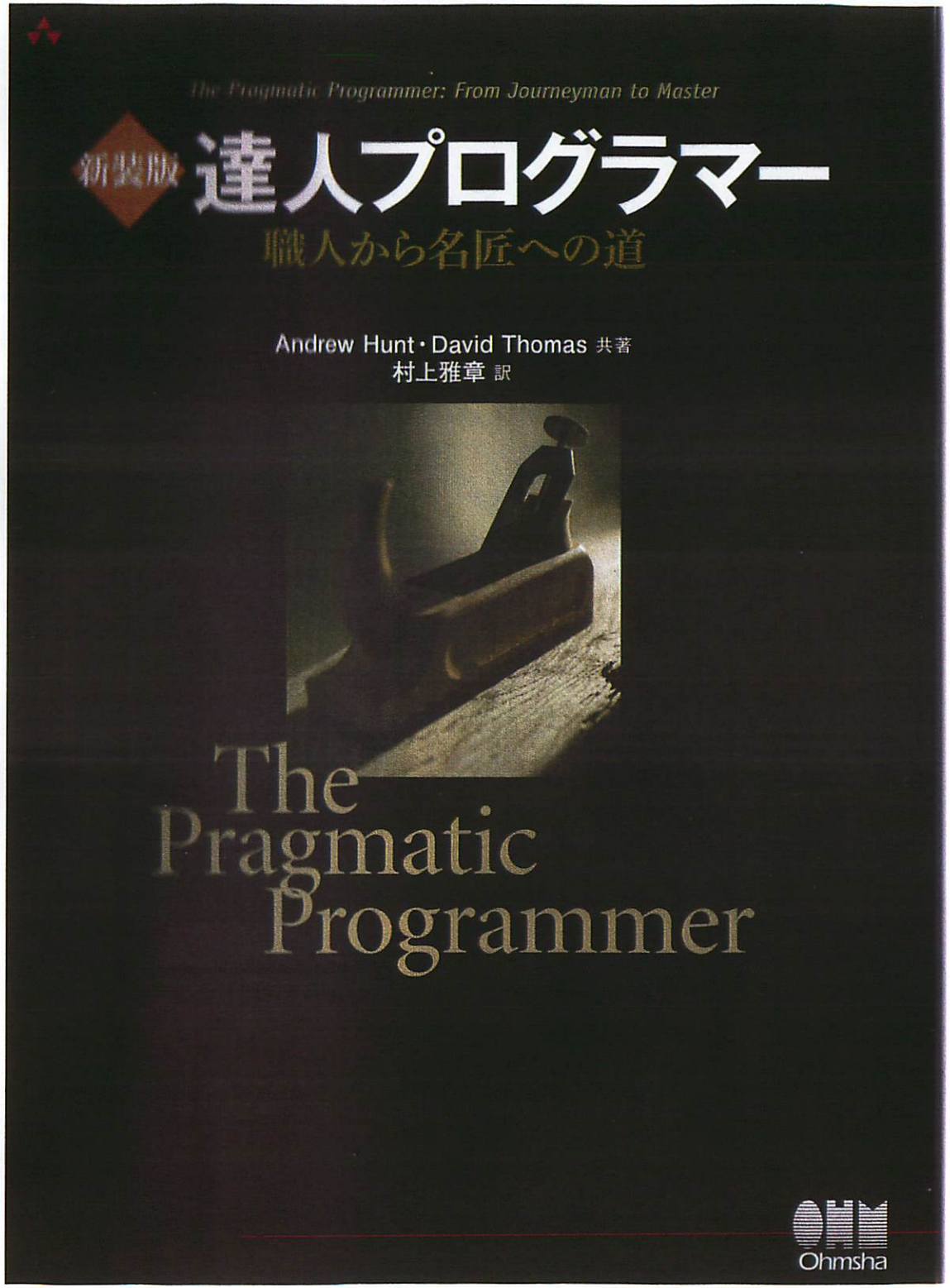


# 序文

*Foreword*

私は校閲者として、本書を早いうちから読む機会に恵まれました。当時、それはまだ下書きの段階であったにもかかわらず、内容には目を見張るものがありました。Dave Thomas と Andy Hunt 両氏は伝えるべきことがらと、それを伝える方法を心得ています。また私は、彼らが行うことと、その効果を間近で見てきました。このため私は、自らこの序文を書きたいと願い出たのです。

簡単に言えば、本書は無理のないプログラム開発方法を教えてくれるものです。本書に書かれていることを実践するのはさほど難しくないはずです。では、なぜ今までこんな本がなかったのでしょうか？ その理由は、今までのプログラミングに関する書籍のほとんどがプログラマーによって書かれたものではなかった、という点にあります。多くの書籍は、言語設計者、および彼らの成果物を宣伝するためにジャーナリストによって書かれたものです。こういった書籍からでもプログラミング言語の書き方は学ぶことができるでしょう——それも確かに重要です、しかしそれはプログラマーが行う日々の作業のうちのほんの一部でしかないのです。

プログラマーはプログラミング言語を使って何をしようとしているのでしょうか？ さぁ、これは難問です。自らが行っていることを説明してくださいと問われた場合、プログラマーの多くは困ってしまうのではないかと思います。これはプログラミングという作業が、しっかりと見据えておかないと見失ってしまうような細かい事象を満載したものだからです。そして、そういった事象の海を何時間も漂流した結果、プログラムのコードが出来上がってくるのです。探し求めることによってすべてのステートメントが姿を現してくるのです。こういったことを注意深く考えないのであれば、プログラミングという作業はプログラミング言語を単に1行ずつ入力するだけになってしまうでしょう。もちろんそれは間違いですが、本当の答えを書店のプログラミング関連のコーナーで見つけ出すことはできないはずです。

本書で Dave と Andy は、我々が実践することのできるプログラム開発方法を示してくれています。なぜ彼らにそのようなことができるのでしょうか？ 彼らと先ほど言及した多くのプログラマーとは、どこが違うのでしょうか？ その答えは、彼らが自らのやっていることを常に見据えている——そして、それをより良くしようと努力している、という点にあります。

あなたが会議に出席していると考えてください。プログラミングしなければな

らないものがあるのに、その会議はまったく終わる気配を見せません。こんな時 Dave と Andy なら、なぜこの会議が必要なのか、また、この会議の一部を何か別の手段で置き換えられないのか、そしてその部分を自動化すれば、次からは会議でなければできないことだけをやれるだろうと考えるのです。そして、彼らはそのとおりのことを実践するのです。

これが Dave と Andy の考え方です。会議は彼らからプログラミングを遠ざけるものではなかったのです。つまり、この考え方こそがプログラミング、つまり改善のためのプログラミングなのです。こういった彼らの考え方は「Tip 2：あなたの仕事について考えること！」から理解できます。

彼らは、こういったものの考え方を数年間続けてきました。これによってさまざまな解決策のコレクションが出来上がったことは、容易に推察できるでしょう。そして、その解決策の実践を数年間続け、難しすぎるものや常に成果が上がるとは限らないものを捨てていったのです。そう、まさにこのアプローチがプラグマティック*1なのです。その後、彼らがこのようにして身につけた解決策を数年間かけて書籍にしたと考えてください。「これは宝の山だ」と思われたのではないでしょうか。そうです、まさにそのとおりなのです。

著者はプログラムの作り方を教えてくれます。しかも、我々が実践できるやり方で教えてくれるのです。この 2 つ目の文章はあなたが思っている以上に重要な意味を持っています。

著者は本書がソフトウェア開発理論の提案になってしまうことのないよう、注意を払っています。これは良いことです。というのは、もしも彼らがそうしていたら、そういった理論を補強するために各章がやむを得ずねじ曲がったものとなってしまうからです。こういったねじ曲げは、理論や仮説が法則化されたり、そのまま忘れ去られたりする、言わば自然科学の進め方です。一方、プログラミングにはほとんど法則というものがありません。このため法則のように記述されたプログラミングのアドバイスは、一見すると聞こえが良いのですが、実用的なものにはならないのです。これが方法論について書かれている多くの書籍がうまく機能していない理由なのです。

私はこの問題を 10 年以上にわたって研究してきました。その結果、最も見込みがあるのはパターンランゲージと呼ばれる手段を用いることだと実感しています。ひとことで言えば、パターンとは解決策であり、パターンランゲージとはお互いを補強し合う解決策のシステム（系）なのです。すべての方法論は、こういったシステムを模索しながら形成されていったものなのです。

この本は Tip を集めただけのものではありません。これは羊の毛皮をまとったパターンランゲージなのです。つまりそれぞれの Tip が経験に裏打ちされており、具体的なアドバイスとして記述され、解決策のシステムを構成するよう、互いに関連づけられているからです。こういった特徴によりパターンランゲージの学習と実践が可能になるのです。彼らはまさにパターンランゲージを使っているのです。

*1　［訳注］実践的ということです。まえがきの訳注を参照してください。



本書のアドバイスは具体的なので、無理なく従うことができるはずです。そこには曖昧な抽象概念など存在しません。あなたのプログラミング経歴を活性化させるための生きた戦略となるようなヒントを、Dave と Andy が直接あなたのために書き起こしてくれたのです。彼らは簡潔に、ストーリーを語り、軽いタッチで、あなたが試みた際に出てくるであろう疑問に答えてくれています。

まだあります。あなたが 10 個から 15 個の Tip を読み終える頃には、あなたの眼前に別次元の世界が広がり始めるはずです。我々はそれを *QWAN*——無名の質（*Quality Without A Name*）と呼んでいます。この本にはあなたの意識に染み入り、あなた自身の糧となる哲理が含まれているのです。これは宗教の勧誘ではありません。単にどうなるかということを伝えているだけです。しかし伝えておくことによって、より効率的にそれを実感できるはずです。哲理があり、それをもったいぶらずに示唆している点がこの本の良いところなのです。

まとめると、本書はプログラミングの実践全体についての読みやすく、そして使える本だということです。この序文では、本書が有効となる理由についてさまざまな側面から語ってきました。しかし、そんな理由よりも本書が使いものになるのかどうかのほうが気にかかるという人もいるでしょう。大丈夫です。本当に使いものになる本だということは、すぐに分かるはずです。

Ward Cunningham

# まえがき

## *Preface*

本書はあなたがより良いプログラマーになるためのお手伝いをするものです。

あなたが一匹狼のプログラマーであるか、大規模プロジェクトチームの一員であるか、多くの顧客とともに働くコンサルタントであるかは関係ありません。本書はあなたという個人がより良い仕事を行えるよう、お手伝いするためのものです。本書では、理論を説くようなことはしていません——あなた自身の経験を活かして見識ある判断をできるよう、現実的なトピックスを具体的なかたちで取り扱っています。本書タイトルの達人[*2]という言葉はラテン語の *pragmaticus*——実務上の熟達した——これ自身はギリシャ語の「行うこと」を表す $\pi\rho\alpha\tau\tau\epsilon\iota\nu$ から来ています。

プログラミングとは技芸です。その最も純粋な部分では、あなた（あるいはあなたのユーザー）がやりたいことをコンピュータに伝えるための作業と言えます。そして、プログラマーであるあなたは、聞き手、アドバイザー、翻訳者、独裁者と自らの役割を演じ分けるのです。とらえどころのない要求をつかみ、その表現手段を模索することで、単なる機械が素晴らしい力を発揮できるようになります。あなたが仕事を文書化することで、周囲の人たちはその仕事を理解できるようになります。また、あなたが仕事を巧みに組み立てることで、他の人たちはさらなる上を目指せるようになるのです。さらに、プロジェクトの容赦ないスケジュールと戦いながら、これらすべてをあなた自身で行うのです。つまり、あなたは毎日小さな奇跡を起こしながら仕事を進めていくというわけです。

これは大変な仕事です。

多くの人々があなたに助けを申し出てくるでしょう。ツールベンダーは、自分たちの製品なら奇跡を起こせると吹聴します。方法論のグル達も、自分たちの技法なら結果を保証できると約束します。皆、口々に自分たちのプログラミング言語が最良であると主張し、自分たちのオペレーティングシステムによって考えられるすべての問題が解決されると喧伝します。

もちろん、どれも本当ではありません。簡単な答えなんてないのです。つまり、こういったツール、言語、オペレーティングシステムだけでは最適な解決策に到達できないのです。あるのは特定の状況に応じた、より適切なシステムだけ

*2 ［訳注］原題は *The Pragmatic Programmer* であり、“pragmatic” は「実践的」といった意味があります。

です。

このため、プラグマティズムというものが必要となってくるのです。何らかの特別な技術に対する執着ではなく、十分に幅広いバックグラウンドと経験が基盤にあってこそ、特定の状況に応じた優れた解決策を選択できるのです。つまり、コンピュータ科学の基本的原理を理解することによるバックグラウンドと、幅広い分野における実践的なプロジェクト経験が大事であるというわけです。そして、理論と実践を組み合わせれば、強力な武器を手にできるのです。

また、現在の状況や環境に合わせたアプローチの調整も必要です。プロジェクトに影響を与えるものすべての重要度を判定し、経験に基づき適切な解決策を生み出すのです。そして、こういったことを作業の進捗とともに継続的に行っていきます。達人プログラマーはこのように仕事をし、そしてうまくやり遂げるのです。

## 誰が本書を読むべきなのか？

本書は、より効率的、そして、より生産的なプログラマーになりたいと願う方々のためのものです。このような願いの背景には、自分自身の潜在的な力を発揮できていないという不満があるのかもしれません。また、仲間がツールを駆使して高い生産性を達成している様子を見てのことかもしれません。また、現在の仕事が古い技術を使用しており、あなたの仕事に新技術がどのように適用できるのかを知りたいということかもしれません。

我々はすべて（あるいはほとんど）の答えを知っているとか、すべての状況に我々のアイデアが適用可能である、と主張するつもりは毛頭ありません。ただこのアプローチに従えば、経験を積む速度が飛躍的に向上し、生産性が高まり、開発プロセス全体のより良い理解も得られるようになるはずです。そして、結果的により良いソフトウェアを構築することができるようになるのです。

## 達人プログラマーになるためには？

各開発者には、それぞれ異なった長所と短所、好き嫌いがあります。個人の環境というものは、時とともに築き上げられていきます。こういった環境は、プログラマーの趣味、衣服、髪型といった個性を強烈に反映したものとなります。もしあなたが達人プログラマーなのであれば、以下の性格の多くを併せ持っているはずです。

- 新しい物好き　あなたは技術や技法に対する才覚を備えており、それを試すことに生き甲斐を感じています。新しいものを見つけると、あなたはすぐにそれを理解し、既に得ている知識に即座に取り込むことができます。



そして、その経験から自信を育んでいきます。

- 研究好き　あなたは疑問を感じやすい傾向にあります。「こりゃいいや——どういうふうになってるんですか？」「そのライブラリで何か問題が発生したことはある？」「ちらっと話に出てきたこの BeOS って一体何だろう？」「シンボリックリンクってどうやって実現されているんだろう？」。あなたは、ちょっとしたことの収集魔でもあり、そうやって収集したことを何年か先の意志決定の糧にするのです。
- 批判的　あなたは、事実がはっきりするまで物事を額面どおりに受け取りません。仲間が「今までもそうやってきているから」と言った場合や、ベンダーがすべての問題を解決すると約束してきた場合、あなたは問題の臭いを嗅ぎ取ります。
- 現実的　あなたは直面している各問題に潜んでいる本質を理解しようと努めます。こういった現実主義的なものの考え方により、ものごとがいかに難しいか、そしてどれだけの時間がかかるのか、といったことに対する鋭敏な感覚を養っているのです。また、あるプロセスが難しいはずだとか、完遂するには時間がかかるはずだということも理解できるため、それに対処するスタミナを保ち続けることもできます。
- 何でも屋　あなたは幅広い分野の技術と環境に慣れ親しもうと常に努力しながら、それと並行して新たな開発を行い続けます。そして、現在の仕事でスペシャリストになることを要求された場合、あなたはすぐにその新しい分野に挑戦することができます。

これ以外にも、最も基本的な性格があります。それはすべての達人プログラマーが持っている性格であり、Tip として挙げておく価値のあるものです。

**Tip 1**

自らの技術に関心を持つこと

あなたが自ら使っている技術に関心を持たない限り、ソフトウェア開発には何の意味もないと我々は感じています。

**Tip 2**

あなたの仕事について考えること！

達人プログラマーになるためには、自分自身がいま何をやっているのか、常に

考え続ける必要があります。これは、どこかのタイミングで１度だけ作業を振り返るという話ではありません–日々の意志決定、あるいは各開発におけるすべての意志決定に対して継続的かつ批判的な評価が必要となるのです。絶対に漫然と行ってはいけません。絶え間なく考え続け、リアルタイムで自らの作業を批判的に見るのです。IBM が古くから企業モットーとして掲げている「THINK!」（考えろ！）は達人プログラマーの唱えるべきマントラ（真言）と言えます。

これを難しいと感じたあなたは、地に足のついたものの考え方をする方です。確かにこういったことは貴重な時間を必要とします——時間は既に大きなプレッシャーとなって、あなたにのしかかっているのかもしれません。しかしその見返りとして、愛する仕事に対するより深い充足感、広い分野にわたる支配感、継続的な向上を感じることへの喜びが待ち受けているのです。時間への投資は長い目で見た場合、あなた自身とあなたのチームの効率化を促進し、保守しやすいコードの生産を可能にし、会議時間を削減するといった効果となって返ってくるのです。

## 達人と大規模チーム

大規模チーム、あるいは複雑なプロジェクトでは、個人の入り込む余地などないと感じている人がいるかもしれません。「ソフトウェアの構築は技術的戒律によって守られるべきで、チームメンバー個人が意志決定を行うと破綻をきたしてしまう」という考え方です。

これには賛成しません。

確かにソフトウェアの構築は技術的戒律に従うべきものです。しかしそういった戒律は個人の技芸に対する心意気を排除してしまうものではありません。中世のヨーロッパで建築された巨大な聖堂を考えてみましょう。こういったものを建築するには何十年にもわたる数千人年の労力が必要であったはずです。そこで得られた貴重な体験は次の世代の建築家に受け継がれていき、完成した建築技術へと昇華していったのです。そして大工、石切工、彫刻家、ガラス細工家といった職人はすべて、技術的要求を自ら解釈することにより、建築物の単なる構造的側面を超越した全体美を生み出していったのです。個人の貢献がプロジェクトを支えている、という彼らの信念がこういったことを可能にしたのです。

> 単に石を切り出す場合でも、常に心に聖堂を思い描かねばならない。
>
> ——採石場作業者の心得

プロジェクトの全体構造の中には常に個性と技芸に対する心意気の入り込む余地があります。これはソフトウェア技術において、特に正しいと言える点です。今から 100 年後、我々の技術は現代建築家の目から見た中世聖堂建築家の技法のように廃れているかもしれません。しかし、我々の技芸に対する心意気は賞賛され続けるでしょう。



## 継続は力なり

> イギリスのイートンカレッジを訪れた観光客が、どのようにしたらこのように完璧な芝生を育てられるのか庭師に尋ねました。
> 「簡単でさぁ、」と庭師は言いました。
> 「毎朝芝生の露をふき取ってやって、１日おきに芝を刈って、週にいっぺんローラーをかけてやるだけでさぁ」
> 「それだけなんですか？」と観光客は尋ねました。
> 「あぁ、」と庭師は返します。
> 「それを 500 年ほど続ければ、あんたん所も同じような芝生になりまさぁ」

素晴らしい芝生を育てるには、毎日少しずつの世話が必要なのです。偉大なプログラマーについても同じことが言えます。経営コンサルタントは会話の中によく「カイゼン」という言葉を挟みます。「カイゼン」とは「数多くの小さな進歩を継続して行う」という概念を表す日本の言葉です。この言葉は、日本の工業製品の生産性と品質を劇的に向上させた主な理由として考えられ、今や世界中に広まっています。持っているスキルを日々磨き、新たなツールのレパートリーを増やしていくのです。イートンカレッジの芝生とは異なり、成果は数日で目に見え始めるはずです。何年かすれば、経験の蓄積、技の成長にあなた自身が驚かれることでしょう。

## 本書の構成について

本書は短いセクションを集めたかたちで構成されています。各セクションはそれぞれで完結しており、特定の話題に特化しています。また多くのクロスリファレンスによって、それぞれの話題の理解を深められるようにもなっています。このため、どのような順でセクションを読んでいっても構いません——本書を前から順に読み進める必要はまったくないのです。

時折、「Tip *nn*」という見出しの付いた箱書きを目にするはずです（例えば「Tip 1：自らの技術に関心を持つこと」(p. xiii)）。これは本書において強調すべき重要な点であるとともに、生きた言葉でもあります——この言葉は日々、我々とともにあるのです。なお、すべての Tip を一覧にしたリファレンスが、付録 C として本書巻末に載っています*3。

付録 A はさまざまなリソース集となっており、本書の参考文献、インターネット上にあるリソースの URL 一覧、お勧めの定期刊行物、書籍や専門機関の一覧

*3　［訳注］原書では巻末折り込みのカードとなっていたものを、本書では付録として掲載しています。

を収録しています*4。本書中にある [KP99] や [URL 18] といったリファレンスは、それぞれ参考文献と URL の一覧を参照するものです。

また適宜、演習問題とチャレンジを設けています。演習問題は比較的直接的な解答が出せるものであるのに対し、チャレンジは幅広い解釈が行えるものとなっています。我々の考えを少しでも有効に伝えることができるよう、付録 B には演習問題の解答を収録しています。しかし正しい解答が一つしかないものはほとんどありません。チャレンジはグループ討議の議題として、またはより進んだプログラミングコースの自由回答問題とすることもできるものです。

## 本書で使用する専門用語について

> 「おれがある言葉を使うと」とハンプティ・ダンプティはいくらかせせら笑うような調子でいいました、
> 「おれが持たせたいと思う意味をぴったり表すのだ——それ以上でも、それ以下でもない」
>
> ——ルイス・キャロル*5『鏡の国のアリス』(岡村忠軒訳)

本書中のあちこちには、さまざまな専門用語がちりばめられています。そのうちのいくつかは技術色を排除した一般的な言葉であり、またあるものはコンピュータ科学者によって息を吹き込まれた、言語に対する挑戦とも取れるような身の毛もよだつ造語です。こういった専門用語は、最初に定義を行うか、少なくともその意味についてのヒントを与えた上で使用しています。しかし、どこかで漏らしてしまったものがあるかもしれません。また、「オブジェクト」や「リレーショナルデータベース」のように定義するまでもなく十分一般化しているという理由で説明を割愛しているものもあります。もし今までに見たことがない用語に遭遇した場合、それを読み飛ばしてしまわないようにしてください。時間をかけて調べれば、Web 上やコンピュータ科学の教科書中に説明を見つけることができるはずです。また、我々に e-mail を送っていただければ、次の版でその用語の定義を追加することもできると思います。

我々はこういったことすべてを考慮した上で、コンピュータ科学者に対して挑戦することにしました。概念を表す、完璧に定義された専門用語であっても、意図的に使用していないものがあるのです。なぜでしょうか？ そういった用語はたいていの場合、特定の問題領域や特定の開発フェーズに意味が限定されてしまっているからです。とは言うものの、プログラムコードや設計、ドキュメント、チーム編成に対するモジュール化など、われわれが推奨する技法のほとんどを普遍的なものにするというのが本書における基本的哲学のひとつとなっています。このため幅広い文脈中で以前からある専門用語を使用すると、元の用語に付随した余計な概念によって混乱を招く場合が出てきます。そういった場合には、やむなく我々自身で新たな用語を作成しています。

*4　[訳注] 初版出版当時にメジャーであった言語名やツール名などが登場する部分については、本書の全体にわたって、可能な限り、現在取って代わられている技術の名前を併記し、本質を理解するうえでの妨げにならないようにしています。また極力、旧来の技術名も残すようにしています。

*5　[訳注] Lewis Carrol (1832–1898)。オックスフォード大学の数学教授 Charles Lutwidge Dodgson の童話作家の時のペンネーム。



## ソースコードとその他のリソース

本書中に記載されているソースコードのほとんどは、以下の Web サイトから、コンパイル可能なソースファイルの形式で入手することができます。

- https://pragprog.com/titles/tpp

上記の URL には、本書の更新情報や他の達人プログラマーが開発したものに関するニュースとともに、有益なリソースへのリンクも収録しています。

## フィードバックをお待ちしています

読者の方からのコメント、ご意見、本書中の間違い、演習問題中に潜む不具合、すべて歓迎します。E-mail アドレスは以下のとおりです。

- ppbook@pragmaticprogrammer.com

## 謝辞

本書の執筆を開始した時点では、完成までどれだけの労力が必要か見当もつきませんでした。

Addison-Wesley 社は、アイデアの段階から完全版下に至るまでのすべての製本プロセスを、未熟な 2 人のハッカーに懇切丁寧に教えてくれました。初期段階でサポートしていただいた John Wait 氏と Meera Ravindiran 氏、熱心な編集者の（そして簡潔明瞭な表紙をデザインしていただいた！）Mike Hendrickson 氏、製作を助けていただいた Lorraine Ferrier 氏と John Fuller 氏、そして皆をまとめてくれた根気強い Julie DeBaggis 氏に深く感謝します。

次に本書の校閲をしていただいた Greg Andress 氏、Mark Cheers 氏、Chris Cleeland 氏、Alistair Cockburn 氏、Ward Cunningham 氏、Martin Fowler 氏、Thanh T. Giang 氏、Robert L. Glass 氏、Scott Henninger 氏、Michael Hunter 氏、Brian Kirby 氏、John Lakos 氏、Pete McBreen 氏、Carey P. Morris 氏、Jared Richardson 氏、Kevin Ruland 氏、Eric Starr 氏、Eric Vought 氏、Chris Van Wyk 氏にも感謝します。彼らの注意深いコメントと貴重な見識がなければ

本書は読みにくく、不正確で倍ほど長ったらしいものになっていたことでしょう。彼らが提供してくれた時間と知恵にも感謝します。

本書には、我々が何年もかけて多くの革新的なユーザーと作業を続け、積み上げ、洗練させてきた経験が盛り込まれています。最近、我々はいくつかの大規模プロジェクトで Peter Gehrke 氏とともに働く機会に恵まれました。我々の技法に対する彼のサポートと熱意にも感謝します。

本書は LaTeX、pic、Perl、dvips、ghostview、ispell、GNU make、CVS、Emacs、XEmacs、EGCS、GCC、Java、iContract、SmallEiffel、および Linux 上の Bash と zsh シェルを使用して作成しました。これらの膨大なソフトウェアは、すべて無償で利用できるという素晴らしいものです。我々はこういったツールやその他の作業で使用したものすべてに貢献した、世界各地にいる達人プログラマーにも深い感謝の気持ちを抱いています。特に iContract でお世話になった Reto Kramer 氏には深く感謝します。

最後に、家族に対しても決して少なくない借りがあります。深夜のタイピング、多額の電話代、常に上の空でいたことに目をつぶるとともに、何度も著作物に目を通してくれたことにも感謝します。夢を見させてくれてありがとう。

Andy Hunt
Dave Thomas

# 目次