目次

第 1 章　ソフトウェアアーキテクチャの原則　1

1.1　ソフトウェアアーキテクチャの定義 …… 2
1.1.1　ソフトウェアアーキテクチャとソフトウェアデザイン …… 2
1.1.2　ソフトウェアアーキテクチャの性質 …… 3
1.2　ソフトウェアアーキテクチャの特性 …… 3
1.2.1　ストラクチャの定義 …… 3
1.2.2　重要な要素の明確化 …… 5
1.2.3　デザインの方針の決定 …… 5
1.2.4　ステークホルダーの要件の管理 …… 6
1.2.5　ストラクチャに与える影響 …… 6
1.2.6　環境から与えられる影響 …… 7
1.2.7　システムの文書化 …… 8
1.2.8　パターンへの準拠 …… 8
1.3　ソフトウェアアーキテクチャの重要性 …… 9
1.4　システムアーキテクチャとエンタープライズアーキテクチャ …… 11
1.5　アーキテクチャの品質属性 …… 14
1.5.1　修正容易性 …… 14
1.5.2　テスト容易性 …… 17
1.5.3　スケーラビリティ …… 19
1.5.4　パフォーマンス …… 20
1.5.5　可用性 …… 21
1.5.6　セキュリティ …… 23
1.5.7　デプロイ容易性 …… 24
1.6　まとめ …… 25

第 2 章　修正容易性と可読性　27

2.1　修正容易性とは …… 27
2.2　修正容易性に関連する品質属性 …… 27

2.3 可読性とは 28
2.3.1 Python と可読性 29
2.3.2 可読性のアンチパターン 30
2.4 可読性向上のテクニック 33
2.4.1 ドキュメンテーション 33
2.4.2 コーディングガイドラインとスタイルガイドライン 39
2.4.3 コードレビューとリファクタリング 40
2.4.4 コードコメント 40
2.5 凝集度と結合度 41
2.5.1 凝集度と結合度の分析――配列の演算処理 42
2.5.2 凝集度と結合度の分析――文字列処理 44
2.6 修正容易性向上のテクニック 46
2.6.1 明示的なインタフェースの提供 46
2.6.2 双方向結合の削減 46
2.6.3 抽象共通サービス 47
2.6.4 継承の活用 48
2.6.5 遅延バインディング 51
2.7 静的解析ツールとメトリクス測定 52
2.7.1 コードの臭い 53
2.7.2 循環的複雑度 54
2.7.3 メトリクスのテスト 55
2.7.4 静的解析ツールの活用 57
2.8 コードリファクタリング 63
2.8.1 複雑度の削減 64
2.8.2 コードの臭いの削減 66
2.8.3 スタイリングの修正 68
2.9 まとめ 68

第 3 章 テスト容易性 69

3.1 テスト容易性とは 69
3.1.1 テスト容易性に関連する性質 70
3.1.2 様々なソフトウェアアーキテクチャの性質に対するテスト 70
3.2 テストの戦略 72
3.2.1 複雑さの削減 73
3.2.2 予測可能性の改善 73
3.2.3 外部依存の制御と分離 74

3.3 ホワイトボックステスト 78
3.3.1 単体テスト 78
3.3.2 コードカバレッジ 86
3.3.3 モックの便利な利用方法 90
3.3.4 doctest ――インラインドキュメントによるテスト 94
3.3.5 結合テスト 97
3.3.6 Selenium WebDriver による自動テスト 99
3.4 テスト駆動開発 101
3.4.1 テスト駆動開発の実践 102
3.5 まとめ 107

第 4 章 パフォーマンス 109

4.1 パフォーマンスとは 110
4.2 ソフトウェアパフォーマンスエンジニアリング 110
4.3 パフォーマンステストツールと測定ツール 111
4.4 計算量 113
4.5 パフォーマンス測定 114
4.5.1 コンテキストマネージャによる時間計測 115
4.5.2 `timeit` モジュールによる時間計測 118
4.5.3 グラフによる計算量の決定 120
4.5.4 `timeit` を用いた CPU 時間の計測 125
4.6 プロファイリング 126
4.6.1 決定論的プロファイリング 126
4.6.2 `cProfile` と `profile` を用いたプロファイリング 127
4.6.3 プロファイリング結果の保存と出力 131
4.6.4 サードパーティ製のプロファイラ 132
4.7 その他のツール 140
4.7.1 Objgraph 140
4.7.2 Pympler 142
4.8 データ構造のプログラミングパフォーマンス 144
4.8.1 可変コンテナオブジェクト――リスト，辞書，セット 144
4.8.2 不可変コンテナオブジェクト――タプル 146
4.8.3 ハイパフォーマンスのコンテナ―― `collections` モジュール 147
4.8.4 確率的データ構造―― Bloom Filter 154
4.9 まとめ 158

第5章　スケーラビリティ　160

5.1　スケーラビリティとパフォーマンス 161
5.2　並行性 163
5.2.1　並行性と並列性 164
5.3　マルチスレッディング 165
5.3.1　サムネイルジェネレータ 166
5.3.2　サムネイルジェネレータ――プロデューサ/コンシューマモデル 167
5.3.3　サムネイルジェネレータ――ロック 172
5.3.4　サムネイルジェネレータ――セマフォ 176
5.3.5　ロック vs. セマフォ 178
5.3.6　サムネイルジェネレータ―― `Condition` 179
5.3.7　Python と GIL 186
5.4　マルチプロセッシング 187
5.4.1　素数チェッカーの実装 187
5.4.2　ファイルのソート 190
5.4.3　ファイルのソート――カウンタ 191
5.4.4　ファイルのソート――マルチプロセッシング 194
5.5　マルチスレッディング vs. マルチプロセッシング 196
5.6　非同期処理 197
5.6.1　プリエンプティブマルチタスクと協調的マルチタスク 198
5.6.2　`asyncio` 201
5.6.3　`async` と `await` 204
5.6.4　`concurrent.futures` ――ハイレベルな並行処理 207
5.7　並行処理の選択肢 210
5.8　並行処理のライブラリ 211
5.8.1　`joblib` 212
5.8.2　`PyMP` 213
5.9　Web を用いたスケール 219
5.9.1　MQ ――メッセージキュー 219
5.9.2　Celery ――分散型タスクキュー 220
5.10　WSGI 224
5.10.1　uWSGI 226
5.10.2　Gunicorn 228
5.10.3　Gunicorn vs. uWSGI 228
5.11　スケーラビリティアーキテクチャ 229
5.11.1　垂直スケーラビリティアーキテクチャ 229

5.11.2 水平スケーラビリティアーキテクチャ 229
5.12 まとめ 234

第 6 章 セキュリティ 235

6.1 情報セキュリティアーキテクチャ 236
6.2 セキュアコーディングとは 237
6.3 一般的な脆弱性 238
6.4 Python のセキュリティ 243
6.4.1 入力の読み込み 244
6.4.2 任意の入力の評価 247
6.4.3 オーバーフローエラー 250
6.4.4 シリアライズ 252
6.5 Web アプリケーションのセキュリティ 256
6.5.1 サーバーサイドテンプレートインジェクション 256
6.5.2 サーバーサイドテンプレートインジェクションへの対策 260
6.5.3 DoS 攻撃 261
6.5.4 XSS 264
6.5.5 DoS 攻撃と XSS への対策 265
6.6 セキュアコーディングの注意点 266
6.7 セキュアコーディングの開発方針 273
6.8 まとめ 274

第 7 章 デザインパターン 275

7.1 デザインパターンの構成要素 276
7.2 デザインパターンのカテゴリ 277
7.2.1 プラグ可能なハッシュアルゴリズム 278
7.2.2 プラグ可能なハッシュアルゴリズム実装から見えること 281
7.3 生成に関するパターン 281
7.3.1 Singleton パターン 281
7.3.2 Singleton vs. Borg 286
7.3.3 Factory パターン 288
7.3.4 Prototype パターン 290
7.3.5 Builder パターン 298
7.3.6 生成に関するパターンのまとめ 303

7.4 構造に関するパターン 303
7.4.1 Adapter パターン 304
7.4.2 Facade パターン 312
7.4.3 Proxy パターン 318
7.5 振る舞いに関するパターン 322
7.5.1 Iterator パターン 322
7.5.2 Observer パターン 325
7.5.3 State パターン 332
7.6 まとめ 337

第 8 章 アーキテクチャパターン 339

8.1 MVC の概要 339
8.2 Django 341
8.2.1 Django admin ――管理システム 342
8.3 Flask 343
8.4 イベント駆動型プログラミング 344
8.4.1 `select` を用いたチャットサーバー/クライアント 344
8.4.2 イベント駆動 vs. 並行プログラミング 349
8.4.3 Twisted 350
8.4.4 Eventlet 357
8.4.5 Greenlets と Gevent 358
8.5 マイクロサービスアーキテクチャ 360
8.5.1 Python でのマイクロサービスフレームワーク 361
8.5.2 マイクロサービスの例 362
8.5.3 マイクロサービスの利点 364
8.6 パイプとフィルタのアーキテクチャ 365
8.6.1 Python におけるパイプとフィルタの例 365
8.7 まとめ 370

第 9 章 デプロイ容易性 371

9.1 デプロイ容易性とは 372
9.1.1 重要な要素 372
9.2 マルチティアアーキテクチャ 374

9.3 Python でのデプロイ ... 375
9.3.1 パッケージング ... 375
9.3.2 pip ... 376
9.3.3 virtualenv ... 377
9.3.4 pip と virtualenv ... 379
9.3.5 仮想環境の再配置 ... 380
9.3.6 PyPI ... 381
9.3.7 PyPI へのアップロード ... 382
9.3.8 PyPA ... 388
9.3.9 Fabric ... 388
9.3.10 Ansible ... 390
9.3.11 Supervisor ... 391
9.4 デプロイパターン ... 392
9.5 まとめ ... 395

第 10 章 デバッグのテクニック 396

10.1 `print` によるデバッグ ... 396
10.1.1 `print` の挿入 ... 397
10.1.2 分析と修正 ... 398
10.1.3 処理速度の最適化 ... 401
10.2 シンプルなデバッグテクニック ... 403
10.2.1 単語検索プログラム ... 403
10.2.2 コードブロックのスキップ ... 406
10.2.3 実行停止 ... 407
10.2.4 外部依存への対策 ... 407
10.2.5 関数のモック化による戻り値の置き換え ... 408
10.3 ロギング ... 417
10.3.1 シンプルなロギング ... 418
10.3.2 発展的なロギング── logger オブジェクト ... 419
10.4 デバッガ ... 424
10.4.1 pdb ... 425
10.4.2 pdb ──拡張ツール ... 427
10.5 発展的なデバッグ──トレース ... 429
10.5.1 `trace` ... 429
10.5.2 `lptrace` ... 430

10.5.3 strace 431
10.6 まとめ 432

索引 433