

Spis treści

Wstęp

17

Część I. Koncepcje i składniki architektury oprogramowania

Rozdział 1. Znaczenie architektury oprogramowania i zasady dobrego projektowania	23
Wymagania techniczne	24
Zrozumienie pojęcia „architektura oprogramowania”	24
Różne sposoby patrzenia na architekturę	25
Uświadomienie sobie znaczenia właściwej architektury	25
Rozpad oprogramowania	26
Przypadkowa architektura	26
Badanie podstaw dobrej architektury	26
Kontekst architektury	27
Interesariusze	27
Otoczenie biznesowe i techniczne	27
Opracowywanie architektury według zasad podejścia zwinnego	28
Projektowanie dziedzinowe	28
Filozofia języka C++	30
Stosowanie zasad SOLID i DRY	32
Zasada jednej odpowiedzialności	33
Zasada otwarty-zamknięty	33
Zasada podstawiania Liskov	34
Zasada podziału interfejsu	35
Zasada odwracania zależności	37
Zasada DRY	40
Sprzężenie i spójność	41
Sprzężenie	41
Spójność	42

Podsumowanie	45
Pytania	45
Materiały dodatkowe	45
Rozdział 2. Style architektoniczne	46
Wymagania techniczne	46
Wybór pomiędzy podejściem stanowym i bezstanowym	47
Usługi bezstanowe i stanowe	49
Zapoznanie się z monolitami — dlaczego należy ich unikać i jakie są wyjątki	50
Zrozumienie działania usług i mikrouslug	51
Mikrouslugi	52
Badanie architektury opartej na zdarzeniach	56
Najczęściej spotykane topologie oparte na zdarzeniach	57
Pozyskiwanie zdarzeń	58
Zrozumienie działania architektury warstwowej	59
Zaplecza dla interfejsów	61
Zapoznanie się z architekturą opartą na modułach	62
Podsumowanie	63
Pytania	63
Materiały dodatkowe	64
Rozdział 3. Wymagania funkcjonalne i нефункционалне	65
Wymagania techniczne	66
Zapoznanie się z typami wymagań	66
Wymagania funkcjonalne	66
Wymagania нефункционалне	67
Rozpoznanie wymagań istotnych dla architektury	68
Wskaźniki świadczące o istotności dla architektury	69
Trudności w rozpoznawaniu wymagań istotnych dla architektury	70
i jak sobie z nimi radzić	71
Zbieranie wymagań z różnych źródeł	71
Poznanie kontekstu	71
Poznanie istniejącej dokumentacji	72
Poznanie interesariuszy	72
Zbieranie wymagań od interesariuszy	73
Dokumentowanie wymagań	73
Dokumentowanie kontekstu	74
Dokumentowanie zakresu	74
Dokumentowanie wymagań funkcjonalnych	75
Dokumentowanie wymagań нефункционалnych	76
Zarządzanie historią wersji dokumentacji	77
Dokumentowanie wymagań w projektach zwinnych	77
Inne części	78
Dokumentowanie architektury	79
Zapoznanie się z modelem 4+1	79
Zapoznanie się z modelem C4	84
Dokumentowanie architektury w projektach zwinnych	86

Wybór właściwych perspektyw do udokumentowania	87
Perspektywa funkcjonalna	88
Perspektywa informacyjna	89
Perspektywa współbieżności	89
Perspektywa implementacyjna	90
Perspektywy wdrożeniowa i operacyjna	90
Generowanie dokumentacji	92
Generowanie dokumentacji wymagań	92
Generowanie diagramów na podstawie kodu	92
Generowanie dokumentacji (interfejsów API) na podstawie kodu	93
Podsumowanie	98
Pytania	99
Materiały dodatkowe	99

Część II. Projektowanie i wytwarzanie oprogramowania w języku C++

Rozdział 4. Projektowanie architektur i systemów	103
Wymagania techniczne	104
Zrozumienie specyfiki systemów rozproszonych	104
Różne modele usług i okoliczności ich stosowania	104
Unikanie błędnych założeń dotyczących przetwarzania rozproszonego	108
Twierdzenie CAP i spójność ostateczna	111
Zapewnienie systemowi dostępności i odporności na uszkodzenia	114
Obliczanie poziomu dostępności systemu	114
Budowanie systemów odpornych na uszkodzenia	115
Wykrywanie usterek	119
Minimalizowanie skutków usterek	120
Integrowanie systemu	122
Wzorzec Potoki i filtry	122
Konkurujący odbiorcy	123
Przechodzenie ze starszych systemów	124
Osiąganie wydajności w dużej skali	125
CQRS i pozyskiwanie zdarzeń	125
Pamięci podręczne	128
Wdrażanie systemu	130
Wzorzec Przyczepka	130
Wdrażanie bez przestojów	135
Zewnętrzny magazyn konfiguracji	136
Zarządzanie interfejsami API	137
Bramy interfejsów API	138
Podsumowanie	138
Pytania	139
Materiały dodatkowe	139

Rozdział 5. Wykorzystywanie cech języka C++	140
Wymagania techniczne	141
Projektowanie doskonałych interfejsów API	141
Korzystanie z idiomu RAII	141
Definiowanie interfejsów kontenerów w języku C++	142
Używanie wskaźników w interfejsach	145
Określanie warunków wstępnych i końcowych	146
Wykorzystywanie wbudowanych przestrzeni nazw	147
Korzystanie z typu <code>std::optional</code>	148
Pisanie deklaratywnego kodu	149
Prezentacja galerii z wyróżnionymi przedmiotami	151
Wprowadzenie standardowych zakresów	155
Przenoszenie obliczeń na czas kompilacji	158
Pozwól, by kompilator Ci pomagał — stosuj <code>const</code>	160
Wykorzystanie potęgi bezpiecznych typów	160
Ograniczanie parametrów szablonów	161
Pisanie modularnego kodu C++	165
Podsumowanie	167
Pytania	168
Materiały dodatkowe	168
Rozdział 6. Wzorce projektowe a język C++	169
Wymagania techniczne	169
Pisanie idiomatycznego kodu C++	170
Automatyzowanie operacji wyjścia z zakresu przy użyciu strażników RAII	170
Zarządzanie możliwościami kopiowania i przenoszenia	171
Korzystanie z ukrytych funkcji zaprzyjaźnionych	173
Zapewnianie bezpieczeństwa wyjątkowego przy użyciu idiomu „skopiuj i przestaw” (<code>copy-and-swap</code>)	174
Pisanie niebłoidów	176
Idiom klas parametryzowanych wytycznymi	178
Ciekawie rekurencyjny wzorzec szablonu	180
Rozeznanie, kiedy używać polimorfizmu dynamicznego, a kiedy statycznego	180
Implementowanie polimorfizmu statycznego	180
Przerywnik — wymazywanie typów	183
Tworzenie obiektów	185
Korzystanie z fabryk	185
Korzystanie z budowniczych	190
Śledzenie stanu i odwiedzanie obiektów w języku C++	194
Efektywne postępowanie z pamięcią	197
Ograniczenie dynamicznych alokacji dzięki optymalizacji SSO/SOO	198
Oszczędzanie pamięci dzięki technice COW	198
Korzystanie z alokatorów polimorficznych	199
Podsumowanie	203
Pytania	204
Materiały dodatkowe	204

Rozdział 7. Budowanie i pakowanie	206
Wymagania techniczne	206
Wykorzystanie kompilatorów do granic ich możliwości	207
Korzystanie z kilku kompilatorów	207
Skracanie czasu kompilacji	208
Znajdowanie potencjalnych problemów z kodem	211
Używanie narzędzi zorientowanych na kompilatory	213
Zapewnianie abstrakcji procesu budowania	214
Wprowadzenie do narzędzia CMake	214
Korzystanie z wyrażeń generatorów	218
Korzystanie z modułów zewnętrznych	219
Pobieranie zależności	220
Korzystanie ze skryptów wyszukiwania	221
Pisanie skryptów wyszukiwania	222
Korzystanie z menedżera pakietów Conan	225
Dodawanie testów	227
Wielokrotne korzystanie z kodu o dobrej jakości	229
Instalowanie	229
Eksportowanie	232
Korzystanie z narzędzia CPack	233
Pakowanie przy użyciu narzędzia Conan	235
Tworzenie skryptu conanfile.py	235
Testowanie pakietu Conan	238
Dodawanie kodu pakującego narzędzia Conan do pliku CMakeLists	239
Podsumowanie	240
Pytania	241
Materiały dodatkowe	241

Część III. Architektoniczne atrybuty jakościowe

Rozdział 8. Pisanie testowalnego kodu	245
Wymagania techniczne	245
Po co testować kod?	246
Piramida testów	247
Testowanie niefunkcjonalne	248
Testowanie regresyjne	249
Analiza przyczyn źródłowych	249
Podstawa do dalszych ulepszeń	250
Wprowadzenie do frameworków testowych	252
Przykład użycia frameworka GTest	252
Przykład użycia frameworka Catch2	252
Przykład użycia frameworka CppUnit	253
Przykład użycia frameworka Doctest	254
Testowanie kodu czasu kompilacji	255
Zapoznanie się z atrapami i imitacjami	255
Różne zamienniki testowe	256
Inne zastosowania zamienników testowych	256
Pisanie zamienników testowych	256

Projektowanie klas sterowane testami	260
Kiedy testy i projekt klasy nie pasują do siebie	260
Programowanie defensywne	260
Nudna stara śpiewka — najpierw pisz testy	262
Automatyzowanie testów na potrzeby ciągłej integracji/ciągłego wdrażania	262
Testowanie infrastruktury	264
Testowanie za pomocą narzędzia Serverspec	264
Testowanie za pomocą narzędzia Testinfra	265
Testowanie za pomocą narzędzia Goss	265
Podsumowanie	266
Pytania	266
Materiały dodatkowe	267
 Rozdział 9. Ciągła integracja i ciągłe wdrażanie	 268
Wymagania techniczne	269
Zapoznanie się z ciągłą integracją	269
Wydawaj wcześniej, wydawaj często	269
Zalety ciągłej integracji	270
Mechanizm bramkowy	271
Implementowanie potoku z użyciem narzędzia GitLab	271
Recenzowanie zmian w kodzie	273
Zautomatyzowane mechanizmy bramkowe	273
Przegląd kodu — ręczny mechanizm bramkowy	274
Różne podejścia do przeglądu kodu	275
Stosowanie żądań ściągnięcia (scalenia) na potrzeby przeglądu kodu	275
Badanie automatyzacji sterowanej testami	276
Programowanie sterowane zachowaniem	276
Pisanie testów na potrzeby ciągłej integracji	278
Ciągłe testowanie	279
Zarządzanie wdrażaniem jako kodem	280
Korzystanie z narzędzia Ansible	281
Jak narzędzie Ansible wpasowuje się w potok CI/CD	281
Używanie komponentów do tworzenia kodu wdrożeniowego	282
Budowanie kodu wdrożeniowego	283
Budowanie potoku CD	283
Ciągłe wdrażanie i ciągłe dostarczanie	284
Budowanie przykładowego potoku CD	284
Korzystanie z nieziennej infrastruktury	286
Czym jest niezmienna infrastruktura?	286
Korzyści z nieziennej infrastruktury	287
Budowanie obrazów instancji narzędziem Packer	288
Orkiestracja infrastruktury za pomocą narzędzia Terraform	289
Podsumowanie	291
Pytania	292
Materiały dodatkowe	292

Rozdział 10. Bezpieczeństwo kodu i wdrażania	293
Wymagania techniczne	293
Sprawdzanie zabezpieczeń kodu	294
Projektowanie z troską o bezpieczeństwo	294
Bezpieczne programowanie, wytyczne i biblioteka GSL	299
Defensywne programowanie, weryfikowanie wszystkiego	299
Najczęściej spotykane luki w zabezpieczeniach	301
Sprawdzanie, czy zależności są bezpieczne	302
Common Vulnerabilities and Exposures	303
Zautomatyzowane skanery	303
Zautomatyzowane zarządzanie uaktualnianiem zależności	304
Utwardzanie kodu	304
Zorientowany na bezpieczeństwo alokator pamięci	305
Zautomatyzowane sprawdzenia	305
Izolacja procesów i użycie piaskownicy	310
Utwardzanie środowiska	310
Konsolidacja statyczna kontra dynamiczna	311
Losowe generowanie układu przestrzeni adresowej	312
DevSecOps	312
Podsumowanie	312
Pytania	313
Materiały dodatkowe	313
Rozdział 11. Wydajność	314
Wymagania techniczne	314
Mierzenie wydajności	315
Przeprowadzanie dokładnych i istotnych pomiarów	315
Wykorzystywanie różnego typu narzędzi pomiarowych	315
Korzystanie z mikrotestów porównawczych	317
Profilowanie	325
Śledzenie	327
Pomaganie kompilatorowi w generowaniu wydajnego kodu	328
Optymalizacja całych programów	328
Optymalizowanie w oparciu o rzeczywiste wzorce użytkowania	328
Pisanie kodu zoptymalizowanego pod kątem pamięci podręcznej	329
Projektowanie kodu z myślą o danych	329
Zrównoleglanie obliczeń	331
Zrozumienie, czym się różnią wątki od procesów	331
Używanie standardowych algorytmów równoległych	332
Zrównoleglanie obliczeń przy użyciu frameworków OpenMP i MPI	333
Używanie koprocedur	334
Rozpoznanie narzędzi z biblioteki cpcpro	335
Zagłądanie pod maskę obiektów oczekiwanych i koprocedur	337
Podsumowanie	342
Pytania	342
Materiały dodatkowe	342

Część IV. Zasady projektowania natywnego dla chmury

Rozdział 12. Architektura zorientowana na usługi	347
Wymagania techniczne	347
Zapoznanie się z architekturą zorientowaną na usługi	348
Podejścia do implementacji	348
Korzyści wynikające z architektury zorientowanej na usługi	354
Wyzwania związane z SOA	355
Wdrażanie zasad wymiany komunikatów	356
Niskonarzutowe systemy wymiany komunikatów	357
Systemy wymiany komunikatów obsługiwane przez broker	358
Korzystanie z usług sieciowych	359
Narzędzia do debugowania usług sieciowych	359
Usługi sieciowe oparte na kodzie XML	360
Usługi sieciowe oparte na kodzie JSON	363
REpresentational State Transfer (REST)	365
GraphQL	371
Wykorzystywanie usług zarządzanych i dostawców chmury	372
Przetwarzanie w chmurze jako rozszerzenie architektury SOA	373
Architektura natywna dla chmury	377
Podsumowanie	377
Pytania	378
Materiały dodatkowe	378
Rozdział 13. Projektowanie mikrousług	379
Wymagania techniczne	379
Wniknięcie w temat mikrousług	380
Zalety mikrousług	380
Wady mikrousług	382
Wzorce projektowe mikrousług	383
Budowanie mikrousług	386
Delegowanie na zewnątrz zarządzania pamięcią	386
Delegowanie na zewnątrz magazynowania	389
Delegowanie na zewnątrz przetwarzania	389
Obserwowanie mikrousług	390
Rejestrowanie zdarzeń	390
Monitorowanie	395
Śledzenie	396
Zintegrowane rozwiązania z zakresu obserwowalności	397
Łączenie mikrousług	397
Interfejsy programowania aplikacji (API)	398
Zdalne wywołania procedur	398
Skalowanie mikrousług	400
Skalowanie wdrożenia typu jedna usługa na host	401
Skalowanie wdrożenia z wieloma usługami na hoście	401
Podsumowanie	402
Pytania	402
Materiały dodatkowe	403

Rozdział 14. Kontenery	404
Wymagania techniczne	404
Reaktywacja kontenerów	405
Poznawanie typów kontenerów	406
Wzrost popularności mikrouslug	406
Decydowanie, kiedy użyć kontenerów	407
Budowanie kontenerów	409
Wyjaśnienia na temat obrazów kontenerów	409
Budowanie aplikacji przy użyciu plików Dockerfile	410
Nazewnictwo i dystrybucja obrazów	411
Aplikacje kompilowane a kontenery	412
Definiowanie wielu architektur docelowych za pomocą manifestów	414
Alternatywne sposoby budowania kontenerów aplikacji	415
Integrowanie kontenerów z narzędziem CMake	417
Testowanie kontenerów i integrowanie ich ze sobą	419
Biblioteki środowiska uruchomieniowego wewnątrz kontenerów	419
Alternatywne środowiska uruchomieniowe kontenerów	421
Zapoznanie się z orkiestracją kontenerów	421
Rozwiązania utrzymywane samodzielnie	422
Usługi zarządzane	428
Podsumowanie	430
Pytania	430
Materiały dodatkowe	430
Rozdział 15. Projektowanie rozwiązań natywnych dla chmury	431
Wymagania techniczne	432
Zapoznanie się z rozwiązaniami natywnymi dla chmury	432
Cloud-Native Computing Foundation	432
Chmura jako system operacyjny	433
Orkiestracja obciążeń natywnych dla chmury przy użyciu platformy Kubernetes	435
Struktura platformy Kubernetes	436
Możliwe podejścia do wdrażania platformy Kubernetes	437
Zrozumienie koncepcji platformy Kubernetes	437
Sieć na platformie Kubernetes	440
Kiedy korzystanie z platformy Kubernetes jest dobrym pomysłem?	440
Obserwowalność w systemach rozproszonych	441
Czym śledzenie różni się od rejestrowania	442
Wybór rozwiązania do śledzenia	443
Instrumentacja aplikacji w standardzie OpenTracing	445
Łączenie usług za pomocą siatki usług	446
Zaznajomienie się z siatką usług	446
Rozwiązania siatki usług	447
Podejście GitOps	449
Zasady GitOps	450
Zalety podejścia GitOps	452
Narzędzia GitOps	453

Podsumowanie	454
Pytania	455
Materiały dodatkowe	455
Dodatek A	457
Dodatek B	461