

まえがき	iii
------	-----

▶▶ PartI ソフトウェアテストと品質マネジメント 1

第 1 章	ソフトウェアテストとは	2
1.1	ソフトウェアテストの目的	2
1.2	高スピードと高品質の両立を支えるソフトウェアテスト	3
	優れたソフトウェアテストとは	3
	優れたソフトウェアテストの実現	5
	ソフトウェアテストの施策を支える基礎	7
1.3	ソフトウェアテストの課題	8
	ソフトウェアテストが抱える制約	8
	ソフトウェアテストは本質的に制約だらけで力不足	10
	ソフトウェアテストは高度な知的作業	10
第 2 章	品質マネジメントとソフトウェアテスト	12
2.1	ソフトウェアテストの強みを活かし弱みを補う	12
2.2	品質マネジメント	13
2.3	品質マネジメントシステムの実現	14
2.4	品質マネジメントの視座でソフトウェアテストを活用する	15
2.5	ソフトウェア開発に求められる品質マネジメント	15
2.6	品質マネジメントを担う部門の陳腐化	16
	品質保証部門の陳腐化は品質を棄損する	17
	品質保証部門の陳腐化の兆候	18
	品質保証部門の陳腐化の予防・脱却	19
2.7	品質とは	20
	自分が手掛ける品質を考える	20
	チーム全体で顧客視点の品質を追求する	21
	プロダクト品質とプロセス品質	21
	品質モデル	22
	アンチパターン：標準品質モデルをそのまま使用	23
第 3 章	ソフトウェアテストの全体像	24
3.1	テストに関わる概念と用語	24
	テストとデバッグ	24
	品質保証、品質管理、品質マネジメント、テスト	24
	動的テストと静的テスト	27
	エラー、バグ、故障	27
3.2	テストの構成要素	28
	テストレベル	28
	テストタイプ	29
	テストフェーズ	30

〔第 4 章〕	テストの戦略立て	32
4.1	テスト戦略とは	32
	テスト戦略として管理する対象	32
	テスト戦略の実体	33
4.2	テスト戦略の構築	34
	テスト戦略のインプット	34
	テスト戦略構築のタイミング	34
	テスト戦略構築の種類	35
	テスト戦略の担い手と運用	36
〔第 5 章〕	定番のテスト戦略	37
5.1	シフトレフトテストの戦略	37
	シフトレフトテストの種類	38
	シフトレフトテストの適用先	39
	シフトレフトテストを支える戦略	40
	シフトレフトテストを支えるテストアーキテクチャ設計の戦略	43
5.2	リグレーションテストの戦略	45
	リグレーションテストの戦略の重要性	46
	リグレーションテストの方針の難しさ	46
	リグレーションリスクには様々な手段で対応する	47
	リグレーションテストの戦略	48
5.3	Wモデルの戦略	49
	前提となるVモデル	49
	Wモデル	50
	テスト分析・設計を通した仕様改善フィードバックとは	51
	Wモデルを支える人的リソース	52
	Wモデルのためのプロセス設計	52
	ATDD (受け入れテスト駆動開発)	53
5.4	シフトライトテストの戦略	53
	リアクティブなテストとプロアクティブなテスト	54
	シフトライトテストの適用先	55
	カナリアテスト	55
	スクィーズテスト	56
	カオスエンジニアリング	56
〔第 6 章〕	アジャイル開発でのテスト戦略	58
6.1	アジャイル開発の価値・原則を支えるテスト戦略	58
	アジャイルソフトウェア開発宣言	58
	アジャイルの原則	59
	アジャイル開発を支えるテスト戦略	59
6.2	アジャイルテストの原則	61
6.3	アジャイルテストの四象限	62

6.4	アジャイル開発プロセスへの対応	63
	スクラムとは	63
	スクラムにおけるテスト	64
	スクラムでのテストの体制	65
6.5	顧客満足やビジネス価値よりアジャイルの形式を優先しない	66
	品質要求の厳しさに基づくテラリング	67
	ユーザへのデプロイの頻度と迅速さに基づくテラリング	67
	要件の確度に基づくテラリング	67
6.6	リリーススプリントのリスクに対応する	68
6.7	ゾンビスクラムとその対策	68
〔第 7 章〕	継続的デリバリーでのテスト戦略	70
7.1	継続的デリバリーとは	70
7.2	デプロイメントパイプラインとは	72
7.3	継続的デリバリーを支えるテストアプローチ	72
	テスト自動化を推進する	72
	手動テストと自動テストの責務分担の一貫性を確保する	73
	チーム全体アプローチでテストを進める	73
	テストの保守性と信頼性を作りこむ	73
	自動テストのテストプロセスを整備する	74
	デプロイメントパイプラインをテストプロセスのモデルにする	74
7.4	継続的デリバリーで活用できるテストアプローチ	75
	時系列の変化に着目したテストの活用	75
	統合的なテスト結果分析	76
	リリース成果物に対するフィードバックの迅速化	76
7.5	継続的デリバリーのテストを支える体制	76
〔第 8 章〕	DevOpsでのテスト戦略	77
8.1	DevOpsとは	77
8.2	継続的テストとは	77
8.3	ホリスティックテストとは	78
8.4	DORA Four Keysとテストでの対応	80
	Four Keysの注意点	81
8.5	DevOpsを支えるテストの体制	81
8.6	DevOpsを支えるテストのアプローチ	81
	ビジネス、開発、運用、テスト共同でテストを推進	81
	CI/CDへのテストの組み込み	82
	テスト自動化の促進	82
	テスト容易性の作りこみ	82
	シフトライトテストの推進	82

〔第 9 章〕	ソフトウェアプロダクトライン開発でのテスト戦略	83
9.1	ソフトウェアプロダクトライン開発とは	83
9.2	ソフトウェアプロダクトライン開発のプロセス	84
9.3	ソフトウェアプロダクトライン開発を支えるテスト体制	85
9.4	ソフトウェアプロダクトライン開発を支えるテストアプローチ	85
	ドメインテストとアプリケーションテストの分担を最適化する	85
	ドメインテストは段階的に育て洗練させる	86
	テストをプロダクトコードのパラダイムに合わせる	87
	可変性はプロダクトマネジメントの管理下に置き、テスト単独で対処しない	87
	自動化されたリグレッションテストおよびテスト自動化容易性を充実させる	88
	テストのモジュール性を確保する	88
	持続可能なテストウェアやテストシステムを構築する	89
〔第 10 章〕	テストを支えるプロセスの構築と運用	90
10.1	開発プロセスとは	90
	プロセスの役割	91
	プロセスのモデル記法	92
	開発プロセスモデルと開発方法論	92
10.2	テストを支える開発プロセス設計	93
	顧客満足・ビジネス価値からのフィードバックを確保する	93
	反復開発を導入する	94
	シフトレフトを推進する	95
	品質の積み上げを導入する	95
	改善サイクルを組み込む	96
10.3	テストプロセス設計	96
	プロセスタイプごとにテストプロセスを設計する	97
	テストプロセスにバリエーションを持たせる	98
	開発とテストのコミュニケーションをプロセスで支える	99
	反復ステージごとにプロセスを定義する	100
10.4	開発プロセス・テストプロセスの構築の流れ	101
	(1) プロセスの設計	101
	(2) プロセスの運用	102
10.5	関連プロセスの整備	102
	支援プロセス	103
	運用プロセス	104
	管理プロセス	105
10.6	テストプロセスの改善	105
	テストプロセス改善の要点	105
	テストプロセス改善のステップ	107
	テストプロセス改善モデルを使用した改善	110
	開発プロセス改善モデルを使用した改善	114

〔第 11 章〕	テストの分析・設計・実装の全体像	116
11.1	テストの作成で求められる工夫	116
	何をテストすべきかよく分析する	116
	対応が必要なプロダクトリスクや品質要求を深掘りする	117
	関心の分離を実施し、テストを構造化する	117
	テストの責務に対し、どのようにテストするかを適切なアプローチで組み立てる	117
11.2	テストを作成する全体の流れ	117
〔第 12 章〕	テスト基本分析	119
12.1	テスト基本分析とは	119
	テスト基本分析の実施タイミング	120
12.2	テスト要求・制約のステークホルダを特定する	120
12.3	必要なテストベースを特定する	121
	必要なテストベースの手配	122
	入手時期のすり合わせ	122
12.4	テストの要求・制約を収集し分析する	123
	テストの要求分析の段階的詳細化	123
	アンチパターン：テストベースの多重化	123
12.5	テスト対象を整理し理解する	124
	構造の観点での整理	124
	ふるまいの観点での整理	125
	品質特性の観点での整理	125
12.6	プロダクトリスクを分析する	125
12.7	必要なテスト環境を分析する	126
12.8	テストの責務を具体化する	126
	テスト対象の分析	126
	テストのスコープの分析	127
	テストの目的の定義	127
	テストの十分性基準の設定	127
	テストで対応すべき重大な制約や課題の分析	127
〔第 13 章〕	テストアーキテクチャ設計	129
13.1	テストアーキテクチャとは	129
13.2	テストアーキテクチャ設計とは	130
	テストアーキテクチャ設計の目的	130
	テストアーキテクチャ設計の実施タイミング	131
	テストアーキテクチャ設計の担い手	131
13.3	テストアーキテクチャ設計の進め方	132
	テストアーキテクチャに関わる要求・制約を分析する	132
	既存のテストアーキテクチャを参考にする	133
	大規模で複雑な設計を行う場合、基本設計方針をたてる	133

課題やリスクへの対応検討の中でテスト活動を識別する	134
プロダクトの要求や品質のタイプに応じてテスト活動を識別する	135
プロダクトの設計構造に応じてテスト活動を識別する	135
テスト活動の詳細度・厚みを調整する	135
全体の構造を整理する	136
すり合わせ	137
13.4 テストアーキテクチャの評価と改善	138
事前評価	138
事後評価	139
評価手法の例：シナリオウォークスルー	139
13.5 高スピードと高品質を両立させるテストアーキテクチャ設計	140
テストアーキテクチャ設計をとりまく環境の発展	140
テストアーキテクチャ設計の二つのアプローチ	142
高スピードと高品質を両立するテストアーキテクチャ設計の前提	143

〔第14章〕 **テストアーキテクチャ設計手法：VSTeP** 145

14.1 VSTePとは	145
VSTePの恩恵	145
VSTePの適用先	146
モビングの手法へ	146
VSTePの流れ	147
14.2 VSTePの基本概念	147
テスト観点	147
テストコンテナ	147
テスト観点図	148
テストコンテナ図	149
テストフレーム	150
14.3 VSTePの進め方	150
テスト観点を出す	150
テスト観点を整理する	151
テストコンテナを設計する	152
テストコンテナの関係性を設計する	153
テストフレームを設計する	153
テスト条件分析・テストケース設計を行う	154
14.4 まとめ	154

〔第15章〕 **テスト詳細分析とテスト設計** 155

15.1 テスト詳細分析・テスト設計とは	155
15.2 テスト詳細分析・テスト設計	156
15.3 テスト観点を識別・整理する	156
テスト観点の抽出	157
気がかりレビューの実施	157
テスト観点分析を行うタイミング	158
15.4 テスト設計のアンチパターン：CPM法	158
15.5 テスト対象を項目分けする	159

テストベースの項目分け・整理	160
仕様項目の形式の選択	160
仕様項目の分類	161
仕様モデルを利用した分析	161
15.6 テスト条件を分析する	162
15.7 テスト網羅基準を検討する	162
15.8 テストケースを設計する	163
15.9 テスト設計からフィードバックしてテストベースを改善する	163
15.10 テスト設計を保守・改善する	163
テスト設計の技術的負債への対応	164
テストの弱化への対応	164

〔第16章〕 テスト設計技法の活用	166
16.1 テスト設計技法とは	166
テスト設計技法は多くのエンジニアにとって有益	167
テスト設計技法が扱うテストモデルは思考を強化する型	167
16.2 テスト設計技法のタイプ	167
16.3 同値分割法	168
適用先	168
技法の手順	169
留意事項	170
16.4 境界値分析	171
適用先	171
境界値分析を行う理由	171
技法の手順	171
留意事項	173
16.5 クラシフィケーションツリー法	174
適用先	174
クラシフィケーションツリーのモデル	174
技法の手順	175
留意事項	178
16.6 デシジョンテーブルテスト	178
適用先	178
デシジョンテーブルの記法	179
技法の手順	182
留意事項	183
16.7 状態遷移テスト	184
適用先	184
状態遷移図の記法	185
状態遷移表の記法	186
状態遷移図と状態遷移表の使い分け	187
状態遷移図の表現の洗練や簡略化	188
状態遷移に対する網羅基準	189
技法の手順	191

	留意事項	192
16.8	nワイズテスト	192
	概要	192
	適用先	192
	nワイズテストにおける組み合わせについて	193
	直交表技法とHAYST法	193
	技法の手順	194
	留意事項	195
16.9	エラー推測	196
	概要	196
	適用先	196
	技法の手順	196
	留意事項	197
16.10	制御フローテスト	198
	概要	198
	適用先	198
	カバレッジの種類	198
	コードカバレッジの限界	202
	技法の手順	202
	留意事項	203
16.11	テスト設計技法の使い分け	204
〔第17章〕	応用的なテスト設計のアプローチ	206
17.1	ユニットテストのテスト設計	206
	ユニットテストのテスト設計方針	207
	コードカバレッジは補完的に活用する	208
	ユニットテストのコードカバレッジ目標の設定	209
17.2	性能テストのテスト設計アプローチ	210
	計画での性能テストへの配慮	210
	性能テストの分析・設計の要点	212
	テスト実行の要点	215
17.3	組み合わせに対するテスト設計アプローチ	216
	パラメータと値を削減する。組み合わせを実現不能にする	216
	組み合わせに起因するプロダクトリスクを下げる	217
	組み合わせのテストが得意な他活動にテストの責務を渡す	217
	重要な箇所はピンポイントのテストを充実させ、全体の一律網羅を緩和する	218
	組み合わせの削減には、組み合わせ起因のプロダクトリスクの軽減が不可欠	218
〔第18章〕	テスト実装	220
18.1	テスト実装とは	220
	テスト管理ツールの導入・整備	220
18.2	テスト手順の作成	221
	テストケースのテスト手順書化	221
	テスト実装でのテスト条件の補完	222
	テスト実行のための情報補完	223

18.3	テストの選択・絞り込み	223
18.4	予測的テスト選択 (Predictive Test Selection)	224
18.5	テストの優先付け・順序設定	224
	テストをブロック・無効化するリスクを最初に確認する	224
	テスト実行の事前条件、事後条件の制約に従う	225
	デバッグの手間が大きいバグを早めに見つける	225
	プロジェクトリスクの高いバグを見つけられるテストの順序を早める	225
	開発者やマネジメントが気にする情報を早めに見つける	226
	プロダクトの品質・信頼性を高める期間を最後に確保する	226
〔第19章〕	テスト環境の構築	227
19.1	テスト環境構築の重要性	227
	テスト環境の構成要素	227
19.2	テスト環境構築のプロセス	229
	早期に対応すべきテスト環境構築活動	229
	プロダクトの種類に応じて早期対応が求められるテスト環境構築活動	230
19.3	テスト環境構築からテストへの逆提案	232
19.4	テスト環境の品質を作りこむ	232
	テスト環境で注意が必要な品質	232
19.5	テスト環境のテスト	234
〔第20章〕	テストの実行と結果判定	235
20.1	テスト実行を準備する	235
	テスト実行のトラブルに備える	235
	テスト実行に関わる必要な合意形成を行う	236
	テスト実行を効率化する技術を獲得する	237
20.2	テストを実行する	237
	テスト実行特有の課題・リスクに対応する	237
	チームのボトルネックに注力する	239
20.3	テスト実行の学習効果を活かす	239
	様々な立場の人をテスト実行に巻き込む	240
	探索的テストを推奨する	240
	テスト実行者の気づきや学習を残して活かす	241
	テスト実行者ファーストを推進する	241
20.4	忍者式テスト	241
20.5	バグを報告する	242
	開発者と信頼関係を構築する	242
	バグ対応のやりやすさに配慮する	242
	バグ情報を資産として使えるようにする	243
20.6	テスト結果を評価しリリース判定する	243
	テスト実行を評価する	243
	テスト結果を総合判断する	244
	リリース判定	244

〔第21章〕	リスクベースドテスト	247
21.1	リスクベースドテストとは	247
21.2	リスクとリスクマネジメント	247
	リスクの構造	248
	リスクマネジメントとは	249
21.3	リスクマネジメントの活動	249
	リスク特定	249
	リスク分析・評価	251
	リスクのコントロール	252
21.4	リスクベースドテストの概要	253
21.5	リスク事象に基づいてテストする	253
	使いどころ	253
	進め方	254
	留意点	255
21.6	機能や特性ごとのリスクレベルに基づいてテストする	255
	使いどころ	255
	進め方	255
	留意事項	256
21.7	構成要素ごとのリスクレベルに基づいてテストする	256
	使いどころ	257
	進め方	257
	留意点	257
21.8	リスクベースドテストの恩恵	258
〔第22章〕	探索的テスト	259
22.1	探索的テストとは	259
	探索的テストのメリット	259
	探索的テストのデメリット	260
	スクリプトテストと探索的テストは補完関係	261
	探索的テストの適用領域	261
	探索的テストの実施者に求められる能力	262
22.2	探索的テストのスタイル	262
	フリースタイルの探索的テスト	262
	テストチャータを使った探索的テスト	263
	セッションベーステスト	264
22.3	探索的テストの効果を引き出す	265
	テスト環境を早期確保する / テスト環境の制約を早期対策する	265
	探索的テストの活躍どころを計画で確保する	265
	必要な知識と能力をチームとして確保する / 学習機会を確保しチームを育てる	265
	正しい方向にテストを方向付けする / フィードバックサイクルを回して方向性を正す	266
	効率化のためのテスト技術を蓄積する	266
〔第23章〕	ユーザーストーリーテスト	267
23.1	ユーザーストーリーテストとは	267

	ユーザーストーリーテストはコラボレーションベースの探索的テスト	267
	リリース方針にテストの十分性基準を合わせる	268
	スクラムのPBIの単位とした場合はユーザーストーリーの粒度に注意する	268
23.2	ユーザーストーリーの記述	269
	ユーザーストーリー記述の原則	269
	ユーザーストーリーの受け入れ基準の記述	270
	ユーザーストーリーに関連するDoDの記述	270
23.3	ユーザーストーリーテストの進め方	271
	(1) ユーザーストーリーおよび受け入れ基準の改善	271
	(2) ユーザーストーリーテストのテスト設計	271
	(3) ユーザーストーリーテストの実行	272
第24章	静的テスト	273
24.1	静的テストとは	273
	静的テストの対象	273
24.2	レビュー	274
	レビューの役割	274
	レビューの種類	275
	テクニカルコミュニケーションのためのレビュー	276
	開発のインフラやツールと紐づいたレビュー	276
	レビューの技法	277
	レビューの流れ	278
24.3	静的解析	279
	静的解析の強み	280
	静的解析を効率的に行うための前提	280
	静的解析に基づいたリスクベースドテスト	280
24.4	静的テストで支えるコーディング規約	281
24.5	デプロイメントパイプラインへの静的テストの統合	282
24.6	静的テストで動的テストを削減する	282

▶▶ PartIV 自動テストの活用 285

第25章	自動テストの活用	286
25.1	自動テストとは	286
25.2	自動テストの効果	286
25.3	自動テストの注意点	288
25.4	自動テストの成功とは	289
	自動テストの費用	290
	自動テストの効果	291
	自動テストの費用対効果	292
	自動テストの目的	292
25.5	テストレベルに応じた自動テストの構築	293

ユニットテスト	293
統合テスト	294
システムテスト	294
25.6 自動テストの構築・運用の流れと要点	294
(1) 自動テストの要求の把握	295
(2) 自動テストの目的・目標の設定	295
(3) 自動テストの計画・段取りの構築	298
(4) 自動テストの構築	300
(5) 自動テストの評価	302
(6) 自動テストの運用と改善	303
25.7 自動テストの成功を支えるチームと仕組み	303
自動テストを支える開発インフラを整備する	304
自動テストの能力を持つ人とチームを確保する	304
自動テストを支えるチーム文化を形成する	305
プロセスでテスト自動化活動を支援する	306
〔第26章〕 自動テストの品質の作りこみ	307
26.1 自動テストの保守性の確保	307
自動テストの保守性とは	307
26.2 レガシーテスト/テストの技術的負債への対応	310
テストの技術的負債の予防	310
テストの技術的負債への対策	310
レガシーテストの予防	311
レガシーテストへの対策	311
26.3 脆いテスト（フラジャイルテスト）への対応	312
脆いテストが引き起こす問題	312
脆いテストの原因	312
脆いテストの改善	313
26.4 自動テストの信頼性の確保	313
26.5 フレーキーテストへの対応	314
フレーキーテストが引き起こす問題	315
フレーキーテストの原因	315
フレーキーテストの対策	316
〔第27章〕 自動テストの評価	319
27.1 自動テストの目的達成と費用対効果を評価する	319
目的達成の評価	319
費用対効果が妥当か評価する	320
目的達成や費用対効果の評価を誤らせる要因	321
27.2 自動テストの評価	323
27.3 フォールトインジェクションの必要性	324
27.4 ミューテーションテストによる評価	324
ミューテーションテストの基本的な進め方	324
ミューテーションテストの用途	326

第28章	自動テストの設計・実装の原則	327
28.1	プロダクトコードと同じように設計・実装を工夫する	327
	アンチパターン：キャプチャーリプレイ	327
28.2	FIRSTの原則	328
	Fast (すばやく実行できること)	329
	Independent (テスト間の結合性が低くテストが独立していること)	329
	Repeatable (再現可能であること)	330
	Self-validating (期待値比較まで一通りの処理をテストがカバーしていること)	330
	Timely (適時に実行できること)	330
28.3	「DRYではなくDAMP」の原則	331
	DRY原則	331
	DAMP原則	331
	「DRYではなくDAMP」の原則とは	331
28.4	xUTPの原則	332
	Write the Tests First (テストファーストでテストを書くこと)	332
	Design for Testability (テスト容易性のために設計すること)	332
	Use the Front Door First (正面のドアを使用すること)	332
	意図を伝える (Communicate Intent)	333
	テスト対象の改造を避ける (Don't Modify the SUT)	333
	テストの疎結合を保つ (Keep Tests Independent)	334
	テスト対象を切り出す (Isolate the SUT)	334
	テストの重複を最小化する (Minimize Test Overlap)	334
	テスト不能なコードを最小化する (Minimize Untestable Code)	335
	テストの処理をプロダクトコードに埋め込まない (Keep Test Logic out of Production Code)	335
	一つのテストで一つのテスト条件をテストする (Verify One Condition per Test)	335
	テストの気がかりを分けてテストする (Test Concerns Separately)	336
	適度な作業量・責務で対応する (Ensure Commensurate Effort and Responsibility)	336
第29章	自動テストコードのパターンやイディオム	337
29.1	パラメータ化テスト/データ駆動テスト	337
	パラメータ化テストの実装	337
	適用先	338
29.2	ドメイン特化テスト言語/Robotパターン/Page Objectパターン	339
	Robotパターン	339
	適用先	342
29.3	プロパティベーステスト/メタモルフィックテスト	343
	実例ベーステスト	343
	プロパティベーステスト	344
	プロパティベーステストと類似概念	346
29.4	テスト設計自動化におけるテストオラクル問題への対応	347
	対称性のある関係を活用する：ラウンドトリップテスト	347
	入出力の組の関係性を活用する：メタモルフィックテスト	348
	仕様モデルから期待値を得る：モデル駆動アプローチ	348
29.5	テストダブルのパターン	349
	テストダブルとは	349

	テストダブルの実装パターン	349
〔第30章〕	開発者テスト	353
30.1	開発者テストとは	353
30.2	保護して変更する (Cover & Modify)	353
30.3	リファクタリングのためのリグレッションテスト	354
30.4	仕様書としてのテスト (TaD) / 仕様化テスト	354
30.5	「テストがないコードはレガシーコードだ」	356
30.6	デバギングテスト	357
30.7	責任をもって自分のコードに対する自動テストを書く	357
	プログラマがテストコードを書いていることを保証する仕組み	358
〔第31章〕	テスト駆動開発	361
31.1	テスト駆動開発の概要	361
31.2	テスト駆動開発の進め方	361
	(1) 対象の振る舞いをテストリストとして整理する	361
	(2) テストファーストでコードを書く。必要に応じてリファクタリングを行う	362
31.3	テスト駆動開発の実例	363
	1 対象の振る舞いをテストリストとして整理する	363
	2 テストファーストでコードを書く	363
	3 リファクタリングでプロダクトコードを綺麗にする	364
31.4	テスト駆動開発のメリット	365
	仕様の理解と改善の促進	365
	コードの品質の向上	365
	すばやいフィードバックの実現	365
	テスト容易性の向上	366
	テストコードの充実	366
31.5	テスト駆動開発の適用先と前提条件	366
31.6	テスト駆動開発を持続可能にするための活動	367
	こまめなテストの設計・実装の改善	367
	CI/CDの導入	368
	アーキテクチャレベルでの設計品質の確保	368
31.7	テスト駆動開発を導入しやすくするための留意点	368
	テスト駆動開発は個人ではじめてよい	369
	全てテストファーストでなくてもよい。後からテストを書いてよい	369
	ユニットに対するテストでなくてもよい	370
	テスト駆動開発のテストは使い捨てでも良い	370
	テスト駆動開発の適用は一部だけでよい	371
31.8	テスト駆動開発のプラクティス	371
	二重のテストループ	371
	デュアルターゲットテスト	372
	Outside-In TDD	373

第32章	テスト計画	376
32.1	テスト計画とは	376
	テスト計画と開発計画の関係	377
	テスト計画の担い手	377
	テスト計画のドキュメンテーション	378
32.2	アンチパターン：テンプレート穴埋め計画	378
32.3	テスト計画のタイミングや順序	379
	フェーズごとのテスト計画	379
	テスト計画と関連活動の実行順序	380
	テスト計画の改善サイクルをまわす	380
32.4	テスト計画の作業	381
	【事前活動】テスト計画を準備する	381
	テストの前提と制約を調査する	382
	テストのステークホルダマネジメントを計画する	383
	テストの責務を計画する	383
	テスト活動のリスクを分析する	385
	テスト戦略を策定する	385
	プロセスを整備する	385
	組織体制を計画する	385
	テストの全体像を計画する	386
	テストのスケジュールと人員リソースを計画する	386
	テスト環境リソースを計画する	387
	予算を計画する	387
	テストのモニタリングとコントロールを計画する	387
32.5	テスト計画の不確実性をマネジメントする	388
32.6	技術やツールの実現性と持続可能性をマネジメントする	389
	テスト計画のためのフィジビリティのマネジメント	389
	テスト計画のための持続可能性のマネジメント	390
32.7	テスト計画のためにチームの能力をマネジメントする	391
32.8	シフトレフトによる平準化でテスト計画をサポートする	392
32.9	テストの見積もりアプローチ	393
	対象に応じた見積もりアプローチの使い分け	393
	見積もりの手法	394
	見積もりは同じ種類のプロジェクトの値をベースにする	395
	見積もりで迎合しない	395
	ネガティブなパターンを見積もる	396
32.10	アジャイル開発でのテスト計画	396
	アジャイル開発での計画のスコープ	396
	本当にアジャイルなのか	397
	アジャイル開発でのテスト計画の進め方	398

〔第33章〕	テストのモニタリングとコントロール	400
33.1	テストのモニタリング	400
33.2	事前のモニタリング設計	400
	モニタリング指標は目的ありき	401
	良い計画とプロセスづくりが良いモニタリングにつながる	401
	チームにとっての重要な指標：KPIを定めてチームを方向づける	401
	メカニズムを分析してモニタリング設計を行う	402
33.3	テストのモニタリングの要点	403
	三現主義でチームについて現実的に・具体的に知る	403
	グッドハートの法則に注意	403
	本質を見極める	404
33.4	テスト活動のモニタリング対象	404
	テストの作成のモニタリング	404
	テストの実行のモニタリング	406
	デバッグのモニタリング	407
	その他テストマネジメントに関するモニタリング	407
33.5	不適切なモニタリング指標	409
	バニティメトリクス	409
	演出型メトリクス	410
	バグ曲線（信頼度成長曲線）による予実管理の注意点	410
33.6	テストコントロール	411
33.7	テストのモニタリングとコントロールの仕組みの構築	411
33.8	クライシスマネジメントとしてのテストコントロール	412
	順序の依存性	412
	パーキンソンの法則	412
	テストの成果物の見えにくさ	412
33.9	クライシスにテストエンジニアとしてどう対応するか	413
	テストの十分性のコミットメントで妥協しない	413
	とにかく重大なバグを見つける	414
	健診テスト（全体を網羅する簡易的なリグレッションテスト）を継続実施する	414
	品質の低さを可能な限り早く可視化し全体に理解してもらう	415
	開発者に寄り添う。開発とテストを対立させない	415
33.10	プロダクト品質のモニタリング手法：健診テスト	415
	健診テストの目的	415
	健診テストのテスト内容	415
	健診テストの実行	416
	健診テストの適用先	416
〔第34章〕	プロジェクトリスクのマネジメント	417
34.1	テストコントロールとして実施するリスクマネジメント	417
	課題マネジメントとリスクマネジメント	417
	受動的リスクマネジメントと能動的リスクマネジメント	418
	いつ実施するか	418
34.2	ネガティブなプロジェクトリスクはチャンス	418

34.3	リスクマネジメントの活動	419
	リスク特定	419
	リスク分析・評価	420
	リスクコントロール	421
34.4	プロジェクトリスクのマネジメントの要点	422
	リスクの生の感触を把握する	422
	心理的安全性を確保する	422
	余力をもって重要なリスクに注力する	423
	重要なプロジェクトリスクの監視とコントロールに注力する	424
〔第35章〕	テストで求められる能力	425
35.1	テストで求められる能力	425
	テストエンジニアの能力の構成要素	425
35.2	テストを支えるソフトスキル	427
35.3	テストを支えるロール	427
	テスト活動のマネジメント	428
	技術リード	428
	テスト基本分析、テストアーキテクチャ全体の主導	428
	テストの作成と実行	428
	テストの自動化	429
35.4	テストに関わるスキルモデル/ロールモデル	429
	ISTQB/JSTQB	429
	総合的なソフトウェアエンジニアのスキルモデル	429
	Test.SSF	429
	体系的なテストに関わる知識体系	430
35.5	個の力と組織の力の両方が求められるテスト活動	430
〔第36章〕	テストを担う組織の構築	431
36.1	人材の採用	431
	(1) 採用の準備	431
	(2) 選考と採用	432
	(3) オンボーディングと人材採用の評価	433
	閑話：求職者の視点からみたテストエンジニアの採用市場	433
36.2	ロール横断による人材獲得	434
36.3	テストの適切なコミュニケーションを支える三本柱と心理的安全性	434
	コミュニケーションを支えるHRTの三本柱	435
	コミュニケーションを支える心理的安全性	435
36.4	テストに関わるチーム文化の形成	437
	テストにおける良いチーム文化の形成	437
36.5	テストを担う組織作りの方針	438
	プロジェクトの段階に応じたテスト専門化の調整	438
	大規模なテストに合わせた組織構築	439
36.6	チーム全体アプローチ	440

〔第37章〕	CI/CDの構築	442
37.1	テストを支えるCI/CDの価値	442
37.2	CI/CDとは	443
	継続的インテグレーション (CI) とは	443
	継続的デリバリー (CD) とは	444
	CI/CDとは	444
37.3	CI/CDを支える開発インフラ	444
	デプロイメントパイプラインの記述の柔軟性	444
	動的に追加・削除できる仮想環境への対応	445
	テストデータのセキュリティ確保	445
	持続可能性	445
37.4	CI/CDの導入	445
	(1) ビルドの自動化	445
	(2) インテグレーションとリリースの手作業の自動化	446
	(3) ユニットテストの組み込み	446
	(4) その他リリースに必要な活動の組み込み	446
37.5	CI/CDによるフェーズテスト方式の進化	446
〔第38章〕	バグ管理とバグチケット設計	448
38.1	バグ管理とバグ管理システム	448
	バグ管理システム	448
38.2	バグチケットの運用やフロー、フィールドの設計	449
	バグとは何かを定義する	449
	バグ管理システムを設計する	450
	バグチケットフローを設計する	450
	バグチケットフィールドを設計する	452
38.3	バグの分析	454
38.4	ODC分析	455
	ODC分析で使用する属性	455
	分析のアプローチ	455
〔第39章〕	テスト容易性の確保	457
39.1	テスト容易性とは	457
39.2	テスト容易性の重要性	458
39.3	テスト容易性の確保	459
	テスト容易性を低下させる要因の影響範囲を小さくし、分離・置換できるようにする	459
	結合度を低く、凝集度を高く	460
	テストにとって十分な観測点、制御点を設ける	460
	テスト容易性を拡張可能にする	460
	テストでとり得る条件を制限する	461
	品質特性のバランスを取る	461

	設計やコードをリーダブルに保つ	462
39.4	テスト容易性を拡張可能にする実装例	462
	対象コード	462
	テスト容易性を拡張可能にする	462
	テスト容易性を拡張してユニットテストを実現する	463
39.5	テスト容易性をいつ確保するか	464
	テスト分析の段階で要件を特定する	464
	早期からテスト対象を実行可能にすることを目指す	464
	フィードバックサイクルをまわして継続的に改善する	465
〔第40章〕	テスト設計を支えるモデリング	466
40.1	モデリングとは	466
	モデリングの目的	466
	モデルの記法	467
	モデリングの方向づけ	467
	モデリングは探索的	468
40.2	モデリングの立脚点となる観点分析	469
40.3	テスト設計を支えるモデリング	470
	(1) ステークホルダとの連携を支える	470
	(2) 仲間との協業を支える	471
	(3) 分析作業をサポートする	472
	(4) テスト対象を網羅可能な形式に整理する	473
	(5) モデルを使った有益なテストツールやテスト手法を適用可能にする	473
40.4	テスト設計におけるモデリングの活用例	473
	題材	474
	モデリングを活用したテスト設計手順	474
	題材に対するモデリングのまとめ	481
〔第41章〕	テストを支える契約による設計	482
41.1	契約による設計 / 責務設計とテスト	482
41.2	契約による設計とは	483
	事前条件・事後条件・不変条件	483
	契約による設計の具体例	483
	防御的プログラミングー辺倒の弊害	484
	アーキテクチャの境界に対して防御的プログラミングを徹底する	485
41.3	契約による設計で支えるテスト設計	485
41.4	表明による契約遵守の確認	486
	契約による設計とオブジェクト指向	487
〔第42章〕	ソースコードのブランチ管理とテストの連携	489
42.1	自動テストに影響するブランチ管理	489
42.2	CI/CD方針、テスト方針と連動させたブランチ管理方針	490
42.3	3分類ブランチの運用	491
	変更頻度高・維持品質高ブランチカテゴリ	491

変更頻度低・維持品質高ブランチカテゴリ	491
維持品質低ブランチカテゴリ	491
42.4 既存のブランチ戦略と3分類ブランチの関係性	491
トランクベース開発	492
GitLab-Flow	492
42.5 テスト活動をサポートするブランチ管理のプラクティス	493
設計とインテグレーションの工夫で「変更頻度高・維持品質高」のブランチを最小化する	493
CI/CDのテストを充実させて「変更頻度低・維持品質高」カテゴリを最小化する	493
メインブランチに対してテストを行い、メインブランチの品質を育てる	494
設計の工夫で、ブランチ作業に起因するプロダクトリスクを抑える	494
第43章 システムエンジニアリングで支えるテスト	495
43.1 システムとは	495
43.2 ソフトウェアからシステムに視野を広げる	495
(1) システム全体の品質の確認が必要	496
(2) システムレベルでないと実現できないテスト容易性がある	496
43.3 システムエンジニアリングのプロセス	496
システムエンジニアリングプロセスの特徴	497
43.4 システムエンジニアリングとして実施すべきテスト活動	499
システム全体で実現する包括的な品質に対するテスト (例: UI/UXテスト)	500
未知のリスクについてのテスト (例: セキュリティテスト)	500
サブシステムの品質の重ね合わせで実現する品質に対するテスト (例: 性能テスト)	500
43.5 テストを支えるシステムエンジニアリング活動	501
モジュール化の推進	501
テスト自動化容易性の改善	501
参考文献	502
索引	510