2019.6 のブログ：「プログラミング心得の普遍性」
（→　http://www.1968start.com/M/blog/index.html#1906b　）の別紙

# 「プログラミング心得の普遍性」

中所 武司

## ■本考察のきっかけ

情報処理学会の学会誌「情報処理」の 2019 年 6 月号の特集
（情報処理，Vol.60，No.6，pp.484-524）
「フレッシュマンに向けたプログラミングのススメ」の感想：
★半世紀近くの間にプログラミングスタイルは大きく変わったと思うが、
本特集の 10 件の解説論文で述べられているプログラミング心得は、
私が現役のプログラマだったころと通じるものがある。
★そこで、1984 年に社内教育用に作成した資料「プログラミング心得」を
思い出し、懐かしさも手伝って掲載することにした。
★特に、付録に付けた、当時の同じ職場の人たちからの一言集は興味深いかも・・・

【私のプログラミング略歴】
・1970 年ごろ（大学院時代）：Fortran（日立の 5020 用）
（ニューラルネットのシミュレーションに使用）
・1971～1974（新入社員時代）：アセンブリ言語（日立の大型計算機用）
（グラフィック作図ルーティン、教育用 Fortran コンパイラの開発に使用）
・その後：Pascal、C、C++、Java などを用いて研究試作

【35 年前の自作の社内教育資料】
「プログラミング心得」（1984.8.10）の紹介
★原本の表紙（pdf）
http://www.1968start.com/M/blog/1984progCover.pdf
★原本（pdf）＜手書き＞図表の確認用として掲載
http://www.1968start.com/M/blog/1984progKnowhow.pdf

★以下は、手書きの原本からの手書き部分を手入力したもの　(^^;;

**********
第６期ＳＳコンピュータ基本技術教育（Ｓ59.8.10　10：30～14：15）

# プログラミング心得

（シ研）２／３０１Ｕ　中所

＜目次＞



## １．良いプログラムとは

### （１）“良い”プログラムの変遷

| | | ［評価基準］ | ［背景］ |
|---|---|---|---|
| ① | 1950-1960 年代 | スピードとメモリ | ハードが高価 |
| ② | 1970-1980 年代 | 理解容易性 | ソフトの大規模化 |
| ③ | 次世代 | 対象指向性？ | 非専門的プログラマの増加 |

②の補足：労働集約的生産方式では、ソフトウェアの大規模化に伴って深刻化した
ソフトウェアの生産性と信頼性向上の課題を解決できないという
“ソフトウェア危機”が発生し、ソフトウェア工学の研究が 70 年代に活発化した。
その結果として、プログラムの Understandability／Readability 向上のための
種々の技法が開発された。

### （２）理解容易性は何故重要か

| ［機能仕様書］ | | ［理解容易なプログラムの特徴］ |
|---|---|---|
| ↓ | ・モジュール設計 | →誤り混入防止、誤り早期検出 |
| ↓ | ・コーディング | →誤り混入防止、誤り早期検出 |
| ［プログラミング］→ | ・テスト | →　誤り早期検出 |
| ↓ | ・デバッグ | →修正／変更容易性 |
| ↓ | ・保守 | →修正／変更容易性 |
| ［検証済みプログラム］ | | |

（参考データ）

【B.W.Boehm 1976 の図】＜縦軸：コスト比、横軸：年代＞

1985 年のコンピュータシステムのコスト比は、以下の通り：

ハード：　10%
ソフト開発：　25%
ソフト保守：　65%

【M.V.Zelkowitz 1978 の図】開発コスト比

要求分析：10%
仕様作成：10%
設計：15%
コーディング：20%
モジュールテスト：25%
統合テスト：20%

【A.R.Sorkowitz 1979 の図】

＜横軸：誤り発見フェーズ　縦軸：誤り修正コスト＞

要求設計：　1
単体テスト：　5
統合テスト：　36

## 2．プログラミングの基本要素

### （1）概要　【階層図】

▲方法論▲
▲　言語　▲
▲　コンパイラ　▲
▲プログラミング環境▲
▲　人間（プログラマ）　▲

### （2）プログラミング方法論・・・モジュール化技法、構造化技法

(1) 階層構造設計　機能単位のモジュール化への分割
(2) 段階的詳細化　同上
(3) 情報隠ぺい　同上
(4) データ抽象化　同上
(5) 複合設計　同上
(6) ワーニエ法　データ中心のモジュール分割
(7) ジャクソン法　同上
(8) 構造化コーディング（構造化プログラミング）
(9) プログラム複雑度

## （３）プログラミング言語　→記述例（次頁）

・アセンブリ言語
・手続き型高級言語　－Fortran, Cobol, Algol, PL/I, Pascal, C, S-PL/H,
・同上（データ抽象化支援）　－CLU, Mesa, SPL, Ada, Iota
・非手続き型言語　－Lisp, APL, Prolog, Smalltalk-80, VULCAN, S-Lonli
・その他　－簡易言語、文字処理言語、数式処理言語、問題向き言語、・・・

記述例（→原本（p.4）参照）

## （４）プログラミング環境

(1) エディタ：行エディタ／画面エディタ（例：DESP）／テキストエディタ／
　　構造エディタ（例：PARSE＊）
(2) コンパイラ：(分類)
　(a) 用途　　→教育用（実行型 Fortran)、デバッグ用（チェック型 PL/I)、実用
　(b) 処理方式　→コア常駐型（WATFOR)、１パス（Pascal（標準))、多パス
　(c) コンパイル単位　→　一括コンパイル（Pascal（標準))、
　　分割コンパイル（Ada, SPL＊)、独立コンパイル
　(d) 稼働マシン　→レジデント、クロス
　(e) 実行方式　　→コンパイラ、インタープリタ (Basic, Prolog)、高級言語マシン (Lisp)

(3) テスト・デバッガ：
　　シンボリックデバッガ、テスト支援システム、テスト網羅率測定（HITS＊）
(4) ＯＳ
(5) ハード

## （５）属人性

ソフトウェア生産性への寄与要因：「個人の能力」への依存度が大
（参考データ）【B.W.Boehm 1981 の図】Personnel/Team Capability 4.18

# ３．プログラミングの“勘どころ”

## （１）基本５則

1．設計に時間をかけよ。
・設計書はきちんと書く。
・設計レビューは入念に行う。
2. よい言語を選べ。
・開発環境と稼働環境を確認する。
・将来性（移植性、再利用性など）を考慮する。

3. テストデバッグ方法はコーディングの前に考えよ。
- 単体テスト、結合テスト、統合テストの手順を決める。
- テストデータ作成方法、テスト結果確認方法も決める。

4. ソースリストをドキュメントにせよ。
- 構造化、インデンテーションを心掛ける。
- 詳細設計書レベルのコメントを極力記入する。

5. 保守は他人に任せよ。
- 他人に任せられるようなわかり易いプログラムを書く。
- 他人に任せられるように設計仕様書、テスト仕様書を整える。

（注）Ｎｏ．６以降は、各自、自分用のものを作成すること。（参考）次頁以降

## （２）規則集

【出典：B.W.kernighan and P.J.Planger: the Elements of Programming Style
（木村泉 訳：プログラム書法、共立出版、1976） 】<→原本（p.7-p.8）参照>

## 付録．プログラミング心得－金言集

（シ研）の先輩たちがどのような点に着目してプログラミングを行っているかについてのアンケート調査結果を参考として載せておきます。
なお、アンケートの質問は以下の通りです。

**********

301U　各位

個性豊かな先輩が新人に贈るキツーイ一言を募集。
「プログラミング心得集」に適したもので、採用分には薄謝進呈。

**********

【利用上の注意】
（１）逆説的表現が多いので、真意を読み取ること。
（２）本テキストをファイルして保存する場合、この付録は削除してもよい。
（３）受講生の中で、自分の体験を通じて得た“心得”があれば、
下記用紙に記入して送ってください。

-----切り取り線----------

301U（チュウ）行　「プログラミング金言集」一般公募用紙

-----切り取り線----------

【良いプログラムとは】
1. バグがあってもすぐ見つかるプログラム（バグのないプログラムはプログラムじゃない）
2. 一画面におさまるぐらい（小さい）プログラム
3. 単純なことしかやってくれないプログラム
4. この世のありとあらゆるプログラム（もちろん自分自身も）を消去し、
などと考える不逞の輩を放逐せしめる霊験あらたかなプログラム
5. することが一言で言えるようなプログラム

6. 構造がシンプルかつ美しいこと
7. 仕様書通りに動く
8. 読めばわかるプログラム
9. 与えられた使命を単純に行い、よけいなことはやらない
10. ほかの人がデバッグできるプログラム
11. 性能が良いこと
12. サイドエフェクトがないこと

【プログラミング心得】
13. 良い言語を選ぶ（「間違いだらけの言語選び」にならぬように）
14. あまり作らないようにする
15. できるだけ自分でやらず、他人にやらす
16. （少々まじめに）タイプの練習をする（タイピングスピードで効率が全然ちがってくる）
17. 作ったプログラムは早晩捨て去られる運命にあることを肝に銘じてそれなりの手抜きをする
18. 決して実行しない（ソースプログラムのままかざりにする）
19. 設計ドキュメントも残しておく
20. コメントをいやというほど入れましょう
21. プログラミングの目的ごとに大幅に異なる。目的に合ったものを選ぶ
22. 分割コンパイルやオーバーレイを用いなくとも動かせる大きさで作る
23. 大きくなったら適当にやめる
24. 設計だけして、コーディングは他人にまかせる
25. 仕様決定の重要さ、難しさを悟らせること
26. ネスティングレベルがあまり深くならないようにする

【プログラミングを 10 倍楽しむ法】
27. パッチでロードモジュールを作る（コンパイル／リンク不要）
28. バグが見つかるたびに、自分が使っているＯＳの悪口を言う
（コンパイラをののしってもよい！！）
29. 笑ってごまかせ自分のバグ、激しくののしれ他人のバグ
30. バグをいっぱい出してデバッグの方法をマスターしよう
31. 自分では絶対作らないで他人のプログラムの講評のみをする
32. 他人の作ったプログラムのバグをよけて使う
33. 端末と早く良いお友達になること
34. 禁煙中の人はデバッグ中だけはたばこを吸ってもよいことにする
35. デバッグ中、バグを人のせいにして、その人の悪口を言う。
ただし、その人がそばにいないこと
36. PL/I の全機能を用いたプログラムを書く（ Ada だと５倍くらい）
37. １ステートメントの実行毎に内部データをすべてダンプする
38. 端末と楽しくお話して

【見栄プログラミング】
39. goto 文のある言語を使っても、絶対、goto 文は使わない

40. ネストがどんなに深くなっても、必ず２カラムずつインデンテーションを徹底する
41. 名前は、（日本語→ローマ字）を使わず、英語にする
　　ただし、スペルをまちがえたりすると、はずかしいことこの上なし。要注意
42.「頭が単純だ！」とバカにされても、とにかくわかりやすいプログラムを！ Simple is best
43. コメントを多用（1statement につき 10 行ぐらいが望ましい）し、開発ステップ数を
　　水増しする（有用なコメントであるか否かはもちろん問わない）
44. やみくもに小さな手続きに分割し、中にさりげなくアセンブラルーティンと
　　リンクするようなプログラムにする
45. 高度な技術を用いたプログラミングをして、デバッグを楽しむこと
46. 他言語のステートメントを使ってコメントを書く
　　例：-- THE FOLLOWING PROCEDURE EVALUATES THE SUM OF THE ARRAY A(1:256)
　　などとは書かずに、さらりと
　　-- +/A
　　と書く。特に効果があるのは Lisp と APL
47. *, #, %などをコメントに用い、美しいプログラムにみせる
48. コメントにプログラムの正しさの証明を示す
49. 入出力はアルゴリズムに入らないという先達の哲学を思い出し、
　　入出力文のないプログラムしか書かない
50. デバッグ中に他人が来たら、急いで save して動くふりをする
51. グローバル変数は絶対に参照しない
52. 自分の使っている言語をけなしながらプログラミングする。
　　バグが出たら言語の使いにくさを強調する
53. begin end を目立つように書いて構造化プログラミング風に見せる

**********
以上