



Gojko Adzic

SPECYFIKACJA NA PRZYKŁADACH

Poznaj zwinne metody pracy
i dostarczaj właściwe oprogramowanie

Skutecznie zbieraj wymagania!

Tytuł oryginału: Specification by Example: How Successful Teams Deliver the Right Software

Tłumaczenie: Arkadiusz Romanek

ISBN: 978-83-246-9118-0

Original edition copyright © 2011 by Manning Publications Co.
All rights reserved.

Polish edition copyright © 2014 by HELION SA.
All rights reserved.

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 jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

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

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/speprz>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Wprowadzenie	11
Podziękowania	22
O autorze	23
O ilustracji na okładce	24

CZEŚĆ I Zaczynamy!

1 Kluczowe korzyści 27

Sprawniejsze wprowadzanie zmian	30
Wyższa jakość produktu	32
Mniej przeróbek	36
Lepsze dostosowanie aktywności	39
Pamiętaj	41

2 Wzorce kluczowych procesów 43

Zdefiniowanie zakresu prac w oparciu o cele biznesowe	45
Wspólne specyfikowanie	45
Opisywanie z wykorzystaniem przykładów ilustrujących	46
Udoskonalanie specyfikacji	47
Automatyzacja walidacji bez zmiany specyfikacji	47
Częsta walidacja	49
Tworzenie systemu dokumentacji	49
Praktyczny przykład	50
Cel biznesowy	50
Przykład poprawnego celu biznesowego	51
Zakres	51
Historyjki użytkowników podstawowego elementu systemu lojalnościowego	51
Kluczowe przykłady	51
Kluczowe przykłady: Darmowa dostawa	52
Specyfikacja z przykładami	52
Darmowa dostawa	52
Przykłady	53
Wykonywalna specyfikacja	53
Żyjąca dokumentacja	53
Pamiętaj	54

3 Żyjąca dokumentacja 55

Dlaczego potrzebujemy pewnej dokumentacji?	56
Testy mogą być dobrą dokumentacją	57
Tworzenie dokumentacji na podstawie wykonywalnej specyfikacji	58
Zalety modelu zorientowanego na dokumentację	60
Pamiętaj	61

4 Inicjowanie zmian 63

Jak rozpocząć zmianę procesu?	64
Wdrażaj specyfikację przez przykłady jako część rozległego procesu zmian	65
Kiedy: W projektach typu greenfield	65
Skup się na poprawie jakości	65
Zacznij od automatyzacji testów funkcjonalnych	66
Kiedy: Zmiany dotyczą istniejącego projektu	66
Wprowadź narzędzie do wykonywalnych specyfikacji	68
Kiedy: Zależnie od potrzeb własnych testerów	68
Wykorzystaj TDD jako odskocznię	70
Kiedy: Deweloperzy mają dużą wiedzę na temat TDD	70
Jak zacząć zmieniać kulturę zespołu?	70
Unikaj używania terminów sugerujących zwinność lub bycie „agile”	70
Kiedy: Pracujesz w środowisku opornym na zmiany	70
Zadbaj o uzyskanie wsparcia kierownictwa	72
Sprzedaj specyfikację przez przykłady jako lepszą metodę wykonywania testów akceptacyjnych	73
Niech automatyzacja testów nie będzie celem końcowym	74
Nie koncentruj się wyłącznie na narzędziu	75
W czasie migracji niech jedna osoba ciągle pracuje nad starszymi skryptami	75
Kiedy: Wprowadzasz automatyzację funkcjonalną do istniejących systemów	75
Sprawdź, kto wykonuje testy automatyczne	76
Kiedy: Deweloperzy niechętnie podchodzą do uczestnictwa w procesie	76
Jak zespoły wdrażają zasady współpracy w procesach iteracyjnych i przepływu?	77
Zespół Global Talent Management z Ultimate Software	77
Zespół Sierra w BNP Paribas	80
Sky Network Services	81
Radzenie sobie z potrzebą formalnego zatwierdzenia i identyfikowalnością	82
Zachowaj wykonywalne specyfikacje w systemie kontroli wersji	83
Uzyskaj zatwierdzenie na eksportowanej żyjącej dokumentacji	84
Kiedy: Zatwierdzasz iterację po iteracji	84
Uzyskaj zatwierdzenie zakresu, a nie specyfikacji	84
Kiedy: Zatwierdzasz odleglejsze kamienie milowe	84
Uzyskaj zatwierdzenie „odchudzonych” przypadków użycia	85
Kiedy: Formalne zatwierdzenia wymagają uzupełnienia o szczegóły	85
Wprowadź realizację przypadków użycia	86
Kiedy: Formalne zatwierdzenia wymagają uwzględniania wszystkich szczegółów	86
Znaki ostrzegawcze	87
Uważaj na testy, które często dają różne wyniki	87
Uważaj na bumerangi	88
Uważaj na niedopasowanie organizacyjne	88
Uważaj na kod „na wszelki wypadek”	89
Uważaj na „chirurgię śrutówką”	90
Pamiętaj	90

CZĘŚĆ II Wzorce kluczowych procesów

5 Definiowanie zakresu na podstawie celów 93

Określanie odpowiedniego zakresu	95
Znajdź odpowiedzi na pytania „Dlaczego?” i „Kto?”	96
Zrozum, skąd bierze się wartość	98
Dowiedz się, jakich wyników oczekują użytkownicy biznesowi	99
Niech deweloperzy zapewnią część „chcę” historyjek użytkownika	100
Kiedy: Użytkownicy biznesowi ufają zespołowi zajmującemu się wytwarzaniem oprogramowania	100
Współpraca w celu zdefiniowania zakresu bez kontroli wysokiego poziomu	101
Zapytaj o to, jak coś może być przydatne	102
Zapytaj o rozwiązanie alternatywne	103
Nie patrz na projekt wyłącznie z perspektywy najniższego poziomu	103
Zadbaj, aby zespoły dostarczały kompletne funkcje	104
Kiedy: Pracujesz nad dużymi projektami z częściami zespołów w różnych lokalizacjach	104
Więcej informacji	105
Pamiętaj	106

6 Wspólne specyfikowanie 107

Dlaczego podczas definiowania specyfikacji musimy ze sobą współpracować?	108
Najpopularniejsze modele współpracy	109
Spróbuj zorganizować duże warsztaty dla wszystkich członków zespołu	109
Kiedy: Zaczynasz wdrażać zasady specyfikacji przez przykłady	109
Wypróbuj spotkania w mniejszym gronie („trzej amigos”)	111
Kiedy: Domena wymaga składania częstych wyjaśnień	111
Programujcie w parach	113
Kiedy: Pracujecie nad dojrzałymi produktami	113
Spraw, aby testerzy przed iteracją regularnie sprawdzali testy	115
Kiedy: Analitycy tworzą testy	115
Spróbuj nieformalnych rozmów	115
Kiedy: Interesariusze biznesowi są łatwo dostępni	115
Przygotowanie współpracy	116
Organizuj spotkania przygotowawcze	117
Kiedy: W projekcie uczestniczy wielu interesariuszy	117
Zdobądź zaangażowanie interesariuszy	118
Dobrze przygotuj się do wstępnych spotkań z interesariuszami	119
Kiedy: Interesariusze nie są dostępni na miejscu	119
Niech członkowie zespołu przejrzą historyjki na wczesnym etapie	121
Kiedy: Analitycy/eksperci ds. domeny są wąskim gardłem procesu	121
Przygotuj tylko wstępne przykłady	122
Kiedy: Interesariusze są łatwo dostępni	122
Nie utrudniaj dyskusji przez przesadne przygotowania	123
Wybór modelu współpracy	124
Pamiętaj	125

7 Wykorzystanie przykładów ilustrujących 127

Uzupełnienie specyfikacji z wykorzystaniem przykładów ilustrujących: przykład	130
Przykłady powinny być precyzyjne	131
Nie używaj w swoich przykładach systemu zamkniętych odpowiedzi (tak/nie)	131
Kiedy: Bazowe koncepcje nie są definiowane osobno	131
Unikaj używania abstrakcyjnych klas równoważności	132
Kiedy: Możesz zdefiniować konkretny przykład	132
Przykłady powinny być kompletne	133
Eksperymentuj z danymi	133
Pytaj, czy istnieje alternatywna metoda sprawdzenia funkcjonalności	133
Kiedy: Pracujesz ze złożoną/starą infrastrukturą	133
Przykłady powinny być realistyczne	134
Unikaj generowania zmyślonych danych	134
Kiedy: Podczas prac nad projektem opartym na danych	134
Pozyskaj podstawowe przykłady bezpośrednio od klientów	135
Kiedy: Pracujesz dla klientów korporacyjnych	135
Przykłady powinny być zrozumiałe	137
Unikaj pokusy zbadania wszelkich możliwych kombinacji	138
Szukaj ukrytych koncepcji	138
Ilustrowanie wymagań niefunkcjonalnych	140
Zdobądź precyzyjne wymagania wydajnościowe	140
Kiedy: Wysoka wydajność jest funkcją kluczową	140
Wykorzystaj uproszczone prototypy interfejsów użytkownika	141
Wypróbuj model QUPER	142
Kiedy: Wymagania zmieniają się i są skalowalne	142
Wykorzystaj listę kontrolną podczas dyskusji	143
Kiedy: Pojawiają się obawy przekrojowe	143
Stwórz przykład referencyjny	144
Kiedy: Wymagania są niemożliwe do oszacowania	144
Pamiętaj	145

8 Udoskonalanie specyfikacji 147

Przykład dobrej specyfikacji	149
Darmowa dostawa	149
Przykłady	149
Przykład złej specyfikacji	150
Na co należy zwrócić uwagę podczas udoskonalania specyfikacji?	152
Przykłady powinny być precyzyjne i testowalne	152
Skrypty to nie specyfikacje	152
Nie twórz opisów w formie przepływów	154
Specyfikacje powinny dotyczyć funkcjonalności biznesowej, a nie projektu oprogramowania	154
Unikaj tworzenia specyfikacji, które są ściśle powiązane z kodem	155
Oprzyj się pokusie obejścia trudności technicznych w specyfikacjach	156
Kiedy: Pracujesz na starym systemie	156
Nie pozwól uwieźć się przez szczegóły interfejsu użytkownika	157
Kiedy: Pracujesz nad projektami internetowymi	157
Specyfikacje powinny być czytywne	157
Użyj opisowego tytułu i wyjaśnij cel, stosując krótkie zdania	158
Pokaż i milcz	158
Kiedy: Ktoś pracuje nad specyfikacją samodzielnie	158
W celu: Sprawdzenia, czy specyfikacja jest czytywa i nie wymaga dodatkowych tłumaczeń	158

Nie upraszczaj nadmiernie przykładów	159
Zaczynij od podstawowych przykładów, a następnie rozszerz zakres przez eksplorowanie	161
Kiedy: Opisujesz reguły za pomocą wielu kombinacji parametrów	161
Specyfikacje powinny być ostre	161
Zastosuj wzorzec „Zakładając/Jeżeli/Wtedy”	162
W celu: Sprawienia, by testy były łatwiejsze do zrozumienia	162
Nie definiuj jawnie wszystkich zależności w specyfikacji	163
Kiedy: Musisz poradzić sobie ze skomplikowanymi zależnościami/integralnością referencyjną	163
Zastosuj ustawienia domyślne w warstwie automatyzacji	164
Nie polegaj na domyślnych wartościach w każdym przypadku	164
Kiedy: Pracujesz z obiektami o wielu atrybutach	164
Specyfikacje powinny być napisane w języku domeny	165
Udoskonalanie specyfikacji w praktyce	165
Pamiętaj	168

9 Automatyczna walidacja bez zmiany specyfikacji 169

Czy automatyzacja jest w ogóle potrzebna?	170
Rozpoczęcie automatyzacji	172
Aby poznać narzędzia, wypróbuj je najpierw w prostym projekcie	172
Kiedy: Pracujesz z istniejącym systemem	172
Zaplanuj automatyzację z wyprzedzeniem	173
Nie opóźniaj i nie odsuwaj od siebie prac związanych z automatyzacją	175
Unikaj automatyzacji istniejących skryptów testów ręcznych	175
Zdobądź zaufanie dzięki testom interfejsu użytkownika	176
Kiedy: Członkowie zespołu są nastawieni sceptycznie do wykonywalnych specyfikacji	176
Zarządzanie warstwą automatyzacji	178
Nie traktuj kodu automatyzacji jak kodu drugiej kategorii	178
Opisz procesy walidacji w warstwie automatyzacji	179
Nie powielaj logiki biznesowej w warstwie automatyzacji testów	180
Automatyzuj wzdłuż granic systemu	181
Kiedy: Integracje są skomplikowane	181
Nie sprawdzaj logiki biznesowej za pomocą interfejsu użytkownika	182
Automatyzacja pod skórą aplikacji	183
Kiedy: Sprawdzasz ograniczenia sesji i przepływu	183
Automatyzacja interfejsów użytkownika	184
Określ funkcjonalność interfejsu użytkownika na wyższym poziomie abstrakcji	186
Funkcjonalność interfejsu użytkownika sprawdzaj tylko ze specyfikacją interfejsu użytkownika	188
Kiedy: Interfejs użytkownika zawiera złożoną logikę	188
Unikaj zarejestrowanych testów interfejsu użytkownika	188
Ustaw kontekst w bazie danych	189
Zarządzanie danymi testowymi	191
Unikaj wykorzystywania danych wstępnie wypełnionych	191
Kiedy: Definiujesz logikę, która nie bazuje na danych	191
Spróbuj wykorzystać wstępnie przygotowane dane referencyjne	192
Kiedy: W systemach bazujących na danych	192
Wyciągnij prototypy z bazy danych	193
Kiedy: W starych, dojrziałych systemach bazujących na danych	193
Pamiętaj	194

10 Częsta walidacja 195

Zmniejszenie zawadności	197
Znajdź najbardziej irytujący Cię element, napraw go, a następnie powtórz całą operację	198
Kiedy: Pracujesz w systemie z kiepskim wsparciem dla testów automatycznych	198
Określ niestabilne testy, korzystając z historii testów ciągłej integracji	199
Kiedy: Modernizujesz i dopasowujesz automatyczne testy do starego systemu	199
Utwórz dedykowane środowisko ciągłej walidacji	199
Zastosuj w pełni zautomatyzowaną procedurę instalacji	200
Utwórz uproszczonych „dublerów” systemów zewnętrznych	201
Kiedy: Pracujesz z zewnętrznymi źródłami danych referencyjnych	201
Odizoluj wybrane systemy zewnętrzne	202
Kiedy: Na Waszą pracę mają wpływ systemy zewnętrzne	202
Wypróbuj walidację wielostopniową	202
Kiedy: Pracujesz w dużych grupach lub z grupami w różnych lokalizacjach	202
Wykonaj testy w transakcjach	203
Kiedy: Wykonywalne specyfikacje modyfikują dane referencyjne	203
Wykonaj szybkie testy danych referencyjnych	204
Kiedy: W systemach opartych na danych	204
Oczekuj zdarzeń, zamiast nastawiać się na określony czas trwania	204
Uczyń przetwarzanie asynchroniczne rozwiązaniem opcjonalnym	205
Kiedy: Pracujesz nad projektami typu greenfield	205
Nie wykorzystuj wykonywalnych specyfikacji w funkcji walidacji kompleksowej typu end-to-end	206
Kiedy: W projektach typu brownfield	206
Szybsze uzyskiwanie informacji zwrotnej	207
Wprowadź operacyjny czas działania	207
Kiedy: Musisz radzić sobie z ograniczeniami czasowymi	207
Podziel duże zestawy testów na mniejsze moduły	208
Unikaj wykorzystywania do testów baz danych przechowywanych w pamięci	209
Kiedy: W systemach bazujących na danych	209
Oddziel testy szybkie od wolnych	210
Kiedy: Niewielka część testów zajmuje większość czasu przeznaczonego na ich wykonanie	210
Utrzymaj stabilność uruchamianych na noc pakietów testów	210
Kiedy: Niemrawe testy są rozpoczynane w nocy	210
Stwórz pakiet aktualnej iteracji	211
Wykonuj testy równolegle	212
Kiedy: Możesz stworzyć kilka środowisk testowych	212
Spróbuj wyłączyć testy, z których wykonaniem wiąże się mniejsze ryzyko	213
Kiedy: Feedback z testów jest bardzo niemrawy	213
Zarządzanie testami, które kończą się niepowodzeniem	214
Stwórz pakiet znanych nieudanych testów regresji	215
Sprawdź automatycznie, które testy są wyłączone	216
Kiedy: Nieudane testy zostają zneutralizowane, a nie trafiają do oddzielnego pakietu	216
Pamiętaj	217

11 Tworzenie systemu dokumentacji 219

Żyjąca dokumentacja powinna być łatwa do zrozumienia	219
Nie twórz długich specyfikacji	220
Nie używaj wielu specyfikacji do opisanie jednej funkcji	220
Szukaj koncepcji wyższego poziomu	221
Unikaj stosowania w testach technicznych pojęć automatyki	221
Kiedy: Interesariusze nie mają wystarczającej wiedzy technicznej	221

Żyjąca dokumentacja powinna być spójna	222
Ewoluuący język	223
Tworząc język specyfikacji, bazuj na personach	224
Kiedy: W projektach internetowych	224
Promuj współpracę w celu zdefiniowania słownika języka	225
Kiedy: Rezygnujesz z warsztatów specyfikacji	225
Gromadź dokumentację swoich bloków konstrukcyjnych	226
Żyjąca dokumentacja powinna być zorganizowana	
 zgodnie z regułami ułatwiającymi dostęp	227
Organizuj bieżącą pracę, segregując ją według historyjek	228
Zorganizuj historyjki na podstawie obszarów funkcjonalnych	228
Pogrupuj specyfikacje według dróg nawigacji w interfejsie użytkownika	229
Kiedy: Dokumentujesz interfejsy użytkownika	229
Zorganizuj specyfikacje według procesów biznesowych	230
Kiedy: Wymagana jest identyfikowalność przypadków użycia typu end-to-end	230
Używaj znaczników zamiast adresów URL, odnosząc się do specyfikacji wykonywalnych	231
Kiedy: Potrzebujesz identyfikowalności specyfikacji	231
 Sluchaj swojej żyjącej dokumentacji	232
 Pamiętaj	233

CZĘŚĆ III Studia przypadków

12 uSwitch 237	
Rozpoczęcie zmiany procesu	238
Optymalizacja procesu	240
Obecny kształt procesu	244
Efekt końcowy	245
Najważniejsze lekcje	245
13 RainStor 247	
Zmiana procesu	247
Obecny kształt procesu	250
Najważniejsze lekcje	251
14 Iowa Student Loan 253	
Zmiana procesu	253
Optymalizacja procesu	255
Żyjąca dokumentacja jako przewaga konkurencyjna	258
Najważniejsze lekcje	259
15 Sabre Airline Solutions 261	
Zmiana procesu	261
Poprawa współpracy	263
Efekt końcowy	265
Najważniejsze lekcje	265

16 ePlan Services 267

Zmiana procesu	267
Żyjąca dokumentacja	270
Obecny proces	271
Najważniejsze lekcje	273

17 Songkick 275

Zmiana procesu	276
Obecny kształt procesu	278
Najważniejsze lekcje	280

18 Podsumowanie 283

Współpraca przy definiowaniu wymagań buduje wzajemne zaufanie interesariuszy i członków zespołu	283
Współpraca wymaga przygotowania	284
Współpraca może przybierać wiele różnych form	285
Przydaje się umiejętność spojrzenia na cel końcowy jak na dokumentowanie procesów biznesowych	286
W dłuższej perspektywie prawdziwą wartością dla zespołu jest system żyjącej dokumentacji	287

Dodatek A. Źródła 289

Skorowidz 293

Wprowadzenie

Książka, którą trzymasz w dłoniach (lub którą widzisz na monitorze swojego komputera), jest efektem końcowym cyklu badań nad procesem, w czasie którego zajmujące się wytwarzaniem oprogramowania zespoły na całym świecie definiują, rozwijają i dostarczają *właściwe* produkty — wolne od błędów i opracowywane w krótkich cyklach projektowych. Ta książka zawiera wiedzę zgromadzoną dzięki analizie prawie pięćdziesięciu projektów inżynierii oprogramowania, podczas których zespoły pracowały nad dostarczeniem szerokiego zakresu produktów — począwszy od ogólnodostępnych stron internetowych aż po wewnętrzne systemy typu *back-office*. Zbierając materiały do tej książki, przeanalizowałem działania zespołów rozmaitych typów — były to zarówno małe grupy ludzi pracujących w tym samym biurze, jak i powiązane sieci programistów czasami rozlokowane nawet na różnych kontynentach, osoby posługujące się różnorodnymi metodologiami procesów inżynierii: programowaniem ekstremalnym (ang. *Extreme Programming*), Scrumem, Kanbanem czy podobnymi metodami (często połączonymi ze sobą i przybierającymi formy metod zwinnych, tj. *agile* i *lean*). Wszystkie te zespoły łączyło jedno: ścisła współpraca w zakresie opracowywania poprawnych specyfikacji oraz testów, a także końcowy sukces i korzyści, jakie stały się ich udziałem dzięki tej współpracy.

Specyfikacja przez przykłady

Różne zespoły stosują różne terminy dla nazwania tego, co robią ze specyfikacjami i testami, a przecież w każdym przypadku ludzie ci odwołują się do tych samych kluczowych zasad i koncepcji, które moim zdaniem w gruncie rzeczy są zawsze takie same. Wśród używanych przez zespoły terminów i skrótów znajdujemy takie jak:

- Zwinne testowanie akceptacyjne.
- ATDD (ang. *Acceptance Test-Driven Development*), czyli wytwarzanie oprogramowania w oparciu o testy akceptacyjne.
- EDT (ang. *Example-Driven Development*), tj. wytwarzanie oprogramowania w oparciu o przykłady.
- Testowanie przez historyjki użytkownika (ang. *story testing*).
- BDD (ang. *Behavior-Driven Development*), czyli wytwarzanie oprogramowania w oparciu o analizę zachowań użytkowników.
- Specyfikacja przez przykłady (ang. *Specification by Example*).

Już sam fakt, że te same działania mają tak wiele różnych nazw, może nam wiele powiedzieć o ogromnej innowacyjności, z jaką mamy do czynienia w tej dziedzinie. Mnogość stosowanych terminów i definicji świadczy także o tym, że praktyki opisane w tej książce wpływają na zmianę podejścia zespołów do specyfikacji oraz do zagadnienia wytwarzania oprogramowania i jego testowania. Kierując się dążeniem do zachowania spójności przekazu, musiałem

jednak wybrać jeden termin. Zdecydowałem się na *specyfikację przez przykłady* i będę używał tego określenia w całej książce. Wyjaśnię mój wybór w części tego wprowadzenia zatytułowanej „Kilka słów na temat terminologii”, gdzie znajdzie się także kilka szczegółów dotyczących reszty terminów zastosowanych dla nazwania elementów opisywanego procesu.

W świecie rzeczywistym...

Szczegóły proponowanej przeze mnie metodyki przedstawiam, odwołując się do licznych studiów przypadków i zarejestrowanych rozmów z członkami grup programistów. Wybrałem to podejście świadomie, z myślą o tym, żeby czytelnicy mieli szansę dowiedzieć się, jak radziły sobie prawdziwe zespoły, pracujące zgodnie z regułami tej metody i czerpiące z niej spore korzyści. Specyfikacja przez przykłady w żadnym razie nie jest ciemniejszą stroną sztuki, mimo że niektóre popularne media przekonują, że jest inaczej.

Prawie wszystko, o czym mówimy w tej książce, pochodzi z rzeczywistego świata inżynierii oprogramowania, dotyczy działających w tym świecie zespołów oraz ich — jak najbardziej realnych — doświadczeń. Tylko nieliczne praktyki zostały przedstawione w formie sugestii nieopartych studium przypadku. Są to zwykłe pomysły, które — moim zdaniem — będą w przyszłości stanowić ważny przyczynek do dyskusji. I właśnie jako takie są przeze mnie przedstawiane.

Jestem przekonany, że moje wnioski, a także badania, jakie przeprowadziłem na potrzeby tej książki, nie zostaną uznane za poważne studia naukowe przez sceptyków, którzy twierdzą, iż zwinne praktyki się nie sprawdzają, a branża powinna wrócić do „prawdziwej inżynierii oprogramowania”¹. Akceptuję to. Zasoby, do jakich miałem dostęp podczas zbierania materiałów na potrzeby tego projektu, można uznać za niewystarczające w porównaniu z tym, czego wymaga się od poważnych badań naukowych. I nawet gdybym dysponował zasobami spełniającymi kryteria uznanych metodologii, to przecież ani nie jestem naukowcem, ani nie zamierzam przedstawiać siebie jako naukowca. Jestem praktykiem.

Kto powinien zapoznać się z tą książką?

Jeśli tak jak ja jesteś praktykiem i zarabiasz na chleb, pracując przy wytwarzaniu oprogramowania lub wspierając pracę nad oprogramowaniem, ta książka ma Ci wiele do zaoferowania. Powstała wszak przede wszystkim z myślą o zespołach, które starały się wdrożyć w swoich środowiskach zwinne praktyki, ale napotkały problemy wpływające niekorzystnie na jakość produktu końcowego, zwiększające nakład pracy i prowadzące do tego, że dzieło rąk programistów nie spełniało oczekiwań klientów. (Tak... wówczas mamy do czynienia

¹ Jeśli chcesz poznać argumenty tych, którzy łudzą się, iż rygorystyczne podejście do inżynierii oprogramowania przynosi korzyści, tak jak gdyby była to jakaś drugorzędna gałąź fizyki, zajrzyj na stronę <http://www.semat.org>. Dobre kontrargumenty zawiera prezentacja Glenna Vanderburga zatytułowana *Software Engineering Doesn't Work!*, z którą można zapoznać się na stronie <http://confreaks.net/videos/282-lsrc2010-real-software-engineering>.

z prawdziwymi kłopotami i w takiej sytuacji zwykła iteracja nie na wiele się zda, ponieważ staje się „objazdem”, a nie rozwiązaniem problemu). Specyfikacja przez przykłady, zwinne testowanie akceptacyjne, BDD i wszystkie alternatywne nazwy określają ten sam proces, zorientowany na rozwiązywanie tych problemów. Ta książka pomoże Ci poznać podstawy wspomnianych dobrych praktyk inżynierii oprogramowania. Dowiesz się z niej, jak możesz mieć większy pozytywny wpływ na zespół, niezależnie od tego, czy jesteś testerem, deweloperem, analitykiem, czy właścicielem produktu.

Jeszcze kilka lat temu większość ludzi, których spotykałem na konferencjach, nigdy wcześniej nie słyszała o opisanych w tej książce praktykach. Gros tych, z którymi rozmawiam dziś, ma świadomość istnienia wspomnianych praktyk, ale wielu z nich nie udało się wdrożyć ich prawidłowo. Istnieje niewielka baza literatury fachowej traktującej o problemach, jakie napotykają zespoły podczas szeroko pojętego wdrażania zwinnych praktyk, dlatego każdy zniechęcony zespół stwierdza, że jego przypadek musi być wyjątkowy, a sformalizowane koncepcje po prostu nie sprawdzają się w „jego świecie”. Ludzie z takich zniechęconych zespołów dziwią się, jak po zaledwie pięciu minutach rozmowy udaje mi się trafnie zidentyfikować ich trzy lub cztery największe problemy. Często są bardzo zdziwieni, słysząc, że wiele innych zespołów napotyka na swojej drodze te same przeszkody.

Jeśli pracujesz w takim zespole, książka, którą właśnie trzymasz w dłoniach, uświadomi Ci przede wszystkim, że nie jesteś sam. Zespoły, z którymi rozmawiałem podczas zbierania materiałów do tej pozycji, wcale nie były zespołami idealnymi. Także one napotkały na swojej drodze mnóstwo problemów! Jednak zamiast rezygnować po zderzeniu z murem przeciwności, ludzie Ci postanowili go „objechać” lub zburzyć. Taka świadomość wspólnoty często pozwala zobaczyć swoje problemy w innym świetle. Mam nadzieję, że po przeczytaniu tej książki będziesz właśnie jedną z tych wyjątkowych osób, które potrafią spojrzeć na sytuację z innej perspektywy.

Jeśli akurat znajdujesz się w trakcie wdrażania specyfikacji przez przykłady, ta książka dostarczy Ci przydatnych wskazówek, dzięki którym dowiesz się, jak poradzić sobie z bieżącymi problemami i czego możesz spodziewać się w przyszłości. Mam nadzieję, że będziesz uczyć się na błędach innych ludzi i niektórych z nich uda Ci się uniknąć.

Ta książka została napisana także z myślą o doświadczonych praktykach, osobach, którym udało się wdrożenie zasad specyfikacji przez przykłady. Gdy zaczynałem rejestrowanie wywiadów z zespołami, spodziewałem się, że nie dowiem się niczego nowego. Myślałem, że „wiem, co jest grane”, i szukałem tylko potwierdzenia. Tymczasem ostatecznie musiałem przyznać, że zaskoczyło mnie to, jak wiele różnych pomysłów można zastosować zależnie od kontekstu. Dowiadywałem się o rozwiązaniach, które wcześniej trudno mi było sobie wyobrazić! Wiele się przy tym nauczyłem. Dlatego mam nadzieję, że i Ty, drogi Czytelniku, wiele nauczysz się dzięki lekturze tej książki. Przedstawione w niej praktyki i pomysły powinny zainspirować Cię do wypróbowania alternatywnych rozwiązań problemów i dać Ci kilka podpowiedzi, jak usprawnić pracę zespołu, gdy tylko natkniesz się na historię przypominającą jakiś opisany tu przykład.

Co znajdziesz w środku?

W pierwszej części książki przedstawię podstawy koncepcji specyfikacji przez przykłady. Zamiast przekonywać Cię do tego, byś przestrzegał zasad opisanych w tej książce, pokażę Ci — w sposób charakterystyczny dla metodyki specyfikacji przez przykłady — konkretne korzyści, jakie zespoły pracujące nad wytwarzaniem oprogramowania mogą uzyskać dzięki zastosowaniu prezentowanych tu reguł. Jeśli zastanawiasz się nad zakupem tej książki, najpierw przejrzyj rozdział 1. i odpowiedz sobie na pytanie, czy którejs z opisanych tam korzyści można spodziewać się także podczas realizacji Twojego projektu. W rozdziale 2. przedstawię najważniejsze modele procesów i kluczowe artefakty specyfikacji przez przykłady. W rozdziale 3. omówię szczegółowo koncepcję żyjącej dokumentacji. W rozdziale 4. przedstawię typowe punkty wyjścia dla zainicjowania zmian procesu i kultury pracy zespołu oraz wskażę, na co należy zwrócić uwagę podczas implementacji metodyki.

Jednym z moich celów jako autora tej książki jest stworzenie spójnego słownika dla wzorców, idei i artefaktów, który będzie mógł być stosowany podczas wdrażania zasad specyfikacji przez przykłady. W branży inżynierii oprogramowania używa się tuzina różnych terminów dla nazwania samej praktyki i dwa razy więcej dla nazwania poszczególnych elementów kluczowych metodyki. Ludzie nazywają ten sam element: plikami właściwości, przypadkami testowymi, plikami BDD, testami akceptacyjnymi itp. Z tego powodu w rozdziale 2. przedstawię najlepsze (moim zdaniem) terminy stosowane dla określenia wszystkich kluczowych elementów omawianej metodyki. Nawet jeśli jesteś doświadczonym praktykiem, proponuję, żebyś zapoznał się z tym rozdziałem i odpowiedział sobie na pytanie, czy tak samo rozumiemy kluczowe terminy, wyrażenia i wzorce, o których mówimy w tej książce.

W drugiej części przedstawię najistotniejsze praktyki, z których zespoły opisane w studiach przypadków korzystały, by wdrożyć zasady specyfikacji przez przykłady. Zespoły działające w różnych środowiskach podejmowały różnorakie decyzje — czasem diametralnie różne, a czasami wręcz sprzeczne — aby jednak ostatecznie uzyskać takie same efekty. Opis tych praktyk uzupełnię dodatkowo charakterystyką kontekstów, w których działały zespoły podczas wdrażania reguł specyfikacji przez przykłady. Siedem rozdziałów drugiej części podzieliłem mniej więcej zgodnie z podziałem na obszary procesów.

W branży takiej jak inżynieria oprogramowania nie istnieją uniwersalne, idealne rozwiązania, ale na pewno można mówić o dobrych pomysłach, które da się zastosować ze świetnym skutkiem w różnych kontekstach. W drugiej części tej książki znajdziesz ikonki przedstawiające kciuk uniesiony w górę lub skierowany w dół. Takie grafiki będą pojawiać się przy nagłówkach z opisem działań, które badane zespoły uznały za użyteczne, lub problemów, z którymi często musiały się mierzyć. Potraktuj te wskazania jak propozycje wypróbowania przedstawionego przeze mnie rozwiązania lub ostrzeżenie przed sytuacją, jakiej warto unikać. Nie są to w żadnym razie gotowe recepty, które należy koniecznie zastosować. Ikony przedstawiające strzałki podkreślają najistotniejsze koncepcje związane z każdym z prezentowanych działań.

Wytwarzanie oprogramowania nie jest sztuką statyczną, ponieważ stale zmieniają się zarówno zespoły, jak i środowiska, w których pracują ludzie. Procesy rozwoju produktu muszą nadążać za tymi zmianami. W części trzeciej tej książki przedstawię studia przypad-

ków opisujące „podróże” kilku wybranych zespołów. Omówię procesy, ograniczenia i konteksty, analizując ewolucję projektów. Przedstawione w tej części historii pozwolą Ci lepiej przygotować się do czekających Cię zmian lub zachęcą do wykonania następnego kroku, pomogą w poszukiwaniach inspiracji, a także zainspirują do odkrywania nowych metod działań i rozwiązań.

W ostatnim rozdziale podsumuję najważniejsze wnioski, do których doszedłem dzięki analizie licznych studiów przypadków zebranych na potrzeby tej książki.

Coś więcej niż podstawy

Gdyby odwołać się do tradycyjnego modelu doskonalenia *Shu-ha-ri*², ta książka znajduje się na poziomie *ha*. Na tym etapie uczeń łamie obowiązujące zasady i dowiadyuje się, że istnieje wiele różnych sprawdzających się modeli. Mój model i moje doświadczenie przedstawiłem w *Bridging the Communication Gap*. Natomiast w książce, którą właśnie trzymasz w dłoniach, bardzo staram się zachować obiektywne spojrzenie i uniknąć ulegania wpływom moich dotychczasowych osobistych doświadczeń. O projektach, nad którymi pracowałem osobiście, mówię tylko wtedy, gdy naprawdę zależy mi na przekazaniu ważnych informacji, a żaden z ankietowanych zespołów nigdy nie zetknął się z podobną sytuacją. W tym sensie specyfikacja przez przykłady zaczyna się tam, gdzie skończyłem *Bridging the Communication Gap*.

Podstawowe zasady opisuję krótko w rozdziale 2. Nawet jeśli nigdy wcześniej nie słyszałeś o żadnej z przedstawionych tam koncepcji, lektura tego rozdziału powinna zapewnić Ci dość danych, żebyś mógł zrozumieć resztę książki. Staram się przy tym nie zagłębiać za bardzo w podstawy. Szczegółową analizę fundamentów teorii specyfikacji przez przykłady znajdziesz w *Bridging the Communication Gap* i nie jest moim celem kopiowanie tych informacji.

Jeśli chcesz poszerzyć swoją wiedzę w zakresie podstaw, zajrzyj na stronę <http://specificationbyexample.com>, zarejestruj posiadany egzemplarz tej książki, a otrzymasz za darmo plik PDF z *Bridging the Communication Gap*.

Nie sądzę, że kiedykolwiek powstanie kontynuacja tej pozycji — to znaczy książka mojego autorstwa na poziomie *ri* — ponieważ moim zdaniem poziom ten jest w ogóle nieosiągalny dla książek. Z drugiej strony wierzę, że ta książka pomoże Ci przejść na wyższy poziom. Gdy zaczniesz rozumieć, że wybór konkretnego narzędzia nie ma znaczenia, będzie to znaczyło, że znalazłeś się na poziomie *ri*.

² *Shu-ha-ri* to model doskonalenia związany z nauką aikido. *Shu-ha-ri* znaczy mniej więcej tyle co: *podporządkuj się – zerwij – odejdź*. Na pierwszym poziomie (*shu* — *podporządkuj się*) uczeń zdobywa wiedzę, pilnie naśladując przedstawiony mu model. Na drugim poziomie (*ha* — *zerwij*) uczeń dowiadyuje się, że istnieją inne modele i rozwiązania. Na trzecim poziomie (*ri* — *odejdź*) uczeń przekracza granicę, wychodząc poza działania polegające na bezrefleksyjnym powtarzaniu znanych rozwiązań.

Ta książka nie zawiera kodu źródłowego i nie wyjaśnia działania żadnych narzędzi

Nie znajdziesz w tej książce kodu źródłowego ani instrukcji opisujących, jak użyć konkretnego narzędzia. Czuję się zobligowany do uprzedzenia o tym już teraz, ponieważ nawet w trakcie trwania procesu wydawniczego byłem zmuszony do składania wyjaśnień w tej kwestii (zwykle chodziło o odpowiedź na pytanie: „Co właściwie masz na myśli? Książka o programowaniu bez kodu źródłowego? Jak to możliwe?”).

Zasady i praktyki specyfikacji przez przykłady wpływają przede wszystkim na sposób komunikacji międzyludzkiej w zespołach pracujących nad dostarczaniem oprogramowania, a także na współpracę zespołów z użytkownikami biznesowymi i interesariuszami (ang. *stakeholders*). Jestem pewien, że wielu sprzedawców narzędzi informatycznych będzie próbować sprzedać Ci gotowe rozwiązania. Z kolei wielu menedżerów chętnie nawet zapłaciłoby za to, aby ktoś sprawił, żeby ich problem po prostu zniknął. Niestety mamy do czynienia przede wszystkim z problemem ludzkim, a nie technicznym.

Bill Gates mawiał: „Pierwszą zasadą dotyczącą wszelkich technologii wykorzystywanych w działalności gospodarczej jest to, że zautomatyzowanie skutecznego działania potęguje efektywność. Druga zasada mówi, że automatyzacja zastosowana do nieefektywnego działania zwiększa poziom nieskuteczności”. Wiele zespołów, którym nie udało się poprawne wdrożenie specyfikacji przez przykłady, powiększyło swoją nieefektywność poprzez automatyzację dotychczasowych (niepoprawnych) praktyk. Zamiast więc skupiać się na konkretnym narzędziu, chcę zaproponować rozwiązanie problemów, które są prawdziwą przyczyną niepowodzeń. Jeśli tylko uda Ci się zapanować nad komunikacją i ustalić zasady współpracy, przyjdzie czas na wybranie odpowiednio dopasowanego narzędzia. Jeśli po przeczytaniu tej książki będziesz chciał poszerzyć swoją wiedzę na temat narzędzi, które wspierają wdrożenie specyfikacji przez przykłady, odwiedź stronę <http://specificationbyexample.com> i przejrzyj znajdujące się tam zasoby.



Kilka słów na temat terminologii

Jeśli jest to Twój pierwszy kontakt ze specyfikacją przez przykłady, ATDD, ze zwinnymi testami akceptacyjnymi, BDD czy z którymkolwiek z terminów, których ludzie używają na określenie zbioru praktyk będącego podstawowym tematem tej książki, oznacza to, że udało Ci się uniknąć wielu lat zamieszania spowodowanego powielaniem i kreowaniem mylących nazw. Powinieneś się z tego ucieszyć i możesz pominąć tę część wprowadzenia. Jeśli jednak zetknąłeś się kiedyś z którymkolwiek z wymienionych terminów, określenia używane przeze mnie w tej książce mogą Cię zaskoczyć. Za chwilę wyjaśnię, dlaczego wybrałem te, a nie inne terminy, i dlaczego uważam, że Ty także powinieneś zacząć z nich korzystać.

Podczas prac nad tą książką borykałem się z problemami, które nie są obce wszystkim praktykom piszącym na temat zautomatyzowanych specyfikacji. Jeśli terminologia ma mieć sens, musi być spójna. Często rozumiemy to dopiero wtedy, gdy sami zabieramy się do opisywania jakiegoś zagadnienia. Ponieważ moja książka jest produktem bazującym na cyklu wywiadów, a wiele osób, z którymi rozmawiałem, używało różnych terminów do nazwania tej samej rzeczy, uzyskanie spójności terminologicznej okazało się zadaniem niezwykle trudnym.

Zdałem sobie sprawę, że specjaliści realizujący reguły specyfikacji przez przykłady w praktyce — na czele ze mną — sami są sobie winni, bo to przecież oni używają zdradliwych terminów technicznych, wprowadzających w błąd nie tylko ich samych, ale i wszystkich tych, którzy próbowali wdrożyć owe praktyki. Wtedy zdecydowałem, że jednym z celów tej książki będzie opracowanie spójnej terminologii, którą będą mogli posługiwać się wszyscy członkowie społeczności wytwórców oprogramowania. Jeśli zależy nam na większym zaangażowaniu w prace projektowe użytkowników biznesowych — co jest przecież jednym z naszych głównych celów — musimy zacząć używać właściwych terminów do nazwania właściwych rzeczy. Czas przestać wprowadzać zamieszanie!

Doskonale rozumiemy potrzebę zachowania spójności, gdy tworzymy specyfikacje. Wiemy, że musimy zachować powtarzalność terminów i unikać wprowadzania pojęć niejednoznacznych. Tyle że jakoś zapominamy o tym, gdy zaczynamy mówić o procesach. Na przykład gdy mówimy o *ciągłej integracji* w kontekście specyfikacji przez przykłady, tak naprawdę wcale nie chodzi nam o wykonywanie testów integracyjnych. Dlaczego więc mielibyśmy używać tego terminu, a następnie wyjaśniać, czym różnią się testy akceptacyjne od testów integracyjnych? Zanim sam nie zacząłem używać terminu *warsztat specyfikacji* do nazwania spotkania, w którym uczestniczą osoby odpowiedzialne za wspólne opracowanie testów akceptacyjnych, miałem spore trudności, aby przekonać użytkowników biznesowych do udziału w takich zgromadzeniach. Zwykła zmiana nazwy sprawiła, że problem przestał istnieć. Dzięki użyciu lepszego glosariusza terminologicznego można uniknąć zupełnie bezsensownych dysput, sprawiając, że wszyscy dobrze się rozumieją.

Dlaczego specyfikacja przez przykłady?

W pierwszej kolejności chciałbym wyjaśnić, dlaczego wybrałem termin „specyfikacja przez przykłady” dla nazwania zbioru praktyk. Dlaczego nie zwinne testowanie akceptacyjne, BDD czy ATDD?

Podczas konferencji „Domain Driven Design eXchange”, która odbyła się w 2010 roku w Londynie³, Eric Evans stwierdził, że termin *zwinny* (ang. *agile*) stracił już wszelkie znaczenie, bo obecnie wszystko można nazwać zwinnym. Niestety Eric ma rację. Widziałem o wiele za dużo zespołów, próbujących wdrożyć proces, który nie miał szans zakończenia się powodzeniem, choć wszystkim wydawało się, że etykieta *agile* w jakiś magiczny sposób sprawi, że okaże się skuteczniejszy, niż był w rzeczywistości. Przecież mamy do swojej dyspozycji ogromną bibliotekę dostępnego piśmiennictwa omawiającego prawidłowe wdrażanie zasad programowania ekstremalnego, Scruma czy innych mniej popularnych procesów z bogatego wachlarza praktyk zwinnych.

³ <http://skillsmatter.com/event/design-architecture/ddd-exchange-2010>.

Aby odejść od tych wprowadzających zamieszanie niejasności oraz dyskusji na temat tego, czy zwinne praktyki się sprawdzają, czy nie (i wyjaśnić, co to właściwie znaczy *być agile*), w tej książce staram się ze wszystkich sił unikać terminu *zwinny*. Używam go tylko wtedy, gdy nawiązuję do zespołów, które rozpoczęły realizację dobrze zdefiniowanych procesów zbudowanych na zasadach określonych w Manifeście Agile. Tak więc w sytuacji, gdy nie mogę umieścić wtrętu ze słowem „agile” w co drugim moim zdaniu, użycie terminu „zwinne testowanie akceptacyjne” jest po prostu nie do przyjęcia.

Opisane w tej książce praktyki nie stanowią kompletnej metodyki tworzenia oprogramowania. Są za to uzupełnieniem innych metod — zarówno tych bazujących na iteracji, jak i na przepływach — zapewniając dyscyplinę specyfikacji i testów, poprawiając komunikację między interesariuszami i członkami zespołów deweloperskich, ograniczając liczbę zbędnych przeróbek i ułatwiając wprowadzanie zmian. Nie chcę zatem używać żadnego terminu zawierającego element „Driven Development” (a zwłaszcza *Behavior-Driven Development* — BDD). Nie chodzi o to, że mam coś przeciwko BDD. Wręcz przeciwnie! Kocham BDD i uważam, że większość z tego, co znajdziesz w tej książce, w rzeczywistości stanowi kluczowy element BDD. Ale BDD również jest ofiarą problemu z nazewnictwem.

Znaczenie terminu BDD podlega ciągłym zmianom. Dan North, największy autorytet w kwestii metodyki BDD oraz człowiek, który jako jeden z niewielu ma prawo decydować, czym jest i czym nie jest BDD, stwierdził podczas konferencji „Agile Specifications, BDD and Testing Exchange” w 2009 roku, że BDD jest *metodyką*⁴. (A dokładniej powiedział, że BDD jest: „zwinną metodyką drugiej generacji, typu *outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation*”). Aby uniknąć nieporozumień i niejasności w kwestii różnicy pomiędzy definicją BDD zaproponowaną przez Dana Northa a tym, co ja sam uważam za BDD, rezygnuję ze stosowania także tego terminu. Ta książka mówi o precyzyjnie zdefiniowanym zestawie praktyk, które można stosować w ramach różnych metod, w tym również metodyki takiej jak BDD (jeśli przyjąć, że BDD naprawdę jest metodyką).

Chciałbym również uniknąć nadużywania terminu *test*. Wielu menedżerów i użytkowników biznesowych niestety uważa testy za aktywność uzupełniającą o czysto technicznej naturze, a nie coś, w czym chcieliby brać udział. Przecież mają od tego wyspecjalizowanych testerów! Specyfikacja przez przykłady wymaga aktywnego uczestnictwa interesariuszy i członków zespołu dostarczającego oprogramowanie, włącznie z deweloperami, testerami i analitykami. Jeśli nie możemy nadużywać terminu *test*, trudno mówić o testowaniu historyjek, zwinnym testowaniu akceptacyjnym i posługiwać się innymi, podobnymi określeniami.

W ten oto sposób doszliśmy do terminu specyfikacja przez przykłady, który staje się najsensowniejszym wyborem, ponieważ niesie najmniejszy bagaż negatywnych konotacji.

⁴ <http://skillsmatter.com/podcast/java-jee/how-to-sell-bdd-to-the-business>.

A

analiticy, 121
analiza
 kaskadowa, 78
 wstępna, 123
aplikacje typu back-office, 102, 230
ATDD, Acceptance Test-Driven Development, 11, 55, 116
automatyzacja
 elementów systemu, 68
 interfejsu użytkownika, 183
 scenariuszy, 79
 testów, 20, 66, 74
 walidacji specyfikacji, 169, 172, 194
 wykonywalnych specyfikacji, 69, 72, 177, 232
 zmiany zegara, 208

B

BDD, Behavior-Driven Development, 11, 18, 55
biblioteki
 automatyzacji interfejsu użytkownika, 184
 automatyzacyjne przeglądarki, 184
bumerang, 88

C

cel biznesowy, 50, 94
 zakres prac, 45
chirurgia śrutówką, 90
CI, continuous integration, 195
ciągła
 integracja, CI, 195, 213
 walidacja, 199
CRUD, Create, Read, Update, Delete, 96
czas reakcji, 141

D

dane
 referencyjne, 192, 193
 testowe, 191
DDE, 80
dedykowane środowisko testowe, 200
definicje kroków, step definitions, 170
definiowanie
 historyjek, 100
 wymagań, 283
 zakresu, 19, 93, 95
 mapowanie efektu, 105
 mapowanie historyjki użytkownika, 105
 wstrzykiwanie funkcjonalności, 19, 105
demo produktu, 79
dokumentacja, 29, 49, 56, 219
dokumentowanie procesów
 biznesowych, 286
dzielenie testów, 211

E

EDT, Example-Driven Development, 11
efektywne dostarczanie oprogramowania, 276
eksperci, 121

F

feature injection, 19, 105
feedback, 77, 78
fikstury, fixtures, 170
fikstury testowe, 181
formułowanie celów, 99
formy współpracy, 285
funkcjonalność
 biznesowa, 154
 interfejsu użytkownika, 186, 188

G

gromadzenie wiedzy, 147
grupowanie
 nieudanych testów, 215
 specyfikacji, 229

H

hierarchia systemu żyjącej dokumentacji, 229
historyjki, 78–81, 95, 101, 228

I

identyfikowalność wymagań, 83
ilustrowanie wymagań
 niefunkcjonalnych, 140
informacja zwrotna, 207
inicjowanie zmian, 63
integracja zespołów
 multidyscyplinarnych, 78
interfejs użytkownika, 141, 157
 automatyzacja, 184
 funkcjonalność, 186, 188
 poziomy automatyzacji, 190
 test, 185, 188
iteracje, 80, 81

J

język
 specyfikacji, 223, 224
 wszechobecny, Ubiquitous Language, 165

K

kamienie milowe, 84
Kanban, 30
klasy równoważności, 132
kod automatyzacji, 178
komunikacja w zespole, 242
koncepty wyższego poziomu, 221
konsultant, 173
kontekst w bazie danych, 189
kontrola
 jakościowa, QA, 114
 wersji, 83
koszt
 automatyzacji, 171
 utrzymania wykonywalnych specyfikacji, 170
kryteria akceptacji, 250

L

lista kontrolna, 143, 144
logika biznesowa, 180, 182

M

mapowanie
 efektu, 105
 historyjki użytkownika, 105
menedżer produktu, 136
metadane, 250
metaprogramowanie, 119
metodyka scrumowa, 81
metodyki zwinne, 65
migracja, 75
minimalne zbywalne właściwości, 99
model
 ATDD, 55
 BDD, 55
 QUPER, 142
 tradycyjny, 28
 Zakładając/Jeżeli/Wtedy,
 163, 168
 zorientowany na
 dokumentację, 60
modele współpracy, 109, 124
modyfikacja przykładów, 138

N

narzędzia, 290
 do automatyzacji, 84, 172
 specyfikacji, 170
 testów, 182
 monitorujące, 77
 open source, 259
narzędzie
 Balsamiq Mockups, 142
 Cucumber, 84
 FIT, 171, 261
 FitNesse, 262, 272
niedopasowanie organizacyjne, 88

O

odchudzone wytwarzanie
 oprogramowania, 95
opis, 154
 procesów walidacji, 179, 180
 testów akceptacyjnych, 154
optymalizacja
 praktyk testowania, 87
 procesu, 240

organizacja warsztatów
 specyfikacji, 111

P

PBR, Product Backlog
 Refinement, 111
planowanie automatyzacji, 173
podział historyjek, 104
poprawa jakości, 65
powielanie logiki biznesowej, 180
poziom
 przepływu pracy
 użytkownika, 190
 reguł biznesowych, 190
 techniczny aktywności, 190
priorytetyzacja, 98, 244
priorytetyzacja historyjek, 118
proces
 asynchroniczny, 206
 konfiguracji, 238
 RUP, 86
 wytwarzania
 oprogramowania, 245
programiści, 121
programowanie
 ekstremalne, XP, 30, 195
 w parach, 113
projekt
 typu brownfield, 34
 typu greenfield, 34, 199
projektowanie testów, 270
projekty oparte na danych, 266
przepływ procesu, 183
przetwarzanie asynchroniczne, 204
przykłady
 funkcjonalne, 142
 kompletne, 133
 podstawowe, 161
 przypadków brzegowych, 138
 realistyczne, 134, 136
 referencyjne, 144, 145
 reprezentatywne, 159
 wymagań funkcjonalnych, 140
 zrozumiałe, 137
przypadki
 brzegowe, 239
 użycia, use cases, 85, 86

Q

QA, 114
QUPER, 142

R

realizacje przypadków użycia, 86
refaktoryzacja, 186
refaktoryzacja specyfikacji, 87
reguły biznesowe, 87
reorganizacja
 pracy, 104
 warstwy automatyki, 187
restrukturyzacja skryptów, 154
rola Batmana, 76
rozwiązania alternatywne, 103
RUP, Rational Unified Process, 86
rzucanie wyzwania wymaganiom, 96

S

SBE, Specification by Example, 65
Scrum, 30
SKIT, 79
skrypt, 152
skrypty testów ręcznych, 175
słownik języka specyfikacji, 225
specyfikacja, 45, 107, 147–168
 dobra, 149
 domyślne wartości, 164
 funkcjonalność biznesowa, 154
 implementacja
 oprogramowania, 155
 integralność referencyjna, 163
 interfejs użytkownika, 157
 jedna reguła biznesowa, 161
 język domeny, 165
 oczekiwane wyniki, 158
 reprezentatywne przykłady, 159
 trudności techniczne, 156
 udoskonalanie, 20, 147–168
 utrzymywanie, 61
 wartości wejściowe, 158
 wdrażanie, 147
 zła, 150
specyfikacja przez przykłady, 11,
 20, 28–30, 54, 61, 75, 78
 dostosowanie aktywności, 39
 identyfikowalność, 82
 jakość produktu, 32
 wdrażanie, 74
 wprowadzanie zmian, 30
 zatwierdzenia, 82
 znajdowanie błędów, 36
specyfikacje
 grupowane, 229
 rozbudowane, 220
 wykonywalne, 20, 48, 53, 58

spike, 172
spotkanie
 ATDD, 116
 przygotowawcze, 117
 trzech amigos, 112
 z interesariuszami, 119
spójność żyjącej dokumentacji, 223
sprawdzanie
 logiki biznesowej, 182
 testu, 87
 funkcjonalności, 133
stabilność automatycznych testów, 198
studia przypadków, 235
system kontroli wersji, 84
systemy zewnętrzne, 202
szacowanie historyjek, 99

Ś

śledzenie
 bumerangów, 88
 procesu przepływu, 182
 przebiegu zatwierdzeń
 wymagań, 82

T

tablica Kanban, 78
TDD, 70
test, 50
 akceptacyjny, 50, 100, 196
 automatyczny, 57
 Cucumbera, 239, 244
 first, 19
 FitNesse, 269
 funkcjonalny, 66, 74
 integracyjny, 206, 207
 jednostkowy, 196
 niefunkcjonujący, 69
 nieudany, 215, 216
 niskiego poziomu, 104
 regresji, 215
 równoległy, 183, 212
 sanity testing, 263
 szybki, 210
 techniczny, 155
 typu end-to-end, 207
 wolny, 210
 wyższego poziomu, 104
testerzy, 120, 279
testowanie
 baz danych, 209
 danych referencyjnych, 204

interfejsu użytkownika, 176,
 185, 188
 przez historyjki użytkownika,
 11
 regresyjne, 50
 specyfikacji wykonywalnej,
 157
 w chmurze, 213
 w transakcjach, 203
timing, 44
transakcje, 203
tworzenie
 baz danych, 191
 dokumentacji, 49, 55–62, 219
 strony internetowej, 60
 testy automatyczne, 57
 wykonywalna
 specyfikacja, 58
 dublera systemu, 201
 procesu przepływu, 82
 przykładów, 127
 specyfikacji, 46, 107, 147–168
 przykłady ilustrujące, 130

U

UAT, User Acceptance Testing, 35
udokonalanie specyfikacji, 20,
 147–168
utrzymywanie specyfikacji, 61
użyteczność, 141, 144
użytkownik biznesowy, 99
używanie znaczników, 231

W

walidacja, 20, 47, 49, 195
 procesów asynchronicznych, 204
 specyfikacji, 169, 172
 typu end-to-end, 206
 wielostopniowa, 202
warstwa
 automatyzacji, 163, 178, 179
 interfejsu użytkownika, 183
warsztaty
 PBR, 111
 specyfikacji, 110, 116, 239
wartość, 98
wdrażanie
 specyfikacji, 147
 zasady współpracy, 77
wersje oprogramowania, 73
właściciel produktu, 118

wsparcie kierownictwa, 72
wspomaganie definiowania
 zakresu, 105
współpraca, 109, 116, 263,
 283–285
wstrzykiwanie funkcjonalności,
 feature injection, 19, 105
wybór
 modelu współpracy, 124
 zakresu automatyzacji, 185
wydajność, 140
wymagania
 interesariuszy, 136
 niefunkcjonalne, 140, 141
 przekrojowe, 144
 wydajnościowe, 140
wzorze
 kluczowych procesów, 43, 44
 procesu specyfikacji, 54

X

XP, Extreme Programming, 30

Z

zaangażowanie interesariuszy,
 118, 119
zakres
 automatyzacji, 185
 implementacji, 45
 projektu, 93
zamknięte odpowiedzi, 131
zarządzanie
 danymi testowymi, 191
 testami, 197, 214
 warstwą automatyzacji, 178
 wymaganiami, 257
zastępowanie specyfikacji, 221
zatwierdzanie, 82
zatwierdzenie przypadków użycia,
 85
zaufanie, 283
zespół
 Knighta, 136
 Andrew Jackmana, 114
 Bekk Consulting, 163
 Clare McLennan, 198
 ePlan Services, 267–273
 Global Talent Management,
 77, 83
 Google, 87
 Iana Coopera, 154, 202

zespół

Iowa Student Loan, 216,
253–259
Jodie Parker, 123
Marty Gonzalez Ferrero, 128
Pierre’a Veragena, 189
QA, 238
RainStor, 211, 247–252
Rakesha Patela, 220
Roba Parka, 201
Sabre Airline Solutions,
261–266
Sierra, 80, 120, 226
SNS, 81, 83

Songkick, 275–281

Steera, 86

Stuarta Taylora, 113

TechTalk, 131

uSwitch, 100, 173, 214,
237–246

zestawy testów, 208

zmiana procesu, 64

zmniejszanie zawodności, 197

znaczniki, 231

zwiększanie niezawodności

systemu, 198

zwinne testowanie akceptacyjne,
11, 18

Ż

źródła wartości, 98

Ż

żyjąca dokumentacja, 30, 53–62,
270, 287

hierarchia, 229

jako przewaga konkurencyjna,
258

łatwy dostęp, 219, 227

spójny język, 223

utrzymywanie, 233

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**

SPECYFIKACJA NA PRZYKŁADACH

Poznaj zwinne metody pracy
i dostarczaj właściwe oprogramowanie

Dokładne poznanie wymagań klienta to klucz do w pełni wydajnej aplikacji. Jest niezbędne, by sprostać oczekiwaniom jej przyszłych użytkowników. Metoda SBE (ang. *specification by example*) zachęca do zwinnego (*Agile*) podejścia do tego tematu, dzięki czemu zebranie wymagań będzie przebiegało zdecydowanie sprawniej.

Ta książka rozwieje wszystkie Twoje wątpliwości! Poznasz kluczowe wzorce procesu oraz nauczysz się wprowadzać w nich zmiany. Podejście SBE wymaga zmiany kultury pracy zespołu. Nie jest to zadanie łatwe, dlatego znajdziesz tu najlepsze praktyki stosowane w tej sytuacji. Ostatnie rozdziały książki zostały poświęcone omówieniu przykładów z życia wziętych, a dotyczących najczęściej spotykanych problemów. To szczególnie cenne informacje, które pozwolą Ci wybrać najlepsze sposoby uniknięcia typowych błędów. Książka ta jest obowiązkową lekturą dla wszystkich twórców oprogramowania!

Dzięki tej książce:

- poznasz zalety SBE
- dowiesz się, dlaczego wspólne specyfikowanie jest tak istotne
- nauczysz się definiować cel z uwzględnieniem wzorców
- zmienisz kulturę pracy Twojego zespołu
- skutecznie wprowadzisz SBE w Twojej organizacji

Poznaj zalety SBE!

helion.pl
księgarnia
internetowa

Nr katalogowy: 23153



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nawosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN: 978-83-246-9118-0



9 788324 691180

Cena: 59,00 zł

Informatyka w najlepszym wydaniu