［フレッシュマンに向けたプログラミングのススメ］ 基般

3 命名に時間をかけよう



柿本正憲 | 東京工科大学

業務プログラムで重要な可読性

社会人になり，仕事としてプログラミングに取り組もうとしている人にはSandi Metzの次の言葉を贈りたい[1]．

「私たちは，一石二鳥の仕事を選んだのです．書いたコードが役に立つという確信のもとに，純粋にコードを書く行為を楽しめます．」

とは言っても，企業で仕事として行うプログラミングには一見して制約が多い．代表的なものはコーディング規約である．ソフトウェア開発の企業や部署であれば通常は定められた規約がある．そのほかにも，コメントの書き方，バージョン管理，コードレビュー方法，テスト手順など，企業によって流儀はそれぞれである．

このような流儀を制約と感じるとしたらそれは筋違いである．あなたのプログラムが実現する機能に制約が生じることは一切ない．良い仕事をするための手段であり，少々面倒と感じてもまずは流儀を理解し実践すべきである．

その上で，単に自分が面倒なだけでなく全体の効率化を阻害していると感じるのであれば，遠慮なく流儀の改善を提案すればよい．まともな上司であれば新人のフレッシュな意見は進んで聞くはずである．提案が採用されるかは中身次第である．

業務で作るプログラムはあとでソースコードを読まれることが前提になる．特に近年は修正を繰り返すアジャイル開発が広まり，頻繁に既存コードを読む場面が増えた．同僚が読みやすい，つまり可読性の高いコードを書くことはきわめて重要である．ここでいう同僚にはもちろん数日後あるいは数年後の自分も含む．

可読性の高いコードを書くための実践的な指南書は多く出ている．可読性改善も含む広義の技法であるリファクタリングとして体系的にまとめられており[2]，プログラマ必読といってよい．ほかの指南書も含め，多くの企業では社内に置いてあるだろう．

本稿では，可読性の中でも，特に名前付けについて述べる．重要なことなのだが，意外と体系化はされていない．名前付けは，Boswellらの『リーダブルコード』の冒頭で触れられている[3]．具体的な名前付けのノウハウはこれらの書籍に譲る．ここでは主になぜ名前付けが重要かの理由を中心に論じる．ソースコード行数が少なくとも数千行～数万行程度以上のプログラムを対象として想定するが，短いプログラムでも有用なことは多いと考える．

不良名称発生の心理

プログラムを書く際に必ず，そして頻繁に生じるのが，変数，定数，関数，クラスなどの名前を自分で付ける作業である．新規の名前を初めて書く行は，アルゴリズムの論理を展開して実装する部分とはやや離れていることが多い．たとえば関数定義の行や変数宣言の行がそれに当たる．

ロジックの場所ではないのに名前を考えるのに時間がかかると，プログラマの心理としては停滞感が強い．さっさと次の処理記述に進みたい．ここで固まっている場面を周囲に見られたくない．結果，意味不明の変数名を付けたままにしてしまう．ちなみ

に，関数の名前を考えるためにプログラマが立ち止まる場所は，上述した関数定義の行ではなく，その関数を呼び出す行となる場合が多い．元々複数行の一連の処理記述を新たに独立した関数として切り出すリファクタリングを行うときである．

さて，不適切な名前を放置して先に進んでしまうもう1つの，そして恐らく最大の心理は，「プログラムが動かなくなるわけではない」という，とりあえずは確実な事実に依拠した一種の安心感である．しかし，自分でもちょっと不適当かなと思うとしたら，他人が読んだらまず意味不明ぐらいになってしまうことは念頭に置くべきである．このような意味不明な名称がソースコード中に散在したときの損失やリスクは計り知れない．

命名に時間をかける理由

名前付けが難しいときには，納得いくまで考えたり同僚と議論したりすべきである．1時間や2時間かかることだって珍しくない．私のかつての同僚のスーパープログラマは，半日かけて名前を考えることも本当にあると豪語していた．

付けた名前は今後コードの多くの場所で使ったり呼び出したりする．このことを考えると，新規の名前を付けるときに時間をかけるのは理にかなった行為である．

コードを読んでいて意味不明の名前に遭遇するたびに1秒から数秒のタイムロスが生じる．そうやって思考がかく乱されたオーバヘッドも考えると実質の損失は十数秒かもしれない．これが1日何度も，そして開発の全期間にわたって起こるのである．

単なる引っかかりならそのうち奇妙な名前にも慣れて思考のかく乱もなくなるかもしれない．しかし，そのような慣れは別のより深刻な問題を引き起こす危険がある．たとえば，誤解を生む名前が使われ続けるうちに真の解釈として無条件に受け入れられるようになる．早晩，同僚が変数や関数の使い方をエラーが表面化しない程度に間違ってしまうリスクがある．そのまま同僚がもろもろ書き進めたあとで全部修正ということも起こり得る．

こういう事象が，意味不明な名前を付けてから数日程度あるいは1回だけで収まる保証はない．継続的に起こる恐れは大きい．数カ月後，1年後に別の新任担当者が被害に遭うことだってある．誤解を生む名前であることの判明は，むしろ月日が経つほど困難となる．このようなことが想定できれば，変数宣言の行のところで1時間ぐらい停滞することは何でもなく思えてくるだろう．

今の統合開発環境なら関数名の一括置換程度は容易だから，まずは仮の名前で書き進もう，とは思わない方がいい．名称考案はたとえ1回でもまとまった時間がかかるのだから，一括置換の機会はずっと先延ばしされる確率がきわめて高い．そのあいだに別の不良名称が次々と現れてはびこってしまう．継続的なタイムロスはますます積み重なっていく．

例外として，変数に可読性のない名前が付く場合を2つ挙げておこう．1つは数行のスコープ内だけで使いきる一時的な変数で，一文字でこと足りることも多い．for ループの繰り返しカウンタの“i”が代表例である．もう1つは，特に論文などで書かれた数式を実装する場合で，論文中の変数名を忠実に使い（const や final で宣言），コメントとして論文参照を付記する．

スペル確認と類語辞典

コード中で名前を付けるにあたっては基本的には省略しない英単語の組合せを使いたい．プログラマの中には英語の苦手な人も多いかもしれないが，訳語やスペルのチェックをするのはほんの数秒の手間である．

前述の引っかかりのある名前の代表例がスペルミスである．プログラムを読む人に与えるストレスとタイムロスの長期累積を考えると，手間を惜しむべ

きではない．英語が苦手な人は，l（エル）とrは必ず確認する，カタカナでオの段のスペルがauなのかoを使うかは必ず確認する，などと決めておくだけでスペルミスはかなり減らせる．

また，類似の概念でも微妙にニュアンスが違う場合がある．たとえば，データを取得する意味を持つ動詞にはget, take, retrieve, acquire, obtain, capture, fetch, load, computeなどがある．取得元がどこであるかによって，あるいは取得の仕方の特殊性によって使い分ける必要がある．シソーラス（thesaurus）を使えばこのような類語は簡単に調べることができる．

ただし，単に変化をつけるために類語を使ってはいけない．同一の概念やモノに同じ単語を使うことは徹底すべきである．エッセイやブログのような文章であれば，同じ単語の繰り返しは語彙不足の露呈であり，読解にもどちらかというと悪影響を及ぼす．しかし，プログラムでは技術文書や論文と同様，徹底した一貫性が重要である．

瞬発力プログラミングとの違い

ハッカソンやゲームジャムのようなプログラミングコンテストが盛んに行われている．これらに参加することで集中力が鍛えられることは疑いない．チーム開発の場合はコミュニケーション能力の訓練にもなる．48時間ですべてを完成させる必要があり，デモ実行後は決してバージョンアップすることのないコードを書く．その際にじっくり名前を考えるわけにはいかない．

腕に覚えのある読者の中には学生時代にこのようなコンテストに出ていた人もいるかもしれない．コンテストが100メートル走だとすれば，仕事でのプログラミングは一般にはマラソン競技や駅伝にたとえられるだろう．要求される能力や作法に違いがあることは認識しておくべきである．

生活習慣病のアナロジー

リファクタリング全般に対してよく言われることだが，命名に時間をかけることは生活習慣病予防にたとえることができる．今やらなくてもすぐに病気になるわけではない．しかし，不摂生が継続するとあるとき大病を患ってしまう．もちろん数々の予兆も現れていたはずだ．苦痛を伴う大手術が必要となり，多くの時間とコストを失う．

単発的な不摂生がやむを得ないケースがあるのは，人生でもプログラム開発でも同様である．納期が3日後なのに変数の命名に2時間かけるわけにはいかない．逆に言えば，緊急の状況でない限りは極力命名に時間をかけるべきである．

やみくもに1人で考えるよりも，同僚に相談して議論した方が早く良い結果が得られる．摂生の習慣を継続するのに主治医や指導者がいた方がいいのと少し似ている．ペアプログラミングやコードレビューはそのための実践手段である．

呼び出し側のコードの見え方を想像する

変数名や関数名として良い名称がついていれば，プログラムを読む人が処理の流れを追う際に何の処理をやっているか分かりやすくなる．コメントも最小限となる．コメントは実装時の特殊事情説明（コードの意図や，真のあるいは裏の理由や，改善の余地があるという言い訳）に限ることができる．

コメントなしでも何をやっているか読めるには，特に関数名（メソッド名）が重要である．呼び出す側での書かれ方を想定した関数名にしたい．関数中の処理の方式を説明する名前ではなく，何を出力するのかを説明する名前の方が望ましい．また，動詞の暗黙の主語は呼び出す側の関数を想定するのがよい．

抽象度の調整

プログラム中で使う名詞の名前を考える場合，そのモノや概念の抽象度は重要である．特に，スーパークラスを抽出することになった場合，その候補クラス名は，元のクラス（サブクラス）に対して適切な抽象化がされている必要がある．

対象物を適切に言い表していてなおかつ抽象度も適切という名前を付けるには，ある程度経験が必要である．こうすればよいという速効ノウハウはなかなかないし体系化もされていない．

その中で，モデルベース思考[4]では抽象度の調整法を豊富な実例で示している．命名に際して抽象度を熟慮することは，結果的にプログラム処理の対象をより深く理解することになる．可読性のみならず，アルゴリズムも洗練されることが期待できる．

最後に実践例を 2 つ

本稿が読まれるころには初孫が生まれている予定だ．命名の相談をしてきた息子には「誰が読んでも迷わず同じ音になる名前」という条件を課した．呼ぶ側が迷う名前によってその子が一生の間にこうむる損失の累積は測り知れないからである．もちろん，息子夫婦は時間をかけて名前を考えている．

あと一例は，本稿のタイトルの命名である．この段落を除く本文を全部書き終えたあとでようやく決まったことは言うまでもない．

参考文献

1) Metz, S.，髙山泰基（訳）：オブジェクト指向設計実践ガイド，技術評論社（2016）．
2) Fowler, M.，児玉公信ほか（訳）：新装版 リファクタリング―既存のコードを安全に改善する，オーム社（2014）．
3) Boswell, D. and Foucher, T.，角　征典（訳）：リーダブルコード，オライリー・ジャパン（2012）．
4) 吉田　塁：スーパープログラマーに学ぶ最強シンプル思考術，ディスカヴァー・トゥエンティワン（2016）．

（2019 年 2 月 12 日受付）

■**柿本正憲（正会員）**　kakimotoms@stf.teu.ac.jp

1982 年東京大学工学部卒業．富士通研究所，日本シリコングラフィックス，シリコンスタジオなどを経て 2012 年より東京工科大学教授．コンピュータグラフィクス，イメージメディアに関する研究に従事．博士（情報理工学）．

