

はじめに..... iii

**第 1 部 開発プロセスと生産性** 1

プロダクト開発の5W1H..... 3  
 Why (なぜ開発するのか)..... 4  
 What (何を開発するのか)..... 6  
 Who、Where、When (誰がどこでいつまでに開発するのか)..... 8  
 How (どのように開発するのか)..... 9  
 プロダクトの継続的な成長への対応..... 11  
 デプロイやリリースの効率化..... 12  
 継続的な品質の確保..... 13  
 各章の内容..... 15  
 1章 Product Requirements Document..... 16  
 2章 Design Doc..... 16  
 3章 ブランチ・リリース戦略..... 16  
 4章 リアーキテクトにおけるテスト戦略..... 16

**第 1 章 Product Requirements Document** 17

1-1 Whatを書くPRD..... 18  
 チーム開発の難しさ..... 18  
 チーム参加者の職種の多様さ..... 19  
 コミュニケーションの難しさ..... 19  
 異なる認識によるブレ..... 20  
 合意形成が遅れることによる弊害..... 22  
 PRDとはプロダクトへの要求が書かれた文書..... 23  
 プロダクトへの要求を明確にする..... 24  
 関係者間の認識を統一する..... 25  
 プロダクトの品質を向上させる..... 25  
 後で振り返ることができる..... 26  
 PRDと似た文書..... 27  
 プレスリリース..... 27  
 インセプションデッキ..... 28  
 基本設計書と外部設計書..... 28  
 1-2 PRDに記載すべき内容..... 30  
 決まった章立てで書く..... 30  
 PRDの全体像を把握しやすくなる..... 30  
 チームメンバー間での合意形成がしやすくなる..... 30

PRDの改訂や更新がしやすくなる	31
代表的な章立て	31
「なぜ開発するのか (Why)」を含めてよいか	33
「概要」の章	33
「背景」の章	34
「製品原則」の章	37
「対象ユーザー」の章	38
「ユースケース」の章	41
「市場分析」の章	43
「競合分析」の章	45
「機能要求」の章	46
「その他の技術的要求」の章	49
「スコープ」の章	51
「KPI」の章	51
「リリーススケジュールおよびマイルストーン」の章	52
「マーケティング計画」の章	54
<b>1-3 運用時の注意点</b>	<b>56</b>
PRDの書き進め方	56
企画立案者によるドラフト版の執筆	57
マーケティングに関する執筆	59
機能に関する執筆	61
進め方に関する執筆	62
PRDを書く単位	63
既存のPRDを更新する方式	65
新規にPRDを作成していく方式	66
<b>1-4 まとめ</b>	<b>67</b>

## 第2章

# Design Doc

69

<b>2-1 Design Docとは</b>	<b>70</b>
コードレビューの限界	71
書くべきか、書かざるべきか	74
書くタイミング	76
何を使って書くか	77
ウォーターフォール開発の設計書との比較	79
<b>2-2 Design Docの構成</b>	<b>80</b>
構成の基本方針	81
タイトル	83
目次	84
目的セクション	84
背景セクション	86

設計の概要セクション	88
重要な要素：モジュール、データモデル、クラス、APIなど	88
設計の基本的なアイデア	92
簡略化したシーケンスやデータフロー	94
設計の詳細セクション	95
設計の説明の細分化	95
代替案の比較	97
APIなどの一覧	99
使い方の例示	99
計画セクション	100
その他の関心事セクション	101
Design Docのメタデータ	102
<b>2-3 運用上の注意点</b>	103
短いフィードバックサイクルを心がける	103
タスクごとに使い捨てる	104
ドキュメントへのリンクを活用する	104
Design Docの共通認識を持つ	105
Design Doc以外の選択肢を持つ	105
<b>2-4 まとめ</b>	107

## 第3章

# ブランチ・リリース戦略

109

<b>3-1 現代のソフトウェア開発におけるブランチ・リリース戦略の重要性</b>	113
<b>3-2 CI/CD（継続的インテグレーション、継続的デリバリー）</b>	115
CI/CDとは	115
DevOpsとの関係	115
代表的なCIの選択肢	116
代表的なCDの選択肢	117
デプロイ環境	117
CI/CD実践のための考え方	119
<b>3-3 ブランチ戦略</b>	119
ブランチの種類と役割の定義	119
main ブランチ	119
develop ブランチ	120
feature ブランチ	120
release ブランチ	120
代表的なブランチモデルの特徴と比較	120
GitFlow	121
GitLab Flow	122
GitHub Flow	123
Trunk Based Development	124

	Stacked Diff	127
	CI/CDとの関係	129
	GitFlow	129
	GitLab Flow	129
	GitHub Flow	129
	Trunk Based Development	129
	feature ブランチの扱い	130
	release ブランチの扱い	131
	ケーススタディ：モバイルアプリのブランチ間オートマージ	132
<b>3-4</b>	<b>リリースサイクル戦略</b>	134
	技術ドメインごとのリリースサイクル戦略	134
	バックエンドのリリース	135
	Webフロントエンドのリリース	135
	モバイルアプリのリリース	136
	最近のリリース戦略の傾向	136
	任意のタイミングでの頻繁なリリース	136
	定期的なリリースサイクル	136
	ケーススタディ：モバイルアプリのリリースサイクルアップデート	138
<b>3-5</b>	<b>リポジトリ戦略</b>	139
	Polyrepo	139
	Monorepo	141
	選択にあたっての論点	143
	分散管理と集中管理	143
	依存関係の解決	143
	アクセス制御とセキュリティ対策	144
	ケーススタディ：Monorepo ベースの開発体験	145
<b>3-6</b>	<b>フィーチャーフラグの活用</b>	146
	フィーチャーフラグの実装	147
	静的または動的なフィーチャーフラグ	147
	自前のフィーチャーフラグシステム	148
	サードパーティのフィーチャーフラグシステム	148
	フィーチャーフラグのクリーンアップ	149
	フィーチャーフラグの用途	149
	実装中の機能のガード	150
	A/Bテスト	150
	キルスイッチ	150
	特定のユーザーグループに向けた機能のロールアウト	150
	パーセンテージベースのロールアウト	150
	ベータリリース	150
	ブランチ戦略との関連性	151
<b>3-7</b>	<b>まとめ</b>	152
<b>3-8</b>	<b>参考文献</b>	153

4-1	リアーキテクトにおけるテスト	157
	検証と妥当性確認	157
	仕様の把握およびその背景に対する理解を深める	157
	要件を満たしているか、ステークホルダーの期待に沿うか	157
	限られた開発期間におけるテスト	158
	選択的テスト手法による開発期間の有効活用	159
	リグレッションテストの活用	159
4-2	リアーキテクトにおけるテスト戦略例	160
	リアーキテクトにおけるテスト戦略	160
	各工程詳細	162
	テスト計画～テスト設計	162
	機能ごとの優先度算出	163
	優先度に基づくテスト分析の繰り返しとテスト設計への反映	165
	テスト実装～テスト実行	166
	リアーキテクトにおけるテストの課題への対応	166
4-3	ケーススタディ：出前館アプリリアーキテクトにおけるテスト	168
	出前館におけるソフトウェア開発	168
	プロダクトマネジメント	168
	開発プロセス	170
	実践結果	171
	2023年5月～ 既存アプリへの機能追加	172
	2023年6月 テスト対象の優先度算出	173
	2023年7月～8月 優先度に基づくテスト分析の繰り返し	174
	2023年9月～ テスト実行	174
	2023年11月 不具合の収束と段階的リリースの実施	176
	注意点	177
4-4	まとめ	177
4-5	参考文献	179

	ソフトウェアエンジニアリング組織として取り組むべき課題	182
	ソフトウェアエンジニアの採用とスキル向上	182
	開発プロセスの標準化と組織文化の改善	183
	誰がどう課題に取り組むか	184
	1つの開発チーム内での取り組み	184
	専門のグループによる活動	184

トップダウンによる意思決定	185
各章の内容	186
5章 実践エンジニア組織づくり	186
6章 エンジニアリングイネーブルメント	186
7章 開発基盤の改善と開発者生産性の向上	186

第5章

実践エンジニア組織づくり

187

5-1	なぜエンジニア組織を作ることになったか	188
	筆者がやることになった経緯	188
	なぜ社内で開発したいのか	189
	社員として採用する必要があるのか	191
	メガベンチャーから転職したらエンジニアが10人と少しの会社	193
	社内でサービス開発ができるようになるには何をすればよいか	194
5-2	兎にも角にも採用	195
	エンジニアフレンドリーな会社にするために	195
	専門職向けに等級・評価・報酬制度をつくる	196
	給与設計をする	198
	採用ページの作成	199
	中途採用と新卒採用	199
	中途採用	199
	新卒採用	201
	魅惑の未経験者採用	202
	エンジニアだけ増えてもよくない	203
5-3	定着と育成	203
	オンボーディングは大事	204
	ブートキャンプチーム	204
	定着のためにできそうなこと	205
	やりたそうな仕事を渡す	205
	新しいことをしている	206
	自分よりすごい人がいる	207
	育成について	207
	技術書を読みやすく。自己研鑽だけに頼らない	207
	入社時の選書	208
	実は優秀な人は勝手に育ってくれる（そして羽ばたいていく）	209
5-4	開発と運用どっちもやる	210
	「実装だけをする開発組織」からの脱却	210
	運用支援ツールの導入	211
	技術的負債の返済は専任チームでやるべきか	213
	DevOps/SRE/プラットフォームエンジニアリング	214

5-5	チームで動き、スケールさせる	216
	なぜ少人数チームを単位にするか、なぜチームの大きさはピザ2枚か	218
	いわゆる「大規模アジャイル」をいくつか	219
	フィーチャーチームかコンポーネントチームか	221
	アーキテクチャと組織構造、コンウェイと逆コンウェイの話	222
	全貌は誰が見るか。分割統治をしすぎると……	223
	コミュニケーションは大事。エスパーではない	225
5-6	まとめ	226

第 6 章

エンジニアリングイネーブルメント

227

6-1	エンジニアリングイネーブルメントとは	228
	エンジニアリングイネーブルメントが目指す姿	228
	エンジニアリングイネーブルメントが必要な理由	230
6-2	能力向上と成果向上	230
	能力向上と成果向上のためにチームを分化	231
6-3	エンジニア職以外でのイネーブルメントの活動	232
6-4	エンジニアリングイネーブルメントの活動の進め方	233
	エンジニアリングイネーブルメントの活動の外観	233
	開発プロセスの全体像の定義	233
	能力向上施策の定義と実施	233
	成果向上施策の定義と実施	234
6-5	開発プロセスの全体像	234
	開発ステップの定義	235
	開発プロセスの成功ファクターの定義	236
	計画フェーズ	237
	設計フェーズ	237
	開発フェーズ	238
	リリースフェーズ	238
	運用フェーズ	239
	開発プロセスの見直し	239
6-6	能力向上のための仕組み	239
	スキルマップの作成	240
	スキルの抜き出し	240
	スキルレベルの定義	241
	チームのスキルの把握	244
	スキルレベル表の公開	245
	スキルマップに基づいた能力向上施策	246
	勉強会・研修	246

ヘアプロ・モブプロ	247
研修費補助・資格取得費補助	247
イネープリングチームの組成	248
<b>6-7 成果向上のための施策</b>	248
アウトカムを高めることが成果向上に必要ではなかったのか?	249
木こりのジレンマ	250
ドキュメントの整備	250
ドキュメントの作成	250
ドキュメントの更新	251
ドキュメントの整理	252
オンボーディングプロセスの整備	253
オンボーディングプロセスの整備	253
オフボーディングプロセスの整備	255
ツールの整備	256
プラットフォームエンジニアリングチームの組成	256
<b>6-8 エンジニアリングイネーブルメントは誰がやるべきか</b>	257
<b>6-9 まとめ</b>	258

第 7 章

開発基盤の改善と  
開発者生産性の向上

259

<b>7-1 開発基盤の改善とは</b>	261
<b>7-2 誰が基盤を開発するのか</b>	261
小規模なチームでの開発基盤	262
大規模なチームでの開発基盤	262
<b>7-3 開発者生産性とは、価値提供の速度を高めること</b>	264
<b>7-4 開発基盤の改善の様々な取り組み</b>	264
実装、価値検証のサイクルを高速化する	264
オペレーション業務を省力化する	265
開発上の迷いや不安点を減らす	266
<b>7-5 開発効率の改善のサイクルと全体像</b>	267
1. 抽象課題の発見	267
2. 課題の具体化と改善策の発見	268
3. メトリクスを使ったベンチマークの考案と検証	268
4. 改善の実行	268
<b>7-6 抽象課題の発見 (Step 1)</b>	269
プロダクトコードとの接触を増やす	269
コードレビューを通した課題発見	269
可能な限り自分で実装する機会を持つ	270



プロダクト開発者との対話を通して課題を発見する	270
プロダクト開発者との交流を仕組み化する	271
他チームの動向を把握する	271
<b>7-7 課題の具体化と改善策の発見 (Step 2)</b>	272
抽象課題：CIの実行時間が長い	272
課題の具体化のアイデアを出すには	273
一次情報と技術トレンドを追う	274
<b>7-8 メトリクスを使ったベンチマークの考案と検証 (Step 3)</b>	275
開発者生産性とメトリクス	276
定量化できないメトリクス	276
生産性を定義するメトリクスの考案	277
Four Keys	277
SPACEフレームワーク	278
モバイルアプリ開発の生産性をSPACEフレームワークで定義する	280
モバイルアプリのビルドにまつわる課題	280
Satisfaction & Well-Being	280
Performance	281
Activity	281
Communication & Collaboration	282
Efficiency & Flow	282
メトリクスを多角的に評価する	283
ビルドの待ち時間を減らしたい	284
ビルドについての質問を減らしたい	284
SPACEを使ったメトリクスの評価	285
開発者生産性の定義のまとめ	285
<b>7-9 改善の実行 (Step 4)</b>	286
開発基盤改善のアンチパターン	286
アンチパターン1：特定領域への理解が薄れてしまう	286
解決策1：プロダクトコードや手薄な部分への理解を持ち、抽象課題を発見し続ける	286
アンチパターン2：定量化が局所最適を起こしてしまう	287
解決策2：複合的なメトリクスを設定する	288
アンチパターン3：改善を優先して開発を止めてしまう	288
解決策3：前方互換を重視した移行計画を設計する	289
アンチパターン4：過剰に規約や規律、正当性を重視してしまう	291
解決策4：落としどころを明確にする	291
他の開発者と協調していくために	292
心理的に相談してもらいやすい環境を作る	292
小まめな状況の共有	293
夢を語る	293
<b>7-10 まとめ</b>	294
索引	295
著者プロフィール	299