アのコアではなく、他のソフトウェアと組み合わせて実現できるものに関しては、明確にNOと言いましょう。これが結果として、よいデザインやシンプルさを達成する原動力になります。

ただ、顧客からの強い要望で、NOということはできない場合もあります。その際には、ソフトウェア本体で実現するのではなく、本体をラップする形の拡張で実現したり、プラグインのようにコアのコードを変更することなく動作を変更できるようなデザインにして、本体のシンプルさを守りましょう。

「プリンシプルオブプログラミング-3年目までに身につけたい一生役立つ101の原理原則」
上田勲,(秀和システム,2016)

**出典書籍**
『人月の神話』Jr Frederick P. Brooks, 丸善出版(2014)

**関連書籍**
『プログラマが知るべき97のこと』和田卓人他, オライリー・ジャパン(2010)



principle 7.8

# 車輪の再発明

英　語　Reinvented wheel
　　Reinventing the wheel

## どういうこと？ 既にあるのに作る

ある機能について、既にそれを実現しているコードやライブラリがあるのに、自分で改めて同じ機能をプログラミングしてしまうことがあります。これは、既に世の中にある「車輪」のようなものを、わざわざ手間をかけてもう一度発明してしまうような、無駄な作業です。

使えるものがあるのに、わざわざもう一つ発明してしまうのは、工数の浪費となってしまいます。規模が大きかった場合、その被害は甚大です。例えば、「様々なサービスが動くようなサーバーが欲しい」と思いたち、Webサーバーのように大きなソフトウェアを再発明してしまったら、計り知れない工数が無駄になります。

しかも、再発明したものより、既にあるものの方が、たいていの場合、品質は優れています。例えば、今自分で作ったライブラリより、従来からある標準ライブラリの方が優秀です。なぜなら、標準ライブラリを提供した専門家の知識と、それを使用した人々の経験が反映されているからです。さらに、標準ライブラリであれば、自分では何もしなくても、時間とともに障害や機能やパフォーマンスが勝手に改善されていくメリットもあります。

また、標準の規格を無視して、独自のプロトコルでコードを書いてしまった場合、その後もずっと独自路線を歩まなくてはいけません。ローカルなプログラマだけで、世の中の流れに、追従できるわけがありません。さらに、いわゆる「コンセントの形」も独自になるので、将来その部分を差し替えようとしても、それができない状態になってしまいます。

## どうして？ 「車輪を知らない」「車輪を作りたい」

車輪を再発明してしまうパターンは、大きく2つあります。

**・車輪を知らない**

プログラマが「車輪を知らなかった」ことによるものです。つまり、意図しない再発明です。

これは、プログラマの知識不足・勉強不足によるものです。言語の標準ライブラリと同じ機能のコードを作成してしまったり、標準のプロトコルがあるにも関わらず、独自のフォーマットで通信機能のコードを作成してしまう場合などが当てはまります。

**・車輪を作りたい**

プログラマの「車輪を作りたい」という欲求によるものです。つまり、確信犯の再発明です。

これは、「ここで発明されたものじゃない（NIH: Not Invented Here）」症候群と呼ばれる問題です。改めて作成する必要はないのに、技術的興味や、他の人が作った部品を使いたくないというプライドから、プログラマが自分自身で作ったもの以外を拒み、自分自身で何でも作りたがってしまうという場合に当てはまります。

## どうすれば？ 「車輪」以外に注力する

車輪の再発明を避け、本来やるべき作業に注力しましょう。

そのためには、コードを書く前に、同じ機能が標準ライブラリにないか、オープンソースライブラリにないか、標準のプロトコルがないか、必ずチェックするようにしましょう。

また、チームミーティングなどで相談して、他のプログラマから情報をもらうようにしましょう。もしかしたら他のプログラマが既にその機能を作っていることに気付き、チーム内での重複を防ぐことができるかもしれません。

そして、エゴのないプログラミングをチームで徹底しましょう。

作りたいから作るというのは、プログラマのエゴです。しかし、ソフトウェアの目的は、プログラマの欲求を満たすことではなく、ユーザーの要求を満たすことです。ユーザーのために「品質」「期間」「費用」それぞれについてベストを尽くすために、普段から使えそうなものを調査し、高品質のオープンソースないし商用ツールを把握しておくようにしましょう。





## 発展 車輪を再発明すべき時

「車輪の再発明」をあえて行う場合もあります。

**・ビジネス目的**

ビジネス上、核となる部分であれば、自前で作る必要があります。

既存の何かを再利用するということは、その部分に「依存性」が発生するということです。そして、依存するということは、その部分をコントロールできないということです。

そこに致命的な問題が潜んでいた場合でも、それを自ら修正することはできません。修正を依頼できたとしても、いつリリースされるか、本当に改善されたかはわかりません。品質や納期の点で、ビジネス上回復できないダメージとなるかもしれません。

それに、既存の何かを再利用するということは、その部分の「差別化」をあきらめる、ということです。よって、ビジネス上コアな部分であるなら、原理上その部分を自前で作る必要があります。その部分を自ら作り、工夫して差別化を図り、経験して技術を蓄積していくことで、世の中に貢献できる、ユニークなソフトウェアが開発できるようになるのです。

**・学習目的**

優れたプログラマになるためには、質の高い経験を積むことがどうしても必要です。

ソフトウェア開発のパターンや、設計、プログラミングについて書かれた良書は、たくさんあります。しかし、読むのと、実際にやってみるのでは、かなりの隔たりがあります。

同様に、既に存在するコードを流用するのと、自分でゼロからソフトウェアを設計し、テストし、障害修正をして品質を高めていくのとでは、得られる経験値のレベルがまったく異なります。

とはいえ、ソフトウェアの核になる部分を自らプログラミングする機会を持てるプログラマは、ごくわずかです。大部分のプログラマは、流用する側に回ります。その場合、どう動いているかわからないので、「ブラックボックス」として利用するしかありません。

水面だけを見ていたのでは、その下にどんな危険が潜んでいるかわかりません。底の方で何が起きているのかがわからなければ、真に使いこなすことはできません。自分の手で何かを作ることが必要です。そのための「車輪の再発明」は、プログラマが学び、技術を高める上で非常に効果的な方法です。

もちろん、その結果、失敗することもあります。しかし、一度でうまくいくよりも、貴重な体験になるはずです。

自分の手でゼロからコードを書き、あれこれと試行錯誤をすることを通して学ぶことと、技術書を読んで学ぶことには、異なる利点があります。どちらも重要で、必要不可欠な体験です。

**出典書籍**

『プロジェクト・マネジャーが知るべき97のこと』神庭弘年他, オライリー・ジャパン(2011)

**関連書籍**

『EFFECTIVE JAVA 第2版』Joshua Bloch, 丸善出版(2014)
『プログラマが知るべき97のこと』和田卓人他, オライリー・ジャパン(2010)
『Ship It! ソフトウェアプロジェクト成功のための達人式ガイドブック』Jared Richardson 他, オーム社(2006)
『コード・シンプリシティ――ソフトウェアの簡潔性を保つ事実、法則、真理』Max Kanat-Alexander, オライリー・ジャパン(2012)
『情熱プログラマーソフトウェア開発者の幸せな生き方』Chad Fowler, オーム社(2010)
『アンチパターン―ソフトウェア危篤患者の救出』W.J. ブラウン, ソフトバンククリエイティブ(2002)
『Joel on Software』Joel Spolsky, オーム社(2005)




# ヤクの毛刈り

英　語　Yak Shaving

## どういうこと？ 本当の問題にたどり着かない

ヤクという家畜がいます。牛の一種で、その体をおおう毛が、極端に多いのが特徴です。夏になる前に毛を刈るのですが、本体に到達するまでには、相当たくさんの毛を刈らなければなりません。

問題に取り組んでいる時、まるでヤクの毛刈りのように、ある問題を解こうと思ったら別の問題が出てきて、なかなか本体、つまり大元の問題(の解決)にたどり着かないことがあります。問題を解こうと思ったら、さらに別の問題が出てきて、ということが延々と続くのです。

この状態が長いと、もはや何を解決しようとしていたか、元の問題を忘れてしまうことすらあります。

## どうして？ トラブルは芋づる式

問題は、芋づる式に次々と際限なく現れてきます。

例えば、Webサーバーで動作するタスク自動化ツールを入れて、作業を効率化しようと考えたとします。

・「まず、Webサーバーをダウンロードしよう」
・「あれ、ファイルが大きくてダウンロードできない」
・「よし、ダウンロードツールを入れよう」
・「あれ、ダウンロードツールが動かない」
・「ああ、前提モジュールが必要なのか」
・「よし、前提モジュールをダウンロードしよう」
・「ああ、ユーザー登録が必要なのか」
・「よし、ユーザー登録しよう」
・「あれ、ユーザー登録ページが動かない」
・「ああ、ブラウザのバージョンが古いのか」