第 6 章　こまめに整理，無駄をなくす，丁寧に書く

バグの入り込みにくいプログラムの書き方

山崎 辰雄

ここでは，トラブルを起こしにくいプログラムの書き方について解説する．一般に「バグ」と呼ばれているものは，開発工程で混入した誤りが発現したものである．本稿では，誤りの正体を見極め，それらの誤りを起こさないようにする構造やプログラミング・テクニックについて解説する．（編集部）

1．バグの正体を見極める

そもそも，バグ（虫）とは何でしょうか？

一般に，「プログラムが期待通りに動かないこと」をバグと呼んでいるようです．それは誰の「期待」でしょうか．作成者（あなた）の？ それとも使う人の？ あなたにプログラム作成を依頼した人の？ 立場によって，バグの内容は異なるものです．

プログラムが期待通りに動かなかったとき，「あ，それはバグ」の一言で終わらせてしまっては，あいまいすぎて対策の立てようもありません．「バグ」というと，「勝手に入り込んでしまったもの」という印象を持たせ，責任者不在の論理に落ち込んでしまうからです（**図1**）．そこでテストの専門家たちは「バグ」という言葉を避け，「誤り」や「欠陥」という言葉を使うように心がけています．なぜここで「テストの専門家」の言葉を借りるのかというと，彼らはこのような機能不全を検出し，指摘する立場にいるため，これらの問題についてとてもよく考え抜いているからです．

プログラムが期待通りに機能しなければ，裁判になることもあります．このときには，期待通りに機能しない「機能不全」を「瑕疵（かし）」と呼んでいます．

● ソフトウェア開発工程で混入する「誤り」

「誤り」（または「欠陥」，「瑕疵」）は，ソフトウェア開発の各工程（**表1**）において発生します．誤りの内訳としては，過剰，欠如，理解不足，単純な誤り，手順の誤りなどが考えられます．

発生する誤りは，例えば次のようなものです．

1）要件を満たしていない

「このような仕様・機能を実現すべきこと」という，発注者，利用者視点の期待を満たせなかった場合を指します．

図1　「バグ」が勝手に入り込んだ？

表1　ソフトウェア開発工程

工　程	内　容
要件定義	プログラムで実現したいことを定める
設　計	定義した要件をどのようにして実現するのかを定める
実　装	「設計」に従ってソース・コードを書き，実際に動作するもの（実行形式）を作る
テスト	「要件」，「設計」通りに「実装」されているかを確認する．あらゆる入力に対して，異常な振る舞いがないかを確認する
公　開	ソフトウェアを依頼主に引き渡す（工程というより，区切り）．パッケージ・ソフトウェアの世界では，一般に「リリース」という
保　守	「公開」後，機能拡張したり，報告された問題を修正したりする

KEYWORD ── バグ，誤り，Joel のチェック・リスト，分割，見える化，スコープ，ハンガリアン表記，コメント，局所化

そもそも依頼内容があいまいだったりするのが原因となることが多いのですが，結果として受け取り拒否，支払い拒否などの大問題に発展することがあります．

2) 設計の誤り

アルゴリズム誤り，データ設計誤り，(ユーザを含む) インターフェース設計誤りなど，「どのように構成するのか/作るのか」という誤りです．形のないソフトウェアではイメージしにくいのですが，具体的なもの，例えば船，飛行機，建物などの設計図に誤りがあったらどのようなことが起こるかを考えると，分かりやすいと思います．

3) 実装 (プログラミング) 誤り

「実装」は「設計書」に基づいて行います．この「実装誤り」は，「設計」通りに作り込めなかった場合を指します．その原因は，単純なタイプ・ミスから始まり，「設計書」の読み違い，プログラミング言語の理解不足から来るものなど，多岐にわたります．

4) 開発・実行環境の誤り

利用しているコンパイラやライブラリの誤り，OS (Operating System) の誤りなど，プログラムを開発，実行する環境そのものの誤りに起因するものです．開発環境に誤りがあれば，たとえプログラムが正しく記述されていても誤った実行形式に変換されるため，記述通りに動作しません．

本稿では主に「実装誤り」と「開発・実行環境の誤り」をいかに回避するかを考えます．また，できるだけ「保守」にも目を向けることにします．

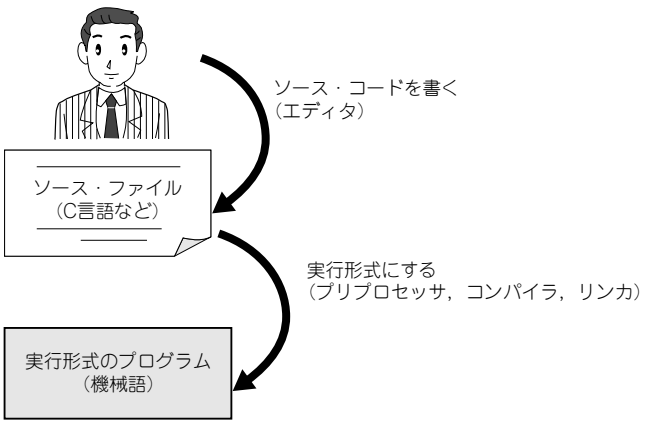

図2　実装工程

「設計」に従ってソース・コードを書き，機械語を生成して実行形式にするのが実装工程である．

● 実装はテストや保守にも配慮して

「設計」に従ってソース・コードを書き，機械語を生成して実行形式にするのが実装工程です (**図2**)．ただし，後工程の「テスト」がやりやすく，しかも「保守」できるように配慮したコードにするべきでしょう．

ソース・コードを書いた人が必ずしも保守まで担当するとは限りません．開発後，何年か経ってから手を加えるという保守もあるからです．また，既にあるソース・コードを基に，幾つかの派生モデルを作ることもあります．このとき，必ずしもオリジナルの作成者が派生モデルの開発を担当するとは限りません．

ソース・コードに手を加えると決まったら，「オリジナルの作成者ではない人」がソース・コードを読んで理解することから始めます．そのため，実装段階からあらかじめソース・コードの「読みやすさ」に配慮して作らなければなりません．

実装工程では，実行形式にしたプログラムを試しに実行してみて，希望通りに動かなければ「バグ」を取る作業 (デバッグ，虫取り) を行います．このときに使うツール (道具) を「デバッガ」といいます[注1]．

コンピュータは，プログラム作成者の「思った (意図した)」通りに動くのではなくて，「指示された」通りに動きます．「指示」はソース・コードとして提示します．一方，コンピュータが実行するのは「機械語」です．「機械語」はC言語で書いたソース・コードを「プリプロセッサ」や「コンパイラ」が変換し，それをリンカがライブラリをリンクして実行できるファイルを作ります．「ソース・コード」は，今のところ人手で書いています[注2]．

● 「実装誤り」について追求する

実装に誤りがあったときは，何が起こるのでしょうか．

● とにかくプログラムが「動かない」

プログラムを書いて，コンパイルしてエラーが出ずに，実行形式のバイナリが得られたとします．でも実行すると

注1：「エディタ」，「プリプロセッサ」，「コンパイラ」，「リンカ」，「デバッガ」はずっと昔，それぞれ独立したプログラムだったが，近年ではそれらを一体にした統合開発環境 (IDE：Integrated Development Environment) として提供されている．ソース・コードの編集から実行形式を作り出すまでに必要なツールがすべてマウスで操作できるようになっていることが多い．

注2：現在，モデル駆動開発 (MDD：Model Driven Development) という手法で，モデルからソース・コードを自動生成する技術の開発が進んでいる．