



## Architektura oprogramowania w praktyce

Wydanie II

TWÓRZ DOSKONAŁE PROJEKTY ARCHITEKTONICZNE OPROGRAMOWANIA!

- Czym charakteryzuje się dobra architektura oprogramowania?
- Jak przebiega proces jej projektowania?
- Jak ją dokumentować?

Len Bass, Paul Clements, Rick Kazman

## » Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

## » Katalog książek

- Katalog online
- Zamów drukowany katalog

## » Twój koszyk

- Dodaj do koszyka

## » Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

## » Czytelnia

- Fragmenty książek online

## » Kontakt

Helion SA  
ul. Kościuszki 1c  
44-100 Gliwice  
tel. 32 230 98 63  
e-mail: helion@helion.pl  
© Helion 1991–2011

## Architektura oprogramowania w praktyce. Wydanie II

Autorzy: Len Bass, Paul Clements, Rick Kazman

Tłumaczenie: Paweł Koronkiewicz, Tomasz Walczak

ISBN: 978-83-246-3302-9

Tytuł oryginału: [Software Architecture in Practice, Second Edition](#)

Format: 172×245, stron: 464



### Twórz doskonałe projekty architektoniczne oprogramowania!

- Czym charakteryzuje się dobra architektura oprogramowania?
- Jak przebiega proces jej projektowania?
- Jak ją dokumentować?

Współczesne systemy informatyczne to zaawansowane, skomplikowane mechanizmy, składające się z wielu współdziałających ze sobą komponentów. Ich wyodrębnienie, a także określenie sposobu komunikacji i interakcji między poszczególnymi elementami, jest nie lada wyzwaniem dla architektów. Od ich decyzji zależy, czy system uda się zrealizować, czy będzie on efektywny, stabilny i łatwy w utrzymaniu.

Na szczęście istnieją metodologie, narzędzia oraz sposoby analizy efektów ułatwiające i porządkujące cały ten proces. W tej książce znajdziesz wszystko, o czym trzeba pamiętać przy projektowaniu oprogramowania. Poznasz sposoby projektowania z wykorzystaniem Metody Analizy Kompromisów w Architekturze (ATAM) oraz oceniania aspektów finansowych przy użyciu Metody Analizy Kosztów i Korzyści (CBAM). Autorzy przedstawią wiele studiów przypadków, które pozwolą Ci na zapoznanie się z rzeczywistymi problemami i ich rozwiązaniami. Ponadto nauczysz się stosować język UML do wizualnej reprezentacji architektury systemu oraz zobaczysz, jak przygotować dobrą dokumentację projektu. Książka ta sprawdzi się idealnie w rękach każdego architekta oprogramowania.

- Proces wytwarzania oprogramowania a cykl biznesowy architektury
- Wzorce architektury
- Struktury i perspektywy architektury
- Określenie i uzyskanie atrybutów jakościowych
- Projektowanie architektury pod kątem wysokiej dostępności
- Proces projektowania architektury
- Dokumentowanie architektury oprogramowania
- Język UML
- Metody rekonstrukcji architektury i inżynierii odwrotnej
- Metoda Analizy Kompromisów w Architekturze (ATAM)
- Metoda Analizy Kosztów i Korzyści (CBAM)
- Ponowne wykorzystanie elementów architektury
- Dokumentowanie architektury

**Poznaj najlepsze metodologie projektowania architektury!**

---

# Spis treści

<b>Przedmowa</b> .....	<b>9</b>
<b>Podziękowania</b> .....	<b>13</b>
<b>Wstęp</b> .....	<b>15</b>
<b>I. Wizja architektury</b> .....	<b>21</b>
<b>1. Cykl biznesowy architektury</b> .....	<b>23</b>
1.1. Skąd się biorą architektury? .....	26
1.2. Proces wytwarzania oprogramowania a cykl biznesowy architektury .....	31
1.3. Czym się charakteryzuje dobra architektura? .....	33
1.4. Podsumowanie .....	35
1.5. Pytania do dyskusji .....	35
<b>2. Czym jest architektura oprogramowania?</b> .....	<b>37</b>
2.1. Czym jest, a czym nie jest architektura oprogramowania? .....	37
2.2. Inne perspektywy .....	40
2.3. Wzorce architektury, modele referencyjne i architektury referencyjne .....	41
2.4. Dlaczego architektura jest tak ważna? .....	43
2.5. Struktury i perspektywy architektury .....	50
2.6. Podsumowanie .....	56
2.7. Literatura .....	57
2.8. Pytania do dyskusji .....	59
<b>3. System awioniki A-7E — studium wykorzystania struktur architektury</b> .....	<b>61</b>
3.1. Położenie w cyklu biznesowym architektury .....	62
3.2. Wymagania i atrybuty jakościowe .....	62
3.3. Architektura systemu awioniki A-7E .....	67
3.4. Podsumowanie .....	78
3.5. Literatura .....	79
3.6. Pytania do dyskusji .....	79

<b>II. Tworzenie architektury .....</b>	<b>81</b>
<b>4. Atrybuty jakościowe .....</b>	<b>83</b>
4.1. Architektura a funkcje systemu .....	84
4.2. Architektura a atrybuty jakościowe .....	84
4.3. Atrybuty jakościowe systemu .....	85
4.4. Scenariusze atrybutów jakościowych w praktyce .....	89
4.5. Inne atrybuty jakościowe systemu .....	103
4.6. Biznesowe atrybuty jakościowe .....	103
4.7. Atrybuty jakościowe architektury .....	104
4.8. Podsumowanie .....	105
4.9. Literatura .....	105
4.10. Pytania do dyskusji .....	106
<b>5. Uzyskiwanie atrybutów jakościowych .....</b>	<b>107</b>
5.1. Taktyki atrybutów jakościowych .....	107
5.2. Taktyki dostępności .....	109
5.3. Taktyki modyfikowalności .....	112
5.4. Taktyki wydajności .....	118
5.5. Taktyki bezpieczeństwa .....	122
5.6. Taktyki testowalności .....	124
5.7. Taktyki funkcjonalności .....	126
5.8. Taktyki atrybutów jakościowych a wzorce architektury .....	128
5.9. Wzorce i style architektury .....	129
5.10. Podsumowanie .....	130
5.11. Pytania do dyskusji .....	131
5.12. Literatura .....	131
<b>6. Kontrola ruchu lotniczego</b>	
— projektowanie pod kątem wysokiej dostępności .....	<b>133</b>
6.1. Powiązania w cyklu biznesowym architektury .....	135
6.2. Wymagania i atrybuty jakościowe .....	135
6.3. Architektura systemu .....	138
6.4. Podsumowanie .....	152
6.5. Literatura .....	152
6.6. Pytania do dyskusji .....	153
<b>7. Projektowanie architektury .....</b>	<b>155</b>
7.1. Architektura w cyklu życia oprogramowania .....	155
7.2. Projektowanie architektury .....	157
7.3. Kształtowanie struktury zespołów .....	167
7.4. Tworzenie systemu szkieletowego .....	170
7.5. Podsumowanie .....	171
7.6. Literatura .....	172
7.7. Pytania do dyskusji .....	173
<b>8. Symulator lotniczy — architektura</b>	
<b>ukierunkowana na łatwość integracji .....</b>	<b>175</b>
8.1. Powiązania w cyklu biznesowym architektury .....	176
8.2. Wymagania funkcjonalne i jakościowe .....	177
8.3. Architektura .....	180

8.4. Podsumowanie .....	193
8.5. Literatura .....	195
8.6. Pytania do dyskusji .....	195
<b>9. Dokumentacja architektury oprogramowania .....</b>	<b>197</b>
9.1. Funkcje dokumentacji .....	198
9.2. Perspektywy architektury .....	200
9.3. Wybieranie perspektyw architektury .....	201
9.4. Opisywanie perspektywy architektury .....	202
9.5. Ogólna część dokumentacji .....	208
9.6. Zunifikowany język modelowania — UML .....	211
9.7. Podsumowanie .....	220
9.8. Literatura .....	221
9.9. Pytania do dyskusji .....	221
<b>10. Rekonstrukcja architektury oprogramowania .....</b>	<b>223</b>
10.1. Wprowadzenie .....	223
10.2. Ekstrakcja informacji .....	226
10.3. Budowanie bazy danych .....	228
10.4. Scalanie informacji .....	230
10.5. Rekonstrukcja .....	232
10.6. Przykład .....	237
10.7. Podsumowanie .....	245
10.8. Literatura .....	245
10.9. Pytania do dyskusji .....	246
<b>III. Analiza i weryfikacja architektury .....</b>	<b>247</b>
<b>11. ATAM — kompleksowa metoda analizy architektury .....</b>	<b>253</b>
11.1. Uczestnicy procesu ATAM .....	253
11.2. Materiały wyjściowe procesu ATAM .....	255
11.3. Fazy procesu ATAM .....	256
11.4. Studium przypadku — weryfikacja metodą ATAM systemu Nightingale .....	267
11.5. Podsumowanie .....	281
11.6. Literatura .....	282
11.7. Pytania do dyskusji .....	282
<b>12. CBAM — ilościowe podejście do decyzji konstrukcyjnych .....</b>	<b>283</b>
12.1. Kontekst podejmowania decyzji .....	284
12.2. Podstawy metody CBAM .....	285
12.3. Stosowanie metody CBAM .....	289
12.4. Studium przypadku — projekt ECS w agencji NASA .....	291
12.5. Rezultaty analizy CBAM .....	298
12.6. Podsumowanie .....	299
12.7. Literatura .....	299
12.8. Pytania do dyskusji .....	299
<b>13. Współdziałanie w World Wide Web — studium przypadku .....</b>	<b>301</b>
13.1. Powiązania z cyklem biznesowym architektury .....	301
13.2. Wymagania funkcjonalne i atrybuty jakościowe .....	303
13.3. Architektura .....	307
13.4. Nowy cykl ABC — ewolucja architektur handlu elektronicznego w WWW .....	313

13.5. Wymagania jakościowe .....	317
13.6. Współczesny cykl biznesowy architektury .....	318
13.7. Podsumowanie .....	319
13.8. Literatura .....	320
13.9. Pytania do dyskusji .....	321

## **IV. Od jednego systemu do wielu ..... 323**

### **14. Rodziny produktów — ponowne użycie elementów architektury .....325**

14.1. Wprowadzenie .....	325
14.2. Co sprawia, że linia oprogramowania jest udana? .....	326
14.3. Określanie zakresu .....	328
14.4. Architektury linii produktów .....	331
14.5. Co sprawia, że rozwijanie linii oprogramowania jest trudne? .....	334
14.6. Podsumowanie .....	337
14.7. Literatura .....	337
14.8. Pytania do dyskusji .....	337

### **15. CelsiusTech — studium przypadku rodziny produktów .....339**

15.1. Związki z cyklem ABC .....	339
15.2. Wymagania i atrybuty jakościowe .....	355
15.3. Rozwiązanie architektoniczne .....	357
15.4. Podsumowanie .....	364
15.5. Literatura .....	365
15.6. Pytania do dyskusji .....	365

### **16. J2EE/EJB. Studium przypadku — standardowa dla branży infrastruktura obliczeniowa .....367**

16.1. Związki z cyklem biznesowym architektury .....	368
16.2. Wymagania i atrybuty jakościowe .....	368
16.3. Rozwiązanie architektoniczne .....	371
16.4. Decyzje związane z wdrażaniem systemu .....	383
16.5. Podsumowanie .....	388
16.6. Literatura .....	388
16.7. Pytania do dyskusji .....	388

### **17. Architektura Luther. Studium przypadku — aplikacje przenośne oparte na J2EE .....389**

17.1. Związki z cyklem ABC .....	390
17.2. Wymagania i atrybuty jakościowe .....	393
17.3. Rozwiązanie architektoniczne .....	396
17.4. Jak w architekturze Luther zrealizowano cele z obszaru jakości? .....	410
17.5. Podsumowanie .....	410
17.6. Literatura .....	411
17.7. Pytania do dyskusji .....	411

### **18. Budowanie systemów z gotowych komponentów .....413**

18.1. Wpływ komponentów na architekturę .....	415
18.2. Niedopasowanie architektury .....	416
18.3. Budowa z gotowych komponentów jako proces poszukiwań .....	421

18.4. Przykład — system ASEILM .....	424
18.5. Podsumowanie .....	433
18.6. Literatura .....	433
<b>19. Przyszłość architektury oprogramowania .....</b>	<b>435</b>
19.1. Cykl biznesowy architektury .....	436
19.2. Budowa architektury .....	437
19.3. Architektura w cyklu życia oprogramowania .....	438
19.4. Wpływ komponentów komercyjnych .....	439
19.5. Podsumowanie .....	441
<b>Skróty .....</b>	<b>443</b>
<b>Bibliografia .....</b>	<b>449</b>
<b>Skorowidz .....</b>	<b>455</b>

# Uzyskiwanie atrybutów jakościowych

Felix Bachmann, Mark Klein i Bill Wood<sup>1</sup>

*W pewnym stężeniu każda dobra jakość staje się szkodliwa.*

— Ralph Waldo Emerson

W rozdziale 4. omówiliśmy różne atrybuty jakościowe systemu. Naszym podstawowym narzędziem były scenariusze. Dokładne zrozumienie znaczenia każdego z atrybutów pozwala formułować praktyczne wymagania jakościowe. Wciąż nie jest to jednak rozwiązaniem problemu tego, jak zapewnić, by system faktycznie posiadał tę czy inną cechę. Temu właśnie poświęcony jest niniejszy rozdział. Dla każdego z sześciu atrybutów jakościowych opisywanych w rozdziale 4. przedstawimy praktyczne wskazówki, które mogą być istotną pomocą w odpowiednim konstruowaniu architektury. Nie próbujemy przy tym odnieść się do wszystkich możliwych cech systemu. Warto zwrócić uwagę, że analogiczne porady dotyczące zapewniania łatwości integracji zostaną przedstawione w rozdziale 8.

W tym rozdziale interesuje nas to, w jaki sposób architekt zapewnia uzyskanie różnych cech jakościowych. Wymagania jakościowe opisują reakcje oprogramowania niezbędne do realizacji celów biznesowych. W centrum naszego zainteresowania są techniki, z których może skorzystać architekt, by utworzyć odpowiedni projekt, stosując pewne wzorce konstrukcyjne, wzorce architektury i strategie architektury — będziemy nazywać je **taktykami atrybutów jakościowych**. Przykładowo celem biznesowym może być przygotowanie linii produktów. Środkiem do osiągnięcia tego celu jest zapewnienie wymienności pewnych klas funkcji.

Przed podjęciem decyzji o zastosowaniu pewnego systemu wzorców architekt powinien rozważyć pożądane taktyki modyfikowalności. Stają się one podstawą podejmowanych wyborów. Wzorzec lub strategia architektury to zbiór pewnych taktyk. Temat tego rozdziału to powiązania między wymaganiami jakościowymi (opisanymi w rozdziale 4.) a decyzjami dotyczącymi architektury.

## 5.1. Taktyki atrybutów jakościowych

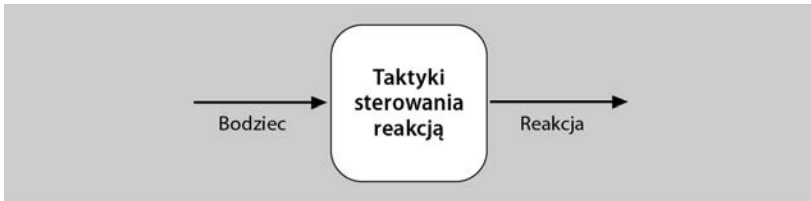
Co daje projektowi przenośność, dużą wydajność czy łatwość integracji? Każda z tych cech ma swoje źródło w kluczowych decyzjach konstrukcyjnych. W tym rozdziale przyjrzymy się dokładniej tym decyzjom. Będziemy operować pojęciem **taktyk atrybutów jakościowych** (ang.

<sup>1</sup> Pracownicy Instytutu Inżynierii Oprogramowania Carnegie Mellon University.



*quality attribute tactics*). Taktyka pewnej cechy to pewien wybór o znaczącym wpływie na kontrolę uzyskiwaną nad danym atrybutem. Zbiór taktyk nazywamy **strategią architektury** (ang. *architectural strategy*). Zajmiemy się nimi w rozdziale 12. Z kolei **wzorzec architektury** (ang. *architectural pattern*) to połączenie taktyk w sposób, który opiszemy dokładniej w podrozdziale 5.8.

Projekt systemu to zbiór decyzji konstrukcyjnych. Niektóre z nich pozwalają uzyskać wpływ na poziom, w jakim system uzyskuje pewne atrybuty jakościowe. Inne prowadzą do uzyskania funkcji systemu. W tym rozdziale sprowadzamy decyzje dotyczące atrybutów jakościowych do pojęcia taktyk. Ich rolę ilustruje rysunek 5.1. Taktyki te nie są bynajmniej nowością. Architekci stosują je od lat, a my podejmujemy jedynie próbę ich identyfikacji i opisu. Nie jest naszym celem prezentowanie nowych metod pracy, a jedynie systematyzacja działań znanych z praktyki.



**Rysunek 5.1.** *Taktyki wyznaczają kontrolę nad reakcją na bodziec*

Każda taktyka jest dla architekta pewną możliwością. Przykładowo jedna z taktyk wiąże się z wprowadzeniem redundancji w celu zwiększenia dostępności systemu. Jest to jedna z możliwości zwiększenia dostępności, ale w żadnym wypadku jedyna. Zwiększanie dostępności drogą redundancji wiąże się zazwyczaj z koniecznością wprowadzenia mechanizmów synchronizacji (by kopia nadawała się do użytku w przypadku zakłócenia dostępności oryginału). W tym prostym przykładzie widać dwie podstawowe zależności:

**1. Taktyka ogólniejsza podlega specjalizacji.** Zidentyfikowaliśmy ogólną taktykę redundancji. Można jednak mówić o nadmiarowości danych (w bazie) lub nadmiarowości obliczeń (w osadzonym systemie sterowania). Każde z tych podejść jest pewną taktyką. Każde z nich można uściślać dalej, definiując precyzyjniej określone taktyki. Dla każdego atrybutu jakościowego można wskazać pewną hierarchię taktyk.

**2. Wzorce to praktyczne zbiory taktyk.** Sięgnięcie po wzorzec zapewniający dostępność będzie prawdopodobnie równoznaczne z połączeniem zastosowania taktyki redundancji i taktyki synchronizacji. Co więcej, we wzorcu pojawiają się zapewne jeszcze precyzyjniej określone wersje tych taktyk. Na końcu tego rozdziału przedstawimy przykładowy opis wzorca projektowego w kategoriach taktyk.

Opis taktyk porządkujemy jako hierarchie powiązane z kolejnymi atrybutami jakościowymi. Należy pamiętać, że żadna z tych hierarchii nie jest kompletna. Możliwość stworzenia nowych taktyk istnieje zawsze, można więc jedynie mówić o pewnych przykładach. Dla każdego z sześciu atrybutów opisanych w rozdziale 4. (dostępność, modyfikowalność, wydajność, bezpieczeństwo, testowalność i funkcjonalność) przedstawiamy zbiór typowych rozwiązań. Dla każdego proponujemy pewną hierarchię taktyk, która — w połączeniu z krótkim omówieniem zagadnienia — powinna być dla architekta znaczącą pomocą w poszukiwaniu własnej ścieżki.

## 5.2. Taktyki dostępności

Przypomnijmy słownictwo związane z dostępnością, którym operowaliśmy w rozdziale 4. Awaria następuje wtedy, gdy system przestaje zapewniać usługę wynikającą z jego specyfikacji; jest to widoczne dla użytkowników. Awarię może spowodować uszkodzenie (lub pewne połączenie uszkodzeń). Ważnym aspektem dostępności jest przywrócenie funkcjonowania systemu, czyli jego naprawa. Taktyki opisywane w tym podrozdziale mają zabezpieczyć przed awarią systemu w przypadku uszkodzenia, a przynajmniej ograniczyć jego skutki i umożliwić naprawę. Ilustruje to rysunek 5.2.



Rysunek 5.2. Cel taktyk dostępności

Wiele omawianych taktyk zapewnianych jest przez standardowe środowiska wykonawcze, takie jak system operacyjny, serwer aplikacji lub system zarządzania bazami danych. Nie umniejsza to znaczenia dobrego zrozumienia stosowanej taktyki, ponieważ efekty każdej z nich mają istotne znaczenie przy projektowaniu i ocenie projektu. Każda technika związana z dostępnością wiąże się z pewnym rodzajem redundancji, monitorowaniem stanu w celu wykrycia awarii i schematem przywracania stanu systemu. Monitorowanie i przywracanie mogą być zautomatyzowane lub nie.

Rozpoczniemy od omówienia zagadnień związanych z trzema podstawowymi składnikami dostępności: **wykrywaniem uszkodzeń**, **przywracaniem stanu** i **zapobieganiem uszkodzeniom**.

### Wykrywanie uszkodzeń

Trzy szeroko stosowane taktyki wykrywania uszkodzeń to: ping/echo, puls i wyjątki.

- **Ping/echo.** Pewien komponent wysyła specjalny komunikat i oczekuje, że komponent monitorowany odeśle w predefiniowanym czasie „echo” jego sygnału. Mechanizm taki można zastosować w grupie komponentów wspólnie odpowiadających za pewne zadanie w systemie. Jest też często wykorzystywany przez klienty do monitorowania parametrów wydajnościowych serwera i ścieżki komunikacyjnej. Mechanizmy wykrywania uszkodzeń można łączyć w hierarchie, w których mechanizm najniższego poziomu monitoruje procesy pracujące na tym samym procesorze, a mechanizm na wyższym poziomie monitoruje stan kontrolerów na niższym poziomie. Pozwala to uniknąć obciążania kanału komunikacyjnego komunikacją zdalnego monitora z wszystkimi procesami.
- **Puls.** Komponent monitorowany wysyła regularnie komunikat pulsu do komponentu monitorującego. Gdy komunikat pulsu nie zostaje odebrany, następuje powiadomienie komponentu odpowiedzialnego za usunięcie uszkodzenia. Komunikaty pulsu mogą zawierać dane. Rolę pulsu może pełnić na przykład przesyłany okresowo dziennik pracy bankomatu. Komunikat taki jest jednocześnie komunikatem zawierającym przekazywane do przetwarzania dane.
- **Wyjątki.** Jedną z metod monitorowania uszkodzeń jest przechwytywanie wyjątków, jak te zgłaszane przy wystąpieniu błędów należących do klas wymienionych w przykładzie w rozdziale 4. Procedura obsługi wyjątku jest zazwyczaj wykonywana w tym samym procesie, w którym został on wygenerowany.

Taktyki ping/echo i pulsu bazują na odrębnych procesach, mechanizm wyjątków działa w obrębie jednego procesu. Procedura obsługi wyjątku przeprowadza zazwyczaj transformację semantyczną uszkodzenia do postaci umożliwiającej jego przetwarzanie.

## Przywracanie stanu systemu

Na przywracanie stanu systemu składa się przygotowanie do przywracania i właściwa naprawa. Poniżej przedstawiamy wybór taktyk przygotowania do przywracania i naprawy systemu.

- **Głosowanie.** Procesy pracujące na nadmiarowych procesorach przyjmują takie same dane wejściowe i obliczają prostą wartość wyjściową, która jest przekazywana do mechanizmu głosowania. Jeżeli mechanizm ten wykryje, że zachowanie jednego procesora odbiega od większości, proces ten zostaje uznany za uszkodzony. Przy głosowaniu może być stosowany algorytm „rządów większości”, „komponentu preferowanego” lub inny. Jest to metoda stosowana do wykrywania niepoprawnego działania algorytmów i uszkodzeń procesorów, często spotykana w systemach sterowania. Jeżeli wszystkie procesory pracują według tych samych algorytmów, redundancja pozwala wykryć jedynie uszkodzenie procesora, ale nie chroni przed błędami algorytmów. Jeżeli konsekwencje awarii są dotkliwe (na przykład utrata życia), zróżnicowanie nadmiarowych komponentów może być bardzo duże.

Skrajnym przykładem zróżnicowania jest przekazanie pracy nad każdym z nadmiarowych komponentów innemu zespołowi ze wskazaniem innej platformy. Nieco mniej wyszukany podejściem jest opracowywanie wersji tego samego składnika pracujących na różnych platformach. Wprowadzanie zróżnicowań tego rodzaju jest zawsze kosztowne — zarówno na początku, jak i przy późniejszej konserwacji systemu, więc stosuje się je tylko w wyjątkowych przypadkach, na przykład w mechanizmach awioniki. Mechanizm głosowania jest zazwyczaj stosowany w systemach sterowania, w których porównywane wyjścia są proste i łatwe w klasyfikacji jako zgodne lub nie, obliczenia mają charakter cykliczny, a wszystkie nadmiarowe komponenty mogą otrzymywać równoważne dane wejściowe z czujników. Taktyka ta nie wiąże się z przerwą w pracy, ponieważ system głosowania może działać także po wykryciu awarii. Znaną odmianą tego podejścia jest metoda Simplex, polegająca na korzystaniu z wyników komponentu „preferowanego”, dopóki nie odbiegają one od przesyłanych przez komponent „zaufany”. W przypadku awarii następuje przełączenie na dane z komponentu zaufanego. Synchronizacja nadmiarowych komponentów następuje automatycznie, ponieważ zakłada się, że wszystkie pracują jednocześnie z tym samym zespołem wejść.

- **Redundancja aktywna (gorący restart).** Wszystkie nadmiarowe komponenty reagują na zdarzenia jednocześnie. Są więc w tym samym stanie. Wykorzystywana jest odpowiedź jednego komponentu (zazwyczaj pierwszego, który odpowie), a pozostałe są odrzucane. W przypadku wystąpienia uszkodzeń przerwa w pracy systemów stosujących tę taktykę trwa zazwyczaj milisekundy, ponieważ komponent zapasowy jest aktywny, a jedyny potrzebny czas to czas przełączania. Redundancja aktywna to metoda często stosowana w konfiguracjach klient-serwer, takich jak systemy zarządzania bazami danych, gdzie szybkie reakcje na wystąpienie uszkodzenia są niezbędne. W systemie rozproszonym o wysokiej dostępności nadmiarowość może dotyczyć ścieżek komunikacyjnych. Przykładowo pożądane może być zastosowanie sieci LAN o wielu równoległych ścieżkach i umieszczenie każdego nadmiarowego komponentu na odrębnej ścieżce. Wówczas pojedyncza awaria mostu lub ścieżki nie powoduje, że wszystkie składniki systemu stają się niedostępne.

Synchronizacja polega na zapewnieniu, że wszystkie komunikaty przesyłane do komponentu z redundancją są przesyłane do wszystkich jego nadmiarowych wersji. Jeżeli komunikacja może ulec zakłóceniu (ze względu na przeciążone lub zawodne łącza), przywraca-

nie stanu może umożliwiać niezawodny protokół transmisji. Protokół taki wprowadza wymóg przesłania potwierdzenia przez wszystkich odbiorców. Towarzyszy temu pewna forma kontroli integralności komunikacji, na przykład suma kontrolna. Jeżeli nadawca nie może stwierdzić, że wszyscy odbiorcy otrzymali komunikat, to ponawia on próbę przesłania do komponentów, które nie potwierdzają odbioru. Ponowne wysyłanie nieodebranych komunikatów (czasem z użyciem różnych ścieżek komunikacji) jest kontynuowane do chwili, gdy algorytm nadawcy pozwoli uznać odbiorcę za wyłączony.

- **Redundancja pasywna (ciepły restart, podwójna redundancja, potrójna redundancja).** Jeden komponent (komponent główny) reaguje na zdarzenia i informuje pozostałe komponenty (komponenty awaryjne) o wymaganych aktualizacjach stanu. Gdy następuje awaria, system musi przede wszystkim zweryfikować, czy kopia zapasowa jest wystarczająco aktualna, by można było wznowić udostępnianie usług. To podejście także stosuje się w systemach sterowania, przede wszystkim wtedy, gdy dane wejściowe są pobierane z kanałów komunikacyjnych lub czujników, które muszą zostać przełączone w przypadku awarii na korzystanie z komponentów awaryjnych. W rozdziale 6., w którym jest omawiany przykład systemu kontroli ruchu lotniczego, spotkamy się z tą taktyką w praktyce. W tym przypadku komponent pomocniczy decyduje o tym, kiedy przejąć zadania komponentu głównego, ale często można spotkać się z tym, że za tę decyzję odpowiadają inne składniki. O tym, czy taktyka spełnia swoje zadanie, w dużej mierze decyduje zdolność komponentów awaryjnych do poprawnego przejścia pracy. Okresowe wymuszanie przełączania — na przykład raz na dobę lub raz na tydzień — zwiększa dostępność systemu. Niektóre systemy baz danych wymuszają przełączenie pamięci masowej dla każdego nowego elementu danych. Nowy element jest zapisywany w stronie *shadow*, a jej wcześniejsza wersja staje się kopią zapasową. W takich rozwiązaniach przerwa w pracy może zostać skrócona do kilku sekund.

Za synchronizację odpowiada komponent główny, który może zagwarantować ją poprzez atomowe emisje do komponentów awaryjnych.

- **Zapas.** Przygotowanie zapasowej platformy obliczeniowej przeznaczonej do całościowego zastępowania grupy zróżnicowanych komponentów. Po awarii musi ona zostać uruchomiona, a jej stan zainicjalizowany. Uzyskanie właściwego stanu zapasu umożliwia regularne zapisywanie punktów kontrolnych systemu i zmian stanu z użyciem urządzenia zapewniającego trwałe składowanie danych. Praktyczną formą zastosowania tej taktyki jest zapewnianie zapasowej stacji roboczej, z której może korzystać użytkownik w przypadku awarii jego głównego stanowiska pracy. Przerwy w pracy mierzy się zazwyczaj w minutach.

Niektóre taktyki uwzględniają wznowianie pracy komponentu — naprawiony po awarii komponent może zostać ponownie wprowadzony do systemu. Przykłady takich taktyk to: powielanie, resynchronizacja stanu i odwoływanie.

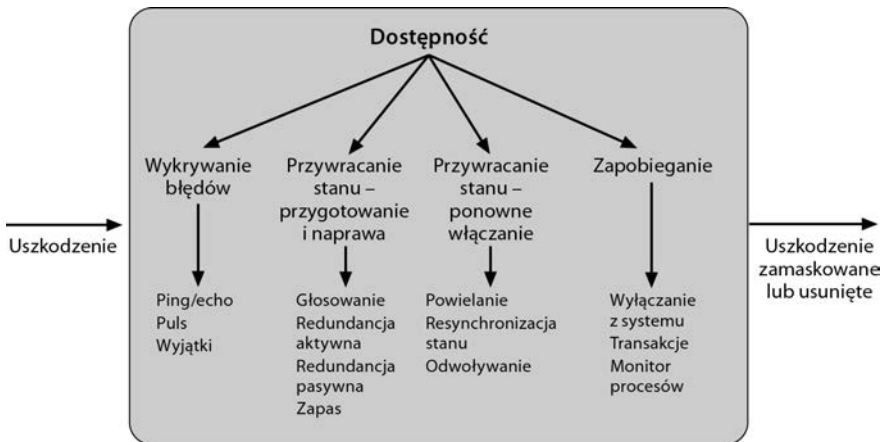
- **Powielanie** (ang. *shadow operation*). Komponent, który wcześniej uległ awarii, może przez pewien czas pracować w „trybie powielania” w celu weryfikacji, że jego działanie jest zgodne z działaniem komponentów, których praca nie uległa zakłóceniu.
- **Resynchronizacja stanu.** Taktyki redundancji aktywnej i pasywnej wymagają, by przywracany komponent został przed włączeniem do usługi odpowiednio zaktualizowany. Podejście do aktualizacji zależy od dopuszczalnego czasu przerwy w pracy, rozmiaru aktualizacji i liczby niezbędnych do jej przeprowadzenia komunikatów. O ile to możliwe, najlepszym rozwiązaniem jest pojedynczy komunikat opisujący stan. Przyrostowe aktualizowanie stanu łączone z okresami aktywności komponentu prowadzi do znacznego wzrostu złożoności.
- **Punkty kontrolne i odwoływanie** (ang. *rollback*). Punkt kontrolny to zapis spójnego stanu, który tworzy się okresowo lub w reakcji na pewne zdarzenia. Zdarza się, że system ulega nietypowej awarii i w zauważalny sposób traci integralność. W takiej sytuacji funkcjonowanie systemu można przywrócić, używając wcześniejszego punktu kontrolnego oraz dziennika transakcji wykonanych od czasu jego zapisania.

## Zapobieganie uszkodzeniom

Oto taktyki zapobiegania uszkodzeniom:

- **Wyłączanie z systemu.** Taktyka polegająca na usuwaniu z systemu wybranych komponentów w celu poddania ich pewnym działaniom mającym zapobiec typowym awariom. Przykładem może być reinicjalizowanie komponentu w celu zabezpieczenia systemu przed katastrofalnymi skutkami potencjalnych „wycieków” pamięci. Jeżeli wyłączanie z pracy następuje automatycznie, można zaprojektować odpowiednią strategię architektury. Jeżeli wyłączanie ma być ręczne, również należy zadbać o to w konstrukcji systemu.
- **Transakcje.** Transakcja to połączenie sekwencji kroków, które zapewnia, że sekwencja ta będzie mogła zostać wycofana jako całość. Transakcje wykorzystuje się w celu zabezpieczenia przed wpływem na jakiegokolwiek dane, w sytuacji gdy jeden z kroków procesu nie zostaje poprawnie wykonany, a także by zapobiegać kolizjom wątków korzystających z tych samych danych.
- **Monitor procesów.** Po wykryciu uszkodzenia w jednym z procesów proces monitorujący może go usunąć i utworzyć nowy o odpowiednio zainicjalizowanym stanie.

Rysunek 5.3 podsumowuje omówione taktyki.

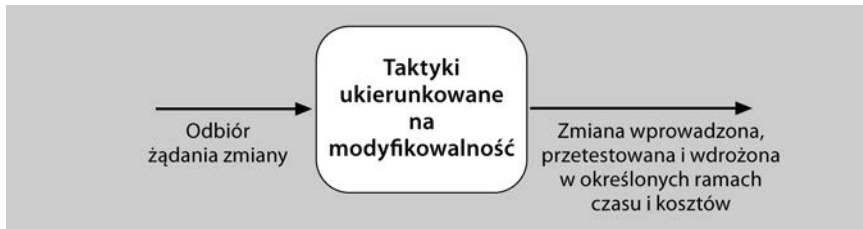


Rysunek 5.3. Taktyki dostępności

## 5.3. Taktyki modyfikowalności

Przypomnijmy z rozdziału 4. — taktyki modyfikowalności mają na celu uzyskanie wpływu na czas i koszty implementacji, testowania i wdrażania. Relację tę ilustruje rysunek 5.4.

Taktyki modyfikowalności łączymy w grupy według ich celów. Pierwszy zbiór łączy cel zmniejszenia liczby modułów, na które zmiana bezpośrednio wpływa. Jest to nazywane **lokalizowaniem modyfikacji**. Drugi zbiór taktyk ma na celu ograniczenie zakresu zmian w modułach, które nie mogą pozostać nienaruszone. Są to taktyki, które **zapobiegają efektowi domina** — powstawaniu łańcucha zależności, w którym zmiany w jednym miejscu prowadzą do zmian w innym. Operujemy przy tym istotnym rozróżnieniem na moduły, na które zmiana wpływa



Rysunek 5.4. Cel taktyk modyfikowalności

bezpośrednio (pewnej modyfikacji ulega ich zakres odpowiedzialności), i takie, na które zmiana wpływa pośrednio (zakres odpowiedzialności pozostaje, ale modyfikacja jest konieczna ze względu na moduły pod bezpośrednim wpływem zmiany). Trzecia grupa taktyk ukierunkowana jest na czas i koszty wdrażania. W tym przypadku chodzi przede wszystkim o **opóźnienie czasu wiązania**.

## Lokalizowanie modyfikacji

Choć nie można ogólnie mówić o ściślejszej relacji między liczbą modułów, na które wpływa pewien zespół zmian, a kosztami wprowadzenia tych zmian, nie ulega wątpliwości, że ograniczenie zakresu koniecznych modyfikacji do jak najmniejszej liczby modułów jest skuteczną metodą zmniejszania kosztów. Celem taktyk z opisywanej tu grupy jest uzyskanie takich przypisań zakresów odpowiedzialności modułów, by zakres przewidywanych zmian był jak najmniejszy. Prezentujemy pięć takich taktyk.

- **Utrzymywanie spójności semantycznej.** Spójność semantyczna odnosi się do relacji między zakresami odpowiedzialności w module. Celem jest zapewnienie, że wszystkie te zakresy współdziałają bez nadmiernego uzależnienia od innych modułów. Środkiem do osiągnięcia tego celu jest przestrzeganie w formułowaniu tych zakresów zasad spójności semantycznej. Są w tym pewną pomocą różne miary siły powiązań i poziomu spójności. Brakuje im jednak odniesienia do kontekstu zmian. Spójność semantyczną należy oceniać w perspektywie pewnego zbioru zmian przewidywanych w przyszłości. Szczególnie ważną taktyką w tej grupie jest **tworzenie abstrakcji wspólnych usług**. Zapewnianie wspólnych usług poprzez wprowadzenie wyspecjalizowanych modułów jest zazwyczaj postrzegane jako skuteczny środek ułatwiający ponowne wykorzystywanie elementów. Nie jest to jedyna korzyść — uogólnienie wspólnych usług sprzyja też modyfikowalności. Po wyłączeniu usług wprowadzane w nich modyfikacje nie muszą być powielane w każdym wykorzystującym je module. Co więcej, modyfikacje w modułach korzystających z tych usług nie mają wpływu na inne moduły, których praca opiera się na tych samych usługach. Jest to więc taktyka nie tylko sprzyjająca lokalizowaniu modyfikacji, ale też zabezpieczająca przed efektem domina. Przykłady wyłączenia wspólnych usług to między innymi stosowanie platform aplikacji (ang. *application frameworks*) i korzystanie z innego rodzaju oprogramowania *middleware*.
- **Przewidywanie prawdopodobnych zmian.** Przewidywanie zbioru oczekiwanych zmian pozwala ocenić zastosowany podział zakresów odpowiedzialności. Podstawowe pytanie brzmi: „Czy dla każdej ze zmian proponowana dekompozycja ogranicza zespół modułów, które muszą zostać zmodyfikowane w celu jej wprowadzenia?”. Towarzyszy mu również druga kwestia do rozważenia: „Czy zasadniczo różniące się zmiany wpływają na te same moduły?”. Czym różni się to od spójności semantycznej? Przypisywanie zakresów odpowiedzialności

na bazie spójności semantycznej wiąże się z założeniem, że również oczekiwane zmiany będą spójne semantycznie. Taktyka przewidywania zmian nie koncentruje się na spójności zadań modułu, ale na ograniczaniu skutków zmian. W praktyce jest ona trudna do zastosowania niezależnie od innych, ponieważ nie jest możliwe przewidzenie każdej zmiany. To powoduje, że zazwyczaj stosuje się ją w połączeniu z taktyką spójności semantycznej.

- **Uogólnienie modułu.** Zapewnienie modułowi większej ogólności prowadzi do uzyskania obsługi większej liczby funkcji. Dane wejściowe można traktować jak język definiujący moduł. Może to sprowadzać się do zastąpienia stałych parametrami. Może też prowadzić do rozwiązań bardziej złożonych, na przykład implementowania modułu w formie interpretera, kiedy to parametry wejściowe stają się jego językiem. Im bardziej ogólny moduł, tym bardziej prawdopodobne jest to, że wymagane zmiany będzie można wprowadzić drogą dostosowania języka na wejściu, bez modyfikowania kodu.
- **Ograniczanie dostępnych możliwości.** Zmiany, zwłaszcza w przypadku linii produktów (patrz rozdział 14.), mogą mieć bardzo szeroko zakrojone skutki i wpływać na wiele modułów. Ograniczanie zakresu dostępnych możliwości pozwala na redukcję wpływu modyfikacji. Przykładowo punkt zróżnicowania w projektowanej linii produktów może pozwalać na zmianę procesora — ograniczenie możliwości wymiany procesora do podzespołów należących do tej samej rodziny zmniejsza zakres dostępnych możliwości.

## Zapobieganie efektowi domina

O efekcie domina mówimy wtedy, gdy zmiana wymusza modyfikacje w modułach, na które nie ma bezpośredniego wpływu. Przykładowo moduł A zostaje zmodyfikowany w celu wprowadzenia pewnej zmiany, co prowadzi do konieczności modyfikacji modułu B, która wynika wyłącznie z tego, że zmienił się moduł A. Moduł B wymaga zmiany, ponieważ jest w taki czy inny sposób powiązany z modułem A.

Omówienie efektu domina zaczniemy od rozważenia różnych rodzajów zależności między modułami. Można wyróżnić osiem typów takiego powiązania:

### (1) Składnia:

- **danych** — aby moduł B był poprawnie kompilowany (lub wykonywany), typ (lub format) danych generowanych przez moduł A i pobieranych przez moduł B musi być zgodny z typem (lub formatem) danych, których oczekuje moduł B;
- **usług** — aby moduł B był poprawnie kompilowany i wykonywany, sygnatura usług zapewnianych przez moduł A i wywoływanych przez moduł B musi być zgodna z założeniami modułu B.

### (2) Semantyka:

- **danych** — aby moduł B był poprawnie wykonywany, semantyka danych generowanych przez moduł A i pobieranych przez moduł B musi być zgodna z założeniami modułu B;
- **usług** — aby moduł B był poprawnie wykonywany, semantyka usług zapewnianych przez moduł A i używanych przez moduł B musi być zgodna z założeniami modułu B.

### (3) Kolejność:

- **danych** — aby moduł B był poprawnie wykonywany, musi on otrzymywać dane generowane przez moduł A w ustalonej kolejności; na przykład nagłówki pakietu danych musi w trakcie odbierania poprzedzać właściwą treść (w przeciwieństwie do protokołów, które dołączają do danych numery sekwencyjne);

- **sterowania** — aby moduł B był poprawnie wykonywany, wcześniej, w określonych ramach czasowych, muszą zostać wykonane procedury modułu A. Przykładowo procedury modułu A muszą zostać wykonane nie dłużej niż 5 milisekund przed rozpoczęciem wykonywania modułu B.
- (4) **Tożsamość interfejsu modułu A** — moduł A może mieć wiele interfejsów. Aby moduł B był poprawnie kompilowany i wykonywany, tożsamość (nazwa lub uchwyt) interfejsu musi być zgodna z oczekiwaniami modułu B.
- (5) **Lokalizacja modułu A w czasie wykonywania** — aby moduł B był poprawnie wykonywany, lokalizacja modułu A w czasie wykonywania musi być zgodna z oczekiwaniami modułu B. Przykładowo moduł B może zakładać, że moduł A pracuje w innym procesie na tym samym procesorze.
- (6) **Jakość usług/danych zapewnianych przez moduł A** — aby moduł B był poprawnie wykonywany, pewne własności dotyczące danych lub usług zapewnianych przez moduł A muszą być zgodne z założeniami modułu B. Przykładowo dane dostarczane przez czujnik muszą mieć pewną dokładność, aby algorytmy w module B działały poprawnie.
- (7) **Istnienie modułu A** — aby moduł B był poprawnie wykonywany, moduł A musi istnieć. Jeżeli na przykład obiekt B żąda usługi obiektu A, a obiekt ten nie istnieje lub nie może zostać dynamicznie utworzony, działanie obiektu B nie będzie poprawne.
- (8) **Sposób korzystania z zasobów** — aby moduł B był poprawnie wykonywany, sposób korzystania z zasobów przez moduł A musi być zgodny z założeniami modułu B. Może to dotyczyć użytkowania zasobów (A używa tej samej pamięci co B) lub własności zasobów (B rezerwuje zasób, który A uważa za własny).

Dysponując tak sprecyzowanymi kategoriami zależności, możemy przejść do omówienia dostępnych architektowi taktyk pozwalających zapobiegać efektowi domina dla wybranych z nich.

Warto zwrócić uwagę, że żadna z prezentowanych taktyk nie zapewnia zabezpieczenia przed efektem domina dla zmian semantycznych. Rozpocznijmy omówienie od taktyk, które odnoszą się do interfejsów modułu — ukrywania informacji i utrzymywania istniejących już interfejsów, aby następnie przejść do taktyk przerywania łańcucha zależności — użycia pośrednika.

- **Ukrywanie informacji.** Ukrywanie informacji to dekompozycja zakresu odpowiedzialności pewnej jednostki (systemu lub części systemu) na mniejsze elementy i określanie, które z nich mają mieć charakter prywatny, a które publiczny. Publiczny zakres odpowiedzialności to zakres dostępny za pośrednictwem interfejsów. Celem jest doprowadzenie do izolacji zmian wewnątrz modułu i zapobieżenie ich propagacji na inne moduły. Jest to najstarsza technika ograniczania skutków zmian. Jest ona mocno powiązana z przewidywaniem oczekiwanych modyfikacji, ponieważ modyfikacje te stają się podstawą dekompozycji.
- **Zachowywanie istniejących interfejsów.** Jeżeli moduł B jest zależny od nazwy i sygnatury interfejsu modułu A, utrzymanie tego interfejsu i jego składni pozwala uniknąć zmian w module B. Oczywiście, nie jest to taktyka bezwzględnie skuteczna, gdy moduł B jest zależny od modułu A semantycznie, ponieważ warstwa znaczeniowa danych i usług jest trudniejsza do zamaskowania. To samo można powiedzieć o maskowaniu zależności bazujących na jakości danych lub usług, poziomie wykorzystania zasobów lub własności zasobów. Stabilność interfejsu można też osiągnąć oddzielając interfejs od implementacji. Pozwala to tworzyć abstrakcyjne interfejsy maskujące różnicowanie. Odmiany można realizować w ramach istniejącego zakresu odpowiedzialności lub przez zastępowanie implementacji.

Wzorce bazujące na tej taktyce to przede wszystkim:

- **dodawanie interfejsów** — większość języków programowania pozwala na stosowanie wielu interfejsów; nowe usługi lub dane mogą być udostępniane poprzez nowe interfejsy, dzięki czemu starsze pozostają niezmienione i zachowują sygnatury;



- **dodawanie adaptera** — polega na dodaniu do modułu A adaptera, który go osłania i zapewnia pierwotną sygnaturę;
- **korzystanie z wersji zastępczej** — jeżeli modyfikacja prowadzi do usunięcia modułu A, to zapewnienie jego wersji zastępczej pozwoli zachować moduł B w niezmienionej postaci (jeżeli moduł B jest zależny tylko od sygnatury modułu A).
- **Ograniczanie ścieżek komunikacji.** Taktyka polegająca na ograniczaniu liczby modułów, z którymi dany moduł wymienia lub współużytkuje dane — projektant dąży do uzyskania jak najmniejszej liczby modułów, które pobierają dane z modułu rozważanego lub przekazują je do niego. Efekt domina zostaje zredukowany, ponieważ generowanie i pobieranie danych odpowiada za wprowadzenie powodujących go zależności. Wzorzec korzystający z tej taktyki zostanie zaprezentowany w rozdziale 8. (symulator lotu).
- **Użycie pośrednika.** Jeżeli moduł A łączy z modulem B pewien rodzaj zależności inny niż semantyczna, to jest możliwe umieszczenie pośrednika między B i A, którego zadaniem jest zarządzanie czynnościami związanymi z tą zależnością. Nazw takich pośredników jest wiele. Tutaj omówimy poszczególne z nich w kategoriach wyliczonych wcześniej typów zależności. Podobnie jak wcześniej czarnym scenariuszem jest brak możliwości kompensacji zmian semantycznych.
  - **Dane (składnia).** Rolę pośrednika między producentem i konsumentem danych przyjmują magazyny danych (pasywne lub typu podręcznej tablicy). Niektóre wzorce publikowania-subskrypcji (w których dane przepływają przez pewien komponent centralny) mogą też zapewniać konwersję składni generowanej przez moduł A do tej, której oczekuje moduł B. Wzorce MVC i PAC zapewniają konwersję danych z jednej postaci (związanej z urządzeniem wejściowym lub wyjściowym) na inną (używaną w modelu w MVC lub w abstrakcji w PAC).
  - **Usługi (składnia).** Fasada, most, mediator, strategia, proxy i fabryka to wzorce wprowadzające pośrednika konwertującego składnię usługi. Każdy z nich prowadzi do zabezpieczenia przed propagacją zmian.
  - **Tożsamość interfejsu modułu A.** Wzorzec brokera pozwala zamaskować zmiany w tożsamości interfejsu. Jeżeli moduł B jest zależny od tożsamości interfejsu modułu A i tożsamość ta ulega zmianie, dodanie tożsamości do brokera i pozostawienie brokerowi tworzenia połączenia z nową tożsamością A pozwala pozostawić moduł B bez zmian.
  - **Lokalizacja modułu A w czasie wykonywania.** Serwer nazw pozwala zmieniać lokalizację A bez wpływu na B. A odpowiada za rejestrowanie swojej bieżącej lokalizacji na serwerze nazw, a B pobiera z niego aktualne informacje.
  - **Sposób korzystania z zasobów lub kontrola zasobu przez moduł A.** Menedżer zasobów to pośrednik odpowiedzialny za ich alokowanie. Istnieją menedżery gwarantujące zaspokojenie wszystkich żądań mieszczących się w określonych ograniczeniach (na przykład menedżery z przypisywaniem monotonicznym według częstotliwości, stosowane w systemach czasu rzeczywistego). Oczywiście, zastosowanie menedżera wiąże się z przejęciem przez niego kontroli nad zasobem.
  - **Istnienie modułu A.** Wzorzec fabryki daje możliwość tworzenia obiektów odpowiednio do potrzeb, dzięki czemu problem zależności obiektu B od istnienia obiektu A rozwiązuje specjalny moduł fabryki.

## Opóźnianie czasu wiązania

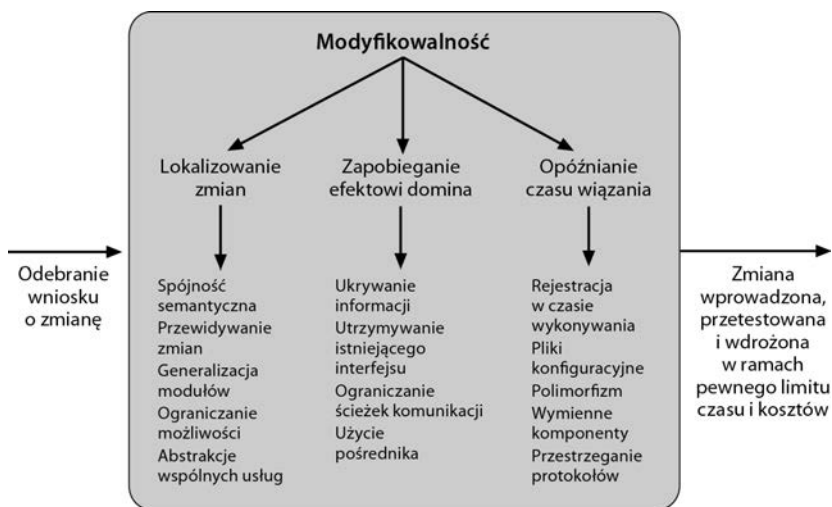
Omówione kategorie taktyk prowadzą do zmniejszenia liczby modułów wymagających modyfikacji w celu wprowadzenia zmiany. W naszych scenariuszach modyfikowalności pojawiły się jednak dwa elementy, których nie można sprowadzić do redukcji liczby modyfikowanych części programu — czas wdrożenia i umożliwienie wprowadzania zmian osobom innym niż programiści. Realizację tych scenariuszy umożliwia opóźnianie czasu wiązania. Z taktykami tego rodzaju niezmiennie wiąże się istotny koszt dodatkowej infrastruktury, która umożliwi późne wiązanie.

**Wiązanie** pewnego wyboru z działaniem systemu może nastąpić w różnych momentach. Tutaj zajmiemy się jedynie aspektami, które mają wpływ na czas wdrażania. Wdrażanie systemu jest pewnym procesem — po wprowadzeniu przez programistę zmiany następują zazwyczaj procedury testowania i dystrybucji. Występuje więc istotne opóźnienie między modyfikacją a dostępnością nowej wersji czy konfiguracji programu. Wprowadzenie wiązania w czasie wykonywania oznacza, że system został odpowiednio przygotowany i ukończono wszystkie kroki związane z testowaniem i dystrybucją. Taktyka opóźniania czasu wiązania prowadzi do powstawania mechanizmów umożliwiających użytkownikowi lub administratorowi operowanie parametrami, które zmieniają zachowanie systemu.

Wszystkie wymienione poniżej taktyki mają wpływ na system w czasie ładowania lub w czasie wykonywania.

- **Rejestracja w czasie wykonywania** umożliwia uzyskanie mechanizmu typu *plug-and-play* kosztem dodatkowego obciążenia operacjami związanymi z rejestrowaniem. Przykładem może być rejestracja typu publikacja-subskrypcja, którą można implementować w czasie wykonywania lub w czasie ładowania.
- **Pliki konfiguracyjne** określają parametry uruchomieniowe.
- **Polimorfizm** umożliwia późne wiązanie wywołań metod.
- **Wymiennosc komponentów** pozwala na wiązanie w czasie ładowania.
- **Przestrzeganie protokołów** pozwala wiązać w czasie wykonywania niezależne procesy.

Taktyki modyfikowalności podsumowuje rysunek 5.5.



Rysunek 5.5. Taktyki modyfikowalności

## 5.4. Taktyki wydajności

Przypomnijmy z rozdziału 4., że celem taktyk wydajności jest uzyskanie reakcji na odbierane przez system zdarzenie w ramach pewnych ograniczeń czasowych. Zdarzenie lub ciąg zdarzeń inicjuje pewne obliczenia lub funkcje. Sygnałem może być nadejście komunikatu, upływanie pewnego czasu, wykrycie znaczącej zmiany stanu w środowisku systemu itp. System przetwarza zdarzenia i generuje reakcję. Taktyki wydajności pozwalają uzyskać kontrolę nad ilością czasu, który musi upłynąć między zdarzeniem a reakcją. Ilustruje to rysunek 5.6. Czas między odebraniem zdarzenia a wygenerowaniem odpowiedzi określa się terminem latencja.



Rysunek 5.6. Cel taktyk wydajności

Po odebraniu zdarzenia następuje jego przetwarzanie. Może ono zostać z pewnych przyczyn zablokowane. Dwa podstawowe czynniki o dużym wpływie na czas odpowiedzi to: poziom wykorzystania zasobów i czas blokad.

**1. Poziom wykorzystania zasobów.** Zasoby to przede wszystkim procesor, magazyny danych, pasmo komunikacyjne sieci i pamięć, ale pojęcie to obejmuje też obiekty definiowane w ramach konstrukcji systemu. Przykładami mogą być wymagające zarządzania buforzy lub konieczność zapewnienia sekwencyjnego dostępu do sekcji krytycznych. Zdarzenia mogą mieć różny charakter i dla każdego typu zdarzenia wymagana jest pewna sekwencja przetwarzania. Przykładowo jeden komponent generuje komunikat, przekazuje go do sieci i komunikat ten zostaje odebrany przez inny komponent. Zostaje wówczas umieszczony w buforze; w pewien sposób przekształcony (tak zwany *marshalling* w terminologii OMG); przetworzony zgodnie z pewnym algorytmem; przesłany do wyjścia; umieszczony w buforze wyjściowym i przekazany dalej, do innego komponentu lub systemu albo do użytkownika. Każda z tych faz ma swój udział w ogólnej latencji przetwarzania zdarzenia.

**2. Czas blokad.** Procedury przetwarzania mogą nie uzyskać dostępu do zasobu — ze względu na jego nadmierne obciążenie, całkowitą niedostępność lub ponieważ niezbędny jest wynik innych procedur, który nie jest jeszcze dostępny.

- **Spory o zasoby.** Rysunek 5.6 przedstawia odbierane przez system zdarzenia. Odbiór może następować w pojedynczym strumieniu lub wielu strumieniach. Różne strumienie potrzebujące tego samego zasobu lub różne zdarzenia w tym samym strumieniu, które potrzebują tego samego zasobu, wprowadzają często istotne opóźnienia. Ogólnie — im więcej sporów o zasoby, tym większe prawdopodobieństwo wystąpienia dodatkowej latencji. Zależy to jednak od metod arbitrażu sporów i sposobu traktowania poszczególnych żądań.
- **Dostępność zasobów.** Nawet gdy nie występują spory o zasoby, obliczenia mogą zostać zablokowane przez ich niedostępność. Przyczyną może być na przykład wyłączenie lub awaria. Architekt powinien zawsze zidentyfikować miejsca, w których niedostępność zasobów może mieć znaczący wpływ na ogólną latencję reakcji.

- **Zależność od innych obliczeń.** Obliczenia mogą zostać wstrzymane przez konieczność synchronizacji z wynikami innych obliczeń lub oczekiwanie na wynik operacji składowych. Przykładowo w trakcie odczytu z dwóch różnych źródeł, gdy ich odczytywanie jest sekwencyjne, latencja jest większa niż przy równoległym odczycie z obu.

Na tym kończymy wprowadzenie i przechodzimy do omówienia trzech kategorii taktyk. Operują one w trzech obszarach: zapotrzebowania na zasoby, zarządzania zasobami i arbitrażu dostępu do zasobów.

## Zapotrzebowanie na zasoby

Strumienie zdarzeń stanowią źródło zapotrzebowania na zasoby. Dwa parametry opisujące to zapotrzebowanie to odstęp czasu między zdarzeniami (jak często w strumieniu pojawia się nowe żądanie) oraz stopień wykorzystania zasobu przez poszczególne żądania.

Jedną z taktyk zmniejszania latencji jest redukcja wymaganych do przetwarzania zdarzeń zasobów. Metody ograniczania zapotrzebowania na zasoby to między innymi:

- **Zwiększenie efektywności przetwarzania.** Jednym z kroków przetwarzania zdarzenia lub komunikatu jest użycie pewnego algorytmu. Usprawnianie algorytmów wykorzystywanych w krytycznych miejscach może doprowadzić do zmniejszenia latencji. Czasem można zmniejszyć poziom wykorzystania jednego zasobu kosztem innego. Przykładowo wartości pośrednie można przechowywać w repozytorium lub generować — zależy to od dostępnego miejsca i czasu. Jest to taktyka związana przede wszystkim z procesorem, ale można korzystać z niej także w odniesieniu do innych zasobów, na przykład dysku.
- **Zmniejszenie obciążenia.** Jeżeli nie nadchodzą żądania związane z pewnym zasobem, zapotrzebowanie maleje. W rozdziale 17. zobaczymy przykład wykorzystania klas języka Java w miejsce zdalnych wywołań metod (ang. *Remote Method Invocation* — RMI) — pozwala to zredukować wymagania dotyczące komunikacji. Korzystanie z pośredników (tak ważne dla modyfikowalności) zwiększa poziom wykorzystania zasobów w trakcie przetwarzania strumienia zdarzeń. Usunięcie pośrednika może więc być skuteczną metodą zmniejszenia latencji. Jest to typowy przykład kompromisu między modyfikowalnością a wydajnością.

Kolejna grupa taktyk redukcji latencji opiera się na zmniejszaniu liczby wymagających przetwarzania zdarzeń. Można to osiągnąć dwiema drogami:

- **Zarządzanie częstotliwością zdarzeń.** Jeżeli jest możliwe zredukowanie częstotliwości próbkowania danych przechowywanych w opisujących środowisko zmiennych, jest to prosta droga do zmniejszenia zapotrzebowania na zasoby. Czasem możliwość zmniejszenia częstotliwości zdarzeń jest skutkiem nadmiernej, niepotrzebnej złożoności. Niekiedy zawyżoną częstotliwość próbkowania uzasadnia potrzeba uzyskania zharmonizowanych odstępów dla różnych strumieni zdarzeń — niektóre z nich są odczytywane częściej jedynie dla zachowania synchronizacji z innymi.
- **Sterowanie częstotliwością próbkowania.** Jeżeli nie można uzyskać kontroli nad odbieraniem generowanych zewnętrznie zdarzeń, można próbować z mniejszą częstotliwością żądania przechowywane w kolejce — z uwzględnieniem ewentualnej utraty danych lub nie.

Jeszcze inną drogą do opanowania zapotrzebowania na zasoby jest sterowanie korzystaniem z tych zasobów.

- **Ograniczanie czasu wykonania.** Można określić limit czasu reakcji na zdarzenie. W przypadku algorytmów iteracyjnych, bazujących na danych taką metodą jest wyznaczenie limitu liczby powtórzeń.
- **Ograniczenie rozmiaru kolejki.** Jest to określenie maksymalnego rozmiaru kolejki odbioru, a więc bezpośrednie ograniczenie zasobów wykorzystywanych do przetwarzania zawartości tej kolejki.

## Zarządzanie zasobami

Nawet jeżeli zapotrzebowanie na zasoby nie może zostać poddane wystarczającej kontroli, skrócenie czasu reakcji można uzyskać przez odpowiednie zarządzanie tymi zasobami. Do takich taktyk należą:

- **Wprowadzenie współbieżności.** Jeżeli żądania mogą być przetwarzane równolegle, czas blokad może ulec skróceniu. Współbieżność można wprowadzić dwoma metodami: poprzez przetwarzanie różnych strumieni zdarzeń w odrębnych wątkach lub tworzenie wątków wykonujących pewne zbiory operacji. Po wprowadzeniu współbieżności ważne jest właściwe przypisywanie wątków do zasobów (równoważenie obciążenia).
- **Przechowywanie wielu kopii danych lub funkcji obliczeniowych.** W schemacie klient-serwer klienty są replikami funkcji obliczeniowych. Zapewniają one zmniejszenie rywalizacji o zasoby, która byłaby znaczącym problemem w przypadku wykonywania całości przetwarzania na serwerze. Metodą replikacji danych jest buforowanie (pamięć podręczna) — niezależnie od tego, czy szybkość pracy magazynów danych jest jednolita, czy różna, zmniejsza ono spory o dostęp do zasobów. Ponieważ buforowane dane to zazwyczaj kopia przechowywanych we właściwym magazynie danych, system obciąża dodatkową odpowiedzialność za utrzymanie spójności i synchronizacji pamięci podręcznej.
- **Zwiększenie dostępnych zasobów.** Szybsze procesory, dodatkowe procesory, dodatkowa pamięć, szybsza sieć — to proste rozwiązania o istotnym potencjale zmniejszania latencji. Wiąże się z nimi zazwyczaj problem kosztów. Do tego rodzaju kompromisu między kosztem a wydajnością powrócimy jeszcze w rozdziale 12.

## Arbitraż dostępu do zasobów

Zawsze, gdy pojawia się spór o pewien zasób, pojawia się też szeregowanie. Szereguje się dostęp do procesorów, buforów i sieci. Zadaniem architekta jest poznanie specyfiki wykorzystania każdego zasobu i wybranie dopasowanej do niego strategii szeregowania.

Schemat szeregowania składa się z dwóch elementów: przypisywania priorytetów i przydzielania zasobu. O określaniu priorytetów można mówić w każdym schemacie. Najprostsza reguła to FIFO (ang. *first-in/first-out* — pierwszy wchodzi, pierwszy wychodzi). O priorytecie może decydować limit czasu żądania lub wynikający z semantyki poziom ważności. Lista potencjalnych kryteriów — które z reguły konkurują ze sobą — obejmuje między innymi ważność żądania, dążenie do optymalnego wykorzystania zasobów, minimalizowanie liczby używanych zasobów, minimalizowanie latencji, maksymalizowanie przepustowości i dążenie do uniknięcia „zagłodzenia” (równość traktowania klientów). Architekt musi zdawać sobie sprawę z tego, które z nich są istotne i jaki wpływ ma na nie wybierana taktyka.

Strumień zdarzeń o wysokim priorytecie może zostać przydzielony tylko wtedy, gdy jest dostępny odpowiedni zasób. Czasem wymaga to wywłaszczenia bieżącego użytkownika zasobu. Można wybrać jeden z trzech schematów wywłaszczania: w dowolnym momencie, w ściśle określonych punktach lub bez wywłaszczania pracujących procesów. Jako popularne schematy szeregowania warto wymienić:

**1. FIFO (pierwszy wchodzi, pierwszy wychodzi).** Kolejka FIFO traktuje wszystkie żądania jednakowo i zaspokaja je kolejno. Podstawowym problemem jest w tym przypadku potencjalne opóźnienie przetwarzania żądania, które następuje po takim, które okazuje się wyjątkowo czasochłonne. Jeżeli komunikaty są faktycznie równorzędne, nie jest to zazwyczaj kłopotliwe. Jeżeli jednak w strumieniu są żądania ważniejsze od innych, metoda FIFO bywa niewystarczająca.

**2. Szeregowanie ze stałym priorytetem.** W tym schemacie każdemu źródłu żądań dostępu do zasobów przypisywany jest pewien priorytet, który decyduje następnie o kolejności przypisań. Zapewnia to lepszą obsługę żądań o wysokim priorytecie, ale dopuszcza możliwość, że żądanie o niskim priorytecie, ale wciąż istotne będzie oczekiwać na obsługę bardzo długo — jeżeli poprzedzi je seria żądań ze źródła uprzywilejowanego. Trzy popularne klucze przypisywania priorytetów to:

- **Waga semantyczna.** Strumienie otrzymują statyczne priorytety oparte na cechach zadań, z którymi są powiązane (cechach z dziedziny problemu). Takie podejście do szeregowania stosuje się w systemach *mainframe*, gdzie cechą z dziedziny problemu jest czas zainicjowania zadania.
- **Przypisywanie monotoniczne według limitu czasu.** Statyczna metoda przypisywania priorytetów, w której preferowane są strumienie o krótszych limitach czasu przetwarzania. Stosowana najczęściej przy szeregowaniu strumieni o różnych priorytetach powiązanych z limitami przetwarzania w czasie rzeczywistym.
- **Przypisywanie monotoniczne według częstotliwości.** Statyczna metoda przypisywania priorytetów strumieniom o charakterze okresowym, w której preferowane są strumienie o krótszym okresie. Jest to szczególny przypadek przypisywania według limitu czasu, a zarazem metoda bardziej znana i częściej dostępna w różnych systemach operacyjnych.

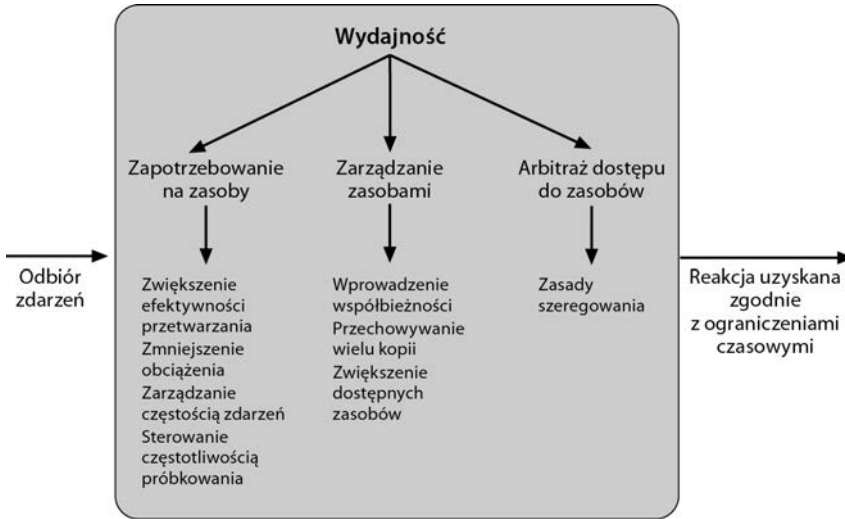
**3. Szeregowanie z dynamicznie określonym priorytetem.**

- **Karuzelowe.** Strategia polegająca na uporządkowaniu żądań, które następnie kolejno otrzymują zasoby — za każdym razem, gdy pojawia się możliwość przypisania żądania. Szczególną formą szeregowania karuzelowego jest zasada inicjowania nowych przypisań w stałych odstępach czasu.
- **Według terminów wykonania.** Najwyższy priorytet otrzymują żądania z najbliższym terminem wykonania.

**4. Szeregowanie statyczne.** Cykliczny mechanizm oparty na strategii, dla której punkty wywłaszczania i sekwencja przypisań zostały określone przed uruchomieniem systemu.

Lista polecanej literatury na końcu rozdziału zawiera tytuły książek poświęconych szeregowaniu.

Rysunek 5.7 przedstawia podsumowanie taktyk wydajności.



Rysunek 5.7. Podsumowanie taktyk wydajności

## 5.5. Taktyki bezpieczeństwa

Taktyki ukierunkowane na zapewnienie bezpieczeństwa operują w trzech obszarach: odpierania ataków, wykrywania ataków i przywracania stanu po ataku. Każdy z nich jest istotny. Można przytoczyć czytelną analogię — zamek w drzwiach to metoda obrony przed atakami, czujnik ruchu wewnątrz domu to system wykrywania ataków, a posiadanie ubezpieczenia to rozwiązanie umożliwiające przywrócenie stanu. Rysunek 5.8 ilustruje cele taktyk bezpieczeństwa.



Rysunek 5.8. Cel taktyk bezpieczeństwa

## Obrona przed atakami

Przedstawiona w rozdziale 4. lista celów opisujących bezpieczeństwo obejmuje niezaprzeczalność, poufność, integralność (spójność) i pewność tożsamości. W dążeniu do ich osiągnięcia pomocne jest łączenie poniższych taktyk:

- **Uwierzalnianie użytkowników.** Uwierzalnianie to sprawdzanie, czy użytkownik lub komputer jest tym, za kogo się podaje. Można stosować hasła, hasła jednorazowe, certyfikaty lub identyfikację biometryczną.

- **Autoryzowanie użytkowników.** Autoryzowanie to sprawdzanie, czy uwierzytelniony użytkownik ma prawa dostępu i modyfikacji danych lub usług. Jest to zazwyczaj zapewniane przez wprowadzenie w systemie pewnych schematów kontroli dostępu. Prawa dostępu przyznaje się użytkownikom lub klasom użytkowników. Definiowanie klas użytkowników może opierać się na pojęciu grupy bądź roli lub listach użytkowników.
- **Poufność danych.** Dane powinny być zabezpieczone przed nieautoryzowanym dostępem. Poufność uzyskuje się zazwyczaj przez wprowadzenie szyfrowania danych i połączeń komunikacyjnych. Szyfrowanie jest dodatkowym zabezpieczeniem danych trwałych, które w znaczący sposób uzupełnia mechanizmy autoryzowania dostępu. Co więcej, łącza komunikacyjne nie zapewniają zwykle autoryzacji. Szyfrowanie jest jedyną osłoną przy przekazywaniu danych przez łącza dostępne publicznie. Połączenie szyfrowane można implementować jako prywatną sieć wirtualną (ang. *virtual private network* — VPN) lub, w przypadku komunikacji WWW, jako warstwę gniazd z zabezpieczeniami, czyli SSL (ang. *Secure Sockets Layer*). Szyfrowanie może być symetryczne (obie strony używają tego samego klucza) lub niesymetryczne (klucze publiczny i prywatny).
- **Utrzymywanie integralności.** Dane powinny zostać dostarczone w postaci zgodnej z oczekiwaniami. Ich poprawność można weryfikować w oparciu o dołączane informacje nadmiarowe — takie jak sumy kontrolne i skróty — szyfrowane niezależnie od właściwych danych lub w połączeniu z nimi.
- **Redukcja narażenia na atak.** Podstawą ataku jest zazwyczaj możliwość wykorzystania jednego słabo chronionego punktu, który pozwala uzyskać dostęp do wszystkich danych i usług stacji. Architekt może tak zaprojektować przypisania komponentów do stacji, by każda z nich oferowała jak najwięcej zespołu usług.
- **Ograniczanie dostępu.** Zapory sieciowe ograniczają dostęp w oparciu o źródło komunikatu lub port docelowy. Komunikaty z nieznanymi źródłami mogą być formą ataku. Nie zawsze jednak można ograniczyć dostęp wyłącznie do znanych źródeł. Publiczna witryna WWW może oczekiwać żądań z dowolnych stacji sieciowych. Jedną z używanych w takich przypadkach konfiguracji jest tak zwana strefa zdemilitaryzowana (ang. *demilitarized zone* — DMZ). Konfiguracja DMZ daje dostęp do usług internetowych, ale nie do sieci lokalnej. Strefa zdemilitaryzowana to obszar funkcjonujący pomiędzy internetem a zaporą chroniącą sieć wewnętrzną. Umieszcza się w nim urządzenia, które powinny odbierać komunikaty z dowolnych źródeł. Pracują na nich głównie usługi WWW, pocztowe i DNS.

## Wykrywanie ataków

Wykrywanie ataku zapewnia zazwyczaj specjalny **system wykrywania włamań** (ang. *intrusion detection system*). Jego działanie opiera się na porównywaniu schematów ruchu sieciowego z wzorcami w bazie danych. Wykrywanie nadużyć oznacza porównywanie z wzorcami znanych ataków. Wykrywanie anomalii z kolei polega na porównywaniu bieżącego ruchu w sieci z zarejestrowanymi wcześniej schematami tego ruchu. Aby porównanie było możliwe, często konieczne jest filtrowanie pakietów. Filtrowanie może opierać się na protokole, znacznikach TCP, rozmiarach ładunku, adresie źródłowym, adresie docelowym lub numerze portu.

System wykrywania włamań musi dysponować pewnego rodzaju czujnikiem wykrywającym ataki, oprogramowaniem zarządzającym, które analizuje informacje i podejmuje odpowiednie czynności, bazą danych do przechowywania rejestru zdarzeń w celu późniejszej analizy, narzędziami do budowania analiz i raportów oraz konsolą sterującą, która umożliwia analitykowi modyfikowanie podejmowanych w przypadku włamania działań.



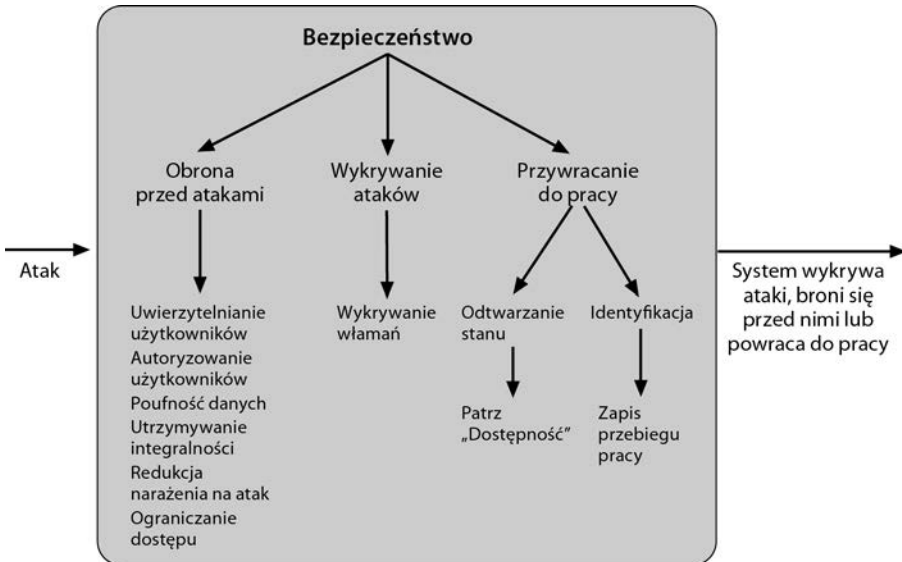
## Przywracanie do pracy po ataku

Taktyki wspomagające przywracanie stanu systemu po ataku można podzielić na bezpośrednio związane z przywróceniem normalnego funkcjonowania oraz ukierunkowane na identyfikację atakującego (jeżeli nie w celu ukarania, to po to, by zabezpieczyć się przed ponownymi atakami).

Taktyki przywracania systemu lub danych do właściwego stanu pokrywają się z taktykami zapewniającymi dostępność — w obu przypadkach chodzi o przywrócenie spójności. Główną różnicą jest konieczność szczególnego traktowania nadmiarowych kopii danych o charakterze administracyjnym, takich jak hasła, listy kontroli dostępu, konfiguracje usług nazw i dane profili użytkowników.

Podstawowa taktyka wspomagająca identyfikowanie atakujących to utrzymywanie **zapisu przebiegu zdarzeń** (ang. *audit trail*). Jest to kopia każdej transakcji na danych w systemie wraz z informacjami umożliwiającymi identyfikację. Informacje te można wykorzystywać do śledzenia działań atakującego, w realizacji zasady niezaprzeczalności (jest to dowód odebrania określonego żądania) lub przy operacjach przywracania stanu systemu. Dzienniki zdarzeń są często jednym z celów ataku, więc powinny być odpowiednio przechowywane.

Rysunek 5.9 przedstawia podsumowanie taktyk bezpieczeństwa.



Rysunek 5.9. Taktyki bezpieczeństwa

## 5.6. Taktyki testowalności

Celem taktyk testowalności jest umożliwienie łatwiejszego testowania, gdy kończona jest praca nad kolejną częścią oprogramowania. Rysunek 5.10 ilustruje zasadę ich stosowania. Budowie architektury pod kątem wysokiej testowalności oprogramowania nie poświęca się tyle uwagi co lepiej zbadanym aspektom jakości — modyfikowalności, wydajności czy dostępności, ale — jak pisaliśmy w rozdziale 4. — ponieważ testowanie odpowiada za dużą część kosztów systemu, to wszystko, co można zrobić w kierunku jego usprawnienia, przyniesie znaczące korzyści.



Rysunek 5.10. Cel taktyk testowalności

Choć w rozdziale 4. wspomnieliśmy o przeglądach projektu jako metodzie testowania, tutaj zajmujemy się jedynie testowaniem uruchomionego systemu. Celem procedur testowania jest wykrycie usterek. Wiąże się to z koniecznością zapewnienia dla testowanego oprogramowania danych wejściowych oraz mechanizmu przechwytywania danych wyjściowych.

Do uruchamiania testów jest potrzebne dodatkowe oprogramowanie przekazujące dane wejściowe i odbierające dane wyjściowe — jest to tak zwane **jarzmo testów** (ang. *test harness*). Nie będziemy tu szczegółowo rozpatrywać problemów związanych z jego konstrukcją i generowaniem. Czasem praca nad jarzmem testów jest znaczącą częścią pracy nad całym systemem.

Omówimy dwie kategorie taktyk wspomagających testowanie: związane z przekazywaniem i odbieraniem danych oraz związane z wewnętrznym monitorowaniem systemu.

## Wejście-wyjście

Można wyróżnić trzy taktyki zarządzania wejściami i wyjściami:

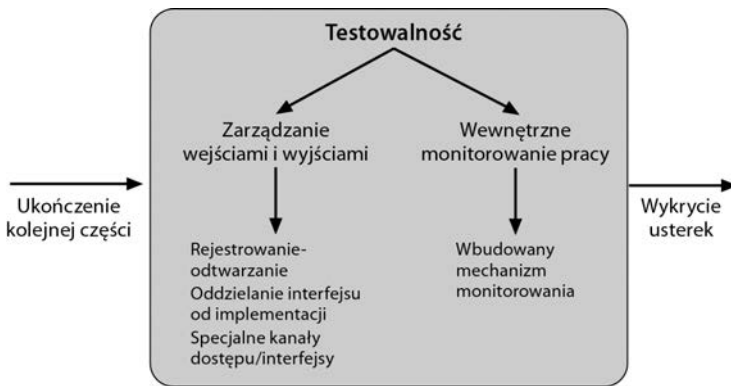
- **Rejestrowanie-odtworzenie.** Metoda, w której informacje przechodzące przez interfejs są przechwytywane i wykorzystywane jako dane dla jarzma testów. Informacje przechodzące przez interfejs w trakcie normalnej pracy zostają zapisane w pewnym magazynie — reprezentują one dane wyjściowe jednego komponentu, a zarazem dane wejściowe innego. Ich rejestrowanie rozwiązuje problem generowania danych wejściowych dla drugiego z tych komponentów oraz problem dostępności do porównań danych opisujących pracę pierwszego.
- **Oddzielanie interfejsu od implementacji.** Oddzielenie interfejsu od implementacji pozwala wymieniać implementacje na czas różnego rodzaju testów. Elementy zastępcze umożliwiają testowanie pozostałej części systemu, mimo że właściwe wersje tych elementów nie są jeszcze gotowe. Po wprowadzeniu ukończonych elementów wycofywany kod może zostać dołączony do jarzma testów.
- **Specjalne kanały dostępu/interfejsy.** Przygotowanie specjalnego interfejsu wspomagającego testowanie pozwala przechwytywać lub określać wartości zmiennych komponentu niezależnie od jego normalnej pracy. Przykładem może być udostępnienie specjalnego interfejsu dostępu do metadanych, wykorzystywanego w pracy jarzma testów. Interfejsy i kanały dostępu przeznaczone do tego rodzaju zastosowań powinny być wyraźnie oddzielone od tych, które wykorzystuje się w realizacji wyspecyfikowanych funkcji systemu. Wprowadzenie w architekturze hierarchii interfejsów służących do testowania prowadzi do sytuacji, kiedy testy mogą być uruchamiane na różnych poziomach oprogramowania, które dysponują ogólnymi mechanizmami umożliwiającymi badanie reakcji.

## Wewnętrzne monitorowanie pracy

Testowanie mogą wspomagać taktyki dotyczące wewnętrznych informacji o stanie:

- **Wbudowany mechanizm monitorowania.** Interfejs może udostępniać informacje o stanie, obciążeniu, dopuszczalnym obciążeniu, zabezpieczeniach i inne. Może to być trwały interfejs komponentu lub interfejs wprowadzany tymczasowo — przy użyciu takich metod, jak makra preprocesora lub programowanie aspektowe. Włączenie monitorowania oznacza zazwyczaj rozpoczęcie rejestrowania zdarzeń. Warto zwrócić uwagę, że korzystanie z takiego rozwiązania może doprowadzić do wydłużenia testów, ponieważ muszą one być powtarzane także po wyłączeniu śledzenia. Jednak korzyści z uwidocznienia przebiegu pracy komponentu bardzo często przewyższają koszty dodatkowego testowania.

Rysunek 5.11 przedstawia podsumowanie taktyk testowalności.



Rysunek 5.11. Taktyki testowalności

## 5.7. Taktyki funkcjonalności

Przypomnijmy z rozdziału 4., że pojęcie funkcjonalności odnosi się do tego, jak łatwo użytkownik może wykonywać swoje zadania oraz na jaką pomoc ze strony systemu może liczyć. Do zwiększenia funkcjonalności prowadzą dwie kategorie taktyk. Każda z nich dotyczy innej klasy „użytkowników”. Pierwsza kategoria, taktyki czasu wykonywania, polega na wspomaganiu działań użytkownika w trakcie pracy systemu. Druga wiąże się z iteracyjnym charakterem procedur projektowania interfejsu użytkownika. Jest to pomoc dla twórcy interfejsu w czasie projektowania. Jest ona zbliżona w swoim charakterze do omawianych wcześniej taktyk modyfikowalności.

Rysunek 5.12 przedstawia cel taktyk funkcjonalności dla czasu wykonywania.



Rysunek 5.12. Cel taktyk funkcjonalności dla czasu wykonywania

## Taktyki dla czasu wykonywania

W trakcie pracy systemu podstawowym mechanizmem sprzyjającym funkcjonalności jest przekazywanie użytkownikowi informacji zwrotnych, które opisują wykonywane przez system czynności i dają użytkownikowi możliwość korzystania ze specjalnych poleceń (o kilku wspomnieliśmy w rozdziale 4.). *Anuluj*, *Cofnij*, *Scal* i *Pokaż wiele widoków jednocześnie* to typowe funkcje wspierające użytkownika w korygowaniu błędów lub dążeniu do efektywnej pracy.

Badacze interakcji między człowiekiem a komputerem używają pojęć „inicjatywa użytkownika”, „inicjatywa systemu” i „inicjatywa mieszana” — wskazują one, która strona utrzymuje inicjatywę w trakcie wykonywania różnych czynności i decyduje o ich przebiegu. Scenariusze przedstawione w rozdziale 4. łączą obie perspektywy. Przykładowo w celu anulowania komendy użytkownik korzysta z polecenia *Anuluj*, co powoduje reakcję systemu — jest to inicjatywa użytkownika. Już w trakcie anulowania system może wyświetlić pasek postępu — jest to już inicjatywa systemu. Przy anulowaniu polecenia mamy więc do czynienia z inicjatywą mieszaną. Rozróżnienie to przydaje się przy omawianiu taktyk, które można zastosować w celu realizacji tego czy innego scenariusza.

Gdy inicjatywę podejmuje użytkownik, architekt projektuje reakcję, podobnie jak przy każdej innej funkcji systemu. Niezbędne jest określenie obowiązków systemu w zakresie reakcji na polecenie. Wracając do przykładu polecenia *Anuluj*: gdy użytkownik wydaje polecenie, system musi być gotowy do jego odbioru (odpowiada za zapewnienie obserwatora, który nie będzie zablokowany w związku z wykonywaniem anulowanych operacji); anulowana operacja musi zostać przerwana; zasoby używane przez tę operację muszą zostać zwolnione; wszystkie komponenty, których praca wiąże się z anulowaną operacją, muszą zostać poinformowane, by mogły podjąć odpowiednie w takiej sytuacji działania.

Gdy inicjatywę podejmuje system, podstawą są pewne informacje o użytkowniku, wykonywanym przez użytkownika zadaniu lub stanie samego systemu. Są to różne modele, z których każdy wymaga innego rodzaju danych wejściowych. Taktyki związane z inicjatywą systemu określają modele stosowane przez system do przewidywania albo własnych zachowań, albo zamierzeń użytkownika. Zahermetyzowanie takich informacji ułatwia architektowi dopracowywanie i modyfikowanie modeli. Dostosowanie i modyfikacja mogą być dynamiczne, oparte na zachowaniach użytkownika albo bardziej tradycyjne, wykonywane w trakcie budowy oprogramowania.

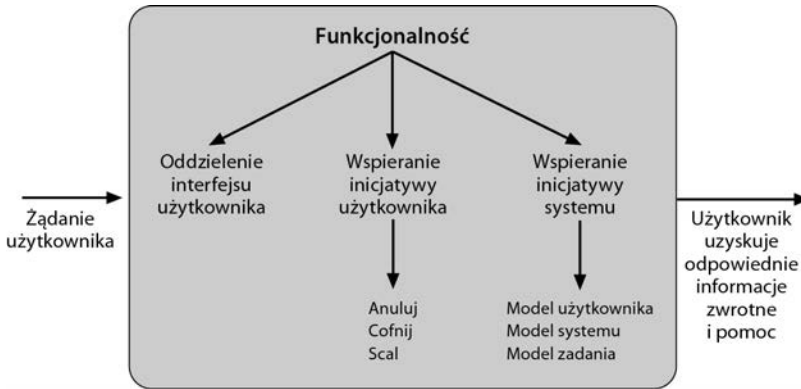
- **Utrzymywanie modelu zadania.** Model zadania służy do śledzenia kontekstu pracy w sposób umożliwiający systemowi określenie zamiarów użytkownika i uaktywnienie właściwych mechanizmów pomocy. Przykładowo wiedza o tym, że zdania rozpoczyna się zazwyczaj wielką literą, pozwala aplikacji korygować zdania, w których użytkownik nie dostosował się do tej reguły.
- **Utrzymywanie modelu użytkownika.** Model użytkownika służy do określania jego znajomości systemu, jego zachowań w kategoriach oczekiwanego czasu reakcji lub innych charakterystyk specyficznych dla pewnej osoby czy klasy osób. Przykładem zastosowania modelu użytkownika jest dopasowanie szybkości przewijania do szybkości czytania.
- **Utrzymywanie modelu systemu.** Model systemu służy do określania oczekiwanego zachowania systemu po to, by przekazać odpowiednie informacje zwrotne użytkownikowi. Jest to na przykład oszacowanie czasu potrzebnego do skończenia bieżącej operacji.

## Taktyki dla czasu projektowania

Procedury testowania wiążą się często z wieloma zmianami w interfejsie użytkownika — specjalista zajmujący się funkcjonalnością regularnie przekazuje programistom poprawki do bieżącego projektu interfejsu, a ci wprowadzają odpowiednie zmiany. W usprawnieniu tego schematu pomaga taktyka, która jest specjalizacją opisanej wcześniej taktyki spójności semantycznej:

- **Oddzielenie interfejsu użytkownika od reszty aplikacji.** Lokalizacja oczekiwanych zmian jest głównym uzasadnieniem spójności semantycznej. Ponieważ interfejs użytkownika ulega częstym zmianom zarówno przed wdrożeniem, jak i po nim, odseparowanie jego kodu jest pierwszym krokiem w kierunku lokalizacji. Można wymienić kilka wzorców architektury, które wykorzystują tę taktykę i ułatwiają modyfikowanie interfejsu:
  - Model-View-Controller (model-widok-kontroler),
  - Presentation-Abstraction-Control (prezentacja-abstrakcja-sterowanie),
  - Seeheim,
  - Arch/Slinky.

Rysunek 5.13 przedstawia podsumowanie taktyk funkcjonalności dla czasu wykonywania



Rysunek 5.13. Taktyki funkcjonalności dla czasu wykonywania

## 5.8. Taktyki atrybutów jakościowych a wzorce architektury

Przedstawiliśmy zbiór taktyk, których stosowanie pomaga architektowi uzyskać różnego rodzaju atrybuty jakościowe. W praktyce podejmowana decyzja konstrukcyjna to najczęściej wybór pewnego wzorca lub zbioru wzorców, które mają zapewnić użycie jednej lub większej liczby taktyk. Jednak każdy wzorec nieodmiennie wprowadza zbiór wielu taktyk — pożądanych lub nie. Niech ilustracją będzie analiza wzorca projektowego **Active Object** (aktywny obiekt), opisanego w [Schmidt 00]:

Wzorec Active Object usuwa ściśle powiązanie między wykonywaniem metody a jej wywołaniem, aby zwiększyć współbieżność i uprościć synchronizowany dostęp do obiektów we własnym wątku sterowania.

Wzorec składa się z sześciu elementów: **pośrednika**, który zapewnia interfejs pozwalający klientom wywoływać publicznie dostępne metody aktywnego obiektu; **żądania metody**, które definiuje interfejs wykonywania metod aktywnego obiektu; **listy aktywacyjnej**, przechowującej bufor oczekujących żądań metod; **jednostki szeregującej** decydującej o tym, które żądanie metody zostanie wykonane jako następne; **wykonawcy** definiującego zachowania i stan, modelowane przez aktywny obiekt; a także **przyszłej wartości**, która pozwala klientowi pobrać wynik wywołania metody.

Celem stosowania tego wzorca jest uzyskanie współbieżności. Jest to cel dotyczący wydajności. Architekt stosuje więc taktykę wydajności „wprowadzenie współbieżności”. Zwróćmy jednak uwagę na inne taktyki, które wprowadza wzorzec.

- **Ukrywanie informacji (modyfikowalność).** Każdy element ma pewien zakres odpowiedzialności, ale ukrywa sposób działania.
- **Pośrednik (modyfikowalność).** Pośrednik buforuje zmiany w sposobie wywoływania metod.
- **Czas wiązania (modyfikowalność).** Wzorzec aktywnego obiektu opiera się na założeniu, że żądania docierają do obiektu w czasie wykonywania. Kwestia czasu wiązania klienta z pośrednikiem pozostaje otwarta.
- **Zasady szeregowania (wydajność).** Jednostka szeregująca posługuje się pewnym zbiorem zasad.

Każdy wzorzec to połączenie taktyk, często powiązanych z różnymi atrybutami jakościowymi, i każda implementacja wzorca wiąże się z wyborami dotyczącymi tych taktyk. Przykładowo implementacja dziennika żądań może jednocześnie umożliwiać przywracanie systemu, zapewniać zapis przebiegu pracy i ułatwiać testowanie.

Architekt musi dobrze znać i rozumieć wprowadzone w konstrukcji systemu taktyki, a jego pracą jest poszukiwanie takiego ich połączenia, które pozwoli uzyskać wszystkie wymagane cechy systemu.

## 5.9. Wzorce i style architektury

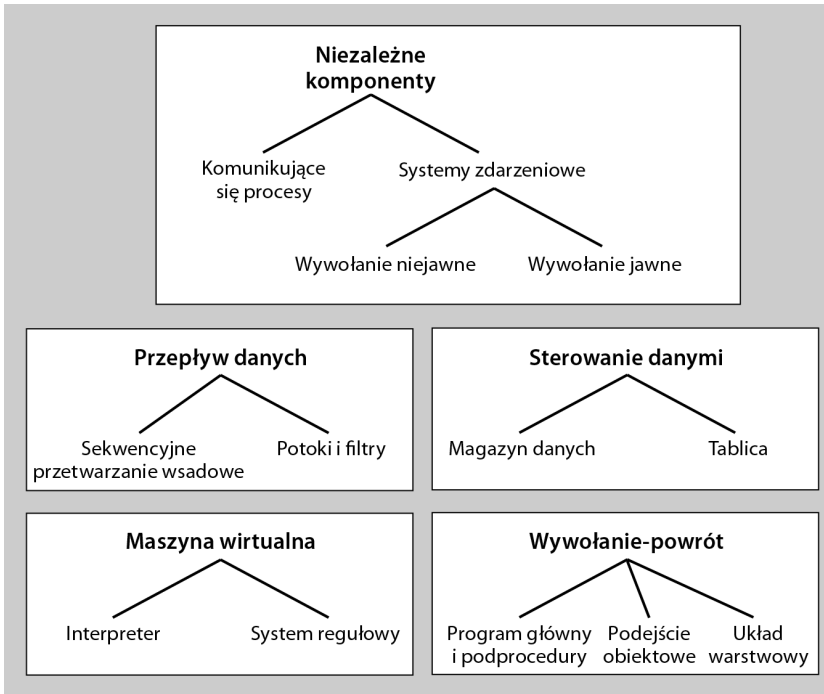
Wzorce architektury oprogramowania nazywa się czasem stylami architektury. Jest to trafna analogia ze stylami architektury w budownictwie, takimi jak gotyk, secesja czy barok. Styl buduje kilka istotnych elementów i reguł łączenia ich w sposób niezakłócający spójności. Wzorzec architektury jest określany przez:

- zespół typów elementów (takich jak magazyn danych lub komponent, który oblicza funkcję matematyczną),
- topologię elementów, która wyraża związki między nimi,
- zespół ograniczeń semantycznych (na przykład filtry w stylu z potokami i filtrami to czyste przetworniki danych — przekształcają one strumień wejściowy w strumień wyjściowy, ale nie mają kontroli nad elementami wcześniejszymi lub późniejszymi),
- zespół mechanizmów interakcji (jak wywołanie procedury, subskrypcja zdarzeń, tablica podręczna), które określają sposób koordynacji elementów w ramach dozwolonej topologii.

Mary Shaw i David Garlan podjęli znaczącą próbę skatalogowania zbiorów wzorców architektury, które nazwali **stylami** lub **idiomami**. W toku ewolucji inżynierii oprogramowania z podejścia tego wyłoniły się lepiej znane dzisiaj wzorce architektury, analogiczne do wzorców projektowych i wzorców pisania kodu.

Inspiracją dla autorów [Shaw 96] było spostrzeżenie, że istnieje wiele znanych wysokopoziomowych abstrakcji złożonych systemów, ale nie zostały one wcześniej systematycznie zbadane i skatalogowane, podobnie jak robi się w innych dziedzinach technicznych.

Poza bardzo regularnym występowaniem w projektach systemów drugą charakterystyczną cechą wzorców architektury jest to, że nie są one łatwo rozpoznawalne — przede wszystkim dlatego, że różne zastosowania prowadzą do różnego nazewnictwa. Odpowiedzią było powstanie katalogów zawierających opisy typowych wzorców, ich własności i znaczenia dla systemu. Przykład takiego katalogu widać na rysunku 5.14.



Rysunek 5.14. Mały katalog wzorców architektury uporządkowany według relacji „jest...”

Widoczne na rysunku wzorce zostały sklasyfikowane w grupy tworzące pewną hierarchię dziedziczenia. Na przykład system zdarzeniowy jest „podstylem” ogólniejszego stylu niezależności elementów. Z kolei wśród systemów zdarzeniowych można wyróżnić dwie podkategorie: z wywołaniem jawnym i z wywołaniem niejawnym.

Jaka jest relacja między wzorcami architektury a taktykami? Jak pisaliśmy wcześniej, traktujemy taktykę jako podstawowy „blok konstrukcyjny” projektu, z którego budowane są wzorce i strategie.

## 5.10. Podsumowanie

W tym rozdziale pokazaliśmy, w jaki sposób architekt osiąga wymagane przez specyfikację atrybuty jakościowe niezbędne do realizacji celów systemu. Zajmowaliśmy się **taktykami** stosowanymi przez architekta w celu skonstruowania właściwego projektu z użyciem różnych wzorców i strategii architektury.

Przedstawiliśmy listę szeroko znanych taktyk osiągania sześciu atrybutów omówionych w rozdziale 4.: dostępności, modyfikowalności, wydajności, bezpieczeństwa, testowalności i funkcjonalności. Taktyki podane dla każdego z nich są w powszechnym użyciu i łatwo zastosować je w praktyce.

Jak napisaliśmy, stosowanie wzorców architektury rozpoczyna się od wyboru pożądaných taktyk. W każdym projekcie pojawia się wiele taktyk i wiedza o tym, na które atrybuty wpływają one pozytywnie, jakie są ich skutki uboczne oraz czym grozi rezygnacja z innych, ma kluczowe znaczenie w pracy każdego architekta.

## 5.11. Pytania do dyskusji

- (1) Zastanów się nad popularną witryną WWW, taką jak na przykład Amazon lub eBay. Podobnie jak przy pytaniu 3. w rozdziale 4. utwórz kilka scenariuszy atrybutów jakościowych. Jakie taktyki powinieneś rozważyć przy wybieraniu wzorców i strategii architektury, aby ułatwić spełnienie zidentyfikowanych wymagań?
- (2) Dla zbioru taktyk utworzonego w odpowiedzi na pytanie 1.: jakich kompromisów z innymi atrybutami jakościowymi (na przykład bezpieczeństwem, dostępnością lub modyfikowalnością) możesz oczekiwać po zastosowaniu tych taktyk?
- (3) Funkcjonalności nie poświęca się często wystarczającej uwagi przy projektowaniu architektury. Takie traktowanie utrudnia osiągnięcie istotnych celów jakościowych. Zastanów się nad systemem, którego architekturę dobrze znasz, i spróbuj wyliczyć zastosowane w nim taktyki funkcjonalności.

## 5.12. Literatura

Więcej informacji o zabezpieczeniach systemu można znaleźć w [Ramachandran 02]. Związki między funkcjonalnością a wzorcami architektury oprogramowania są omawiane w [Bass 01a]. O technikach dostępności systemów rozproszonych traktuje [Jalote 94]. Z kolei [McGregor 01] jest dobrym źródłem informacji o zagadnieniach związanych z testowalnością.

Dwuczęściowy katalog wzorców architektury, [Buschmann 96] i [Schmidt 00], zawiera omówienie wzorców MVC i PAC (tom 1.) i koncepcji architektury opartej na wzorcach (tom 2.).

Opis wspomagającej uzyskiwanie wysokiej dostępności architektury Simplex można znaleźć na stronie <http://www.sei.cmu.edu/simplex/>.

W [Bachmann 02] omówienie stosowania taktyk jest podstawą do analizy modyfikowalności i wydajności, [Chretienne 95] zawiera opisy mechanizmów szeregowania, a w [Briand 99] można znaleźć opis miar siły powiązań modułów.

Pełną dokumentację wzorca Model-View-Controller można znaleźć w [Gamma 95], wzorca Presentation-Abstraction-Control — w [Buschmann 96], wzorca Seeheim — w [Pfaff 85], a Arch/Slinky — w [UIMS 92].



---

# Skorowidz

## A

- ABC, Patrz* cykl biznesowy architektury
- Active Object, 128
- adaptation data, *Patrz* dane adaptacji
- ADD, Patrz* metoda Projektowania według Atrybutów
- algorytm
  - komponentu preferowanego, 110
  - rządów większości, 110
- allocation structure, *Patrz* struktura alokacji
- analiza wymagań, *Patrz* wymagania analiza
- analizator
  - abstrakcyjnego drzewa składniowego, 227
  - leksykalny, 227
- aplikacja, 183, 335, 401
- application, *Patrz* aplikacja
- architectural
  - mismatch, *Patrz* architektura oprogramowania
  - niedopasowanie
  - pattern, *Patrz* architektura oprogramowania
  - wzorzec
  - strategy, *Patrz* architektura oprogramowania
  - strategia
- Architecture Business Cycle, *Patrz* cykl biznesowy architektury
- architecture reconstruction, *Patrz* architektura oprogramowania
- rekonstrukcja
- Architecture Tradeoff Analysis Method, *Patrz* metoda Analizy Kompromisów w Architekturze
- architekt, 25, 26, 29, 32, 34, 42, 50, 56, 101, 198, 223, 415, 437, 439
- architektura
  - J2EE, *Patrz* J2EE
  - systemowa, 49
  - architektura oprogramowania, 23, 25, 26, 29, 30, 34, 37, 38, 40, 45, 46, 47, 49, 57, 59, 61, 83, 84, 155, 169, 197, 357, 435
  - analiza, 33, 248
  - czynniki kształtujące, 156, 158, 159
  - diagram kontekstu, 203
  - dokumentacja, 39, 149, 167, 198, 202, 208, 210, 438
  - implementacja, 45
  - implementacja przyrostowa, 34
  - instrukcja różnicowania, 203
  - katalog elementów, 203
  - linia produktów, 331
  - ocena, 333
  - Luther, 389, 390, 391, 396, 400, 401, 402, 403, 409, 410
  - miary, 250
  - niedopasowanie, 49, 416
  - obiektowa, 435
  - ocena, 247, 250, 251, 252
  - metody, 250, 252
  - perspektywa, *Patrz* perspektywa
  - prezentacja, 32
  - projektowanie, 157, 438
  - przeglądy, 29
  - referencyjna, 42, 66
  - rekonstrukcja, 39, 199, 223, 225, 228, 233, 237
  - baza danych, 228, 229, 237
  - ekstrakcja informacji, 226, 237
  - identyfikacja wzorców, 232
  - interakcje, 232
  - konflikt nazw, 231
  - metoda warsztatowa, 224, 229
  - narzędzia, 224, 227
  - scalanie informacji, 230, 231, 238
  - wizualizacja, 232, 234, 235

architektura oprogramowania  
 reprezentacja, 39, 51  
 schemat główny, 203  
 słownik terminów, 204  
 strategia, 108, 287, 288, 291, 295, 298  
 struktura, 169, *Patrz* struktura  
 styl, *Patrz* architektura oprogramowania  
 wzorzec  
 tworzenie, 32, 33, 51  
 uzasadnienie, 204  
 warstwowa, 74  
 weryfikacja, *Patrz* architektura  
 oprogramowania ocena  
 wzorzec, 41, 42, 49, 108, 128, 129, 130, 157,  
 159, 183, 232, 420, 438  
 zgodność, 33  
 znaczenie, 43

artefakt, *Patrz* atrybuty jakościowe artefakt  
 AST, *Patrz* analizator abstrakcyjnego drzewa  
 składniowego

atak, 122  
 obrona, 122  
 przywracanie systemu, 124  
 wykrywanie, 123

ATAM, *Patrz* metoda Analizy Kompromisów  
 w Architekturze

atrybuty jakościowe, 32, 34, 42, 46, 58, 64, 67, 81,  
 83, 85, 128, 147, 199, 274, 275, 284, 285, 295, 299,  
 369, 370, 375, 383, 390, 393, 421, 425, 437  
 artefakt, 86, 87, 90, 92, 94, 96, 98, 101  
 bezpieczeństwo, 81, 85, 95, 122, 284, 313, 315  
 biznesowe, 103  
 bodziec, 86, 90, 92, 94, 96, 98, 100  
 czas projektowania, 127  
 dostępność, 81, 89, 90, 109, 124, 136, 138, 284,  
 313, 316, 317  
 funkcjonalność, 81, 84, 85, 99, 100, 102, 104,  
 126, 284  
 integrowalność, 194  
 kompletność, 105  
 łatwość testowania, 81  
 łatwość wprowadzania zmian, *Patrz* atrybuty  
 jakościowe modyfikowalność  
 miara reakcji, 86, 87, 91, 92, 94, 97, 99, 101  
 modyfikowalność, 61, 68, 81, 85, 88, 91, 112,  
 149, 160, 194, 195, 284, 313, 315, 317  
 niezawodność, 85  
 poprawność, 105  
 przenośność, 103  
 reakcja, 86, 87, 91, 92, 94, 96, 98, 101, 118  
 scenariusz, 86, 88, 89, 90, 92, 94, 96, 98, 100,  
 101, 157, 158, 165, 166  
 skalowalność, 103, 313, 316, 317, 395  
 spójność koncepcyjna, 104  
 środowisko, 86, 87, 90, 92, 94, 96, 98, 101, 135, 177

taktyka, 107, 109, 110, 112, 113, 115, 118, 119,  
 122, 124, 126, 130, 149, 157, 159, 437, 438  
 testowalność, 97, 124  
 wydajność, 81, 85, 93, 95, 104, 118, 136, 160,  
 178, 194, 284, 313, 315, 316, 317  
 wykonalność, 105  
 źródło bodźca, 86, 87, 90, 92, 94, 96, 98, 100

Attribute Driven Design, *Patrz* metoda  
 Projektowania według Atrybutów  
 availability, *Patrz* atrybuty jakościowe dostępność  
 awaria, 90, 109, 110, 143, 148

## B

bezpieczeństwo, *Patrz* atrybuty jakościowe  
 bezpieczeństwo  
 biblioteka WWW, *Patrz* libWWW  
 błąd, *Patrz* obsługa błędów  
 Bohr Niels, 435, 436  
 bridge, *Patrz* most  
 Brooks Fred, 57, 168, 441

## C

CBAM, *Patrz* metoda Analizy Kosztów i Korzyści  
 cele biznesowe, 156, 414  
 CGI, 311  
 choroba symulatorowa, 179  
 ciepły restart, *Patrz* redundancja pasywna  
 collaboration, *Patrz* system współdziałające zespoły  
 COM, 317  
 Common Object Request Broker Architecture,  
*Patrz* CORBA  
 component-and-connector structure, *Patrz*  
 struktura komponenty-złącza  
 CORBA, 317, 367, 368, 418  
 Cost Benefit Analysis Method, *Patrz* metoda  
 Analizy Kosztów i Korzyści  
 cross-side scripting, 431  
 cykl  
 życia oprogramowania, 438  
 biznesowy architektury, 25, 26, 30, 31, 35, 62,  
 135, 176, 247, 301, 313, 318, 339, 368, 411,  
 436, 437  
 czas reakcji, 26

## D

Dali, 225, 229, 232, 237  
 dane  
 adaptacji, 148, 149  
 trwałe, 53  
 współużytkowane, 53, 54  
 dekompozycja, 52, 84, 113, 149, 157, 159, 166, 190,  
 191, 192

deskryptor wdrożenia, 381, 383  
 diagram, *Patrz* schemat  
 Dijkstra Edsger, 58, 59, 141  
 dostępność, *Patrz* atrybuty jakościowe dostępność

## E

efekt domina, 112, 113, 114, 115, 160  
 EJB, 367, 371, 373

## F

filtr, 123, 129, 215, 217, 219, 240, 316  
 framework, *Patrz* platforma  
 FTP, 309  
 funkcja, 83, 84  
 funkcjonalność, *Patrz* atrybuty jakościowe funkcjonalność

## G

Garlan David, 49, 129  
 Glass Bob, 77  
 gorący restart, *Patrz* redundancja aktywna grupa  
   docelowa, 32, 198  
   interesu, 26, 28, 34, 35, 43, 101, 135, 198, 199, 201, 202, 255, 285, 339, 437

## H

hermetyzowanie informacji, *Patrz* ukrywanie informacji  
 Hybertsson Henryk, 24

## I

IDE, 317  
 idiom, *Patrz* architektura oprogramowania wzorzec implementacja, 84, 189  
   ograniczenia, 422, 427  
 instrumentacja kodu, 227  
 integrated development environment, *Patrz* IDE  
 interface, *Patrz* interfejs  
 interface mismatch, *Patrz* interfejs niedopasowanie  
 interfejs, 205, 377, 378, 394, 416  
   API, 397, 401, 402  
   komponentów, 215  
   negocjowany, 421  
   niedopasowanie, 416, 417, 419, 420  
   parametryzowany, 421  
   publiczny, 68  
   semantyka, 206  
   składnia, 206

szablon opisu, 206  
 użytkownika, 393, 396, 397, 399, 400, 401, 415  
 wariantowość, 208  
 interoperacyjność, 436  
 intrusion detection system, *Patrz* system wykrywania włamań  
 inżynieria odwrotna, 223  
 Iverson Ken, 57

## J

J2EE, 317, 367, 368, 371, 375, 388, 396, 403, 409, 411  
 J2EE/EJB, 367, 383, 388  
 jarzmo testów, 98, 125  
 Java, 368  
 Java 2 Enterprise Edition, *Patrz* J2EE  
 Język UML, *Patrz* UML

## K

kernel, 85, 145  
 klasa, 53, 215, 217, 403  
 kod  
   adaptacyjny, 421  
   korygujący, 417, 418  
 kompletność, *Patrz* atrybuty jakościowe kompletność  
 komponent, 51, 53, 55, 58, 110, 111, 118, 204, 215, 401, 402, 416  
   aktualizacja, 111  
   bezstanowy sesyjny, 374, 383, 384, 387, 408  
   EJB, 373, 377, 379, 381, 384, 387, 403  
   encyjny, 374, 375, 380, 384, 385, 386, 387, 407, 408  
   funkcji ogólnych, 402  
   komercyjny, 439  
   kompatybilność, 416  
   kwalifikacja, 419  
   pakiet, 403  
   podstawowy, 402  
   selekcja, 413  
   specyficzny dla dziedziny, 402  
   stanowy sesyjny, 374, 383, 384, 387  
   z funkcjami wielokrotnego użytku, 404  
   zarządzania organizacją pracy, 402, 404  
   zastępczy, 170  
 komputer ubraniowy, 390  
 konflikt, 34  
 Kruchten Philippe, 56

## L

latencja, 119  
 libWWW, 307, 309

linia oprogramowania, 325, 326, 328, 344  
 ewolucja, 335  
 narzędzia, 329  
 ocena, 333  
 trudności, 334  
 wdrażanie, 334  
 zakres, 328  
 zróżnicowanie, 331  
 linia produktów, 48, 59, 339, 341, 437  
 lokalizowanie modyfikacji, 112, 113, 160

## M

mapowanie, 54, 76, 144, 185, 189, 211, 218, 225, 234, 255, 431, 439  
 mechanizm wywołań niejawnych, 28  
 mediator, 417, 418, 421  
 menedżer  
 dostępu, 310  
 interfejsu użytkownika, 310  
 pamięci, 310  
 protokołów, 310  
 strumieni, 310, 311  
 metoda, 420  
 Analizy Kompromisów w Architekturze, 33, 47, 156, 199, 252, 253, 255, 267, 281, 283, 284, 285, 298  
 faza analizy, 257  
 faza procesu, 256, 268, 269  
 Analizy Kosztów i Korzyści, 252, 283, 285, 291, 298, 333, 437  
 fazy, 289, 292  
 podstawy, 285  
 ATAM, *Patrz* metoda Analizy Kompromisów w Architekturze  
 CBAM, *Patrz* metoda Analizy Kosztów i Korzyści  
 oceniania architektury przed rozpoczęciem budowy, 25  
 oparta na pomiarach, 250  
 oparta na pytaniach, 250  
 parametryzacji, 421  
 Projektowania według Atrybutów, 157, 158, 199  
 przyrostowej budowy oprogramowania, 25  
 Rational Unified Process, 157  
 Simplex, 110  
 warsztatowa, *Patrz* architektura oprogramowania  
 rekonstrukcja metoda warsztatowa  
 miara ilościowa, 34  
 miara reakcji, *Patrz* atrybuty jakościowe miara reakcji  
 middleware, 91, 113, 157, 170, 436  
 model  
 jednowątkowy, 309  
 programowania, 317  
 referencyjny, 42, 180  
 strukturalny, 175, 177, 180, 183, 186

modifiability, *Patrz* atrybuty jakościowe modyfikowalność  
 module structure, *Patrz* struktura modułów  
 moduł, 34, 51, 52, 113, 168, 212 *Patrz też* procedura brokera danych, 70  
 decyzji, 69  
 dekompozycja, 67, 68, 158, 159  
 filtrów, 70  
 generowania systemu, 71  
 generujący dane, 34, 37  
 interfejs, 34, 71, 164  
 kopia, 164  
 modeli fizycznych, 70  
 narzędziowy, 71  
 obsługi funkcji, 71  
 opis, 68  
 pobierający dane, 34  
 rozszerzający, 308  
 sekret, *Patrz* sekret typu danych aplikacji, 70  
 ukrywania warstwy sprzętowej, 69  
 ukrywania zachowań, 69  
 uogólnienie, 114  
 zależności, 114  
 modyfikowalność, *Patrz* atrybuty jakościowe modyfikowalność  
 most, 417, 418

## N

NET, 317, 367  
 niedopasowanie architektralne, *Patrz* architektura oprogramowania  
 niedopasowanie

## O

obiekty rozproszone, 28  
 obsługa błędów, 58, 72, 148  
 odwoływanie, 111  
 opóźnienie czasu wiązania, 113, 117  
 oprogramowanie  
 samokonfigurujące się, 421  
 wytwarzanie, 31  
 organizacja, 25, 392  
 cele, 25, 30  
 struktura, 27, 29, 46, 167, 169, 336, 348, 392  
 osłona, 417, 418, 421  
 osoby zainteresowane, *Patrz* grupa interesu

## P

parametryzacja, 421  
 Parnas David, 58, 65, 416, 441  
 parser, 227, 308

perspektywa, 51, 55, 162, 200, 201, 203, 229, 439  
 alokacji, 56, 201, 220  
 budowy oprogramowania, 56  
 dekompozycji, 140, 149, 162, 164, 201, 206, 310, 359  
 dynamiczna, 34  
 fizyczna, 56  
 implementacji, 201  
 katalog, 210  
 klient-serwer, 143, 149, 201  
 kodu, 56, 144  
 komponenty-złącza, 56, 162, 201, 202, 206, 215  
 koncepcyjna, 56  
 logiczna, 56  
 mapowanie, 210  
 modułów, 56, 164, 201, 212  
 odporności na błędy, 149  
 odporność na uszkodzenia, 147  
 powiązań modułów, 56  
 procesów, 55, 56, 141, 149, 202, 357  
 relacje, 149  
 rozmieszczenia, 55, 162, 163, 164, 201  
 statyczna, 34  
 szablon, 210  
 użycia, 201, 202, 206  
 warstwowa, 145, 201, 202, 359  
 współbieżności, 162, 164  
 wykonawcza, 56  
 PICS, 306  
 Platform for Internet Content Selection, *Patrz* PICS  
 platforma, 48, 50, 69, 73, 91, 103, 104, 110, 111, 113, 171, 175, 180, 195, 241, 303, 304, 308, 309, 323, 353, 355, 357, 359, 368, 370, 435  
 poprawność, *Patrz* atrybuty jakościowe poprawność  
 port, *Patrz* interfejs komponentów  
 pośrednik, 116, 128  
 potok, 41, 129, 215, 219  
 powielanie, 111  
 praca w czasie rzeczywistym, 26, 177, 179  
 problem modelowy, 422, 427  
 procedura, *Patrz też* moduł  
 dostępowa, 68  
 interfejsów urządzeń, 75  
 modeli fizycznych, 75  
 obsługi błędów, *Patrz* obsługa błędów  
 usług wspólnych, 75  
 proces, 35, 53, 75, 141  
 inicjowany na żądanie, 76  
 okresowy, 75  
 profilowanie, 227  
 projektowanie obiektowe, 58, 68  
 protokół  
 HTTP, 307, 309, 315, 316, 396, 398  
 HTTPS, 305, 309, 315, 316  
 NNTP, 309  
 SSL, 315  
 Usenet, 309  
 WAP, 399

prototyp, 47  
 budowa iteracyjna, 29  
 przenośność, *Patrz* atrybuty jakościowe  
 przenośność  
 przetwarzanie  
 rozproszone, 386, 395  
 równoległe, 34  
 przewidywanie zmian, 113  
 przybornik, *Patrz* warsztat  
 przywracanie stanu systemu, 110  
 punkt kontrolny, 111

## Q

quality attribute scenario, *Patrz* atrybuty jakościowe scenariusz  
 quality attribute tactics, *Patrz* atrybuty jakościowe taktyka

## R

Rational Unified Process, 56  
 reakcja, *Patrz* atrybuty jakościowe reakcja  
 redundancja, 148, 150  
 aktywna, 110, 111  
 pasywna, 111  
 podwójna, *Patrz* redundancja pasywna  
 sprzętowa, 284  
 reference model, *Patrz* model referencyjny  
 rekonstrukcja, *Patrz* architektura oprogramowania  
 rekonstrukcja  
 relacja  
 ma prawo używać, 73  
 używa, 71, 72  
 wywołuje, 71  
 repozytorium, 53, 119, 232, 258, 331, 355, 357, 363, 408  
 resynchronizacja stanu, 111  
 return on investment, *Patrz* zwrot z inwestycji  
 reverse engineering, *Patrz* inżynieria odwrotna  
 Rigi, 232  
 ROI, *Patrz* zwrot z inwestycji  
 rola, 177, 180  
 rollback, *Patrz* odwoływanie  
 rozwiązanie modelowe, 422, 427, 428, 429

## S

scenariusz, 81, 156, 255, 256, 260, 261, 262, 264, 266, 273, 274, 276, 278, 279, 285, 286, 290, 291, 293, 294, 408, 437  
 schemat  
 karuzelowy, 316  
 macierzowy, 191  
 sekwencji, 204  
 stanów, 204

sekret, 67, 69, 70, 71  
 sequence diagram, *Patrz* schemat sekwencji  
 serwet, 372, 379, 393, 400, 403, 409, 429, 431  
 shadow operation, *Patrz* powielanie  
 Shaw Mary, 129  
 signature, *Patrz* sygnatura  
 skalowalność, *Patrz* atrybuty jakościowe  
   skalowalność  
 skalowanie  
   pionowe, 386  
   poziome, 386  
 skrypt CGI, *Patrz* CGI  
 software product line, *Patrz* linia oprogramowania  
 specyfikacja wymagań, 26, 66  
 spójność  
   konceptyjna, *Patrz* atrybuty jakościowe  
   spójność konceptyjna  
   semantyczna, 113, 127, 141, 160  
 stakeholders, *Patrz* grupa interesu  
 statechart, *Patrz* schemat stanów  
 sterta, 387  
 strona shadow, 111  
 Structural Model, *Patrz* model strukturalny  
 struktura, 51, 54, 61, 201  
   alokacji, 51, 53  
   dekompozycji, 52, 54, 55, 61, 67, 68, 69  
   dominującą, 55  
   implementacja, 54  
   klasy, 53, 54  
   klient-serwer, 53, 54  
   komponenty-złącza, 51, 53, 55  
   modułów, 51, 52  
   podział pracy, 54  
   proces, 53, 54  
   procesów, 61, 67, 75, 77  
   rozmieszczenie, 53  
   użycia, 52, 54, 61, 66, 67, 71, 73, 74  
   warstwowa, 58  
   współbieżność, 53, 54  
 stub, 170  
 styl, *Patrz* architektura oprogramowania wzorzec  
 styl architektury, *Patrz* architektura  
   oprogramowania wzorzec  
 sygnatura, 206  
 system  
   czasu rzeczywistego, 170  
   klient-serwer, 170  
   podsystem UML, 218  
   regułowy, 170  
   szkieletowy, 34, 170, 189  
   wieloprosowy, 170  
   współdziałające zespoły, 219  
   wykrywania włamań, 123  
   zamknięty obiekt, 218  
 szablon, 50

## S

ścieżka komunikacji, 34  
 środowisko, *Patrz* atrybuty jakościowe środowisko  
   techniczne, 24, 25, 28, 29, 31

## T

tajemnica, *Patrz* sekret  
 taktyka, *Patrz* atrybuty jakościowe taktyka  
 TELNET, 309  
 test harness, *Patrz* jarzmo testów  
 testability, *Patrz* atrybuty jakościowe testowalność  
 testowalność, *Patrz* atrybuty jakościowe  
   testowalność  
 transakcja, 403, 404, 409  
 transakcje rozproszone, *Patrz* przetwarzanie  
   rozproszone  
 tworzenie abstrakcji wspólnych usług, 113

## U

ukrywanie  
   informacji, 54, 58, 62, 65, 66, 67, 68, 69, 78, 115,  
   160, 168  
   warstwy sprzętowej, 69  
   zachowań, 69  
 UML, 211, 212, 214, 215, 217, 220  
 Unified Modeling Language, *Patrz* UML  
 urządzenie przenośne, 394, 398  
 usability, *Patrz* atrybuty jakościowe  
   funkcjonalność  
 usługi wspólne, 113  
 uszkodzenie, 90, 109, 148  
   wykrywanie, 109  
   zapobieganie, 112  
 utility, *Patrz* wartość użytkowa  
 uzasadnienie biznesowe systemu, 32  
 użyteczność, *Patrz* wartość użytkowa

## V

Vasa, 24  
 virtual threads, *Patrz* wątek wirtualny

## W

warstwa, 74, 145, 168, 310, 314, 402  
   globalnych systemów informacyjnych, 373  
   internetowa, 372  
   klienta, 371  
   komponentów biznesowych, 373  
 warsztat, 224  
 wartość użytkowa, 285, 286, 287, 288, 291, 294, 296

wątek, 40, 53, 387, 439  
  fizyczny, 53  
  logiczny, 53  
  pseudowątek, 309  
  wirtualny, 162  
węzeł, 220, 356  
wiązanie, 117  
workbench, *Patrz* warsztat  
wrapper, *Patrz* osłona  
współbieżność, 204, 431  
wydajność, *Patrz* atrybuty jakościowe wydajność  
wykonalność, *Patrz* atrybuty jakościowe  
  wykonalność  
wymagania, 23, 25, 26, 28, 30, 362, 368, 375, 393,  
  410, 425, 437  
  analiza, 32, 157  
  jakościowe, 158  
  specyfikacja, 25, 26, 32  
  techniki gromadzenia, 32  
wzorzec, 232  
  interakcji, 35  
  model strukturalny, *Patrz* model strukturalny  
wzorzec architektury, *Patrz* architektura  
  oprogramowania wzorzec

## Z

zachowanie, 204  
założenie, 420  
  lista, 420  
  wymaga, 416, 419, 420, 421  
  zapewnia, 416, 419, 420, 421  
zapas, 111  
zapora, 315  
zarządzanie czasem, 181, 183  
zasoby, 162, 404, 420  
  blokada, 118  
  ograniczanie zapotrzebowania, 119, 120  
  poziom wykorzystania, 118, 416  
  priorytet dostępu, 120  
  pula, 387  
  zarządzanie, 120  
zespół sektora, 137  
zintegrowane środowisko programowania, *Patrz* IDE  
złącze, 51, 53  
zunifikowany język modelowania, *Patrz* UML  
zwrot z inwestycji, 285, 288, 296, 298, 325

## Z

źródło bodźca, *Patrz* atrybuty jakościowe źródło  
  bodźca

# ARCHITEKTURA OPROGRAMOWANIA W PRAKTYCE

Wydanie II

## KANON INFORMATYKI

Współczesne systemy informatyczne to zaawansowane, skomplikowane mechanizmy, składające się z wielu współdziałających ze sobą komponentów. Ich wyodrębnienie, a także określenie sposobu komunikacji i interakcji między poszczególnymi elementami jest nie lada wyzwaniem dla architektów. Od ich decyzji zależy, czy system uda się zrealizować, czy będzie on efektywny, stabilny i łatwy w utrzymaniu.

Na szczęście istnieją metodologie, narzędzia oraz sposoby analizy efektów ułatwiające i porządkujące cały ten proces. W tej książce znajdziesz wszystko, o czym trzeba pamiętać przy projektowaniu oprogramowania. Poznasz sposoby projektowania z wykorzystaniem Metody Analizy Kompromisów w Architekturze (ATAM) oraz oceniania aspektów finansowych przy użyciu Metody Analizy Kosztów i Korzyści (CBAM). Autorzy przedstawiają wiele studiów przypadków, które pozwolą Ci poznać rzeczywiste problemy i ich rozwiązania. Ponadto nauczysz się stosować język UML do wizualnej reprezentacji architektury systemu oraz dowiesz się, jak przygotować dobrą dokumentację projektu. Książka ta jest idealną propozycją dla każdego architekta oprogramowania.

### Poznaj najlepsze metodologie projektowania architektury!

- Proces wytwarzania oprogramowania a cykl biznesowy architektury
- Wzorce architektury
- Struktury i perspektywy architektury
- Określenie i uzyskanie atrybutów jakościowych
- Projektowanie architektury pod kątem wysokiej dostępności
- Proces projektowania architektury
- Dokumentowanie architektury oprogramowania
- Język UML
- Metody rekonstrukcji architektury i inżynierii odwrotnej
- Metoda Analizy Kompromisów w Architekturze (ATAM)
- Metoda Analizy Kosztów i Korzyści (CBAM)
- Ponowne wykorzystanie elementów architektury
- Dokumentowanie architektury

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 6810

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

Zamówienia telefoniczne:  
**0 801 339900**  
**0 601 339900**



**Helion**

Sprawdź najnowsze promocje

🔗 <http://helion.pl/promocje>

📖 Książki najchętniej czytane

🔗 <http://helion.pl/bestsellery>

📧 Zamów informacje o nowościach

🔗 <http://helion.pl/nowosci>

Helion SA  
ul. Kosciuszki 1c, 44-100 Gliwice  
tel. 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-3302-9



Cena 79,00 zł

Informatyka w najlepszym wydaniu