

〔増補改訂〕

# 良いコード を書く技術

読みやすく保守しやすいプログラミング作法

Agata Toshitaka

縣俊貴

〔著〕

技術評論社

●初出

本書は、小社刊『WEB+DB PRESS』Vol.44～Vol.49の連載「良いコードへの道——普通のプログラマのためのステップアップガイド」をもとに、大幅に加筆と修正を行い書籍化したものです。

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた運用は、必ずお客様自身の責任と判断によって行ってください。これらの情報の運用の結果について、技術評論社および著者はいかなる責任も負いません。

本書の情報は第1刷発行時点のものを記載していますので、ご利用時には変更されている場合があります。

本書に記載されている会社名・製品名は、一般に各社の登録商標または商標です。本書中では、™、©、®マークなどは表示していません。

上記をご承諾いただいたうえで、本書をご利用願います。これらの注意事項をお読みいただくずにお問い合わせいただいても、著者・出版社は対処しかねます。あらかじめ、ご承知おきください。

支えてくれたみなさんへ

## はじめに

あなたはプログラマーですか？ もしもプログラマーなら何のためにプログラムを書いていますか？ 生活のため、それとも楽しいからでしょうか。理由は人それぞれでしょう。だけど、もしあなたがプログラマーなら、「良いコード」が書けるようになることで仕事をもっと楽しくなるはずです。

本書は良い仕事をしたい普通のプログラマーが、読みやすく保守しやすい良いコードを書けるようになるための解説書です。ここでの普通のプログラマーとは、初級プログラマーや初級から中級になりかけのプログラマーのことを指します。私が考える良いコードの具体例は本書の中で解説していきますが、一番大切なのは「良い仕事をしたい」「良いソフトウェアを作りたい」という気持ちです。この本を手にとってくれたあなたの中にも、きっとそういう気持ちがあると思います。

良い仕事をすると仕事にやりがいを感じられるようになる、つまり仕事が楽しくなります。そして職人として仕事に誇りが持てるようになります。良い仕事をするプログラマーは仲間や顧客からの信頼が厚くなり、仕事を依頼される機会も増えていきます。ということは収入もアップすることでしょう。

と、なれば理想ですが、実際には職人として良いコードが書けるプログラマーは、あくまでも私の実感ですが業界の2割ぐらいしかいません。あとの8割はいわゆる普通のプログラマーです。その中には、良い仕事をしたいと思っているものの、なかなかそのコツをつかめなかったり、何から手を付けてよいかわからない人が多いようです。

プログラミングはとても創造的でエキサイティングで楽しいものなのに、これではもったいないです。たとえば、泳ぎ方がよくわからないままプールに行ってもおもしろくありません。むしろ苦痛なぐらいです。しかし、25メートル、50メートルと泳げるようになるとだんだん楽しくなります。さらに思いのまま泳げるようになると、快感へと変わっていきます。

私が良いコードを書けるようになったのは、良い師匠や書籍と巡り合えたおかげです。本書が良い仕事をしたい普通のプログラマーのみなさんの道しるべとなることを願っています。

2021年4月 縣 俊貴

## 謝辞

本書は多くの方から支えていただいて執筆することができました。

まず、地元福岡のプログラマー仲間である桜井雅史さん、上津竜太郎さん、きしだなおきさんには、プログラミングや設計、教育や勉強会についてたくさんの議論をしていただきました。その議論での気づきが本書にふんだんに盛り込まれています。山本竜三さん、染田貴志さん、南里範子さん、鳥谷部昇さん、城和典さんには、校正時に地道な作業を手伝ってもらい、たいへん助けていただきました。本書に彩りを添えている素敵なプログラマーのイラストは、高田美奈子さんに描いていただきました。馬場保幸さんからは、オープンソースのWebアプリケーションフレームワークCubbyと一緒に開発していく中で、本書のヒントをいただきました。株式会社ヌーラボのみなさんからは、日々のシステム開発、サービスの運営を通して、ソフトウェア技術についてあらゆることを教わりました。この場を借りてみなさんに感謝の言葉を贈ります。

遅筆な筆者を時には励まし、時には尻を叩いて支えてくれた技術評論社の稲尾尚徳さんには、どれだけ感謝の言葉を言っても足りないぐらいです。本当にありがとうございました。

## 増補改訂での更新点

増補改訂では、コード例を執筆時点で最新のJava 15に対応しました。古いライブラリやフレームワークも、すべて新しいものに置き換えています。

また、良いコードの実現に重要なデータ構造についての理解を深めるために、第7章「データ構造」を書き下ろしました。加えて第11章「メタプログラミング」では、メタプログラミングの題材を、外部DSLから、より導入が手軽で保守性が高い内部DSLに変更しました。これらの加筆、改訂により、より実践で役立つ内容となっています。

### コード例について

本書には多くのコードが例として登場します。コード例で使用する言語は基本的にJavaです。RubyやJavaScript、TypeScriptなどほかの言語を使用する場合のみ、本文中にその言語名を明記します。ただし、言語に特化した話はあまりありませんので、ほかの言語の人でも十分に参考になる構成になっています。

コード例はサポートページからダウンロードできます。このコードはパブリックドメインですので、仕事や趣味のプログラムに自由に利用していただいてもかまいません。

### サポートページ

本書のサンプルコードは、下記サポートページから入手できます。正誤情報なども掲載します。

**URL** <https://gihyo.jp/book/2021/978-4-297-12048-1/support>

[増補改訂] **良いコードを書く技術** 読みやすく保守しやすいプログラミング作法 ● 目次

はじめに .....	iv
謝辞 .....	v
増補改訂での更新点 .....	vi
コード例について .....	vi
サポートページ .....	vi
目次 .....	vii

## 第 1 章

**良いコードとは何か** ..... 1

## 1.1 良いコードの定義と価値 ..... 2

## 1.2 良いコードの定義 ..... 2

保守性が高い ..... 2

すばやく効率的に動作する ..... 3

正確に動作する ..... 3

無駄な部分がない ..... 4

## 1.3 良いコードの価値 ..... 4

プロジェクトを強かに推し進める ..... 4

プログラマーとしての評価が高まる ..... 5

仕事に満足感や自信が持てるようになる ..... 5

## 1.4 代表者の声 ..... 6

良い仕事をしたい普通のプログラマー ..... 6

達人プログラマーを目指す中級プログラマー ..... 6

達人プログラマー ..... 6

## 1.5 まとめ ..... 7

## 第 2 章

**良いコードを書くための5つの習慣** ..... 9

## 2.1 良いコードは1日にしてならず ..... 10

## 2.2 習慣その1 読む コードを読んで読んで、読みまくれ! ..... 10

	<b>Column</b> GitHubでコードの海を泳ぐ.....	11
<b>2.3</b>	<b>習慣その2 書く</b> とにかくコードを書こう.....	11
	<b>Column</b> 1人でプログラムを書けますか? .....	12
<b>2.4</b>	<b>習慣その3 道具を磨く</b> 使う道具は常に磨いておこう.....	13
	エディタ/統合開発環境.....	13
	自動化.....	13
	バージョン管理ツール.....	13
	UNIX/Linux/macOS .....	14
	<b>Column</b> 良い道具に乗り換える.....	14
<b>2.5</b>	<b>習慣その4 知る</b> 良い知識を得よう .....	15
	書籍 原典とHow To本の2冊買いがお勧め.....	15
	リファレンスや仕様書などのドキュメント.....	16
	Webサイト.....	16
<b>2.6</b>	<b>習慣その5 聞く</b>	
	アウトプットと人からのフィードバックでさらなる成長を.....	17
	コードレビューを受ける.....	17
	ブログを書く.....	18
	コミュニティや勉強会に参加する.....	18
	成果を発表する.....	18
<b>2.7</b>	<b>まとめ</b> .....	18

## 第3章

### 名前付け.....19

<b>3.1</b>	<b>良いコードは良い名前から生まれる</b> .....	20
<b>3.2</b>	<b>代表者の声</b> .....	20
	良い仕事をしたい普通のプログラマー.....	20
	達人プログラマーを目指す中級プログラマー.....	21
	達人プログラマー.....	21
<b>3.3</b>	<b>良い名前の条件</b> .....	21
	説明的で意味・意図を表している.....	21
	省略のコツ.....	22
	一貫性がある.....	23



英語で付けられている .....	23
オレオレ単語を作らない .....	24
スペルミスに注意する .....	24
誤訳に注意する .....	24
自信がないときは .....	24
イディオムに従っている .....	25
コーディング標準に従っている .....	25
Column 名前の流行り廃り .....	26
<b>3.4 変数名</b> .....	27
基本は説明的な名前を付ける .....	27
グローバル変数、クラス変数、インスタンス変数 意味を正しく表現する .....	28
Column 変数の種類 .....	28
ローカル変数 スコープの長さで使い分け .....	31
イディオムに従う／従わない .....	31
変数名を短くしたほうが可読性が向上する場合もある .....	32
メソッドの引数名 わかりやすく簡潔に .....	33
<b>3.5 メソッド名</b> .....	34
Column メソッドの種類 .....	35
<b>3.6 クラス名</b> .....	36
クラス名のボキャブラリは経験とともに高まる .....	36
<b>3.7 パッケージ／ネームスペース名</b> .....	37
<b>3.8 プロジェクト名</b> .....	38
<b>3.9 まとめ</b> .....	39
Column あるWebアプリケーションフレームワークでのクラス名の変遷 .....	40

## 第4章

### スコープ .....

4.1 スコープを意識していますか? .....	42
4.2 スコープって何? .....	42
4.3 スコープを小さくして覚えておくことを減らそう! .....	43

4.4	代表者の声	44
	良い仕事をしたい普通のプログラマー	44
	達人プログラマーを目指す中級プログラマー	44
	達人プログラマー	44
4.5	変数のスコープ	45
	ローカル変数	45
	変数は使用する直前で宣言する	47
	メソッドに抽出する	47
	イテレータの一時変数のスコープをループ内に閉じ込める	48
	代入されない変数にはfinalを付ける	48
	Column JavaScriptの奇妙なスコープ	49
	インスタンス変数	50
	クラス変数	51
4.6	メソッドのスコープ	53
	インスタンスメソッド	54
	クラスメソッド	54
	メソッドの引数の情報量	55
4.7	クラスのスコープ	56
	インナークラス	58
	無名クラス	59
4.8	キャストを使用した可視性の制御	60
4.9	より大きな粒度のスコープ	61
4.10	まとめ	62

## 第5章

# コードの分割

5.1	適切な長さにコードを分割する	64
5.2	なぜコードを分割するのか	64
	可読性が向上する	64
	保守性が向上する	64
	再利用性が向上する	65

5.3	<b>代表者の声</b> .....	65
	良い仕事をしたい普通のプログラマー.....	65
	達人プログラマーを目指す中級プログラマー.....	65
	達人プログラマー.....	65
5.4	<b>2つの方向からの分割</b> .....	66
	トップダウン方式.....	66
	ボトムアップ方式.....	68
5.5	<b>お題</b> <b>クライアントにXMLを返すWeb APIの処理を分割する</b> .....	70
5.6	<b>ステップ1 ベタなコードで書いてみる</b> .....	71
5.7	<b>ステップ2 共通処理をメソッドに抽出して分割する</b> .....	73
	考察 どこまで共通化を行えばいいの?.....	74
5.8	<b>ステップ3 処理単位で分割する</b> .....	76
	考察 制御構造と処理の分け方.....	77
	ループと、ループの中の処理.....	77
	if文と、その中の処理.....	79
	try~catch~finallyと、その中の処理.....	79
5.9	<b>ステップ4</b> <b>状態を持つ処理をクラスに抽出して分割する</b> .....	80
	考察 インナークラスとして実装しているのはなぜ?.....	83
	実際に使用される箇所のすぐ近くに	
	クラスがあるのでわかりやすい.....	83
	もとのクラスとの依存関係をstaticで制御できる.....	83
	新しいファイルを増やさなくてよい.....	83
	考察 必要なデータをコンストラクタで渡しているのはなぜ?.....	84
	考察 インスタンス変数として保持したほうがよいものは何?.....	84
	考察 さらにリファクタリングしてみる.....	85
5.10	<b>まとめ</b> .....	86
<b>第6章</b>		
<b>コードの集約</b> .....		87
6.1	<b>コードの重複は悪</b> .....	88

6.2	代表者の声	88
	良い仕事をしたい普通のプログラマー	88
	達人プログラマーを目指す中級プログラマー	88
	達人プログラマー	89
6.3	メソッドに抽出してまとめる	89
6.4	継承でまとめる	90
	継承でのまとめ方には注意	91
	親クラスが肥大する	91
	単一継承の制限	92
6.5	ユーティリティクラスにまとめる	92
	Javaではstaticインポートと組み合わせると便利	94
	ユーティリティクラスの名前付け	94
6.6	サービス層にまとめる	94
6.7	オブジェクトにまとめる	95
6.8	定数にまとめる	97
6.9	列挙型 (enum) にまとめる	98
6.10	まとめ	99
	(Column) まとめすぎにご用心	99

## 第7章

# データ構造 101

7.1	データ構造で勝負が決まる	102
7.2	代表者の声	102
	良い仕事をしたい普通のプログラマー	102
	達人プログラマーを目指す中級プログラマー	102
	達人プログラマー	103
7.3	データ構造とは?	103
	日付のデータ構造の例	104
	年月日をそれぞれの値で保持する場合	104
	年月日を配列で保持する場合	105

年月日をオブジェクトのプロパティとして保持する場合 .....	106
複数のメールアドレスのデータ構造の例 .....	106
すべてのメールアドレスを配列で保持する場合 .....	106
メインとサブのメールアドレスを別のプロパティで保持する場合 ..	107
<b>7.4 データ構造の指針</b> .....	108
仕様や制約が明確 .....	108
処理するコードが書きやすい .....	109
パフォーマンスが劣化しにくい .....	110
<b>7.5 お題 美容室の予約画面のHTMLを出力する</b> .....	111
<b>7.6 ステップ1</b> データベースのデータ構造をそのまま利用する .....	113
<b>7.7 ステップ2 処理に最適なデータ構造を把握する</b> .....	115
<b>7.8 ステップ3 最適なデータ構造に変換して利用する</b> .....	116
考察 データの変換処理とHTMLの出力処理を分ける .....	119
考察 効率的なデータ変換の方法 .....	119
考察 クラスを使うとどうなる? .....	120
<b>7.9 まとめ</b> .....	121
<b>Column</b> データ構造はどこにでも存在している .....	121

## 第8章

<b>コードのパフォーマンス</b> .....	123
<b>8.1 パフォーマンスを意識していますか?</b> .....	124
<b>8.2 代表者の声</b> .....	125
良い仕事をしたい普通のプログラマー .....	125
達人プログラマーを目指す中級プログラマー .....	125
達人プログラマー .....	125
<b>8.3 パフォーマンスは計算量で決まる</b> .....	126
アルゴリズムの選択で変わる .....	126
クラスの選択で変わる .....	129
ライブラリの使い方 で変わる .....	131
<b>8.4 パフォーマンスチューニングの手順</b> .....	133

①	まずは測定する.....	133
	ログ出力による測定.....	134
	プロファイラによる測定.....	134
②	原因を特定する.....	135
③	チューニングする.....	135
④	チューニング結果を測定する.....	135
8.5	アルゴリズムの選択以外のパフォーマンスチューニング.....	136
	SQLやテーブル設計を変更する.....	136
	キャッシュする.....	136
	インフラを強化する.....	138
8.6	パフォーマンスチューニングの指針.....	138
	どのタイミングでチューニングするのがベストなの?.....	138
	適切な量のテストデータを用意しよう.....	139
	常にパフォーマンスを意識しよう.....	139
8.7	まとめ.....	140

## 第9章

# ユニットテスト.....141

9.1	テストはお好きですか?.....	142
9.2	ユニットテストって何?.....	142
9.3	代表者の声.....	144
	良い仕事をしたい普通のプログラマー.....	144
	達人プログラマーを目指す中級プログラマー.....	144
	達人プログラマー.....	144
9.4	ユニットテストの効能.....	145
	網羅的なテストを自動化できる.....	145
	回帰テストによりコードが壊れていないことを保証できる.....	146
	設計の改善につながる.....	146
9.5	お題 Webアプリケーションのセキュリティテスト.....	146
9.6	ステップ1 データベースにテストデータを登録する.....	147
9.7	ステップ2 画面の実装.....	147

9.8	ステップ3 画面のユニットテスト (正常系)	148
9.9	ステップ4 画面のユニットテスト (異常系)	150
9.10	ユニットテストの指針	151
	テストのポリシーを決めておこう	151
	テストしやすい部分はどこ?	151
	テストしにくい部分はどうする?	151
	初期データのセットアップ	152
	モックオブジェクト	152
9.11	まとめ	155
	(Column) 言語別のテストフレームワーク	156

## 第 10 章

### 抽象化

10.1	抽象化がプログラミングのパワーを最大化する	158
10.2	配列／リストって何?	158
10.3	配列／リストを利用した抽象化とは?	160
10.4	代表者の声	161
	良い仕事をしたい普通のプログラマー	161
	達人プログラマーを目指す中級プログラマー	161
	達人プログラマー	162
10.5	お題 画像ファイルの一覧を表示するWebアプリケーション	162
10.6	ステップ1 ベタなコードで書いてみる	164
10.7	ステップ2 可読性を高めるためのメソッド抽出	167
10.8	ステップ3 関連するデータのデータ構造を整理	170
10.9	ステップ4 配列／リスト化して抽象化	172
10.10	抽象化の指針	174
	どのタイミングで抽象化するのがベストなの?	174

コードの重複をまとめるな!?	174
これって単なるループなんじゃない?	175
10.11 まとめ	175
第 11 章	
<b>メタプログラミング</b>	177
11.1 プログラミングをプログラミングする	178
11.2 代表者の声	178
良い仕事をしたい普通のプログラマー	178
達人プログラマーを目指す中級プログラマー	178
達人プログラマー	179
11.3 メタプログラミングって何?	179
コードの自動生成	179
DSL	180
外部DSL	180
内部DSL	182
(Column) 流れるようなインタフェースとstaticインポートによる内部DSL	184
11.4 お題 Javaコードを使った内部DSL	185
固定長電文解析処理をDSLで処理する	185
11.5 ステップ1 ベタなコードで書いてみる	186
考察 インナークラスを使っているのはなぜ?	186
11.6 ステップ2 メタデータを内部DSLに移動する	188
考察 メタデータって何?	191
11.7 ステップ3 変換ルールに対応する	191
ストラテジパターン	194
考察 複数の変換パターンを適用したい場合は?	194
考察 変換ルールに引数を追加したい場合は?	195
考察 DSLの構文を改善するには?	196
コンストラクタをなくす	197
もっとDSLっぽくする	197
考察 DSLのテストはどうする?	198
考察 DSLのデバッグはどうする?	198



考察	これはフレームワーク? それともDSL?.....	199
11.8	まとめ.....	199
第12章		
	<b>フレームワークを作ろう</b> .....	201
12.1	フレームワークの動作原理を知る.....	202
12.2	代表者の声.....	202
	良い仕事をしたい普通のプログラマー.....	202
	達人プログラマーを目指す中級プログラマー.....	203
	達人プログラマー.....	203
12.3	お題 Webアプリケーションフレームワークを作ろう.....	203
12.4	ステップ1 素のサーブレットで書いてみる.....	205
12.5	ステップ2 フロントコントローラとアクションクラスの導入.....	208
12.6	ステップ3 ルーティング情報の外部ファイル化.....	213
	(Column) JavaのリフレクションAPI.....	215
12.7	ステップ4 よく使う処理を 簡単に実行できるように共通化する.....	216
	考察 これからさらにフレームワークを拡張するには?.....	219
	クエリパラメータの自動バインディング.....	219
	クエリパラメータの入力検証.....	220
	その他.....	221
12.8	ステップ5 フレームワークをパッケージ化する.....	221
	①フレームワークのコードとアプリケーションコードを分ける.....	221
	②パッケージ化する.....	222
	③バージョンを付けて管理する.....	222
12.9	まとめ.....	223

付録 **A**

<b>コードリーディングの方法</b> .....	226
A.1 コードには動的な読み方と静的な読み方がある.....	226
A.2 お題 Apache Commons IOのコードを読む.....	226
A.3 <b>ステップ1</b> 対象のコードをダウンロード(チェックアウト)する.....	226
A.4 <b>ステップ2 静的な方法でコードを読む</b> .....	228
検索コマンドを使う.....	231
統合開発環境を使う.....	232
A.5 <b>ステップ3 動的な方法でコードを読む</b> .....	232
デバッガで実行する.....	232
ユニットテストを実行する.....	233
コードを修正して実行する.....	234
部分的なコードをコピーして使ってみる.....	234
A.6 <b>まとめ</b> .....	235

付録 **B**

<b>解説付き参考文献</b> .....	236
あとがき.....	239
索引.....	240
著者プロフィール.....	246

# 第 1 章

## 良いコードとは何か

## 1.1

# 良いコードの定義と価値

良いコードとはいったいどのようなものでしょうか。良いコードを書くとは何か、何が出来るのでしょうか。そもそも良いコードを書く必要があるのでしょうか。

本章では序論として、「良いコード」がどのようなものか定義し、その価値について考えていきます。

## 1.2

# 良いコードの定義

一口に良いコードと言っても、組織やプロジェクト、プログラマーか管理者かなど、状況が異なると定義も変わってきます。本書では次の4つを満たすものを良いコードと定義します。

- 保守性が高い
- すばやく効率的に動作する
- 正確に動作する
- 無駄な部分がない

定義した良いコードを実現するために必要な知識を、本書では図1.1の構成で解説していきます。

### 保守性が高い

私たちが書いたコードは、私たちが想像するよりも長く利用されます。あとから見て何をやっているのか理解不能なコードは良いコードとは言えません。将来の自分は記憶力において他人と同然です。つまり、他人が見て理解できるコードであれば、将来の自分が見ても理解できる良いコードであると言えます。

本書では第3章「名前付け」にて、変数やメソッドの適切な名前の付け方を

解説します。続く第4章「スコープ」にて、依存性が低く保守性が高いコードの書き方を紹介します。そして第5章「コードの分割」と第6章「コードの集約」、第7章「データ構造」にて、長いコードの分割や重複コードの集約、データ構造の改善を行うことで、読みやすく保守しやすいコードを実現します。

### すばやく効率的に動作する

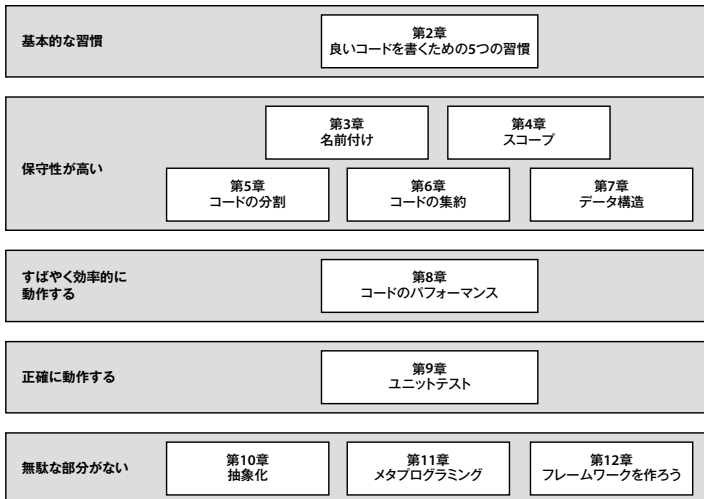
似たような実装がいくつか考えられるとき、あきらかに効率の悪いものを選択する必要はありません。良いコードは適切なパフォーマンスで動作します。

本書では第8章「コードのパフォーマンス」で、すばやく効率的に動作するコードについて解説します。

### 正確に動作する

確実に動作し、信頼性が高いことは良いコードの条件です。「防御的プログラミング」という言葉がありますが、これは「正常な値が来るはず」という

図 1.1 本書の構成



決め付けをせずに、不正な値が来ても被害を受けないように防御的にプログラミングを行うことです。良いコードは防御的で、不測のバグを生み出しにくい作りになっています。

本書では第9章「ユニットテスト」で、バグが少なく正確に動作するコードを実現するために、テストの自動化について解説します。

### 無駄な部分がない

無駄がないコードは理解するのも修正するのも簡単で時間がかからないため、良いコードと言えます。コード内に繰り返し現れるパターンを劇的に短くする方法として抽象化があります。

本書では第10章「抽象化」で簡単な抽象化について触れ、第11章「メタプログラミング」でより高度な抽象化に挑戦します。第12章「フレームワークを作ろう」では簡単なフレームワークの作成を通してフレームワークの動作原理を知り、本格的な抽象化と柔軟で再利用可能なフレームワーク作成のコツを学びます。



これらの定義を満たす良いコードを書くためには、基本的な日々の習慣を身に付けている必要があります。そこで第2章では、「良いコードを書くための5つの習慣」を紹介します。

## 1.3

### 良いコードの価値

では、私たち開発者が良いコードを書けるようになると、具体的にどんなメリットがあるのでしょうか。

### プロジェクトを強力に推し進める

良いコードは、プロジェクトを推し進めて成功へと導くための基本的な

要素となります。

達人たちは、シンプルで保守性が高く、安定したコードを、ものすごいスピードで書き上げていきます。場合によっては、単純作業を自作のDSL (Domain Specific Language、ドメイン特化言語)<sup>注1</sup>に置き換えたり、テストが難しいレガシーなコードをテスト可能で検証できるコードに変更したりすることで、品質や生産性を数百倍に高めることさえあります(おおげさではなく、本当に数百倍の場合もあるのです!)

これはプロジェクトの成功にとって大きなアドバンテージです。もちろん、良いコードがあれば必ずプロジェクトが成功するわけではありません。実際は、開発プロセスやマネジメント、コミュニケーションなどほかの要素により左右されることも多いのですが、それを差し引いたとしても良いコードの持つ力は大きいと言えます。

### プログラマーとしての評価が高まる

良いコードを書くプログラマーは、総じてプログラマーとして信頼され、評価されます。プログラマーとしての評価が組織としての実際の評価や収入に結び付くかどうかは、所属する組織の評価制度やプログラミング以外の仕事ぶりも含めて決まるのが現実です。ですが、「良いコードを書けること」がマイナス評価につながることはないでしょう。

### 仕事に満足感や自信が持てるようになる

もう二度と触りたくない保守が不可能なコードを書いたことはありませんか? そのような低いクオリティの仕事をしてしまったときは、仕事に対する満足感を得ることは難しいでしょう。

逆に、自分の意志で適切に良いコードを書き、品質の高い安定したソフトウェアを開発したときは、満足感も高く、自信を持って仕事に取り組めたはずです。長いプログラマー人生を考えると、良いコードが書けるレベ

---

注1 ある特定の問題に対応するための言語のことです。DSLについては第11章で詳しく取り上げます。

ルを目指すのは合理的なことです。

## 1.4 代表者の声

本書の対象読者は次の3グループを想定しています。それぞれの代表者にコメントしてもらいましょう。



### ● 良い仕事をしたい普通のプログラマー

別にハッカーを目指すわけでもないし、休日はパソコンに向かうより家族との時間を大切にしたいです。でも、だからといってプログラムが嫌いなわけではありません。良いコードは書けるようになりたいし、高いクオリティの仕事をしたいです。

特段に「達人プログラマー」を目指しているわけではないが、良い仕事と成果を出したいと考えているプログラマーの人は、本書により普段知ることのない新しい概念を知り、興味関心の対象を広げることができるでしょう。



### ● 達人プログラマーを目指す中級プログラマー

新しい技術は大好きです。書籍やWebでの情報収集は常識なので、良いコードは書けていると思うです。ただ、最近はインプットが多すぎて、ちょっと消化不良気味。ときにはアウトプットしていきたいなあ。

向上心が高く達人を目指すプログラマーにとって、本書が良いドキュメントとしての役割を果たすはずです。



### ● 達人プログラマー

良いコードが書けるようになるには時間が必要じゃ。焦らず積み上げていけば必ずや誰でも良いコードが何かわかるようになるはずじゃぞ。すべては、やるか、やらないかじゃ。



すでに達人な人には、「この書籍を新人などに見せれば教育に使えるな」という観点で見ていただくとよいでしょう。

## 1.5

---

### まとめ

本章では良いコードの定義を行いました。「どのようなものが良いコードか」というのは、それぞれのプロジェクトの特性やチーム構成、技術的な要素、1回きりしか使われないシステムなのか長期的に使用されるシステムなのかなどにより、優先事項が変わります。ただ、ここで紹介した良いコードの定義は基本かつ普遍的なものばかりです。ぜひ、コードを書き終えたら「良いコードの定義を満たしているか」と自分に問いかけてみてください。