oprogramowania zespół może pracować wspólnie z klientami nad ciągłym wprowadzaniem poprawek. Oprogramowanie, które zespół utworzy, może nie być różne od pierwotnie planowanego, ale *to dobrze*, ponieważ ostatecznie powstanie oprogramowanie potrzebne klientom.

Zasada numer 1. Naszym priorytetem jest zapewnianie satysfakcji klientów dzięki szybkiemu i ciągłemu dostarczaniu wartościowego oprogramowania

Pierwsza zasada porusza trzy różne i ważne kwestie: szybkie udostępnianie oprogramowania, ciągłe dostarczanie wartości i zapewnianie satysfakcji klientów. Aby w pełni pojąć istotę tej zasady, trzeba zrozumieć zależności między tymi aspektami.

Zespoły projektowe funkcjonują w rzeczywistym świecie, gdzie sprawy nigdy nie toczą się idealnie. Nawet zespół, który wykona świetną pracę przy zbieraniu i spisywaniu wymagań, z pewnością pominie niektóre kwestie, ponieważ dla żadnego systemu nie da się bezbłędnie wykryć wszystkich wymagań. Nie chcemy przez to powiedzieć, że nie należy się starać w tym obszarze. Metodyki zwinne są oparte na dobrych praktykach przekazywania i spisywania wymagań. Trzeba jednak pamiętać, że dopóki klienci nie zaczną używać działającego oprogramowania, trudno im będzie dokładnie wyobrazić sobie, *jak* będzie ono pracować.

Dlatego skoro klient może przekazać rzetelne informacje zwrotne dopiero po zobaczeniu działającego oprogramowania, najlepiej jest zapewnić sobie te informacje dzięki **szybkiemu dostarczaniu** produktu. Udostępnij pierwszą pracującą wersję oprogramowania tak szybko, jak to możliwe. Nawet jeśli pokażesz tylko jedną działającą funkcję, z której klient będzie mógł korzystać, i tak warto to zrobić. Jest to korzystne dla zespołu, ponieważ klient może przekazać rzetelne informacje, które pomogą nadać właściwy kierunek pracom nad projektem. Jest to dobre także dla klientów, gdyż dzięki działającemu oprogramowaniu mogą zrobić coś, co wcześniej było niemożliwe. Ponieważ oprogramowanie funkcjonuje i klienci mogą je stosować do wykonywania potrzebnych zadań, zespół zapewnił prawdziwą wartość. Możliwe, że jest ona niewielka, ale to i tak dużo więcej w porównaniu z niedostarczaniem żadnej wartości — zwłaszcza jeśli alternatywą jest coraz bardziej sfrustrowany użytkownik, który musi długo czekać na otrzymanie oprogramowania od zespołu.

Wadą szybkiego dostarczania jest to, że oprogramowanie udostępniane początkowo klientom jest bardzo niekompletne. Użytkownikom i interesariuszom czasem *naprawdę* trudno jest się z tym pogodzić. Choć niektórym łatwo jest zaakceptować wczesne wersje oprogramowania, inni czują się z nimi niekomfortowo. Wiele osób przeżywa trudne chwile, gdy ma korzystać z oprogramowania, które jest dalekie od doskonałości. W praktyce w wielu firmach (szczególnie w dużych organizacjach, w których ludzie przez lata współpracują z programistami) zespół pracujący nad oprogramowaniem musi dosłownie negocjować warunki dostarczania produktu interesariuszom.

Kup ksi k Pole ksi k

Jeśli współpraca między zespołem programistów a przyszłymi użytkownikami oprogramowania nie układa się dobrze, użytkownicy i interesariusze po otrzymaniu niekompletnego produktu mogą ocenić go bardzo surowo, a nawet wpaść w panikę, jeżeli zabraknie w nim oczekiwanej funkcji.

Podstawowe wartości podejścia zwinnego zapewniają rozwiązanie tego problemu. Jest nim współpraca z klientem przy negocjowaniu kontraktu. Zespół związany niezmienną specyfikacją i nieelastycznymi biurokratycznymi barierami nie ma możliwości wprowadzania poprawek w projekcie oprogramowania. Grupa pracująca w takich warunkach musi uruchomić nowy proces zarządzania zmianą, co wymaga nowych negocjacji kontraktu z klientem. Natomiast zespół ściśle współpracujący z klientami może na bieżąco wprowadzać wszystkie niezbędne zmiany. Na tym polega **ciągłe dostarczanie** oprogramowania.

Z tego powodu metodyki zwinne zwykle są oparte na iteracjach. Zespoły stosujące podejście zwinne w ramach planowania iteracji wybierają funkcje i wymagania, które zapewnią największą wartość. Jedyny sposób na ustalenie tych funkcji polega na współpracy z klientem i uwzględnianiu informacji zwrotnych uzyskanych w poprzedniej iteracji. Dzięki temu zespół możesz szybko zaspokoić potrzeby klienta i zademonstrować wartość oprogramowania. W dłuższej perspektywie pozwala to dostarczyć gotowy produkt zapewniający możliwie dużą wartość.

Zasada numer 2. Jesteśmy otwarci na zmiany wymagań nawet na późnych etapach prac. Procesy zwinne pozwalają poradzić sobie ze zmianami w celu zapewnienia klientom przewagi konkurencyjnej

Wielu odnoszących sukcesy praktyków podejścia zwinnego ma początkowo duże trudności z akceptacją tej zasady. Łatwo jest teoretycznie mówić o otwartości na zmiany. Jednak gdy w trakcie gorączkowych prac nad projektem zespół natrafi na zmianę, która wymaga dużo wysiłku, sytuacja może się stać napięta. Dotyczy to przede wszystkim tych programistów, którzy wiedzą, że przełożony będzie wymagał od nich dotrzymania wcześniejszych terminów niezależnie od ilości pracy koniecznej do uwzględnienia zmian. Poradzenie sobie z tym problemem — zwłaszcza w kulturze, w której ludzi obwinia się o opóźnienia — bywa trudne. Może jednak przynieść też wiele satysfakcji, ponieważ otwartość na zmiany wymagań to jedno z najwartościowszych narzędzi w przyborniku zwinnych zespołów.

Dlaczego zmiany w projekcie budzą tyle emocji? Zrozumienie tej kwestii jest kluczem do omawianej zasady. Przypomnij sobie sytuację, w której pracowałeś nad projektem i odkryłeś, że musisz zmienić rozwijane rozwiązanie. Jak się czułeś? Wcześniej uważałeś, że prace nad projektem przebiegają sprawnie. Prawdopodobnie podjąłeś już liczne decyzje: jak podzielić pracę, co napisać, co dostarczyć klientom. Nagle ktoś spoza projektu mówi, że część planów i pracy jest niepotrzebna — że *Ty* zrobiłeś coś źle.

Bardzo trudno jest zaakceptować czyjeś stwierdzenie, że zrobiłeś coś źle, szczególnie jeśli wykonujesz zadanie dla tej właśnie osoby. Dla większości inżynierów oprogramowania ważna jest *duma z dobrze wykonanej pracy*. Chcemy dostarczać produkty, pod którymi możemy się podpisać i które zaspokajają potrzeby użytkowników. Zmiana w projekcie jest zagrożeniem, ponieważ oznacza zakwestionowanie wybranej drogi i poczynionych założeń.

Kup ksi k Pole ksi k

Bardzo często ktoś prosi Cię o zmianę kierunku prac po wcześniejszym bezpośrednim wskazaniu obecnego kierunku. Jeżeli dana osoba poprosiła Cię o utworzenie danej funkcji, a Ty przystąpiłeś do pracy i ukończyłeś już połowę zadania, bardzo frustrujące są słowa: „W zasadzie teraz, kiedy o tym myślę, uważam, że powinniśmy zbudować coś zupełnie innego". Możesz odebrać to jako brak szacunku dla Twojej pracy. Teraz musisz się cofnąć i pozmieniać kod, który uważałeś za skończony. Trudno jest *nie* przyjąć wtedy postawy defensywnej. Jeszcze gorsze jest, jeśli to Ciebie obwinia się za to, że nie potrafisz czytać klientowi w myślach i spowodowałeś przekroczenie terminu.

Prawie każdy programista przynajmniej raz znalazł się w takiej sytuacji. Jak w świetle tego można zachować otwartość na zmiany w wymaganiach?

Pierwszy krok w kierunku otwierania się na zmiany w wymaganiach polega na próbie spojrzenia na świat z perspektywy klienta. Nie zawsze jest to łatwe, ale jest za to pouczające. Czy uważasz, że klient, który pierwotnie zasugerował Ci zły kierunek, zrobił to celowo? Jak sądzisz — co sobie pomyślał, gdy odkrył, że kilka miesięcy temu poprosił Cię o przygotowanie niewłaściwego oprogramowania, i gdy uświadomił sobie, że z tego powodu zmarnowałeś mnóstwo czasu? Dla niego kontakt z Tobą i poproszenie o zmiany oznacza przyznanie się do pomyłki, która będzie kosztować Cię dużo pracy. Nie jest to łatwe. Dlatego nie jest niczym dziwnym, że klienci często długo odwlekają zakomunikowanie zespołowi potrzeby wprowadzenia zmian. Jest to krępujące zadanie i klienci wiedzą, że przynoszą złe wieści. Prawdopodobnie nie zdołasz dotrzymać terminu, podobnie jak sam klient. Jeśli użytkownik ma potrzeby i firma wydaje pieniądze na zbudowanie oprogramowania, które ma je zaspokajać, to gdy produkt nie jest dostosowany do tych potrzeb, nie zapewnia wartości. Winny jest wtedy klient, ponieważ na początku projektu przekazał Ci niewłaściwe informacje.

Oznacza to, że dwie osoby muszą dokonać niemożliwego. Od Ciebie oczekuje się, że będziesz czytał w myślach klienta. Z kolei on ma umieć przewidywać przyszłość. Gdy spojrzysz na sytuację w ten sposób, łatwiej będzie Ci zachować otwartość na zmiany. Jeśli jednak wolisz unikać zmian i uparcie trzymać się planu przygotowanego na początku prac nad projektem, rozwiązanie jest proste — przyjmuj do zespołu wyłącznie telepatów i jasnowidzów.

Czym jest otwartość na zmiany? Oznacza ona, że:

- Nikt nie robi problemów, gdy potrzebna jest zmiana. Członkowie zespołu (i ich przełożeni) mają świadomość, że są tylko ludźmi i są omylni. Wiedzą też, że dla firmy lepsze jest pozwolenie na popełnianie błędów i częste ich naprawianie niż oczekiwanie doskonałości już przy pierwszym podejściu.
- Wszyscy płyniemy na jednej łodzi. Każdy członek zespołu, włącznie ze współpracującymi z nim klientami, jest odpowiedzialny za wymagania i wprowadzane w nich zmiany. Jeśli wymagania są błędne, winę za to ponoszą zarówno zespół, jak i klienci. Nie ma więc sensu obwiniać się wzajemnie o zmiany.
- Nie odwlekamy zmian do czasu, gdy jest już za późno. To prawda, popełnienie błędu jest krępujące. Jednak ponieważ wszyscy mają tego świadomość, warto starać się jak najszybciej rozwiązać problem. Pozwala to ograniczyć zakres szkód.

Kup ksi k Pole ksi k

- Nie traktujemy zmian jak błędów. Dokonaliśmy najlepszych wyborów, jakie były możliwe w świetle dostępnych wcześniej informacji. To, że te rozwiązania okazały się niewłaściwe, można było odkryć tylko dzięki temu, iż podjęte decyzje pomogły dostrzec potrzebę wprowadzenia dokonywanych teraz zmian.
- Wyciągamy wnioski ze zmian. Jest to najskuteczniejszy sposób rozwoju zespołu i poprawy umiejętności w obszarze wspólnego tworzenia oprogramowania.

Zasada numer 3. Często dostarczamy działające oprogramowanie; zajmuje to od kilku tygodni do kilku miesięcy, przy czym preferowane są krótsze okresy

Może uważasz, że zachowanie otwartości na zmiany w wymaganiach to ciekawy pomysł, który pomoże Ci w pracy nad projektami. Możesz jednak sądzić, że to bezsensowne podejście. Nie jesteś w tym odosobniony. Wielu członków zespołów programistycznych (zwłaszcza konserwatywni kierownicy projektów) początkowo ma wątpliwości co do otwartości na zmiany. Kierownicy każdego dnia muszą radzić sobie ze zmianami, a nastawienie do zmian w podejściu zwinnym jest bardzo odmienne od tradycyjnego ich traktowania. Praktycy metod zwinnych opisują tradycyjne nastawienie do zmian jako podejście **„dowódź i kontroluj**".

Pojęcie „dowódź i kontroluj" jest zapożyczone ze słownictwa militarnego. W książce *Beautiful Teams* z 2010 roku przeprowadziliśmy wywiad z głównym inżynierem z firmy Norhtrop Grumman, Neilem Siegelem, który przedstawił definicję tego określenia:

> Andrew: Nie znam się na systemach militarnych — czym jest system dowodzenia i kontroli?
>
> Neil: Jest to system informacyjny dla dowódców wojskowych. Umożliwia im komunikowanie się między sobą i zachowanie orientacji w sytuacji. Pozwala ustalić, gdzie znajdują się jednostki i jaki jest ich status. W ten sposób dowódca określa, co się dzieje. Kiedyś pola bitwy były niewielkie. Dowódca mógł stać na wzgórzu z lornetką i obserwować sytuację. Jednak od około 1900 roku pola bitwy stały się na tyle duże, że nie wystarczyło zająć pozycji na wzgórzu, jak niegdyś robił to Napoleon. Do śledzenia całego pola bitwy potrzebne zaczęły być technologie. Właśnie do tego używane są systemy dowodzenia i kontroli.

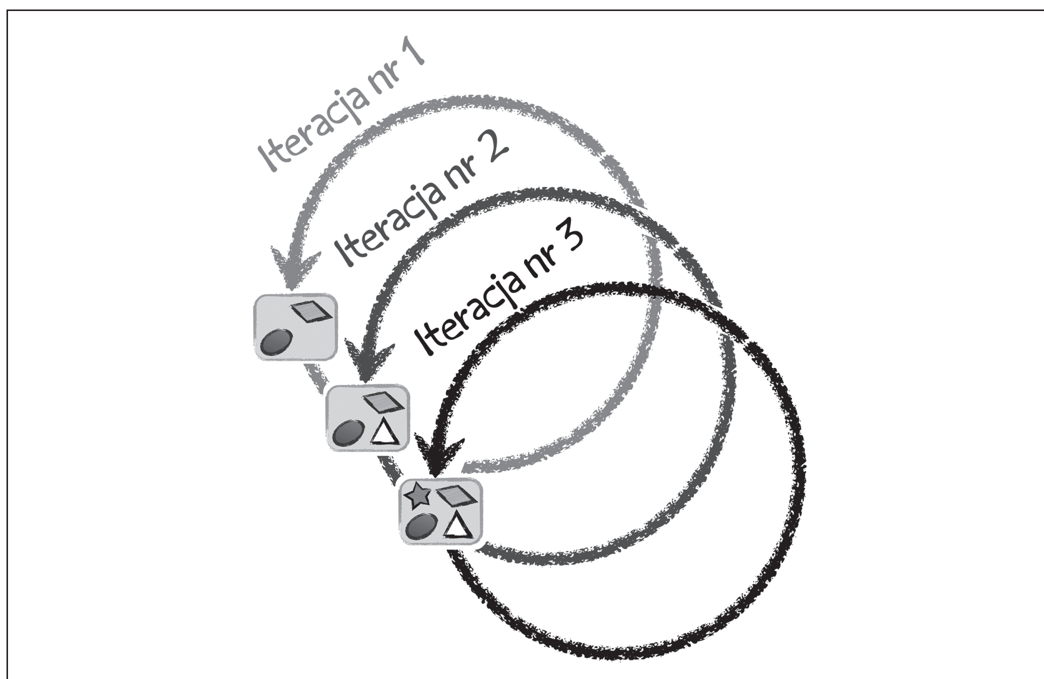
Zarządzanie projektem w modelu „dowódź i kontroluj" przypomina dowodzenie i kontrolę w wojsku.

- Termin „dowodzenie" określa sposób przypisywania zadań zespołowi przez kierownika projektu. Możliwe, że zespół nie odpowiada bezpośrednio przed daną osobą, ale może ona kontrolować przydzielanie prac. Ta osoba dzieli projekt na porcje, opracowuje harmonogram i przypisuje zadania członkom zespołu.
- Słowo „kontrolowanie" dotyczy sposobu zarządzania zmianami. Zmiany w projektach są nieuniknione. Praca zajmuje więcej czasu, niż oczekiwano, ludzie biorą chorobowe lub odchodzą z firmy, sprzęt jest niedostępny albo zepsuty — może wydarzyć się wiele nieoczekiwanych rzeczy. Kierownicy projektu nieustannie śledzą takie sytuacje, a w ramach kontroli projektu po ich wykryciu aktualizują plany, aby uwzględnić zmiany w harmonogramie i dokumentacji, przydzielają zespołowi nowe zadania i zarządzają oczekiwaniami interesariuszy, żeby wszyscy wiedzieli, co się dzieje.

Kup ksi k Pole ksi k

Przyczyną początkowego braku otwartości na zmiany u tradycyjnych kierowników projektu jest to, że kierownik wie, iż opisane problemy będą pojawiać się także w projektach prowadzonych metodami zwinnymi, a zespół będzie musiał sobie z nimi poradzić. Zaakceptowanie zmian i otwartość na nie wydają się prostą drogą do wywołania chaosu w projekcie. Jeśli w grupach stosujących podejście zwinne nie jest stosowana metoda „dowódź i kontroluj", jak mogą one jednocześnie radzić sobie ze zmianami i z codziennymi problemami typowymi dla zespołów projektowych?

Aby zachować otwartość na zmiany i uniknąć chaosu, trzeba często dostarczać działające oprogramowanie. Zespół wykorzystuje iteracje do podziału projektu na równe okresy. W ramach pokazanych na rysunku 3.1 iteracji programiści tworzą działające oprogramowanie. Na koniec każdej iteracji zespół przeprowadza pokaz, by przedstawić klientom, co zbudował, i retrospektywnie wyciąga wnioski z danego okresu. Później następuje sesja planowania, w trakcie której określany jest zakres prac z następnej iteracji. Przewidywalny harmonogram i stałe punkty kontrolne pomagają zespołowi szybko wykryć potrzebne zmiany, a także zapewniają wszystkim wolne od przypisywania winy środowisko, w którym można przedyskutować każdą zmianę i opracować strategię pozwalającą uwzględnić poprawki w projekcie.



Rysunek 3.1. Zespół posługuje się iteracjami, aby często udostępniać oprogramowanie, i w każdej nowej wersji wprowadza nowe funkcje

W tym kontekście podejście zwinne staje się bardzo atrakcyjne dla tradycyjnych kierowników projektu stosujących nastawienie „dowódź i kontroluj". Taki kierownik chce kontroli nad terminami. Iteracje ze ściśle ustalonymi ramami czasowymi dają taką kontrolę. Iteracje rozwiązują też jeden z największych problemów kierowników — radzenie sobie ze zmianami pojawiającymi się na końcowych etapach prac. Jednym z najtrudniejszych zadań tradycyjnego kierownika projektu

Kup ksi k Pole ksi k

jest monitorowanie zmian. Codzienne przeglądy i retrospektywne analizy iteracji pozwalają kierownikowi wykorzystać cały zespół do wczesnego wykrywania potrzebnych poprawek przed wystąpieniem problemów w projekcie.

Rola kierownika projektu przestaje być związana z dowodzeniem i kontrolą. Wcześniej kierownik przedstawiał zespołowi dzienny plan bitwy i nieustannie go dostosowywał, aby zachować właściwy kierunek prac. Obecnie kierownik współpracuje z zespołem i dba o to, by każdy jego członek cały czas pamiętał o ogólnej perspektywie i dążył do tych samych celów. Łatwiej jest to robić, gdy zespół pracuje w ramach krótkich iteracji prowadzących do dostarczenia działającego oprogramowania. W ten sposób każdy ma konkretne cele i dużo lepiej wie, co ma robić. Ponadto każda osoba ma poczucie, że odpowiada nie tylko za własne zadania, ale też za to, co cały zespół dostarczy na koniec iteracji.

Lepszy sposób dostarczania oprogramowania do czytnika e-booków

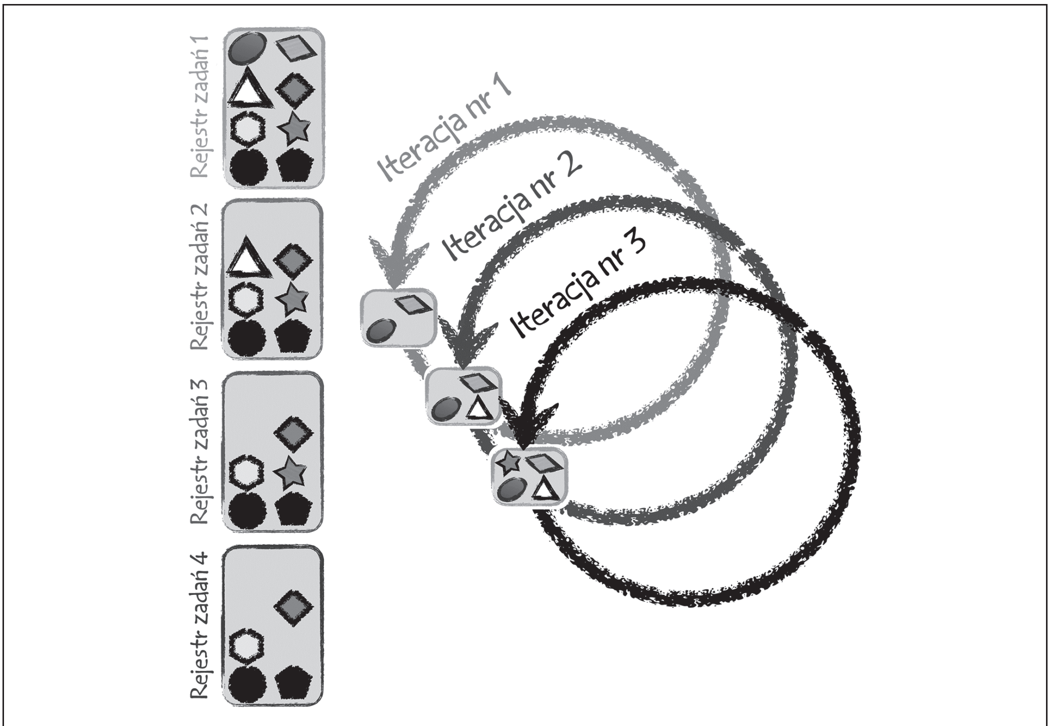
W jaki sposób opisane zasady mogą pomóc w przypadku nieudanego projektu oprogramowania do czytnika e-booków? Wróćmy do problemów, na jakie natrafił zespół. Produkt okazał się porażką, ponieważ nie posiadał ważnych funkcji oferowanych przez konkurencję (obsługi standardowego formatu e-booków i możliwości przesyłania e-mailem dokumentów do urządzenia). Poza tym udostępniał rozwiązania niedostosowane do rynku (sklep internetowy).

Zacznijmy projekt od początku. Tym razem przyjmijmy, że kierownik projektu współpracuje z interesariuszami, a zespół wykonuje jednomiesięczne sprinty. Projekt przebiega więc zupełnie inaczej:

- Po trzecim sprincie jeden z programistów informuje, że zatwierdzony został nowy standardowy format e-booków. Zespół decyduje się zaimplementować w czwartym sprincie bibliotekę obsługującą ten format. W piątym sprincie biblioteka ma zostać wbudowana w interfejs użytkownika czytnika.
- Po dziesięciu miesiącach prac powstaje działające oprogramowanie, które można załadować do prototypowych urządzeń i udostępnić użytkownikom wersji beta. Kierownik projektu rozmawia z tymi użytkownikami i odkrywa, że woleliby mieć możliwość przesyłania do czytników dokumentów Worda i artykułów prasowych. Zespół w ramach następnego sprintu dodaje do czytnika integrację z pocztą elektroniczną, dzięki czemu użytkownicy mogą przesyłać e-mailem artykuły do urządzeń.
- Po roku interesariusze informują zespół, że sklep internetowy jest niepotrzebny, bo wszyscy sprzedawcy korzystają ze standardowego formatu e-booków. Na szczęście sklep nie był traktowany priorytetowo i poświęcono mu bardzo niewiele czasu, gdyż w trakcie sprintu zajmowano się ważniejszymi funkcjami.

Ponieważ zespół po każdym sprincie udostępniał działające oprogramowanie, usunięcie funkcji z kolejki zadań pozwoliło dostarczyć produkt przed czasem! Wydawnictwo było gotowe do sprzedaży książek, bo wszyscy pracujący w nim starsi menedżerowie otrzymali wczesną wersję oprogramowania i prototypowe urządzenia do wypróbowania. Dzięki temu wydawnictwo było zaangażowane w projekt i miało motywację do udostępnienia książek bezpośrednio po pojawieniu się pierwszej wersji produktu. Nowy model pracy przedstawia rysunek 3.2.

Kup ksi k Pole ksi k



Rysunek 3.2. Na początku każdej iteracji zespół wybiera z rejestru zadań funkcje, nad którymi będzie pracował

Dzięki ciągłemu udostępnianiu, otwartości na zmiany i dostarczaniu po każdej iteracji działającego oprogramowania w projekcie czytnika e-booków udało się przed czasem ukończyć udany produkt. W odróżnieniu od zespołu korzystającego z modelu kaskadowego, w którym programiści po zatwierdzeniu wymagań zostali odcięci od klientów, zespół w podejściu zwinnym pozostawał w stałym kontakcie z użytkownikami. Dzięki temu mógł reagować na zmiany sytuacji i zbudować lepszy produkt.

Jednak w zespole pracującym nad czytnikiem e-booków nie wszystko funkcjonuje perfekcyjnie. Programiści dostarczają działające oprogramowanie w ramach iteracji, ale są przytłoczeni dokumentacją. Wszyscy są bardzo zadowoleni, że utknęli w pracy nad oprogramowaniem, które nie będzie się sprzedawać. Lecz za każdym razem, gdy zespół wykryje ciekawą zmianę, którą należy wprowadzić w projekcie, połowa grupy musi wracać do specyfikacji i aktualizować ją, tak by plany były aktualne i prace toczyły się w odpowiednim kierunku. Można odnieść wrażenie, że aktualizowanie dokumentacji zajmuje równie dużo czasu co pisanie kodu.

Członkowie zespołu rozmawiali o tym, jak ograniczyć ilość pracy niezbędnej do aktualizowania dokumentacji. Prowadzono długie dyskusje na temat odpowiedniego poziomu jej szczegółowości. Jednak za każdym razem, kiedy próbowano z czegoś zrezygnować, znajdowała się osoba, która (słusznie) zwracała uwagę na to, że jeśli zespół nie opisze funkcji, wymagania, projektu lub przypadku testowego, ktoś może czegoś nie zrozumieć. Jeżeli potem implementacja okaże się błędna, zespół zostanie o to obwiniony. Wygląda więc na to, że wszystkie fragmenty dokumentacji są niezbędne, ponieważ bez tego zespół narazi się na tworzenie nieodpowiedniego oprogramowania.

Kup ksi k   Pole ksi k

Czy jest sposób na ograniczenie tego obciążenia bez ryzykowania problemów w projekcie? Czy w ogóle istnieje „właściwy” poziom szczegółowości dokumentacji projektów?

Punkty kluczowe

- *Dwanaście zasad programowania zwinnego*, które są dodatkiem do manifestu Agile, zapewnia użytkownikom podejścia zwinnego kierunek prac i wgląd w praktyki oraz metody.
- Zespoły stosujące podejście zwinne zapewniają klientom satysfakcję dzięki *szybkiemu uzyskiwaniu informacji zwrotnych* i *ciągłemu udostępnianiu oprogramowania*, co gwarantuje, że otrzymane informacje są aktualne (zasada numer 1).
- Zespoły stosujące podejście zwinne są *otwarte na zmiany* i traktują je jako pozytywne i korzystne poprawki w projekcie (zasada numer 2).
- Dzięki ograniczonym czasowo iteracjom, które pozwalają *często udostępniać działające oprogramowanie*, zespoły stosujące podejście zwinne nieustannie dostosują projekt, co prowadzi do zapewniania maksymalnej wartości klientom (zasada numer 3).

Komunikacja i współpraca

Zespoły programistyczne od zawsze zmagają się z pytaniem: jak szczegółowa ma być dokumentacja? Zespoły od lat szukają uniwersalnej metodyki w celu rozwiązania problemów z procesem, programowaniem i dostarczaniem rozwiązań, starają się też opracować uniwersalny system lub szablon tworzenia dokumentacji, który pozwoli w magiczny sposób zapisać wszystko, co jest potrzebne do utworzenia oprogramowania i jego późniejszej konserwacji.

Oto tradycyjny sposób myślenia o dokumentacji oprogramowania: potrzebujemy systemu zarządzania dokumentacją, aby każdy mógł w nim zapisywać dostępne informacje i łączyć je z danymi innych osób. Jeśli używany jest system śledzenia wpisów i można ustalić wszystkie powiązania między informacjami, otrzymamy prawie doskonały wgląd w to, co należy zbudować, jak to przetestować, jak zainstalować i konserwować. To podejście opiera się na założeniu, że programiści mogą powiązać każdy element projektu z wymaganiem, a testerzy — połączyć każdy przypadek testowy z aspektami projektu, wymagań i zakresu prac. Gdy zespół będzie musiał na przykład zmodyfikować fragment projektu, będzie mógł precyzyjnie określić wymagające zmian wymagania, kod, zakres prac i przypadki testowe. Dzięki temu nie trzeba będzie marnować dużych ilości czasu na analizy. Inżynierowie oprogramowania nazywają ten proces *analizą wpływu* i często starają się tworzyć rozbudowane macierze powiązań, w których wszystkie aspekty zakresu prac, wymagań, projektu i testów są połączone ze sobą. Kierownik projektu może wykorzystać takie macierze do wiązania kodu, raportów o błędach, elementów projektów i wymagań z ich źródłami.

Wróćmy więc do pytania, które sprawia zespołom trudności — ile dokumentacji należy tworzyć? Dla zespołów stosujących podejście zwinne odpowiedź brzmi: tyle, aby można było zbudować projekt. Dokładny zakres dokumentacji zależy od zespołu, poziomu komunikacji i rozwiązywanego problemu.

Kup ksi k Pole ksi k

Pomyśl o różnych rodzajach dokumentacji przygotowywanych przez zespoły programistów. Trzeba zapisać w postaci długich notatek wszystkie decyzje podjęte na każdym spotkaniu, przygotować dokumenty z odnośnikami opisujące każdy aspekt wszystkich źródeł i magazynów danych, a także utworzyć skomplikowane macierze z wyszczególnieniem ról, obowiązków i zasad powiązanych z każdą z osób. Przydatność każdej dokumentacji da się uzasadnić. Jeśli jednak nie pomaga ona w budowaniu oprogramowania i nie jest potrzebna z innych przyczyn (na przykład prawnych lub dla inwestora, starszych menedżerów albo innych interesariuszy), zespół zwinny nie będzie jej pisał. Jeżeli natomiast zespół stwierdzi, że *naprawdę* pomoże mu przygotowanie wymagań funkcjonalnych, może je opracować i nadal stosować podejście zwinne. Wszystko zależy od tego, co sprawdza się w zespole. Jest to jeden z aspektów swobody, jaką daje podejście zwinne.

Duża część dodatkowej dokumentacji, wykraczającej poza to, czego zespół potrzebuje do budowania oprogramowania (chodzi tu o wyczerpującą dokumentację, macierz śledzenia i materiały do analizy wpływu), ma przede wszystkim umożliwiać pełną analizę wpływu zmian. Choć w podejściu zwinnym stosowany jest inny (i często znacznie wydajniejszy) sposób zarządzania zmianami w projekcie, praktycy powinni pamiętać, że tradycyjna dokumentacja ma służyć podobnym celom. W modelu kaskadowym wyczerpująca dokumentacja ma pozwalać na lepsze radzenie sobie ze zmianami.

Paradoksalnie rozbudowana dokumentacja częściej przeszkadza w zarządzaniu zmianami, niż w tym pomaga. Marzeniem, którego spełnienie mają umożliwić perfekcyjna dokumentacja i mechanizmy śledzenia, jest system pozwalający zespołowi na automatyczne określenie wpływu każdej zmiany, co daje możliwość precyzyjnego ustalenia, jakie elementy wymagają poprawek.

Niestety, w większości zespołów zarządzanie zmianami z wykorzystaniem kompletnej dokumentacji nie jest takie proste. Na początku projektu trzeba poświęcić bardzo dużo czasu na próby prognozowania przyszłości i precyzyjnego zapisania dokumentacji. W trakcie projektu konieczne jest aktualizowanie napisanych dokumentów i śledzenie nowych mechanizmów. Jeśli zmieniła się pierwotna wizja rozwijanego produktu, trzeba zmodyfikować także wszystkie związane z nią dokumenty. Z czasem może to prowadzić do dezaktualizacji wielu dokumentów, a ich przygotowywanie i modyfikowanie wymaga od zespołu dużo pracy.

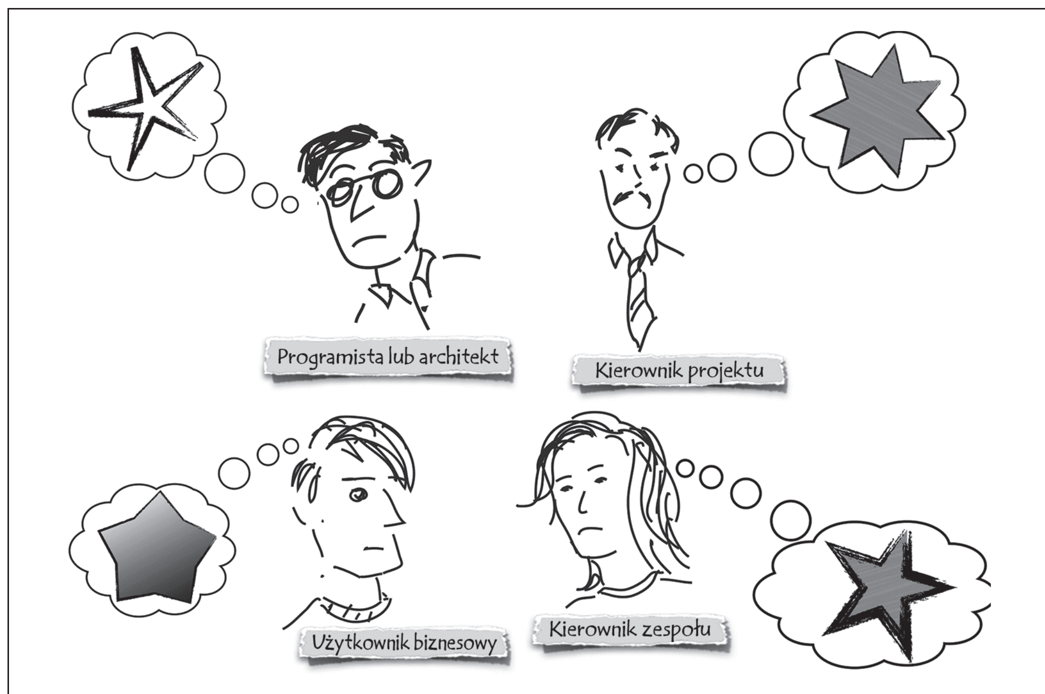
Ponieważ wyczerpująca dokumentacja nie jest doskonała, prowadzi do niepotrzebnych zmian i marnowania wysiłku. Każdy fragment dokumentacji jest pisany z perspektywy osoby pełniącej określoną funkcję. Analityk biznesowy przygotowuje wymagania ze swojego punktu widzenia, architekt rozwija projekty według własnej wizji, a inżynierowie z działu jakości tworzą plany testów z jeszcze innej perspektywy. Gdy trzeba powiązać wszystkie te aspekty w macierzy śledzenia, pojawiają się niespójności, których można oczekiwać ze względu na różne punkty widzenia. Tworzenie dokumentacji staje się celem samym w sobie i wymaga dodatkowego wysiłku na początku projektu. Kiedy w końcu pojawią się zmiany, całą pracę włożoną w uzgadnianie perspektyw i tworzenie jednego wyczerpującego zestawu dokumentacji trzeba wykonać od nowa. To sprawia, że zespół marnuje czas na ponowne pisanie dokumentacji, odtwarzanie macierzy śledzenia i rozwiązywanie konfliktów, podczas gdy powinien pisać kod, testować oprogramowanie i wprowadzać zmiany.

Musi istnieć wydajniejszy sposób tworzenia oprogramowania.

Kup ksi k Pole ksi k

Zasada numer 4. Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji zespołowi programistów i komunikowania się w jego ramach jest bezpośrednia rozmowa

Praktycy podejścia zwinnego wiedzą, że dokumentacja to jedna z form komunikowania się[4]. Jeśli piszę dokumentację i Ci ją przekazuję, *celem nie jest napisanie dokumentacji*. Ważne jest to, aby informacje z mojego i Twojego umysłu były jak najbardziej zbliżone do siebie (inaczej niż na rysunku 3.3). W wielu sytuacjach dokumentacja jest dobrym narzędziem, które pozwala uzyskać taki efekt. Jednak nie jest ona jedynym dostępnym sposobem komunikacji.



Rysunek 3.3. Gdy członkowie zespołu nie komunikują się ze sobą, mogą zgadzać się ze sobą na ogólnym poziomie, ale dążyć do różnych celów. Wyczerpująca dokumentacja jeszcze pogarsza sytuację, ponieważ może powodować niejednoznaczność

Bezpośrednia rozmowa to prawie zawsze lepsze od dokumentacji narzędzie wymiany informacji w zespole projektowym. Każdy wie, że osobiste omówienie problemu to najskuteczniejszy sposób na zrozumienie nowego zagadnienia. Ludzie łatwiej zapamiętują informacje, gdy uzyskają je w trakcie rozmowy, niż gdy przeczytają je z kartki lub w dokumencie Worda. To dlatego w podejściu zwinnym ważna jest głównie komunikacja między poszczególnymi osobami. Dokumentacja rezerwowana jest dla sytuacji, w których po pewnym czasie trzeba przypomnieć sobie skomplikowane informacje.

---

[4] Po prawdzie tradycyjni kierownicy projektów też zdają sobie z tego sprawę. Typowy kierownik projektu zna różnice między komunikacją formalną i nieformalną, a także między komunikacją pisemną i ustną. Uczy się też dostrzegać niewerbalne wskazówki w komunikacji oraz tego, że komunikacja twarzą w twarz to najskuteczniejszy sposób przekazywania informacji. Te zagadnienia są nawet poruszane na egzaminie PMP!

Kup ksi k Pole ksi k

Na szczęście zespołom, które są przyzwyczajone do używania wyczerpującej dokumentacji, łatwo jest przejść do skuteczniejszej komunikacji bezpośredniej. Wynika to z tego, że większość zespołów nie próbuje opracować idealnej, kompletnej dokumentacji i macierzy śledzenia. Inżynierowie oprogramowania są z natury praktyczni. Gdy widzą, jak dużo pracy wymaga przygotowanie wyczerpującej dokumentacji, i tak wybierają bezpośrednią rozmowę. Ostatecznie jest to jedyny sposób pozwalający skutecznie rozwijać oprogramowanie. Dzięki uznaniu, że tworzenie wyczerpującej dokumentacji jest często niewłaściwym rozwiązaniem, mogą przestać czuć się winni tego, że nie potrafią dokonać niemożliwego (czyli zbudować idealnej kompletnej dokumentacji). Przecież nawet gdyby przygotowali wyczerpującą dokumentację, nie byłaby ona przydatna w projekcie.

Najskuteczniejszą metodą przekazywania ważnych zagadnień w projekcie jest zadbanie o to, aby wszyscy myśleli w ten sam sposób. Dzięki temu każdy może na bieżąco podejmować właściwe decyzje. Jeśli grupa ludzi patrzy na świat z tej samej perspektywy i otwarcie dzieli się pozytywnymi oraz negatywnymi przemyśleniami, uzyskuje wspólny punkt widzenia. Gdy pojawia się zmiana wymagająca wyboru nowego kierunku, członkowie zespołu nie muszą wiele sobie tłumaczyć.

Ostatecznym celem komunikacji w zespole jest zbudowanie poczucia wspólnoty, tak aby duża część wiedzy znajdowała się już w umysłach poszczególnych osób. W końcu wielokrotne wyjaśnianie tych samych rzeczy nie jest wydajnym podejściem. Bez poczucia wspólnoty osobom pełniącym różne funkcje trudniej jest dopasować się do perspektywy innych członków zespołu. Im większe jest poczucie wspólnoty i bardziej uwspólniony punkt widzenia, tym częściej poszczególne osoby niezależnie dojdą do podobnych odpowiedzi na te same pytania. Pozwala to zapewnić stabilniejsze fundamenty w momencie wprowadzania zmian, ponieważ łatwiej jest rozwiązać konflikty i zacząć pracę nad kodem — i to bez rozpraszania się na aktualizowanie dokumentacji.

Zasada numer 5. Pracownicy biznesowi i programiści muszą codziennie współpracować przy projekcie

Zespoły stosujące podejście zwinne czasem zapominają, że współpracujący z nimi użytkownicy biznesowi też mają swoje zadania. To prowadzi do naturalnych rozbieżności między programistami a pracownikami biznesowymi, dla których tworzone jest oprogramowanie.

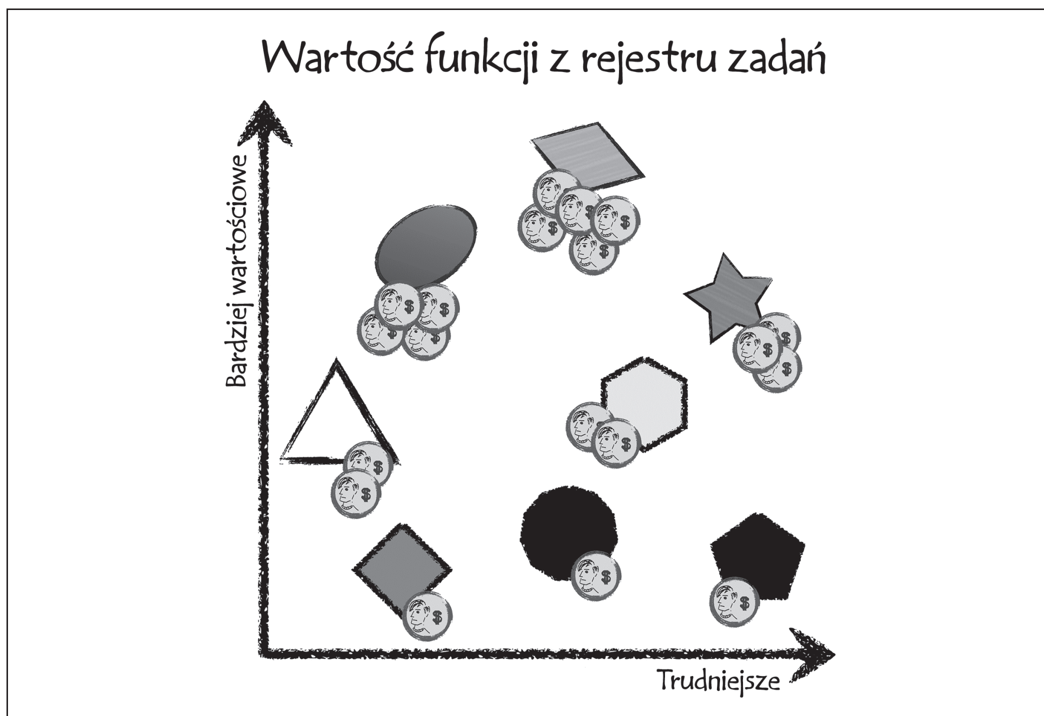
Aby zbudować dobre oprogramowanie, zespół musi przeprowadzić wiele bezpośrednich rozmów z pracownikami biznesowymi, ponieważ potrzebuje wiedzy na temat problemów firmy, które rozwijany produkt ma rozwiązywać. Pracownicy biznesowi posiadają tę wiedzę, gdyż rozwiązują te same problemy bez pomocy oprogramowania. Dlatego dla większości osób współpracujących z programistami uczestnictwo w projekcie związanym z oprogramowaniem stanowi tylko niewielką część zadań. Takie osoby najchętniej ograniczyłyby kontakty z programistami do minimum. Idealny scenariusz to wzięcie udziału w jednym lub dwóch spotkaniach, określenie, co oprogramowanie ma robić, i otrzymanie po krótkim czasie perfekcyjnie działającego oprogramowania.

Natomiast zespoły dążą do jak najczęstszych kontaktów z pracownikami biznesowymi. Programiści muszą poznać problemy biznesowe, które mają zostać rozwiązane. Robią to przez rozmowy z użytkownikami biznesowymi, towarzyszenie im przy pracy i obserwowanie jej efektów. Programista, który potrzebuje takich informacji, chce pełnej uwagi ze strony każdego pracownika biznesowego. Im dłużej musi czekać na odpowiedzi na swoje pytania, tym wolniej projekt posuwa się naprzód.

Kup ksi k Pole ksi k

Jednak pracownicy biznesowi nie chcą spędzać całego czasu z zespołem projektowym, ponieważ mają zadania do wykonania. Kto wygra w tej sytuacji?

Zespoły stosujące podejście zwinne mają rozwiązanie tego problemu, pozwalające odnieść zwycięstwo zarówno pracownikom biznesowym, jak i programistom. Należy zacząć od zrozumienia, że zespół dostarcza firmie *wartościowe* oprogramowanie. Gotowy produkt jest dla firmy warty określoną kwotę. Jeśli wartość oprogramowania jest wyższa niż koszt jego wytworzenia, opłaca się w nie zainwestować. Dobry projekt powinien być na tyle wartościowy, żeby pracownicy biznesowi dostrzegli, iż jest on wart włożenia w niego wysiłku. Rysunek 3.4 przedstawia wykres uwzględniający wartość i koszt rozwijanych funkcji.



Rysunek 3.4. Niektóre funkcje z rejestru zadań są dla firmy bardziej wartościowe od innych. Zespół musi znaleźć równowagę między wartością (ile dana funkcja jest warta) a kosztami (ile pracy trzeba włożyć w zbudowanie danego mechanizmu) przy określaniu, które rozwiązania należy opracować w każdej iteracji

Gdy pracownicy biznesowi i programiści budują oprogramowanie jako zespół, w dłuższej perspektywie najwydajniejsza jest ich codzienna współpraca przy projekcie. W przeciwnym razie pracownicy biznesowi zapoznają się z efektami prac zespołu i będą mogli udzielić informacji zwrotnych na późnym etapie prac, a wprowadzanie zmian pod koniec projektu jest bardzo kosztowne. Wczesne wykrywanie zmian wymaga znacznie mniej czasu. Dlatego w przekroju całego projektu codzienna współpraca z zespołem zajmuje każdemu użytkownikowi biznesowemu mniej czasu.

To z tego powodu zespoły, które często dostarczają działające oprogramowanie, powinny priorytetowo traktować najbardziej wartościowe funkcje. Dzięki temu pracownicy biznesowi szybko otrzymają coś wartościowego. To jeden z aspektów umowy. Dlatego dobre zespoły stosujące podejście

Kup ksi k Pole ksi k

zwinne uważają pracowników biznesowych za ważną część grupy, równie istotną jak programiści. Grupy stosujące podejście zwinne bardzo różnią się pod tym względem od tradycyjnych zespołów. Te ostatnie traktują użytkowników biznesowych jak klientów, z którymi trzeba negocjować. Zespół zwinny *współpracuje* z klientem (jest nim zwykle właściciel produktu) jak z kimś, kto ma równie duże prawo do wpływu na przebieg projektu. Sprowadza się to do podstawowej wartości podejścia zwinnego, zachęcającej do współpracy z klientem zamiast do negocjowania kontraktu.

Dobry właściciel produktu pomaga ograniczyć czas spędzany przez pracowników biznesowych z zespołem. Obie strony muszą się codziennie spotykać, jednak właściciel produktu może skoncentrować się na zrozumieniu wartości oprogramowania i problemów biznesowych, które należy rozwiązać. Dzięki temu zespół może wykorzystać czas spędzany z pracownikami biznesowymi na weryfikowanie informacji, które otrzymał już od właściciela produktu.

Zasada numer 6. Opieraj projekty na zmotywowanych osobach. Zapewnij im potrzebne środowisko i wsparcie oraz uwierz, że wykonają zadanie

Projekty przebiegają najlepiej, gdy wszyscy w firmie uważają, że zespół rozwija wartościowe oprogramowanie, a także gdy wszystkie osoby (włącznie z właścicielem produktu) rozumieją, co sprawia, że oprogramowanie jest cenne dla organizacji.

Problemy mogą natomiast wystąpić w środowisku, w którym ludzie nie widzą wartości w oprogramowaniu lub nie są nagradzani za poprawne rozwijanie produktu. Nie jest niczym niezwykłym, że firma opracowuje systemy oceny wydajności i naliczania wynagrodzeń, które zniechęcają do rozwijania oprogramowania w efektywny, zwinny sposób. Może to negatywnie wpływać na przebieg projektu. Oto często spotykane „programy motywacyjne", które szkodzą zespołom stosującym podejście zwinne:

- Dawanie programistom złych ocen, gdy w trakcie sprawdzania kodu wykrywane są błędy. Nagradzanie za kod, w którym nie wykryto usterek. To podejście powoduje, że programiści przestają znajdować błędy w trakcie sprawdzania kodu.
- Nagradzanie testerów za liczbę zgłoszonych błędów. To zachęca do nadmiernej drobiazgowości i tworzenia mało wartościowych raportów. Ponadto zniechęca to testerów do współpracy z programistami, ponieważ buduje między obiema grupami antagonistyczne nastawienie.
- Ocenianie pracy analityków biznesowych na podstawie ilości napisanej dokumentacji, a nie według wiedzy przekazanej zespołowi.

Ostatecznie pracę całego zespołu należy oceniać na podstawie dostarczonego produktu, a nie według pełnionych funkcji. Oczywiście jeśli programista wykonuje kiepską pracę lub dezorganizuje projekt, przełożony powinien zwrócić mu na to uwagę. Pracowników należy oceniać na podstawie ich wkładu (albo jego braku) w realizowanie celów całego zespołu. Jednak *zdecydowanie nie wolno* zniechęcać ich do wykraczania poza przypisane im role. W odpowiednim środowisku należy nagrodzić programistę, który dostrzegł pominięty aspekt problemu biznesowego, oraz testera, który wykrył usterki w kodzie lub architekturze i poinformował o tym zespół. W tym modelu wszyscy członkowie zespołu otrzymują potrzebne wsparcie, a projekt ma większe szanse powodzenia.

Kup ksi k
Pole ksi k

Wyczerpująca dokumentacja i macierze śledzenia są wyjątkowo podstępnym źródłem problemów związanych ze środowiskiem i wsparciem w zespole. Zamiast budować atmosferę zaufania, zachęcają do „chronienia swojego tyłka". Zespół przyjmuje wtedy podejście oparte na negocjowaniu kontraktu zamiast na współpracy z klientem.

Tester dbający o „chronienie tyłka" poświęca czas na upewnianie się, że dla każdego wymagania istnieje test, i to niezależnie od tego, czy pomaga zapewnić wysoką jakość oprogramowania. Programiści z takim nastawieniem dosłownie traktują wymagania bez zastanawiania się nad tym, czy pisany kod jest wartościowy dla użytkowników. Dzieje się tak, ponieważ w środowisku, w którym ważne jest „chronienie tyłka", próba zbudowania czegoś, co jest klientowi naprawdę potrzebne, może prowadzić do nagany za tworzenie oprogramowania niezgodnie ze specyfikacją. Analitycy biznesowi i właściciele produktu „chronią tyłek", upewniając się, że zakres prac i wymagania są do siebie precyzyjnie dopasowane. Często pomijają przy tym wartościowe, ale kłopotliwe wymagania, które nie wynikają bezpośrednio z istniejącej dokumentacji.

„Chronić tyłki" najczęściej muszą członkowie zespołów pracujący w środowisku, w którym zmiana jest traktowana jak coś złego. Gdy zespół posługuje się wyczerpującą dokumentacją, łatwo jest postrzegać zmiany jako niekorzystne zjawisko, ponieważ wymagają one ponownej oceny zakresu prac, zaktualizowania specyfikacji, zmodyfikowania projektu, poprawienia macierzy śledzenia itd. To prowadzi do podziałów, gdyż kierownicy w takich firmach starają się znaleźć kozła ofiarnego, którego można obwinić o dodatkową pracę związaną ze zmianą. Ludzie w takim zespole starają się pisać zabezpieczającą ich dokumentację, aby chronić się przed obwinieniem. To zmusza wszystkich w grupie do „chronienia tyłków". Każdy dba wtedy o to, by móc wskazać fragment dokumentacji, do którego się stosował. Pozwala to uniknąć złej oceny lub nagany.

Odwrotnością „chronienia tyłka" jest zaufanie. Jeśli w firmie przygotowywana jest tylko minimalna dokumentacja niezbędna w projekcie, powstaje środowisko, w którym zespołowi wierzy się, że w obliczu zmian podejmie właściwe decyzje. Stosujący zwinne podejście zespół z nastawieniem „wszyscy płyniemy na jednej łodzi", w którym to zespole w razie niepowodzenia wszyscy czują się za to odpowiedzialni, nie potrzebuje „chronić tyłków". Łatwiej jest wtedy radzić sobie ze zmianami, ponieważ nie trzeba przygotowywać niepotrzebnej dokumentacji. Zamiast tego można rozwiązać problem za pomocą bezpośredniej komunikacji i zapisywać tylko niezbędne informacje. Zespół może tak pracować, bo wie, że firma wierzy w jego kompetencje — *nawet jeśli konieczne jest przedłużenie terminu wykonania projektu.*

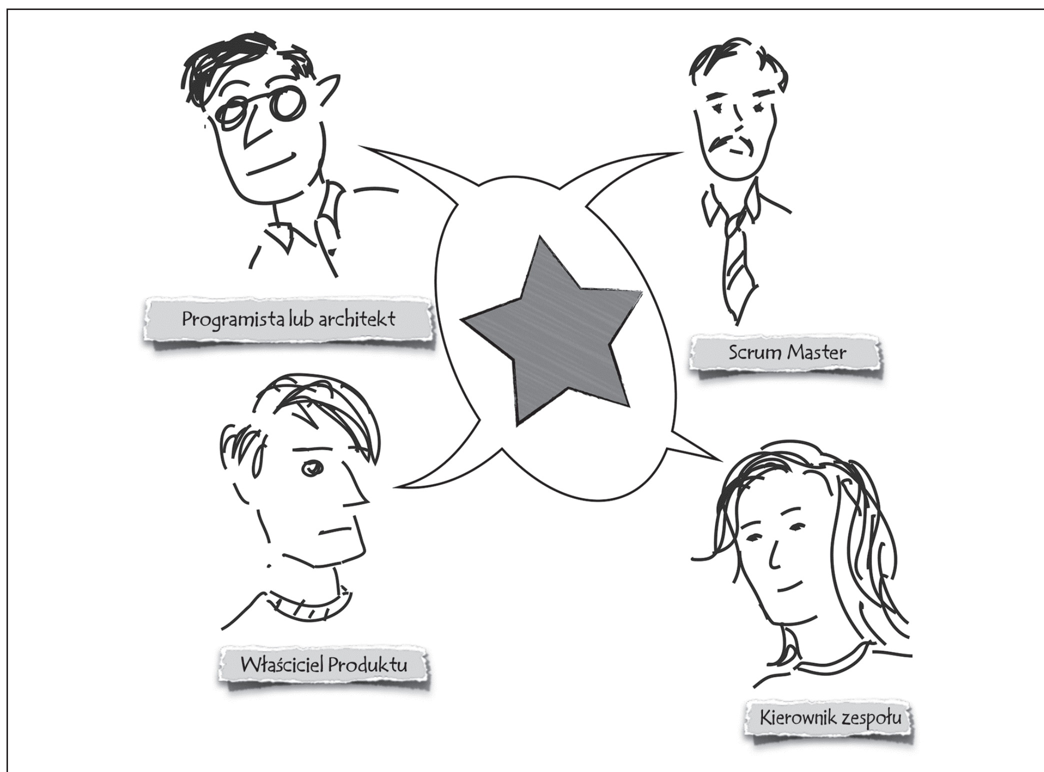
Lepsza komunikacja w projekcie czytnika e-booków

W projekcie czytnika e-booków z pewnością przyda się lepsza komunikacja. Pamiętasz intensywne spotkania, na których analitycy biznesowi starannie tworzyli rozbudowany zestaw wymagań? Początkowo nikt nie traktuje go jako wyrafinowanego narzędzia do „chronienia tyłka". Wszyscy w zespole szczerze wierzą, że dzięki wielodniowym dyskusjom na temat każdego szczegółu oprogramowania uda się uwzględnić wszystkie zagadnienia, co pozwoli zbudować możliwie najlepszy produkt. Ponieważ poświęcono na specyfikację tyle czasu, każdy jest gotów trzymać się jej i bronić, nawet jeśli się okaże, że końcowy produkt może nie odnieść sukcesu rynkowego. Gdyby zespół

Kup ksi k Pole ksi k

potrafił dokładnie przewidzieć, czego użytkownicy będą potrzebowali za dwa lata, takie podejście świetnie by się sprawdziło! Szkoda tylko, że tak się nie stało. Przynajmniej nikt nie stracił pracy, bo wszyscy potrafili udowodnić, że ściśle trzymali się specyfikacji.

Co by się stało, gdyby zespół od początku posługiwał się lepszym modelem komunikacji? Jakie zmiany są możliwe, gdy zamiast przygotowywać kompletny zestaw wymagań, zespół zapisuje jedynie **minimalną dokumentację**, niezbędną do rozpoczęcia pracy? Odpowiedzią jest rysunek 3.5.



Rysunek 3.5. Gdy zespół w większym stopniu polega na bezpośrednich rozmowach i korzysta z minimalnej ilości dokumentacji potrzebnej do zbudowania projektu, wszystkim łatwiej jest uzgodnić informacje

Członkowie zespołu muszą sobie ufać, aby podejmować właściwe decyzje. Byłoby to bardzo pomocne w trakcie prac nad formatem e-booków, gdyż pozwoliłoby to uniknąć przywiązania do nieaktualnego formatu ustalonego na początku projektu i w zamian wprowadzić nowszy format. Jeszcze lepsze jest to, że w momencie rozpoczęcia pracy nad sklepem internetowym może być już oczywiste, iż nie jest to dobry pomysł. Wtedy można po prostu z niego zrezygnować. Nie jest to jednak dopuszczalne, gdy sklep jest częścią dokumentacji, której zespół musi się trzymać. Lepsza komunikacja pozwala aktualizować projekt i dostarczyć bardziej wartościowy produkt.

Wyobraź sobie, jak może to wyglądać w zespole pracującym nad czytnikiem. Teraz pracownicy mają lepsze nastawienie do dokumentacji, ponieważ stanowi ona znacznie mniejsze obciążenie. Zespół uważa, że może zaoszczędzić dużo czasu dzięki lepszej komunikacji i rezygnacji ze zbędnych dokumentów. Co się jednak stanie, jeśli mimo to projekt nie będzie przebiegał zgodnie z planem?

Kup ksi k Pole ksi k

Z niewiadomych przyczyn nie udało się przyspieszyć prac. Wygląda na to, że pracownicy częściej niż kiedykolwiek wcześniej pracują wieczorami i w weekendy, próbując dodać wszystkie funkcje obiecane przez właściciela produktu w poszczególnych iteracjach. Im zwinniejsze podejście stosuje zespół, tym więcej ma pracy i tym częściej jego członkowie spędzają wieczory i weekendy z dala od rodzin. To nie jest żadna poprawa! Czy można jakoś rozwiązać problem, zanim pracownicy do reszty się wypalą?

Punkty kluczowe

- Nadmiernie rozbudowana dokumentacja zwiększa ryzyko wieloznaczności, nieporozumień i błędów w komunikacji między członkami zespołu.
- Zespoły stosujące podejście zwinne komunikują się najskuteczniej, gdy *koncentrują się na bezpośrednich rozmowach* i przygotowują minimalną ilość dokumentacji niezbędną w projekcie (zasada numer 4).
- Programiści współpracują z użytkownikami biznesowymi każdego dnia, dlatego mogą *zapewnić maksymalną wartość* (zasada numer 5).
- Wszyscy w zespole stosującym podejście zwinne *czują się odpowiedzialni* za projekt i za jego sukces (zasada numer 6).

Przebieg projektu — posuwanie się do przodu

Lepsza komunikacja i zaufanie w zespole to świetny punkt wyjścia. Gdy wszyscy dobrze ze sobą współpracują i znają swoje miejsce w projekcie, mogą zacząć zastanawiać się nad podstawowym problemem: jak wykonywać codzienne zadania? W jaki sposób zespół zwinny posuwa się do przodu?

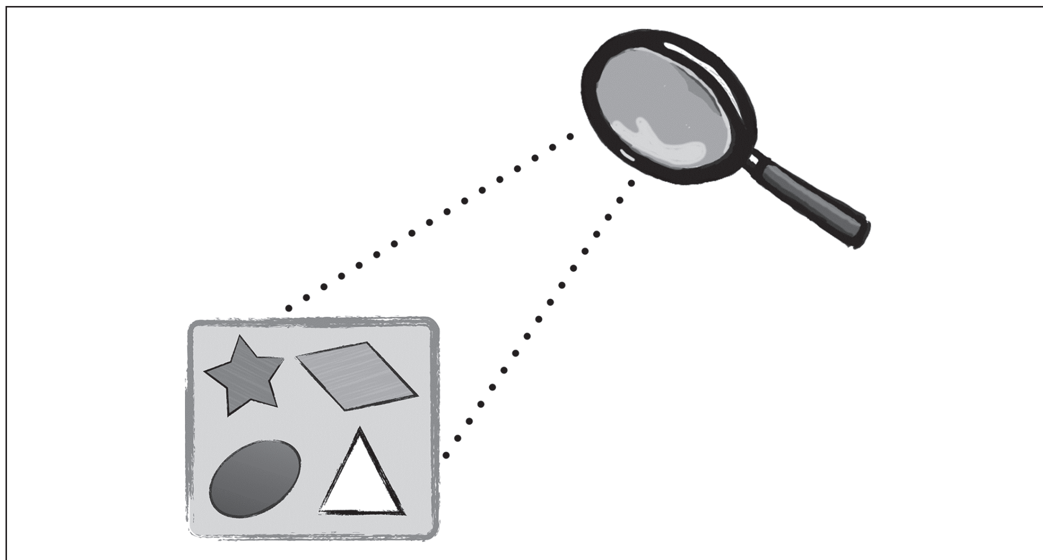
Zasada numer 7. Główną miarą postępów jest działające oprogramowanie

Dobry zespół pracuje tak, aby każdy (członkowie zespołu, menedżerowie, interesariusze i klienci) zawsze dobrze znał aktualny stan projektu. W jaki sposób przekazywać informacje na ten temat? Ten problem okazuje się nieoczekiwanie trudny.

Typowy kierownik projektu stosujący podejście „dowódź i kontroluj” stara się utrzymać kierunek prac i udostępnia wszystkim aktualne rozbudowane harmonogramy i raporty o stanie projektu. Jednak trudno jest uchwycić w takim raporcie prawdziwą istotę projektu. Sam raport to niedoskonałe narzędzie komunikacji. Często trzy różne osoby po zapoznaniu się z tym samym raportem mają trzy zupełnie inne opinie na temat postępów prac. Ponadto na zawartość takich raportów wpływ mają kwestie polityczne. Niemal każdy kierownik projektu od czasu do czasu woli pominąć w raporcie coś, co stawia samego kierownika lub zespół w złym świetle. Prawie zawsze jest to informacja potrzebna komuś do podjęcia decyzji. Jak więc informować o postępie prac, skoro raporty o stanie nie są wystarczająco dobre?

Kup ksi k Pole ksi k

Odpowiedzią jest działające oprogramowanie. Gdy tylko zobaczysz produkt, od razu zrozumiesz stan prac. Dowiesz się, co działa, a co jeszcze nie jest gotowe. Jeśli kierownik obiecał udostępnić funkcję, która nie znalazła się w oprogramowaniu, może to być dla niego krępujące, ale nie da się tego ukryć, ponieważ oprogramowanie mówi samo za siebie (zob. rysunek 3.6).



Rysunek 3.6. Działające oprogramowanie lepiej niż raport o stanie informuje wszystkich o postępach prac i jest najskuteczniejszym sposobem, w jaki zespół może pokazać, co udało mu się osiągnąć

Jest to jeden z powodów, dla których zespoły stosujące podejście zwinne korzystają z iteracji. Dzięki udostępnianiu po każdej iteracji działającego oprogramowania i prezentowaniu rzeczywistego produktu, aby pokazać wszystkim, czego zespół dokonał, można informować o postępach prac w sposób, którego właściwie nie da się nie zrozumieć.

Zasada numer 8. W procesach zwinnych ważna jest możliwość utrzymania tempa programowania. Sponsorzy, programiści i użytkownicy powinni móc w nieskończoność utrzymywać stałe tempo pracy

Zespół odpowiedzialny za czytnik e-booków nie jest pierwszym, który pracuje w nadgodzinach, by zmieścić się w nierealistycznym terminie. Trudne do zrealizowania i krótkie terminy to podstawowe narzędzie w przyborniku kierowników o nastawieniu „dowódź i kontroluj". Gdy zbliża się dzień oddania produktu, pierwszym pomysłem na zdążenie na czas jest praca wieczorami i w weekendy. Nierealistyczne terminy bywają też nieeleganckim sposobem na wymuszenie na zespole większego wysiłku i wyciśnięcie z pracowników dodatkowych godzin pracy każdego tygodnia.

Jednak w dłuższej perspektywie to podejście się nie sprawdza. Wiadomo, że zespół może harować przez kilka tygodni i wykonać w tym czasie więcej pracy, jednak potem następuje zwykle nagły spadek wydajności. Jest to zrozumiałe — pracownicy stają się zmęczeni i zdemotywowani.

Kup ksi k Pole ksi k

Wszystkie przekładane na potem sprawy i wydarzenia życiowe wymagają w końcu uwagi. W praktyce zespoły, w których liczba nadgodzin jest skrajnie duża, w dłuższym okresie robią mniej niż grupy pracujące w normalnych godzinach, a ponadto zwykle tworzą rozwiązania niższej jakości.

To dlatego w zespołach stosujących podejście zwinne liczy się **możliwe do utrzymania tempo pracy**. Należy zaplanować udostępnienie rozwiązań, które da się ukończyć w przeznaczonym na nie czasie. Podejście iteracyjne to ułatwia, ponieważ znacznie prościej jest oszacować funkcje oprogramowania, które uda się dostarczyć w ciągu następnych dwóch, czterech lub sześciu tygodni, niż ocenić zakres prac możliwych do ukończenia w półtora roku. Dzięki zobowiązaniu się do dostarczenia tylko tego, co zespół potrafi zbudować, można otrzymać środowisko, w którym praca wieczorami i w weekendy jest rzadkością[5].

Zasada numer 9. Ciągła troska o techniczną doskonałość i dobry projekt są zgodne z podejściem zwinnym

Niepoprawne szacunki nie są jedynym źródłem pracy wieczorami i w weekendy. Większość programistów zna nieprzyjemny ucisk w żołądku, który pojawia się po stwierdzeniu, że zaprojektowanie pozornie prostego fragmentu kodu okazuje się prawdziwym koszmarem. Wiadomo wtedy, że kilka następnych weekendów trzeba będzie poświęcić na wykrywanie błędów i poprawianie kodu.

W dłuższej perspektywie znacznie szybsze jest unikanie usterek niż naprawianie ich później. Ponadto zdecydowanie łatwiej jest konserwować dobrze zaprojektowany kod, gdyż jest on tak zbudowany, aby pozwalał na proste rozszerzanie.

W ostatnich dwóch dziesięcioleciach nastąpiła prawdziwa rewolucja w projektowaniu oprogramowania. Projektowanie obiektowe i analiza obiektowa, wzorce projektowe, architektura rozdzielona i oparta na usługach oraz inne innowacje zapewniły programistom metody i narzędzia, które umożliwiają uzyskanie technicznej doskonałości w każdym projekcie.

Nie oznacza to jednak, że zespoły stosujące podejście zwinne zawsze na początku prac nad oprogramowaniem poświęcają dużo czasu na tworzenie rozbudowanych planów. Programiści korzystający z tego podejścia rozwijają w sobie cenne nawyki, pomagające im pisać dobrze zaprojektowany kod. Nieustannie zwracają uwagę na problemy z projektem oraz kodem i rozwiązują je natychmiast po ich wykryciu. Dzięki poświęceniu niewielkiej ilości dodatkowego czasu na pisanie solidnego kodu i rozwiązywanie problemów na bieżąco budują bazę kodu, który później łatwo będzie konserwować.

[5] Jest to dobry przykład upraszczania, o czym była mowa w rozdziale 1. Tu warto tylko wspomnieć, że „możliwe do utrzymania tempo" oznacza zapewnienie zespołowi wystarczającej ilości czasu na zbudowanie oprogramowania w standardowych godzinach, bez pracy wieczorami i w weekendy. W dalszej części książki szczegółowo wyjaśniamy, jak możliwe do utrzymania tempo wpływa na środowisko pracy zespołu i kulturę firmy, a także na jakość rozwijanego oprogramowania.

Kup ksi k Pole ksi k

Lepsze środowisko pracy dla zespołu rozwijającego czytnik e-booków

Zespół odpowiedzialny za czytnik e-booków (oraz partnerzy i partnerki jego członków) z pewnością doceniają możliwe do utrzymania tempo prac. Jednak także sam projekt przebiega teraz lepiej. W opisanym wcześniej podejściu zespół był od początku skazany na pracę wieczorami i w weekendy, bo nie miał narzędzi potrzebnych do opracowania realistycznego planu, który pozostanie aktualny również za półtora roku.

Co gorsza, ponieważ zespół na początku prac przygotował projekt i architekturę oprogramowania pod kątem bardzo szczegółowej specyfikacji, ostatecznie powstał niezwykle skomplikowany kod, którego rozwijanie było trudne. To spowodowało, że wiele drobnych zmian trzeba było wprowadzać w ramach obszernych poprawek. W efekcie w kodzie bazowym pojawił się niezrozumiały kod. Gdy zespół stosuje podejście iteracyjne i udostępnia działające oprogramowanie, może zaplanować każdą iterację tak, aby utrzymać stałe tempo pracy. Prostsze podejście do przygotowywania architektury, polegające na planowaniu tego, co w danym momencie potrzebne, zapewnia większą swobodę i ułatwia wprowadzanie zmian. Wykorzystanie lepszych praktyk związanych z projektem, architekturą i pisaniem kodu pozwoliłoby zespołowi otrzymać kod łatwiejszy w konserwacji i modyfikowaniu.

Załóżmy teraz, że zespół odpowiedzialny za czytnik zastosował nowe zasady i pracuje teraz jak dobrze naoliwiona maszyna do produkcji oprogramowania. Wykonuje iteracje i regularnie dostarcza działające oprogramowanie, a także cały czas wprowadza poprawki, by mieć pewność, że buduje jak najbardziej wartościowy produkt. Komunikacja przebiega sprawnie, a w dokumentacji uwzględniane są tylko niezbędne zagadnienia. Grupa stosuje świetne praktyki projektowe i rozwija łatwą w konserwacji bazę kodu — a wszystko to bez nadgodzin. Zespół stał się zwinny!

Jednak w następnym projekcie zaczynają się problemy. Nowy kierownik projektu właśnie wysłał do każdego, do kogo udało mu się dotrzeć, zaproszenie na duże spotkanie. Uczestnicy zaczynają akceptować zaproszenia, zarezerwowana jest sala konferencyjna i rozpoczynają się dyskusje na temat wymagań, które trzeba udokumentować. Wszyscy z nowego, zwinnego zespołu zaczynają odczuwać dziwny ucisk w żołądku.

Wiedzą, co ich czeka. Zaczynają pojawiać się pierwsze z wielu specyfikacji, planów i diagramów Gantta. Jak zespół może zadbać o to, aby w następnym projekcie nie wpaść w te sam pułapki, z których z takim trudem się wydostawał?

Punkty kluczowe

- Najskuteczniejszym sposobem ilustrowania postępów w projekcie jest *dostarczanie działającego oprogramowania* i udostępnianie go użytkownikom (zasada numer 7).
- Zespoły są najbardziej produktywne, gdy *pracują w możliwym do utrzymania tempie* i unikają heroicznych zrywów, skrótów oraz nadgodzin (zasada numer 8).
- Oprogramowanie *dobrze zaprojektowane i napisane* zostaje udostępnione najszybciej, ponieważ najłatwiej się je modyfikuje (zasada numer 9).

Kup ksi k Pole ksi k

Nieustanne ulepszanie projektów i zespołu

Jedną z najbardziej podstawowych zasad projektowych — i to nie tylko w dziedzinie oprogramowania, ale w całej inżynierii — jest reguła KISS (ang. *keep it simple, stupid*, czyli „nie komplikuj, głupku"). Zespoły zwinne stosują się do niej w obszarze planowania projektu, budowania oprogramowania i funkcjonowania grupy.

Zasad numer 10. Konieczna jest prostota rozumiana jako sztuka maksymalizowania niewykonywanej pracy

Dodawanie kodu do istniejącego projektu często zwiększa poziom złożoności — zwłaszcza gdy potem trzeba napisać jeszcze więcej powiązanego kodu. Zależności między systemami, obiektami, usługami itd. sprawiają, że kod staje się bardziej skomplikowany i trudniejszy w modyfikowaniu. Zależności zwiększają prawdopodobieństwo tego, że jedna zmiana wpłynie na inną część systemu, co z kolei będzie wymagało poprawek w trzeciej części itd. W efekcie powstaje efekt domina i każda zmiana prowadzi do wzrostu złożoności. Zastosowanie iteracji i tworzenie na początku projektu tylko minimalnej dokumentacji pomaga zespołowi uniknąć pisania niepotrzebnego kodu.

Jednak wielu programistów czuje się niekomfortowo, kiedy pierwszy raz słyszy zwroty typu: „Zastosuj iteracje" lub „Ogranicz planowanie do minimum niezbędnego do rozpoczęcia prac nad projektem". Te osoby mają wrażenie, że pisanie kodu przed podjęciem wielu decyzji na temat projektu i architektury jest przedwczesne. Sądzą, że jeśli zaczną od razu pisać kod, będą musiały wkrótce go usunąć, gdy projekt się zmieni.

To zrozumiała reakcja, ponieważ w wielu projektach spoza obszaru tworzenia oprogramowania zalecane tu podejście nie ma sensu. Nierzadko zdarza się, że programista poznający podejście zwinne zgłasza następujące zastrzeżenia: „Jeśli zatrudniam ekipę do remontu domu, chcę najpierw zobaczyć kompletny plan. Nie chcę, aby pracownicy po krótkiej rozmowie od razu przystępowali do wyburzania ścian".

Te argumenty są uzasadnione w kontekście renowacji domu. Jest tak, bo przy remoncie domu najbardziej destrukcyjną rzeczą, jaką możesz zrobić, jest wzięcie do ręki młota i wyburzanie nim ścian. Jednak oprogramowanie różni się od budynków i innych obiektów fizycznych. Usuwanie kodu nie jest destrukcyjne, ponieważ zwykle można odzyskać go z systemu kontroli wersji. Najbardziej destrukcyjne jest tu napisanie nowego kodu, potem dodanie kodu od niego zależnego, opracowanie dalszego kodu, zależnego od tego ostatniego itd. Prowadzi to do boleśnie znajomego efektu domina i kaskadowego wprowadzania zmian. Ostatecznie otrzymujesz niemożliwy do konserwacji niezrozumiały kod.

Maksymalizowanie pracy, która *nie* jest wykonywana, pomaga tego uniknąć. Najlepszym rozwiązaniem jest budowanie systemów, w których nie występuje wiele zależności i niepotrzebnego kodu. Najskuteczniejszym sposobem na uzyskanie tego efektu jest współpraca z klientami i interesariuszami oraz budowanie tylko najbardziej przydatnego i wartościowego oprogramowania. Jeśli dana funkcja nie jest wartościowa, dla firmy w dłuższej perspektywie często taniej jest w ogóle jej nie rozwijać, gdyż koszty konserwowania dodatkowego kodu są wyższe niż wartość, jaką daje on firmie.

Kup ksi k Pole ksi k

Gdy zespół pisze kod, może zachować prostotę projektu dzięki budowaniu niewielkich i niezależnych jednostek (klas, modułów, usług itd.), które wykonują tylko jedno zadanie. To podejście pomaga uniknąć efektu domina[6].

Zasada numer 11. Najlepsze architektury, wymagania i projekty są owocem pracy samoorganizujących się zespołów

Nadmiernie skomplikowane projekty są często spotykane w zespołach, które poświęcają za dużo czasu na wstępne plany. Nie jest to zaskoczeniem. Wróć do ilustracji modelu kaskadowego z rozdziału 2. Występuje w nim cały etap poświęcony wymaganiom oraz inny, dotyczący projektu i architektury. Wykonanie najlepszej możliwej pracy na etapie planowania projektu i architektury polega na opracowaniu możliwie najlepszej architektury, jaką zespół potrafi przygotować. W zespołach pracujących w ten sposób niewielka liczba wymagań i prosty projekt intuicyjnie wydają się niewystarczające. Czy dziwi więc to, że rozwijają rozbudowaną dokumentację wymagań i skomplikowane projekty? Oczywiście nie, ponieważ bezpośrednio tego właśnie się od nich wymaga przez poświęcanie na takie zagadnienia całego etapu w procesie prac.

Z kolei **zespół samoorganizujący się** nie stosuje jawnie wydzielonego etapu tworzenia wymagań lub projektu. Cały taki zespół współpracuje przy planowaniu projektu (zamiast polegać na jednej osobie, która jest właścicielem planu) i modyfikowaniu planów. Zespół działający w ten sposób zwykle rozbija projekt na historie użytkowników i inne niewielkie porcje oraz zaczyna pracę od tych zagadnień, które zapewniają firmie największą wartość. Dopiero potem zaczyna zastanawiać się nad szczegółowymi wymaganiami, projektem i architekturą.

To sprawia, że praca tradycyjnego architekta oprogramowania jest trudniejsza, ale też daje więcej satysfakcji. Standardowo architekt oprogramowania siedzi w gabinecie za zamkniętymi drzwiami i myśli o abstrakcyjnych problemach, które trzeba rozwiązać. Oczywiście wielu architektów postępuje inaczej, jednak nie jest niczym niezwykłym sytuacja, w której architekt jest nieco odcięty od codziennej pracy zespołu.

W zespołach zwinnych wszyscy są odpowiedzialni za architekturę. Starszy architekt lub projektant oprogramowania nadal odgrywa ważną rolę, ale nie pracuje w izolacji. Gdy zespół rozwija oprogramowanie element po elemencie, zaczynając od najbardziej wartościowych fragmentów, zadanie architekta staje się trudniejsze, choć często jednocześnie ciekawsze. Zamiast tworzyć na początku prac jeden rozbudowany projekt z uwzględnieniem wszystkich wymagań, architekci w zespołach zwinnych rozwijają **projekt przyrostowo**. Stosują przy tym techniki umożliwiające projektowanie systemów, które nie są kompletne, za to pozwalają zespołowi na łatwe wprowadzanie zmian.

[6] To kolejne uproszczenie. Dalej dokładnie opisujemy, jak zespoły mogą rozwijać świetny kod bez tworzenia na początku rozbudowanych projektów, jaki wpływ ma to na prowadzone prace, a także jak stosować to podejście i zachować otwartość na późniejsze zmiany.

Kup ksi k Pole ksi k

Zasada numer 12. Zespół regularnie zastanawia się nad tym, jak zwiększyć swoją efektywność, a następnie odpowiednio usprawnia i dostosowuje swoje postępowanie

Zespół nie jest zwinny, jeśli nie dba o stałe usprawnianie procesów rozwijania oprogramowania. Zespoły zwinne cały czas badają sytuację i adaptują się do niej. Analizują przebieg wcześniejszych projektów i wykorzystują zdobytą wiedzę do późniejszego wprowadzania usprawnień. Robią to nie tylko po zakończeniu projektu. Każdego dnia w trakcie spotkań szukają możliwych modyfikacji i jeśli jest to uzasadnione, wprowadzają zmiany[7]. Musisz być brutalnie szczery z sobą samym i współpracownikami w kwestii tego, co działa, a co się nie sprawdza. Jest to ważne zwłaszcza w zespołach, które dopiero uczą się podejścia zwinnego. Jedyny sposób na poprawę funkcjonowania zespołu to ciągłe analizowanie dotychczasowych dokonań, aby ocenić jakość współpracy i opracować plan poprawy.

Ma to sens i prawie wszyscy zgadzają się z tym, że to dobre podejście. Jednak patrzenie wstecz i ocenianie, co się udało, a co się nie powiodło, to jedna z rzeczy, które wiele zespołów *naprawdę* zamierza robić, ale ostatecznie pomija. Jednym z powodów jest to, że początkowo to podejście budzi nieprzyjemne odczucia. Często wymaga przyjrzenia się konkretnym problemom i błędom, a bardzo niewiele osób czuje się dobrze, publicznie wytykając błędy innych osób z zespołu. Z czasem członkowie zespołu zyskują coraz większą swobodę przy omawianiu takich kwestii. Ostatecznie wszyscy zaczynają postrzegać takie rozmowy jako w większym stopniu konstruktywne niż poświęcone krytyce.

Inną przyczyną, dla której zespoły rezygnują z takich analiz, jest to, że nie rezerwują na nie czasu — a nawet jeśli to robią, szybkie rozpoczęcie prac nad następnym projektem wydaje się ważniejsze niż rozważania nad poprzednim. Zespoły, które rozpoczynają pracę od zarezerwowania w końcowej części każdej iteracji i samego projektu czasu na spotkanie, analizy, ocenę i opracowanie planu poprawy, częściej *rzeczywiście się spotykają* i rozmawiają o tym, jak sobie poradziły. Pomaga to wyciągać wnioski z doświadczeń i zwiększać efektywność.

Punkty kluczowe

- Zespoły zwinne *dbają o maksymalną prostotę rozwiązań*. W tym celu unikają budowania niepotrzebnych funkcji lub nadmiernie skomplikowanego oprogramowania (zasada numer 10).
- Samoorganizujące się zespoły *są współodpowiedzialne za wszystkie aspekty projektu* — od udostępnienia produktu, przez zarządzanie projektem, po projektowanie i implementację kodu (zasada numer 11).
- Dzięki poświęcaniu czasu na *spojrzenie wstecz i przedyskutowanie wyciągniętych wniosków* po każdej iteracji i na zakończenie projektu zespoły zwinne nieustannie stają się lepsze w rozwijaniu oprogramowania (zasada numer 12).

[7] To jeszcze jedno uproszczenie. Tu tylko pokrótce wspominamy, że zespoły zwinne analizują projekty i się adaptują. Z rozdziału 4. dokładnie dowiesz się, jak przebiega to w zespołach stosujących podejście Scrum i co wspólnego z tym mają samoorganizujące się zespoły.

Kup ksi k Pole ksi k

Projekt w podejściu zwinnym — łączenie wszystkich zasad

Podejście zwinne jest czymś wyjątkowym w historii inżynierii oprogramowania. Różni się od proponowanych wcześniej uniwersalnych metodyk, które obiecywały rozwiązywać problemy zespołów programistycznych za pomocą kombinacji magicznych praktyk, wspaniałych narzędzi i — nierzadko — wysokich faktur wystawianych przez firmy konsultingowe.

Jedna z różnic między zespołami, które stosując podejście zwinne, osiągają przeciętne i dobre wyniki, polega na tym, że te drugie nie traktują podejście zwinnego jak menu z technikami do wyboru. Kluczem do stosowania wszystkich tych praktyk jednocześnie jest nastawienie, z którym zespół przystępuje do projektu. To nastawienie wynika z wartości i zasad podejścia zwinnego.

Podejście zwinne jest różne od innych metodyk, ponieważ punktem wyjścia są tu wartości i zasady. Zespół korzystający z tego podejścia musi uczciwie przyjrzeć się nie tylko sposobom budowania oprogramowania, ale też interakcjom między jego członkami i z innymi osobami z firmy. Zespół zwinny dzięki temu, że najpierw poznaje zasady, a dopiero potem stosuje metodykę (wiedząc, że będzie to wymagać dużo pracy, analiz i poprawek), może realistycznie liczyć na znalezienie lepszych sposobów prowadzenia projektów. Zapewnia to konkretną drogę do większej zwinności oraz pozwala rozwijać i dostarczać lepsze oprogramowanie.

Często zadawane pytania

Jestem prawdziwą gwiazdą programowania. Chcę tylko, żeby ludzie nie przeszkadzali mi w pisaniu świetnego oprogramowania! Dlaczego mam przejmować się takimi rzeczami jak tablice zadań czy wykres postępu prac?

Wszystkim dobrym programistom zdarzało się poczuć frustrację, gdy po zbudowaniu świetnego fragmentu kodu musieli znacznie go zmodyfikować i poprawić, ponieważ osoba, która nie wiedziała nic o strukturze kodu, w ostatniej chwili zażądała zmian. Dla kogoś, komu naprawdę zależy na jakości kodu, bardzo frustrujące jest godzenie się na niepotrzebne kompromisy techniczne, bo jakiś nieprogramista dopiero w połowie projektu stwierdził, że potrzebuje danej funkcji.

To dlatego tak wielu świetnych programistów chce pracować w zespołach zwinnych. To prawda, wymaga to planowania i przyzwyczajenia się do stosowania związanych z tym narzędzi, na przykład tablic zadań i wykresów postępu prac (ang. *burndown chart*). Metodyka zwinna jest oparta na takich technikach. Zostały one specjalnie wybrane, ponieważ są proste i ograniczone do minimum niezbędnego do efektywnego planowania i prowadzenia projektu. Ważną korzyścią uzyskiwaną, gdy jesteś zaangażowany w planowanie projektu (i gdy ma to dla Ciebie znaczenie), jest możliwość uniknięcia frustrujących zmian wprowadzanych w ostatniej chwili. W tym celu wystarczy zadać trudne pytania, aby uzyskać niezbędne odpowiedzi już na etapie planowania projektu. Sprawdza się to tylko dzięki temu, że skuteczne zespoły zwinne od początku projektu cały czas komunikują się z użytkownikami. Wielu dobrych programistów przekonuje się do planowania w momencie, gdy po raz pierwszy zadają użytkownikom trudne

Kup ksi k Pole ksi k

pytania prowadzące do zmian, które w innym podejściu zostałyby odkryte na późnym etapie projektu. Możliwość uniknięcia irytujących poprawek na ostatnią chwilę, wymagających usuwania i poprawiania kodu, okazuje się świetną zachętą do stosowania podejścia zwinnego.

W planowaniu w podejściu zwinnym ważne jest też komunikowanie się z resztą zespołu. Pomaga to świetnym programistom w rozwoju. „Gwiazdy" programowania cały czas się uczą, jednak w zespołach o nastawieniu „dowódź i kontroluj" są odizolowane od pozostałych członków grupy, dlatego w większości samodzielnie określają poznawane zagadnienia. Natomiast zespołach samoorganizujących się komunikacja jest bardziej nasilona. Nie są to jednak nieskończone, bezsensowne spotkania poświęcone stanowi projektu, w których programiści nie znoszą uczestniczyć. Zamiast tego *członkowie zespołu sami decydują, o czym chcą porozmawiać, aby projekt przebiegał właściwie*. Nie tylko prowadzi to do lepszych projektów, ale też oznacza, że jeśli programista siedzący obok Ciebie wie coś, czego możesz się od niego nauczyć, prawdopodobnie do momentu zakończenia prac również będziesz to wiedział. Na przykład jeżeli inny członek zespołu stosuje nowy wzorzec projektowy w nieznany Ci wcześniej sposób, zobaczysz, czy okaże się to dobrym pomysłem. Jeśli tak, będziesz mógł dodać dany wzorzec do swojego przybornika. W ten sposób uczysz się automatycznie, bez wkładania w to dodatkowego wysiłku, gdyż cały zespół skutecznie się komunikuje. Jest to jeden z powodów, dla których osoby stosujące podejście zwinne rozwijają swoje umiejętności techniczne i mają poczucie, że stają się lepszymi programistami.

Jestem kierownikiem projektu i nie mam pewności, jak ma wyglądać moja praca w zespole zwinnym. Jaka jest w nim moja rola?

Jeśli jesteś kierownikiem projektu, prawdopodobnie odgrywasz jedną z trzech tradycyjnych ról:

- Jesteś planistą, który dokonuje szacunków, tworzy harmonogramy projektu i kieruje codzienną pracą zespołu.
- Jesteś ekspertem od produktu i przypuszczalnie pełnisz funkcję analityka biznesowego, który ustala wymagania, przekazuje je zespołowi i dba o to, by zbudował zgodne z nimi oprogramowanie.
- Jesteś nadzorcą, który współpracuje ze starszymi menedżerami i dyrektorami z firmy oraz informuje ich o tym, jaki zwrot dają ponoszone przez nich inwestycje w projekt.

W rozdziale 4. poznasz Scruma, najczęściej stosowaną metodykę zwinną, a także role w korzystających z niej zespołach. Jeśli jesteś kierownikiem, który uczestniczy w pracach zespołu, prawdopodobnie będziesz pełnił funkcję Mistrza Młyna. Jego zadania polegają na pomocy zespołowi w planowaniu i likwidowaniu przeszkód w projekcie, tak aby mógł dostarczyć oprogramowanie. Jeżeli starasz się poznawać potrzeby firmy i przekazywać je zespołowi, prawdopodobnie będziesz Właścicielem Produktu. W tej sytuacji będziesz zarządzał rejestrem zadań, decydował, które funkcje należy dodać w poszczególnych iteracjach, a także odpowiadał na szczegółowe pytania zespołu, tak żeby zachował właściwy kierunek prac i zbudował odpowiednie oprogramowanie.

Jeśli jesteś kierownikiem projektu pełniącym funkcje nadzorcze, prawdopodobnie nie znajdziesz się w zespole zwinnym. Nie ma w tym nic złego. Zamiast tego będziesz odgrywał jedną z najważniejszych ról — będziesz promotorem podejścia zwinnego, zachęcającym zespoły

Kup ksi k Pole ksi k

oraz menedżerów do stosowania praktyk zwinnych i przestrzegania wartości związanych z tym podejściem. Zauważysz, że gdy masz rejestr zadań podzielonych na iteracje i szczegółowy opis bieżącej iteracji, natychmiast otrzymujesz informacje potrzebne dyrektorom i starszym menedżerom. A im lepiej zespół potrafi ocenić postępy prac i stopień realizacji celów projektowych, tym bardziej realistyczne informacje może Ci przekazać. Jednak aby współpraca była efektywna, musisz dobrze zrozumieć sposób pracy zespołów zwinnych. Dzięki temu będziesz mógł używać ich języka i przetwarzać uzyskane informacje na postać zrozumiałą dla dyrektorów.

Chwileczkę — skoro cały zespół uczestniczy w planowaniu, to czy nikt nim nie kieruje? To bardzo niepraktyczne. W jaki sposób podejmowane są decyzje?

To zależy od samych decyzji. Jeśli masz na myśli rozwiązywanie konfliktów, powinno się to odbywać dokładnie tak samo jak w zespole, w którym obecnie pracujesz. Pomyśl o nim lub o ostatnim zespole projektowym, którego byłeś członkiem. Kto nim kierował? Kto rozwiązywał spory, z którymi sami członkowie zespołu nie mogli sobie poradzić? Kto oceniał waszą pracę? Hierarchia może powstawać na wiele sposobów. Zespół zwinny powinien umieć funkcjonować w ramach dowolnej hierarchii. Jednak zespoły zwinne zwykle lepiej radzą sobie z samodzielnym rozwiązywaniem konfliktów, ponieważ koncentrują się na wzajemnej komunikacji i zależy im na tych samych celach w większym stopniu niż w przypadku innych zespołów.

Jeśli jednak pytasz o to, kto decyduje, które funkcje znajdą się w oprogramowaniu i jak zostaną zbudowane, to zazwyczaj odpowiadają za to osoby odgrywające określone role w zespole zwinnym. W zespole stosującym podejście Scrum to właściciel produktu ustala, która funkcje należy dodać. Zespół powinien przy tym zaakceptować tylko te funkcje, które można wykonać w ramach danej iteracji. Więcej na ten temat dowiesz się z rozdziału 4. Plan należy jednak do wszystkich, ponieważ tak działają zespoły samoorganizujące się.

Ale samo to, że właścicielem planu jest cały zespół, nie oznacza jeszcze, że nikt nie jest szefem. *Oczywiście, że jest przełożony*. Jeśli dopiero zaczynasz stosować podejście zwinne, obecny przełożony prawdopodobnie będzie nim także za rok. Różnica polega na tym, że szef na tyle wierzy w podejście zwinne, że pozwala zespołowi podejmować decyzje związane z projektem i je popiera — nie podważa ich ani nie kontroluje grupy na nadmiernie szczegółowym poziomie. Jest to jedyne nastawienie sprawdzające się w praktyce.

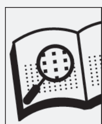
Co możesz zrobić już dziś?

Oto kilka pomysłów, które możesz wypróbować już dziś (samodzielnie lub razem z zespołem):

- Jeśli pracujesz nad projektem, usiądź razem z zespołem przed rozpoczęciem pracy nad kodem i przez piętnaście minut porozmawiajcie o funkcjach, które zamierzacie dodać. Czy zdarzają się sytuacje, w których dwie osoby w różny sposób myślą o tym, co należy zbudować?
- Zapisz listę funkcji, nad którymi obecnie pracujesz. Spróbuj uporządkować je według wartości i trudności.

Kup ksi k Pole ksi k

- Poświęć kilka minut na zapisanie listy wszystkich dokumentów, które razem z zespołem piszecie lub stosujecie. Czy potrafisz wykryć pozycje, których zespół nie używa przy pisaniu kodu?
- Gdy następnym razem będziesz pracował do późna, pomyśl o tym, co było tego przyczyną. Czy potrafisz wymyślić sposób na uniknięcie takich sytuacji? Czy termin był zbyt agresywny? A może w ostatniej chwili dodano dodatkowe zadania? Uznanie, że nadgodziny stanowią problem, i zrozumienie przyczyn tego problemu to pierwszy krok do naprawienia sytuacji.



Skąd czerpać dodatkową wiedzę?

Oto materiały, które pomogą Ci dowiedzieć się czegoś więcej o zagadnieniach opisanych w tym rozdziale:

- Więcej o wartościach i zasadach z manifestu Agile oraz o jego powstaniu dowiesz się z książki Alistaira Cockburna *Agile Software Development: The Cooperative Game, 2nd Edition* (wydawnictwo Addison-Wesley, 2006).
- Dodatkowe informacje o wartości, iteracjach i innych aspektach zarządzania projektami w metodykach zwinnych znajdziesz w książce Jima Highsmitha *Agile Project Management: Creating Innovative Products, 2nd Edition* (wydawnictwo Addison-Wesley, 2009).
- Więcej o wyzwaniach czekających na zespoły wprowadzające podejście zwinne i ich przezwyciężaniu dowiesz się z książki Mike'a Cohna *Succeeding with Agile* (wydawnictwo Addison-Wesley, 2009).
- Dodatkową wiedzę na temat radzenia sobie z nastawieniem „dowódź i kontroluj" znajdziesz w książce Lyssy Adkins *Coaching Agile Teams* (wydawnictwo Addison-Wesley, 2010).

Wskazówki dla coachów

Oto wskazówki dla coachów metodyk zwinnych, którzy pomagają zespołom przyswoić sobie wiedzę z tego rozdziału:

- Pomóż członkom zespołu zauważyć, że praca w nadgodzinach powoduje, iż piszą mniej kodu, a nie więcej, a ponadto jest on niższej jakości.
- Porozmawiaj z poszczególnymi członkami zespołu o ich pracy. Co ich motywuje? Co frustruje? Na jakiej podstawie podejmują decyzje?
- Poproś poszczególnych członków zespołu o to, aby wybrali trzy zasady podejścia zwinnego, które wywarły na nich największy wpływ — pozytywny lub negatywny. Prawdopodobnie będą oni zaskoczeni tym, że współpracownicy wskazali inne reguły. Pomoże to też ustalić, które zasady są ważne dla wszystkich.
- Wykorzystaj ważne dla wszystkich zasady jako punkt wyjścia do ustalenia, które techniki najlepiej pasują do charakteru zespołu.

Kup ksi k Pole ksi k

Skorowidz

A

adaptacja, 109
Agile, 16
akceptacja odpowiedzialności, 192
aktualizacje, 106
antywzorce, 206, 216
architekt, 28

B

bardzo duże klasy, 209
bezlitosna refaktoryzacja, 219
biblioteka, 213
BRUF, 29
budowanie
  funkcji, 136
  platformy, 212
  właściwego nastawienia, 183
bufor, 175, 236, 319

C

cel stosowania poszczególnych technik, 42
chirurgia odłamkowa, 209, 216
ciągła integracja, 169, 175, 220
ciągły wyścig, 287
coach metodyk zwinnych, 53, 333, 347
codzienne spotkania, 35, 107, 112
codzienne spotkania na stojąco, 43, 50
cykl widoczność – kontrola – adaptacja, 107
czas
  przebywania elementów, 307
  realizacji zamówienia, 266
członek zespołu, 87, 101
czytnik e-booków, 66

D

dług techniczny, 219, 236
dokumentacja, 43
dokumentacja nadmiernie rozbudowana, 76
doskonalenie procesu, 294
dostarczanie, 269
  działającego oprogramowania, 79
  projektu, 61, 66
dostrzeganie
  całości, 265
  marnotrawstwa, 261
dwanaście zasad, 58
dziesięciominutowy proces budowania, 169, 175

E

ekonomia, 192
eliminowanie marnotrawstwa, 257
energiczna praca, 226, 236
etap
  ha, 347
  ri, 347
  shu, 347
ewolucja eksperymentalna, 289

F

FIFO, 269
funkcja MMF, 261, 271, 280

H

haczyk, 211
historie użytkowników, 35, 38, 42, 54, 134, 136
historyjka o świni i kurczaku, 95

I

informacje zwrotne, 107, 183, 185, 193
innowacje, 228
inspirujące cele, 121
integralność
  koncepcyjna, 264
  postrzegana, 263
integrowanie, 174
integrowanie kodu, 169
iteracje, 35, 51, 67

J

jakość, 193
jednostki robocze, 292

K

Kanban, 51, 285, 290, 292, 322
kierownik projektu, 17, 28

Kup ksi k Pole ksi k

kod, 205
  lasagne, 209
  niesamodzielny, 209
  powtarzający się, 209
  spaghetti, 209
  wielokrotnego użytku, 213
kolejka, 269, 302
komunikacja, 68, 183
komunikacja przez osmozę, 172, 175
koncentracja, 154
kontrakt, 232
kontrola
  pracy, 107
  zadań, 109
koszt opóźnienia, 276
kreowanie herosów, 255
kryzys oprogramowania, 37

L

Lean, 51
lepsza komunikacja, 42
lider zespołu, 28
likwidowanie długu technicznego, 219
limit prac w toku, 300, 319

Ł

łączenie wszystkich zasad, 83

M

Manifest Agile, 42, 47
mapa
  przepływu pracy, 293
  strumienia wartości, 261, 262, 280
marnotrawstwo, 245, 257
metoda „dowódź i kontroluj”, 110
metodyka, 31
  Lean, 51
metodyki zwinne, 15, 49, 53
Mistrz Młyna, 50, 87, 93, 98
MMF, minimal marketable feature, 261
model
  iteracyjny, 117
  kaskadowy, 27, 29, 32
  kaskscrumowy, 41
  przyrostowy, 117
  Shuhari, 342, 347
monolityczne projekty, 222
muda, 280
mura, 280
muri, 280
myślenie
  magiczne, 255
  odchudzone, 246
  systemowe, 291
  w kategoriach opcji, 253

N

narzędzia, 43
negocjowanie kontraktu, 44
niepowodzenia, 192
niespójna perspektywa, 37, 39, 41
nieustanne doskonalenie, 285

O

odwaga, 154, 183, 185
ogólnie przyjęte
  praktyki, 134
  praktyki scrumowe, 147
ograniczanie
  liczby zadań, 300
  zakresu prac, 275
opcje, 248
opowiadania, 22
oprogramowanie, 43
otwartość, 154
otwartość na zmiany, 165, 174

P

pętla informacji zwrotnych, 109
pętle z informacjami zwrotnymi, 193, 290
Planning Poker, 159
planowanie, 98, 170, 174, 190
  pracy, 110
  sprintu, 87, 122, 144
  w Scrumie, 131
  zespołowe, 42
plany wersji, 35
platforma, 213
platformy wielokrotnego użytku, 213
podejmowanie decyzji, 217
podejście
  iteracyjne, 116
  Kanban, 288, 294
  przyrostowe, 116
  Scrum, 35
  Water-Scrum-Fall, 41
  zwinne, 27, 33, 58
podział zadań, 214
polityki, 289
polityki obowiązujące w procesie, 321, 324
pomiar przepływu, 306, 307
porażka, 344
powiązania, 223
pozłacanie, 136
pracownicy biznesowi, 71
praktyki scrumowe, 147
prawo Little’a, 310, 314, 324
problemy w projekcie, 39
proces uczenia się, 339
procesy, 43
programista, 18
programowanie, 174
  ekstremalne, XP, 16, 51, 167
  oparte na opcjach, 248, 252
  sterowane testami, 35, 168, 175
  w parach, 169
projekt, 205
  emergentny, 233
  monolityczny, 222
projektowanie
  przyrostowe, 203, 224, 230
  z myślą o wielokrotnym użyciu, 230
prosta
  interakcja, 233
  komunikacja, 232
prostota, 183, 185
prostota rozwiązań, 82
przebieg projektu, 76
przecieki, 209

Kup ksi k Pole ksi k

przegląd sprintu, 88
przeładowanie, 304
przepływ, 194, 285, 306
przyczyny problemów, 30
przypadki brzegowe, 210
przypisywanie punktów historiom, 140
pułapki budowania platformy, 212
punkt wyjścia, 288

R

radiator informacji, 175
reagowanie na zmiany, 45
realizowanie planu, 45
refaktoryzacja, 217
reguła
  KISS, 80
  tworzenia harmonogramu, 111
rejestr zadań
  produktu, 50
  sprintu, 50
retrospekcja, 35, 43, 88
role, 87
rozumienie
  planowania, 190
  procesu uczenia się, 339
  produktu, 262
  technik, 192
rozwiązania uniwersalne, 31
rytm dostarczania, 319

S

samoorganizujące się zespoły, 82, 87
Scrum, 35, 49, 87, 91
serwer budowania, 35
skumulowany wykres przepływu, 307, 311, 324
skuteczne codzienne spotkania, 112
specyfikacja, 29
spotkanie codzienne, 17
sprint, 50, 87, 89, 115
stałe tempo pracy, 227
sukces sprintu, 118
sygnały ostrzegawcze, 337
system pull, 276
szacunek, 183, 185
szybkość pracy w projekcie, 139

Ś

śledzenie postępów, 46
środowisko izolowane, 169

T

tablica
  kanbanowa, 295
  zadań, 35, 46, 47
TDD, test-driven development, 168
techniki, 45, 151
  holistyczne, 224
  kierownika projektu, 41
  lidera zespołu, 41
  programisty, 41
  właściciela produktu, 41
  XP, 167
  związane z integrowaniem kodu, 169
  związane z planowaniem, 170, 228
  związane z programowaniem, 168
  związane z projektowaniem, 228
  związane z zespołem, 171, 228
  zwinne, 41, 42
tempo przybywania, 307
teoria
  kolejek, 269
  ograniczeń, 275
testy
  A/B, 252
  integracyjne, 233
  jednostkowe, 168
trudności z wprowadzaniem zmian, 337
tworzenie harmonogramu, 111

U

unikanie marnotrawstwa, 245
uprawnienia zespołu, 253
ustalanie ram czasowych, 51

W

wartości
  Agile, 27
  metodyki, 342
  Scruma, 102, 150, 153
  XP, 178, 183
wartość, 119
  historii, 139
  komunikacji, 195
  prostoty, 195
warunki satysfakcji, 137
wbudowywanie integralności, 263
widoczność, 108, 119
wizualizowanie przepływu pracy, 295
Właściciel Produktu, 35, 47, 87, 94, 101
wprowadzenie podejścia zwinnego, 34
wspomaganie uczenia się, 248
wspólne
  decyzje, 228
  doskonalenie, 289
  siedzenie, 171
  zobowiązanie, 101, 131, 135
współpraca, 68
  z klientem, 44
  z użytkownikami, 179
wykorzystanie
  historii, 144
  punktów, 144
  tablicy zadań, 144
  zadań, 144
wykres
  postępu prac, 35, 141
  prac w toku, 307, 316
  przepływu, 307, 311, 324
wykrywanie problemów, 220
wymagania, 29

Kup ksi k   Pole ksi k

X

XP, eXtreme Programming, 51, 49, 165
holistyczne techniki, 224
podstawowe techniki, 167
projektowanie przyrostowe, 203
rozumienie zasad, 188
wartości, 183
zasady, 189

Z

zachowania emergentne, 322
zakres prac, 275
zapach kodu, 208, 215
zapobieganie zaskoczeniu, 134
zarządzanie
przepływem, 306, 311, 319
rejestrem zadań produktu, 87
zasada, 45
częste dostarczanie oprogramowania, 64
miara postępów, 76
osoby zmotywowane, 73
otwartość na zmiany wymagań, 62
prostota, 80
regularne zwiększanie efektywności, 82
rozmowa bezpośrednia, 70
stałe tempo pracy, 77
techniczna doskonałość, 78
współpraca, 71
zapewnianie satysfakcji klientów, 61
zespół samoorganizujący się, 81
zasady
Agile, 57
coachingu, 345
podejścia Scrum, 89
podejścia zwinnego, 58
XP, 188, 189
zator, 275
zautomatyzowany skrypt budowania, 35
zespół, 174
zespół samoorganizujący się, 81, 100
zestaw zasad, 188
złożoność, 215
zmiany, 172, 288
zobowiązanie, 94, 153
zwinny słoń, 48

Kup ksi k Pole ksi k

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

http://program-partnerski.helion.pl

O'REILLY®

Sprawdź, jak wykorzystać siłę zwinności!

W XXI wieku ogromnie wzrosło tempo rozwoju cyfrowych usług. Tradycyjne sposoby wytwarzania oprogramowania nie są już w stanie nadążyć za oczekiwaniami klientów. Dziś nikt nie będzie czekał, aż dopracujesz wszystkie zaplanowane funkcje i wypuścisz produkt na rynek. Konkurencja Cię wyprzedzi! Czy dasz się jej pokonać? Odpowiedzią na to pytanie jest słowo, które robi furorę w branży IT: Agile. Zwinne wytwarzanie oprogramowania to przyszłość — przekonaj się sam!

W ramach Agile istnieje wiele podejść do wytwarzania oprogramowania. Jeżeli zastanawiasz się nad tym, które najlepiej zadziała w Twoim środowisku, trafiłeś na odpowiednią książkę. Znajdziesz w niej omówienie ścieżek takich jak Scrum, Lean, Kanban oraz XP (ang. eXtreme Programming), ale najpierw poczytasz trochę na temat tego, czym jest zwinność i do czego możesz ją wykorzystać. Poznasz zalety i zagrożenia związane z konkretnymi podejściami oraz obszary, w których każde z nich sprawdzi się najlepiej. Ta książka pozwoli Ci zmienić sposób pracy. Dzięki temu masz szansę stworzyć wyjątkowo produktywny zespół, którego nikt nie zatrzyma. Bądź zwinny!

Dzięki tej książce:

- zrozumiesz zwinne podejście do wytwarzania oprogramowania
- zobaczysz, jak można wdrożyć je w życie
- dowiesz się, jak działają Scrum, Kanban oraz Lean
- dowiesz się, w jakich miejscach sprawdzi się podejście XP
- zbudujesz wydajny i zwinny zespół

Andrew Stellman — programista, architekt, menedżer projektów, Agile Coach. Ma ponad 20-letnie doświadczenie w budowaniu zaawansowanych systemów informatycznych. Zarządzał międzynarodowymi zespołami oraz doradzał organizacjom takim jak Microsoft, Bank of America oraz MIT.

Jennifer Greene — analityk biznesowy, Agile Coach, testerka oraz prelegentka. Ma ogromne doświadczenie w zakresie tworzenia oprogramowania, potwierdzone ponad 20-letnią praktyką w różnych obszarach. W swojej karierze rozwiązywała skomplikowane problemy we współpracy ze znakomitymi programistami.

36563 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 978-83-283-0940-1

9 788328 309401

Informatyka w najlepszym wydaniu

cena: 59,00 zł