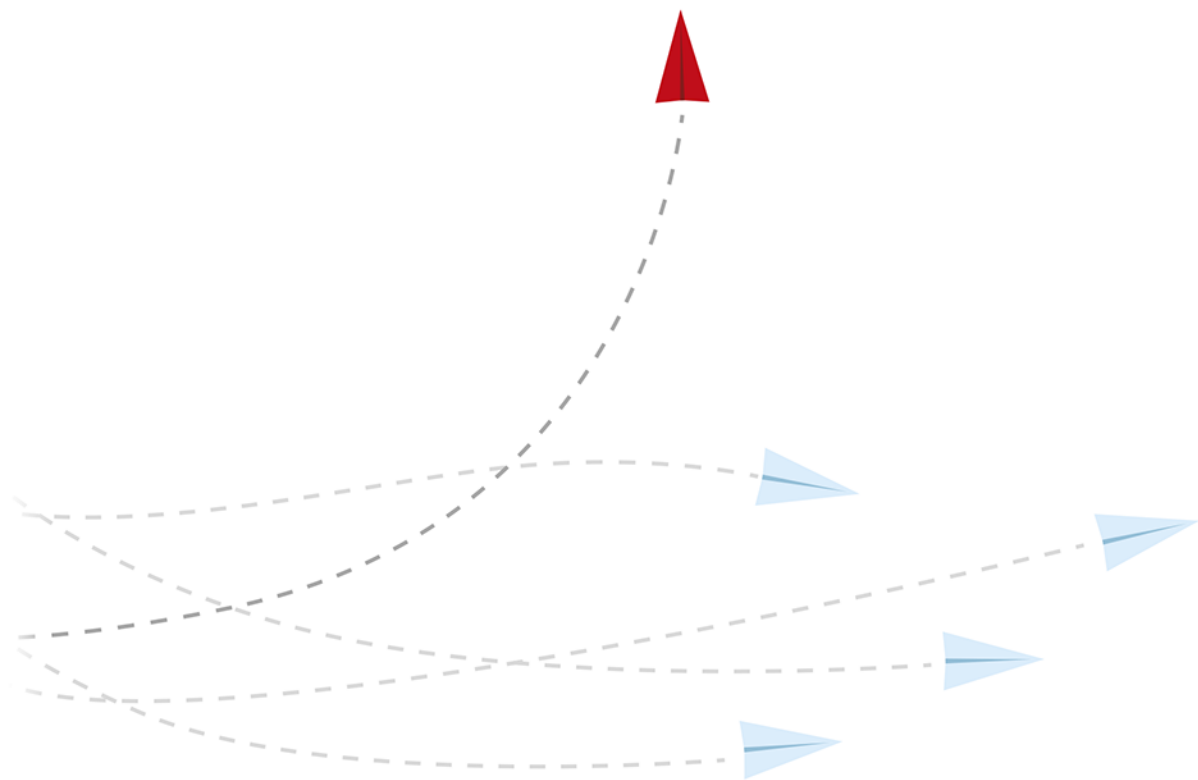




# PROGRAMUJ TAK, ABY NIE NAPRAWIAĆ

[ Planowanie projektów i systemów ]



Tytuł oryginału: Righting Software

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-6793-7

Authorized translation from the English language edition, entitled RIGHTING SOFTWARE, 1st Edition by LÖWY, JUVAL, published by Pearson Education, Inc, publishing as Addison-Wesley Professional, Copyright © 2020 Pearson Education, Inc.

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.

POLISH language edition published by Helion SA, Copyright 2021.

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 Helion SA 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 Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto: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/protak>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

---

# SPIS TREŚCI

---

<b>Wstęp</b>		<b>19</b>
<b>Podziękowania</b>		<b>29</b>
<b>O autorze</b>		<b>31</b>
<b>Rozdział 1</b>	<b>Metoda</b>	<b>35</b>
	Czym jest metoda?	36
	Weryfikacja projektu	37
	Presja czasu	38
	Eliminowanie paraliżu analizy	39
	Komunikacja	41
	Czym metoda nie jest?	42
<b>CZĘŚĆ I</b>	<b>PROJEKT SYSTEMU</b>	<b>43</b>
<b>Rozdział 2</b>	<b>Dekompozycja</b>	<b>45</b>
	Unikanie dekompozycji funkcjonalnej	46
	Problemy dekompozycji funkcjonalnej	46
	Wnioski o dekompozycji funkcjonalnej	52
	Unikanie dekompozycji dziedziny	55
	Błędna motywacja	57
	Możliwości testowania i projekt	58
	Przykład: system handlowy zaprojektowany funkcjonalnie	60

Dekompozycja w oparciu o niestabilność	63
Dekompozycja, utrzymanie i wdrażanie	65
Uniwersalna zasada	65
Dekompozycja w oparciu o niestabilność i testowanie	67
Wyzwanie niestabilności	67
Identyfikacja niestabilności	70
Niestabilny a zmienny	70
Osie niestabilności	70
Rozwiązania udające wymagania	74
Lista niestabilności	75
Przykład: system obrotu akcjami tworzony z użyciem dekompozycji w oparciu o niestabilność	75
Nie ulegać syreniemu śpiewowi	81
Niestabilność a biznes	81
Projektowanie z myślą o konkurentach	84
Niestabilność i długowieczność	85
Znaczenie praktyki	86

<b>Rozdział 3</b>	<b>Struktura</b>	<b>89</b>
	Przypadki użycia i wymagania	90
	Wymagane zachowania	90
	Podjęcie warstwowe	93
	Stosowanie usług	93
	Typowe warstwy	94
	Warstwa klienta	95
	Warstwa logiki biznesowej	96
	Warstwa dostępu do zasobów	98
	Warstwa zasobu	99
	Pasek narzędzi	99
	Wskazówki dotyczące klasyfikacji	100
	Co zawiera nazwa?	100
	Cztery pytania	101
	Proporcja menedżerów do silników	102
	Kluczowe obserwacje	103
	Podsystemy i usługi	105
	Konstrukcja inkrementalna	105
	O mikrousługach	107
	Architektury otwarte i zamknięte	110
	Architektura otwarta	110
	Architektura zamknięta	111
	Architektura półzamknięta/półotwarta	111

---

	Rozluźnianie reguł	112
	Czego należy unikać?	115
	Dążenie do symetrii	117
<b>Rozdział 4</b>	<b>Kompozycja</b>	<b>119</b>
	Wymagania i zmiany	119
	Awersja do zmian	120
	Główne zalecenie programowe	120
	Projekt umożliwiający kompozycję	121
	Podstawowe przypadki użycia	121
	Misja architekta	122
	Brak możliwości	127
	Obsługa zmian	128
	Opanowanie zmian	129
<b>Rozdział 5</b>	<b>Przykład projektu systemu</b>	<b>131</b>
	Przegląd systemu	132
	Stary system	133
	Nowy system	135
	Firma	135
	Przypadki użycia	136
	Wysiłki antyprojektowe	141
	Monolit	142
	Szczegółowe elementy konstrukcyjne	142
	Dekompozycja na podstawie dziedziny	144
	Zgodność z działalnością biznesową	146
	Wizja	146
	Cele biznesowe	147
	Myśl przewodnia	148
	Architektura	149
	Słownik systemu TradeMe	149
	Obszary niestabilności systemu TradeMe	150
	Architektura statyczna	153
	Koncepcje operacyjne	156
	Menedżer toku pracy	159
	Weryfikacja projektu	160
	Przypadek użycia: dodanie fachowca/dewelopera	161
	Przypadek użycia: żądanie wyboru fachowca	162
	Przypadek użycia: dopasowanie fachowca	163
	Przypadek użycia: przypisanie fachowca	166
	Przypadek użycia: zakończenie pracy fachowca	169

---

Przypadek użycia: zapłata fachowcowi	170
Przypadek użycia: utworzenie projektu	171
Przypadek użycia: zamknięcie projektu	171
Co dalej?	173

## **CZĘŚĆ II PLAN PROJEKTU 175**

### **Rozdział 6 Motywacja 177**

Do czego jest potrzebny plan projektu?	177
Plan projektu i zdrowy rozsądek	179
Instrukcja wykonania	179
Hierarchia potrzeb	180

### **Rozdział 7 Przegląd planu projektu 183**

Definiowanie sukcesu	183
Raportowanie sukcesu	184
Początkowa obsada projektu	185
Architekt, nie architekci	185
Podstawowy zespół	187
Mądre decyzje	190
Plany, a nie plan	190
Przegląd planu realizacji oprogramowania	191
Usługi i programiści	192
Projekt a wydajność zespołu	193
Ciągłość zadań	195
Szacowanie pracochłonności	196
Klasyczne błędy	197
Techniki szacowania	199
Ogólne szacunki projektu	201
Szacunki dotyczące czynności	204
Analiza ścieżki krytycznej	205
Sieć projektu	206
Ścieżka krytyczna	209
Przydzielanie zasobów	210
Określanie harmonogramu czynności	215
Dystrybucja obsady	216
Koszty projektu	223
Efektywność projektu	224

---

Planowanie wartości wypracowanej	226	
Klasyczne błędy	228	
Płytką krzywa S-kształtna	230	
Role i odpowiedzialności	233	
<b>Rozdział 8</b>	<b>Sieć i zapas</b>	<b>235</b>
Diagram sieci	235	
Diagram węzłów	236	
Diagram strzałkowy	236	
Diagramy strzałkowe a diagramy węzłów	237	
Zapasy	240	
Zapas całkowity	240	
Zapas swobodny	241	
Obliczanie zapasów	242	
Wizualizacja zapasów	243	
Planowanie w oparciu o zapasy	245	
Zapas i ryzyko	247	
<b>Rozdział 9</b>	<b>Czas i koszty</b>	<b>249</b>
Przyspieszanie projektów programistycznych	249	
Skracanie harmonogramu	252	
Stosowanie lepszych zasobów	253	
Praca równoległa	253	
Praca równoległa i koszty	255	
Krzywa zależności czas-koszt	256	
Punkty na krzywej zależności czas-koszt	257	
Modelowanie dyskretne	259	
Unikanie klasycznych błędów	260	
Wykonalność projektu	261	
Znajdowanie rozwiązania normalnego	263	
Elementy kosztu projektu	265	
Koszty bezpośrednie	265	
Koszty pośrednie	266	
Księgowanie a wartość	266	
Koszt całkowity, koszty bezpośrednie i pośrednie	266	
Skracanie i elementy kosztów	268	
Obsada a elementy kosztów	272	
Koszty stałe	273	
Skracanie sieci	274	
Przebieg skracania	275	

---

---

<b>Rozdział 10</b>	<b>Ryzyko</b>	<b>277</b>
	Wybór wariantu	278
	Krzywa zależności czas-ryzyko	278
	Faktyczna krzywa zależności ryzyko-czas	280
	Modelowanie ryzyka	281
	Normalizacja ryzyka	281
	Ryzyko a zapasy	282
	Ryzyko i koszty bezpośrednie	283
	Ryzyko krytyczności	283
	Ryzyko Fibonacciego	286
	Ryzyko czynności	288
	Ryzyko krytyczności a ryzyko czynności	290
	Skracanie a ryzyko	290
	Ryzyko wykonania	291
	Dekompresja ryzyka	292
	Sposoby przeprowadzania dekompresji	292
	Cel dekompresji	293
	Metryki ryzyka	295
<b>Rozdział 11</b>	<b>Planowanie projektu w praktyce</b>	<b>297</b>
	Cel	298
	Stacyczna architektura	298
	Łącuchy wywołań	299
	Lista czynności	302
	Diagram sieci	303
	Założenia do planu	305
	Znajdowanie rozwiązania normalnego	307
	Nieograniczone zasoby (iteracja 1.)	307
	Problemy z siecią i zasobami	309
	Najpierw infrastruktura (iteracja 2.)	309
	Ograniczone zasoby	311
	Zejscie na poziom podkrytyczny (iteracja 7.)	315
	Wybór rozwiązania normalnego	318
	Skracanie sieci	319
	Skracanie poprzez wykorzystanie lepszych zasobów	319
	Wprowadzanie pracy równoległej	321
	Koniec iteracji skracania	329
	Analiza przepustowości	330
	Analiza efektywności	332
	Krzywa zależności czas-koszt	333
	Modele korelacji zależności czas-koszt	333
	Strefa śmierci	335

---



---

Planowanie i ryzyko	337
Dekompresja ryzyka	337
Ponowne wyznaczenie krzywej zależności czas-koszt	342
Modelowanie ryzyka	343
Włączanie i wykluczanie ryzyka	346
Przegląd PRO	347
Prezentacja wariantów	348
<b>Rozdział 12 Techniki zaawansowane</b>	<b>349</b>
Boskie czynności	349
Postępowanie z boskimi czynnościami	350
Punkt przecięcia ryzyka	351
Wyznaczanie punktu przecięcia ryzyka	351
Znajdowanie celu dekompresji	355
Ryzyko geometryczne	357
Geometryczne ryzyko krytyczności	358
Geometryczne ryzyko Fibonacciego	359
Geometryczne ryzyko czynności	360
Zachowanie geometrycznego ryzyka czynności	360
Złożoność wykonania	363
Złożoność cyklomatyczna	363
Typ projektu i złożoność	364
Skracanie i złożoność	365
Bardzo duże projekty	367
Złożone systemy i wrażliwość	367
Sieć sieci	370
Tworzenie sieci sieci	371
Małe projekty	374
Planowanie w oparciu o warstwy	375
Zalety i wady	376
Warstwy i konstruowanie	377
<b>Rozdział 13 Przykład planowania projektu</b>	<b>379</b>
Szacunki	380
Szacunki poszczególnych czynności	380
Ogólne oszacowanie projektu	383
Zależności i sieć projektu	383
Zależności behawioralne	383
Zależności niebehawioralne	384
Nadpisywanie niektórych zależności	385
Sprawdzenie sensowności	386

---

---

Rozwiązanie normalne	386
Diagram sieci	387
Planowane postępy	388
Planowany rozkład obsady	388
Koszt i efektywność	389
Podsumowanie wyników	390
Rozwiązanie skrócone	390
Dodanie czynności umożliwiających skrócenie	390
Przydzielanie zasobów	392
Planowane postępy	392
Planowany rozkład obsady	392
Koszt i efektywność	394
Podsumowanie wyników	394
Planowanie w oparciu o warstwy	395
Planowanie w oparciu o warstwy i ryzyko	396
Rozkład obsady	396
Podsumowanie wyników	396
Rozwiązanie subkrytyczne	397
Czas realizacji, planowane postępy i ryzyko	397
Koszt i efektywność	398
Podsumowanie wyników	398
Porównanie wariantów	399
Planowanie i ryzyko	400
Dekompresja ryzyka	400
Przeliczenie kosztu	403
Przygotowanie przeglądu PRO	404
<b>Rozdział 14 Wnioski podsumowujące</b>	<b>407</b>
Kiedy planować projekt?	407
Prawdziwa odpowiedź	408
Wybieganie w przód	409
Wskazówki ogólne	411
Architektura a szacunki	411
Podejście do planowania	412
Opcjonalność	412
Skracanie	413
Planowanie i ryzyko	416
Planowanie planu projektu	417
Z perspektywy	419
Podsystemy i oś czasu	420

---

Przekazanie	421
Przekazanie do młodszych programistów	422
Przekazanie do starszych programistów	422
Starsi programiści a młodszy programiści	423
W praktyce	424
Przeglądanie planów projektu	425
Kilka słów o jakości	427
Czynności związane z kontrolą jakości	428
Czynności związane z zapewnianiem jakości	429
Jakość i kultura	430
<b>Dodatek A Śledzenie projektów</b>	<b>431</b>
Cykl życia i stan czynności	432
Warunki zakończenia fazy	434
Waga fazy	435
Stan czynności	435
Stan projektu	437
Postęp i wartość wypracowana	437
Łączny wysiłek	438
Łączny koszt pośredni	439
Śledzenie postępu i wysiłku	439
Prognozowanie	440
Prognozy i czynności korekcyjne	443
Wszystko jest w porządku	443
Niedoszacowanie	444
Wyciek zasobów	446
Przeszacowanie	447
Więcej informacji o prognozach	448
Kwintesencja projektu	449
Postępowanie w przypadku zmian zakresu	449
Budowanie zaufania	450
<b>Dodatek B Projektowanie kontraktu usługi</b>	<b>451</b>
Czy ten projekt jest dobry?	452
Modularność a koszt	453
Koszt na usługę	454
Koszt integracji	455
Obszar minimalnych kosztów	455

---

Usługi i kontrakty	456
Kontrakty i aspekty	456
Od projektu usługi do projektu kontraktu	457
Cechy dobrego kontraktu	457
Wyodrębnianie kontraktów	459
Przykład projektu	460
Wyodrębnianie w dół	460
Wyodrębnianie w bok	461
Wyodrębnianie w górę	463
Metryki projektów kontraktów	463
Pomiary kontraktów	464
Metryki wielkości	464
Unikanie właściwości	465
Ograniczanie liczby kontraktów	466
Stosowanie metryk	466
Wyzwanie projektowania kontraktów	467
<b>Dodatek C</b>	
<b>Standardy projektowania</b>	<b>469</b>
Dyrektywa podstawowa	470
Dyrektywy	470
Wskazówki dotyczące projektowania systemu	470
Wytyczne planowania projektów	472
Wytyczne dotyczące śledzenia projektów	474
Wytyczne dotyczące projektowania kontraktów usług	474

---

# METODA

---

*Zen of Architects*<sup>1</sup> stwierdza jedynie tyle, że początkujący architekt wszystko może robić na wiele sposobów. Natomiast dla mistrzów architektury istnieje jedynie kilka opcji, a przeważnie tylko jedna.

Początkujący architekci często są przytłoczeni mnogością wzorców, podejść, metod oraz możliwości projektowania tworzonego systemu oprogramowania. Branża komputerowa pęka w szwach od liczby koncepcji, jak również osób pragnących uczyć się i samodoskonalić, a Ty, Czytelniku tej książki, zaliczasz się do tego grona. Niemniej jednak, ponieważ istnieje bardzo niewiele prawidłowych sposobów na wykonanie danych zadań projektowych, równie dobrze można się skoncentrować tylko na nich i pominąć pozostały szum. Mistrzowie projektowania oprogramowania wiedzą, że tak należy robić; dzięki jakby nadnaturalnej inspiracji są w stanie błyskawicznie wskazać i zastosować prawidłowe rozwiązanie projektowe.

*Zen of Architects* znajduje zastosowanie nie tylko podczas projektowania systemów, lecz także w projektach, w ramach których te systemy powstają. Owszem, istnieją niezliczone sposoby określania struktury projektów i przypisywania zadań poszczególnym członkom zespołu, jednak czy wszystkie one są równie bezpieczne, szybkie, tanie, użyteczne i efektywne? Mistrzowie architektury oprogramowania planują także projekty, w ramach których będą tworzone systemy, a nawet pomagają kadrze kierowniczej w podejmowaniu decyzji, czy organizację będzie stać na wykonanie projektu.

Osiągnięcie prawdziwego mistrzostwa w każdej dziedzinie jest podróżą. Oprócz bardzo nielicznych wyjątków nikt nie rodzi się ekspertem. Doskonałym tego przykładem jest moja

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Zen\\_Mind,\\_Beginner's\\_Mind](https://en.wikipedia.org/wiki/Zen_Mind,_Beginner's_Mind)

własna kariera. Rozpocząłem pracę jako młodszy architekt niemal 30 lat temu, kiedy określenie „architekt oprogramowania” nie było powszechnie używane w branży komputerowej. Następnie pełniłem funkcję architekta projektu, kierownika działu projektowania, a w końcu, pod koniec lat 90. ubiegłego wieku, byłem głównym projektantem w jednej z firm z listy Fortune 100 w Dolinie Krzemowej. W roku 2000 założyłem IDesign — firmę zajmującą się jedynie projektowaniem oprogramowania. Od tamtej pory zaprojektowaliśmy w niej setki systemów i projektów. Choć każde zlecenie miało swoją specyficzną architekturę i własny plan tworzenia projektu, zauważyłem, że niezależnie od klienta, projektu, systemu, technologii czy programistów moje zalecenia projektowe były — w pewnym abstrakcyjnym sensie — takie same.

Dlatego zadałem sobie proste pytanie: czy naprawdę trzeba być mistrzem architektury oprogramowania dysponującym kilkudziesięcioma latami doświadczenia i dziesiątkami wykonanych projektów, by wiedzieć, co należy robić? I drugie pytanie: czy można w jakiś sposób usystematyzować tę wiedzę, by każdy dysponujący dobrym zrozumieniem stosowanych metod programista mógł tworzyć solidne systemy i dobre projekty?

Odpowiedź na to drugie pytanie jest bezsprzecznie twierdząca. Nazywam to *metodą* i to właśnie ona jest tematem niniejszej książki. Po zastosowaniu *metody* w bardzo wielu projektach i nauczaniu jej kilku tysięcy architektów na całym świecie mogę zagwarantować, że prawidłowo zastosowana, da ona dobre efekty. Nie dyskredytuję tu znaczenia właściwego nastawienia, umiejętności technicznych i możliwości analitycznych. Są one elementami koniecznymi do osiągnięcia sukcesu niezależnie od stosowanej metody. Niestety jednak nie są one jednocześnie składnikami wystarczającymi — często spotykam się z projektami, które zakończyły się porażką pomimo tego, że były tworzone przez osoby posiadające doskonale kwalifikacje i przymioty. Niemniej kiedy zostaną one połączone z *metodą*, składniki te zapewnią siłę niezbędną do odniesienia sukcesu. Opierając swój projekt na solidnych podstawach inżynierskich, nauczysz się unikać wszechobecnych złych praktyk i błędnej intuicji.

## Czym jest metoda?

*Metoda* jest prostą i efektywną techniką analizy i projektowania. Można by ją wyrazić, używając takiego oto wzoru:

$$\textit{metoda} = \text{projekt systemu} + \text{plan projektu}$$

W ramach projektu systemu *metoda* określa sposób podziału dużych systemów na mniejsze, modularne komponenty. Udostępnia ona wytyczne na temat struktury, roli oraz semantyki komponentów oraz określa, jak komponenty te powinny ze sobą współdziałać. Efektem jest architektura systemu.

Jeśli chodzi o plan projektu, *metoda* pomaga zapewnić zarządzanie tworzeniem systemu, udostępniając przy tym kilka opcji. Każda z tych opcji stanowi połączenie harmonogramu,

kosztów oraz ryzyka. Oprócz tego każda z tych opcji stanowi instrukcję konstruowania systemu i przygotowuje projekt pod kątem jego realizacji i śledzenia.

Planowanie projektu jest tematem drugiej części tej książki i ma znacznie większe znaczenie dla odniesienia końcowego sukcesu niż projekt systemu. Nawet dysponując przeciętnym projektem systemu, można odnieść sukces, jeśli będziemy dysponować odpowiednim czasem i zasobami oraz jeśli ryzyko jest możliwe do zaakceptowania. Niemniej jednak nawet jeżeli będziemy dysponować najlepszym projektem systemu, nasz projekt zakończy się porażką, jeśli nie będziemy dysponować odpowiednim czasem czy zasobami albo jeśli projekt będzie zbyt ryzykowny. Oprócz tego planowanie projektu jest bardziej skomplikowane niż projekt systemu i, co za tym idzie, wymaga dodatkowych narzędzi, koncepcji oraz technik.

Ponieważ *metoda* składa się zarówno z projektu systemu, jak i planu projektu, stanowi ona proces projektowy. Na przestrzeni lat branża informatyczna poświęciła bardzo wiele uwagi, by stworzyć proces wytwarzania oprogramowania, a jednocześnie nie przykładała jej do procesu projektowania. Niniejsza książka ma wypełnić te luki.

## Weryfikacja projektu

Weryfikacja projektu ma kluczowe znaczenie, gdyż organizacja nie powinna ryzykować, by zespół rozpoczął prace nad projektem o nieodpowiedniej architekturze lub takim, na którego stworzenie organizacji w ogóle nie stać. *Metoda* wspiera i umożliwia wykonanie tego krytycznego zadania, pozwalając architektowi na ustalenie, z rozsądnym poziomem pewności, czy proponowany projekt jest odpowiedni, czyli czy spełnia dwa kluczowe cele: po pierwsze, projekt musi zaspokajać wymagania klienta, a po drugie, musi uwzględniać możliwości i ograniczenia organizacji bądź zespołu.

Kiedy zostaną już rozpoczęte prace nad kodem, modyfikowanie architektury zazwyczaj będzie już niedopuszczalne ze względu na wiążące się z tym koszty i opóźnienia. W praktyce oznacza to, że bez weryfikacji projektu systemu istnieje ryzyko utrwalenia architektury, która w najlepszym razie będzie niedoskonała, a w najgorszym — monstrualna. Organizacja będzie musiała starać się działać, używając takiego systemu przez kilka następnych lat, aż po kilku jego wersjach zdecyduje się na znaczącą modyfikację lub przepisanie. Kiepsko zaprojektowane oprogramowanie może w dużym stopniu zaszkodzić prowadzonej działalności, ograniczając możliwości reagowania na okazje biznesowe, a nawet doprowadzić organizację do bankructwa wskutek rosnących kosztów jego utrzymania.

Wczesne weryfikowanie projektu ma ogromne znaczenie. Na przykład, choć stwierdzenie po trzech latach od rozpoczęcia prac, że dana koncepcja lub cała architektura jest nieprawidłowa, jest interesujące pod względem intelektualnym, to jednak pod względem praktycznym nie ma ono żadnej wartości. W idealnym przypadku po tygodniu od rozpoczęcia prac nad projektem należy wiedzieć, czy dana architektura będzie właściwa. Każde przekroczenie

tego czasu zwiększa ryzyko prac nad architekturą, której poprawność będzie dyskusyjna. W kolejnych rozdziałach dokładnie opiszę, w jaki sposób należy weryfikować projekt systemu.

Należy zwrócić uwagę, że odnoszę się tutaj do ogólnego projektu systemu — architektury — a nie szczegółowego projektu systemu. Projekt szczegółowy dla każdego z komponentów architektury tworzy kluczowe artefakty implementacji, takie jak hierarchie interfejsów i klas oraz kontrakty danych. Tworzenie szczegółowych projektów trwa dłużej, można je wykonywać podczas prac nad projektem, a oprócz tego mogą one ulegać zmianom w trakcie ewolucji konstruowanego systemu.

Analogicznie należy także weryfikować plan projektu. Przekroczenie limitu czasu lub dostępnego budżetu (bądź obu) podczas prac nad projektem jest całkowicie niedopuszczalne. Nieumiejętność wywiązywania się z zobowiązań może zaszkodzić naszej karierze. Dlatego należy aktywnie weryfikować plan projektu, by upewnić się, że zespół jest w stanie zrealizować projekt.

Oprócz przygotowania planów architektury i projektu celem *metody* jest wyeliminowanie projektu jako ryzyka dla powodzenia realizacji całego przedsięwzięcia. Żaden projekt nie powinien zakończyć się niepowodzeniem dlatego, że architektura była zbyt złożona, by programiści byli w stanie ją przygotować i utrzymać. *Metoda* pozwala określić architekturę sprawnie i efektywnie i zrobić to dość szybko. Dokładnie to samo dotyczy planu projektu. Żaden projekt nie powinien zakończyć się niepowodzeniem dlatego, że od samego początku na jego wykonanie brakowało czasu lub zasobów. Tak książka pokazuje, w jaki sposób można dokładnie wyliczyć czas prac nad projektem i jego koszty oraz jak podejmować przemyślane decyzje.

## Presja czasu

Używając *metody*, można stworzyć cały projekt systemu w ciągu kilku dni, zazwyczaj od trzech do pięciu. Przygotowanie planu projektu zajmuje podobny okres czasu. Zważywszy na szczytne cele podejmowanych wysiłków, a konkretnie przygotowanie architektury systemu oraz planów projektu nowego systemu, ten czas może się wydawać zbyt krótki. Typowe systemy biznesowe dają możliwość tworzenia nowych projektów jedynie co kilka lat. Dlaczego nie można by poświęcić 10 dni na przygotowanie architektury? Wziąwszy pod uwagę późniejszy czas stosowania takiego systemu, liczony w latach, te pięć dodatkowych dni nie stanowi nawet błędu zaokrąglenia. Niemniej jednak wydłużenie czasu projektowania często nie poprawia jego efektów, a niekiedy nawet sprawia, że będą one gorsze.

Większość środowisk roboczych cechuje się horrendalnie niewydajnym zarządzaniem czasem, co głównie wynika z natury ludzkiej. Presja czasu zmusza nas (jak również wszystkie inne zaangażowane osoby) do skoncentrowania się, określenia priorytetów oraz wykonania projektu. *Metodę* należy zrealizować szybko i zdecydowanie.



Ogólnie rzecz biorąc, projektowanie (w przeciwieństwie do implementacji) nie jest zadaniem czasochłonnym. Architekci budowlani na projekt budynku poświęcają przeważnie nie więcej niż dwa tygodnie, zazwyczaj mieszcząc się w tygodniu, z kolei wzniesienie budynku na podstawie takiego projektu może trwać nawet dwa lub trzy lata, o ile zatrudnimy z podwykonawców.

Presja czasu pomaga także unikać cyzelowania projektu. Prawo Parkinsona<sup>2</sup> głosi, że prace zawsze przeciągają się, by zająć cały przeznaczony na nie czas. Mając 10 dni na wykonanie projektu, który można zakończyć w 5 dni, architekt najprawdopodobniej będzie nad nim pracował przez 10 dni. Dodatkowy czas architekt poświęci na stworzenie błahych aspektów projektu, które nie dodają do niego niczego oprócz złożoności, co w nieproporcjonalny sposób zwiększy w następnych latach koszty implementacji i utrzymania. Ograniczanie czasu zmusza nas na przygotowania projektu, który będzie jedynie wystarczająco dobry.

## Eliminowanie paraliżu analizy

Paraliż analizy to kłopot, który przydarza się, kiedy osoba (albo grupa), która jest zdolna, inteligentna, a nawet pracowita (jak to zazwyczaj jest z architektami oprogramowania), utknie w nieskończonym cyklu analizy, projektowania, nowych odkryć i ponownej analizy. Taka osoba (lub grupa) ulega paraliżowi i nie jest w stanie stworzyć żadnego produktywnego wyniku.

## Drzewo decyzyjne projektu

Głównym powodem występowania paraliżu jest brak wiedzy o istnieniu drzewa decyzyjnego zarówno dla systemu, jak i dla projektu. Drzewo decyzyjne projektu jest ogólnym pojęciem odnoszącym się do wszelkich zadań projektowych — nie jest powiązane wyłącznie z inżynierią oprogramowania. Każdy projekt o pewnym poziomie złożoności jest zbiorem wielu mniejszych decyzji projektowych, uporządkowanych hierarchicznie w formie struktury drzewiastej. Każde rozgałęzienie w tym drzewie reprezentuje możliwą opcję decyzyjną, prowadzącą do kolejnych, bardziej szczegółowych decyzji projektowych. Liście tego drzewa reprezentują gotowe rozwiązania projektowe dla istniejących wymagań. Każdy z tych liści jest spójnym, unikalnym i prawidłowym rozwiązaniem, które w pewien sposób różni się od wszystkich pozostałych.

Kiedy osoba lub grupa odpowiedzialna za stworzenie projektu nie zna prawidłowego drzewa decyzyjnego, może zacząć prace od innego miejsca niż jego korzeń. W takim przypadku jakaś decyzja projektowa w nieuchronny sposób doprowadzi do konfliktu z którąś z poprzednich decyzji, a co za tym idzie, konieczne będzie odrzucenie wszystkich decyzji pośrednich. Projektowanie w taki sposób można by porównać z sortowaniem bąbelkowym drzewa decyzyjnego projektu. Ponieważ liczba operacji w przypadku

<sup>2</sup> Cyryl N. Parkinson, *Parkinson's Law*, „The Economist”, 19 listopada 1955.

sortowania bąbelkowego odpowiada kwadratowi liczby sortowanych elementów, konsekwencje takiego postępowania będą poważne. Prostszy system, wymagający podjęcie 20 decyzji związanych z zaplanowaniem systemu i projektu, może potencjalnie wymagać 400 iteracji projektu, o ile nie będziemy pracować w oparciu o drzewo decyzyjne. Przeprowadzenie takiej liczby spotkań (nawet jeśli będą one rozłożone w czasie) doprowadzi do paraliżu. Jest mało prawdopodobne, byśmy mieli czas na wykonanie nawet 40 iteracji projektu. Kiedy zakończy się czas przeznaczony na zaplanowanie systemu i projektu, pisanie kodu rozpocznie się, kiedy ani system, ani projekt nie będą jeszcze dostatecznie dojrzałe. To z kolei opóźnia odkrycia prowadzące do unieważniania decyzji projektowych, odsuwając je w jeszcze odleglejszą przyszłość, kiedy czas, wysiłki oraz artefakty zostaną już skojarzone z niewłaściwymi wyborami. W zasadzie oznacza to maksymalizację kosztów niewłaściwych decyzji projektowych.

### Drzewo decyzyjne projektu systemu software'owego

Jak się okazuje, większość software'owych systemów biznesowych ma ze sobą wiele wspólnego i przynajmniej zarys drzewa decyzyjnego będzie dla nich nie tylko wspólny, lecz także jednakowy. Oczywiście liście tych drzew będą odmienne.

*Metoda* określa postać drzew decyzyjnych dla typowych systemów biznesowych, i to zarówno dla projektu systemu, jak i dla planu projektu. Dopiero po zaprojektowaniu systemu jest sens zabierać się za planowanie projektu, który pozwoli zbudować ten system. Każdy z tych wysiłków projektowych, zarówno związany z systemem, jak i z projektem, stanowi odrębne poddrzewo decyzji projektowych. *Metoda* prowadzi nas przez te decyzje, zaczynając od korzenia drzewa, dzięki czemu pozwala uniknąć przerabiania i ponownego oceniania wcześniejszych decyzji.

Jedną z najcenniejszych technik oczyszczania drzewa decyzyjnego jest stosowanie ograniczeń. Jak zauważył doktor Frederick Brooks<sup>3</sup>, w przeciwieństwie do powszechnej opinii i założeń, najgorszym problemem projektowym jest pusta karta. Bez żadnych ograniczeń projekt powinien być bardzo prosty, nieprawdaż? Otóż nie. Pusta karta powinna przerażać każdego architekta. Istnieje nieskończenie wiele sposobów, by zboczyć z właściwej drogi albo przekroczyć ograniczenia, które nie zostały jawnie określone. Im więcej jest ograniczeń, tym projektowanie jest prostsze. Im mniej mamy swobody, tym projekt będzie bardziej czywisty i przejrzysty. W systemie, w którym występują same ograniczenia, nie ma nic do projektowania — system jest jednoznaczny. Jednak zawsze istnieją jakieś ograniczenia (jawne bądź niejawne), dlatego w przypadku postępowania zgodnie z drzewem decyzyjnym *metoda* narzuca rosnące ograniczenia zarówno na system, jak i na projekt, dzięki czemu plan wyłania się i kształtuje całkiem szybko.

---

<sup>3</sup> Frederick P. Brooks Jr., *The Design of Design: Esseys from a Computer Science*, Addison-Wesley, Upper Saddle River, 2010.

## Komunikacja

Ważną zaletą *metody* jest komunikowanie koncepcji projektowych. Kiedy osoby biorące udział w pracach poznają już strukturę architektury i semantykę projektu, *metoda* pozwoli im dzielić się koncepcjami projektowymi i precyzyjnie przekazywać wymagania projektu. Proces myślowy leżący u podstaw projektu można przekazywać zespołowi. Należy dzielić się informacjami o kompromisach i pomysłach, które prowadziły nas do powstania architektury, dokumentując w jednoznaczny i zrozumiały sposób przyjmowane założenia do planu i wynikające z nich decyzje projektowe.

Ten poziom przejrzystości w intencjach projektowych ma krytyczne znaczenie dla przetrwania architektury. Dobry projekt to taki, który został poprawnie wymyślony, przetrwał proces implementacji i zakończył się, działając na komputerach klientów. Musimy być w stanie przekazać ten projekt programistom i zagwarantować, że oni zrozumieją i docenią pomysły leżące u jego podstaw. Ten projekt należy wyegzekwować, stosując kontrole, inspekcje oraz mentoring. *Metoda* celuje w komunikacji tego typu dzięki wykorzystaniu połączenia dobrze zdefiniowanej semantyki usług i struktury.

Można mieć pewność, że jeśli programiści pracujący nad stworzeniem systemu nie rozumieją i nie docenią projektu, spartaczą go. I żaden plan ani żadne inspekcje nie będą w stanie temu zapobiec. Celem tych kontroli powinno być możliwie wczesne wychwytywanie niezamierzonych odstępstw od zaplanowanej architektury.

Dokładnie tak samo dzieje się w przypadku komunikowania planu projektu menadżerom projektu, kierownictwu oraz interesariuszom. Przejrzyste, jednoznaczne i zapewniające możliwość porównywania opcje mają kluczowe znaczenie dla podejmowania świadomych i przemyślanych decyzji. Kiedy ludzie podejmują błędne decyzje, często wynika to z niezrozumienia projektu i wykształcenia nieprawidłowego mentalnego modelu sposobu jego działania. Tworząc prawidłowe modele projektu dla czasu, kosztów i ryzyka, architekt może umożliwić podejmowanie właściwych decyzji. *Metoda* zapewnia doskonały słownik i wskaźniki do prowadzenia prostej i świadomej komunikacji z osobami odpowiedzialnymi za podejmowanie decyzji. Kiedy menadżerowie będą już znali możliwości planu projektu, staną się jego największymi orędownikami i będą nalegać na pracę w proponowany sposób. Żadna ilość żarliwych argumentów nie jest w stanie dokonać tego, co prosty zbiór wykresów i liczb. Co więcej, plan projektu jest ważny nie tylko na początku prac. Kiedy prace się już rozpoczną, narzędzi planu projektu można używać do przekazywania kierownictwu efektów i zauważalnych zmian. W dodatku A przedstawiam zagadnienia związane ze śledzeniem projektu i zarządzaniem zmianami.

Oprócz komunikowania projektu programistom i menadżerom *metoda* pozwala architektom na precyzyjne i proste prezentowanie projektu innym architektom. Pomysły, jakie można zyskać dzięki takim opiniom i konstruktywnej krytyce, są wprost nieocenione.

## Czym metoda nie jest?

W 1987 roku Brooks napisał, że „nie istnieje żadna srebrna kula”<sup>4</sup>. Bez wątplenia nie jest nią także *metoda*. Stosowanie *metody* nie gwarantuje sukcesu i może pogorszyć sytuację, jeśli będzie używana w izolacji lub jedynie dla samego faktu jej użycia.

*Metoda* nie odbiera architektom kreatywności ani nie zwalnia ich z konieczności włożenia wysiłku w stworzenie dobrej architektury. To wciąż architekt będzie ponosił odpowiedzialność za błędy w projektowanej architekturze oraz za nieumiejętność przekazania informacji o niej programistom i przeprowadzenia prac programistycznych w sposób umożliwiający oddanie gotowego produktu bez odstępstw od zaplanowanej architektury, a wszystko to pod coraz większą presją. Co więcej, zgodnie z wyjaśnieniami podanymi w drugiej części książki architekt jest odpowiedzialny za stworzenie wykonalnego planu projektu wynikającego z przyjętej architektury. Architekt musi dostosować projekt do dostępnych zasobów, do tego, co można stworzyć, dysponując określonymi zasobami, do ryzyka związanego z projektem oraz do wyznaczonego terminu. Przechodzenie przez wszystkie prace projektowe tylko dla nich samych jest pozbawione sensu. Architekt musi wyeliminować wszystkie uprzedzenia i stworzyć prawidłowy zestaw założeń do planu i wynikające z nich obliczenia.

*Metoda* stanowi dobry punkt początkowy do prac nad projektem systemu i planem projektu i zawiera także listę rzeczy, których należy unikać. Niemniej jednak będzie ona działać wyłącznie wówczas, gdy podejmiemy do niej rzetelnie oraz poświęcimy czas i wysiłek intelektualny, by zebrać wszystkie niezbędne informacje. Trzeba uważnie dbać o proces projektowy i jego efekty.

---

<sup>4</sup> Frederick P. Brooks, *No Silver Bullet: Essence and Accidents of Software Engineering*, „Computer”, 20, nr 4, kwiecień 1987.

# PROGRAM PARTNERSKI

— GRUPY HELION —

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

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

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

GRUPA  
**Helion** 



# MISTRZ W PROJEKTOWANIU OPROGRAMOWANIA

## ROZWAŻA BARDZO NIEWIELE OPCJI!



Tworzenie oprogramowania, zwłaszcza dużych i złożonych aplikacji, bywa problemem. Często objawy złego planowania projektu nie mają wiele wspólnego z uwarunkowaniami technicznymi: wysoki poziom stresu, duża rotacja pracowników, wypalenie zawodowe, brak zaufania, niska samoocena, a nawet różnego rodzaju dolegliwości fizyczne. Przyczyny tego stanu rzeczy są podobne: określone w nierealny sposób koszty, terminy i wymagania. Później okazuje się, że wewnętrznej złożoności systemu nikt nie rozumie, nad koniecznymi ciągłymi zmianami nikt nie panuje, a wytworzone oprogramowanie jest niemal niemożliwe do utrzymania. Jak widać, problem jest wielowymiarowy. Wielowymiarowe musi więc też być narzędzie, które umożliwi jego rozwiązanie.

W tym praktycznym przewodniku uniwersalne zasady projektowania zostały dostosowane do specyfiki wytwarzania oprogramowania. Znalazł się tu zbiór najważniejszych zasad inżynierii oprogramowania, jak również wyczerpujący zestaw narzędzi i technik do stosowania w projektach programistycznych. Ich zastosowanie sprawi, że gotowy system będzie łatwy do utrzymania, rozszerzalny, nie będzie zbyt kosztowny, a jego wykonanie będzie realne pod względem czasu i ryzyka. Koncepty opisane w książce stanowią doskonały punkt wyjścia, gdyż pozwalają sporządzić dobry projekt systemu oprogramowania oraz dobry plan budowy tego systemu. Dopiero to umożliwi ukończenie pracy w wyznaczonym terminie i bez przekroczenia założonego budżetu, a wytworzony produkt będzie łatwy w utrzymaniu, możliwy do rozszerzania oraz wielokrotnego użycia.

W książce między innymi:

- koncepcja projektowania systemu i planowania projektu
- dekompozycja systemu, jego struktura i łączenie komponentów
- narzędzia i techniki potrzebne w planowaniu i projektowaniu
- mierzenie i wyciąganie ryzyka projektu i jego wariantów
- zaawansowane techniki projektowania systemów o wysokiej złożoności



### JUVAL LÖWY,

jest światowej klasy architektem oprogramowania. Specjalizuje się w projektowaniu systemów i planowaniu projektów. Jest uznawany za jednego z najlepszych w swojej dziedzinie, uczestniczył

w tworzeniu takich produktów jak C#, WCF i związanych z nimi technologii. Wydał kilka best-sellerowych książek oraz opublikował niezliczone artykuły poświęcone niemal wszystkim aspektom nowoczesnego tworzenia oprogramowania.

**Helion**  
helion.pl  
HELION SA  
ul. Kościuszki 1c  
44-100 Gliwice  
tel.: 32 230 98 63  
helion@helion.pl

Sprawdź nasze szkolenia  
SZKOLENIA  
AKADEMIA IT & BUSINESS  
HELIONSZKOLENIA.PL

KOD KORZYŚCI  
Sięgnij po więcej!



ISBN 978-83-283-6793-7



9 788328 367937

**Pearson**  
Addison-Wesley

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 89,00 zł