第１章

組み込みシステム開発とオブジェクト指向の遠くて近い関係

組み込み技術者に必要なプログラミング技術とは？

組み込みシステムのソフトウェアを開発してきた筆者が，「オブジェクト指向的な手法」で，開発効率をどのように向上させてきたのかを解説する．「オブジェクト指向を使うと遅いし，大きくなるから組み込みシステムには向かない」という説に対する，筆者の現場での対策を紹介する．（編集部）

1.1 『オレ流』のはじめに

オブジェクト指向のはじまりをSimula(1967)だとすると，それから今までざっと40年近くが経過したことになります．

オブジェクト指向が，ただ1人の人の考えたものであるならば，その人の考えたことがオブジェクト指向ということで，正解というものがあり得ます．しかし，オブジェクト指向はシミュレーション分野の考え方から始まり，子供にわかりやすいコンピュータの使い方として，または自分がプログラムを書くための道具として，さまざまな人たちがいろいろと発案し続けた考え方です．結局のところ，オブジェクト指向には正解というものが存在しません．

1990年頃，生っ粋の組み込みソフトウェア開発者である筆者が，とあるセミナでオブジェクト指向というものについて聞いたときのことです．そのときの説明では，オブジェクト指向とは要するにSmalltalkのことであり，「組み込み設計でオブジェクト指向を使うということは考えられていない」とのことでした．筆者は，オブジェクト指向を使おうなどと考えてそのセミナを聴講したわけではなく，「自分の仕事に役立てるための参考に」そのセミナを聴講したのです．だから，「世の中で組み込み設計にオブジェクト指向を使うことが考えられていないとしても，これが便利で役に立つならば，典型的な組み込みソフトウェア開発にでも使ってやろうじゃないか」と考え，組み込みソフトウェア開発に適する自己流のオブジェクト指向を作って，使ってきました．本書は，この「組み込みソフトウェア開発に適するオレ流オブジェクト指向」を解説したものです．

注意してほしいのは，「オレ流オブジェクト指向であって，アナタ流ではない」ということです．しかし，アナ

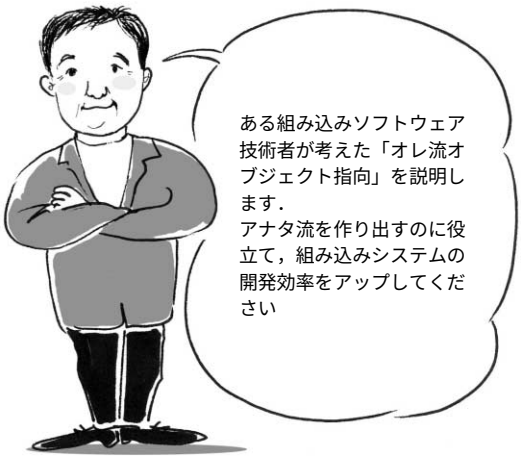

イラスト1.1　アナタ流オブジェクト指向で開発を効率化してください

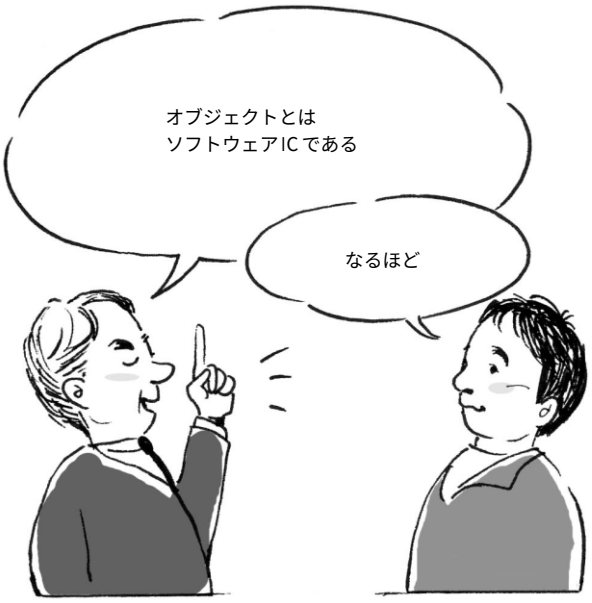

イラスト1.2　オブジェクトとはソフトウェアICである

タ流ではないから本書が無意味だと考えるのは早計です．しょせん，あなたの問題はあなたにしかわからないのです．アナタ流を教えることができるのはあなただけなのです．ただ，アナタ流を作り出したり，改良したりするのに，オレ流が参考になれば，筆者としてはうれしいかぎりです(**イラスト1.1**)．

1.2　組み込みだから，時間制約が，容量制約が厳しいから……

よくいわれるように組み込みソフトウェアは，時間制約も容量制約も厳しいものがあります．1990年当時なら，いや2005年の今でも「オブジェクト指向を使うと遅いし，大きくなるから組み込みシステムには向かない」と言われることがほとんどです．しかし，筆者はもともとオブジェクト指向を「速く，容量を少なくするために」使い始めたのです．OSどころかモニタすら搭載することのできない時間制約と容量制約の中で，どのようにして独立した二つのタスク(仕事)を実現させるのか．この答えの一つとして「ハードウェアの真似をするプログラム」という考えで，プログラムを作っていました．これが，筆者のオブジェクト指向への第1歩です．後に，オブジェクト指向という言葉を聞き，そして「オブジェクトとはソフトウェアICである」という説明を受けたとき，筆者は「あぁ，あのことか」と得心したものです(**イラスト1.2**)．

仮想関数を使うと遅いという話がよくありますが，仮想関数がなければ代わりになるのは`switch`文です．このswitch文を高速化するために仮想関数を使うのです．状態遷移ドライバを高速化するためにステート・パターンを，個々のサブルーチンをできるだけ共通化して容量を削減するためにテンプレート・メソッド・パターンを，時間制約を守るために，容量制約を守るためにオブジェクト指向の技術を使うのです．時間制約と容量制約の厳しい組み込みソフトウェアだからこそ，オブジェクト指向が必要なのです．

1.3　二つの事例を考察する

実用的なレベルのほぼ同じ機能をもつソフトウェア開発で，オブジェクト指向を使ったものとそうでないものを比較する機会はそうそうありません．筆者もそのような経験はあまりありませんが，今まで経験した二つの事例について紹介します．

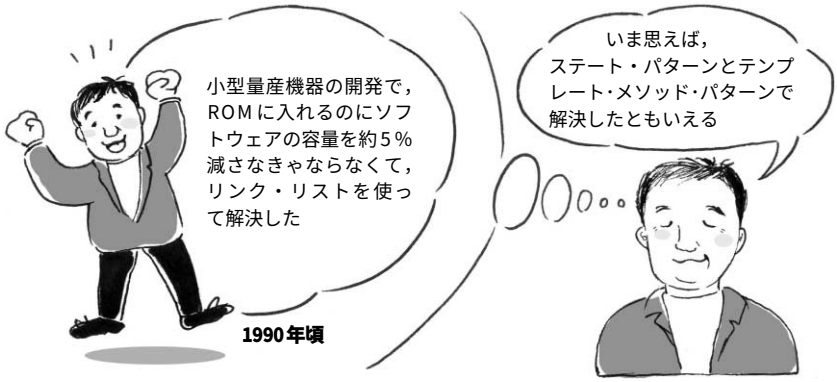

イラスト1.3　オブジェクト指向でコード・サイズを5%削減した

● 開発の最終段階で――容量制約をクリアした

1990年頃の小型量産機器の開発です．あと5%ほどプログラムを小さくしないとROMに入らず，出荷できないというケースがありました．8ビットのCPUをターゲットとしたアセンブリ言語での開発でした．すでにほとんどのテストも終了して出荷が近く，大幅な変更などできません．多くの箇所で状態遷移が使われており，状態どうしがよく似ていることに気がついた筆者は，似ている状態テーブルを共通化し，違う部分だけを記述できないかと考えました．そして似ている状態テーブルどうしをリンク・リストで結び，「とくに記述されていない場合は，リンク・リストをたどる」という解決策をとりました．これは結局，ステート・パターンを用いて状態オブジェクトを作り，その上で状態オブジェクトどうしを継承関係から整理したことになります．さらに，このメカニズムを使って，「ほとんど同じだが，ほんのちょっと違う」という処理を，「そっくりな部分をベースにおいて，違う部分だけ自分に向かってイベントを送出する」として共通化しました．これはつまりテンプレート・メソッド・パターンになります．

この二つの対策によって，ようやく容量制限をクリアし，出荷にこぎつけました．ということはつまり，少なくとも応急手当的にであっても，オブジェクト指向[注1]技術を使ったほうが使わないよりも5%は小さかったのです(**イラスト1.3**)．

● 開発の初期段階で――ソース行数で30%の削減

もう一つの事例は，パソコン上で動くアプリケーションでした．開発言語はC++でしたが，C++を使ったからといってオブジェクト指向を使っているとは限りません．こちらは，「すでにいったんアプリケーションは完成したが，今後の拡張を考えて今のうちに綺麗にしておきたい」という話から始まりました．これは，一種の大規模なリファクタリングです．この開発では，簡単な分析とアーキテクチャ設計にオブジェクト指向の技術を用い，詳細設計以降は前回作ったバージョンをできるだけ活用するという方針をとりました．

そして，上流設計にオブジェクト指向を用いた場合の効果は，ソース行数でざっと30%の削減になりました．上流工程にオブジェクト指向技術を用いることによって，個々のパッケージの役割がはっきりし，「あちこちで似たような処理を記述する」ことが少なくなったため，プログラムが小さくなったのです．

品質とは不具合の数ばかりで比べるものではない(ISO9126参照のこと)のですが，簡単に比較できるものさしとして，これを使って考えてみましょう．とはいえ，オブジェクト指向で作り直したものと，作り直す前の二つのバージョンの品質を直接比較することはできません．実際，後のバージョンであるオブジェクト指向を使ったもののほうが圧倒的に不具合も少なかったのは，前回のバージョンをかなり流用していることが影響しているはずです．そのため推測になりますが，ソース行数が30%小さいということの意味を品質という点から考えてみます．

プログラムの規模が大きくなると複雑度も急激に大きくなることが，経験的に知られています．もし，ソース

注1：当時筆者は，デザインパターンという言葉どころかオブジェクト指向という言葉も聞いたことはなかった．

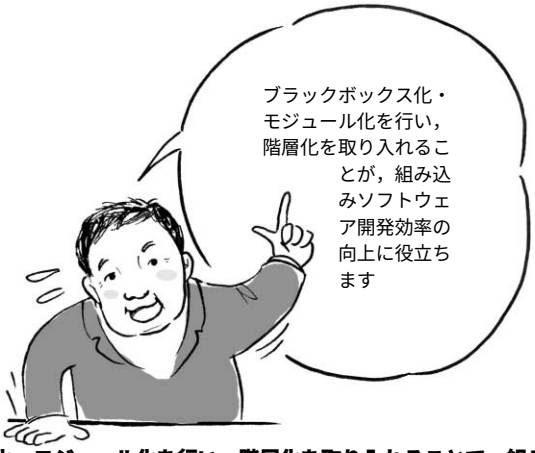

イラスト1.4　ブラックボックス化・モジュール化を行い，階層化を取り入れることで，組み込みソフトウェアの開発効率は上がる

行数が2倍になったとき複雑度が4倍になるとすれば，ソース行数における30%の違いは，複雑度なら倍の違いになります．複雑度が小さければ，当然不具合も少なく，複雑度と不具合の数が比例関係にあると乱暴に仮定すれば，オブジェクト指向を使ったほうが，不具合はざっと半分になると考えられます．

いささか乱暴な考察であり，乱暴な仮定が2回も入っていることから，結論の数値自身は信用できません(笑)．しかし，オブジェクト指向技術を上流工程に用いることの意義はわかってもらえると思います．

1.4 モジュール化と階層化

ここまで，オブジェクト指向の良いところばかり述べてきましたが，では，ソフトウェアの開発はオブジェクト指向だけでよいのかというと，そうではありません．

根本的な話になります．基本的に，ソフトウェアというものはそのままでは複雑すぎ，人間の頭で理解できるようなものではないともいえます．人間の頭をCPUやプログラムにたとえてみると，レジスタの数は七つ程度，OSがタスク切り替えを行う速度はきわめて遅いことでしょう．その人間の頭で，何万行もあるようなソフトウェアというものが(そのまま)理解できるわけはありません．しかし，どうにかして理解できるように考えなければなりません．結局，ソフトウェアほど複雑で大きなものを考えるためには，ブラックボックス化・モジュール化する，階層構造に分ける，といった，オブジェクト指向以前から知られていたことを遵守するしかないのです．

オブジェクト指向とは，ある意味において徹底的なモジュール化を推し進めるためのコツでありテクニックです．隠ぺいがオブジェクト指向のもっとも重要な原則だとすると，オブジェクト指向とは，まさにブラックボックス化・モジュール化そのものです．そのため，モジュール化ということを忘れてオブジェクト指向がうんぬんといっても意味はなく，オブジェクト指向がとくには述べていない階層化も取り入れる必要があるでしょう．

逆に，オブジェクト指向を使わなくても，ブラックボックス化・モジュール化を行い，階層化を取り入れるなら，プログラムを作るのにおそらくオブジェクト指向と同じぐらい役に立つことでしょう(**イラスト1.4**)．