

[フレッシュマンに向けたプログラミングのススメ]



4 「いつでも」「どこでも」「だれでも」動かせるコードを目指して



榊 剛史 | ホットリンク (株)

IT エンジニアとしての新生活に向けて

「フレッシュマンに向けたプログラミングのススメ」ということで、本特集を読んでいる皆様は趣味や研究のツールとしてプログラミングに興味を持ち、業務上でプログラミングを使う職種（以下では総称してITエンジニアと呼ぶ）に就職される／された方々であろう。本稿では、そのような方々に向けて、筆者が今までの経験から、業務におけるプログラミングにおいて、重要だと思うことをざっくりばらんに紹介していきたい。強い根拠がある話ではないため、参考程度に読んでいただければ幸いである。

まず自己紹介をしておく、筆者は、現在、ソーシャルメディア分析サービスを企業向けに提供する企業で研究開発部に所属している。業務としては、機械学習をはじめとするAI関連技術を用いて、自社サービスで用いる分析機能の研究・開発を行っている。

はじめに述べておくと、筆者自身は業務で日常的にプログラミングを行っているが、「何かを作り出す」というのは楽しい行為であり、それを日常的な業務にし、お金を稼げるというのは、恵まれた環境であると思っている。メディアやWeb上の記事などで、ITエンジニアの業務のネガティブな側面が取り上げられることもあるが、それらは往々にしてプログラミング以外の要因（人間関係や労働環境）であり、プログラミング自体が苦痛だという声はあまり聞かれない。その意味で、筆者と同様に、プログラミングを楽しめる方であれば、ITエンジニアに向いており、楽しみながら業務を進めることができるだろう。

一方、業務としてプログラミングを行う場合、お金をもらっている以上、趣味や研究上でのプログラミングと異なり、一定の制約の中でプログラミングを進める必要がある。また、ITエンジニアの業界でキャリアアップしていくためには、業務を通じて、自身の能力を高めていく必要がある。

筆者自身は、略歴にもあるとおり、修士・博士課程およびポスドクという立場で学術研究を行った後、現在は企業の研究開発部という立場でサービス開発を行っている。本稿では、そのような筆者の経験に基づき、「研究としてのプログラミング」と「業務としてのプログラミング」の違いを述べる。具体的には、「再現性」という観点から、業務としてのプログラミング上で注意すべき点を紹介する。また、変化の速度が速く多様性が大きいITエンジニアリングの領域において、その変化と多様性に追従するために筆者が心がけている点をいくつか紹介したい。

研究としてのプログラミングと業務としてのプログラミングの違い

本誌を読んでいるような学生の方々であれば、研究における実験といえば、プログラミングを書いてそれを実行することであろうと思う。その際に、どのようなことを意識されていることであろうか？ 下記に、筆者や筆者の周囲の人間が、学生時代、研究上のプログラミングで優先していた項目を列挙してみた。

1. 研究室ミーティング／学会発表／論文投稿／学位論文締切に間に合うように

2. 自分のPCや研究室サーバで動くプログラミングを
3. 一早く完成させること

そのほかにも項目はあると思うが、多くの人が上記3項目のいずれかまたはすべてを優先しているのではないだろうか？

人によって異なる意見があるであろうが、筆者は研究上のコードは上記のような項目を優先してプログラミングするものと考えている。研究においては、まず優先すべきは「論文を書くための材料をそろえること」である。「ほかの人が研究を追試できるようにすること」「研究室でほかの人が研究を引き継げるようにすること」なども重要ではあるが、優先順位の一歩は材料集めと思っている^{☆1}。

では、業務におけるプログラミングで優先すべきことはなんだろうか？ それは、「機能の再現性を高める」ことであると筆者は考えている。

本稿で言う「機能の再現性」とは、「あるコードで提供される機能が『いつでも』『どこでも』『だれでも』決められた仕様通りに動作させることができること」である。多くの企業において重要なのは、事業の継続性である。なので、事業において提供しているシステムは継続的に動くことが重要である。そのような中でシステム上で動作している機能が「決められた仕様通りに動作させる」ことが保証されなければ、その事業の継続性が揺らいでしまう^{☆2}。一方で、システムを取り巻く環境というのは容易に変化してしまう。担当者は入れ替わる、使っていたツールやパッケージの仕様が変わる・提供がとまる、故障が保証期限切れによりサーバがリプレースされる……など環境が変化する要因は無数にある。その中で、いかに「決められた仕様通りに動作する = 機能の再現性を保つか」が重要になってくるのである。

^{☆1} ただし、だからといってコードを書き散らすのではなく、可能な範囲でコードをリポジトリで管理するようしたり、きちんとコメントを書いたり、ドキュメント化する努力を怠ってはならない。

^{☆2} ここでの「仕様」とは受託開発で顧客から発注される仕様の意味ではなく、「開発着手時または完了時に想定した機能の構成/動作/性能」というより広い意味である。

以下では、「いつでも」「どこでも」「だれでも」という観点から再現性を高く保つために心がけるべきことを紹介する。

だれでも動かせる

前節で、「システムを取り巻く環境というのは容易に変化してしまう」と述べたが、その中でも特に変化するのが人材である。システム開発の企業やプロジェクトにおいては、常に人材が流動している。そのため、「だれでも」動かせるコードを書くのは非常に重要である。

ドキュメント

「だれでも動かせるコードを書く」ことを考える上でまず挙がるのは、「ドキュメントを書く」ことであろう。プログラミングを得意とする学生などで「コードを読めば、大体動作が分かる」という人もいるが、開発する機能が大規模になればなるほど、コードを読んで動作を理解するのに多くの時間がかかるようになり、非効率である。また特にさまざまな要素を組み合わせで構築するシステムの場合、要素間の関係性を表す図がないと、動作を正しく理解することは困難である。そのために、システムを開発するときは、そのコードとともにドキュメントをきちんと整備する必要がある。

以下は、筆者が README.md を書く場合に、入れるようにしている項目である

- 機能概要
- 詳細な使用方法
- 必要な動作環境
- 動作環境の構築方法
- インストール方法
- テスト方法

テスト

研究上のプログラミングで軽視されがちだが、業務上のプログラミングで重要なのが「テスト」である。テストとは「コンピュータのプログラムから仕様のない振舞いまたは欠陥（バグ）を見つけ出す作

業のこと^{☆3}」である。前述の「決められた仕様通りに動作する」ことを確認することがテストである。ソフトウェアテストは実に幅広い領域なので、ここでは詳細については割愛するが、筆者が普段から気をつけていることは、「開発したコードには、必ずテストコードを添付する」ということである。テストコードとは、「それを実行することで、あるコードが仕様通りに動いていることを自動でテストする」コードを意味する。自分が開発したコードを色々な人が色々な環境で実行する場合に、そのコードが仕様通りに動作していることを保証するのがテストコードの役割である。

深刻なバグがある場合、一見正常に実行されたように見えるのに、適切な結果が得られてない、ということが起き得る。この場合には、バグの存在に気づかずにリリースされてしまうリスクを孕んでいる。このような状況を避けるために、開発する機能の再現性を保証する目的で、テストコードを書いて、それを自分が開発したコードとともにまとめておくことは欠かせない作業の1つである。

どこでも動かせる

業務上のソフトウェア開発でよく起きるのが「あるコードが開発環境では動いたが、本番環境では動かない」という事象である。これはある種当然ではあるが、そのコードを動作させる環境を適切に構築できないことが理由である。

これは、開発環境・本番環境という話にとどまらず、さまざまな場合によくあることである。研究上でも、「研究室の共用サーバで開発していたら、他の誰かがPythonのバージョンをアップしたために、自分のコードが動かなくなった」というようなことはないだろうか？(筆者は数回体験したことがある)

残念ながら、自分が開発している環境とほぼ同じ環境を再構築することは非常に難しい。一方で、「ど

んな環境でも動くコードを書く」ということもまた困難である。なので、さまざまな工夫を凝らして、「より多くの環境で動作するコードを書く」ことが必要となってくる。

開発環境と本番環境の共通化

これは多くの企業、特に大企業で行われているアプローチである。システム開発においては、本番環境の仕様が明確に定められており、その仕様を変えられないことが多い。そのような場合に、各開発者が自身のPC/サーバ上に独自の開発環境を構築してしまうと、そこで開発したコードが本番環境で正常に動作しない可能性がある。そこでコードを開発する環境と本番の環境をまったく同じにすることで、開発環境で開発した機能が本番環境で仕様通りに動作することを保証するというアプローチが用いられる。利用するパッケージやツールは限られるものの、ほぼ確実に開発環境で開発したコードを本番環境で動作させることができる。

所属企業で、社内で標準の開発環境が定められているのであれば、仮に非効率な環境に見えたとしても、まずはその環境でシステム開発をすることをお勧めする(ただし、本当に非効率な場合もあるので、その環境でシステムを開発した後、その上で非効率な点を指摘すべきである)。

仮想化技術の利用

開発したコードをより汎用的に使えるようにするためには、仮想化技術を用いてそのコードが動作する環境自体をコードに添付してしまうというアプローチがある。仮想化技術とは、物理サーバ上で別のサーバを仮想的に動作させる技術である。最近では、Dockerを始めとするコンテナ型仮想化技術の発展により、より容易に仮想環境を動作させることが可能になった。速度面やリソース面で制約はあるものの、コードを確実に動作させる目的としては優れた解決策の1つであり、近年はコンテナ型仮想環境を前提とした開発手法やサービスも発展している。

筆者も、全部のコードではないが、最近では開発したコー

^{☆3} <https://ja.wikipedia.org/wiki/ソフトウェアテスト>

ドとともに仮想環境を添付する機会が増加している。

いつでも動かせる

IT エンジニアリングの技術は変化速度がかなり速いため、ツールやパッケージの環境もめまぐるしく変化してしまう。そのため、ある機能の開発に利用していたパッケージやツール、言語がサポートされなくなってしまう、ということが少なくない。

たとえば、今一番勢いのある Python でも Python2.x 系のサポートは 2020 年には終了してしまう。また Oracle の Java SE は Version11 から有料化となっている。C++ などコンパイルを前提とした言語で開発されたツールは、OS でサポートしているコンパイラのバージョンが変わってしまうと、コンパイルが通らなくなるなどよく起きる事態である。

このように変化の激しい環境において、「いつでも動かせる」を実現することは容易ではない。なので、業務でプログラミングをする上では、言語の仕様の継続性やパッケージのメンテナンス性などについて、常に新しい情報を仕入れながら、採用する技術を選定していく必要がある。

とはいえ、この点をあまり気にしすぎると、新たな技術や独自の技術を使うことが困難になってしまう。あくまで全体のバランスを見ながら、使用する技術を選定する必要がある。

IT エンジニアリング業界に追従するために

前章でも述べたが、IT エンジニアリング業界は変化の速度が速く、多様性が大きい業界である。変化の速度が速いとは、人材流動性の高さもさることながら、使われる開発技法やツール・パッケージ・フレームワークのトレンド変化が激しいことである。多様性が大きいとは、システム開発と一言で言っても、開発するシステムの機能や対象となる業界・業

種によって、開発スタイルや体制が大きく変わることである。

このような変化や多様性に追従するために、筆者が心がけていることを紹介する。

継続的な学習

IT エンジニアリングの技術は変化がめまぐるしいため、最先端の技術にフォローアップしていくには、継続的に勉強・情報収集を続ける必要がある。

システム開発のトレンドは直近 10 年でもめまぐるしく変化している。たとえば、バックエンド開発では、一昔前はオンプレミスサーバによる開発が主流だったが、今はクラウドサービスを前提とした開発が増加している。また仮想化技術の活用とマイクロサービス化という点が注目を浴びており、必要とされる知識は 10 年前からだいぶ変化している。このように、たった 10 年前の知識が通用しないことが起き得る業界である。

そして、変化が激しい故に技術が体系化される前に次の技術が普及されてしまうため、体系的に学ぶのが難しいという側面がある。そのため、論文の収集・整理方法のように自分なりの情報収集と学習の方法を確立する必要がある。筆者でいえば、下記のように目的によって情報収集の手段を使い分けている。

- 言語や技術を体系的に学びたいとき：書籍
- 表面的な問題を解決したいときやノウハウを知りたいとき：Web 上の日本語の記事
- 本質的な問題を解決したいとき：公式ドキュメント
- 普段の情報収集：Twitter、はてなブックマーク

人に質問する

IT エンジニアリングの領域は、開発技法や体制 1 つとっても、さまざまな方式があり得る。その中で自分 1 人で調べるのは限界があることが多く、またそもそも社内独自の話であれば、いくら Web を検索しても出てこないし、社内のドキュメントをひっくり返しても情報が出てこないことはよくある

ことである。

そのようなときは、自分で調べすぎずに詳しいと思われる人に聞くことが重要である。「何を当たり前のことを言っているんだ」と思われるかもしれないが、ことシステム開発においては、Web上から検索する習慣が身につけていると、意外とこの観点が抜けてしまうことがある。もちろん、何も自分で調べずに質問し倒すのも適切な態度ではないので、「どこまで自分で調べてどこまで人に聞くか」は自分の中で一定の基準を置くべきであるが、「人に聞く」という選択肢は常に念頭に置いておくべきである（筆者は30分調べて分からなければ、人に聞くようにしている）。

IT エンジニアとしての将来に向けて

本稿では、「再現性」と「必要な技術への追随」という観点から、筆者のこれまでの経験で重要だと考える点について紹介してきた。

ややハードルの高い話も含めてしまったので、少しネガティブな思いを抱いてしまったかもしれない。しかし、「IT エンジニアとしての新生活に向けて」で述べたように、プログラミングを楽しめる人であ

れば、楽しみながら業務を進めることができると考えている。

また、前述のようにIT業界は変化の速度が速く、多様性が高い業界である。変化の速度が速いということは、常に新たな技術に触れることができるということである。新しいものが好きな人や自身の成長に興味を持っている人には適した業界である。また、多様性が高いということは、自分に適した職場を探しやすいということである。人材流動性が高く、職場を変える障壁も低いいため、新たな職場に移ることも容易である。

そのような状況の中で、自分のやりたい仕事に向けて邁進する／自分が目指すキャリアに向けて能力を磨く／自身が働きやすい職場を見つける、など。自分に合った働き方を探して行っていただきたい。本稿がその一助になれば幸いである。

(2019年2月22日受付)

■榊 剛史 t.sakaki@hottolink.co.jp

2004年東京大学工学部電子情報工学科卒業。2006年同大学院修士課程修了。電力会社通信部門での勤務を経て2009年同博士課程入学。2014年博士課程終了。博士(工学)。東京大学での特任研究員を経て、2015年より(株)ホットリンク R&D 部部长ならびに東京大学客員研究員。専門は、Webマイニング、自然言語処理、計算社会科学。

