

目次

訳者まえがき	v
はじめに	ix
1 章 理解しやすいコード	1
1.1 「優れた」コードって何?	2
1.2 読みやすさの基本定理	3
1.3 小さなことは絶対にいいこと?	4
1.4 「理解するまでにかかる時間」は競合する?	4
1.5 でもやるんだよ	5
第1部 表面上の改善	7
2 章 名前に情報を詰め込む	9
2.1 明確な単語を選ぶ	10
もっと「カラフル」な単語を探す	11
2.2 tmp や retval などの汎用的な名前を避ける	12
tmp	13
ループイテレータ	14
汎用的な名前のまとめ	15
2.3 抽象的な名前よりも具体的な名前を使う	16
例: DISALLOW_EVIL_CONSTRUCTORS	17
例: --run_locally	18
2.4 名前に情報を追加する	19
値の単位	20
その他の重要な属性を追加する	21

2.5	名前の長さを決める.....	22	4.9	まとめ.....	54
	スコープが小さければ短い名前でもいい.....	23			
	長い名前を入力するのは問題じゃない.....	23	5章	コメントすべきことを知る.....	55
	頭文字と省略形.....	24	5.1	コメントするべきでは「ない」こと.....	57
	不要な単語を投げ捨てる.....	24		コメントのためのコメントをしない.....	58
2.6	名前フォーマットで情報を伝える.....	25		ひどい名前はコメントをつけずに名前を変える.....	59
	その他のフォーマット規約.....	26	5.2	自分の考えを記録する.....	60
2.7	まとめ.....	26		「監督のコメンタリー」を入れる.....	60
				コードの欠陥にコメントをつける.....	61
3章	誤解されない名前.....	29		定数にコメントをつける.....	62
3.1	例：filter().....	30	5.3	読み手の立場になって考える.....	63
3.2	例：Clip(text, length).....	30		質問されそうなことを想像する.....	63
3.3	限界値を含めるときは min と max を使う.....	31		ハマりそうな罫を告知する.....	64
3.4	範囲を指定するとき first と last を使う.....	32		「全体像」のコメント.....	66
3.5	包含／排他的範囲には begin と end を使う.....	33		要約コメント.....	67
3.6	ブール値の名前.....	33	5.4	ライターズブロックを乗り越える.....	68
3.7	ユーザの期待に合わせる.....	34	5.5	まとめ.....	69
	例：get*().....	34			
	例：list::size().....	35	6章	コメントは正確で簡潔に.....	71
3.8	例：複数の名前を検討する.....	36	6.1	コメントを簡潔にしておく.....	72
3.9	まとめ.....	39	6.2	あいまいな代名詞を避ける.....	72
4章	美しさ.....	41	6.3	歯切れの悪い文章を磨く.....	73
4.1	なぜ美しさが大切なのか？.....	42	6.4	関数の動作を正確に記述する.....	73
4.2	一貫性のある簡潔な改行位置.....	44	6.5	入出力のコーナーケースに実例を使う.....	74
4.3	メソッドを使った整列.....	46	6.6	コードの意図を書く.....	76
4.4	縦の線をまっすぐにする.....	47	6.7	「名前付き引数」コメント.....	77
	整列すべきなのか？.....	48	6.8	情報密度の高い言葉を使う.....	78
4.5	一貫性と意味のある並び.....	49	6.9	まとめ.....	78
4.6	宣言をブロックにまとめる.....	50	第II部	ループとロジックの単純化.....	81
4.7	コードを「段落」に分割する.....	51	7章	制御フローを読みやすくする.....	83
4.8	個人的な好みと一貫性.....	52	7.1	条件式の引数の並び順.....	84

	解決策を言葉で説明する	163			
	手法を再帰的に適用する	163			
12.4	まとめ	165			
13章	短いコードを書く	167			
13.1	その機能の実装について悩まないで ——きっと必要ないから.....	168			
13.2	質問と要求の分割	168			
	例：店舗検索システム	168			
	例：キャッシュを追加する	169			
13.3	コードを小さく保つ.....	170			
13.4	身近なライブラリに親しむ	172			
	例：Python のリストとセット	172			
	ライブラリの再利用はなぜいいことなのか	173			
13.5	例：コーディングするよりも Unix ツールボックスを使う	173			
13.6	まとめ	175			
第 IV 部	選抜テーマ	177			
14章	テストと読みやすさ	179			
14.1	テストを読みやすくして保守しやすいものにする	180			
14.2	このテストのどこがダメなの?	180			
14.3	テストを読みやすくする	181			
	最小のテストを作る.....	183			
	独自の「ミニ言語」を実装する.....	183			
14.4	エラーメッセージを読みやすくする	185			
	もっといい assert() を使う.....	185			
	手作りのエラーメッセージ	187			
14.5	テストの適切な入力値を選択する	188			
	入力値を単純化する.....	188			
	1つの機能に複数のテスト	190			
14.6	テストの機能に名前をつける	190			
			14.7	このテストのどこがダメだったのか?	192
			14.8	テストに優しい開発.....	193
			14.9	やりすぎ	195
			14.10	まとめ	196
			15章	「分/時間カウンタ」を設計・実装する	197
			15.1	問題点	198
			15.2	クラスのインタフェースを定義する	198
				名前を改善する	199
				コメントを改善する.....	200
			15.3	試案1：素朴な解決策.....	202
				このコードは理解しやすいか?	203
				読みやすいバージョン	203
				パフォーマンスの問題.....	205
			15.4	試案2：ベルトコンベヤー設計	205
				二段階ベルトコンベヤーの実装.....	206
				これで終わり?	208
			15.5	試案3：時間バケツの設計	208
				時間バケツの実装	209
				TrailingBucketCounter を実装する.....	211
				ConveyorQueue の実装	213
			15.6	3つの解決策を比較する	214
			15.7	まとめ	215
			付録	あわせて読みたい.....	217
				高品質のコードを書くための書籍	218
				プログラミングに関する書籍.....	219
				歴史的記録.....	220
			解説 (須藤 功平)	223	
				実際にやる	224
				実際にやるとぶつかること	224

他の人に読んでもらう	225
おさらい	225
当たり前にする	226
既存のコードを読みやすくする前にやること	226
続けることが大事	227
コードで伝える	227
読みやすいコードがもっと当たり前であり続けるために	227
コミットメールのススメ	228
まずはあなたが読む	228
添削コミット	229
おさらい	230
最後に	231
索引	233