

Microsoft®

SZYBKIE PROJEKTOWANIE

ZAPANUJ NAD CHAOSEM ZADAŃ I PRESJĄ CZASU



Steve McConnell

Helion 

Tytuł oryginału: Rapid Development: Taming Wild Software Schedules

Tłumaczenie: Krzysztof Sawka

ISBN: 978-83-283-3270-6

Authorized translation from the English language edition: RAPID DEVELOPMENT: TAMING WILD SOFTWARE SCHEDULES; ISBN 1556159005; by Steve McConnell; published by Microsoft Press, a division of Microsoft Corporation, Inc. Copyright © 1996 by Steve McConnell

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 Pearson Education, Inc., representing Microsoft Press. Polish language edition published by HELION SA. Copyright © 2017.

AT&T is a registered trademark of American Telephone and Telegraph Company. Apple and Macintosh are registered trademarks of Apple Computer, Inc. Boeing is a registered trademark of The Boeing Company. Borland and Delphi are registered trademarks of Borland International, Inc. FileMaker is a registered trademark of Claris Corporation. Dupont is a registered trademark of E.I. Du Pont de Nemours and Company. Gupta is a registered trademark of Gupta Corporation (a California Corporation). Hewlett-Packard is a registered trademark of Hewlett-Packard Company. Intel is a registered trademark of Intel Corporation. IBM is a registered trademark of International Business Machines Corporation. ITT is a registered trademark of International Telephone and Telegraph Corporation. FoxPro, Microsoft, MS-DOS, PowerPoint, Visual Basic, Windows, and Windows NT are registered trademarks and Visual FoxPro is a trademark of Microsoft Corporation. Powersoft is a registered trademark and PowerBuilder is a trademark of PowerSoft Corporation. Raytheon is a registered trademark of Raytheon Company. Other product and company names mentioned herein may be the trademarks of their respective owners.

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/szypro>
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 do polskiego wydania	15
Przedmowa	17
Informacje o autorze	23
Część I Wydajne projektowanie	25
Rozdział 1. Witaj w świecie szybkiego projektowania	27
1.1. Czym jest szybkie projektowanie?	27
1.2. Skuteczne wdrożenie szybkiego projektowania	28
Rozdział 2. Strategia szybkiego projektowania	31
2.1. Ogólna strategia szybkiego projektowania	34
2.2. Cztery wymiary szybkości projektowania	36
2.3. Ogólne rodzaje szybkiego projektowania	44
2.4. Który wymiar ma największe znaczenie?	46
2.5. Alternatywna strategia szybkiego projektowania	48
Literatura uzupełniająca	53
Rozdział 3. Klasyczne błędy	55
3.1. Przykład popełniania klasycznych błędów	55
3.2. Wpływ błędów na harmonogram projektowania	62
3.3. Lista klasycznych błędów	63
3.4. Ucieczka z „wyspy Gilligana”	73
Literatura uzupełniająca	75
Rozdział 4. Podstawy projektowania	77
4.1. Podstawy zarządzania	80
4.2. Podstawy techniczne	85
4.3. Podstawy kontroli jakości	93
4.4. Postępowanie zgodne z instrukcjami	101
Literatura uzupełniająca	102
Rozdział 5. Zarządzanie ryzykiem	103
5.1. Elementy zarządzania ryzykiem	105
5.2. Identyfikacja ryzyka	107
5.3. Analiza ryzyka	112
5.4. Priorytetyzacja ryzyka	115

5.5. Kontrola ryzyka	117
5.6. Ryzyko, wysokie ryzyko i „loteria”	122
Literatura uzupełniająca	125
Część II Szybkie projektowanie	127
Rozdział 6. Główne problemy dotyczące szybkiego projektowania	129
6.1. Czy jeden rozmiar może być uniwersalny?	129
6.2. Jaki rodzaj szybkiego projektowania jest Ci potrzebny?	131
6.3. Szanse na ukończenie projektu w terminie	136
6.4. Postrzeganie a rzeczywistość	138
6.5. Tam, dokąd płynie wolno czas	140
6.6. Kompromisy w strategii szybkiego projektowania	144
6.7. Typowy schemat usprawniania harmonogramu	146
6.8. Droga ku szybkiemu projektowaniu	148
Literatura uzupełniająca	149
Rozdział 7. Planowanie cyklu życia	151
7.1. Klasyczny model kaskadowy	153
7.2. Nieprzemyślane pisanie kodu	157
7.3. Model spiralny	158
7.4. Zmodyfikowane modele kaskadowe	160
7.5. Prototypowanie ewolucyjne	163
7.6. Wieloetapowe dostarczanie produktu	164
7.7. Wytwarzanie dopasowane do harmonogramu	165
7.8. Ewolucyjne dostarczanie produktu	167
7.9. Wytwarzanie dopasowane do narzędzi	168
7.10. Oprogramowanie komercyjne	169
7.11. Wybór najszybszego cyklu życia dla Twojego projektu	170
Literatura uzupełniająca	177
Rozdział 8. Szacowanie	179
8.1. Opowieść o szacowaniu	181
8.2. Zarys procesu szacowania	188
8.3. Szacowanie rozmiaru	189
8.4. Szacowanie wysiłku	196
8.5. Szacowanie harmonogramu	197
8.6. Orientacyjne szacowanie harmonogramu	199
8.7. Zawężanie oszacowań	210
Literatura uzupełniająca	216

Rozdział 9. Sporządzanie harmonogramu	219
9.1. Tworzenie zbyt optymistycznych harmonogramów	220
9.2. Harmonogram pod presją	233
Literatura uzupełniająca	243
Rozdział 10. Projektowanie zorientowane na klienta	245
10.1. Wpływ klienta na strategię szybkiego projektowania	248
10.2. Rozwiązania zorientowane na klienta	250
10.3. Zarządzanie oczekiwaniami klienta	254
Literatura uzupełniająca	257
Rozdział 11. Motywacja	259
11.1. Typowe motywacje projektanta	261
11.2. Stosowanie pięciu najważniejszych czynników motywujących	264
11.3. Korzystanie z pozostałych czynników motywujących	270
11.4. Czynniki niszczące morale	273
Literatura uzupełniająca	279
Rozdział 12. Praca zespołowa	281
12.1. Zastosowania pracy zespołowej w inżynierii oprogramowania	283
12.2. Znaczenie pracy zespołowej w strategii szybkiego projektowania	284
12.3. Utworzenie wydajnego zespołu	286
12.4. Dlaczego zespoły zawodzą?	295
12.5. Długoterminowe budowanie zespołu	298
12.6. Wskazówki dotyczące budowania zespołu	300
Literatura uzupełniająca	301
Rozdział 13. Struktura zespołu	303
13.1. Czynniki wpływające na strukturę zespołu	305
13.2. Modele zespołów	308
13.3. Kierownicy a liderzy techniczni	317
Literatura uzupełniająca	320
Rozdział 14. Regulowanie zestawu funkcji	323
14.1. Początek projektu: redukcja zestawu funkcji	325
14.2. Środek projektu: kontrola przerostu funkcjonalności	334
14.3. Koniec projektu: usuwanie funkcji	343
Literatura uzupełniająca	345

Rozdział 15. Narzędzia zwiększające produktywność	347
15.1. Rola narzędzi zwiększających produktywność w strategii szybkiego projektowania	349
15.2. Strategia wykorzystywania narzędzi zwiększających produktywność	353
15.3. Nabywanie narzędzi zwiększających produktywność	355
15.4. Stosowanie narzędzi zwiększających produktywność	359
15.5. Syndrom „srebrnej kuli”	364
Literatura uzupełniająca	369
Rozdział 16. Ratowanie projektu	371
16.1. Sposoby ratowania projektu	373
16.2. Plan ratowania projektu	375
Literatura uzupełniająca	387
Część III Sprawdzone rozwiązania	389
Wprowadzenie do sprawdzonych rozwiązań	390
Układ rozdziałów opisujących sprawdzone rozwiązania	392
Podsumowanie metod branych pod uwagę jako sprawdzone rozwiązania	395
Podsumowanie sprawdzonych rozwiązań	400
Rozdział 17. Komisja zatwierdzająca zmiany	403
Rozdział 18. Codzienne kompilacje i testy dymowe	405
18.1. Stosowanie codziennych kompilacji i testów dymowych	407
18.2. Zarządzanie ryzykiem w procesie codziennych kompilacji i testów dymowych	412
18.3. Skutki uboczne korzystania z codziennych kompilacji i testów dymowych	412
18.4. Oddziaływanie codziennych kompilacji i testów dymowych z innymi metodami	413
18.5. Konkluzje	413
18.6. Czynniki decydujące o skutecznym stosowaniu codziennych kompilacji i testów dymowych	414
Literatura uzupełniająca	414
Rozdział 19. Przygotowanie architektury nastawione na zmianę	415
19.1. Stosowanie opisywanej metody	416
19.2. Zarządzanie ryzykiem podczas przygotowywania architektury nastawionego na zmianę	421
19.3. Skutki uboczne przygotowania architektury nastawionego na zmianę	422

19.4. Oddziaływanie przygotowania architektury nastawionego na zmianę z innymi metodami	422
19.5. Konkluzje	422
19.6. Czynniki decydujące o skutecznym stosowaniu przygotowania architektury nastawionego na zmianę	423
Literatura uzupełniająca	423
Rozdział 20. Ewolucyjne dostarczanie produktu	425
20.1. Stosowanie ewolucyjnego dostarczania produktu	427
20.2. Zarządzanie ryzykiem w modelu ewolucyjnego dostarczania produktu	429
20.3. Skutki uboczne ewolucyjnego dostarczania produktu	430
20.4. Oddziaływanie ewolucyjnego dostarczania produktu z innymi metodami	431
20.5. Konkluzje	431
20.6. Czynniki decydujące o skutecznym stosowaniu ewolucyjnego dostarczania produktu	432
Literatura uzupełniająca	432
Rozdział 21. Prototypowanie ewolucyjne	433
21.1. Stosowanie prototypowania ewolucyjnego	434
21.2. Zarządzanie ryzykiem w prototypowaniu ewolucyjnym	435
21.3. Skutki uboczne prototypowania ewolucyjnego	440
21.4. Oddziaływanie prototypowania ewolucyjnego z innymi metodami	440
21.5. Konkluzje	441
21.6. Czynniki decydujące o skutecznym stosowaniu prototypowania ewolucyjnego	441
Literatura uzupełniająca	442
Rozdział 22. Ustanawianie celu	443
Rozdział 23. Inspekcje	445
Rozdział 24. Sesje JAD	447
24.1. Stosowanie metodologii JAD	448
24.2. Zarządzanie ryzykiem w metodologii JAD	456
24.3. Skutki uboczne stosowania sesji JAD	457
24.4. Oddziaływania metodologii JAD z innymi rozwiązaniami	458
24.5. Konkluzje	458
25.6. Czynniki decydujące o skutecznym stosowaniu metodologii JAD	459
Literatura uzupełniająca	459

Rozdział 25. Wybór modelu cyklu życia	461
Rozdział 26. Pomiary	463
26.1. Stosowanie pomiarów	464
26.2. Zarządzanie ryzykiem w pomiarach	472
26.3. Skutki uboczne stosowania pomiarów	473
26.4. Oddziaływanie pomiarów z innymi metodami	473
26.5. Konkluzje	473
26.6. Czynniki decydujące o skutecznym stosowaniu pomiarów	474
Literatura uzupełniająca	474
Rozdział 27. Rozbijanie celów na podetapy	477
27.1. Stosowanie metody rozbijania celów na podetapy	480
27.2. Zarządzanie ryzykiem podczas rozbijania celów na podetapy	483
27.3. Skutki uboczne rozbijania celów na podetapy	483
27.4. Oddziaływanie rozbijania celów na podetapy z innymi metodami ...	483
27.5. Konkluzje	484
27.6. Czynniki decydujące o skutecznym rozbijaniu celów na podetapy ...	485
Literatura uzupełniająca	485
Rozdział 28. Zewnętrzni podwykonawcy	487
28.1. Wykorzystywanie zewnętrznych podwykonawców	489
28.2. Zarządzanie ryzykiem związanym z zewnętrznymi podwykonawcami .	495
28.3. Skutki uboczne zatrudniania zewnętrznych podwykonawców	496
28.4. Zatrudnianie zewnętrznych podwykonawców a inne metody	496
28.5. Konkluzje	497
28.6. Czynniki decydujące o skuteczności omawianej metody	497
Literatura uzupełniająca	497
Rozdział 29. Negocjacje zgodne z zasadami	499
Rozdział 30. Środowisko pracy	501
30.1. Zastosowania produktywnego środowiska pracy	503
30.2. Zarządzanie ryzykiem w produktywnym środowisku pracy	505
30.3. Skutki uboczne wprowadzenia produktywnego środowiska pracy ...	506
30.4. Oddziaływania środowiska pracy z innymi metodami	507
30.5. Konkluzje	507
30.6. Czynniki decydujące o skutecznym wdrożeniu produktywnego środowiska pracy	508
Literatura uzupełniająca	508

Rozdział 31. Języki szybkiego projektowania (RDL)	509
31.1. Stosowanie języków RDL	513
31.2. Zarządzanie ryzykiem podczas stosowania języków RDL	513
31.3. Skutki uboczne stosowania języków RDL	515
31.4. Oddziaływanie języków RDL z innymi rozwiązaniami	515
31.5. Konkluzje	516
31.6. Czynniki decydujące o skutecznym stosowaniu języków RDL	516
Literatura uzupełniająca	517
Rozdział 32. Przesiewanie wymagań	519
Rozdział 33. Wielokrotne wykorzystywanie zasobów	521
33.1. Stosowanie wielokrotnego wykorzystywania zasobów	522
33.2. Zarządzanie ryzykiem podczas wielokrotnego wykorzystywania zasobów	529
33.3. Skutki uboczne wielokrotnego wykorzystywania zasobów	530
33.4. Oddziaływanie wielokrotnego wykorzystywania zasobów z innymi metodami	530
33.5. Konkluzje	531
33.6. Czynniki decydujące o skutecznym, wielokrotnym wykorzystywaniu zasobów	531
Literatura uzupełniająca	532
Rozdział 34. Wspólny cel	533
34.1. Stosowanie wspólnego celu	534
34.2. Zarządzanie ryzykiem podczas ustanawiania wspólnego celu	536
34.3. Skutki uboczne wspólnego celu	538
34.4. Oddziaływanie wspólnego celu z innymi metodami	538
34.5. Konkluzje	538
34.6. Czynniki decydujące o sukcesie wspólnego celu	539
Literatura uzupełniająca	539
Rozdział 35. Spiralny model cyklu życia	541
Rozdział 36. Wieloetapowe dostarczanie produktu	543
36.1. Stosowanie wieloetapowego dostarczania produktu	546
36.2. Zarządzanie ryzykiem w wieloetapowym dostarczaniu produktu	549
36.3. Skutki uboczne wieloetapowego dostarczania produktu	550
36.4. Oddziaływanie wieloetapowego dostarczania produktu z innymi metodami	550

36.5. Konkluzje	551
36.6. Czynniki decydujące o skutecznym korzystaniu z wieloetapowego dostarczania produktu	552
Literatura uzupełniająca	552
Rozdział 37. Zarządzanie zgodne z teorią W	553
37.1. Stosowanie zarządzania zgodnego z teorią W	555
37.2. Zarządzanie ryzykiem w teorii W	560
37.3. Skutki uboczne zarządzania zgodnego z teorią W	561
37.4. Oddziaływanie teorii W z innymi metodami	561
37.5. Konkluzje	561
37.6. Czynniki decydujące o właściwym zarządzaniu zgodnym z teorią W ...	562
Literatura uzupełniająca	562
Rozdział 38. Prototypowanie z odrzuceniem	563
38.1. Stosowanie prototypowania z odrzuceniem	564
38.2. Zarządzanie ryzykiem w prototypowaniu z odrzuceniem	565
38.3. Skutki uboczne stosowania prototypowania z odrzuceniem	566
38.4. Oddziaływanie prototypowania z odrzuceniem z innymi metodami	566
38.5. Konkluzje	566
38.6. Czynniki decydujące o skutecznym stosowaniu prototypowania z odrzuceniem	567
Literatura uzupełniająca	567
Rozdział 39. Projektowanie metodą okienek czasowych	569
39.1. Stosowanie projektowania metodą okienek czasowych	571
39.2. Zarządzanie ryzykiem w projektowaniu metodą okienek czasowych .	574
39.3. Skutki uboczne stosowania metody okienek czasowych	575
39.4. Oddziaływanie projektowania metodą okienek czasowych z innymi rozwiązaniami	575
39.5. Konkluzje	576
39.6. Czynniki decydujące o skutecznym projektowaniu metodą okienek czasowych	576
Literatura uzupełniająca	577
Rozdział 40. Zespół narzędziowy	579
Rozdział 41. Lista 10 największych zagrożeń	581

Rozdział 42. Prototypowanie interfejsu użytkownika	583
42.1. Stosowanie prototypowania interfejsu użytkownika	585
42.2. Zarządzanie ryzykiem w prototypowaniu interfejsu użytkownika	588
42.3. Skutki uboczne prototypowania interfejsu użytkownika	589
42.4. Oddziaływanie prototypowania interfejsu użytkownika z innymi rozwiązaniami	590
42.5. Konkluzje	590
42.6. Czynniki decydujące o skutecznym prototypowaniu interfejsu użytkownika	590
Literatura uzupełniająca	591
Rozdział 43. Dobrowolna praca w nadgodzinach	593
43.1. Stosowanie pracy w dobrowolnych nadgodzinach	594
43.2. Zarządzanie ryzykiem przy dobrowolnej pracy w nadgodzinach	599
43.3. Skutki uboczne dobrowolnej pracy w nadgodzinach	600
43.4. Oddziaływanie dobrowolnej pracy w nadgodzinach z innymi rozwiązaniami	600
43.5. Konkluzje	600
43.6. Czynniki decydujące o skutecznym wdrożeniu dobrowolnej pracy w nadgodzinach	601
Literatura uzupełniająca	601
Bibliografia	603
Skorowidz	617

Klasyczne błędy

Zawartość rozdziału

- 3.1. Przykład popełniania klasycznych błędów
- 3.2. Wpływ błędów na harmonogram projektowania
- 3.3. Lista klasycznych błędów
- 3.4. Ucieczka z „wyspy Gilligana”

Tematy pokrewne

- „Zarządzanie ryzykiem”: rozdział 5.
- „Strategia szybkiego projektowania”: rozdział 2.

PROJEKTOWANIE OPROGRAMOWANIA JEST SKOMPLIKOWANYM DZIAŁANIEM.

Typowy projekt daje więcej okazji do wyciągania wniosków z błędów, niż zdarza się to niektórym ludziom przez całe życie. W tym rozdziale przyjrzymy się niektórym z klasycznych błędów, które przytrafiają się w trakcie szybkiego projektowania oprogramowania.

3.1. Przykład popełniania klasycznych błędów

Poniższy przykład przypomina nieco dzieciinną układankę, w której należy znaleźć wszystkie przedmioty o nazwach rozpoczynających się na literę „B”. Jak wiele klasycznych błędów jesteś w stanie wyłapać w opisie?

Przykład 3.1. Klasyczne błędy

W pewien kwietniowy, ciepły poranek Mike, lider techniczny w firmie Giga Safe, firmie zajmującej się ubezpieczeniami zdrowotnymi, jadł śniadanie w swoim biurze i spoglądał przez okno.

„Mike, dostałeś dotację na aplikację Giga-Quote! Gratulacje! — zawołał Bill, szef Mike’a w Giga Safe. — Kierownictwu bardzo spodobał się pomysł zautomatyzowania kosztów ubezpieczenia. Popiera również ideę codziennego przesyłania kosztów ubezpieczenia do centrali, dzięki czemu będziemy od razu mieli potencjalnych klientów w bazie danych. Muszę pędzić na spotkanie, ale później omówimy szczegóły. Dobrze się spisałeś z tą propozycją!”

Mike rozpiisał ofertę aplikacji Giga-Quote kilka miesięcy wcześniej, ale dotyczyła ona programu pozbawionego funkcji łączenia się z centralą. No cóż... Teraz otrzymał szansę poprowadzenia projektu aplikacji typu klient – serwer opracowanej w nowoczesny interfejs użytkownika — marzył o tym od zawsze. Otrzymał niemal rok na wykonanie projektu, a to mnóstwo czasu, żeby dodać nową funkcjonalność. Mike podniósł słuchawkę telefonu i zadzwonił do żony: „Kochanie, chodźmy gdzie dzisiaj na kolację. Mamy powód do świętowania...”

Mike i Bill spotkali się następnego dnia rano w celu omówienia projektu. „No dobra, Bill, o co tu chodzi? To nie jest do końca ta oferta, nad którą pracowałem”.

Bill poczuł ukłucie niepokoju. Mike nie brał udziału w poprawkach, ale nie było czasu, żeby go wtajemniczyć. Gdy tylko kierownictwo zapoznało się z koncepcją aplikacji Giga-Quote, przejęło inicjatywę. „Szefostwu bardzo spodobał się pomysł stworzenia oprogramowania automatyzującego proces zakupu ubezpieczenia zdrowotnego. Dyrektorzy wymagają jednak również automatycznego przesyłania zawartych transakcji do głównego serwera. Chcą też, aby system był gotowy przed wprowadzeniem nowych stawek w dniu 1 stycznia. Przesunęli proponowany przez ciebie termin ukończenia projektu z 1 marca na 1 listopada, co skraca harmonogram do sześciu miesięcy”.

Mike przewidywał, że projekt zajmie mu 12 miesięcy. Nie wierzył w szansę powodzenia przy sześciomiesięcznym harmonogramie i powiedział to Billowi. „Czegoś tu nie rozumiem — stwierdził. — Kierownictwo zażądało dodania rozbudowanej funkcjonalności sieciowej i skróciło harmonogram z dwunastu do sześciu miesięcy?”

Bill wzruszył ramionami. „Mam świadomość, że to będzie duże wyzwanie, ale jesteś kreatywny i wierzę, że sobie poradzisz. Szefowie zatwierdzili zaproponowany przez ciebie budżet, a dodanie łączy komunikacyjnych nie powinno być wcale takie trudne. Prosiłeś o trzydziestostosześciokrotność wynagrodzenia zespołu i ją dostałeś. Możesz zatrudnić, kogo tylko zechcesz, by powiększyć zespół”. Następnie polecił Mike’owi, żeby się spotkał z innymi programistami i ustalił skuteczny sposób terminowego dostarczenia produktu.

Mike spotkał się z Carlem, innym liderem technicznym, i razem zastanawiali się nad sposobami skrócenia czasu projektowania. „Skorzystaj może z języka C++ i programowania obiektowego — zaproponował Carl. — Dzięki temu będziesz wydajniejszy, co powinno pozwolić na zredukowanie harmonogramu o miesiąc bądź dwa”. Nasz bohater uznał to za dobry pomysł. Carl wspomniał również o istnieniu narzędzia raportującego, które powinno skrócić czas projektowania o połowę. Projekt zawierał mnóstwo raportów, dlatego obydwaj rozwiązania pozwoliłyby ustanowić dziewięciomiesięczny harmonogram. Zakup nowego, szybkiego sprzętu pozwoliłby zyskać dodatkowe kilka tygodni. Gdyby Mike zatrudnił najlepszych programistów w branży, zjechałby z terminem ukończenia projektu do siedmiu miesięcy. To był już całkiem niezły plan. Mike podzielił się tymi koncepcjami z Billem.

„Słuchaj — powiedział Bill — dobrze, że udało się wam skrócić termin do siedmiu miesięcy, ale to wciąż za długo. Zarząd dał jasno do zrozumienia, że zależy im na półrocznym harmonogramie. Nie dali mi wyboru. Mogę wam załatwić nowy sprzęt, ale musicie z zespołem w jakiś sposób zmieścić się z projektem w sześciu miesiącach lub pracować w nadgodzinach, jeśli to pomoże”.

Mike uznał, że być może jego pierwotne oszacowanie harmonogramu było wyolbrzymione i że jednak uda mu się zakończyć projekt w pół roku. „W porządku, Bill, wynajmę dwóch podwykonawców do tego projektu. Może uda nam się znaleźć specjalistów w dziedzinie komunikacji, którzy pomogą nam w przesyłaniu danych z komputerów do serwera”.

Do 1 maja Mike zebrał zespół. Jill, Sue i Thomas byli solidnymi programistami zatrudnionymi w tej samej firmie, a tak się składało, że nie przydzielono ich w tym czasie do żadnego projektu. Dodał do zespołu dwoje wolnych strzelców — Keiko i Chipa. Keiko miała doświadczenie zarówno w komputerach stacjonarnych, jak i w typie serwera, na którym zespół opierał strukturę systemu. Jill z Thomasem przeprowadzili rozmowę kwalifikacyjną z Chipem i odradzali przyjęcie go do zespołu, ale Mike’owi zaimponowała jego wiedza. Znał się na komunikacji i był dostępny, więc pomimo sprzeciwów zatrudniono go.

Na pierwszym wspólnym spotkaniu Bill powiedział zespołowi, że aplikacja Giga-Quote ma strategiczne znaczenie dla firmy. Pewni wysocy szczeblem pracownicy będą im się uważnie przyglądać. Jeżeli zespołowi się powiedzie, zarząd nie będzie skąpił nagród. Przyznał, że wierzy w możliwości poszczególnych członków.

Po przemowie motywacyjnej Billa Mike zasiadł z zespołem i wprowadził wszystkich w szczegóły harmonogramu. Zarząd dał im coś na kształt specyfikacji i mieli dwa tygodnie na wypełnienie wszelkich luk. Następne sześć tygodni mieli poświęcić na stworzenie schematu aplikacji, po czym zostałyby im cztery miesiące na jej stworzenie i testowanie. Intuicja podpowiadała Mike'owi, że program w ostatecznej postaci zawierałby ok. 30 tys. wierszy kodu napisanego w języku C++. Pozostali członkowie zgodzili się z tym. Wyzwanie było ambitne, ale wszyscy od początku byli tego świadomi.

W następnym tygodniu Mike spotkał się ze Stacy, główną testerką. Wyjaśniła, że powinni przysłać prototypy aplikacji do testowania nie później niż do 1 września, a gotową wersję przygotować do 1 października. Mike wyraził zgodę.

Zespół szybko uwinął się ze specyfikacją wymagań i zabrał się za tworzenie schematu aplikacji. Przygotowali zarys programu, który mógł w właściwy sposób korzystać z możliwości języka C++.

Do 15 czerwca mieli gotowy zarys aplikacji, co oznaczało, że skończyli ten etap przed czasem i jak szaleni zabrali się za programowanie, gdyż chcieli zdążyć z wypuszczeniem pierwszej wersji testowej przed 1 września. Nie wszystko szło tak gładko, jak należy. Jill i Thomas nie przepadali za Chipem, Sue również narzekała, że chłopak nie dopuszcza nikogo do swojej partii kodu. Mike uważał, że programiści spędzali ze sobą zbyt wiele czasu w pracy, co stanowiło przyczynę konfliktu charakterów. Bez względu na animozje na początku lipca stwierdzili, że mają przygotowane już 85 – 90% aplikacji.

W połowie sierpnia dział aktuarialny wydał stawki obowiązujące na przyszły rok i zespół stwierdził, że musi dostosować system do zupełnie nowej struktury stawek. Nowa metoda obliczania kosztów wprowadzała pytania o takie czynniki jak częstotliwość aktywności fizycznej, uzależnienie od tytoniu i alkoholu, zainteresowania oraz inne elementy, których nie było wcześniej we wzorach na stawkę ubezpieczeniową. Zespół Mike'a pomyślał, że struktura języka C++ powinna ich uchronić przed wpływem takich zmian. Zamierzali po prostu dodać nowe współczynniki do tabeli kosztów. Musieli jednak pozmienić wygląd okien dialogowych, projekt bazy danych, dostęp do niej, a także obiekty komunikacyjne, żeby przystosować całość do nowej struktury. Zespół zabrał się za wprowadzanie poprawek, a Mike powiedział Stacy, że mogą zaliczyć kilka dni poślizgu, zanim przygotują wersję przeznaczoną do testowania.

Wersja testowa nie była gotowa 1 września, a Mike zapewnił Stacy, że zostanie dostarczona w ciągu maksymalnie dwóch dni.

Dni zmieniły się w tygodnie. Termin oddania gotowego projektu do testowania (1 października) nadszedł i przeminął. Programiści ciągle nie przygotowali pierwszej wersji testowej. Stacy umówiła się z Billem na przedyskutowanie harmonogramu. „Nie otrzymaliśmy jeszcze żadnej wersji do testowania — powiedziała. — Mieliśmy ją dostać do 1 września, ale nic takiego nie nastąpiło. Wygląda na to, że mają przynajmniej jeden miesiąc poślizgu. Obawiam się, że trapią ich jakieś kłopoty”.

„Żebyś wiedziała, że mają kłopoty — odparł Bill. — Muszę porozmawiać z zespołem. Obiecałem sześciuset agentom, że do pierwszego listopada otrzymają tę aplikację. Musimy ją wypuścić przed zmianą stawek”.

Bill zwołał spotkanie zespołu. „Jesteście świetną ekipą i powinniście wywiązywać się ze swoich zobowiązań — powiedział im. — Nie wiem, co poszło źle, ale oczekuję od was wszystkich ciężkiej pracy oraz że dostarczycie oprogramowanie na czas. Ciągłe możecie otrzymać nagrody, ale musicie sobie na nie zapracować. Od teraz aż do zakończenia projektu każde z was jest zobowiązane do dziesięciogodzinnego dnia pracy przez sześć dni w tygodniu”. Po spotkaniu Jill i Thomas burknęli Mike’owi, że nie trzeba traktować ich jak dzieci, ale zgodzili się pracować w wyznaczony sposób.

Zespół przesunął termin zakończenia projektu o dwa tygodnie, gwarantując, że przygotują gotowy produkt do 15 listopada. Testerzy mieliby sześć tygodni na jego sprawdzenie, zanim w styczniu zostałyby wprowadzone nowe stawki.

Pierwsza wersja testowa aplikacji pojawiła się cztery tygodnie później — 1 listopada. Członkowie zespołu zebrali się, aby omówić kilka istniejących problemów.

Tomas pracował nad generowaniem raportów i natrafił na przeszkodę. „Strona z podsumowaniem zawiera prosty wykres kolumnowy. Korzystam z generatora raportów, który jest w stanie go wyświetlać, ale przekształca całą stronę w taki wykres. Jednym z wymogów jest umieszczenie wykresu oraz tekstu na tej samej stronie. Odkryłem, że mogę oszukać system, umieszczając tekst raportu jako legendę w obiekcie wykresu. Jest to oszustwo, ale po wydaniu pierwszej wersji mogę spróbować wprowadzić jakieś bardziej formalne rozwiązanie”.

Mike odparł: „Nie wiem, w czym widzisz problem. Mamy wydać produkt i nie mamy czasu, żeby doszlifowywać kod. Bill wyraził się wystarczająco jasno, że nie możemy sobie pozwolić na kolejne poślizgi. Oszukaj system”.

Chip poinformował, że jego moduł komunikacyjny jest gotowy w 95%, ale jeszcze musi przeprowadzić kilka testów. Mike zauważył, jak Jill i Tomas przewracają oczami, ale zignorował to.

Zespół pracował ciężko do 15 listopada, w tym niemal przez całe dwie noce tuż przed końcem terminu, ciągle jednak nie był w stanie wydać ostatecznej wersji. Ludzie byli wycieńczeni, ale to Bill poczuł się bardzo źle rankiem 16 listopada. Stacy poinformowała go telefonicznie, że zespół programistów nie przesłał jej poprzedniego dnia aplikacji do przetestowania. Tydzień wcześniej Bill zapewnił kierownictwo, że wszystko idzie zgodnie z planem. Inna kierowniczką projektu, Claire, przyjrzała się postępowi czynionym przez zespół i stwierdziła, że doszły ją słuchy o poślizgach w terminowym dostarczaniu aplikacji do testowania. Bill nie lubił Claire, ponieważ uważał, że jest zbyt nerwowa. Zapewnił ją, że jego zespół nie ma problemu z harmonogramem.

Bill kazał Mike’owi zebrać zespół, a gdy przyjrzał się ludziom, stwierdził, że wyglądają na pokonanych. Półtora miesiąca 60-godzinnych tygodni pracy zebrało swoje żniwo. Mike zapytał, o której godzinie tego dnia można się spodziewać gotowego produktu, ale odpowiedziała mu cisza. „Co chcesz mi przez to powiedzieć? — zapytał. — Skończymy dzisiaj tę ostateczną wersję, prawda?”.

„Słuchaj, Mike — zaczął Tomas — mogę oddać dzisiaj swój moduł i powiedzieć, że jest »gotowy«, ale prawdopodobnie czekałyby mnie po tym trzy tygodnie jego poprawiania”. Mike zapytał, co Tomas ma na myśli, mówiąc „poprawianie”. „Nie wstawiłem logo firmy na każdej stronie, nie zdążyłem też wstawić danych kontaktowych agenta w stopce wydruku. To nic wielkiego. Wszystkie najważniejsze funkcje działają jak należy. Mój moduł jest gotowy w dziewięćdziesięciu dziewięciu procentach”.

„Ja też jeszcze nie przygotowałam wszystkiego do końca — dodała Jill. — Moja poprzednia grupa wydzwaniała ostatnio często do mnie z prośbą o wsparcie techniczne i codziennie pomagałam im przez bite dwie godziny. Do tego Tomas właśnie mi przypomniał, że mieliśmy udostępnić agentom możliwość dodawania swoich danych osobowych do raportów. Jeszcze nie zaimplementowałam okien dialogowych odpowiedzialnych za tę funkcję, a także kilku innych elementów interfejsu. Nie sądziłam, że będą one potrzebne w tym momencie”.

Teraz z kolei Mike’owi zrobiło się słabo. „Jeżeli nic mi się nie stało z uszami i dobrze słyszę, to chcecie mi powiedzieć, że będziemy mieli gotowy, funkcjonalny produkt za trzy tygodnie. Zgadza się?”

„Najwcześniej za trzy tygodnie” — sprecyzowała Jill. Pozostali programiści przytaknęli. Mike obszedł stół i zapytał z osobna każdego członka zespołu, czy skończy swoje zadania w ciągu trzech tygodni. Wszyscy stwierdzili, że przy odpowiednio ciężkiej pracy jest to osiągalne.

Tego samego dnia po długiej, nieprzyjemnej rozmowie Mike i Bill zgodzili się przedłużyć harmonogram o trzy tygodnie, do 5 grudnia, pod warunkiem że wszyscy członkowie zespołu będą pracować nie po 10, a po 12 godzin dziennie. Bill wyjaśnił, że szefostwo chce widzieć presję wywieraną na zespół. Zmodyfikowany harmonogram oznaczał, że firma musiała jednocześnie testować aplikację i szkolić agentów ubezpieczeniowych — był to jedyny sposób, aby 1 stycznia wypuścić aplikację na rynek. Stacy narzekała, że dział kontroli jakości nie zdąży w tym czasie przetestować produktu, ale została przegłosowana.

Zgodnie z ustaleniami 5 grudnia twórcy aplikacji Giga-Quote oddali przed południem produkt do testowania i wyszli z pracy wcześniej na zasłużony odpoczynek. Pracowali niemal bez przerwy od 1 września.

Dwa dni później Stacy opublikowała pierwszą listę błędów i rozpoczęło się piekło. W ciągu tych dwóch dni zespół testerów wykrył ponad 200 usterek, w tym 23 błędy zaliczane do najwyższej kategorii — „konieczne do naprawienia”. „Nie wyobrażam sobie, żeby to oprogramowanie było gotowe do przekazania agentom pierwszego stycznia — podsumowała. — Prawdopodobnie tyle czasu zajmie mojemu zespołowi napisanie testów regresyjnych dla już odkrytych usterek, a z każdą chwilą znajdujemy kolejne”.

Mike zwołał zebranie zespołu na ósmą rano następnego dnia. Programiści byli rozdrażnieni. Twierdzili, że pomimo kilku poważnych problemów wiele zgłoszonych błędów wcale nie było błędami, a wynikiem niewłaściwego zrozumienia działania programu. Tomas jako przykład podał błąd nr 143: „W raporcie dotyczącym tego błędu stwierdzono, że na stronie podsumowania wykres słupkowy powinien się znajdować po prawej stronie, a nie po lewej. Tego nie można nazwać błędem najwyższej kategorii. To klasyczny przykład przewrażliwienia działu kontroli jakości”.

Mike przekazał każdemu członkowi zespołu kopię raportu. Programiści mieli przeanalizować zawarte tam problemy i oszacować czas potrzebny na ich wyeliminowanie.

Na popołudniowym spotkaniu nie usłyszał dobrych wieści. „Jeżeli mam być realistką, to usunięcie zgłoszonych do tej pory błędów zajmie mi około dwóch tygodni — powiedziała Sue. — Do tego muszę dokończyć algorytmy sprawdzania poprawności bazy danych. Łącznie czekają mnie cztery tygodnie pracy”.

Tomas przesłał błąd numer 143 do ponownej analizy, domagając się zmiany jego priorytetu z najwyższej kategorii na priorytet trzeciego poziomu — „zmiany kosmetyczne”. Dział kontroli jakości wyjaśnił, że raporty podsumowań w aplikacji Giga-Quote powinny wyglądać tak samo, jak raporty generowane przez aplikację odpowiedzialną za przedłużanie polis, które z kolei

miały taki sam styl jak stosowane od lat materiały promocyjne. Agenci ubezpieczeniowi byliprzyzwyczajeni do materiałów zawierających wykres słupkowy po prawej stronie, musiał więc on pozostać w tym miejscu. Utrzymano najwyższy priorytet błędu, co spowodowało dalsze problemy.

„Pamiętasz to oszustwo z wyświetlaniem wykresu i raportu na tej samej stronie? — zapytał Tomas. — Żeby umieścić wykres po prawej stronie, muszę całkowicie przeredagować ten konkretny raport, co oznacza, że muszę stworzyć moduł odpowiedzialny za formatowanie tekstu i grafikę”. Mike się wzdrygnął i poprosił o przybliżony czas tworzenia modułu. Tomas odpowiedział, że zajmie mu to przynajmniej 10 dni, ale musi jeszcze dokładniej się zastanowić nad problemem, żeby się upewnić.

Przed pójściem do domu Mike oznajmił Stacy i Billowi, że zespół będzie pracował w trakcie przerwy świątecznej i usunie wszystkie zgłoszone błędy do 7 stycznia. Bill powiedział, że spodziewał się tego i zatwierdził czterotygodniowy poślizg przed wyruszeniem na trwającą miesiąc wycieczkę na Karaiby, którą planował już od poprzednich wakacji.

Mike spędził cały miesiąc, trzymając swoich żołnierzy w ryzach. Przez cztery miesiące pracowali tak ciężko, jak mogli, i sądził, że już się nie da mocniej ich przycisnąć. Przebywali w biurze po 12 godzin dziennie, ale dużo czasu spędzali na czytaniu gazet, opłacaniu rachunków i rozmowach telefonicznych. Wydawało się, że do furii doprowadza ich każde pytanie dotyczące czasu potrzebnego do usunięcia kolejnych błędów. Na jeden wyeliminowany błąd przypadały dwa nowe. Usterki, których usunięcie powinno zająć kilka minut, miały wpływ na cały projekt, przez co programiści głowili się nad nimi przez wiele dni. Po niedługim czasie zrozumieli, że wyeliminowanie wszystkich błędów do 7 stycznia jest nierealne.

Tego dnia Bill wrócił z urlopu i Mike przekazał mu informację, że zespół będzie potrzebował dodatkowe cztery tygodnie na usunięcie wszystkich usterek. „Chyba żartujesz — odparł Bill. — Mam na głowie sześciuset agentów, którzy mają dość wodzenia za nos przez bandę komputerowców. Zarząd rozważa możliwość anulowania projektu. Musisz w jakiś sposób dostarczyć ten produkt w ciągu dwóch tygodni, bez względu na koszty”.

Mike zwołał zebranie zespołu, żeby rozważyć możliwości. Przekazał im ultimatum Billa, a następnie poprosił, żeby oszacowali przybliżoną datę wydania produktu, najpierw w tygodniach, a później — miesiącach. Nikt się nie odezwał. Żaden członek zespołu nie chciał ryzykować podania terminu, którego prawdopodobnie i tak nie mogliby dotrzymać. Mike nie wiedział, co powiedzieć Billowi.

Po spotkaniu Chip oznajmił Mike’owi, że podpisał kontrakt z inną firmą, który znacznie obowiązywać od 3 lutego. Mike po raz pierwszy poczuł, że może byłoby lepiej, gdyby projekt rzeczywiście został anulowany.

Mike zlecił Kipowi, programiście odpowiedzialnemu za architekturę serwera, pomoc w pozostałych modułach i przydzielił go do eliminacji błędów występujących po stronie komunikacyjnej komputerów klienckich. Kip walczył przez tydzień z kodem napisanym przez Chipa i zrozumiał, że zawiera on pewne podstawowe wady koncepcyjne, przez które moduł nie ma prawa właściwie działać. Został zmuszony do ponownego zaprojektowania i zaimplementowania klienckiej części łączą komunikacyjnego.

W połowie lutego odbyło się spotkanie zarządu, na którym Claire stwierdziła, że usłyszała już wystarczająco wiele i zaproponowała „zaprzestanie” projektu Giga-Quote. Spotkała się w piątek z Mike’em. „Ten projekt wymknął się spod kontroli — zaczęła. — Od miesiący nie

dostałam od Billa żadnych rzetelnych wytycznych do harmonogramu. Miał to być projekt trwający pół roku, a już zaliczyliśmy trzymiesięczny poślizg bez perspektywy ukończenia go w najbliższym czasie. Przyjrzałam się statystykom błędów i twój zespół nie radzi sobie z ich usuwaniem. Pracujecie już tak długo, że teraz nie czynicie żadnych postępów. Daję wam wszystkim wolny weekend; natomiast po powrocie masz sporządzić szczegółowy raport, który ma zawierać wszystkie, powtarzam: *wszystkie*, elementy pozostałe do ukończenia projektu. Nie chcę, abyś na siłę wymyślał jakiś sztuczny harmonogram. Jeżeli tworzenie aplikacji zajmie kolejne dziewięć miesięcy, muszą o tym wiedzieć. Przygotuj ten raport do środowego popołudnia. Nie musi być elegancki, ale ma być kompletny”.

Zespół był bardzo zadowolony z wolnego weekendu i członkowie w ciągu następnego tygodnia przystąpili do sporządzania raportu ze zdwojoną mocą. W środę wyłądował na biurku Claire. Dała go do przeanalizowania Charlesowi — konsultantowi do spraw inżynierii oprogramowania, który również przejrzał raport błędów. Zalecił, aby zespół skupił się na kilku najwrażliwszych modułach, natychmiast przygotował analizę wszystkich błędów projektowych i programistycznych, a także żeby programiści przestali pracować w nadgodzinach, dzięki czemu mogliby dokładnie oszacować, ile wysiłku zostało włożonego w projekt oraz ile jeszcze zostało czasu do jego zakończenia.

Trzy tygodnie później, w pierwszym tygodniu marca, lista błędów po raz pierwszy zaczęła się kurczyć. Morale zespołu nieco się podniosło, a konsultant na podstawie jednostajnych postępów oszacował termin dostarczenia gotowego i przetestowanego produktu do 15 maja. Kolejny wzrost stawek miał wejść w życie 1 lipca, dlatego Claire ustaliła 1 czerwca jako oficjalną datę wydania aplikacji.

Epilog

Program Giga-Quote dostarczono agentom zgodnie z planem, 1 czerwca. Został przyjęty ciepło, chociaż z pewną dozą sceptycyzmu.

Korporacja Giga Safe doceniła ciężką pracę włożoną w projekt przez członków zespołu, dając każdemu z nich premię w wysokości 250 dolarów. Kilka tygodni później Tomas wziął długi urlop, a Jill przeniosła się do innej firmy.

Stworzenie aplikacji Giga-Quote zajęło ostatecznie nie sześć, a 13 miesięcy — poślizg wyniósł ponad 100% pierwotnego planu. Liczba osobomiesięcy (wliczając nadgodziny) wyniosła 98 miesięcy, czyli została przekroczona o 170% w porównaniu do pierwotnie zakładanych 36 osobomiesięcy.

Odnośniki: Więcej informacji na temat orientacyjnego szacowania terminu wykonania projektów o różnych rozmiarach znajdziesz w podrozdziale 8.6. „Orientacyjne szacowanie harmonogramu”.

Wyliczono, że ostateczny produkt składał się z ok. 40 tys. niepustych, niebędących komentarzami wierszy kodu napisanego w języku C++, co stanowiło wartość o 33% większą od rozmiaru zakładanego przez Mike’a. Aplikacja stanowiła hybrydę produktu biznesowego i „celofanowego”, ponieważ została rozprowadzona do 600 punktów w kraju. Produkt tego typu i wielkości jest tworzony przeciętnie przez 11,5 miesiąca oraz wymaga włożenia wysiłku równego 71 miesiący pracy zespołowej. Jak widać, zostały przekroczone obydwie wartości nominalne.

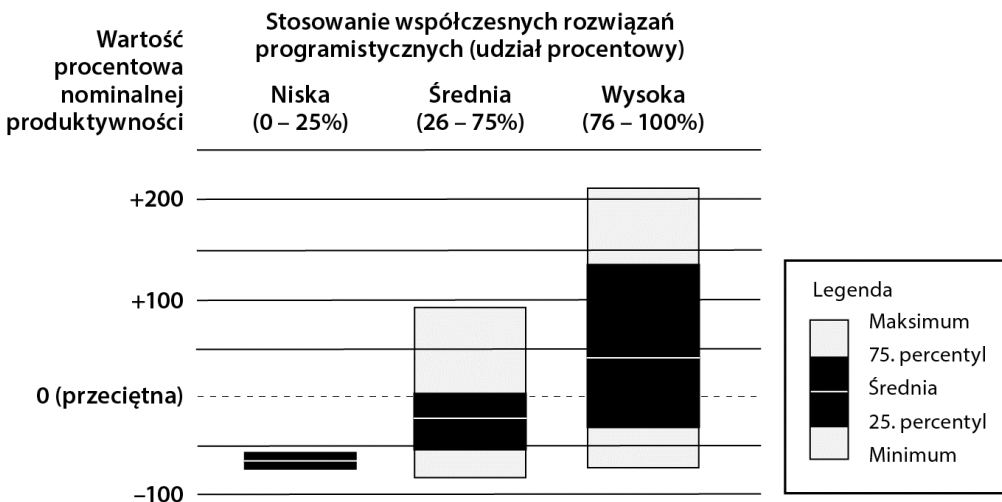
3.2. Wpływ błędów na harmonogram projektowania

Michael Jackson (piosenkarz, nie informatyk) śpiewał, że „jedno zgniłe jabłko nie zepsuje pozostałych, małeńka”¹. Być może jest to prawda w przypadku jabłek, ale nie dla oprogramowania. Jedno zgniłe jabłko *może* zepsuć cały projekt.

Grupa naukowców z instytutu technicznego ITT przeanalizowała 44 projekty z dziewięciu krajów w celu ustalenia wpływu 13 czynników na produktywność (Vosburgh i in. 1984). Obserwowano takie czynniki, jak wykorzystanie współczesnych rozwiązań programistycznych, poziom zaawansowania kodu, wymagania dotyczące wydajności, udział klienta w ustalaniu specyfikacji produktu, doświadczenie pracowników i kilka innych. Badacze podzielili czynniki na kategorie związane z małą, średnią i dużą wydajnością. Przykładowo: rozdzielili czynnik *współczesne rozwiązania programistyczne* na *niewielkie, średnie i częste* zastosowanie. Na rysunku 3.1 widzimy wyniki badań przeprowadzonych na czynniku *współczesne rozwiązania programistyczne*.

Im dłużej przyglądasz się wykresowi z rysunku 3.1, tym robi się on bardziej interesujący. Widoczny na nim wzór odpowiada zasadniczo wszystkim analizowanym czynnikom. Naukowcy odkryli, że projekty należące do kategorii składających się na niską produktywność rzeczywiście nie były zbyt wydajne — słupek ukazany w kategorii *niska* na rysunku 3.1. Jednakże produktywność projektów o wysokiej wydajności cechowała olbrzymia rozbieżność — słupek w kategorii *wysoka*. W tej kategorii wydajność projektów plasowała się pomiędzy *niską* a *znakomitą*.

Odnosiniki: Więcej informacji na temat omawianego wykresu znajdziesz w podrozdziale 4.2. „Podstawy techniczne”.



Rysunek 3.1. Wnioski dotyczące czynnika „wykorzystanie współczesnych rozwiązań programistycznych” (Vosburgh i in. 1984). Właściwe wdrożenie kilku elementów wcale nie gwarantuje szybkiego projektowania. Trzeba również unikać popełniania błędów

¹ W oryginale: „One bad apple don’t spoil the whole bunch, baby” z piosenki *One Bad Apple* — *przyp. tłum.*

Odnosiniki: Więcej informacji dotyczących wpływu błędów na proces szybkiego projektowania znajdziesz w podrozdziale 2.1. „Ogólna strategia szybkiego projektowania”.

Nie powinno dziwić, że projekty o niskiej produktywności osiągnęły takie a nie inne wyniki. Jednak zaskakujący jest fakt, że wiele projektów zorientowanych na wysoką produktywność w rzeczywistości kiepsko sobie radzi. Celem powyższego i innych wykresów umieszczonych w tej książce jest ukazanie, że skuteczne rozwiązania są warunkiem koniecznym, ale niewystarczającym do osiągnięcia maksymalnej szybkości projektowania. Nawet jeżeli stosujesz pewne elementy właściwie (np. wydajnie korzystasz z nowoczesnych rozwiązań programistycznych), wciąż możesz popełnić błąd, który zniweluje korzystny wpływ danego czynnika.

Przy zastosowaniu strategii szybkiego projektowania pojawia się kusząca myśl, że wystarczy zidentyfikować i wyeliminować przyczyny powolnego tworzenia, a produkt powstanie znacznie szybciej. Problem polega na tym, że istnieją również inne czynniki wpływające na szybkość projektowania i ostatecznie wyeliminowanie podstawowych przyczyn na wiele się nie zda. Przypomina to trochę pytanie: „Dlaczego nie jestem w stanie przebiec jednego kilometra w trzy minuty?”. Jestem za stary. Za dużo ważę. Nie mam dobrej kondycji. Nie chce mi się trenować. Nie mam światowej klasy trenera ani odpowiedniego obiektu treningowego. Nawet za młodu nie byłem wystarczająco szybki. Przyczyny można wymieniać w nieskończoność.

Gdy mówimy o wyjątkowych osiągnięciach, pojawia się zbyt wiele przyczyn, dla których ludzie nie osiągają najwyższej formy. Omawiany w przykładzie 3.1 zespół twórców aplikacji Giga-Quote popełnił wiele błędów trapiących świat programistów od zarania dziejów informatyki. Ścieżka inżynierii oprogramowania jest bardzo wyboista, a od tych wybojów w dużej mierze zależy, jak szybko jesteś w stanie zaprojektować oprogramowanie.

W przypadku oprogramowania jedno zgniłe jabłko może zepsuć pozostałe, małeńka. Wystarczy popełnić jeden naprawdę poważny błąd, aby zlecieć do czeluści powolnego programowania; w celu osiągnięcia szybkiego projektowania musisz wystrzegać się popełniania *jakichkolwiek* większych pomyłek. W następnym podrozdziale przyjrzymy się najczęściej popełnianym dużym błędom.

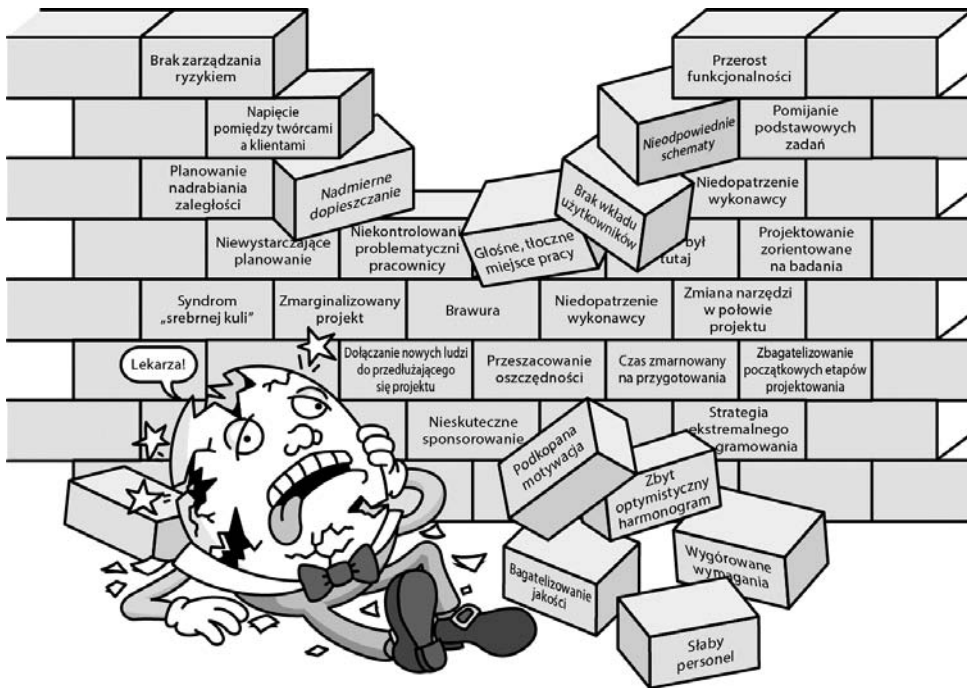
3.3. Lista klasycznych błędów



Klasyczny błąd

Pewne nieskuteczne rozwiązania projektowe są wybierane tak często, przez tak wiele osób i zawsze dają tak samo marne wyniki, że zasługują na nazwanie ich „klasycznymi błędami”. Większość błędów przybrana jest kuszące szaty. Czy musisz uratować spóźniający się projekt? Zatrudnij więcej osób! Czy chcesz skrócić harmonogram? Wymuś większy wysiłek! Czy jeden z pracowników irytuje pozostałych? Poczekaj do końca projektu i wylej go!

Czy musisz się śpieszyć z wydaniem projektu? Zatrudnij wszelkich dostępnych programistów i zacznij działać od zaraz!



Rysunek 3.2. Projekt był najeżony błędami i żaden z kierowników ani liderów nie był w stanie go uratować

Zazwyczaj programiści, kierownicy i klienci mają dobre powody, aby podejmować określone decyzje, a kuszący wygląd klasycznych pułapek stanowi jedną z przyczyn tak częstego ich popełniania. Jednak z powodu tak dużej częstości występowania takich błędów można bardzo łatwo przewidzieć ich skutki. Klasyczne błędy bardzo rzadko przynoszą zamierzone efekty.

Opisuję w tym podrozdziale niemal 40 rodzajów klasycznych błędów. W swojej karierze natrafiłem przynajmniej raz na każdy z nich, a sam mam kilka na sumieniu. Wiele z nich pojawia się w przykładzie 3.1. Wspólnym mianownikiem wszystkich wymienionych tu pomyłek jest fakt, że ich uniknięcie niekoniecznie powoduje skrócenie czasu projektowania, ale ich popełnianie z pewnością go wydłuża.

Jeżeli niektóre z tych błędów wydają się podejrzanie znajome, nie martw się — popełnia je również wiele innych osób. Gdy zrozumiesz ich wpływ na szybkość projektowania, będziesz mogła/mógł wykorzystać tę listę przy sporządzaniu planu projektu oraz zarządzaniu ryzykiem.

Poważniejsze błędy zostały dokładniej omówione w innych częściach książki, innym z kolei nie poświęcam tak wielkiej uwagi. W celu zwiększenia przejrzystości podzieliłem ich przynależność do poszczególnych wymiarów decydujących o szybkości projektowania: czynnika ludzkiego, procesu, produktu i technologii.

Czynnik ludzki

Poniżej prezentuję niektóre klasyczne błędy związane z ludźmi.

Odnosiniki: Więcej informacji na temat prawidłowego i nieprawidłowego stosowania motywacji znajdziesz w rozdziale 11. „Motywacja”.

1. Podkopana motywacja. Poszczególne badania wskazywały, że czynnikiem mającym prawdopodobnie największy wpływ na produktywność i jakość jest motywacja (Boehm 1981). W przykładzie 3.1 kierownictwo podjęło czynności systematycznie podkopujące motywację w czasie trwania projektu — rozpoczęło się od płomiennej przemowy, następnie w środku zmuszono programistów do pracy w nadgodzinach, jeden z dyrektorów wyjechał na urlop w ciężkim dla zespołu momencie, na koniec zaś członkowie zespołu otrzymali premię równowartości nieprzekraczającej jednego dolara za każdą nadgodzinę.

2. Słaby personel. Zaraz po motywacji prawdopodobnie największy wpływ na produktywność mają indywidualne umiejętności poszczególnych członków oraz zdolność do pracy zespołowej (Boehm 1981; Lakhanpal 1993). Zatrudnianie pracowników o niskich kompetencjach zagraża wszelkim próbom wprowadzenia strategii szybkiego projektowania. W przykładzie 3.1 pracownicy byli dobierani pod kątem dostępności, a nie pracowitości. Początek tworzenia produktu może przebiegać zrazu dość szybko, ale zazwyczaj takie tempo nie utrzymuje się do końca.

Odnosiniki: Więcej informacji na temat tworzenia wydajnych zespołów znajdziesz w rozdziale 12. „Praca zespołowa”.

3. Niekontrolowani, stwarzający problemy pracownicy. Brak kontroli nad sprawiającymi problemy pracownikami również zagraża szybkości projektowania. Jest to dość powszechny kłopot, dogłębnie przeanalizowany od czasu wydania w 1971 r. książki *Psychology of Computer Programming* autorstwa Geralda Weinberga. Najczęstszym zarzutem, jakim są obarczani liderzy przez członków zespołu, jest nieumiejętne radzenie sobie z krnąbrnym pracownikiem (Larson i LaFasto 1989). W przykładzie 3.1 zespół wiedział, że Chip jest „zgniłym jabłkiem”, ale lidera to nie obchodziło. Rezultat — całkowite przerabianie modułu Chipa — był do przewidzenia.

Odnosiniki: Więcej informacji na temat brawury i projektów opartych na zaangażowaniu znajdziesz w podrozdziale 2.5. „Alternatywna strategia szybkiego projektowania”, w punkcie „Sporządzanie harmonogramu zorientowane na poświęcenie” (podrozdział 8.5) oraz w rozdziale 34. „Wspólny cel”.

4. Brawura. Niektórzy inżynierowie oprogramowania kładą duży nacisk na indywidualne osiągnięcia w projekcie (Bach 1995). Uważam, że takie brawurowe podejście przynosi więcej szkody niż pożytku. W przykładzie 3.1 kierownik projektu skupił się na wprowadzaniu pozytywnego nastawienia, a nie na systematycznych, stabilnych postępach i tworzeniu konstruktywnych raportów. W wyniku tego tworzone harmonogramy będące na krawędzi możliwości realizacji, przez co poślizgi nie były zauważane, potwierdzane i przekazywane zarządowi aż do ostatniej chwili. Niewielki zespół programistów oraz jego bezpośredni przełożony blokowali całą firmę, ponieważ nie potrafili przyznać, że mają problem z dotrzymaniem terminów. Nacisk kładziony na brawurę zmusza do podejmowania olbrzymiego ryzyka i osłabia współpracę pomiędzy różnymi oddziałami pracującymi nad projektem.

Niektórzy kierownicy dążą do brawurowych zachowań, gdy polegają na zbyt pozytywnym nastawieniu. Jeżeli w wyniku tego nastawienia zespół przestanie tworzyć rzetelne, czasem negatywne raporty, kierownicy sami sobie uniemożliwiają korekcję kierunku, w jakim dąży cały projekt. Zasadniczo nie wiedzą nawet, że muszą podjąć jakieś kroki, dopóki szkoda nie zostanie

już wyrządzona. Tom DeMarco stwierdził, że nazbyt pozytywne nastawienie może przekształcić małe utrudnienia w prawdziwą katastrofę (DeMarco 1995).

Odnosiniki: Alternatywne sposoby ratowania przedłużającego się projektu znajdziesz w rozdziale 16. „Ratowanie projektu”.

5. Dołączanie ludzi do przedłużającego się projektu. To jest prawdopodobnie najbardziej klasyczny z klasycznych błędów. Gdy grozi poślizg, włączenie nowych ludzi do projektu może spowodować zredukowanie produktywności dotychczas zaangażowanych pracowników zamiast jej zwiększenia dzięki obecności „świeżej krwi”. Fred Brooks porównał ten błąd do gaszenia pożaru za pomocą benzyny (Brooks 1975).

Odnosiniki: Więcej informacji na temat wpływu otoczenia na produktywność znajdziesz w rozdziale 30. „Środowisko pracy”.

6. Hałaśliwe, zatłoczone biuro. Większość pracowników ocenia swoje warunki pracy jako niezadowolające. W ok. 60% przypadków są one niewystarczająco ciche lub nie zapewniają prywatności (DeMarco i Lister 1987). Pracownicy przebywający w cichych, prywatnych pomieszczeniach są znacznie bardziej produktywni od kolegów po fachu zajmujących głośne, zatłoczone boksy lub wspólne pomieszczenia.

Odnosiniki: Więcej informacji na temat dobrych relacji z klientami znajdziesz w rozdziale 10. „Projektowanie zorientowane na klienta”.

7. Napięcie pomiędzy programistami a klientami. Napięcie występujące pomiędzy programistami a klientami może się pojawić z kilku powodów. Klienci mogą uznać, że programiści nie chcą współpracować, gdyż nie zamierzają się zgodzić na proponowany harmonogram lub nie są w stanie spełnić danych obietnic. Z kolei programiści mogą stwierdzić, że klienci nierozsądnie żądają zmieszczenia się w nierealnym terminie lub żądają wprowadzenia zmian już po ustaleniu wymagań projektu. Czasami mogą się pojawiać tarcia następujące w wyniku różnic charakteru.

Podstawowym skutkiem tego napięcia jest słaba komunikacja, a w wyniku słabej komunikacji zespół nie rozumie wymogów, projektuje niewłaściwy interfejs użytkownika, w najgorszym przypadku klienci odrzucają produkt. Często napięcie pomiędzy programistami a klientami sięga takiego poziomu, że obydwie strony są gotowe zrezygnować z projektu (Jones 1994). Zniwelowanie napięcia zajmuje bardzo dużo czasu i nie pozwala się skupić na właściwej pracy.

Odnosiniki: Więcej informacji na temat definiowania oczekiwań znajdziesz w podrozdziale 10.3. „Zarządzanie oczekiwaniami klienta”.

8. Wygórowane oczekiwania. Jednym z podstawowych źródeł napięcia powstającego pomiędzy programistami a klientami są wygórowane oczekiwania tych drugich. W przykładzie 3.1 Bill nie miał żadnych podstaw, aby oczekiwać stworzenia produktu w ciągu sześciu miesięcy, ale kierownictwo firmy zażądało takiego terminu. To, że Mike nie był w stanie przekonać zarządu do bardziej realnego harmonogramu, stanowi główne źródło wszystkich pojawiających się później kłopotów.

W innych przypadkach kierownicy projektów lub programiści sami proszą się o kłopoty, otrzymując dotację na podstawie nazbyt optymistycznego harmonogramu. Czasami obiecują niemożliwe do zrealizowania zestawy funkcji.

Chociaż wygórowane założenia same w sobie nie wydłużają procesu twórczego, mogą sprawiać wrażenie, że harmonogram jest zbyt luźny, a to niemal równie szkodliwe zjawisko. Organi-

zacja Standish Group opublikowała listę pięciu najważniejszych czynników mających wpływ na sukces projektów biznesowych, wśród których znalazły się realistyczne oczekiwania (Standish Group 1994).

9. Brak sponsora z prawdziwego zdarzenia. Obecność bogatego sponsora jest niezbędna do rozwoju wielu aspektów szybkiego projektowania, takich jak realistyczne planowanie, kontrola zmian, a także wprowadzanie nowych rozwiązań projektowych. Bez wpływowego patronatu inni członkowie zarządu mogą wymusić niemożliwy do realizacji harmonogram lub skłonić zespół do wprowadzenia zmian psujących cały projekt. Pochodzący z Australii konsultant Rob Thomsett twierdzi, że brak patrona na poziomie zarządu grozi porażką projektu (Thomsett 1995).

10. Brak zaangażowania ze strony wszystkich zainteresowanych. Wszystkie najważniejsze osoby biorące udział w tworzeniu projektu muszą czuć wiarę w sens jego realizacji. Dotyczy to sponsora, lidera zespołu, członków zespołu, personelu z działu promocji, użytkowników ostatecznych, klientów oraz każdego, który ma w danym projekcie jakiś interes. Bliska współpraca pojawiająca się w przypadku zaangażowania wszystkich zainteresowanych stron umożliwia precyzyjną koordynację strategii szybkiego projektowania.

11. Brak wpływu użytkowników. Zgodnie z badaniem przeprowadzonym przez organizację Standish Group głównym powodem sukcesu systemów informatycznych jest wkład użytkowników w powstawanie tych produktów (Standish Group 1994). Projekty, w których opinie użytkowników nie są brane pod uwagę na wczesnych etapach, grożą niewłaściwym zrozumieniem wymogów projektowych oraz są narażone na przerost funkcjonalności w dalszych etapach.

Odniesienia: Więcej informacji na temat prawidłowej polityki znajdziesz w podrozdziale 10.3. „Zarządzanie oczekiwaniami klienta”.

12. Przedkładanie polityki nad treść. Larry Constantine opisał cztery zespoły mające cztery różne orientacje polityczne (Constantine 1995a). „Politycy” skupiali się na „zarządzaniu” — utrzymywali dobre stosunki z kierownikami. „Badacze” zajmowali się wyszukiwaniem i gromadzeniem informacji. „Izolacjoniści” byli bardzo skryci i tworzyli granice projektu, poza które nie wpuszczali nikogo spoza zespołu. „Generaliści” robili wszystko po trochu: utrzymywali kontakty z zarządem, przeprowadzali badania i zbierali dane, a także koordynowali swoje działania z pozostałymi zespołami. Zgodnie z wynikami badania początkowo najbardziej doceniane przez zarząd były zespoły „polityków” i „generalistów”, ale po półtora roku ten pierwszy został najszybciej rozwiązany. Przedkładanie polityki ponad wyniki jest zabójcze dla projektowania zorientowanego na szybkość.

13. Pobożne życzenia. Zawsze mnie zdumiewało, jak wiele problemów w dziedzinie inżynierii oprogramowania wynika z pobożnych życzeń. Ile razy słyszałaś/słyszałeś podobne stwierdzenia od różnych osób:

„Żaden członek zespołu nie wierzył w stworzenie projektu w wyznaczonym terminie, ale uznali, że jeśli każdy będzie ciężko pracował i wszystko pójdzie zgodnie z planem, a do tego jeśli jeszcze uśmiechnie się do nich los, to może zdołają zdążyć”.

„Nie przyłożyliśmy się do skoordynowania interfejsów pomiędzy różnymi modułami aplikacji, ale dobrze się dogadywaliśmy w innych aspektach produktu, a kwestia interfejsów jest w miarę nieskomplikowana, zatem w kilka dni usuniemy wszystkie błędy”.

„Wiemy, że wynajęliśmy najtańszego podwykonawcę do stworzenia podsystemu bazodanowego i aż w oczy kłuł brak doświadczenia poszczególnych pracowników. Nie mieli tak wielkiej wprawy, jak inni podwykonawcy, ale może brak doświadczenia nadrobią entuzjazmem. Prawdopodobnie zdążą z podsystemem na czas”.

„Nie musimy pokazywać klientowi najnowszych zmian wprowadzonych do prototypu. Jestem pewien, że już wiem, czego klient od nas oczekuje”.

„Zespół twierdzi, że zmieszczenie się w terminie będzie wymagało nieprawdopodobnego wprost wysiłku i już zaliczył kilkudniowy poślizg z niektórymi celami, ale myślę, że chłopaki jeszcze zdążą”.

Pobożne życzenia to nie tylko optymizm. To zasłanianie oczu i liczenie na łut szczęścia w momencie gdy nie powinno się opierać na przypadku. Wprowadzone na początku projektu pobożne życzenia wywołują na jego końcu olbrzymi chaos. Dewastują skuteczne planowanie i być może stanowią źródło większej liczby problemów niż wszystkie pozostałe pomyłki razem wzięte.

Proces

Błędy związane z procesem spowalniają projektowanie, ponieważ powodują marnowanie wysiłków i talentu pracowników. Poniżej przedstawiam najgorsze błędy dotyczące procesu.

Odnosiniki: Więcej informacji na temat nadmiernie optymistycznych harmonogramów znajdziesz w podrozdziale 9.1. „Tworzenie zbyt optymistycznych harmonogramów”.

14. Nadmiernie optymistyczne harmonogramy. Wyzwania stojące przed zespołem tworzącym aplikację w ciągu trzech miesięcy są zupełnie inne od wyzwań dotyczących projektu trwającego cały rok. Przygotowanie zbyt optymistycznego harmonogramu skazuje projekt na porażkę z powodu niedocenienia jego rozmiarów, niewłaściwego planowania, i skrócenia czynności początkowych, takich jak analiza wymagań i przygotowanie szkieletu aplikacji. Ponadto taki harmonogram wywiera olbrzymią presję na programistów, co powoduje długoterminowy uszczerbek na morale i produktywności pracownika. Było to podstawowe źródło problemów w przykładzie 3.1.

Odnosiniki: Więcej informacji na temat zarządzania ryzykiem znajdziesz w rozdziale 5. „Zarządzanie ryzykiem”.

15. Niewystarczające zarządzanie ryzykiem. Pewne błędy nie występują na tyle często, żeby można je było uznać za klasyczne. Noszą one nazwę „ryzyko” lub „zagrożenie”. Podobnie jak w przypadku z klasycznymi błędami, jeżeli nie będziesz aktywnie zarządzać ryzykiem, wystarczy, że tylko jedna rzecz pójdzie nie tak, żeby przekształcić szybkie projektowanie w powolne. Nieumiejętność zarządzania ryzykiem jest klasycznym błędem.

Odnosiniki: Więcej informacji na temat podwykonawców znajdziesz w rozdziale 28. „Zewnętrzni podwykonawcy”.

16. Niewywiązanie się podwykonawcy ze zlecenia. Jeżeli firma bardzo się śpieszy z jakimś projektem, czasami zleca jego część zewnętrznemu podwykonawcy. Podwykonawcy często jednak spóźniają się z dostarczeniem swojego produktu, charakteryzuje się on niską jakością lub nie zawiera wymaganych funkcji (Boehm 1989). Takie ryzyko jak niska wydajność lub nieprawidłowy interfejs zawsze staje się bardziej prawdopodobne, gdy jest wynajmowany zewnętrzny podwykonawca. Przy nieodpowiedniej komunikacji taki podwykonawca zamiast przyspieszyć projekt, może go tylko spowolnić.

Odnosiniki: Więcej informacji na temat planowania znajdziesz w punkcie „Planowanie” (podrozdział 4.1).

17. Niedokładne planowanie. Jeżeli nie zaplanujesz dobrze strategii szybkiego projektowania, nigdy jej nie wdrożysz.

Odnosiniki: Więcej informacji na temat planowania pod presją znajdziesz w podrozdziale 9.2. „Harmonogram pod presją” oraz w rozdziale 16. „Ratowanie projektu”.

18. Zaprzestanie planowania pod wpływem presji. Zespoły projektowe często sporządzają plany, a następnie je porzucają, gdy tylko rozpoczną się problemy z harmonogramem (Humphrey 1989). Problemem nie jest samo porzucenie planu, ale niezdolność do stworzenia jego zamiennika, co powoduje przejście w tryb nieprzemyślanego pisania kodu. W przykładzie 3.1 zespół porzucił pierwotny plan tuż po pierwszym poślizgu — jest to typowe zachowanie. Od tego momentu praca stała się niezorganizowana i uciążliwa do tego stopnia, że Jill zaczęła pracować dla swojego poprzedniego zespołu i nikt tego nie zauważył.

19. Czas zmarnowany na przygotowania. „Przygotowaniami” nazywam czas jeszcze przed rozpoczęciem projektu, który standardowo jest przeznaczony na zatwierdzenie i zbieranie budżetu. Dość powszechną praktyką jest spędzanie miesięcy, a nawet lat na przygotowaniach, po czym zaczyna się wyścig z napiętym harmonogramem. O wiele łatwiej, taniej i bezpieczniej jest dobrze wykorzystać te kilka tygodni lub miesięcy, niż zostawić sobie czynności przygotowawcze na później przy harmonogramie trwającym równie długo co faza przedwstępna.

Odnosiniki: Więcej informacji na temat bagatelizowania początkowych etapów znajdziesz w podrozdziale 9.1. „Tworzenie zbyt optymistycznych harmonogramów”.

20. Zbagatelizowanie początkowych etapów projektowania. Wykonywane w pośpiechu projekty zazwyczaj mają usuwane pozornie najmniej istotne elementy projektowe, a takimi łatwymi celami są analiza, tworzenie architektury i układu projektu, ponieważ w tym czasie nie jest bezpośrednio pisany kod. Wziąłem się kiedyś za ratowanie katastrofalnego projektu; gdy poprosiłem o pokazanie schematów, odpowiedziano mi, że „nie mieli czasu na ich przygotowanie”.



Skutki tego błędu — zwanego również „niezaplanowanym programowaniem” — są bardzo przewidywalne. W przykładzie 3.1 wysokiej jakości projekt modułu został zastąpiony oszustwem programistycznym (umieszczenie na jednej stronie wykresu i treści raportu). Przed wypuszczeniem produktu na rynek, oszukany kod tak czy siak musiał zostać usunięty i należało poświęcić czas na opracowanie właściwego rozwiązania. Projekty, w których pracownicy starają się zaoszczędzić w początkowych etapach, zazwyczaj generują na późniejszych etapach 10 – 100 razy większe koszty w porównaniu do podobnego projektu, w którym należycie przeanalizowano poszczególne moduły i sporządzono odpowiednie schematy (Fagan 1976; Boehm i Papaccio 1988). Jeżeli nie potrafisz na początku projektu poświęcić pięciu godzin na wykonanie niezbędnej pracy, skąd weźmiesz 50 godzin na poprawianie w dalszych etapach?

21. Nieodpowiednie przygotowanie schematów. Szczególnym przypadkiem zbagatelizowania początkowych etapów jest nieodpowiednie sporządzenie schematów. W projektach tworzonych pod presją czasu często nie poświęca się wystarczającej uwagi na dobór odpowiedniego środowiska, w którym można w szybki sposób przygotować alternatywne, przemyślane schematy tworzenia aplikacji. Często podczas sporządzania schematu kładziony jest nacisk na doraźne potrzeby, a nie na jakość, dlatego zazwyczaj kończy się to na konieczności przeprowadzenia kilku czasochłonnych cykli projektowych, zanim cały system zostanie całkowicie ukończony.

Odnosiniki: Więcej informacji na temat kontroli jakości znajdziesz w podrozdziale 4.3. „Podstawy kontroli jakości”.



22. Zbogatelizowanie kontroli jakości. Wykonywane w pośpiechu projekty niejednokrotnie cierpią od spływania takich etapów, jak analizowanie konstrukcji i kodu aplikacji, omijania planów testowania oraz przeprowadzania jedynie powierzchownych testów. W przykładzie 3.1 marginalnie potraktowano analizę konstrukcji i kodu oprogramowania w celu zmieszczenia się w wyznaczonym harmonogramie. Okazało się, że gotowy produkt ma zbyt wiele błędów, żeby można go było wydać, a poprawianie zajęło kolejne pięć miesięcy. To jest bardzo typowy rezultat. Pominięcie jednego dnia poświęconego na testowanie podczas początkowych etapów projektowania w późniejszym terminie może kosztować 3 – 10 dni dodatkowej pracy (Jones 1994). Taki „skrót” zdecydowanie przekreśla strategię szybkiego projektowania.

Odnosiniki: Więcej informacji na temat mechanizmów zarządzania znajdziesz w punkcie „Śledzenie” (podrozdział 4.1) oraz w rozdziale 27. „Rozbijanie celów na podetapy”.

23. Niewystarczające mechanizmy zarządzania. W przykładzie 3.1 wykorzystano zbyt mało mechanizmów zarządzania do ostrzegania w odpowiednim momencie o nieubłaganych poślizgach, a te, które były od początku stosowane, zostały porzucone, gdy pojawiły się problemy. Zanim stwierdzisz, że projekt zmierza we właściwym kierunku, musisz najpierw określić, czy w ogóle znajduje się na torach.

Odnosiniki: Więcej informacji na temat przedwczesnej konwergencji znajdziesz w ustępie „Przedwczesna konwergencja (zbieżność)” (podrozdział 9.1).

24. Przedwczesna lub zbyt często występująca konwergencja. Zanim zostaną ustalone terminy wydania gotowego produktu, pojawiają się naciski, aby go odpowiednio przygotować — poprawić wydajność, wydrukować dokumentację, wdrożyć system pomocy, dopracować instalator, usunąć funkcje, które nie zostaną przygotowane na czas itd. W przypadku projektów tworzonych w pośpiechu istnieje tendencja do przedwczesnego łączenia tych elementów. Niemożliwa jest konwergencja produktu w dowolnym momencie, dlatego wiele zespołów stara się ją wymusić przynajmniej kilkakrotnie, zanim odniosą sukces. Te dodatkowe iteracje konwergencji nie wpływają dobrze na produkt. Zabierają jedynie cenny czas i niepotrzebnie wydłużają harmonogram.

Odnosiniki: Listę powszechnie omijanych zadań znajdziesz w ustępie „Nie pomijaj najpowszechniejszych zadań” (podrozdział 8.3).

25. Pomijanie podstawowych zadań. Jeżeli ludzie zignorują sporządzanie dokładnych raportów z poprzednich projektów, zapomną o mniej oczywistych zadaniach, których lista będzie się wydłużać. W wyniku pomijania zadań harmonogram zostaje wydłużony o jakieś 20 – 30% (van Genuchten 1991).

Odnosiniki: Więcej informacji na temat ponownego szacowania znajdziesz w punkcie „Przekalibrowanie” (podrozdział 8.7).

26. Planowanie nadrobienia zaległości. Jednym z rodzajów ponownego szacowania jest niewłaściwe reagowanie na poślizg. Jeżeli pracujesz nad projektem trwającym pół roku, a spełnienie celu wyznaczonego na dwa miesiące zajmuje Ci trzy miesiące, to co robisz? Wiele planów zakłada nadrobienie zaległości w późniejszym terminie, ale tak się nigdy nie dzieje. Wraz z tworzeniem projektu dowiadujesz się o nim coraz więcej, w tym również coraz dokładniej szacujesz czas potrzebny na jego ukończenie. Ta wiedza powinna zostać wykorzystana do ponownego oszacowania harmonogramu.

Inny rodzaj pomyłki związany z ponownym szacowaniem wynika ze zmian produktu. Jeżeli parametry tworzonego przez Ciebie produktu są modyfikowane, wydłuża się również czas potrzebny na wprowadzenie tych zmian. W przykładzie 3.1 główne założenia projektu zmieniły się jeszcze przed rozpoczęciem prac, co nie zostało odzwierciedlone w harmonogramie ani zasobach. Dodawanie nowych funkcji bez zmiany terminu ukończenia projektu musi się skończyć poślizgiem.

Odnosiniki: Więcej informacji na temat strategii nieprzemysłanego pisania kodu znajdziesz w podrozdziale 7.2. „Nieprzemysłane pisanie kodu”.

27. Strategia ekstremalnego programowania. Niektóre firmy uważają, że drogą do szybkiego projektowania jest błyskawiczne, swobodne i niepilnowane pisanie kodu aplikacji. Jeżeli programiści są odpowiednio zmotywowani i doświadczeni, pokonają każdą przeszkodę. Jak się przekonasz podczas dalszej lektury, nie ma bardziej mylnego założenia. Strategia taka jest często przedstawiana jako „świadcząca o przedsiębiorczości”, ale w rzeczywistości stanowi tylko fasadę starego paradygmatu nieprzemysłanego tworzenia kodu połączonego z ambitnym harmonogramem — a taka kombinacja nie sprawdza się prawie nigdy. Jest to doskonały przykład na ukazanie, że dwa złe rozwiązania wcale nie przynoszą dobrego efektu.

Produkt

Poniżej przedstawiam klasyczne błędy związane z definiowaniem produktu.

28. Nadmiar wymagań. Niektóre projektu od samego początku obarczone zostają zbyt wysokimi wymaganiami. Zbyt często wydajność zostaje wymieniona jako niezbędny wymóg, co zwykle powoduje niepotrzebne wydłużenie harmonogramu. Nieraz użytkownicy są mniej zainteresowani zaawansowanymi funkcjami niż twórcy czy dział promocji, a taka złożona funkcjonalność nieproporcjonalnie spowalnia proces tworzenia aplikacji.

Odnosiniki: Więcej informacji o przeroście funkcjonalności znajdziesz w rozdziale 14. „Regulowanie zestawu funkcji”.

29. Przerost funkcjonalności. Nawet jeżeli uda Ci się uniknąć nadmiaru wymagań, są one modyfikowane w czasie trwania przeciętnego projektu średnio o 25% (Jones 1994). Taka zmiana może spowodować wydłużenie harmonogramu również o 25%, co stanowi katastrofę w przypadku projektów zorientowanych na szybkie tworzenie.

Odnosiniki: W podpunkcie „Niejasne lub niemożliwe do spełnienia cele” (podrozdział 14.2) znajdziesz przykład opisujący zupełnie przypadkowe wystąpienie samowoli programisty w trakcie trwania projektu.

30. Samowola programistów. Programiści fascynują się nowymi technologiami i czasami dążą do tego, aby wypróbować nowe funkcje języka lub środowiska programistycznego albo stworzyć własną wersję świetnej funkcji podpatrzonych u konkurencji — chociaż w danym projekcie jest ona zupełnie niepotrzebna. Wysiłek włożony w przygotowanie, wdrożenie, przetestowanie, stworzenie dokumentacji oraz obsługę niepotrzebnych funkcji powoduje tylko zwiększenie ryzyka poślizgu.

31. „Przepychanie” negocjacje. Jedną z dziwniejszych taktyk negocjacyjnych jest odwlekanie terminu przez kierownika zbyt wolno tworzonego projektu po to tylko, aby po wprowadzeniu nowego terminu dorzucić kolejne zadania. Trudno zrozumieć takie postępowanie, ponieważ kierownik zatwierdzający nowy harmonogram pośrednio przyznaje, że pierwotny termin był błędny. Jednakże ta sama osoba po zatwierdzeniu nowego harmonogramu podejmuje jawne

działania dążące do kolejnego poślizgu. Takie działanie w żaden sposób nie pomaga w dotrzymaniu harmonogramu.

32. Projektowanie zorientowane na badania. Seymour Cray, twórca superkomputerów marki Cray, stwierdził, że nie zamierzał skupiać się na więcej niż na jednoczesnym rozwoju dwóch obszarów inżynierii, ponieważ dalsze zwiększenie ich liczby niebezpiecznie przybliży prawdopodobieństwo porażki (Gilb 1988). Wielu programistów powinno czerpać naukę od pana Craya. Jeżeli w swoim projekcie musisz się zająć rozwojem informatyki poprzez stworzenie nowego algorytmu lub nowych rozwiązań obliczeniowych, nie projektujesz aplikacji, tylko skupiasz się na badaniach. Harmonogram inżynierii oprogramowania jest przewidywalny w dość dużym stopniu, natomiast harmonogramu badań nie da się w najmniejszym stopniu ustalić.

Jeżeli wśród celów projektu znajduje się rozwój nowych technologii — algorytmów, wydajności, wykorzystania pamięci itd. — musisz założyć, że harmonogram będzie bardzo niestabilny. W przypadku gdy zajmujesz się rozwojem technologii, a Twój projekt zawiera inne słabe punkty — zbyt mały personel, niewykwalifikowany personel, niesprecyzowane wymagania, niestabilny interfejs tworzony przez zewnętrznego podwykonawcę — możesz od razu wyrzucić założony harmonogram. Jeśli musisz za wszelką cenę rozwinąć się technologicznie, zrób to, ale nie oczekuj, że szybko zakończysz projekt!

Technologia

Pozostałe błędy mają związek z nieużywaniem lub nieprawidłowym stosowaniem współczesnej technologii.

Odnośniki: Więcej informacji na temat syndromu „srebrnej kuli” znajdziesz w podrozdziale 15.5. „Syndrom »srebrnej kuli«”.

33. Syndrom „srebrnej kuli”. W przykładzie 3.1 za bardzo zaufano potencjalnym zaletom reklamowanych, lecz nieużywanych wcześniej technologii (generator raportów, programowanie obiektowe, język C++) i nie przeanalizowano ich działania w tym konkretnym środowisku projektowym. Jeśli zespoły projektowe zdobędą pojedyncze nowe rozwiązanie, nową technologię lub rygorystyczny proces i oczekują od nich lekarstwa na wszystkie problemy z harmonogramem, to srodze się zawiodą (Jones 1994).

Odnośniki: Więcej informacji dotyczących oszczędzania na narzędziach zwiększających produktywność znajdziesz w punkcie „Określanie stopnia redukcji harmonogramu” (podrozdział 15.4).

34. Przeszacowanie oszczędności wynikających ze stosowania nowych rozwiązań lub narzędzi. Firmy sporadycznie zwiększają produktywność w znaczącym stopniu, bez względu na ilość bądź jakość wdrażanych nowych rozwiązań i narzędzi. Zalety korzystania z nowych rozwiązań są częściowo równoważone przez czas potrzebny na nauczenie się ich, a doskonale opanowanie danej techniki wymaga wiele czasu. Z nowymi rozwiązaniami wiążą się również nowe zagrożenia, które ujawniają się dopiero podczas korzystania z danej metody. Spodziewaj się raczej powolnego, stabilnego zwiększania produktywności rzędu kilku procent na dany projekt, a nie olbrzymiego skoku. Zespół omawiany w przykładzie 3.1 powinien założyć zwiększenie produktywności najwyżej o 10% wynikające z nowej technologii, a nie jej podwojenie.

Odnośniki: Więcej informacji na temat wielokrotnego wykorzystywania zasobów znajdziesz w rozdziale 33. „Wielokrotne wykorzystywanie zasobów”.

Specjalnym przypadkiem nadmiernych oszczędności jest ponowne wykorzystywanie kodu stworzonego w poprzednich projektach. Takie rozwiązanie może być bardzo skuteczne, ale oszczędność czasu rzadko bywa równie spektakularna.

35. Zmiana narzędzi w środku projektu. Jest to stare rozwiązanie ratunkowe, które bardzo rzadko daje zamierzony efekt. Czasem warto zaktualizować wykorzystywane oprogramowanie, np. z wersji 3.0 do 3.1, a nawet 4.0, jednak czas poświęcony na naukę, potrzebny na zmodyfikowanie dotychczasowego produktu, oraz nowe rodzaje pomyłek nieodmiennie towarzyszące kolejnej wersji narzędzia zazwyczaj niwelują wszelkie potencjalne oszczędności czasu w trakcie trwania projektu.

Odnosiniki: Więcej informacji na temat kontrolowania kodu źródłowego znajdziesz w punkcie „Zarządzanie konfiguracją oprogramowania” (podrozdział 4.2).

36. Brak zautomatyzowanej kontroli kodu źródłowego. Pominięcie zautomatyzowanej kontroli kodu źródłowego naraża projekt na niepotrzebne ryzyko. Bez tej funkcji dwóch programistów pracujących nad tym samym modulem musi koordynować ręcznie swoje wysiłki. Mogą dojść do porozumienia i porównywać najnowsze wersje swoich plików przed umieszczeniem ich w katalogu głównym, ale zawsze może się zdarzyć, że ktoś nadpisze wyniki pracy drugiej osoby. Ludzie tworzą kod do przestarzałych interfejsów, a później muszą zmodyfikować całość, gdyż korzystali z niewłaściwej wersji interfejsu. Użytkownicy zgłaszają wady, których nie można przeanalizować, ponieważ nie da się skompilować używanej przez nich wersji aplikacji. Przeciętnie kod źródłowy jest modyfikowany o ok. 10% w ciągu miesiąca, a jego ręczna kontrola po prostu nie nadąza za zmianami (Jones 1994).

W tabeli 3.1 zawarłem pełną listę klasycznych błędów.

3.4. Ucieczka z „wyspy Gilligana”

Lista wszystkich klasycznych błędów zajęłaby znacznie więcej stron, ale wymieniłem te najpowszechniej występujące oraz najpoważniejsze. Profesor David Umphress z Seattle University zauważa, że przyglądanie się, jak różne firmy starają się unikać klasycznych błędów, przypomina oglądanie powtórek *Wyspy Gilligana*². Na początku każdego odcinka Gilligan, Skipper lub Profesor opracowują nowy niedorzeczny plan ucieczki z wyspy. Najpierw wszystko zdaje się zmierzać we właściwym kierunku, ale wraz z rozwojem fabuły cała koncepcja zaczyna się rozsypywać i na końcu odcinka bohaterowie trafiają do punktu wyjścia — na wyspę.

Podobnie jest z większością firm: na końcu projektu odkrywają, że został popełniony kolejny klasyczny błąd, po którym nastąpił poślizg w terminie wydania produktu, został przekroczony budżet albo jednocześnie wystąpiły obydwa skutki.

² Amerykański serial komediowy emitowany w latach 1964 – 1967 — *przyp. tłum.*

Tabela 3.1. Lista klasycznych błędów

Błędy związane z czynnikiem ludzkim	Błędy związane z procesem	Błędy związane z produktem	Błędy związane z technologią
Podkopana motywacja	Nadmiernie optymistyczne harmonogramy	Nadmiar wymagań	Syndrom „srebrnej kuli”
Słaby personel	Niewystarczające zarządzanie ryzykiem	Przerost funkcjonalności	Przeszacowanie oszczędności wynikających ze stosowania nowych rozwiązań lub narzędzi
Niekontrolowani, problematyczni pracownicy	Niewywiązanie się podwykonawcy ze zlecenia	Samowola programistów	Zmiana narzędzi w środku projektu
Brawura	Niedokładne planowanie	„Przepychane” negocjacje	Brak zautomatyzowane j kontroli kodu źródłowego
Dołączanie ludzi do przedłużającego się projektu	Zaprzestanie planowania pod wpływem presji	Projektowanie zorientowane na badania	
Hałaśliwe, zatłoczone biuro	Czas zmarnowany na przygotowaniach		
Napięcie pomiędzy programistami a klientami	Zbagatelizowanie początkowych etapów projektowania		
Wygórowane oczekiwania	Nieodpowiednie przygotowanie schematów		
Brak sponsora z prawdziwego zdarzenia	Zbagatelizowanie kontroli jakości		
Brak zaangażowania ze strony wszystkich zainteresowanych	Niewystarczające mechanizmy zarządzania		
Brak wpływu użytkowników	Przedwczesna lub zbyt często występująca konwergencja		
Przedkładanie polityki nad treść	Pomijanie podstawowych zadań		
Pobożne życzenia	Planowanie nadrabiania zaległości		
	Strategia ekstremalnego programowania		

Twoja własna lista najgorszych rozwiązań

Nigdy nie zapominaj o klasycznych błędach. Stwórz listę „najgorszych rozwiązań”, których będziesz unikać w kolejnych projektach. Rozpocznij od listy zawartej w tabeli 3.1. Dodawaj do niej kolejne błędy, czytając sprawozdania z ukończenia projektu, aby poznać pomyłki popełnione przez zespół. Przekonaj zarząd, aby wprowadził sporządzanie sprawozdań również w innych projektach, dzięki czemu będziesz w stanie poznać błędy popełniane przez inne zespoły. Wymieniaj się również opowieściami i doświadczeniami z pracownikami innych firm. Pokazuj sporządzoną przez siebie listę błędów jak największej grupie osób, gdyż ktoś może wyciągnie wnioski i nie popełni po raz kolejny klasycznego błędu.

Literatura uzupełniająca

Kilka książek zajmuje się zagadnieniem pomyłek programistycznym, ale nie spotkałem się z tytułem traktującym o klasycznych błędach w dziedzinie sporządzania harmonogramów projektu. W dalszej części tej publikacji poruszam szczegółowo pewne kwestie dotyczące klasycznych błędów.

Skorowidz

A

alternatywna strategia szybkiego projektowania, 48
analiza
 ryzyka, 112
 wymagań, 251, 252
 zmian, 340, 342
aplikacje niestandardowe, 351
architektura zorientowana na zmianę, 415
autonomia, 267, 293

B

bagatelizowanie projektu, 226
błędy, 55
 związane z czynnikiem ludzkim, 74
 związane z procesem, 74
 związane z produktem, 74
 związane z technologią, 74
budowanie zespołu, 298, 300
bufory, 114

C

całkowity poślizg projektu, 114
codzienne kompilacje
 skutki uboczne, 412
 stosowanie, 407
 zarządzanie ryzykiem, 412
cykl życia, 151, 461
 model spiralny, 541
cykle wydawnicze, 341
czynnik ludzki, 38, 65, 235
czynniki
 higieniczne, 273
 motywujące, 264, 270
 niszczące morale, 273

D

definiowanie rodzin aplikacji, 419
dobór
 cyklu życia, 152
 mierzonych parametrów, 466

dobrowolna praca w nadgodzinach, 593–602
dokładność
 harmonogramu, 224
 oszacowań, 201
dostarczanie produktu, 425, 543

E

ekspozycja, 112
elementy zarządzania ryzykiem, 105
eliminowanie ryzyka, 117
etap
 planowania JAD, 450
 projektowania JAD, 454
 przygotowań, 143
 ustalania wymogów, 141
ewolucyjne dostarczanie produktu, 167, 425, 429
 skuteczne stosowanie, 432
 skutki uboczne, 430
 stosowanie, 427
 zarządzanie ryzykiem, 421, 429

F

funkcje, 323

G

grupa pomiarowa, 465
grupowanie, 342

H

harmonogramy, 62, 81, 200
 lista zagrożeń, 108
 najkrótsze, 202
 negocjacje, 242
 nominalne, 208
 optymistyczne, 220
 pod presją, 233
 redukcja, 360
 schemat usprawniania, 146
 sporządzanie, 219
 wydajne, 206

I

identyfikacja ryzyka, 107
 inspekcje, 445
 instrukcje, 101
 interfejs użytkownika, 583
 istotność zadania, 267

J

JAD, 447
 etap planowania, 450
 etap projektowania, 454
 sesja, 450
 skuteczne stosowanie, 459
 skutki uboczne, 457
 stosowanie metodologii, 448
 zarządzanie ryzykiem, 456
 jakość planowania projektu, 225, 229
 a kompromisy, 146
 języki RDL
 skutki uboczne, 515
 zarządzanie ryzykiem, 513
 języki szybkiego projektowania, 509

K

kierownik, 18, 317, 560
 klient, 248
 kolejność wydawania, 428
 komisja zatwierdzająca zmiany, 403
 kompilacje, 405, 407, 411, 413
 kompromisy, 144, 405
 komunikacja, 292
 koncentracja na projekcie, 226
 konfiguracja oprogramowania, 90
 konstrukcja, 89, 92
 kontrakty, 491
 kontrola, 185
 jakości, 93
 przerostu funkcjonalności, 334
 ryzyka, 106, 117
 zmian, 340
 konwergencja, 227
 kreatywność, 230
 kryteria doboru narzędzi, 357
 kwantyfikacja ryzyka, 194
 kwestie dotyczące kontraktów, 493

L

liderzy techniczni, 18, 317
 lista
 klasycznych błędów, 63, 74
 zagrożeń, 581
 loteria, 122

M

metoda okienek czasowych, 569
 kryteria dopuszczalności, 572
 stosowanie projektowania, 571
 zarządzanie ryzykiem, 574
 metodologia JAD, 447, 459
 metody kontroli zmian, 340
 metodyka SCM, 93
 mieszanie ról, 290
 minimalna specyfikacja, 327
 mnożniki punktów funkcyjnych, 191
 model cyklu życia, 172
 kaskadowy, 153
 z obniżeniem ryzyka, 162
 zawierający podprojekty, 161
 zmodyfikowany, 160
 sashimi, 160
 spiralny, 158, 541
 modele zespołów, 308
 moduły wrażliwe na błędy, 96
 monitorowanie ryzyka, 120
 morale, 263, 273
 motyw przewodni produktu, 328
 motywacja, 229, 263
 możliwość rozwoju, 266

N

nadgodziny, 593–602
 nadzór techniczny, 269
 nagrody, 270
 narzędzia
 CASE, 366
 zwiększające produktywność, 347, 352–359
 negocjacje, 235, 499
 dotyczące harmonogramu, 242

O

obiektywne kryteria, 239
 ocena

- podwykonawcy, 493
- pracowników, 273
- ryzyka, 106, 113

 oczekiwania klienta, 139, 254
 oddziaływania, 405
 określanie

- wymagań, 142
- wizji, 328

 oprogramowanie

- biznesowe, 47
- komercyjne, 169
- systemowe, 47
- technologii informacyjnej, 47
- wbudowane, 47

 orientacyjne szacowanie harmonogramu, 199
 outsourcing, 487

P

papierowe scenopisy, 327
 pisanie kodu, 157
 plan

- nabycia, 355
- ratowania projektu, 375
- zmian, 419

 planowane wykorzystywanie zasobów, 525
 planowanie, 81

- cyklu życia, 151
- optymalnego harmonogramu, 137
- zarządzania ryzykiem, 117

 pocisk, 368
 poczucie własności, 264
 podetapy, 477, 480, 483
 podmiotowość, 293
 podstawy techniczne, 85
 podwykonawcy

- ocena, 493
- zagraniczni, 492
- zarządzanie ryzykiem, 495
- zewnątrzni, 487

 pomiary, 83, 463, 469, 471

- ograniczenia, 471
- stosowanie, 464, 474
- zarządzanie ryzykiem, 471

 poślizg całkowity, 114
 poświęcenie, 198
 poziomy języków programowania, 512
 praca zespołowa, 281
 prawdopodobieństwo straty, 114
 priorytetyzacja ryzyka, 115
 proces, 39, 68
 produkt, 42, 71
 produktywne środowiska pracy, 503, 506
 produktywność, 347

- na osobę a kompromisy, 146
- zespołu, 284

 programowanie

- ekstremalne, 50
- obiektywne, 367
- zautomatyzowane, 367

 projektowanie, 77

- metodą okienek czasowych, 569
- obiektywne, 421
- ukierunkowane
 - na harmonogram, 46
 - na szybkość, 46
- zorientowane na klienta, 245

 projekty

- biznesowe, 47
- celofanowe, 47
- imitujące szybkie projektowanie, 132
- pilotażowe, 271
- systemowe, 47
- z silnymi ograniczeniami czasowymi, 132

 prototypowanie ewolucyjne, 163, 433, 440

- skutki uboczne, 440
- stosowanie, 434, 441
- zarządzanie ryzykiem, 435

 prototypowanie interfejsu użytkownika, 327, 583

- fasada, 586
- oddziaływanie, 590
- skutki uboczne, 589
- wybór języka, 586
- zarządzanie ryzykiem, 588

 prototypowanie z odrzuceniem, 351, 563

- skutki uboczne, 566
- stosowanie, 564, 567
- zarządzanie ryzykiem, 565

 przedwczesna konwergencja, 227
 przeglądy techniczne, 97
 przekalibrowanie, 212
 przerost funkcjonalności, 107, 142

przeróbki, 142
 przesiewanie wymagań, 332, 519
 przesłanie informacji, 417
 punkty funkcyjne, 191

R

ratowanie projektu, 371
 nieskuteczne, 372
 plan, 375
 skuteczne, 384
 sposoby, 373
 redukcja
 harmonogramu, 360
 zestawu funkcji, 325
 regulowanie zestawu funkcji, 323
 relacje z klientami, 226
 rodzaje
 kontraktów, 491
 oprogramowania, 130
 projektów, 47, 548
 szybkiego projektowania, 44, 131
 zespołów, 305
 rodziny aplikacji, 419
 rola
 Dusza zespołu, 291
 Finalizator, 291
 Koordynator, 290
 Lokomotywa, 290
 Obserwator, 291
 Pomysłodawca, 291
 Realizator, 291
 Zaradna dusza, 291
 rozbijanie celów, 477, 480, 483
 skutki uboczne, 483
 stosowanie metody, 480
 zarządzanie ryzykiem, 483
 rozkład czasu na czynności, 141
 rozmiar
 programu, 190
 systemu, 201
 rozwiązania, 29
 zorientowane na klienta, 250, 340
 ryzyko, 103
 analiza, 112
 eliminowanie, 117
 identyfikacja, 107

kontrola, 106, 117
 kwantyfikacja, 194
 monitorowanie, 120
 ocena, 106, 113
 poziomy zarządzania, 106
 priorytetyzacja, 115
 zarządzanie, 103, 123, 412, 421, 429
 zarządzanie systematyczne, 123

S

samodzielni programiści, 18
 satysfakcja z pracy, 263
 schemat, 87, 91, 253
 usprawniania harmonogramu, 146
 SCM, 93
 sesje JAD, 447
 specjalista ds. zarządzania ryzykiem, 122
 specyfikacja, 325
 minimalna, 329, 331
 skrócona na papierze, 327
 w postaci instrukcji obsługi, 327
 wyjściowa, 327
 wymagań, 252
 spiralny model cyklu życia, 541
 sporządzanie
 harmonogramu, 81, 108–112, 198, 219
 minimalnej specyfikacji, 327
 schematów, 253
 specyfikacji, 325
 sposoby ratowania projektu, 373
 spójność, 285
 sprawdzone rozwiązania, 389
 metody, 394
 porównanie, 400
 sprawozdania
 bazowe, 470
 okresowe, 122
 statystyka, 471
 stosowanie
 codziennych kompilacji, 407
 ewolucyjnego dostarczania produktu, 427, 432
 języków RDL, 513
 metodologii JAD, 448, 459
 narzędzi, 368
 obiektywnych kryteriów, 239
 pomiarów, 464, 473
 projektowania metodą okienek czasowych, 571

prototypowania ewolucyjnego, 434, 441
 prototypowania interfejsu użytkownika, 585
 prototypowania z odrzuceniem, 564
 sesji JAD, 457
 wieloetapowego dostarczania produktu, 546
 wielokrotnego wykorzystywania zasobów, 522
 wspólnego celu, 534
 zarządzania zgodnego z teorią W, 555
 strata, 113
 strategia szybkiego projektowania, 31, 34
 strefy planowania, 138
 struktura
 zespołu, 303, 305
 zorientowana na wyniki, 289
 style przedstawiania oszacowań, 194
 SWAT, 313
 syndrom
 „srebrnej kuli”, 364
 „zabójczej aplikacji”, 335
 synergia, 43
 systemy zarządzania bazą danych, 351
 szacowanie, 179, 181–187, 216
 bazujące na scenariuszach, 195
 harmonogramu, 81, 197
 oparte na czynnikach pewności, 196
 orientacyjne harmonogramu, 199
 pierwszego rzędu Jonesa, 198
 punktów funkcyjnych, 189
 rozmiaru, 189
 rozmiaru straty, 113
 wysiłku, 196
 zawężone, 210
 szybkie projektowanie, 17, 27, 127
 rozwiązania, 148

Ś

śledzenie, 82
 środowisko pracy, 501
 skuteczne wdrożenie, 508
 skutki uboczne, 506
 zarządzanie ryzykiem, 505
 zastosowania, 503

T

technologia, 43, 72
 RAD, 366

teoria W, 553
 rodzaje projektów, 559
 skutki uboczne, 561
 zarządzanie ryzykiem, 560
 termin
 ukończenia projektu, 136, 147
 wdrożenia, 359
 testowanie, 97
 testy dymowe, 405, 408, 411, 413
 tożsamość
 zadania, 267
 zespołu, 288
 trening, 360
 trójkąt kompromisów, 144
 tworzenie
 minimalnej specyfikacji, 331
 planu zmian, 419
 schematów, 87, 91
 struktury zespołu, 307
 wydajnego zespołu, 286

U

usprawnianie harmonogramu, 146
 usprawnienia, 183
 ustanawianie celu, 265, 443
 usuwanie funkcji, 343

W

wdrożenie
 produktywnego środowiska pracy, 508
 szybkiego projektowania, 28
 wersjonowanie produktu, 333
 wieloetapowe dostarczanie produktu, 164, 543
 oddziaływanie, 550
 skutki uboczne, 550
 stosowanie, 546
 zarządzanie ryzykiem, 549
 wielokrotne wykorzystywanie zasobów, 521, 531
 wizualizacja postępów, 478
 wrażenie powolnego projektowania, 140
 wspólny cel, 533
 oddziaływanie, 538
 skutki uboczne, 538
 w różnych środowiskach, 535
 zarządzanie ryzykiem, 536

współpraca, 186
wybór
 języka prototypowania, 586
 modelu cyklu życia, 170, 174, 461
 najszybszego cyklu życia, 170
 rozwiązań, 29
wycieńczenie, 230
wydajne projektowanie, 44
wydajność, 285
wykorzystywanie zasobów, 524
 planowane, 525
 zarządzanie ryzykiem, 529
wykres zbieżności oszacowań, 184
wymagania, 86, 91, 332, 519
wymiary szybkości projektowania, 36
wymuszenie zbieżności, 227
wysiłek, 200
Wyspa Gilligana, 73
wytwarzanie
 dopasowane do harmonogramu, 165
 dopasowane do narzędzi, 168

Z

zachęty, 270
zagraniczni podwykonawcy, 492
zagrożenia, 405, 581
 wpływające na harmonogram, 108
zarys procesu szacowania, 188
zarządzanie, 80
 konfiguracją oprogramowania, 90, 93
 oczekiwaniami klienta, 254
 ryzykiem, 103, 123, 412, 421, 429
 wymaganiami, 86, 91
 zespołem, 294
 zgodne z teorią W, 553
 zmianami, 344
zasoby, 521
zastosowania pracy zespołowej, 283
zatrzymywanie zmian, 338
zawężanie oszacowań, 210
zbiór rozwiązań, 29
zespół, 282
 biznesowy, 309
 brak tożsamości, 296
 brak uznania, 296
 brak wspólnej wizji, 295
 brak zaufania, 296
 budowanie, 300
 członkowie, 289
 długoterminowe budowanie, 298
 funkcyjny, 312
 modele, 308
 narzędziowy, 355, 579
 nieskuteczna komunikacja, 296
 poczucie autonomii, 293
 poczucie podmiotowości, 293
 poczucie tożsamości, 288
 produktywność, 284
 ratunkowy, 313
 rozmiar, 294
 rozwojowy, 311
 skuteczna komunikacja, 292
 sportowy, 314
 spójność, 285
 struktura, 303, 305, 307
 SWAT, 313
 teatralny, 315
 utrudnienia produktywności, 296
 wydajność, 285
 wzajemne zaufanie, 292
z głównym programistą, 310
zarządzanie, 294
zastosowania, 283
zorientowany
 na kreatywność, 306
 na rozwiązywanie problemów, 306
 na taktyczne wykonywanie zadań, 306
zewnątrzni podwykonawcy, 487
zmiany, 338, 416
zmniejszenie ryzyka, 249
znaczenie pracy zespołowej, 284
zróżnicowanie umiejętności, 267

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>

Odzyskaj kontrolę nad swoim projektem i zrealizuj go w terminie!

Zespoły projektowe borykają się z ciągłym niedostatkiem czasu. Napięte do granic możliwości terminy wymuszają na software developerach narzucenie morderczego tempa pracy. Takie podejście sprawia, że albo dostarczony produkt nie spełnia oczekiwań, albo nie udaje się dotrzymać terminu. Co gorsza, ciągła praca pod presją czasu powoduje chroniczne przemęczenie i problemy zdrowotne, nie wspominając już o braku sił i czasu na rozwój, który w branży IT ma kolosalne znaczenie.

Książka ta jest praktycznym, zdroworozsądkowym poradnikiem metod projektowania. Opisane w niej strategie pracy pozwolą na usprawnienie i przyspieszenie procesu projektowego. Przedstawiono tu również takie zagadnienia jak zarządzanie ryzykiem, podstawy projektowania aplikacji oraz planowanie cyklu życia projektu. Mimo że nie są bezpośrednio związane z metodami szybkiego projektowania, mają kluczowe znaczenie dla produktywności zespołu. Naturalnie, nie istnieje jedna magiczna metoda przydatna w każdych warunkach — w tej książce opisano i krytycznie przeanalizowano najprzydatniejsze rozwiązania z różnych branż tworzenia oprogramowania.

- strategię szybkiego projektowania i sprawdzone rozwiązania
- rozwiązania przyspieszające realizację projektu, takie jak prototypowanie, języki szybkiego projektowania, zasady motywowania zespołu oraz zasady wydajnej współpracy
- najczęściej popełniane błędy, ich przyczyny i konsekwencje
- studia przypadków oparte na rzeczywistych wydarzeniach
- dobieranie właściwych metod do poszczególnych projektów

Steve McConnell jest głównym inżynierem oprogramowania i dyrektorem generalnym w spółce Construx Software Builders. Jest także członkiem organizacji IEEE Computer Society oraz ACM. Aktywny programista, koncentruje się głównie na projektowaniu komercyjnego oprogramowania „celofanowego” (ang. shrink-wrap). Współpracuje z wieloma znanymi firmami, w tym z korporacją Microsoft. Wraz z żoną i z dziećmi mieszka w Bellevue w stanie Waszyngton.

Microsoft

Helion

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

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

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

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-3270-6



9 788328 332706

cena: 99,00 zł