

# 無能なITエンジニアのための 100の教訓

Ver1.0.0

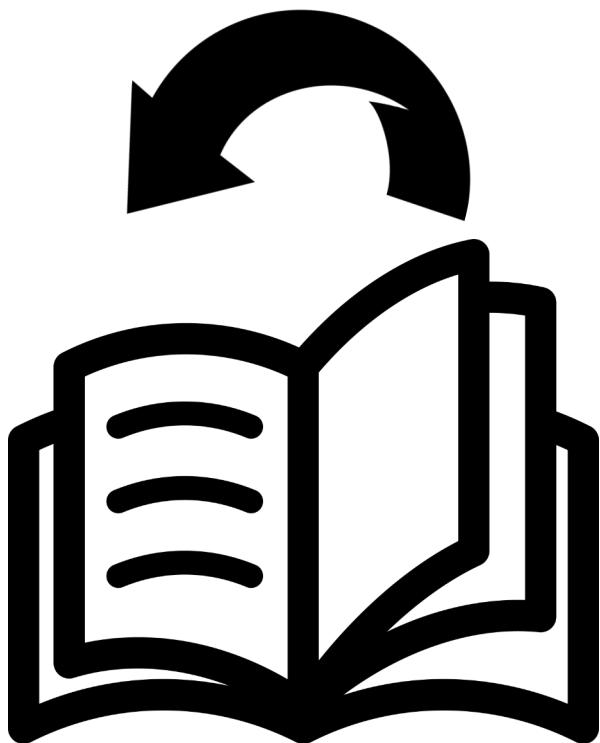
承認	品質	確認	作成
ボンブ	ボンブ	ボンブ	ボンブ

2024年1月1日

無能なボンブ

---

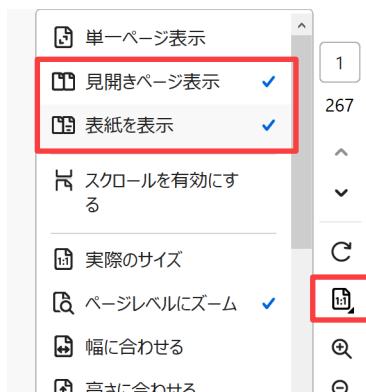
本書は 『左開き』 です。



この書籍はPDF形式で提供しており、PDFリーダーを使用すれば、スマートフォンでも閲覧できますが、より快適に閲覧いただくためにパソコンやタブレットのようなデバイスで、見開き表示をご利用いただくことをおすすめします。以下では、Windows版のAdobe Acrobat Reader (2023.006.20360) を使用してパソコンで快適な閲覧を行う設定方法をご紹介します。

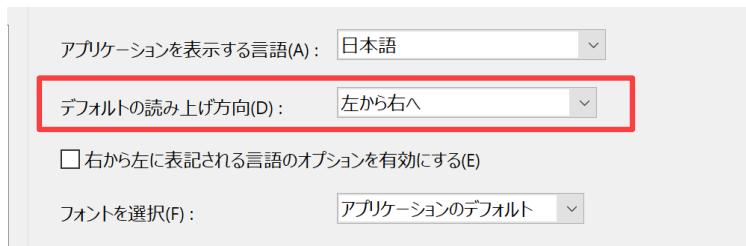
## 見開き設定

Adobe Acrobat Readerの右下のファイルアイコンをクリックし「見開きページ表示」と「表紙を表示」にチェックを入れてください。



## 読み上げ方向

本書を開くと自動で「左開き」で認識されるはずですが、ページが逆になっている場合は、Adobe Acrobat Readerの「メニュー」->「環境設定」->「言語」画面の「デフォルトの読み上げ方向」を「左から右へ」を選択いただくと「左開き」で表示されます。



# はじめに

---

本書は「無能なITエンジニアが、無能なまま生き続けるために必要な100の教訓」をまとめたものです。

筆者がITエンジニアとして働き出した際、無能なITエンジニアとしてスタートしました。仕事は想像以上に難しく、技術の進化や変化についていくどころか、仕事に必要となる当たり前の技術すら使うことが出来ませんでした。筆者は勉強が苦手で、覚えることも遅く、同期たちと比べても非常に緩やかな成長速度でした。また、多くの失敗をしていつも周囲から怒られるか残念な顔をされていました。

それでも、ITが好きだったため仕事を続けたいと思い、書籍やインターネットの情報に頼りました。しかし、ITエンジニア向けの参考書は、技術や方法論を学んだりスキルを高めるためのものばかりでした。自己啓発本は、ひどく抽象的で「頑張る」ことに要約されるようなものも多くありました。これらは筆者にとって、内容を理解することも活用することも非常に難しいものばかりでした。

その後「無能でもITエンジニアを続けるにはどうすれば良いか？」を模索し始めました。周囲から必要と思われるほど技術力を高めることは、勉強が苦手な筆者にとって絶望的だったため、可能な限り技術力に頼らない方法が必要でした。これらの模索から生まれたのが本書の教訓です。

現在の筆者も、相変わらず無能なITエンジニアとして過ごしています。当時から比べると多少の技術力は身につきましたが、得意と言える技術分野はありません。最新の技術も扱えません。新しいことや知らないことを学ぶには多くの時間を必要とします。それでも、教訓によってITエンジニアを続けることが出来ています。

現在の書籍やインターネットを見回しても、無能なITエンジニア向けの情報は未だに見当たりません。ITの技術の内容は変わっていますが、スキルを高めることや資格の大切さを説明するものばかりです。本書は筆者自らが学んだ教訓を1つの情報としてまとめ、同じような悩みを抱える人たちに届けたいという思いから執筆するに至りました。

本書が一人でも多くの無能なITエンジニアの一助となれば幸いです。

## お問い合わせ先

本書の内容に関するご質問やお問い合わせは、以下のX（旧Twitter）アカウントもしくはEメールアドレスまでご連絡ください。

- X（旧Twitter）：@itengr\_matome



- Eメール：itenginner.matome@gmail.com



## 留意点

本書に記載されている会社名・商品名・製品名・サービス名などは、一般的に各組織の登録商標または商標です。本文中にTM・®・©マークは明記しておりません。

本書に記載された内容は、情報の提供だけを目的としています。これらの情報の運用結果について、筆者はいかなる責任も負いません。

本書のI章～X章の本文は、筆者の運営する以下のWebサイトに不定期に公開していくります。

- サイト名：ITエンジニアのまとめ
- 記事名：無能なITエンジニアのための100の教訓
- URL：<https://itenginner-matome.net/archives/42879>



# 序 章

本章では、本書の目的、対象読者、前提条件など  
本書を読む上で必要となることについて説明します。  
また、本書の読者の感想も掲載しています。





謝辞

本書に寄せて

本書を読む前に

本書の構成

# 謝辞

---

本書は、多くの人たちの協力があり完成しました。筆者だけでは完成には至らなかつたでしょう。筆者を支え、応援してくれた皆様に深く感謝します。

本書のすべての読者に感謝します。筆者の「本を書いてみたい」という夢だけでなく「同じような境遇のITエンジニアに教訓を届けたい」という夢も叶えてくれました。皆様にとって一つでも力になるものが見つかれば、これ以上の喜びはありません。

X（旧Twitter）のフォロワーたちに感謝します。彼らは、筆者が本を書いてみたいと呟いた構想のときから、楽しみにしていると返信してくれました。筆者が遅筆で悩んで進まないことを呟くと、励ましの言葉を送ってくれました。彼らがいなければ、執筆の進行はもっと悪くなり、途中で諦めていたかもしれません。これからも仲良くしていただけると幸いです。

本書の先行読者として協力してくれた人たちに感謝します。彼らは多忙な日々の中、貴重なひとときを割いて本書を読み、その感想を率直に寄せてきました。彼らの感想は「本書に寄せて」に掲載しています。これらの感想の中には、筆者では伝えられなかつた多くのことがあり、読者の皆様が本書を読む上で手助けになるはずです。また、誤字脱字や筆者の拙い文章にも指摘をいただきました。すべての指摘を反映することはできませんでしたが、筆者に新たな気づきを与えてくれました。貴重な時間を割いていただき、ありがとうございました。

妻に感謝します。執筆の時間を仕事に使って収入を得ていれば、もう少し生活を豊かにし、将来への貯蓄も増えていたはずでした。彼女は、これらの不安に辛抱強く耐え、毎日おいしいご飯を作つて筆者の生活を支えてくれました。また、本書を執筆するためにMacBookが欲しいという我儘にも快諾してくれました。彼女の協力のおかげで「本を書いてみたい」という夢を叶えることができました。これからは、真面目に働くようにします。

娘に感謝します。彼女と遊ぶはずの時間の多くを執筆の時間として使わせてくれました。文章の構成などに悩んでいるときには、おやつや笑顔を届けにきて筆者に一時の休息

---

を与えてくれました。遊びたい盛りに、父親としては誤った判断だったかもしれません  
が、パパは夢を叶えることができました。これからは、毎週1日は遊べる時間を作れるよ  
うにがんばります。一緒にベッドに入ると足で蹴り落とそうとせず、隣で寝ることを許して  
くれるでしょうか。

これまで多くの気づきを与えてくれた上司や同僚、一緒に働いてくれた多くの人たちに  
感謝します。彼らの言葉がなければ、本書の数々の教訓を得ることはできませんでした。  
また、成長の遅い筆者に根気強く仕事や技術を教え、困った時には一緒に尽力してくれた  
方々には感謝しきれません。彼らの言葉は厳しいこともありましたが、それらがなければ  
筆者はITエンジニアとして生きることを諦めていたかもしれません。

2023年12月 無能なポンプ

# 本書に寄せて

---

本書の発刊前に、X（旧Twitter）で先行読者を募り、その感想を掲載させていただきました。多くの応募の中から、可能な限り業界や経験が異なる人たちに依頼しました。彼らの経験やポジションは多様性に富んでおり、読者の皆様にとって参考になることや筆者では伝えきれなかった内容も多く含まれています。

また、感想をいただいたX（旧Twitter）フォロワーの「名前 @ユーザー名」を併記しました。彼らの日常的につぶやくポストにもご注目ください。掲載順番は、筆者の独断で読みやすい順番にしています。

## Shinji @sakichi01\_

本書は、ポンブ氏自身の経験を元にIT業界での生き残り戦略が具体的に書かれており、個人的に非常に共感を覚えました。

自身の苦労と失敗から学んだ教訓は、読者のみなさんが直面している課題に役立つと思います。各章が独立しているため、自分の興味に合わせて必要なアドバイスを見つけるのも容易いでしょう。

技術的な話だけでなく、コミュニケーションやチームワークの重要性に関する「ソフトスキル」の重要性を説く辺り、著者の読者に対する真摯さを感じます。

技術トレンドには触れていませんが、初心者やキャリアの初期段階にあるエンジニアにとっては問題ではないでしょう。技術者というよりも社会人としての成長を目指すすべてのITエンジニアに理想的なガイドブックの一つです。

## さっとん @NothingMyself37

ポンブさんの著書を読んで、真っ先に思ったのは「あんたは無能じゃない！」ということでした。本当に無能ならここまでわかりやすい文章でアウトプットできないからです。

読みながら、沢山メモを取りました。ネタバレになるのでお見せできないのが残念です。私にとっては金言ばかりで、大変有り難いものでした。これからも、X（旧Twitter）で応援していきます。執筆、お疲れ様でした。

そして、このような機会をいただきありがとうございました。

---

## 三日月 @Orcinus\_orcus

まずこの本は無能が有能になるためのハウツー本ではない。したがって購読したところで劇的に技術力が上がり、仕事ができるようになるわけでもなく出世しまくって稼げるようになるわけでもない。ではこの本は我々に何を提供してくれるのか？

ざつくばらんにいえば「無能が無能らしく生きていくために、頭に留めておくとちょっと助かることがあるかもよ？」的な内容が網羅されているのだ。

人はどうしても輪廻転生モノアニメのように一発逆転を夢見て、世の中に蔓延る情報商材に手を出してしまうことも少なくない。しかし、情報を得たところで有能になれるような吸収能力や実務に適応する応用力が備わっているのであれば、既に今の職場でも一定の評価をされているのではないか（そもそも有能）。

本書に書かれている内容も一見するとビジネスマンとして”当たり前”的なことが羅列されているように思える。

ただ自分自身または職場を見渡してみてこれらの事項が定常にできている組織、人物がどれほどいるだろうか？おそらく出来てない人がほとんどではないかと思う。

具体的には特に、II章のコミュニケーション、VI章の時間管理の考え方などはひょっとしたら新卒研修で教わった人もいるかもしれないが、今一度年次が経ったときにできているか読みながら振り返ってみることを勧める。

また本書のターゲットとして若手やメンバークラスを想定しているが中堅やリーダーになりたてでマネジメントに頭を悩ませている人にも併せてお勧めしたい。

立場が違えば大事とする事や優先順位も異なるため、小さなすれ違いがつもり、メンバとの関係性が悪くなったり、進むべき方向性を見失いがちになる。

そういう際にII章やVIII章の内容はその道しるべの一助となるかもしれない。

最後に重ねてになるが、特別な内容は何一つない。ただその所謂「普通」をこなすことがいかに難しいか、無能が無能らしく生き残るためにそれらを認識したうえでいかに現実でもがいていくか、それを改めて認識させてくれた本である。

---

## まさ@アップデートする情シス @tomatokechap18

本書は、純粋なエンジニアではない事業会社の情シスのマネージャーの視点からみても、とても内容が濃く、たくさんの気付きを与えてくれました。タイトルこそ「無能なITエンジニア」と自分自身を卑下したり、自虐的な内容を想像させますが、全く違います。

本書は、ポンブ氏の過去の失敗や苦い経験、仲間や先輩からの助言や言葉をベースにした「ごく普通のどこにでも居る大多数のITエンジニア」に向けた仕事と人生を豊かにするバイブルだと思いました。

### ▶ 第一印象

「え？ 269ページもあるの？<sup>†1</sup>」まずそのボリュームに愕然とした。そしてポンブ氏を舐めていたことを後悔した。もし、退屈な内容だったとしたら…269ページ読み切りさらに感想文を書いて送ると言う、苦痛かつ退屈な時間を浪費してしまう事になるのだ。そんな恐怖心を持ちながら10.4インチの小さなスマホ画面で読みはじめた。

<sup>†1</sup> 筆者からの補足です。先行者配信したものは「本書に寄せて」が未掲載だったため269ページで構成されていました。

### ▶ 序章を読んで

自分は普段、気合いを入れて読む本は、紙の書籍と決めている。気合いを入れて購入した電子本は一冊もない。

正直、今回の「無能なITエンジニアのための100の教訓」はノリで申し込んだ。恐らくポンブ氏のブログやコラムの寄せ集めだろう、くらいの先入観だったので、休日に寝転びながらスマホ画面で流し読みするスタイルで読ませてもらった。

序章に目を通した。目的や前提条件、対象読者、用語集、大まかな構成が淡々と書かれている。あれ？ 技術解説書のようなちゃんとした構成ではないか。

さらに目次を読み進めた時には、コラムのような少し軽めの内容を想像していた先入観は一気に吹っ飛んだ。

### ▶ 第Ⅰ章を読んで

「私たちの仕事は何でしょう」

第Ⅰ章は、いきなりこの問い合わせから始まる。さて、この問い合わせに対してどれだけの

---

人やエンジニアがきちんと自分の答えを持っているのだろうか？

ポンブ氏の答えはこれだ。

「私たちの仕事は問題を解決することです」

第Ⅰ章には、世の中の普通のITエンジニアに対する、ポンブ氏の想いと1番伝えたいことが書かれていた。

ポンブ氏は、本書で「世の中にたくさんいる普通のエンジニア」に対し、システム開発や運用と言う業務を通して、仕事や人生の本質について気付きや学びを与えるのではなかいか、と思った。

そして、この最初の問いかけとその答えこそ、以降Ⅱ章からX章まで、約250ページにわたる100の教訓のベースとなる考え方なのではないか？と感じとった。

だから、第Ⅰ章は、絶対に読み飛ばさないで欲しい。さらに、1番最初に読んで欲しい。但し、第Ⅰ章には、技術的なことは一切出てこないので、エンジニアの読者の方々は興味を失って読み飛ばしてしまうかもしれない。

#### ▶ 最後まで読み切った感想をみつつ

①仕事をする上での普遍的なテーマなので、エンジニアだけでなく、他の職種の人にも役立つ内容である。たくさんのビジネスマンに紹介したい。

②ポンブ氏の過去の失敗や苦い経験、仲間や先輩からの言葉がベースになっているので、同じ経験をしている多くの人は共感することがたくさんあると思う。逆にエリート街道まっしぐらの自信家の上から目線野郎には響かないだろう(読者層ではないので関係ない)。

③エンジニアらしく、過去や現状、現実を大切にしている。良くある未来志向のイケイケな教訓は一つもない。これは、未来だの、成長だの、変革だのとイケイケなことばかり発信している自分に対して「そればかりでもダメだよ」と警鐘を鳴らしてくれたように思えた。

---

## ▶ 番外編

番外編として、特に自分が印象に残った教訓や情シスのマネージャーとして実業務では是非実践したいと思った教訓をリストアップした。

- 1.7 正しいモノサシを持つ
- 1.9 頑張ったから良い評価ではない
- 1.10 不満を垂れる前に行動を
- 2.7 怒っているのではなく、困っている
- 2.8 綺麗な格好ではなく、似合う格好を
- 2.10 自分の時間、他者の時間
- 3.3 大きさは2倍、早さは半分
- 3.7 基本はみつつ
- 4.3 点で捉える
- 4.8 偽りのフルスタックエンジニア
- 5.1 模倣から始める
- 5.9 手取額10%
- 6.8 無駄な時間？私には必要です
- 6.9 ヒトは1週間、モノは1ヶ月
- 7.2 チームワークはメンバーから始まる
- 7.5 誤った協調性
- 7.8 違和感を大切にする
- 8.1 やはり銀の弾などなかった
- 8.9 顧客の声は誰のもの
- 8.10 マネジメントとメンバーの兼任
- 9.2 休日に勉強はするべきか
- 9.6 ワークライフバランスの幻想
- 10.1 成長による停滞、成長なき後退
- 10.6 転職の葛藤
- 10.8 独立という考え方
- 10.10 未来よりもイマを大切に

---

## かえるくん。 @kaerukun\_geko

はじめに、この本に出会えたことを著者であるボンブ氏に感謝申し上げます。本を完成させるまでには並々ならぬご苦労があったかと思います。その苦労と強い想いが形になり本という形でこの世に産み落とされたことを非常に喜ばしく思います。

あわせて奥様とお嬢様にも感謝申し上げます。家族の理解がなければ本を作り上げることは到底成し遂げることができなかつたことは想像に難くありません。ご家族の理解と協力があってこそ完成できたと思います。一読者として感謝いたします。

そして、先行版読者として選んでいただきありがとうございました。ボンブ氏を知ったのは数年前だったかと記憶しております。歯に衣着せぬウィットに富んだ煽り芸を楽しく拝見させていただいておりました。最近は煽り芸の鋭いナイフが摩耗してしまい切れ味が今ひとつ感じているのも事実です。以前のような切れ味のよい煽り芸を楽しみにしています。

接点がほとんどなくクソリプを送っていた私が、先行版読者に選ばれたのはなぜなあせえ？という気持ちがありますが、誰よりも早く読ませていただけたことを改めて感謝申し上げます。

タイトルは「無能なITエンジニアのための100の教訓」となっていますが、内容はITエンジニア向けというより、もっと広く働く人たちみなさんに必要な教訓だと思います。

著者自身はこの教訓を実践できていないから無能と感じておられるようですが、私が働いてきた世界では私自身も含め、実践できている人はほとんどいませんでした。いかに、著者が志高く仕事をされたかは想像に難くありません。その結果、理想と現実のギャップでご自身を無能と思うようになってしまったのだと感じました。そもそも理想が高すぎたのだと思います。

私自身は、教訓に書かれている内容に気付くまでにかなりの年月がかかりました。実際にこれらの教訓に気づいたのは、ここ数年のことです。これらの教訓を単に頭で理解するだけでなく、体系的に文章にまとめ上げ、さらにはそれを本として出版するに至った著者の能力は、決して無能なものではありません。むしろ、そのような業績を成し遂げた著者は、疑いのない優秀さを持っている方です。

---

---

少し本の中身に触れさせていただくと、私が仕事をする上で一番大事にしているのはコミュニケーション（自分自身課題と認識し常にアップデートを心がけている）なのですが、その中でも著書にも書かれている、「2.6 ありがとう、ごめんなさい」・「2.10 自分の時間、他者の時間」を日々の仕事で非常に大切にしています。これができない人が多すぎて荒れるぐらいです。内容に共感しすぎてつぶやいてしまいました。もちろんこれらの教訓だけではなくほかの教訓からも自分にとってまだまだ足りない部分や、気がついていないことも多く学ばせていただきました。

X（旧Twitter）では優秀なエンジニアさんが目立ち、比較すると落ち込むこともあります、ポンプ氏の本から得られた教訓を一つ一つ習得しレベルアップしていくことで、今後の社会人人生を少しでも意義あるものにしていきたいと考えています。

ただ、残念なことに本当に無能な人は自分のことを無能だとおもって(気がついて)いないので、この本を手にとる機会がないかもしれません。この本を手に取って読む人は少しでも成長したいと考えている少し優秀な人だと思います。

この貴重な機会をいただいたことに心から感謝し、いつか直接お会いできる日を楽しみにしています。

※読書感想文を参考にして本を購入したことで生じた後悔や損害について、かえるくんは一切責任を負いません。また、損害賠償や謝罪の要求にも一切応じません。

かえるくん。 (Powered by ChatGPT)

**ioilante @ioi\_lante**

この本は、有能者でなくとも「魔境・IT業界」を生き抜くためのバイブルです。ポンプ氏直伝のサバイバル術で生き延びることができるでしょう。

ITの世界は魔境です。今日の最新技術が来週には陳腐化したり、今イケてるスキルが来月はゴミになることは日常茶飯事です。そんな中で、自分の居場所を見つけ、長く生き残

---

することはとても大変です。しかし、ボンブ氏のこの本を読めば「生き方」のヒントを得られるかもしれません。

この本は、ITに関するテクニカルガイドではありません。ITエンジニアとしての生き残り方・・・特に「有能ではない」と自覚している人たちに向けた、リアルなサバイバル術が詰まっています。ボンブ氏が自身の経験から得たノウハウを詳細に図入りで解説しており、「これなら私にもできるかも」と思える内容です。

例えば、プロジェクトでミスをした時の対処法、上司とのうまい付き合い方、チーム内の信頼を築くコツ、私生活でどうするべきか・・・など具体的で役立つ体験談が満載です。これを読んで自分で活用できるポイントを真似すれば、明日からでも生き方そのものが変わるかもしれません。

「無能なボンブ」という名前は自虐的な響きですが、ボンブ氏自身がITエンジニアとして完璧ではないと感じているからこそ、他の「普通～普通未満のエンジニア」の苦悩が分かり、だからこそ、非常にリアルで心に響きました。

※なおボンブ氏の言う「無能」はあくまでもITエンジニアとしてのお話です。生き方としてはとても有能の類です。そこに関するヒントになりえる章もあります。

ITという魔境で生き抜くためには、技術だけでなく、人間関係やストレスマネジメントも大切です。この本には、そういった「生きる知恵」がたくさん詰まっています。100の教訓のうち20でも参考になれば生き方自体が変わるかもしれません。

視力の問題で活字が読みにくい自分でもサクサク読めました。周回数を重ねると新しい発見もありました。読んで損はないと思います。是非お手に取ってみてください。

あなたのITエンジニアとして・・・いや、人生そのものに新しい風を吹き込んでくれる可能性があります。

---

## あるだみあ @arudamia

まず初めに、本当に本を出されると思っていなかったので実際に頂いた際とても感動しました。本書は無能なエンジニアに向けた本ということで、新卒1年目の無能なエンジニアである自身のためにも、今後長い目で関わるであろう新人達の人生の一助となるためにも期待を込めて読ませていただきました。

これから長い社会人生活全般に活かせる立ち回り方や考え方、エンジニアとしてのプライベートの過ごし方まで。無能でもそうでなくともエンジニアという仕事をしていく上で、付き合っていかなければならない悩みについて網羅的に書かれています。

1人の無能として今後、長く生き延びる為に活かして行きたいと思いました。

## yashi @yu\_geen269

ポンブってさ、いつも無能無能って言ってるくせに、ほんまは「有能」やろ～！、って思ってきたわ。この本、執筆するだけのことはあるなって。最初は「無能～」って、単に煽ってるだけのタイトルやろなって思っててん。でも、ちょっと読み進めてみたら、なんやこれ、ポンブめっちゃマジメに書いてるやんけ！？

有能なポンブが、無能な俺たちに教えを説いてくれてるんやと思ってな。ITエンジニアを生き抜くノウハウみたいな、聖書みたいな感じやんか。最初はウケ狙いかと思ったけど、結構まともなアドバイスがあるで。

内容はあえて書かへんから、ちゃんと読むんやでっ！ITエンジニア以外でも、普通のサラリーマンや会社員にもオススメやで。他のビジネス書買うんやったら、この本を読みはなれ！ これ新入社員にも、配った方がええんとちやうかあ？！

褒める気全くなかったのに、なぜか褒めてしまったわ。



# 本書を読む前に

---

## 本書の目的

本書は「無能なITエンジニアが、無能なまま生き続ける」ことをコンセプトにしています。周囲の人たちより劣っている場合でも、どうにかしてIT業界でエンジニアを続けるための内容です。本書を読んでも、無能でなくなったり優秀なエンジニアに成長することはありません。内容の多くは、考え方や行動に関するものになっており、以下のような要素は含まれていません。

- 便利なツールや使い方の紹介
- 最新技術や人気のある技術分野の紹介
- 仕事を効率化して最大成果を出す方法

また、明日からすぐに使えるお手軽な方法や近道をする方法もありません。教訓の多くは、ある程度の時間か反復による習慣が必要なものです。

しかし、教訓の内容は小手先の方法や使い捨てのようなものではありません。今後の長いITエンジニア人生の中で使える機会が多く存在するものです。

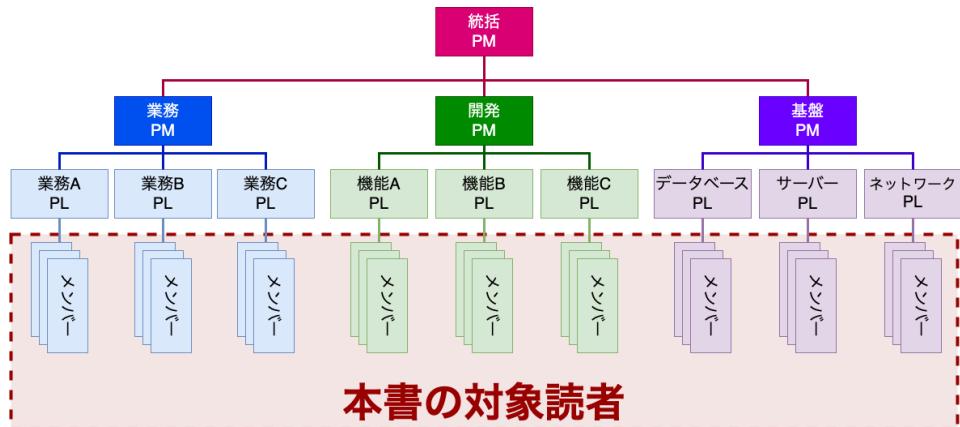
## 本書の留意点

本書を読む上で、2つの留意点をお伝えします。

1つ目は、本書では「～べきです」や「～べきではありません」などの言い切りの表現を多数使用しています。表現の方法について悩みましたが、最終的に言い切る表現を使うことにしました。これらは、筆者が強い意志をもって伝えたいことだったからです。しかし、どのような職種や年代にも当てはまる原理原則のようなものがあるとは考えていません。言い切りの表現が腑に落ちない場合は、「筆者はこのように強く思っているんだなあ」という気持ちで読んでください。

2つ目は、本書の教訓は筆者の経験から得られたものです。経験したことの説明のために、筆者の経験談や案件の事例を記載していますが、人物や案件が特定できない程度に抽象化したり、意味が変わらない範囲で改変しています。

## 本書の対象読者



本書の対象読者としている無能なITエンジニアとは、プロジェクトのメンバーとして数ヶ月以上の経験があり「頑張っているのに・・・」の後に、以下のようなことが当てはまる人を指します。

- 他の人のように成果が出せない
- 他の人はしないような失敗をする
- 技術の習得や仕事が遅いと感じている
- 怒られたり、落胆されることがよくある
- 使えないやつと言われたり、悪い評価をされることがある
- 給与に見合った働きが出来ていない
- 何をすればいいのかわからない

これらの厳密な判断は難しいところですが、どれにも悩みを感じていなければ、本書を開いてはいないでしょう。

現場を経験していない未経験者の方でも本書を読むことは難しくありませんが、共感できることは少ないかもしれません。

---

## 本書で使う言葉

本書で多用する以下の言葉の意味についてまとめました。

言葉	意味
私たち	筆者も含め本書の対象読者である無能なITエンジニアを指しています。
組織	一般的な法人の会社のことを指します。読者の中には、非営利団体や教育機関などに所属する人もいるかと考え「組織」と表記しています。
顧客	私たちのお客様を指します。顧客は組織外だけとは限りません。例えば、情報システム部門の場合は、自社のシステムの利用部門や業務部門は顧客になります。
プロダクト	私たちが取り扱う製品を指します。ハードウェア製品だけでなく、ソフトウェア製品やクラウドサービスなどすべてを含みます。
プロジェクト	目標を達成するための活動を指します。案件名「〇〇プロジェクト」のような意味で使用しています。案件名の無い少人数の場合でも、目標が定められているものはすべてプロジェクトとして扱っています。
チーム	プロジェクトを達成するための人の集まりの意味で使用しています。これらは単体のチームの場合もあれば、複数のチームの場合もあります。
メンバー	チームを構成するITエンジニアを指しています。
PM	プロジェクトマネージャーを指します。プロジェクト全体の計画・実行・監視・制御を行う役割の人です。
PL	プロジェクトリーダーを指します。技術分野やチーム単位でプロジェクトに必要となることを計画・実行・監視・制御を行う役割の人です。
タスク	私たちが行う処理や操作を指します。1つのタスクは数分程度で終わるものもあれば、数日になるものもあります。仕事としては終わらないけれども、1つの処理の区切りがあるものをタスクとしています。
仕事	タスクの集合体です。私たちの役割として果たすべきことの、始まり～終わりまでを1つの仕事として表記しています。

---

## 引用・参考文献について

本書では以下の内容について「†（ダガー）」を使い補足説明を行っています。

- 誤解を招く可能性のある表現
- 複数の意味に捉えられる用語
- 筆者が補足を必要と考えたもの

これらの補足説明とともに、引用や参考にした文献について以下のルールで記載しています。

### インターネットの場合

文献がインターネット上のWebサイトやPDF文書の場合は、以下のルールで表記しています。なお、確認日時は筆者が最後に確認した年月日を記載していますが、Webサイトから更新日時が明確に確認できる場合は更新日時を記載しています。

#### 【ルール】

サイト名. タイトル. 確認日時

URL

#### 【例】

ITエンジニアのまとめ. 無能なITエンジニアのための100の教訓. 2023年10月1日

<https://itenginner-matome.net/archives/42879>

### 書籍の場合

文献が書籍の場合は、以下のルールで表記しています。なお、参考ページ数は明確に参考となった文章がある場合のみ記載し、書籍全体を指す場合は参考ページ数の記載を割愛しています。また、筆者が持っている書籍になるため、改訂などにより新しい書籍が発行されている場合があります。

#### 【ルール】

著者名. 書籍タイトル. 出版社. 出版年月. 参考ページ数

#### 【例】

無能なポンプ. 無能なITエンジニアのための100の教訓. 2023年10月. 1ページ

# 本書の構成

---

本書は、序章・本章・付章の全12章で構成されています。本章のI章～X章は、各章に10の教訓があり、1つの教訓は見開き2ページで構成するようにしています。

どの教訓から読んでも見開き2ページで完結するようになっていますが、読者が必要となる教訓を発見しやすくするために、各章の概要について記載しておきます。

## 序章

筆者が本書を執筆した経緯や読者対象などの前提条件を記載しています。また、筆者のX（旧Twitter）フォロワーからの本書の感想を記載しているため、他者の見解を知ることができます。

## 第Ⅰ章 無能のための考え方

どれほどITの技術を学び、誇りを持っていても、考え方方が誤っていては生きていくことは難しいでしょう。技術の前に考え方を見つめる必要があります。

本章は、他の章の根幹となる考え方をまとめています。さまざまな視点からの考え方が登場するため、少々混乱するかもしれません、他の章を読んだ後に本章を読み返せば、より理解を深めることができるでしょう。

## 第Ⅱ章 無能のためのコミュニケーション

コミュニケーションと聞くと持って生まれた才能や高度なテクニックが必要と思うかもしれませんが、私たちに必要なのは、仕事上で必要となる意志伝達と信頼関係だけです。

## 第Ⅲ章 無能のためのプレゼンテーション

大切なことは、どのように伝えるかです。日常的に行われる会議や会話なども含めたプレゼンテーションに大切なことを見ていきましょう。

## 第IV章 無能のための技術

私たちに必要なのは、最先端や流行りの技術を扱えることではありません。仕事の役割に求められる技術を正しく扱えることです。

---

## 第V章 無能のための学習

最新の技術情報についてもインターネットを用いて情報を入手できるようになりました。私たちは、学習する目的を定め、必要となる学習方法を選択しなければなりません。

## 第VI章 無能のための時間管理

プロジェクトのスケジュールは、私たちが詳細に管理する必要はないでしょう。私たちのタスクを期限内に達成するためには、個人レベルでの時間管理が必要です。

## 第VII章 無能のためのチームワーク

プロジェクトチームとして複数の人たちと協力する際、チームワークは非常に大切です。私たちが、チームのメンバーとして貢献できることは技術以外にも多く存在します。

## 第VIII章 無能のためのマネジメント

本章のマネジメントには、プロジェクトマネージャーとプロジェクトリーダーが含まれます。そして、本章に限り「私たち」はメンバーではなくマネジメントの役割を指します。私たちがマネジメントを行う際は、ITエンジニアとは異なる視点が必要です。

## 第IX章 無能のためのプライベート

完全に安息したプライベートの時間を過ごせることは少ないかもしれません。プライベートを仕事に支配されないように考える必要があります。

## 第X章 無能のための未来

急速に変化するITの世界で、正しいか間違っているかもわからない中から自分の未来を選択しなくてはなりません。正しさは誰にもわかりませんが、間違った選択をしないようにしなければなりません。

## 付章

本書の教訓は多くの人たちの助言や叱咤激励の言葉から生まれました。本章では、教訓が生まれるきっかけになった言葉とその言葉を得たときの状況を紹介します。

# 目次

---

はじめに	3
序章	5
謝辞	7
本書に寄せて	9
本書を読む前に	19
本書の構成	23
目次	25
第Ⅰ章 無能のための考え方	31
1.1 仕事は問題を解決すること	33
1.2 自身の責務範囲を知る	35
1.3 責務範囲のモレとダブり	37
1.4 限りなく最適解に近いもの	39
1.5 役割と上下関係	41
1.6 ウォーターフォールがもたらした最悪のもの	43
1.7 正しいモノサシを持つ	45
1.8 水はモノサシでは測れない	47
1.9 頑張ったから良い評価ではない	49
1.10 不満を垂れる前に行動を	51
第Ⅱ章 無能のためのコミュニケーション	53
2.1 すべてのはじまり	55
2.2 事実と感情	57
2.3 コミュニケーションの種類	59
2.4 一方向は存在しない	61

---

2.5 コミュニケーションに終わりはない	63
2.6 ありがとう、ごめんなさい	65
2.7 怒っているのではなく、困っている	67
2.8 綺麗な格好ではなく、似合う格好を	69
2.9 リモートワークで失うもの	71
2.10 自分の時間、他者の時間	73
<b>第III章 無能のためのプレゼンテーション</b>	<b>75</b>
3.1 プrezentationは自己紹介	77
3.2 文章の前にストーリーを考える	79
3.3 大きさは2倍、早さは半分	81
3.4 立った！立ったわ！	83
3.5 ホワイトボードは怖くない	85
3.6 失礼という誤解	87
3.7 基本はみつつ	89
3.8 内職はしない	91
3.9 長時間会議は悪か	93
3.10 全員に愛される必要はない	95
<b>第IV章 無能のための技術</b>	<b>97</b>
4.1 はじめてのものには、恐怖と敬意と楽しさを	99
4.2 動いた感動、理解した感動	101
4.3 点で捉える	103
4.4 全く同じは存在しない	105
4.5 おまじないという嘘	107
4.6 できないは出発点	109
4.7 必要なものはモダンではない	111

---

---

4.8 偽りのフルスタックエンジニア	113
4.9 私たちは失敗する	115
4.10 未経験者と経験者の違い	117
<b>第V章 無能のための学習</b>	<b>119</b>
5.1 模倣から始める	121
5.2 資格だけでは身を滅ぼす	123
5.3 暗記ではなく、できることを知る	125
5.4 知らないを知る	127
5.5 学びは歴史とともに	129
5.6 学習媒体を使い分ける	131
5.7 情報源は2つ以上	133
5.8 良書との出会い方	135
5.9 手取り額10%	137
5.10 検索エンジンへの聞き方	139
<b>第VI章 無能のための時間管理</b>	<b>141</b>
6.1 時間管理術ができない理由	143
6.2 予定は日で考える	145
6.3 目的と期限の明確化	147
6.4 ズレを許容する準備	149
6.5 実績と私のスケジュール	151
6.6 空き時間なのか、待ち時間なのか	153
6.7 私の集中できる時間	155
6.8 無駄な時間？私には必要です	157
6.9 ヒトは1週間、モノは1ヶ月	159
6.10 残業は悪か	161

---

<b>第VII章 無能のためのチームワーク</b>	<b>163</b>
7.1 同じ方向を向いて進む	165
7.2 チームワークはメンバーから始まる	167
7.3 局所的リーダーシップ	169
7.4 教えてを言う勇気	171
7.5 誤った協調性	173
7.6 柔軟性と譲れない範囲	175
7.7 経験年数で判断しない	177
7.8 違和感を大切にする	179
7.9 横目と仰望	181
7.10 面倒なことは、みんな面倒	183
<b>第VIII章 無能のためのマネジメント</b>	<b>185</b>
8.1 やはり銀の弾などなかった	187
8.2 完璧を捨てる勇気	189
8.3 不満は開始時からはじまっている	191
8.4 些細な苦情はヒント集	193
8.5 嫌われながら好かれる	195
8.6 使えないメンバー	197
8.7 確認、確認、そして確認	199
8.8 自分の限界を知る	201
8.9 顧客の声は誰のもの	203
8.10 マネジメントとメンバーの兼任	205
<b>第IX章 無能のためのプライベート</b>	<b>207</b>
9.1 ITエンジニア、恋愛のすゝめ	209
9.2 休日に勉強はするべきか	211

---

---

9.3	そこに信号があります	213
9.4	休むという強い覚悟	215
9.5	お風呂と通勤と私	217
9.6	ワークライフバランスの幻想	219
9.7	休み中の待機という矛盾	221
9.8	趣味という逃げ場所	223
9.9	気の置けない仲間たち	225
9.10	他者と比較する前に自分の充実を	227
<b>第X章 無能のための未来</b>		<b>229</b>
10.1	成長による停滞、成長なき後退	231
10.2	無くなるのではない、変わるものだ	233
10.3	バズワードの危険性	235
10.4	具体的な未来は考えない	237
10.5	35年定年説は崩れたもの・・・	239
10.6	転職の葛藤	241
10.7	スカスカな職務経歴	243
10.8	独立という考え方	245
10.9	高学歴化への備え	247
10.10	未来よりもイマを大切に	249
<b>付章</b>		<b>251</b>
第I章 無能のための考え方		253
第II章 無能のためのコミュニケーション		255
第III章 無能のためのプレゼンテーション		257
第IV章 無能のための技術		259
第V章 無能のための学習		261

第VI章	無能のための時間管理	263
第VII章	無能のためのチームワーク	265
第VIII章	無能のためのマネジメント	267
第IX章	無能のためのプライベート	269
第X章	無能のための未来	271
あとがき		273



第

# I

章

## 無能のための考え方

本章は、無能なITエンジニアとして生きていくための  
基本的な考え方についての教訓になります。

どれほどITの技術を学び、誇りを持っていても、  
考え方方が誤っていては生きていくことは難しいでしょう。

技術の前に考え方を見つめる必要があります。

**1 仕事は問題を解決すること**

**2 自身の責務範囲を知る**

**3 責務範囲のモレとダブり**

**4 限りなく最適解に近いもの**

**5 役割と上下関係**

**6 ウォーターフォールがもたらした最悪のもの**

**7 正しいモノサシを持つ**

**8 水はモノサシでは測れない**

**9 頑張ったから良い評価ではない**

**10 不満を垂れる前に行動を**

## 1.1 仕事は問題を解決すること

---

私たちの仕事は何でしょうか？

- プログラミングをすること
- データセンターで機器の設定を行うこと
- プロジェクトを推進すること
- 顧客と対話して要件をまとめること
- 先輩から渡された参考書を理解すること

これらすべてが正解かもしれません。「言われたことをやっている」だけであれば、与えられたタスクをこなしているだけです。「仕事」をするためには、与えられたタスクに自らの意志で取り組み、目標を達成するために考え、行動しなければなりません。

私たちにとって、もっともシンプルな定義は「仕事は問題を解決すること」です。仕事は、ニーズや要望から生まれています。例えば「こんなアプリケーションが欲しい」や「こんな機能が欲しい」という要望があり、私たちは「どうすれば実現できるか」という問題の解決に取り組んでいるのです。私たちの多くが嫌気をさし、一部のITエンジニアがウキウキしているような障害対応、不具合対応、炎上プロジェクトの火消しなども、それらの問題解決に取り組んでいます。

問題は単純なものばかりではありません。1つの問題に見えて複数の問題から構成されている場合があります。例えばシステム停止を伴う障害対応では、以下のような問題に取り組む必要があります。

1. 特定の期間どうにか動かしたい（応急処置）
2. 正常動作をさせたい（縮退運転しパフォーマンス劣化は許容する）
3. 正常動作をさせたい（完全な元の状態に戻す）
4. 原因を究明し二度と同じ障害を発生させないようにしたい

このように「障害対応をする」問題の中には、複数の問題が含まれています。そして、どの問題を優先するかは状況により異なります。例えばシステムが使えない状態の場合「4.」を優先して取り組むよりも、正常動作に復旧することが優先でしょう。では、

「2.」を優先すべきでしょうか？極端なパフォーマンス劣化は、使えない状態よりも「顧客からの異なる内容の問い合わせが増大する」などの新たな問題を発生させることもあります。優先順位をどのように決定するかは注意が必要です。

また、対応しなくて良い問題があるかもしれません。最終的に「4.」の問題を解決することが正解にも思われますが、原因を究明できた後「二度と同じ障害を発生させない」ために巨額の予算が必要だった場合、この問題を解決する必要はないかもしれません<sup>†1</sup>。

私たちが仕事をするとき、2つの重要なポイントがあります。

1. 問題解決に取り組む際に、どのような問題があるかを把握すること
2. どの問題を優先して取り組むべきか判断すること

これは学生時代のテストと似ています。「テストで高得点を取得する」ためには、問題文を正しく理解する必要があります。解く順番がある問題を、最後に記載された問題から取り掛かっては、どれだけ考えても正解を導くことはできないでしょう。

私たちの仕事は学生のテストよりも難問です。問題は明らかにされておらず、解いていくべき順番も定まっていません。私たちは、問題を見つけ解いていく順番を決める必要があります。

しかし、悲観的なことばかりでもありません。幸いなことに、いつでもカンニングをすることができます。

- インターネットや参考書を使って調べる
- 知識のある同僚やチームメンバーたちから助言を得る
- 便利なツールやサービスの力を借りる

これは不正ではなく、正しい行為です。多くのITエンジニアは、これらの行動を素晴らしいと評価するでしょう。

このような素晴らしい行動は、私たちが「仕事」をしなければできることです。「与えられたタスクをこなしている」だけでは、問題に気づくことさえもできないでしょう。

あなたの仕事は何でしょうか？どのような問題の解決に取り組んでいますか？

<sup>†1</sup> その代わり、少ない予算で実現可能な「同じ障害が発生しても影響がない方法はなにか？」というような新たな問題に派生することもあります。

## 1.2 自身の責務範囲を知る

責務範囲<sup>†1</sup>とは「義務を果たすべき責任の範囲」です。私たちがプロジェクトで複数の人たちと協力して仕事を行う場合、責務範囲を定義し、認識を一致させることで仕事を円滑に進めることができます。組織であれば契約書、営業の立場では見積書、プロジェクトの場合にはプロジェクトスコープ、ITエンジニアであれば規模見積などがあります。私たちに身近な例としては、アプリケーションの利用規約・サービスレベル契約（SLA）・プログラミングにおける変数のスコープも明確な範囲の定義があるため安心して使うことができます。このように、多岐に渡るシーンで「する範囲」「しない範囲」が明確に規定されています。

ここまで例では、主に文書や仕様書などに明記される相対的に広い範囲の例を取り上げました。では、私たちの仕事の責務範囲とはなんでしょうか？プログラマの場合、担当した機能の実装がその範囲になり、ネットワークエンジニアの場合、ネットワーク機器の設定になるでしょう（図1-2-1<sup>†2</sup>）。



図1-2-1 責務範囲

私たちがタスクを遂行するために「自身の責務範囲を知る」ことは重要ですが、責務範囲を把握せずにタスクに取り組む人たちも存在します。彼らは問題が発生したり、責務に関する話になると呪文を唱え始めます。

1. 言われた通りにやった
2. 知らなかった
3. 私の範囲ではない（から、やらない）

これらの呪文は、上司・先輩・チーム全体・協力ベンダー、ときには顧客へ向けて唱えられることもあります。残念ながら、筆者の経験上これらの呪文が問題を解決した事例は、ほとんどありません。これらの呪文は、他の人たちに次のように聞こえています。

1. 言われたこと「しか」やらなかつた
2. 知ろうとしなかつた
3. 私の範囲と認識できていなかつた（から、できない）

「自身の責務範囲を知る」ことには、以下の3つの重要な側面が存在します。

## 仕事の遂行

1つ目は、仕事の遂行です。私たちはチーム全体で協力して仕事を進めており、個々のメンバーが自身の責務範囲を遂行することで、プロジェクトの仕事が完了できます。「自身の責務範囲を知る」ことは、私たちの仕事を遂行するためだけではなく、プロジェクトの仕事の遂行にも必要なものです。

## 影響範囲の理解

2つ目は、影響範囲の理解です。個々に与えられた責務範囲には、技術的要素以外にもスケジュールや他の連携したシステム、運用管理に関わる要素も含まれることがあります。「自身の責務範囲を知る」ことで、自分の行動がどの範囲に影響を及ぼすか認識できるようになります。

## トラブル時の対処

3つ目は、トラブル発生時の対処です。計画的に進めていてもトラブルが発生することがあります。「自身の責務範囲を知る」ことができていれば、トラブル発生時でも有意義な話し合いが可能であり、短期間で解決に向けた取り組みを行うことができます。

個人のタスクレベルでは、責務範囲が文書として明確に示されないことも珍しいことはありません。しかし、私たちが積極的に知ろうとしなければ、その不明確さは私たちに多くの不幸をもたらします。

†1 「責任範囲」と呼ぶ方が一般的かもしれません。筆者は、仕事に「責任」と「義務」の両方が必要と考えているため「責務範囲」と呼ぶようにしています。

†2 「フロントエンド」「バックエンド」の明確な定義はありません。本節では「フロントエンド」は目に見える部分のプログラマ・コーダー、「バックエンド」は処理を担うプログラマとして表記しています。

## 1.3 責務範囲のモレとダブリ

自身の責務範囲を知ることは重要ですが、同時に「モレ」と「タブリ」があることを常に意識することも必要です。範囲と聞くと、一律に区分されているように感じられるかもしれません。役割や属性に合わせて一律に区分し「モレ」や「ダブリ」を避ける方法論にMECE（ミーシー）<sup>†1</sup>などもありますが、それでも現実では「モレ」や「ダブリ」を避けられないことがあります（図1-3-1<sup>†2</sup>）。

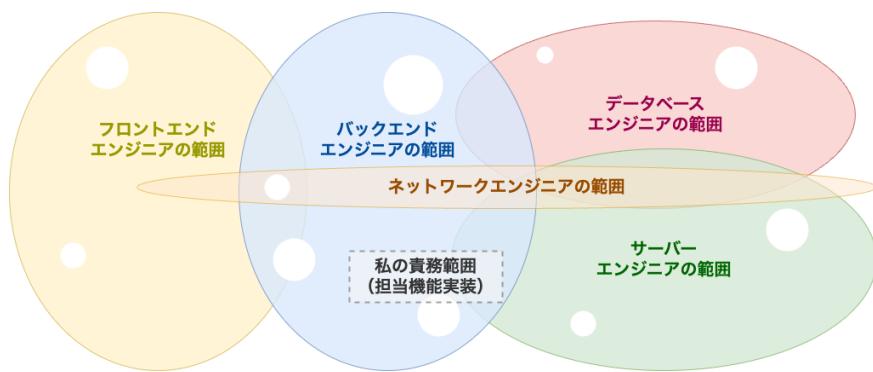


図1-3-1 現実の責務範囲

私たちがバックエンドエンジニアで担当している機能実装を「自身の責務範囲」とイメージしてみましょう。

### 責務範囲のモレ

図1-3-1の白い円が「モレ」を表しています。「モレ」が発生する要因は様々です。

- 責務範囲である私たちの考慮不足
- 要件定義時点でのヒアリング不足
- ビジネス変化に伴う仕様の変更

これらの要因は、私たち自身に落ち度がない場合にも頻繁に発生するものです。

### 責務範囲のダブリ

図1-3-1の各ITエンジニアの範囲の重なる部分が「ダブリ」を表しています。私たちが

関与するシステムは多くの技術要素が連携し合って動作しているため、「ダブリ」は避けられない状況です。この「ダブリ」は必ず存在するため、プロジェクト計画書や設計などで範囲が明確に定義され、一般的に問題となることは少ないでしょう。しかし、開発中や障害などのトラブル時には「ダブリ」の部分が原因となっているか不明確な場合もあり、混乱を招くことがあります。

「モレ」や「ダブリ」は、可能な限り排除するのが理想ですが、必ずしも理想のように排除できるとは限りません。では、それらの状況にどのように対処すべきでしょうか？事前に排除できなかった場合には、協力しかないと筆者は考えています。

「モレ」が私たちの考慮不足によるものであれば、それを是正するためにPMやPL、メンバーと相談しスケジュールやタスク調整などの助けを必要とするかもしれません。私たちが原因ではない場合も「自身の責務範囲」に影響するものであれば、共に解決方法を模索しなければいけません。

「ダブリ」についても同じです。私たちにトラブルが発生しているのであれば、その原因の調査や解決に必要と思われる「ダブっている」別のチームに協力の要請をする必要があります。逆に他のチームから要請があることも考えられます。トラブルの解決には、各ITエンジニアが調査するための処理内容・操作内容・ログ・発生時間など多くのものを共有する必要があります。

これらの対応は、しばしば私たちを疲弊させます。時間がなくなり本来のタスクは遅れ、残業をする必要があるかもしれません。それでも協力を惜しむべきではありません。

「モレ」や「ダブリ」が見つかった場合の多くは、「自身の責務範囲」が「関係者たちの責務範囲」に変わった瞬間です。私たちが協力を欠かせば、それらの対応に、より多くの時間がかかるだけでなく、私たち以外の多くの人たちが疲弊する可能性が高まります。

私たちが「モレ」や「ダブリ」を自身の責務範囲ではないと捉えれば、「私の範囲ではない」という呪文を周囲に唱え始めるでしょう<sup>†3</sup>。

†1 MECE (Mutually Exclusive and Collectively Exhaustive) は、コンサルティング会社マッキンゼー・アンド・カンパニーにより開発された「漏れなく・ダブりなく」物事を整理する論理的手法の1つです。

†2 「フロントエンド」「バックエンド」の明確な定義はありません。本節では「フロントエンド」は目に見える部分のプログラマ・コーダー、「バックエンド」は処理を担うプログラマとして表記しています。

†3 契約や役割により本当に範囲外である場合もあります。しかし「範囲内なのか？」「範囲外なのか？」の確認に周囲の協力は必要になります。

## 1.4 限りなく最適解に近いもの

---

学生時代にテストで100点を取った経験はありますか？筆者は、小学生の国語のテストが最後の100点だったことを記憶しています。中学生以降のテストはあまり良い結果ではなかったため、思い出すのも辛いことですが、皆さんの中には数多くのテストで100点を取った方もいることでしょう。

それでは、社会人になって自身が参画するプロジェクトで100点を取った経験はありますか？おそらくほとんどいないでしょう。学生時代のテストは、予め個人の知識で解答を導き出せるよう作られた問題が多い一方、プロジェクトには明確な正解がないことがほとんどだからです。

その代わりに、プロジェクトには成功と失敗という明確な結果が存在します。成功、失敗の判断基準はさまざまであり、組織やプロジェクトによって異なるため、ここでは詳しく述べることができませんが、私たちは少なからず成功したプロジェクトを目にしてきました<sup>†1</sup>。

成功しているプロジェクトには、共通する要素があります。それは、最適解を模索し続けていることです。私たちの身近には「技術的最適解<sup>†2</sup>」があります。技術的最適解では、技術のベストプラクティス（最適な方法）を採用しバッドプラクティス（悪しき方法）を排除することを重視します。

ITの歴史を振り返ると、管理・設計・実装・運用など、さまざまな分野でベストプラクティスが考案され、バッドプラクティスが明らかにされてきました。これらのノウハウは、私たちが技術的最適解を見つける際の手がかりになりますが、2つのことを忘れてはいけません。

### 技術的最適解とプロジェクトの最適解

1つ目は、技術的最適解が必ずしも「プロジェクトにおける最適解」ではないということです。私たちは設計者としても実装者としても、担当する部分にベストプラクティスを採用しようとします。しかし、現実のプロジェクトでは、人材・納期（時間）・予算など制約の中で成功を達成しなければなりません。どれほど効率的で洗練されたコードを書いても、完成までに10年かかり予算が10倍になってしまえば、そのプロジェクトは失敗です。ベストプラクティスやバッドプラクティスを学ぶことは重要ですが、技術的最適解に固執してはいけません。まずはプロジェクトにおける最適解を追求し、その中で実現可能な技術的最適解を見つけることが大切です。

## 最適解の実現の困難さ

2つ目は、プロジェクトにおける最適解も実現が難しいことです。プロジェクトの「成功の判断基準はさまざま」と述べた通り、最適解だけが成功を意味するわけではありません。便宜上、プロジェクトの成功度を点数で表すと以下のようになります。

- 最高：プロジェクトにおける最適解（100点）
- 変動：プロジェクトにおける最適解に近いもの（61～99点）
- 最低：プロジェクトにおける及第点（60点）

「及第点」では、プロジェクトの契約に定められた範囲を満たす状態を指します。多くのプロジェクトは、60～99点の範囲内で成功として着地しているでしょう。低い点数ほど、技術的満足度も低くなる傾向があります。しかし、プロジェクト全体としては成功しており、技術的最適解を理由に点数を高める試みは、技術に関わらない人たちから反感を招く可能性があります（コードを綺麗にするための残業申請は認められるでしょうか？）。

私たちが目指すべきは「及第点」と「最適解に近いもの」の間です。ここには技術的最適解が必ずしも含まれるわけではありませんが、単に技術的最適解に固執すれば、私たちは「成功をもたらすメンバー」ではなく「面倒なメンバー」となるでしょう。

筆者は、バッドプラクティスの排除を優先するようにしています。バッドプラクティスは、ときおり壊滅的な失敗を招くことがあるためです。余力がある場合には、最適解に近いところを目指します。

何を当たり前のことをと感じる方もいるかもしれません、以下のことについて取り組み正確に答えられる人がどれほどいるでしょうか？

- 「技術的最適解」と「プロジェクトの最適解」の比較
- バッドプラクティスの排除とベストプラクティスの採用にかかるコスト算出
- 自身が「最適解に近いもの」に到達するために必要なコスト算出

これらの取り組みには多くの時間がかかるかもしれません。無駄だと言われることもあるかもしれません、私たちの失敗を減らし、面倒者扱いされることを避ける助けとなるでしょう。

†1 失敗したプロジェクトばかりでしたか？大変な道のりを歩んでこられたことに尊敬の念を禁じえません・・・。

†2 ITの技術的な視点から見た場合の最適解を表す筆者が作った造語です。

## 1.5 役割と上下関係

プロジェクトの中で自分の正しいと考える発言を行えているでしょうか？筆者は、プロジェクトで自身の役割を果たすために積極的に発言すべきだと考えています。そして、役割を果たすために発言する権利は、プロジェクト関係者全員に等しくあるとも考えています。

私たちの中には非常にシャイで無口な人もいるでしょう。このような人には難しいことをいいますが、発言することに慣れていくことが大切です<sup>†1</sup>。しかし、無口になる大きな要因は、プロジェクトの体制を階層構造として表すプロジェクト体制図（図1-5-1）にあると考えています。皆さんも階層構造になっているプロジェクト体制図をみたことがあると思います<sup>†2</sup>。

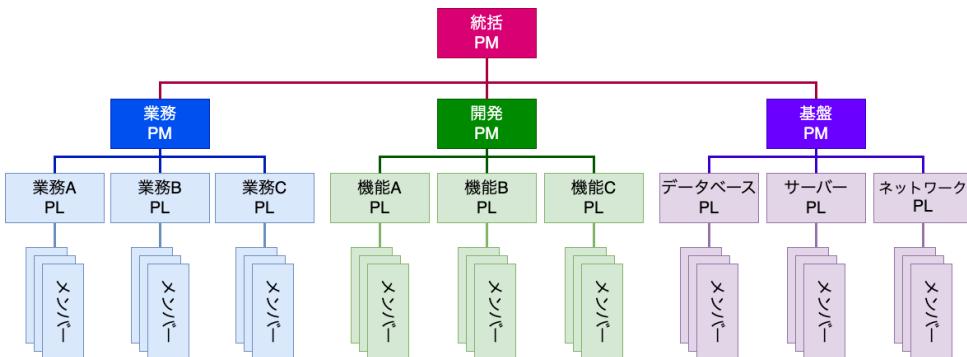


図1-5-1 プロジェクト体制図

詳細度は異なるかもしれません、多くのプロジェクトでは体制を明確に示すために、このような階層構造を持つプロジェクト体制図を作成します。メンバーは階層構造の中で「一番下の位置」に記載されることが多いです。このような体制図を見て「上は偉い、下は下っ端だ。だから上の人の指示に従って黙っておくべきだ。」という考えを持っているなら、その考えを今すぐ捨て去るべきです。

プロジェクトにおいて、私たちは役割によって区分されています。この区分は権限によるものではありません。プロジェクト体制図の目的も以下のようなものであり、権限を区

分することではありません。

- 役割の明確化（誰が何をするか）
- 指揮系統の明確化

「PMは権限が大きいじゃないか！」との意見もありますが、権限があるからPMなのでありません。PMだから権限があるのです。役割が広がることによって権限は増えますが、必要な権限が奪われることはありません。システムの権限管理を想像してみてください。システムを使うためには、最低限の参照権限がAdmin RoleにもUser Roleにも必要になるでしょう。プロジェクトの成功にはメンバーが欠かせず、メンバーにも発言権は必要なのです。

厄介な問題は、あなたの上司がPMやPLを務めている場合です。上司は人事組織図も上位に位置し、あなたを管理する立場にあることが明確です。しかしこの場合でも、プロジェクトにおける発言権は平等であることを忘れないでください。

誤解しないでいただきたいのは、プロジェクト体制図や人事組織図を批判しているわけではありません。これらは私たちの役割や指揮系統を明確にし、認識を共有するために必要なものです。また、好き勝手に自由な発言を奨励しているわけでもありません。指揮系統から逸脱した発言は、プロジェクトを混乱させるでしょう。自己中心的で我儘な発言をするのであれば、ご両親を呼んで注意してもらわないといけないかもしれません。

これら体制図の目的や用途に従いつつも階層構造にとらわれず、自分の役割に必要な発言を積極的にしなければ、プロジェクトは推進せず、私たちは指示通りに動くだけの無口な歯車になってしまうことでしょう。

†1 病気や障害などにより話すことが難しい場合は、この限りではありません。

†2 プロジェクト体制もしくはproject structureのキーワードでインターネットの画像検索を行うと、階層構造で表されるプロジェクト体制図を多く見つけることができます。

## 1.6 ウォーターフォールがもたらした最悪のもの

本節で伝えたいことは、ウォーターフォールモデル<sup>†1</sup>の欠点の追求や画期的な別の開発モデルに関する議論でもありません。ウォーターフォールモデルは現在でも広く使われており、大規模な開発にも適用可能な素晴らしい開発モデルの1つであると考えています（図1-6-1）。

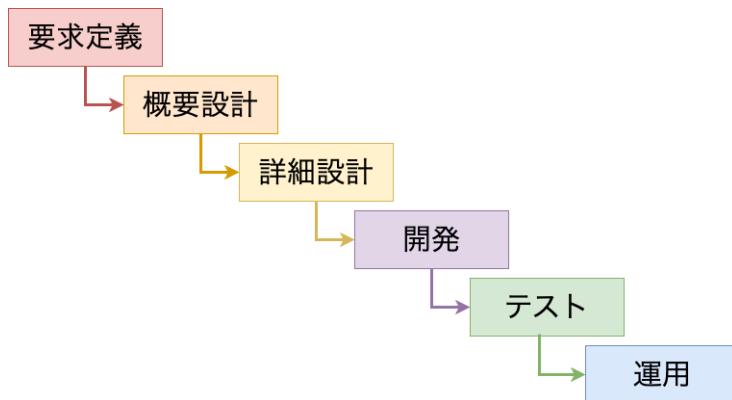


図1-6-1 ウォーターフォールモデルの工程の一例

ウォーターフォールモデルの欠点の1つとして「前工程に間違いがない」という前提が挙げられます。図1-6-1において「概要設計工程」では「要件定義工程」が間違いなく完了しているという前提です。もし間違いが見つかれば、手戻りが発生し間違いのある工程から多くの時間を使い修正が必要となる可能性があります。

筆者が考える「ウォーターフォールがもたらした最悪のもの」とは、この手戻りに関する潜在意識とそれに伴う誤った行動です。「開発工程」で「概要設計工程」に問題を見つけた場合、手戻りに伴う影響が予算・スケジュール・解散した工程メンバーの再招集や再編集などに及ぶことを、私たちは潜在的に意識します。資格学習や先輩からのアドバイス、自身の経験などによって、手戻りが非常に大変になる可能性があることを刷り込まれているためです。

こうした状況のとき、手戻りを避けるために誤った行動をする人たちがいます。彼らは

「他の誰かが問題を発見するまで問題を無視」したり、「強引な方法で解決しよう」とすることがあります。確かに、問題を無視すれば、対処にかかる時間を気にせずに済み、強引でも解決できれば手戻りを避けることが可能かもしれません。しかし、私たちが問題を見つけたのであれば、それを積極的にチームに共有すべきです。

ウォーターフォールモデルは「前工程に間違いがないことを前提」としていますが、私たちは間違いのないものを作ることが、どれだけ難しいか理解しているはずです。問題を見つけた私たちの最良の選択は、チームへの共有と解決策を検討することです。自身で考えた解決方法があったとしても、チームと共有し適切な解決方法なのかを議論する必要があります。これによって、手戻りが発生する可能性はあるかもしれません、さらに後の工程で問題が発生したり手戻りすることを防げるでしょう。

誤った行動をする理由には、日本のIT業界特有の問題も影響していると考えています。

## 請負構造

1つ目は、請負契約によって上流工程が元請けであり、下流工程が2次請け以降という組織による力関係のようなものが構成される可能性が高い点です。その場合、上流工程を担う組織（発注側）の発言力が強い場合があります。

## 上流と下流の技術力

2つ目は、上流工程を担う人材ほど偉い（発言力が強い）という認識を持つてしまうことです。上流工程は幅広い分野の知識が必要であったり、経験を求められることが多いことは否定できない部分です。

これらの問題点は、商流の関係によってそのようになることが実情としてあります。しかし、プロジェクトの観点では各工程の役割分担であり、上流下流による力関係は存在しないことをまず認識すべきです。「手戻り」や「力関係」を恐れて誤った行動をしてはいけません。

ウォーターフォールモデルや他の開発モデルも、それぞれのメリットを活用するために採用しているのであり、デメリットから目を逸らし口をつぐむべきではありません。

私たちが問題を見つけたとき、それは解決できる機会を得られたときなのです。

†1 上流工程から下流工程へ水が落ちる滝（waterfall）のように見えることからウォーターフォールモデルと呼ばれる開発手法の1つです。

## 1.7 正しいモノサシを持つ

---

「モノサシ」を持っていませんか？竹やプラスチック、アルミ製のオシャレなものやカラフルなものなどありますが、「物の長さを測る」ために使うモノサシです。

私たちが、ITエンジニアとして自己や他者の評価を行う際にも、無意識のうちにモノサシを使っています。モノサシは、さまざまな要素で構成されており、基本情報技術者のような国家資格・ITSS<sup>†1</sup>・学位・職務経歴・所属企業・一緒に働いた経験などが含まれます。これは個人ごとに異なり、1cm、2cmと正確に測れるものではありませんが、私たちはモノサシを持っています。

私たちはしばしば「誤ったモノサシ」を使い、誤った評価を持ち続けることがあります。どれほど「正しいモノサシ」持てるのかわかりませんが、可能な限り正しく自己評価を行うためのモノサシを持つよう意識しなければなりません。1cmのものを測って10cmとなるような誤ったモノサシを持つことは避けるべきです。

筆者が、誤ったモノサシで自分を測ってしまった経験を紹介しましょう。

### プロジェクトが終わったとき

筆者の担当した役割がマネジメントであれメンバーであれ、難しいと考えていたプロジェクトが（特に成功という形で）終わったときに、成長を実感しました。よく言われる達成感を味わった瞬間であり、自身が少しばかり成長したことは確かでした。しかし、プロジェクトの完遂には周りの人たちの助けを借りて乗り越えたものばかりでした。

成長や達成感を味わう前に、自身の担当した役割をその分野に精通した人や逆に自分よりも経験が浅い人などの他者が担当した時と比較し、自分自身の行動を振り返る機会を持つべきでした。振り返ることをしなかったために、精通した人たちが使っていた方法論や技術を学ぶ機会を失い、経験の浅い人よりも能力が高いと勘違いした日々を長く過ごしていました。

### 絶え間なく仕事が割り振られたとき

ある程度の経験を積むと、絶え間なくタスクが割り当てられます。このとき「私は高い能力があるから必要とされているのだろう」と勘違いをしていました。しかし、タスクを

割り当てる側の視点で考えると、タスクを遂行できない人に任せることはありません。完遂できるだろうという難易度の範囲内で割り当てます。

そのタスクが「社内初の試み」「経験者が誰もいないチャレンジ」「難易度が高いことが明らか」などであれば、高い能力をもっていると考えることもできますが、タスクの量が多いだけの場合「能力が高いから」振られているわけではありません。

## 本を読み終えたとき

筆者は、知識不足を補うために本を読みます。本以外にもインターネットやITイベントから情報を得ることも多いです。それらの情報を見たり聞いた後に、成長したと感じることがあります。

しかし、これらの情報は各分野の専門家が理解しやすくまとめたものがほとんどです。そして、情報や知識は活用して意味が出てくるものです。「本を読んだこと」自体は、評価を高める要素ではありません。評価を高めるのは「本を読み、それを活用し、実践できるようになること」です。

私たちは、自己評価や他者評価において明確な基本単位を持っていません。そのため、自分勝手な誤ったモノサシで評価してしまうことがあります。誤ったモノサシは、過小評価や過大評価など歪んだ評価を導き出す原因になります。

正しいモノサシを持つためには、常に相対評価が必要です。それは他者との比較か、過去の自分自身との比較かもしれません。

もし、成長したと感じる瞬間に出会った場合、それは私たちが持つ「誤ったモノサシ」を「正しいモノサシ」に修正する絶好の機会です。

†1 ITスキル標準とも呼ばれる、経済産業省が定めるIT関連能力を職種や専門分野ごとに明確化・体系化した7段階の指標です。  
IPA 独立行政法人 情報処理推進機構、ITスキル標準（ITSS）, 2023年9月11日  
<https://www.ipa.go.jp/jinzai/skill-standard/plus-it-ui/itss/index.html>

## 1.8 水はモノサシでは測れない

---

もし、バケツに入った水量を測って欲しいと言われたとき、モノサシで測れるでしょうか？筆者は水量を測るために、計量カップが欲しいと思うでしょう。対象物が異なれば、それに適した計測器が必要です<sup>†1</sup>。

正しいモノサシを持ち続けることは大切ですが、この正しいモノサシは技術という枠組みの中であっても、限られた範囲でのみ有効であることを理解しておかなければなりません。

限られた範囲とは、組織内・部署内・担当する技術やプロダクトの領域などです。モノサシは相対評価により修正できますが、対象が大きく異なるときモノサシを修正するのではなく「モノサシ以外の計測器」を使う方が、遙かに適しているときがあります。

- 組織・部署などが異なるとき
- 扱っている技術・プロダクトが異なるとき
- 役割やポジションなど立ち位置が異なるとき

このような状況のとき、単にモノサシを使うだけでは、意思疎通に齟齬が生じたり正確な判断が行えないことがあります。

### 組織が異なるとき

筆者は、とあるプロダクトを担当しており、そのプロダクトの設計や実装には複数のアプローチが存在していました。筆者は、自社の方針（定石と言われる設計や実装アプローチ）に従っており、それは定石と言われるだけあり疑念を抱くことすらありませんでした。

筆者が転職した後、同じプロダクトを再び担当する機会がありましたが、転職先の組織では異なるアプローチを採用していました。筆者の定石とはいつか異なる点がありました。筆者は間違いを訂正しようと試みましたが、詳細に調べてみると転職先で扱っている周辺コンポーネントなどを考慮すると、転職先のアプローチの方が利点が多いことが明らかになりました。

これは所属していた組織内で通用するモノサシを別の組織に使おうとした際の失敗例です。

## 技術・プロダクトが異なるとき

メンバーで使えないITエンジニアのレッテルを貼られた人物がいました。彼は、筆者たちのチームが担当するプロダクトの経験が浅く、無口なため、他のメンバーよりもパフォーマンスが劣っているようにみえました。しかし、彼はそのシステムに関連する別のプロダクトの経験が豊富でした。システムテスト中にトラブルが発生した際、筆者たちが手をこまねいている間に、彼の知識によって解決できました。

プロダクト内のみで有用なモノサシを使用した失敗例です。

## 役割が異なるとき

既存システムにいくつかの新機能を追加した新システムを構築する案件に携わったときのことです。既存システムを十分に理解できていなかつたため、設計を始める前にインフラストラクチャの状況、プログラミング言語の変更が及ぼす影響、新機能追加のための既存機能の変更点などの資料を作り説明しました。顧客担当者からの質問はなく、ただ頷いていたことを覚えています。

しかし、彼は情報システム部門のメンバーではなく業務部門のメンバーでした。そのシステムは、情報システム部が関与せず、業務部門が片手間に運用していました。また、筆者自身もITに詳しくない顧客とのコミュニケーションは初めてであり、ITエンジニアが「当然のように使う専門用語」を駆使して説明していました。

部署や相手の立ち位置を無視して、モノサシを使用した失敗例です。

これらのケースは比較的明確なものでしたが、もっと微妙な違いに気をつけなければならぬケースもあります。例えば、筆者は端末・client・host・node・edgeといった用語の解釈に悩んだことがあります。これらは「末端の利用者が使用するコンピュータ端末」のような類似の意味を表す場合もありますが、技術分野や業界、資料の文脈が異なれば、host・node・edgeは、サーバー機器などの大型のコンピュータを表したり、IoT機器のような小型のコンピュータを表すこともあります。

これらの違いに対処できる画期的なアイデアはまだ存在しませんが、異なる要素があるとき「モノサシ」を使うのか「別の計測器」を使うのかを意識することで、大きな失敗を回避することができます。

†1 「バケツの横・縦・水位をモノサシで測れば算出することができる」などの野暮な突っ込みは控えていただけると幸いです。

## 1.9 頑張ったから良い評価ではない

---

皆さんは仕事を頑張っているでしょうか？おそらく、多くの人が頑張っていると思います。頑張るには「困難にめげずやり抜く」「自己の意志を貫く」などの意味があります。私たちが頑張ることには、以下のようなものがあります。

- 技術を学び、仕事に活用する
- 厳しい要望に対応する
- 障害に対応する
- 長時間労働をこなす

これらのような困難に立ち向かった経験が少なからずあるでしょう。また、自己の意志を貫くことは、新しい技術を学び実務経験を積むために必要なことです。少々悲観的かもしれませんが、筆者は頑張ることには何かしらの「つらい」と感じることが付随していると考えています。

それでは、仕事で「成果」を上げているでしょうか？仕事の成果とは、何かを達成し、それによって得られる「良い結果」です。頑張っても「悪い結果」であれば、それは成果ではないのです。

頑張ったことを理由に、良い評価を要求する人たちがいます。彼らは成果が出ていないにも関わらず、頑張ったことだけを持ち出し「苦労を評価してくれ！」と主張します。

成果が出ていなくとも、頑張ること自体を人事評価に結びつける組織は存在します。筆者は、子どもが勉強や運動を頑張れば、成果が出ていなくとも良い評価にするでしょう。

しかし、組織の視点では、従業員の不満を解消するための施策であり、筆者の子どもを見る視点では、愛情からくるものです。いずれも、頑張ったことを良い方面に評価するかどうかは、他者視点に依存します。頑張ったことを自己評価することは、自尊心の向上には有益かもしれません、それを他者に「良い評価にしろ」と押し付けるのは間違いです。

人事評価において「成果の出ない頑張り」を評価項目として持たない組織に対して「頑張ったから良い評価をしろ」と主張し続けるのは時間の無駄です。筆者の子どもが勉強を

頑張ったからといって、学校に「100点の採点をしろ！」とは主張しません。「頑張り」と「評価」は異なる視点で見つめるべきです。

もし、他者からの評価が自身の考える評価と違うと感じた場合、3つのことを考えてください。

### 時間による評価

1つ目は、残業時間を評価に結びつけたいと思うのであれば、すでに残業代という対価を受け取っていることを認識し、納得することです。残業代は、成果が出ていない仕事にも、お金が支払われる素晴らしい制度です。みなし残業の場合でも、それを了承して組織に属しているはずなので納得しましょう。もし規定の残業代が支払われていないのであれば、評価に結びつける前に労働基準監督署へ報告するべきです。

### 成果による評価

2つ目は、成績があるのに適切な評価を得られない場合、組織と交渉することです。ただし、自身の感情や主観的な評価を主張するのではなく、同じ立場にある同僚との比較や、同じ立場の他の人たちの成績と比較して主張しましょう。もし、それでも納得できなければ、転職を検討するべきです。組織自体の考えを変えるよりも、居場所を変える方がよほど簡単です。

### 頑張らない選択

3つ目は、頑張らない選択肢も考えることです。「頑張っても成果が出せないが、良い評価をして欲しい」というのは、非現実的な要求です。頑張ることが主張の土台であるならば、頑張ることをやめて良い評価を得られないことも納得がいくでしょう。もちろん、頑張らないことによる将来的なリスクも考慮すべきですが、頑張らなくとも今の報酬から生活できないほどの大きな減額はされないはずです。

年功序列が崩壊中と言われ、成果主義が台頭している現代社会において、特にIT業界はその変化を強く感じる業界の1つになるでしょう。このような状況下で、成果の出せない中で行えることは非常に限られています。

## 1.10 不満を垂れる前に行動を

私たちは多くの不満を抱えながら仕事をしています。一時的に不満を垂れるのは普通であり「ストレスが発散される」「同僚と気持ちを共有する機会となる」など良い面もあります。

しかし、中には同じ不満を延々と繰り返す人たちがいます。このような言動は、自身の幸福感を低下させるだけでなく、一緒に働くメンバーや同僚にも悪影響を及ぼします。心の中だけで繰り返し不満を垂れている場合でも、不満を軽減する方法を考え、軽減に向けて行動しなければなりません。

ここで強調すべきは「不満の完全な解消」ではなく「不満の軽減」です。すべてに不満のない環境、仕事内容を求めるることは現実的ではないため、不満を垂れないレベルまで軽減することを目指しましょう。

不満を軽減する行動をする前に、不満の原因を分類し、軽減するための労力を把握する必要があります。私たちがよく抱えると思われる不満を分類してみました（表1-10-1）。

個人の問題による不満	組織の問題による不満	あいだの問題による不満
やりたい技術ではない	人材不足	給与が見合っていない
最先端の技術を使えない	業務量が多い	残業が多い
同じ業務が続く	研修制度がない	改善案が採用されない
やりがいを感じられない	福利厚生への不満	指示が曖昧
スキルアップができない	職場環境が悪い	出世できない
勉強が大変	セクハラ・パワハラ	人間関係が悪い
リモートワークがない	未払い残業代がある	将来への不安

表1-10-1 不満の分類

## **個人の問題による不満**

私たちの願望や価値観と現実のギャップから発生している不満です。組織は、従業員の満足度を向上させるために問題に取り組む可能性はありますが、積極的な対策を講じることは稀です。多くの組織は営利企業であり、私たちの願望を叶えるために存在しているわけではありません。好意的に捉えれば、私たちの行動（例えば転職など）により、不満の軽減が容易なものです。

## **組織の問題による不満**

組織文化や制度によるものから発生している不満です。組織維持に問題となるものが多く、組織が積極的に取り組む可能性があります。しかし、私たちの行動だけで不満を軽減することは難しいものになります。例えば「研修制度がない」は、私たちが筆頭となり社内勉強会のような文化を形成し、研修制度へ成長させることも可能ですが、多くの時間と労力を必要とします。組織が積極的に介入しない場合、個人の行動で不満を軽減できる可能性は低いです。

## **あいだの問題による不満**

私たちと組織の認識の齟齬により発生している可能性がある不満です。例えば「給与が見合っていない」という不満は、組織的な給与体系の問題の可能性もあれば、私たちの自己評価が高すぎる可能性もあります。私たちが組織の望む行動を意識することで不満が軽減できる可能性があります。認識に齟齬もなく、組織の方針に賛同できない場合は、個人の問題として行動する必要があります。

不満の種類によって軽減するための労力が異なるため、不満の度合いと軽減に必要となる労力を比較しながら行動する必要があります。また、いずれにおいても不満の軽減に一番簡単な方法は、転職・部署異動などの新しい環境に身を置くことですが、不満をさらに大きくしたり、新しい不満が生じる可能性もあります。

軽減に向けた行動をしない場合は、その環境に納得するしかありません。不満を垂れることを続けても、その不満が無くなることも勝手に軽減されることもありません。

不満を垂れ続ける前に、行動しましょう。



第

# II

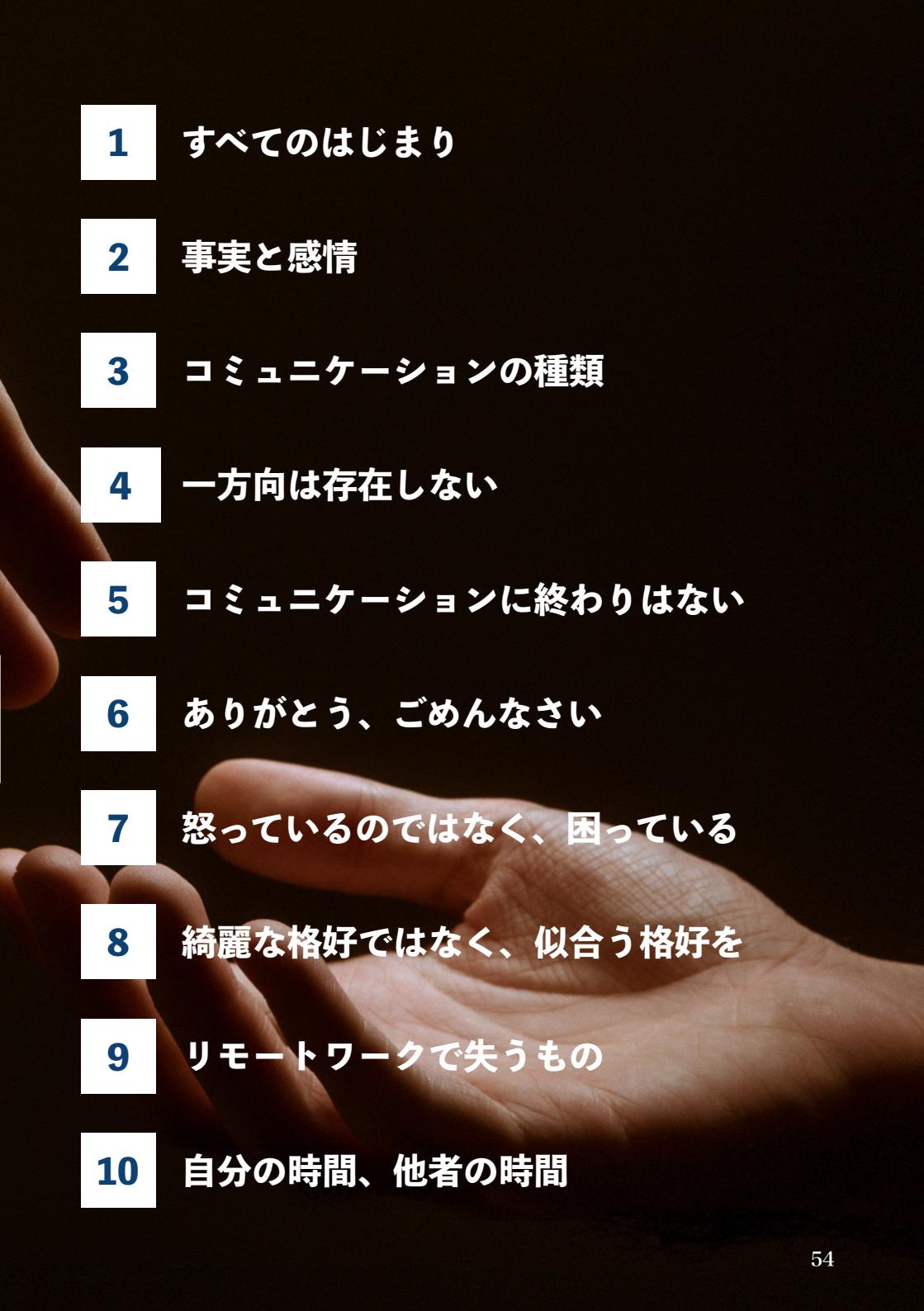
章

## 無能のためのコミュニケーション

本章は、無能なITエンジニアとして生きていくための  
コミュニケーションについての教訓になります。

コミュニケーションと聞くと持つて生まれた才能や  
高度なテクニックが必要と思うかもしれませんが、  
これらの才能や高度なテクニックは必要ありません。

私たちに必要なコミュニケーションは、  
仕事上で必要となる意志伝達と信頼関係だけです。



**1** すべてのはじまり

**2** 事実と感情

**3** コミュニケーションの種類

**4** 一方向は存在しない

**5** コミュニケーションに終わりはない

**6** ありがとう、ごめんなさい

**7** 怒っているのではなく、困っている

**8** 綺麗な格好ではなく、似合う格好を

**9** リモートワークで失うもの

**10** 自分の時間、他者の時間

## 2.1 すべてのはじまり

---

私たちは、プロジェクトチーム・顧客・同僚など、さまざまな人たちとコミュニケーション<sup>†1</sup>を通じて仕事をしています。コミュニケーションは、他者と理解し合うために必要な手段です。そして、私たちが他者と関わる「はじまりの手段」でもあります。

コミュニケーションの方法は、会話・電話・メール・チャットなど多岐に渡りますが、共通していることは「はじまり」と「継続」があることです。

コミュニケーション不足が生じる主な原因是、コミュニケーションを「はじめる」「継続する」ことを軽視したり、その大切さを忘れてしまうことだと考えています。コミュニケーション不足が生じると、以下のような発言を聞いたり、私たち自身が発言する機会が増えます。

- 聞いていなかった
- 知らなかった
- 言われた通りにやった

これらは、大きな問題に発展する可能性があるものです。個人の好奇心や積極性に依存する部分もありますが、コミュニケーションを「はじめる」「継続する」ことによって予防または改善できるものです。

しかし、私たちが「はじめる」「継続する」ことの重要性を認識していても、しばしばこれらを怠ることがあります。以下は、筆者がコミュニケーションを怠ってしまった場面です。

### 怒りっぽさ

仕事の関係者の中には、怒りっぽい人が存在することがあります。彼らに質問などをすると怪訝な顔をし怒鳴りつけるように（時には本当に怒鳴りながら）返答をします。もしくは、険悪な雰囲気を出し無視するような態度をとる人もいます。このような時は「はじめる」「継続する」の両方が極端に難しくなります。

## 多忙さ

私たち自身もしくは相手が多忙で、コミュニケーションに時間をかけることを短くしたり、コミュニケーション自体を避けることがあります。多くの場合、認識に齟齬が生じたり、情報の漏れがでてきます。

## 羞恥心

知識不足や無知な部分が露呈することを恥ずかしいと感じ、コミュニケーションを避けたことがあります。特に私たちは技術や専門用語への理解が乏しいときによく発生します。必要な情報への理解を遅らせ、知識レベルの把握に誤解を生じさせます。また、分野や業界で「暗黙の了解」とされる情報が欠落したままになることがあります。

## 傲慢さ

経験年数や役職などのポジションによって、コミュニケーションを行う双方の力関係や発言力に大きな差があるときに一方的な情報伝達になることがあります。このような場合、コミュニケーションが実質的に行われていないこともあります。

## ツールの経験不足

ツールの使い方やそれらの運用ルールの知識不足などから億劫になりコミュニケーションを取らないことがあります。ツールは、チャットツールのようなものだけでなく、話すことが苦手などの会話というツールも含まれます。

これらは年齢や経験に関係なく、誰にでも起こりうることです。しかし、私たちは、これらを理由にコミュニケーションを怠ってはいけません。プライベートであれば無理に人間関係を維持する必要はありませんが、仕事で関わる人たちは、仕事を続ける上で否応なく関係が続くものです。

コミュニケーションに積極的ではない相手もいるでしょう。しかし、私たちがコミュニケーションを「はじめる」「継続する」ことを続けている間は、コミュニケーションが途切れることはなく、多くの問題を予防するための助けとなるはずです。

†1 「複数（2名以上）の人たちが互いに意思疎通や情報伝達を行う」という意味で用いています。

## 2.2 事実と感情

筆者は、コミュニケーションの中で伝える情報は「事実」と「感情」により構成されていると考えています。話し手が「事実」と「感情」を伝え、聞き手の「フィルタ」を通した「結果」から新たな質問や対話が生まれ、このサイクルがコミュニケーションの基盤となっています（図2-2-1）。

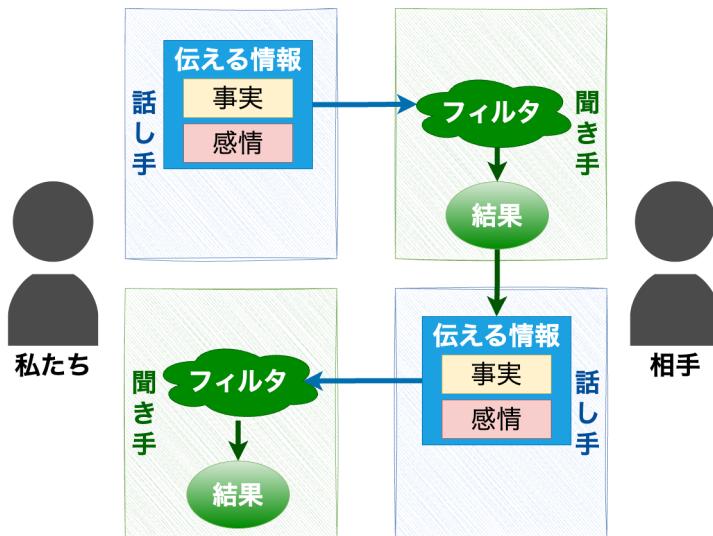


図2-2-1 コミュニケーションの「事実」と「感情」

### 事実の情報

確定的に明らかな情報です。現実の出来事や仕様に基づいた情報などです。話し手・聞き手ともに得る情報は同じですが、事実の整理方法（綺麗な文面か汚い文面かなど）によって、聞き手の感情に作用する場合があります。

### 感情の情報

喜び・悲しみ・怒り・諦め・驚き・嫌悪などの情報です。話し手の表情やジェスチャー、声の抑揚で表現し、文字情報が主体の場合には、「！」や「？」などの記号や絵文字、色<sup>†1</sup>などでも表現することができます。

## フィルタと結果

聞き手は、受け取った情報（事実と感情）を「聞き手のフィルタ」を通して「結果」を導き出します。フィルタは、経験・知識・性格などさまざまな要素で構成され、多くの場合は無意識的に作用します。結果は、新たな疑問や質問・最終的な判断など、フィルタを通して生まれたもの全てを指します。必要に応じて、結果を元にコミュニケーションを継続します。

感動的な映画を例に挙げてみましょう。「話し手は映画」「聞き手は視聴者」です。「事実」は、映像と役者のセリフです。「感情」は、BGMや役者の演技（身振り、セリフの抑揚など）です。同じ情報を受け取っても「聞き手のフィルタ」によって「結果」は異なります。筆者は涙もろいため、ボロボロ泣いていることでしょう。現実主義者は、フィクションを受け入れて涙を流していないかもしれません。辛口の映画評論家は、脚本にダメ出しをしているかもしれません。

聞き手のフィルタを把握することは困難ですが、事実と感情を正確に伝えようとすることは、好感を得ながらコミュニケーションを継続するために必要なことです。筆者が聞き手の場合、好感を得たことには以下のようことがあります。

- 商品を勧められる時、素晴らしい（事実）を熱心（感情）に伝えてくる営業が好き
- 笑顔（感情）で質問（事実）してくる後輩が好き<sup>†1</sup>
- 技術を教えてもらう時、参考URL（事実）と補足情報（感情）があると嬉しい

私たちがコミュニケーションを行うとき、次のことに気をつけなければなりません。

## 私たちが話し手の場合

私たちが話し手の場合、伝えたい情報を「事実」と「感情」で表現できているか。また「聞き手のフィルタ」を通した結果に可能な限り誤解を与えない考慮ができているか。

## 私たちが聞き手の場合

私たちが聞き手の場合、話し手の「事実」と「感情」を受け取れているか。また「自身のフィルタ」を通して誤った結果を出していないか。

<sup>†1</sup> 色は感情に作用すると言われています。私たちが資料などで重要な部分を「赤文字」にしたりするのは「重要なんだよ！」という感情表現と捉えることができるでしょう。

<sup>†2</sup> 本当に困っている時は、困った顔をして欲しいです。システムが停止している中、ヘラヘラと笑う人は好きではありません。

## 2.3 コミュニケーションの種類

1番得意なコミュニケーション方法は何でしょうか？コミュニケーション方法には、さまざまな種類があり、それぞれに強みを持っています。アナウンサーのように流暢に話すことが得意でなくとも、種類を意識することにより改善の余地は十分にあります。筆者が意識しているコミュニケーションの種類を「言語コミュニケーション（表2-3-1）」と「非言語コミュニケーション（表2-3-2）」にわけて紹介します。

### 言語コミュニケーション

会話（対面）	言葉を使い会話をを行う、人類が手にいれた最強ツールの1つ。手軽であるが、言葉だけの場合ミスリーディングが発生しやすい。
オンライン会議	距離を無視し、1対1や多人数と意思疎通が可能。会話（対面）よりもコスト <sup>†1</sup> が大きくなる場合もある。
電話	距離を無視した1対1の意思疎通が可能。声という限定された情報のためミスリーディングが発生しやすい。
メール チャット	文字や図など複数の情報を多人数へ一斉に伝達できる。ミスリーディングが発生した場合の修正コストが大きくなる場合がある。
コード <sup>†2</sup>	特定の人 <sup>†3</sup> への意志伝達が素早く行える。知識レベルに左右され、必要な情報が伝えきれない場合が多い。

表2-3-1 言語コミュニケーション

### 非言語コミュニケーション

見た目	信頼感と納得感を高める。自身を鼓舞するなどの効果もある。
表情	感情を手軽に伝えることができ、共感を高める効果もある。
パラ言語	相槌、声のトーン、速度、身振りなどを意味する。絵文字や記号（！、？など）、色も含まれる。
図形・画像	視覚的情報を使い意志伝達を行う。設計書の図など視覚的なもの全般を指す。準備（図の作成など）に時間を要するなどコストは高いが理解度を高める効果がある。

表2-3-2 非言語コミュニケーション

他の種類が思い浮かんだ人もいるでしょう。特にITの技術に囲まれている私たちは、コミュニケーションに使用できる数多くのツールやサービスを知っています。私たちが、これら全てを完璧に使いこなすのは難しいかもしれません、3つのことを覚えておく必要があります。

## コミュニケーションは複数で1つ

1つ目は、コミュニケーションは複数の種類が絡み合い成り立っているということです。私たちは会話をする際にも、表情を変えたりパラ言語を無意識的に使っています。ポーカーフェースで伝え続けても、どれほど重要なのか相手に伝えることは難しいでしょう。これらの要素が、伝えたい内容に適しているかどうかを意識しなければなりません。

## 自身の得意なこと・苦手なこと

2つ目は、自身の得意・苦手なコミュニケーションの種類を把握し、それぞれのメリット・デメリットを理解することです。筆者は会話が苦手なため、図形を多用する傾向があります。図形は、事細かに会話をせずとも、相手の理解を高めてくれます。これは苦手な部分を補ってくれますが、準備（図の作成）に多くの時間を必要とします。

## 得意なことに固執しない

3つ目は、特定のコミュニケーションの種類に固執しないことです。自身の得意な方法だけにこだわると、相手をイライラさせたり、理解を妨げることがあります。筆者は図形を好みますが、メールの返信は文章を優先します。直接話しかけられた場合は、会話を優先します。彼らは、早い返信を望んでいることが多いからです。筆者がエラーの解消方法を聞きたい場合は、エラーコードとログを送ります。解消方法を知っている人たちは、筆者が作る程度の図は必要ないことが多いからです。

種類を把握せずコミュニケーションを行うと、私たちは不要なコストを支払うだけでなく、多くの問題を抱え込むことになります。

†1 お互いが使用するツールや、対面と比較して声が通りにくいなど、コミュニケーションに必要となる環境や費用、時間などを指しています。

†2 コードとしていますが、「実装そのもの」を指しています。プログラミングの場合はコードですが、ネットワークの場合は機器のconfigファイルになるでしょう。

†3 「実装」の内容を理解できる人を指しています。

## 2.4 一方向は存在しない

---

コミュニケーションという場合、多くは双方向コミュニケーションの意味として使うでしょう。双方向コミュニケーションは、聞き手から何かしらのリアクションがあることを前提としており、一方向コミュニケーションは、話し手からの一方的な情報伝達もしくは情報共有と解釈されます。多くの場合、私たちはITエンジニアという専門家の立場として、専門外<sup>†1</sup>の人たちと双方向のコミュニケーションを通じて仕事をします。

一方向コミュニケーションを行う際には、問題となる2つのことがあります。

1つ目は、聞き手の質問や疑問を置き去りにしてしまうことです。

双方向コミュニケーションでは、聞き手からの質問や疑問のリアクションの機会がありますが、一方向コミュニケーションではその機会が欠落しています。一方向コミュニケーション（一方的な連絡など）の後に、設計レビューや会議などの双方向コミュニケーションの機会があれば、専門としない部分（専門外の技術連携や運用に関するもの）の誤りを訂正することができますが、その機会がない場合は、問題の要因を残したままプロジェクトが進行していきます。

2つ目は、私たち自身への物事を疑う心理を弱めてしまうことです。

一方向コミュニケーションでは、自身の専門分野の知識や行動に対しての批判的思考や懸念が薄れる傾向があります。また、専門外の人たちは、専門家からの情報を「そういうものか」と受け入れる傾向があり、疑問視する機会が減少します。同じ分野の専門家からの指摘がなければ、誤った認識を持ったままプロジェクトが進行したり、誤りを訂正するためのコミュニケーションコストを増大させる可能性があります。

このような問題を回避するためには、「一方向コミュニケーションとなる場合「自身と対話する双方向コミュニケーション」を忘れてはいけません。

### 素晴らしい発見をしたとき

素晴らしい方法論や技術を見つけたとき、「既存技術との比較」や「素晴らしい技術のデメリット」を無視することができます。素晴らしい以外の部分に疑問を持たず、方法論

や技術が採用された場合、負債となる可能性があることを忘れてはいけません。

## 設計をするとき

テンプレートの設計書に沿って穴埋め問題のように作る設計者は賛否両論ありますが、時間的制約があるなど致し方ない場合もあります。しかし、その設計内容は私たちが専門家として担当している部分であり、なぜそのように設計したかの理由があるはずです。その理由が説明できなければ、問題に発展しそうな内容の判断ができないだけでなく、聞き手からの質問があった場合にも答えることもできないでしょう。

## 実装をするとき

設計が正しい場合も、その実装方法が複数存在する場合があります。実装方法について理由を説明できなければ、設計の時と同様に問題に発展しそうな内容の判断ができないだけでなく、聞き手からの質問があった場合にも答えることもできません。

## 情報発信をするとき

技術ナレッジとして、情報をまとめて発信することがあります。聞き手が専門家の場合は、前提条件・バージョン・明確な方法論や技術詳細・参考情報・確認した日付などの情報を必要とするはずです。顧客などの専門外の人たちに発信する場合でも、発信の意図・背景などがなければ、聞き手は混乱するでしょう。

私たちは、常に「もう一人の自分自身と双方向コミュニケーション」を行わなければなりません。もう一人の自分は「これは大丈夫なのか?」「足りない情報はないか?」「この人の立場ならどうか?」と次々に質問をしてきます。これらに答え続けることは、私たちの問題を防ぐだけでなく、聞き手が実在する場合の双方向コミュニケーションコストを大きく下げてくれます。

私たちが仕事をするとき、一方向コミュニケーションは存在しません。常に双方向コミュニケーションをしています。

†1 専門外は、IT知識の疎い顧客だけとは限りません。ITエンジニアの「プログラマ」でも、活躍する業界やプログラミング言語の違いで異なる専門分野を持ちます。

## 2.5 コミュニケーションに終わりはない

---

コミュニケーションは、さまざまな規模や重要度で行われます。小さなコミュニケーションであれば上司や先輩への報告のような日常的なものから、大きなコミュニケーションであればプロジェクト全体に関わる多くの人を対象とするものまで、私たちは常にコミュニケーションが必要になります。

このようなコミュニケーションを維持するために、非常に多くの労力を使うことでしょう。情報の収集や事実確認に要する時間、それらに関連する作業、精神的プレッシャーなど、さまざまな労力が必要です。そして、コミュニケーションが終了した瞬間、私たちはしばしば安堵します。しかし、実際はコミュニケーションに完全な終わりはありません。終わりに代わる表現があるとすれば「一時的な区切り」が正しいでしょう。

筆者の考える一時的な区切りの例をいくつか挙げてみました。

### 設計・実装の区切り

設計や実装工程もしくは工程単位でなかったとしても担当部分のタスクを完了をした場合です。しかし、設計や実装に不備はつきものです。新たな修正や対応タスクが生まれるかもしれません。

### プロジェクトの区切り

プロジェクト全体が終わった（検収が上がったなど）場合です。しかし、関連した新規プロジェクトへの引き合いや該当システムのリプレースなどの案件につながる可能性もあります。すっかり忘れてしまった数年後に、当時の担当者だったという理由から再度担当になることも珍しいことではないでしょう。

### 不具合・改善などの一時的な対応の区切り

問い合わせや技術支援のために一時的に対応したタスクが完了した場合です。依頼時には一時的なものであったとしても、関連タスクや類似の問題があった場合などに新たな依頼がくることがあるかもしれません。

## アドバイスや雑談の区切り

上司・同僚・組織外など、あらゆる人たちとの雑談が終わった場合です。仕事や技術的な内容を雑談の中ですることは往々としてあるものです。そして、雑談から生まれる仕事もあります。その話を理解していると認知され、プロジェクトへ参画やタスクのアサインが決まる可能性も十分に考えられます。

## 人間関係の区切り

転職や部署異動などで、その時の人間関係が解消される場合です。しかし、部署異動であれば、異動後も組織内で連携することは多いでしょう。組織外であったとしても、別プロジェクトでばったり出会うことがないとは言い切れません（IT業界は意外と狭いですから・・・）。

例に挙げたものは全て「コミュニケーションが再継続する」もしくは「関連した新たなコミュニケーションが発生する」可能性のあるものです。コミュニケーションは「継続」と「発展」の機会があり、終わるという考え方をすべきではありません。

これは、終わりという意識が「自分の範囲外になった」という誤解を生まないための予防策です。自分の範囲外と認識したとき、私たちは注意力が欠如し、問題の芽をばら撒くことがあります。

- 設計・実装では運用・保守性・将来の移行性への考慮を軽視する
- プロジェクトでは、顧客やメンバーとの関係性を無視した言動をとる
- 一時的な対応では、継続する人に必要な情報や履歴を適切に記録しない
- アドバイスや雑談では、信憑性やリスクを考慮せず発言する
- 人間関係では、人間関係の形成を疎かにしたり、悪化させる

これらは、周囲の人たちに問題を引き起こすだけではなく、最終的には自分自身に問題をもたらす可能性があります。

もちろん、本来の範囲を逸脱したタスクに至るまで対応したり口を出すべきではありません。

本来の自分の範囲を見極め、将来の私たちへ関連することを忘れないために「終わり」ではなく「一時的な区切り」がついたことに安堵しましょう。それは、私たちの未来を見据えたコミュニケーションに繋がります。

## 2.6 ありがとう、ごめんなさい

---

「ありがとう」と「ごめんなさい」は、魔法の言葉と言われます。この2つの言葉には特別な力があり、コミュニケーションにおいて重要な役割を果たします。

- 「ありがとう」は、感謝を伝え、相手も幸せにする魔法の言葉
- 「ごめんなさい」は、謝罪を伝え、その後の人間関係も円滑にする魔法の言葉

多くの人は幼いころに、この魔法の言葉を私たちを大切に思ってくれる人たちから教えられ、日常的に使っていました。しかし、時間が経つにつれて、これらの言葉を忘れることがあります。

筆者は、魔法の言葉を忘れてしまっていた時期があります。ITエンジニアになり数年ほど経過して、特に自分が主導する仕事が増えてからは、たびたび忘れてしまいました。

### 「ありがとう」を忘れる理由

経験年数が増えてくると、技術的な知識や理解力が不足していることを恥ずかしいと思ってしまうことがあります。せっかく色々と教えてもらっても、それらを理解しきれないことを恥ずかしく思ったり、理解を深めるための質問も「その程度も知らないの？」と思われたくないため避けてしまい「ありがとう」も伝えられませんでした。

また、激務で多忙な状況で働いていると、自分は一生懸命に頑張っていると自己を過大に評価し、他の人たちが頑張るのも当たり前と考えたり、足早に定時で帰宅する人たちに、感謝の気持ちがなくなってしまうこともあります。

### 「ごめんなさい」を忘れる理由

ITエンジニアをしていると不用意に謝罪ができないことがあります。本来は責務範囲外のことだった場合でも、謝罪をしたために無償対応を余儀なくされることもあり得るでしょう。このような経験は、皆さんも少なからずあるのではないかでしょうか？

これらの経験から、「謝罪のための資料作成や関連部署への連携などに時間をかけることが面倒だ」「謝っても別のやることの時間が増えるだけだ」と感じ、会話の言い回しなどで謝罪を回避する方法を模索するようになり「ごめんなさい」を忘れることがあります。

「ありがとう」と「ごめんなさい」を忘れる理由は他にもあります。

- 専門性が高まり悪い方向にプライドが固まってしまう
- 仕事をやらされていると感じている
- 技術力の差などにより相手を軽視する
- 無知を馬鹿にしたり、軽率に扱う人が周りにいる

これらの理由は、自身の置かれている環境や年齢の重ね方などによって変わってきます。しかし、魔法の言葉を忘れ続いていると、再び使えるようにするために多くの勇気と時間を要します。まるで禁忌の魔法のように扱われてしまうのです。

最近「ありがとう」と「ごめんなさい」を唱えていない人がいれば、ぜひ明日から再び唱えてみてください。唱える機会があるか心配ですか？安心してください。「ありがとう」を唱える機会は私たちの周りに溢れています。「ごめんなさい」は機会が訪れた時に唱えられるようにしておきましょう。

そして、これから実践する人のために、魔法の言葉の効果をより一層高める方法を伝えておきます。

1. 魔法の言葉を唱える時は、相手に「身体」を向けて唱えてください
2. 身体を向けるのに慣れたら、「目」を見ながら唱えてください
3. 目を見ることに慣れたら、「心」を開いて唱えてください

「ありがとう」と「ごめんなさい」は合わせて1つの魔法です。どちらか1つではいけません。もし、どちらかだけを使い続ければ、私たちは、また魔法を使えなくなってしまうでしょう。

## 2.7 怒っているのではなく、困っている

私たちは、しばしば他者から怒られることがあります。怒る人は上司・先輩・営業・顧客であったりとさまざまですが、彼らは不思議なほど私たちを怒ることがあります。

昨今では、パラーハラスマント<sup>†1</sup>対策やアンガーマネジメント<sup>†2</sup>の意識が高まり、怒鳴られることは減少しているように感じますが、それでも怒りの目で見てきたり、落胆した雰囲気を感じることはあるでしょう。彼らが怒りそうな事例を以下に挙げてみました。

- 約束の期日を守れなかった
- やることが間違っていた
- 返信が遅い
- システムを止めてしまった

ここで重要なのは、彼らはストレス発散のために怒っているのではなく「困っているから怒っている」ということです。困ったことがないのであれば、怒る必要などありません。

彼らの立場になって考えてみましょう。彼らの立場で困っていることが確かに存在しているはずです。もし、私たちが彼らの立場だったとしても、怒ったり落胆するほど「困る」ことばかりでしょう（表2-7-1）。

怒った要因	困ること
私たちが期日を守れなかった	関係者の再調整が必要かもしれない
私たちのやることが間違っていた	成果が0である (後続の人に影響があるかもしれない)
私たちの返信が遅い	判断が行えなえず待つしかない
私たちがシステムを止めてしまった	顧客は確実にサービスが利用できていない (上司は対応に追われることは確実である)

表2-7-1 怒った要因と困ること

ここから私たちが学べることが2つあります。

## 怒られない対処ができる

1つ目は、困っているのであれば、対処できることです。怒っている理由が私たちをサンドバック代わりに言葉で殴るストレス発散であれば、パワーハラスメントの相談窓口に通報するしかありません。しかし「困る」という視点で考えれば、対処の仕方が見えてきます。

表2-7-1の事例を考えてみると、早急な報連相<sup>†3</sup>や期限の細かな確認をしなければいけないと想像できます。「システムを止めてしまった」などの場合であっても「困る」視点で考えれば、支援や挽回する機会に繋げることができます<sup>†4</sup>。

「怒られている」という自身の立場よりも「怒っている（困っている）」人の立場から考える方が、物事の判断がしやすいことがあります。

## 私たちも怒らなくていい

2つ目は、私たちが怒る（困る）立場になる可能性がある場合のコミュニケーション方法です。私たちが他者にタスクを依頼する際には、成果物のイメージ共有・期限・重要度・緊急性など、困る可能性のあることを明確にして伝える必要があります。

もし、依頼の段階で指定できない場合であっても、不透明な事項はその旨を説明し後日必要な情報を追加で提供するなどの対応策を模索すべきです。

これらを私たちが行えれば、怒る必要はなくなるでしょう。

私たちが怒られる場面や怒ってしまう場面において、こう言っているのです。

「どうなっているんだ！？（困っているから助けて欲しいんだ！！）」

†1 職場内の優位な立場を利用して、叱責や嫌がらせを行う行為のことです。

†2 怒りを予防し制御するための心理療法の1つです。

†3 報連相（ほうれんそう）。報告・連絡・相談の3つの言葉を合わせたビジネス用語です。

†4 システムを停止させても何もせず呆然とするITエンジニアを見たことがあります。少なくとも、そのような行動はしないようになるでしょう。

## 2.8 綺麗な格好ではなく、似合う格好を

---

ITエンジニアにまず求められるものは技術力でしょうか？いいえ、初めに必要なのは信頼です。では、最初に信頼を築く方法とはなんでしょうか？それは「見た目」です。

自分自身でも驚くほど单刀直入なスタートになりましたが、見た目はそれほど重要ということです。ほとんどの人が初対面の人物に対して、無意識のうちにその人に対する印象を形成します。これは人と初めて出会った際、最初に得る情報が声や技術スキルなどよりも、視覚（見た目）が先だからです。私たちが見た目によって、出会った瞬間に信頼を築くことにはいくつかのメリットがあります。

- 一緒に仕事をしたい（してもいい）と思える安心感
- 発言する内容への信頼感の向上
- TPO (Time、Place、Occasion) の判断能力への評価

想像してみてみましょう。初対面のビジネスパートナーと1対1の会議が始まることになりました。この会議は、システム設計に関する事前打ち合わせで、ビジネスパートナーの詳細情報は提供されていません。失礼のないように名前だけは覚えましたが、相手の顔写真や技術スキル、性別などは分かりません。そして、予約した会議室で待っていると、相手が会議室に入ってきました。

Aさんは「**ぼっしゃりした体型** 服は穴の空いた古ぼけたダボダボの寝巻き 髪は腰まであり 足元は壊れかけのサンダル の 中年男性」です。

Bさんは「**ぼっしゃりした体型** 服はジャストサイズのTシャツにジーンズ 髪はショートのツーブロック 足元は黒のレザーワークブーツ の 中年男性」です。

早い段階で目につく特徴を順番に記述してみました。**青字**は同じ特徴で、**赤字**は異なる特徴です。どちらに信頼感を持つでしょうか？おそらく、Bさんの方に信頼感を持つ人が多いと思います。正確に言えば、Bさんも状況によっては、信頼感を与えるには足りないかもしれませんが、少なくとも不快感は与えないでしょう<sup>†1</sup>。同じ体型や年齢の人でも、服装や髪型によって不快感が生まれることを示しています。

これは極端な例ですが、初対面で事前情報のない場合、無意識に外見から印象を決めてしまうことがあるのです。もし、Aさんが旧知の仲であり、プライベートで自宅に遊びに行つた場合、不快感は得なかつたかもしれません。

このような外見からの判断はルッキズム<sup>†2</sup>と呼ばれる差別的な要素を含むことがあります。筆者も、外見だけで人を評価することは避けるべきだと考えています。仕事の評価においても、外見だけが決定要因ではありません。Aさんほどではないにしても、奇抜な格好で、人柄も技術力も高いITエンジニアもいました。

しかし、現実では「見た目」からの印象によって判断することが、少なからず根付いています。そこに意識を向けることで円滑なコミュニケーションを築けるのであれば、積極的に取り組むべきだとも考えています。

誤解を避けるために明確に述べておきますが、以下のようなことを推奨しているわけではありません。

- 整形手術を受ける
- 体重を減らす
- 綺麗にする

整形は個人の自由な選択ですし、標準体重を維持することは健康に良いことかもしれません、少なくとも数ヶ月の時間と強い意志が必要でしょう。綺麗さは個人に依存する部分が大きく再現性は難しいものです。筆者が、イケメン俳優と同じ服を着て、同じ髪型をしてもおそらく不快感を与えると思われます<sup>†3</sup>。

私たちが、見た目から信頼を築くためにできることは、自身に似合う（不快感を与えない）格好を模索することです。似合う服装、髪型などは比較的簡単に試すことができます。幸いなことに、現在ではスーツだけではなくビジネスカジュアルが認められる職場も増えており、自分に似合いながらも働きやすい格好が探しやすい環境が整っています。

「見た目」で、すべてのコミュニケーションが円滑に行えるわけではありませんが、私たちは少なからずの信頼関係を築くことができます。

†1 筆者はフォーマルなスーツで客先を訪れた際に「堅苦しいのでやめてほしい」と言われ、Bさんのようなスタイルに変えた経験があります。何が一番良いのかは、相手の組織文化にも関係してきます。

†2 ルッキズム (lookism)。外見重視主義とも呼ばれます。「あの人は美青年だから仕事ができそう」など、人を外見で評価・判断する差別思想のことと言います。

†3 残念なことに、筆者は小太りで積極的に痩せるつもりも現在のところありません。そして、頭の薄毛は着々と進行しています。

## 2.9 リモートワークで失うもの

---

皆さんが働く環境は、リモートワークでしょうか？それとも出社勤務でしょうか？もしかしたら、リモートと出社のハイブリット形態で働いている方もいるかもしれません。筆者は、働き方改革<sup>†1</sup>が聞こえ始めた頃にリモートワーク<sup>†2</sup>という働き方に魅力を感じながらも「現実には難しいのだろうな」と考えていました。

ところが、そのタイミングで発生したのが、新型コロナウイルス感染症（COVID-19）です。不謹慎であることを重々承知で述べると、悲しいニュースを多くもたらしましたが、リモートワークを加速させる契機になったことは間違ひありません。

世界的にみても、リモートワークの是非を判断するのは難しい状況が続いていますが、リモートワークがもたらすメリットが大きいことは、すでに多くの人たちが実感しています。大都市圏に住む人々は、通勤地獄から解放され睡眠時間を増やすことができました。遠距離の不要な出張を減らし、Web会議が一般的になり距離の制約から解放されました（日帰り出張往復2000km！？馬鹿げている！）。

組織としても、地方の人材活用をしやすくなり、オフィス費用の削減などがメリットとして挙げられます。昨今では、働く条件の最優先事項がフルリモートになる人もいるほど、一般的な考え方になりました。

しかし、これらのメリットを実感する一方で、私たちが失うものも見えてきました。

### コミュニケーションの難易度

Web会議は距離的な制約を排除しましたが、必要となるコミュニケーションの難易度を高めました。声は伝わりにくくなり、何度か聞き返すことが増えました。参加者の表情を見るためには、小さなギャラリービューから注意深く探さなければなりません。ネットワーク回線不調による唐突な退出者の対応も必要になりました。

### 学ぶ機会の減少

出社であれば近くにいる同僚に気軽に質問したり、ちらつと横目でみながら気になればすぐに尋ねたりすることができます。しかし、リモートワークでは学ぶものや知るべきことを明確にし、自分から積極的に取り組む姿勢がより必要になりました。

## 学んでもらう機会の減少

教える側として、近くで見ながら気になった部分を指摘をしたり、状況をみて新しいことを教える機会も減少しました。指導者には明確な指導方針と管理が必要となり、学ぶ方には質問すべき内容の選択や判断がこれまで以上に必要になりました。これは、新人など初学者が孤独を感じる要因にもなっています。

## 偶発的な出会いの減少

他部署や通常は交流の少ない同僚と関係を築く機会も減少しました。エレベーターや休憩所での雑談など、日常的なコミュニケーションの中で起きていた偶発的な出会いは難しいものになっています。これは、ビジネスアイデアの創出などの大きな機会損失だけでなく、異なるバックグラウンドを持つITエンジニア同士のちょっとした技術の意見交換などの機会も奪いました。

## 帰属意識の低下

出社勤務であった強制的なコミュニケーションは減少しました。リモートワークでは個人で過ごす時間が圧倒的に長くなり、共感や団結を感じる機会が減少し組織やグループに所属している帰属意識を低下させる要因になっています。

これらを問題と認識し、チャットツールやその他コミュニケーションツールの運用方法を見直し改善に向けようとする動きも増えています。しかし、新たにIT業界へ入ってくる世代にとっては、コミュニケーションに求められる要求が高くなっていることは間違いないありません。

今後、リモートワークが定着する方向に進む場合、異なる世代や経験を持つITエンジニアたちは、これまで以上にお互いが意識的にコミュニケーションを取り、協力し合うことがますます必要になっていくでしょう。

筆者のように出社勤務があたり前で、リモートワークの過渡期を経験した世代は、これら失うものや環境の変化に柔軟に対応できると考えています。私たちは、リモートワークで得られた利点だけでなく、失うものも認識しそれに立ち向かう方法を改善し続ける必要があります。

†1 2019年4月1日に施行された「働き方改革を推進するための関係法律の整備に関する法律」のことです。

厚生労働省、「働き方改革」の実現に向けて、2023年9月27日

<https://www.mhlw.go.jp/stf/seisakunitsuite/bunya/0000148322.html>

†2 当時はテレワークと呼ぶ方が一般的だったと記憶しています。

## 2.10 自分の時間、他者の時間

---

私たちは、パソコンやスマートフォンを使用してインターネットで多くの情報を得ることができます。それも万能ではありません。私たちがよく調べる技術的な情報だけでも、インターネットを駆使してもわからないことで溢れています。そのような場合、短期間で解決する方法は他者（有識者）に質問することです。

私たちが質問をするとき「他者の時間を奪っている」ことを忘れてはいけません。質問をすることで「内容を聞く時間」「考える時間」「返答する時間」など、時間を使うことを暗黙的に他者に強制しています。端的に言えば「私の時間を無駄にしないために、あなたの時間を使わせろ」と言っています。私たちは他者から奪う時間を最小限にしなければいけません。

「詳しくない技術で発生したエラーを他者に質問する」ことを例に、他者から奪う時間を最小限にする方法を以下のステップで説明します<sup>†</sup>。

### 1. 目的（ゴール）を明確にする

質問によって、どのような結果を得たいのかを明確にします。目的は「エラーを解消すること」という誤解をすることがあります。多くの場合それは正しくありません。「本来の目的」を実現するための実装や操作の中で、エラーが発生しているはずです。どのような処理をしたくて実装しているのか、操作の結果どのようなことをしたかったのかなどの「本来の目的」を明確にする必要があります。

### 2. 概要・詳細をまとめる

質問内容を簡潔に表現し、相手が瞬時に理解できる概要（タイトルのようなもの）を數十文字程度でまとめます。メール件名と考えると想像しやすいでしょう。

次に、詳細をまとめます。詳細の内容は状況により多岐に渡ります。目的の背景・環境情報・エラーの内容・エラーに至るまでの経緯などです。内容にもありますが、筆者は以下の3つを指針にしています。

- 自分の感想は省き事実を列挙する
- 可能な限りわかりやすく箇条書きにする
- 数分以内で説明できる

### 3. 調査内容をまとめる

調査内容と硬い言葉を使いましたが、自身で調べたこと・試したことなどの試行錯誤した内容と結果をまとめます。参考にしたリファレンスや参考書のページ箇所、インターネット情報のURLなども合わせてまとめます。ここでも事実のみとし感想は含めません。

### 4. 自身の見解をまとめる

「1.」～「3.」のステップの情報を元に、自分の見解や考えをまとめます。ここでは、自分の感想などを含めても構いませんが、これまでのステップに含まれない情報を出してはいけません。相手が混乱してしまいます。

- 試したことで解決できると思った根拠
- 原因と考えているもの
- 全く理解できないもの

### 5. 補足資料を準備する（オプション）

質問が簡易な文章だけで意図が伝わりにくい場合は、図解などの説明資料を準備します。時間をかけて綺麗にする必要はなく、意志伝達に必要な最小限のものを準備します。

### 6. 質問する

「1.」～「5.」のステップの情報を元に質問を行います。相手から詳細な確認があった場合は、感想ではなく事実を述べます。わからない内容であれば「わからない」と正直に答える勇気が必要です。

### 7. 「ありがとう」を伝える

質問に対する解決策が得られたかどうかに関わらず感謝を伝えましょう。他者が私たちのために時間を使ってくれたことに変わりありません。

遠回りのように感じますが、ステップを踏むことで私たち自身が質問内容を正確に把握し他者の時間を奪う時間を削減するために役立ちます。そして、これらのステップに使う私たちの時間は、5分～10分程度です。

†1 本節のような遠回りな考え方を好まず「すぐに聞きに来い！」という人もいます。その場合は指示に従いましょう。また、障害など緊急度が高い場合は、このようなステップは踏まず、すぐに助けを求めてください。

第

# III

章

## 無能のためのプレゼンテーション

本章は、無能なITエンジニアとして生きていくための  
プレゼンテーションについての教訓になります。

一般的にプレゼンテーションと言えば、  
大勢の前で行う発表をイメージをしますが、  
私たちは誰しもが小さなプレゼンテーションも行っています。

日常的に行われる会議や会話なども含めた  
プレゼンテーションに大切なことを見ていきましょう。

**1** プレゼンテーションは自己紹介

**2** 文章の前にストーリーを考える

**3** 大きさは2倍、早さは半分

**4** 立った！立ったわ！

**5** ホワイトボードは怖くない

**6** 失礼という誤解

**7** 基本はみっつ

**8** 内職はしない

**9** 全員に愛される必要はない

**10** 長時間会議は悪か

## 3.1 プレゼンテーションは自己紹介

---

プレゼンテーションとは一体何でしょうか？一般的にプレゼンテーションは、情報伝達手段の1つとされています。デザインや画像などのビジュアルな情報は、資料を通じて伝えられ、言葉を話すことで、より正確な情報を伝えることができます。しかし、プレゼンテーションは、単に美しいスライドや大勢の前で行う発表とは限りません。私たちが何かを伝えようとする行為は、すべてプレゼンテーションなのです。

プレゼンテーションは、コミュニケーションと多くの共通点があります。筆者は、以下のように位置付けています。

- コミュニケーションは双方向の情報伝達に重きを置いている
- プrezentationは自身から発信する一方の情報伝達に重きを置いている

会議という場面を考えてみましょう。会議の主要な目的は、参加者全員で情報を伝え合い共有するというコミュニケーションです。そして、私たちが発言をするときは、スピーカーという役割で、聴衆に対して「プレゼンテーション」を通じて情報を共有します。会議の参加者一人一人がプレゼンテーションを行うことで、コミュニケーションが実現しているのです。

私たちがプレゼンテーションを行うとき、忘れてはいけないことが2つあります。

### 私たちが主役である

1つ目は、私たちが発言（プレゼンテーション）をしている間は「私たちが主役である」ことです。会議全体を主導する役割が他にいたとしても、私たちが発言をしている間は、私たちが主役です。聴衆は、私たちのプレゼンテーションに目と耳を傾け、理解することに努めてくれます。

私たちが会議に参加するとき「発言すること」が大事なのではなく、聴衆ができるだけ容易に理解し、コミュニケーションに繋げられるプレゼンテーションを行うことが大切です。これはプレゼンテーションの主役である私たちが担う責務です。

## プレゼンテーションは私たちだけのものである

2つ目は、私たちのプレゼンテーションは「私たちだけのものである」ことです。同じ会議室、同じ参加者、同じプレゼンテーション資料を使う場合でも、それらの情報が伝える印象は発言者によって異なります。「私たちのプレゼン」と「私たち以外のプレゼン」では、同じ情報を並べても聴衆の理解度や納得度も異なってくるのです。

私たちはプレゼンテーション資料の情報を単に伝える役割ではなく「この情報をどう受け止めて欲しいか」という思考や想いも伝えようとしています。もし、情報の伝達だけが目的ならば、プレゼンテーションの主役（私たち）は必要ないでしょう。資料を送付するか、他の人がプレゼンテーションをすれば良いのですから。

私たちは就職や転職活動の際に「自己紹介」を求められると、感情を含めて必死に情報を伝えようとします。素晴らしいプレゼンテーションを行おうと尽力します。しかし、毎日の会議や日常的な小さなプレゼンテーションではどうでしょうか？

プレゼンテーションをする時は思い出してください。

私たちがプレゼンテーションをしている間は「私たちが主役」です。聴衆は私たちに注目してくれています。

私たちのプレゼンテーションは「私たちだけのもの」です。私たちは、自身の想いを含めて情報を伝えようとしています。

多くの場合、聴衆は情報だけを求めているわけではありません。私たちのプレゼンテーションを求めています。

## 3.2 文章の前にストーリーを考える

---

多くの場面で、私たちは文章を書く必要があります。会議資料・議事録・提案書・要件定義書・設計書・運用手順書・製品マニュアル・メール連絡・日報報告・障害報告など、人によって異なりますが多種多様な文章を作成しなければなりません。技術に触れているよりもオフィスソフトと見つめ合っている時間の方が長い人もいることでしょう。

私たちはこのような文章を書くために「綺麗な文章を書くための学習」を行うことがあります。文章フレームワーク<sup>†1</sup>では、PREP法・SDS法・DESC法を学びましたか？文章テクニック<sup>†2</sup>では、主語を明確にする・曖昧な内容を書かない・専門用語を使わないなどを学びましたか？

私たちはもっと簡単で大切なことを思い出すべきです。それは「相手に伝わってほしい」という思いです。そして、相手に伝えたい時に一番必要なのは「ストーリー（物語や筋書き）」を作ることです。多くの人たちは、分かりやすいストーリーに興味を抱き、文章を通じて理解を深めようとしています。どれだけ綺麗な文章であっても、分かりにくいストーリーは、綺麗な文章の理解よりも先に文句が出てくることでしょう。

筆者がストーリーを優先するネガティブな理由もあります。

文章フレームワークは素晴らしい考え方の1つですが、抽象的であり短いプレゼンテーション（ここでは発表会のようなプレゼンを指す）やメールなどの短い文章では役立ちますが、要件定義書・設計書・手順書など数十～数百ページに及び、章ごとに複数要素が絡む文章に適用しようとすると難しいことがあります。私たちがフレームワークに正しくあろうとすると混乱するでしょう。

文章テクニックも素晴らしいノウハウが多くあります。私たちの文章を飛躍的に綺麗にしてくれますが、それはストーリーが組み上がった後の段階です。文章の綺麗さを先に考えると、整合性のないストーリーになる可能性が高く、分かりやすいストーリーの完成よりも資料の作成期限を先に迎えることでしょう。

分かりやすいストーリーを作成するための4つの簡単な考え方を紹介しましょう。

## 読み手の立場を意識する

1つ目は「読み手の立場」を意識することです。「どのような結末（結果）を望んでいるのか」「顧客なのか組織内の人間なのか」「技術的に精通しているのか、不得手なのか」などを読み手の立場になって考えることです。

## 状況を考慮する

2つ目は「状況を考慮する」ことです。同じようなシステムの設計書でも「既存踏襲によるシステム更改」と「新規構築によるシステム更改」では内容が大きく異なります。既存踏襲の場合、既存と異なる部分に注視しなければなりません<sup>†3</sup>。新規構築であれば、既存と異なる部分はもちろんのこと、マイグレーションや運用に関しても既存踏襲よりも多くの考慮が必要になります。

## 自分が主人公にならない

3つ目は「自分が主人公にならない」ことです。私たちはストーリーを作る側ではありますが、主人公ではありません。得意な部分だけ詳しく説明し、苦手な部分を疎かにしてはいけません。私たちが目立つことではなく、ストーリー全体が伝わることが必要です。

## 自分が読み手になる

4つ目は「自分が読み手になる」ことです。ストーリーと文章が一旦完成したら、翌日や数日後に読み手の立場になり読み返してみることです。可能であれば、同僚などから読み手を探し、実際に読んでもらいフィードバックを得ましょう。時間的猶予が必要ですが、私たちに気づきをもたらしてくれます。

私たちが仕事で文章を作成するときの優先事項は「伝わること」「理解してもらえること」です。そのためには、まずストーリーが必要です。ストーリーができたら綺麗な文章作成に取り掛かりましょう。

†1 いざれも簡潔に納得感のある文章を構成するための手法です。

PREP法：Point（結論）、Reason（理由）、Example（事例）、Point（再度結論）

SDS法：Summary（要約）、Detail（詳細）、Summary（詳細を含めた再度のまとめ）

DESC法：Describe（描写する）、Express（説明する）、Suggest（提案する）、Choose（選択する）

†2 本節では文章フレームワークに依存しない細かなテクニック全般を指しています。

†3 既存踏襲なので異なる部分はないはずですか？残念ながら現実はそうなっていません。完全な既存踏襲是不可能でしょう。

### 3.3 大きさは2倍、早さは半分

---

私たちは、日常的に声を使ったプレゼンテーションをしています。リモートワークの普及により、チャットやメールのやりとりが増えましたが「設計レビュー」「難しい問題点の相談」「定例会」「電話連絡」など、声を使って意志伝達する機会は依然として多くあります。このような声を使う場面では「なんて言った?」「もう1回お願いします」などと言われることがあります。声が聞き取りづらかったことが原因でしょう。もう一度同じことを話しましょう。おそらく、聴く側は先ほどより慎重に私たちの声に耳を傾けてくれるはずです。

Web会議などのオンライン環境では、ネットワーク環境の不調などで声が聞き取りづらくなる要因はありますが、対面などのオフライン環境でも、声が聞き取りづらい人は一定数存在します。これらの人たちには、4つのタイプがあると考えています。

1. 声量が小さいタイプ
2. 早口なタイプ
3. 声量や話す早さに問題ないが吃った（どもった）り詰まるタイプ
4. 上記の「1.」～「3.」いずれかの複合技タイプ

筆者は「1.声量が小さいタイプ」と「2.早口なタイプ」の複合技タイプでした。声の聞き取りづらさというのは、私たちにさまざまなデメリットをもたらします。

- 相手に不快感を与えててしまう（シビアな局面では死活問題）
- 自信がない、頼りない印象を与えてしまう
- 意思疎通に時間を要する

これらに立ち向かい改善をすべきですが、大きな問題が2つあります。

#### 自分の声に鈍感

1つ目の問題は、私たちの意識も耳も自分の声には鈍感なことです。手元にスマートフォンはありますか？音声の録音や動画撮影でもかまいません。録音しながら、ビジネスの場で話すように喋ってみてください（どんな声か意識してみましょう）。では、再生してください。まるで別人が喋っている感覚に襲われるはずです。

この問題の対処法は、定期的（1週間や1ヶ月の間隔など負担にならない頻度）に録音し、自分の声を聞き返すことです。もしくは同僚などからフィードバックを定期的に貰うことも有用です。

## 改善には時間が必要

2つ目の問題点は、声量や早さの問題点に指摘をもらったり自身で気付いたとしても改善するために多大な時間を要することです。20年以上続いている話し方を変えるのは容易ではありません。

この問題の対処法は、話をする際に可能な限り意識することです。筆者の場合は、2倍大きな声で話し、早さは半分程度のゆっくりしたイメージをしています。このイメージを持っていても現実ではイメージより小さく、早くなることが多いため、大きすぎる・遅すぎるくらいの感覚を持つようにしています。

筆者は、数年と長い時間を要しましたが、デメリットを回避できる程度には改善できたと感じています<sup>†1</sup>。

話すことが苦手な人は、これらの問題の改善に取り組むことをお勧めしますが、強制するつもりはありません。一般的に、声に関して以下のような誤解があると感じています。

- 練習すれば「誰でも」大きな声が出せる
- 練習すれば「誰でも」聞きやすい話し方ができる

筆者が知っているだけでも声に関する症状では「吃音症」「小声症」があります。これらは、先天性・後天性の要因があり、改善する人もいれば長く苦しんでいる人もいるでしょう。

では、このような人たちはどうすべきでしょうか？本人の希望であれば改善（リハビリ）を頑張ることも1つの方法ですが、筆者は「話さない」選択肢もあると考えています。私たちの仕事には、チャット・図・文章を駆使することで意思疎通を行える方法が多くあります。顧客向ければ得意な人に任せることもできます。そのような理解ある場所を探し「話すこと以外で組織に貢献する。それも選択肢の1つです。

<sup>†1</sup> 現在でも、声が小さくなったり早口になることがあります。特にプレッシャーを感じている時は顕著になります。流暢で安心感のある話し方になるまでには、まだまだ多くの時間が必要なようです。

## 3.4 立った！立ったわ！

---

私たちの仕事は、大勢の前で素晴らしいプレゼンテーションを行い、スタンディングオーベーションを受けることではありません。しかし、私たちにも大勢の前でプレゼンテーションをする機会はあります。

- 技術担当として技術関連の説明や設計・実装レビュー
- 障害対応や対応方針の報告
- 新人採用活動時の（技術部門として）会社説明・新入社員への技術研修
- 業務担当者へのシステム・運用手順説明
- PMやPLとしてメンバーへのプロジェクト説明

これらの場面は、組織内や顧客向けなどさまざまですが、私たちが話す内容を考え、説明資料を準備し、プレゼンテーションを行わなければならないことがあります。筆者は大勢の前で話すことが苦手です。前日などは、緊張して眠れないことがあるほどです。そして、プレゼンテーション当日に悲しいことが起こります。

聴衆の中にポツポツと寝ている人を見つけるのです（こっちは前日疲れもしなかったのに！）。プレゼンテーションが終わったあとも悲しいことがあります。聴衆が寝るプレゼンテーションは、往々にして周りから「残念なプレゼン」と評価されてしまうのです。

筆者の経験から、聴衆を眠たくさせる要因には次のようなものがあります。特に聴衆者が十数名以上であり、1時間以上などの長時間に及ぶ場合は顕著になります。

- 話し方に抑揚がなく単調である
- 同じような話や言葉（～思います。など）を何度も繰り返す
- 一方的に話し続ける
- 資料（投影しているスライドなど）を読み上げるだけ
- 聴衆がプレゼンテーションの内容に興味を持っていない

これらを全て解決した素晴らしいプレゼンテーションを行うには、多くの経験が必要ですが、私たちスピーカーが2つのことに気を付けることで、聴衆の睡眠をある程度抑制することができます。

## 動きを取り入れる

1つ目は、私たちスピーカーが動くことです。無意味に動いては、おかしい人になってしまふため避けるべきですが、重要なポイントや資料のページが変わる際に「ここは重要な部分です」などの言葉を添え、以下のような動きを入れます。

- 座っている場合は立ってみる
- ゆっくりとプロジェクタースクリーンを横切る
- 自分自身に拍手を送るなど音を出す行動をする
- オーバーリアクションをする

これらにより、私たちスピーカーに注目させることができ、聴衆の眠気を軽減できます。Web会議が増え、動きを伝えることが難しくはなりましたが、後者2つはWeb会議でも有用です。

## 質問を促す

2つ目は、聴衆を強制的に一時的な主役にさせることです。区切りの良いタイミングで「ここまでのご質問はありますか？」などの質問を投げかけ、聴衆を聞いているだけではなく、プレゼンテーションに参加させましょう。聴衆の中から質問が出ると、注目度が高まります。また、聴衆の同じグループ（部署や顧客など）の人が質問をすれば、彼らはさらに注目するでしょう。

プレゼンテーション時間を圧迫するかもしれません、眠ってしまい話を聞いてもらえないよりは良いでしょう。

これらは私たちのプレゼンテーションを「素晴らしいプレゼン」にはしてくれませんが、少なくとも「残念なプレゼン」にはならないはずです。聴衆は私たちの言葉を最初から最後まで聞いてくれたのですから。

## 3.5 ホワイトボードは怖くない

---

皆さんは、綺麗な字を書けるでしょうか？日常的に字を書いていますか？筆者は幼い頃に書道教室に通っていましたが、綺麗な字を書くには至りませんでした。今では、手元に筆やペンではなく常にキーボードがあり、文字はディスプレイに表示されるようになりました。普段から字を書かないためか、字の汚さは慘憺(さんたん)たるものです。

それでも、困ると感じることはほとんどありません。キーボードとマウスの使い方を知つていれば、パソコンの中にある多数のフォントとディスプレイが素晴らしい綺麗な文字を表示し、オフィスソフトやデザインソフトは、欲しい図形を描画してくれます。紙媒体が必要な場合も心配には及びません。筆者の近くにはプリンタとさまざまなサイズのコピー用紙があります。

しかし、1つだけ困ったことが起こるかもしれません。私たちが「手で書くことができる」ことを忘れてしまうことです。実際に困った筆者の失敗談を紹介しましょう。

ある時、筆者は複雑で入り組んだシステムの顧客向け設計レビューを控えていました。文章だけの設計書では伝わりにくいと考え、多様な図形とわかりやすく色をつけた関連図を作り設計書に追加しました。会議室に入り、意気揚々とレビューで関連図の説明を始めましたが、その関連図は誤っていることが判明しました。

顧客と認識を合わせなければならぬと思い、顧客の意見に耳を傾け、修正するべき部分を「必死に言葉で」説明しました。実際に図を修正すればよかったかもしれません、残念なことに設計書の図はjpg形式に変換していました。元の図形ファイルは複雑で、1つを修正を行うと他の図形がずれてしまう作りでした。

限られたレビュー時間では説明しきれず、詳細はメールで確認しつつ修正し再レビューを行うことになりました。

何が問題だったのでしょうか？実は、その会議室にはホワイトボードが設置されていましたが、筆者はそれを使おうとすらしませんでした。字が汚いことを理由に、手書きで文字や図を書くことを極端に避ける傾向があったからです。

もし、ホワイトボードを使うことができていれば、図の誤っている部分だけ抜粋したも

のを描き、誤りがあれば消して修正し、レビューに活用すれば、意思疎通はもっとスムーズに行えたことでしょう。マーカーも黒、赤、青の3色はあったはずです（とても綺麗な会議室でしたから！）。

筆者と同じような失敗をする人たちをしばしば見かけます。彼らは、会議などで突発的な話になると「頭の中に描いた図や仕組み」を難解な専門用語や略語を使って説明し始めます。話しながら計算をしつつ数値の話をしだす人たちもいます。ほとんどの場合、彼らの横にはホワイトボードがあります。少人数の会議では、コピー用紙とペンで対応できることも多いでしょう。しかし、彼らはそれらの手段を使わず、言葉だけで話し続けることが多いのです。

ITエンジニアのように常にデジタルに囲まれていると、手書きなどのアナログな方法（ホワイトボード、紙とペン、付箋など）を選択肢から無意識的に除外することがあります。しかし、手書きなどのアナログな方法は「思考の整理」「理解の促進」「意思疎通のサポート」ができる素晴らしいツールです。そして、アナログな方法はデジタルな方法よりも即時性が高いです（紙とペンを使うために、OSの起動を待つ必要はありません）。書き慣れないホワイトボードでも読める字を書くことはできるでしょう。読めなければ、誰かが指摘してくれます。

皆さんのが参加する会議室にホワイトボードはありますか？手元に紙とペンはありますか？これらは決して怖いものではなく、非効率なツールでもありません。私たちを助けてくれる便利なツールです。

さあ！どんなに汚くても、思い切って文字と図形を書き殴りましょう！

## 3.6 失礼という誤解

---

私たちはプレゼンテーションを行う際、相手に失礼にならないように振る舞います。身だしなみを整え、アジェンダを考え、綺麗にまとまった資料を準備し、開始時間に遅れないようにします。プレゼンテーション中は、綺麗な言葉使いを意識して情報を伝えます。これらの行動は大事なことです。「失礼を避ける」ことで、相手を不快にさせず有益な時間を過ごすことができます。

失礼を避けるためには、準備に時間を要することが多いですが、準備時間の無い中で、プレゼンテーションを行わなければならぬこともあります。

- 唐突に概要も知らない会議に呼ばれる
- トラブル対応などで召集される
- 技術的アドバイザーなどの役割で一時的に参加する

このような状況では、私たちは「失礼を避けよう」として「より大きな失礼な行動」をとってしまうことがあります。

以下に2つの例を挙げてみましょう。

### 資料の準備ができていない！

急な会議に参加した際、資料の準備が追いつかないことがあります。残念なことに、アジェンダの資料すら用意できていないこともあります。このとき「資料はないけれど、とりあえずプレゼンテーションを始めなければ！」と焦ります。おそらく、この状態で発言した内容は、多くの聴衆には理解されにくいでしょう。

この場合、プレゼンテーションを始める前に、メモ帳を開いてアジェンダや話す内容を箇条書きで列挙しプロジェクターで投影をするか、Web会議であれば画面共有を行います。その場で書き始めたり、装飾機能もないメモ帳を使うのは失礼でしょうか？いいえ、話す内容が伝わらないほうが失礼です。

資料不足は他にもあります。スケジュールについて相談したいけれど、WBS<sup>†1</sup>を作っていない場合はどうすればよいでしょうか？某大手企業が提供している表計算ソフトのB列1

行目に1を、C列1行目に2を入力しましょう。入力したセルを選択し右下の小さな四角を右にドラッグアンドドロップします（1-31が並んだカレンダーの完成です！）。A列に重要なタスクを書いていきましょう。綺麗ではありませんが、何も無いより遥かにマシです。

これらの即席的な方法は、聴衆の時間を数分ほど奪いますが、私たちのプレゼンテーションは格段に伝わりやすくなります。

### この会議は何がしたいの！？

前提が不明確な会議に参加し、他の参加者の発言が終わり、スピーカー役が回ってくることがあります。「さあ！どうぞ！意見をお聞かせください！」。

私たちは、それまでに聞いた内容と知っている技術知識で取り繕った発言をすることがあります。このような表面上の発言は「皆さんを目指す方向は全く理解できていませんが、とりあえず話します」と同義です。私たちはまず「皆さんが何を達成しようとしているのか教えていただけますか？」と質問しなければいけません。

私たちがしてしまう「大きな失礼」には、以下のようなものがあります。

- 相手に伝わりにくい話をすること
- 目的のわからない状況で発言をすること

私たちはこれらの大きな失礼なことを回避しなくてはなりません。他の小さな失礼なことを避けようとして、より大きな失礼なことをしてはならないのです。

準備不足の会議に呼ばれましたか？小さな失礼は一旦忘れ、大きな失礼を考えましょう。大きな失礼は、プレゼンテーションが開始された後でも回避することができます。

†1 WBS (work breakdown structure) はプロジェクトに必要なタスクをまとめるものです。本書では、WBSにガントチャートを追加したものやWBS辞書などもWBSと表記しています。

## 3.7 基本はみっつ

---

大勢に向けたプレゼンテーションだけでなく設計レビュー やメールなど、日常的に行う小さなプレゼンテーションのために資料や文章の準備が必要なことがあります。そして、情報整理や表現方法について頭を悩めます。世の中には、それぞれの場面に応じた有用なテクニックが存在しますが、それら全てを実践するのは容易ではありません。

筆者の最も簡単な考え方 「基本はみっつ（みっつは数字の3です）」 であることを意識することです。「基本はみっつ」は筆者が作り出した造語<sup>†1</sup>ですが「3」を指針にすることにはいくつかのメリットがあります。

### みっつの技術

どのような技術やプロダクトでも類似したものが複数存在します。類似した3つを比較することで、自身が担当する技術や手法またはプロダクトのメリット・デメリットを把握し、説明に説得力を持たせることができます。

全てを詳細に学ぶのは難しいことですが、概要を理解したり機能比較を行う程度であれば、それほど時間はかかるないでしょう。

### みっつの案

私たちは、提案や設計の際に複数のパターンを考えますが、3つのパターンを考えておくことは有用です。パターンを資料に盛り込むことで、相手に選択肢を与えることができます。その際に「私たちが考える最良」「一般的な最良」「その他」などで考えるとさらに良いパターン分けになります。

資料に盛り込まなくとも、優先案が採用されなかった場合などに代替案として使うことができます。3パターンというのは、考えを狭めずかつ多すぎない数字です。

### みっつのアウトライン

「アウトライン」と書いていますが「深さ」の方がわかりやすいかもしれません。資料などで、章・節・項などで区切る数字の並び「1」「1.1.」「1.1.1.」のことです。これが「1.1.1.1.」など「3」より深くなる場合、冗長になっているか別のグループとして纏められる情報が入っている可能性があります。

## みつつの箇条書き

簡潔さを保つために箇条書きを用いることがあります、多すぎる箇条書きは長文と同じほど把握がしにくくなります。箇条書きが3つより多くなる場合は、情報をまとめたり、他の要素として分割することを検討します。

## みつつの色

資料作成の際にどの色を使うか悩む機会もありますが、3色に絞っておけば悩むことなくシンプルに綺麗にまとめられます。以下はWebデザインなどでもよく用いられている配色ルールの1つですが、どのような資料作成にも応用できる考え方です。

- ベースカラー：白など大部分を占める色
- メインカラー：基本となる色（コーポレートカラーなど）
- アクセントカラー：主張したい色（重要事項や注意事項の説明など）

## みつつのフォント・サイズ

フォントやフォントサイズも、3つに区分することで見やすさが向上します。

- タイトルや見出しフォント
- 本文フォント
- その他（引用やコード表記など）

「基本はみつつ」を意識すれば、3つより少ないときや多いときに、情報の整理や伝え方について異なる視点から考える機会を得ることができます。

誤解してはいけないことは「みつつ」は指針であって「絶対に守るべきものではない」ことです。「みつつ」を守ることを優先すると、伝えるべき情報が漏れてしまったり、過剰に薄い内容になるかもしれません。

私たちの資料やメールを見てみましょう。「みつつ」にできることはありますか？

†1 「基本はみつつ」は筆者の造語ですが「3」を指針とした考え方は以前から存在し「3の法則（Rule of three）」や「マジックナンバー3」と呼ばれることがあります。なお、プログラミング用語でもこのような言葉がありますが、本節とは意味が異なります。

## 3.8 内職はしない

---

私たちは日々、時間との競争に追われています。設計書の作成・問い合わせへの返信・トラブル対応・新しい技術の調査や試行など、仕事は尽きません。そのため、会議中に内職（会議とは関係ない仕事）をして、時間を有効に活用しようする人たちがいます。

内職をしてしまう会議には以下のようなものがあります。

- 会議時間が長い
- 会議の参加者が多い（内職がバレないと考えている）
- 自身の担当範囲とは関係ない時間帯がある
- 自身の発言機会がない（もしくは少ない）
- 社内会議など自分にとって優先度が低い

彼らは、これらの会議中に内職で効率的に仕事を進めることができると誤解しています。内職は仕事の効率化を図るどころか、私たちに多くのデメリットをもたらします。

### 評価を下げる

内職は、私たちの評価を下げます。Web会議が多くなり、気づかれないと考える人もいるかもしれません、内職をしていることは比較的容易に気づかれてしまいます。目がずっと画面の中を泳いでいる、キーボードをずっと打っていたりする様子は、他の参加者は目立つことが多いです。そのような人に対して信頼を持つ人は少ないでしょう。

### 集中力の低下

内職は、集中力を下げます。内職で仕事を進めるとしても、その仕事に集中できているでしょうか？集中力のない仕事は、間違いを誘発する可能性が高まります。後で再度見直しや修正が必要になることが多く、二度手間になる可能性が高いでしょう。また本来の仕事であった会議にも集中できません。

### 時間を奪う

内職は、私たちと他者の時間を奪います。もし突然質問された場合、質問の意図を理解しようとするために「もう一度お願いします」と聞き返し、必要のなかつた時間を使いま

す。自分が知りたいことを聞き逃した場合、議事録を読み返したり他の参加者に質問するかもしれません。相手をイライラとさせ時間まで奪ってしまいます。

会議に呼ばれているということは、私たちがその会議において「必要な人材」であるからこそ呼ばれています。即時の必要性がない場合も、会議に参加して経験を積み将来的に成長して欲しいなどの意図があるかもしれません。

チャットやメールの返信など、1分程度で終わるようなものであれば許容範囲かもしれません、それ以外の場合は内職は行うべきではありません。もし、それでも内職が必要な場合、以下の3つの選択肢が考えられます。

- 会議開始前であれば会議を断る（もしくは時間を変更してもらう）
- 代理人に参加してもらう
- 会議中であれば会議室から抜け出す

会議中に内職が必要な場合は、その内職が会議より重要であるか、緊急であるということです。会議開始前であれば、自身が出席する必要性を確認し、出席を断るか可能であれば代理人に参加してもらうよう依頼しましょう。このような交渉をしても、私たちが必要なのであれば、会議の時間をずらしてくれるはずです。

会議中であれば、退出してしまいましょう。そっと静かに退出すれば、他の参加者も察してくれるでしょう。Web会議であれば、チャットに抜けることを入力すれば良いだけです。相手を不快にさせず、また話が聞けなかったことも理解してもらえます。内容を後から聞き直しても、相手は不快にならないでしょう。

そこまでしなければいけないのか？と感じたのであれば、その内職はここまで優先度も緊急性も高くありません。

内職は仕事を効率化もせず、私たちの評価も落とします。今すぐ内職をやめましょう。

## 3.9 長時間会議は悪か

会議が三度の飯より好きという人は、ごく少数かと思います。しかし、複数の人が話し合える会議は仕事にとって重要なものです。チャットやメールで情報をやりとりすることもできますが、リアルタイムで話し合うことで、文字としては伝えづらい微妙なニュアンスや即時の質問と返答が可能であり、そのメリットは計り知れません。そのため、私たちは少なからず会議を行います。

会議の時間については「長時間会議は悪いもの」「短時間会議は良いもの」という考えが存在します。長時間の明確な定めはないものの、筆者は1時間を超えると「長いな・・・」と感じます。しかし、ときには数時間を超える会議もあります。

会議で時間を使うことによって、他の仕事の時間が圧迫されることは、悪いことのように感じられますが、実際には「悪いものが会議を長くしている」のであり「長時間会議が悪いもの」ではありません。

悪いもの	改善する方法
参加人数が多すぎる	参加人数を制限する（意思決定者だけにする）
資料が多くわかりにくい	多くの（不要な）資料を作らない 事前に配布して参加者が読み込んでおく
集中力が下がる	短時間で終わらせる 休憩を入れる
会議を開くことが目的になっている	不要な会議は開かない
何を決めるか明確でない	決めることを明確にしておく
意見がまとまらない	意見をまとめる人を決めておく

表3-9-1 悪いものと改善する方法

会議を長くする「悪いもの」と「改善する方法」について、いくつかの要点をまとめました（表3-9-1）。これらを見ると、改善によって会議時間が短くなることもあるかもしれません、会議時間の長さが評価基準にならないことが分かると思います。

しかし「他者の時間を奪う長い会議は悪いもの」なので「会議の時間を短くして良いもの」にしようとする人たちがいます。私たちは2つのことを忘れてはいけません。

### 悪い会議とはなにか

1つ目に「悪い会議」とは「決めるべきことが決まらない」もしくは「今後決めるための土台となる意志共有ができない」会議です。短い会議を設定しても目的が達成できなければ、それは悪い会議になってしまうでしょう。また、必要以上に短い時間に迫られた意思決定は、誤った判断を下す可能性を高めます。

### 会議を改善する要因

2つ目に会議を良くする（結果的に会議時間が短くなる）には、さまざまな要因が必要のことです。要因には「会議進行役のテクニック」「参加者側のテクニック」「組織的文化」などが含まれます。進行役がどれほど素晴らしいとも、参加者と組織的文化が確立されていない場合、即時に会議を改善することは、既存のシステムをリファクタリングするより難しいでしょう。少しづつ改善する必要があります。

私たちは長い会議への出席を依頼する際、申し訳ない気持ちになることがあります。しかし、ときには長時間が必要な議題も存在します。会議の改善（時間の短縮）を考えることも大切ですが、参加者が長い会議の理由を理解し納得することも大切です。

一方、私たちが長い会議への参加を依頼された際、ひどく残念な気持ちになることもあります。参加者の立場であっても、資料を事前に共有してもらうように依頼したり、自身の役割の範囲で会議の進行を調整するなど、良い会議にできることを考える必要があります。

長時間会議は悪いものよりもたらされますが、長時間会議が悪ではないのです。

## 3.10 全員に愛される必要はない

本節の愛という言葉には「承認されたい」という意味も含みます。マズローの欲求5段階説<sup>†1</sup>によれば「社会的欲求と愛の欲求」は3段階目、「承認欲求」は4段階目に位置づけられています（図3-10-1）。私たちは本質的に愛と承認を求めているといえるでしょう。



同じ考えは仕事にも当てはまります。同僚や顧客から愛されなければ、楽しく仕事をすることは難しいでしょう。提案書や設計が承認されなければ、仕事は成り立ちません<sup>†2</sup>。

また「全員から愛されることができない」ことも知っています。全ての異性や同性から好意を持たれることは不可能です。そのため、特定の相手から好かれようと、見た目を気にしたり、言動を変えたりします。これらの経験は、多くの人が思春期を通して経験しています。

筆者は、上司にプレゼンテーションをする際、上司に愛されたいと考えます。顧客にプレゼンテーションをする際、顧客から愛されたいです。上司と顧客に同じ情報に関するプレゼンテーションを行ったとしても、資料の構成や話し方は異なるでしょう。「上司が好むもの」と「顧客が好むもの」は異なるからです。それぞれの相手に愛されるようなプレゼンテーションを行おうとします。

これは顧客の中であっても変わります。私たちが、顧客にプレゼンテーションを行うときを想像してみましょう。技術が好きな情報システム部門に対しては、技術的な詳細を伝

えることで愛される可能性が高まりそうです。技術が好きではない情報システム部門に対しては、安定性や運用の利便性に焦点を当てる方が良いかもしれません。現場の部門に説明する場合には、技術的な話は最小限にとどめ、使いやすさについて説明する方が良いかもしれません。

私たちの外見も影響を与えるかもしれません。カジュアルな服装が文化の顧客に、スーツは圧迫感を与えるかもしれません。逆に、スーツが標準的な文化の顧客にTシャツとジーンズで訪問すれば・・・どうでしょうか？<sup>†3</sup>

私たちは、仕事で愛されたいと思いつつ、愛されるための行動を忘れがちです。わずかに言葉を入れ替えた提案書や説明資料、単なる暗記のような伝え方は、再利用という便利な側面がありますが、愛されるためには不十分です。

一部の人たちは、多くの人から愛されるようなプレゼンテーションを行います。大規模なカンファレンスのプレゼンターーやコンサルタントたちです。彼らは愛されるための多くのテクニックと経験を駆使し、膨大な時間をかけて準備をしてプレゼンテーションを行います。彼らのように多くの人たちから愛されることは難しいことです。多くの人は彼らが持つようなテクニックも経験も圧倒的に不足しています。

私たちは、全員から愛される必要はありません。多くの人から愛される必要すらありません。「仕事に関連する目の前の人たちに愛される」だけで十分です。それは、多くの人から愛されるよりも、はるかに容易です。

しかし、本来の自分のまま愛されることは稀です。思春期のころを思い出し、愛されるための行動が必要です。

誰から愛されると仕事が楽しくなるでしょうか？その人は、どのような人を好みそうですか？

†1 アメリカの心理学者アブラハム・ハロルド・マズロー（Abraham Harold Maslow）氏により提唱された、人間の欲求を5段階の階層で説明した心理学の理論です。「自己実現理論」「欲求段階説」と呼ばれることもあります。

†2 日本語の承認とマズロー氏の提唱する承認は異なる意味が含まれるため補足しておきます。提案に対する承認は「認める（allow）」ですが、マズロー氏の承認は「尊重する（Esteem）」です。私たちが仕事で提案をするときは「認められる」と「尊重される」この両方を望んでいます。

†3 筆者は、カジュアルな服装文化を持つ顧客にスーツ姿で訪れた際、威圧感があるためカジュアルにして欲しいと指摘されたことがあります。また、厳格なスーツ文化を持つ顧客に（一般的には許容される）スーツ姿で訪れた際に、ワイシャツやネクタイ、靴の色が適切でないという注意をされたこともあります。

第

# IV

章

## 無能のための技術

本章は、無能なITエンジニアとして生きていくための  
技術についての教訓になります。

ITエンジニアといえば、最先端や流行りの技術を使って、  
有名なサービスを作るイメージがあるかもしれません。

私たちに必要なのは、最先端や流行りの技術を  
扱えることではありません。

仕事の役割に求められる技術を正しく扱えることです。

- 
- 1 はじめてのものには、畏怖と敬意と楽しさを
  - 2 動いた感動、理解した感動
  - 3 点で捉える
  - 4 全く同じは存在しない
  - 5 おまじないという嘘
  - 6 できないは出発点
  - 7 必要なものはモダンではない
  - 8 偽りのフルスタックエンジニア
  - 9 私たちは失敗する
  - 10 未経験者と経験者の違い

## 4.1 はじめてのものには、畏怖と敬意と楽しさを

---

ITの技術の進化速度は驚異的です。新しいことを学んでも、次から次へと便利で使いやすい技術が登場します。これは新規に登場する技術やプロダクトに限った話ではありません。すでに一般的に使用されている技術であっても、バージョンアップなどにより、機能拡張や新機能が追加されることがよくあります。

私たちは担当する技術分野において、知らないこと（はじめてのもの）を学び続ける必要があります。これは10年後も20年後も変わらないことでしょうし、おそらくITエンジニアという仕事をしている間は、必要とされ続けることでしょう。

「はじめてのもの」とは、これまで経験のない新しい技術だけではありません。プログラミング言語・プロダクト・プロトコル・バージョンが異なる（それが古いバージョンであったとしても！）のであれば、それは全て「はじめてのもの」になります。

今後、私たちが「はじめてのもの」を学んだり触れる際に忘れてはならない3つの重要なことがあります。

### 畏怖

「はじめてのもの」に「畏怖」の念を抱かなければなりません。特に新しい技術は、メリットが多く語られますが、デメリットについては語られることが少ない傾向があります。その時点で、デメリットが発見されておらず語れる人がいない場合もあります。

自身が精通していると思っている場合も、新しいバージョンでは仕様が追加されていたり、古いバージョンではこれまで当たり前であった仕様がない可能性もあります。また、公表されていない未知の動作や不具合が含まれているかもしれません。

これらは、お試しで触れたり、検証程度では発見が難しいものです。「畏怖」の念を持つことで、便利な側面や当たり前と思っていることの中に潜むデメリットや問題となる可能性のある部分に気づく機会が増えるでしょう。

### 敬意

「はじめてのもの」に「敬意」を払わなければなりません。新しい技術やプロダクトが一般提供されるまでには、それらを実現するために多くの人たちの労力と資金が費やされています。そして「その技術やプロダクトがあれば、社会に対してメリットがあるだろ

う」という開発者たちの想いも込められています。

私たちは、それらを使うために、ライセンス料や使用料を支払い、ときには不具合に苦しみ不満を持つこともあるかもしれません、それでも敬意を忘れてはいけません。敬意を欠いてしまえば「はじめてのものと」と向かい合うことを避けてしまい、有用に活用する機会を失ったり、誤った使い方をするかもしれません。

## 楽しさ

「はじめてのもの」に「楽しさ」を見つけなければなりません。技術やプロダクトを扱うことにより楽しさを見つけられない場合でも、以下のような観点で小さな楽しさを見つけることがあります。

- 実現している仕組みや方法論
- 活躍できそうなケースや事例
- 特定の機能や活用方法
- 技術の派生元や先、それらの発展過程や時代背景

仕事では、古めかしく需要の低い技術やプロダクトを取り扱わなければならぬこともあります。残念な気持ちを抱くことがあるかもしれません、これらの小さな楽しさは、今後数十年続くITエンジニア人生を楽しく過ごすための一助になります。

これらは大切なことですが、偏りすぎてもいけません。

「恐怖」が極端になると「苦痛」に変わり、その技術に対する拒否反応が起きたり仕事自体を恐れるようになります。

「敬意」が極端になると「盲信」に陥り、比較や他の可能性を考えることがなくなり、選択肢を狭めます。

「楽しさ」が極端になると「楽観」に変わり、仕事の責務を忘れさせ、問題を見過ごします。

筆者は、まず「楽しさ」を見つけるようにしています。「楽しさ」がなければ、仕事には、つらいことが多すぎるからです。

皆さんは、どれから取り掛かりますか？

## 4.2 動いた感動、理解した感動

---

私たちは、ITを使用して「動くもの」を作り出します。開発エンジニアであろうと、インフラエンジニアであろうと、要求された動作を実現できない場合、そのプロジェクトは失敗とみなされます。私たちは、さまざまな制約が存在する中で挑戦し、動いたことに感動します。

「動いたことに感動」という言葉は、未経験者や初学者が勉強の過程で得るものと思うかもしれません、現場で活躍しているITエンジニアでさえ、新しい領域に足を踏み入れ、何時間も悩みながら試行錯誤し、初めて動いた瞬間に感動を覚えることはよくあることでしょう。筆者は「よっしゃ！」とガッツポーズをするほど感動することがしばしばあります。

しかし、私たちには、動かすこと以外にもう1つ重要な仕事があります。それは「説明することができるか？」です。私たちは、趣味で技術を使うアマチュアではありません。お金を貰い、自身の仕事の責務を果たす「プロ」として評価されています。

設計や実装レビューでは理由を問われます。トラブルが発生した場合、原因や対処方法を説明しなければなりません。これらの質問に答えるためには「動いた感動」とともに、「理解した感動」が必要です。理解がなければ説明は不可能だからです。

これと似た話に「コピペプログラマ<sup>†</sup>」があります。彼らは既存コードやインターネット上のコードをコピー＆ペーストし、とりあえず動かすことを優先します。そして、動けば内容を理解せずそのままにします。驚くべきことに、仕事でこれらの行為を行う人もいました。筆者は、コピペそのものではなく、コードを理解しようとしなかったことが問題であると考えています。彼らは「理解する感動」を体験しようとせず、コピペをし続けたのです。

では、どこまで理解すべきでしょうか？

開発エンジニアは実行したコードがCPU内でどのように処理されるか説明が必要でしょうか？おそらくはロジックの説明が先に必要になるでしょう。サーバーエンジニアは、某大企業の開発したOSの内部構造を説明する必要があるでしょうか？おそらくOSが持っている機能の説明が先に必要でしょう。

私たちの役割に応じて、どこまでを理解すべきかは異なり、理解を追求すれば際限がありません。これらに明確な回答を用意はできませんが、筆者は「理解する感動」を求めるとき、以下を指標にしています。

1. 自身の仕事の責務範囲で理解できない用語や概念はあるか？
2. 顧客や同じ分野のITエンジニアは、どこに疑問を持つか？
3. 仕様の理解で良いのか、プロトコルやコードレベルの理解が必要か？
4. 扱っている技術に連携もしくは関連した技術はあるか？
5. その他、個人的に興味のある部分はあるか？

細部まで理解できることが理想ですが、私たちは仕事という限られた時間の中で取り組まなければなりません。筆者は「1.」「2.」を優先し「3.」以降は、時間が許せば取り掛かるようにしています<sup>†2</sup>。

また、私たちを「理解の感動」から遠ざけるものに気をつけなければなりません。

- 穴埋め問題のような設計書
- 使いまわされたコード
- この通りにしろと言わんばかりの手順書や参考書
- 過去の慣習に基づく暗黙のルール

これらがすべて悪いわけではありません。時間の有効活用や関係者が円滑に仕事を行うための約束事であることも多々あり、無闇に変更を加えるべきではありません。しかし、「変更しなくても良い=理解する必要がない」という誤解は避けるべきです。

「動いた感動」を得ましたか？次は「理解した感動」を得ましょう！

<sup>†1</sup> コードを参考書やインターネットからコピー&ペーストし、ロジックを理解できないプログラマを揶揄した言葉です。2000年代中盤～終盤ごろから使われていたように記憶していますが、コードのコピペ是非については、それ以前から議論的になっていました。

<sup>†2</sup> 実際は一人で複数プロジェクト、複数技術を取り扱うことも多いため「1.」「2.」でも相当の時間を使うことがあります。時間が足りない時は「仕様」の部分はある程度割り切って簡易な理解に止め、影響度の高い箇所のみに注力して取り組むこともあります。

## 4.3 点で捉える

「点と点を線で繋げる」という表現は、社会人向け研修などで使われることがあります。解釈は人により多少の違いがある印象を受けます<sup>†1</sup>。筆者は、ITエンジニアとして働き出したころ「今は学んでいる技術の関連性はわからないかも知れないけど、将来的には技術と技術が繋がって仕事ができるようになる」という意味で教えられました。

ITエンジニアとして経験を積む中で、このように点と点を繋げることができる人たちを多く見ました。彼らは、問題を解決策に繋ぎ合わせるための技術的視点を持っており、経験年数にかかわらず最短経路で対処できる能力を持っていました（図4-3-1）。参考図は簡略化しているため、経由点はいくつかあると思いますが、彼らが最短経路を見つけることに長けていたことは間違ひありませんでした。筆者は、このように線で繋ぐことが得意ではなく、多くの遠回りをして解決策に到達することが多いです（図4-3-2）。

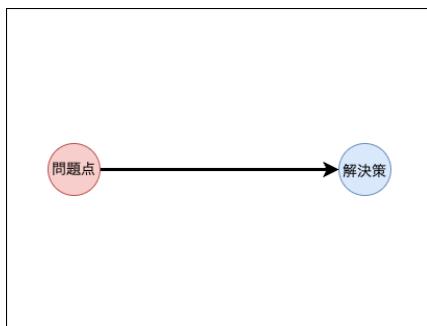


図4-3-1 最短経路の考え方

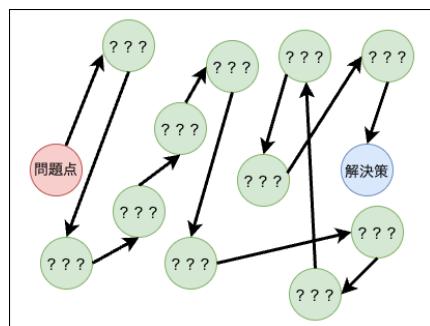


図4-3-2 筆者の考え方

筆者は、彼らのような考え方ができず、会議の話や仕事のスピードについていくことができませんでした。思い悩み、先輩に相談した際に、以下の叱咤激励を受けました。

「馬鹿でも暗記はできるんだから、点で塗り潰せばいいだろ！」

当時、先輩が手元の紙切れに描いたイメージ図を再現しました（図4-3-3）。つまり、論理的思考ができない人は、どんなに悩んでも無駄だから繋げる必要がないほど「知識の点」で埋め尽くせば良いというアドバイスでした。

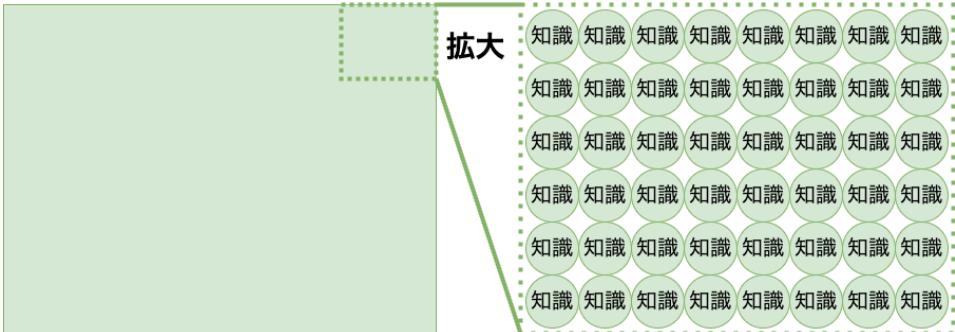


図4-3-3 先輩が描いたイメージ図の再現

知識の点とは、用語や座学的な知識だけではありません。実際に手を動かし、正常な挙動やエラーになる挙動まで思いつく限り試した体験という知識も含まれます。乱暴なアドバイスですが、繋げることが苦手な筆者にとっては、技術を扱うことに対して1つの指針になりました。筆者は現在でも、未知の技術に取り掛かるときは、とりあえず最低限の用語の知識を詰め込み、実際に動かして体験することから始めるようにしています。

ただし、このアドバイスには多くのデメリットも存在します。

- 多くの時間を必要とする
- 費やす時間の割に対応できる範囲は狭い
- 必ず漏れは出てくる（全てを知識にすることは不可能に近い）
- 責務範囲を跨いだ解決策には至らないことがある（開発領域とインフラ領域など）
- PMやPLなどの責務範囲が広い場合の解決策にはならない

これらのデメリットは存在しますが、考えることが苦手な人でも、時間さえあければ、少しづつであったとしても知識は確実に増えていきます。良策ではありませんが、点の繋ぎ方に悩んだり、同じところで悩み続けることに時間をかけるよりも、私たちの助けになるはずです。

<sup>†1</sup> 本節に関係のない余談になりますが、補足しておきます。年齢を重ねて教える側になると、このような表現を使い指導する人が多いことに気づきました。使う人が多い明確な理由はわかりませんが、Appleの共同創業者のスティーブ・ジョブズ（Steve Jobs）氏がスタンフォード大学の卒業生に向けたスピーチの中で「Connecting The Dots（点をつなぐ）」というフレーズを使用したことが一因かと思われます。ジョブズ氏がスピーチで述べた「Connecting The Dots」という発言の意味と本節で述べる「点と点を線で繋げる」の意味合は異なるものです。興味のある方は、インターネットで「Connecting The Dots Steve Jobs」を検索してみてください。スピーチ動画を見るることができます。

## 4.4 全く同じは存在しない

---

過去の経験を過信し、失敗することがあります。経験は、自信をもたせ、問題を未然に防ぎ、仕事の成功率を高めますが、過信は禁物です。

過去に経験したことのある「同じような」仕事に取り組む場面を想像してみましょう。それがプログラミング言語を用いて実装するものだった場合、プログラミング言語が違えば異なるコードになるでしょう。それが、既成のソフトウェアで行われる場合、類似の異なるソフトウェアであれば操作が異なってきます。データセンターで行うような物理的な作業はどうでしょうか？おそらくラック構成も電源構成も違うことでしょう。

このように、同じような経験があったとしても、一部のコンポーネント<sup>†1</sup>が異なれば、前提条件や仕様などが変わってきます。これらの違いを無視すると、私たちが失敗する可能性が高まり、予想以上に多くの時間が浪費される可能性があります。

さらに、同じコンポーネントであっても異なる場合があります。それが「バージョン」です。現代ではプログラミング言語、ソフトウェアなど、さまざまなコンポーネントはセマンティックバージョニング<sup>†2</sup>などのバージョン管理方法に従ってアップデートされています。コンポーネントによって異なりますが、一般的に以下のようないくつかの区分があります<sup>†3</sup>。

- メジャーバージョン：大きな機能変更や後方互換に影響する変更がある
- マイナーバージョン：後方互換はあり一部の仕様変更や機能追加がある
- パッチバージョン：不具合修正や脆弱性の対応など機能影響のない変更がある

これらの変更点は通常、リリースノートに詳細が記載されており、よく確認するものです。メジャーバージョンとマイナーバージョンは機能変更があるため多くの人が注視しますが、パッチバージョンを軽視することがあります。確かにパッチバージョンは軽微な修正が多いいため、確認に多くの時間を使うのは避けたいところです。

しかし、パッチバージョンに含まれる軽微な変更が、私たちの仕事に影響がないと言いたくなるのでしょうか？影響がないと言い切るために、これまでのパッチバージョンの差分コードを比較し、差分の内容を理解する必要があります。ただし、プロプライエタリなソ

ソフトウェアではコードの比較を行うことは不可能です。

他にも異なることは多くあります。プログラミング言語やソフトウェアなど同じコンポーネントであり、同じバージョンであってもデータは異なります。データベースに保存されているデータ、ファイルとして存在するデータなどは、環境に応じて異なります。同じ環境だったとしても、データは時間の経過とともに変化します。蓄積されるデータの種類や量は、増加することはあっても減少することはほとんどありません<sup>†4</sup>。

環境により変わるもののは他にもあります。ロケーションが異なれば使われる言語（英語や日本語など）も異なることがあります。同じ文字コードを使っていても、言語が異なれば、内部では異なる数値で表現されています<sup>†5</sup>。

私たちは、同じような仕事をする時に過去の経験を有効に活用すべきですが、過信してはいけません。「経験したことがある」ことに安心したときに思い出してください。

- 構成しているコンポーネントは同じか？
- 構成しているバージョンは同じか？
- 構成しているデータは同じか？
- 他に異なる要素は存在するか？

私たちがシステムに触れるとき「全く同じ」は存在しないはずです。そして「同じでない」ことは、私たちに失敗やトラブルをもたらします。

†1 本節では、システムを構成する機器、ソフトウェア等全ての要素を指します。

†2 ソフトウェアのバージョニング方法の1つです。「1.23.45」のように「.」で区切って表記されます。

†3 区分に関する説明文は、多くのコンポーネントに見られる方針を簡潔にするため筆者が考へて記載しているものです。セマンティックバージョニングの正式な区分説明は以下を参照してください。

セマンティック バージョニング 2.0.0 . 2023年9月28日

<https://semver.org/lang/ja/>

†4 皆さんが、スマートフォンから不要なデータを削除する時は、ストレージが一杯になり追加で保存できなくなつたときか、買い替えるなどのイベントが発生した時でしょう。このように削除の機会は多くありません。同様に、業務で使うシステムも、日々データは増加しますが、削除される機会はそれほど多くありません。

†5 現代でもマルチバイト文字に関する問題は少なからず存在しています。

## 4.5 おまじないという嘘

---

ITの技術を教えてもらう際に「これは、おまじないなんだよ」という言葉を聞いたことがあるでしょうか？新人研修や初学者向けの指導者や参考書、稀に実務でも耳にすることがあります。

「おまじない（お呪い）」は「神仏やその他不可思議な力を借りて災いを起こしたり、災いから逃れるための呪術」だそうです。しかし、ITの世界にそのようなものは存在しません。すべてには必ず理由があります。

筆者が初学者の頃に聞いて、印象に残っている「おまじない」は、以下の2つです。

- C言語では、#include <stdio.h>を書く
- サーバー機器のファームウェアは最新版にする

しかし、これらは「おまじない」ではなく、それぞれに理由がありました。#include <stdio.h>は、printf関数を使用して文字列を標準出力するために必要なものでした<sup>†1</sup>。そして、ファームウェアを最新版に更新することは、機器の動作不良を防ぐために必要なものでした<sup>†2</sup>。

「おまじない」を使うメリットはいくつかあります。教える側からすると「おまじない」に関連した多くの難解な説明を省略して進めることができます。教えられる側からすると「おまじない」に関連したことを理解せずとも進めることで、混乱を避けることができます。

#include <stdio.h>を理解するためには、ヘッダーファイル・インクルード・コンパイルの仕組みを理解する必要があるでしょう（画面に文字も表示されてないので！）。

ファームウェアを理解するためには、機器の構造・パーツの役割を理解する必要があるでしょう（OSが起動もしていないのに！）。

このように「おまじない」は、知識不足からくる理解のハードルを一時的に下げ、必要な知識を身につけた後で「おまじない」だったことの理解へ導くことができます。

しかし、注意が必要なのは「おまじない」が2つの問題を引き起こす可能性があることです。

## 教える側の問題

1つ目は、「おまじない」を教えた側が、多忙であったり、あるいは当たり前すぎて、本来の意味を教えることを忘れてしまうことです。自分で調べるだろうと、説明を放棄してしまうこともあります。

## 教えられる側の問題

2つ目は、教えられた側が「おまじない」に、疑問を持たずに使い続けることです。何年も「おまじない」を続けることもあります。そして、彼らが教える側になったとき「おまじない」として伝えます。

「おまじない」の問題は、初学者だけの問題ではありません。

- 既存コードを流用する際に「なぜか必要になるコード」
- 手順に書かれているが「意味が不明瞭な操作」
- 理解はできないが「そうしろ」と指示されたこと

これらを「おまじない」のように使う人たちがいます。「おまじない」を理解するためには、多くの時間を費やすことがあります。業務時間内でこれらの「おまじない」を理解するための学習に時間を割くことは難しいでしょう。そのような際「おまじない」と自身に言い聞かせて使ってしまいます。

限られた時間の中で、全ての「おまじない」を排除することは難しいかもしれません。プログラミング言語の仕様やプロダクトの内部動作を完全に把握することは困難を極めます。

そのような時は、仕様であると割り切ることも必要になりますが「おまじない」ではなく「意味がある」ことを常に意識しなければなりません。「おまじない」のままにすれば、バグや不具合の原因となり、問題が発生した際に多くの時間を浪費するはずです。

†1 その後、実務でC言語を使うことはなく素人見解になりますが補足しておきます。#include <stdio.h>がなくともprintf関数の実行は可能です。筆者の環境（「Ubuntu 22.04.2 LTS」「gcc version 11.4.0」）で、#include <stdio.h>を書かずに「Hello World」を試してみました。コンパイル時に警告はですが、実行すると「Hello World」が表示されます。これは、gccはコンパイル時に共有ライブラリ（libc.so.6）をリンクするためです。といっても、やはり#include <stdio.h>は書いておいた方がいいでしょう。

†2 実際のところ、ファームウェアを常に最新版にすれば良いわけではありません。最新版が新たな不具合を抱えている場合もありますし、ドライバやパーツに関連する別のコンポーネントとの互換性を満たさなくなる場合もあります。

## 4.6 できないは出発点

---

私たちは「できる」と「できない」ことの判断に迫られることがあります。これはプロジェクトが開始される際だけでなく、プロジェクト開始前の提案や見積もり段階、プロジェクトの進行中、プロジェクト終了後の運用中のトラブル時などさまざまです。

「技術的には可能だが・・・」の言い回しは、私たちの間でしばしば使われるフレーズです。このフレーズを使うのは、私たちが一般的に思いつくようなことのほとんどが、技術的に「できるだろうな」ということを知っているからです<sup>†1</sup>。それでも「だが・・・」に続く「できない理由」も多く存在します。私たちが「できない」と言ってしまう理由をいくつか挙げてみます。

### 経験不足からの「できない」

すべての技術に精通しているITエンジニアはいません。技術的に可能と知っていた場合でも、経験が不足しているものについては「おそらくできる」という程度にとどまります。私たちが「できる」と言い切るためには、学習や技術検証、PoC<sup>†2</sup>などが必要かもしれません。

### リスクからの「できない」

私たちの仕事にはリスクがつきものです。システム停止・データ消失・リカバリの保証・セキュリティの担保などがリスクとして存在します。これらのリスクを最小限に抑えることは私たちの責務ですが、予期せぬ問題が発生することもあります。もしくは、要望内容が厳しく対応が難しいこともあります。このようなリスクが高い場合、積極的に関わらたくないものです。事前にリスクを説明し承諾を得たり、リスクを抑えるための要望変更が必要かもしれません。

### リソースからの「できない」

システムを作るためには、さまざまなりソースが必要です。人材・時間・予算・コンピュータリソースなどのリソースが必要ですが、これはしばしば不足しています。不足しているリソースを調達するか、なにかを諦める必要があるかもしれません。

## なぜか「できない」

技術的に「できる」はずのことが、なぜか「できない」ことがあります。筆者は以下のように考えることが多いです。

- 数時間悩んでもできない：考えていたより難しいことである
- 1日悩んでもできない：知識が不足している
- 数日悩んでもできない：根本的な勘違いか間違いをしている

時間は参考値です。トラブル時などは、分単位で上記の判断が必要になります。これらの指標は技術的に「できない」理由の判断に役立ちます。

これらは一例にすぎませんが「できる理由」と「できない理由」を明確にしなければなりません。特に「できない理由」は2つの点で重要です。

## 他者に納得してもらうため

1つ目は、他者に「現状のままでは」できないことを納得してもらうためです。営業や上司、顧客たちは、私たちに実現可能かの判断を迫り「できる」と返答すれば安心し、それ以上の言及はありませんが「できない」と返答すれば、常に理由を知りたがります。「できない」というだけでは、彼らは納得しないでしょう。

## 代替案を模索するため

2つ目は「できない」ことを「できる」ように代替案を模索するためです。これらの模索は、私たち個人で完結する場合もあれば、顧客などを巻き込んで模索する場合もあります。どちらにしても「できない理由」が明確でなければ、代替案を模索することすら困難になります。

私たちが「できない」ことを見ついたとき、そこは到着点ではなく出発点です。「できない理由を明確にする」出発点であり、「代替案を模索する」出発点に立ったばかりです。

†1 「知っている」は誇張しすぎたかもしれません。しかし、身の回りにある機器や普段から使っているサービスを想像すると、何かしらの技術を使えば「できるんだろうな」と直感することも多いでしょう。ITという技術を全く知らないければ、それらは魔法のように見え、想像もできないはずです。

†2 PoC（Proof of Concept）。『実証実験』や『概念実証』とも呼ばれます。実際の開発などの前に、実現可能なのか、目的が達成できるかを確認するための検証工程のことです。個人で行う技術検証より、大規模な検証であったりビジネス視点の場合に使われることが多いと感じています。

## 4.7 必要なものはモダンではない

---

ITの世界では「レガシー」「モダン」という言葉がよく使われます。レガシーとは、以前からあるシステム（レガシーシステム）や枯れた技術（広く使われた信頼性の高い技術）を指し、英語のlegacy（遺産、遺物）に由来します。モダンは、新しい技術や新しい開発手法、それらを使用したアプリケーションを指し、英語のmodern（現代の、近代的な）に由来しています。

レガシーはしばしば否定的な意味合いで使用され、モダンは肯定的な印象を持たれることがあります。このような考え方は、DX（デジタルトランスフォーメーション）やモダナイゼーションなどの考え方の普及とともに一般的になりました。

- レガシーは時代遅れの古い技術で作られた負債となるシステム
- モダンは新しい考え方を取り入れた効率的なシステム

ITエンジニアの視点から見た悲しい出来事は、モダンな技術を学ぶことが必要であるか、優先すべきであるとの考え方が広まってしまったことです。「レガシー」や「モダン」の言葉は、技術や手法もしくはシステムを「古くからあるもの」「最近のもの」とわかりやすく分類するのに役立ちますが、これらを単純に「レガシー=悪いもの」「モダン=良いもの」と結びつける考え方は危険です。

私たちが仕事で必要とするのは、仕事の問題を解決するための技術や手法です。その選択は状況に依存し、ときには古典的なレガシーな技術が必要な場合もあります。特定の技術がモダンであるかどうかよりも、私たちの仕事の問題解決の有効な手段であるかどうかを考えなければなりません。

「モダン」という言葉に誤解しがちな3つのことを挙げておきます。

### モダンは自身の価値を高めるという誤解

モダンな技術や手法を知っていることが、ITエンジニアとしての価値を高めるわけではありません。私たちにまず求められるのは仕事の遂行です。モダンな技術は需要が高く、扱えることができれば期待されることも多いのは事実です。しかし「知っている」ことが

私たちの価値を高めるわけではありません。モダンな技術を仕事に活用できて、私たちの価値が高まります。知っていることと活用することの優先順位を間違ってはいけません。

### モダンなものは便利だという誤解

モダンな技術を知ると仕事に取り入れたくなる傾向があります。しかし、モダンなことは往々にして実際の活用事例が少ないものです。問題発生時には、調査などに多くの時間を要する可能性があります。また、それらを扱えるメンバーも限られていることが多いです。採用の理由が単にモダンである場合、一度立ち止まり、それらを導入する利点と学習コストを検討しなければいけません。モダンな技術の利用範囲が、私たち個人だけで完結するのであれば大きな問題にはなりませんが、他者にも広がる場合には、混乱をもたらす可能性があります。

### モダンなものは常に発展するという誤解

モダンであるからと言って、それが将来ずっと発展し続けるとは限りません。モダンなものは、大いに発展する可能性がある一方で、近いうちに廃れてしまう可能性もあります。その発展性を見極めるのは、どんな人にとっても容易なことではありません。発展が未知数な技術よりも、安定したレガシーな技術を学んだり、レガシーなまま改善する方に時間を割り当てた方が良いこともあります。

モダンな技術や手法を身につけることは素晴らしいことです。私たちの可能性を広げ、新たな選択肢を提供してくれます。しかし、モダンなものが全ての問題に適しているわけではなく、現在の私たちに必要なものとも限りません。

筆者は、モダンなものについてある程度の情報収集はしますが、仕事で必要としない限りは、積極的に学んでいません。現在の仕事で求められていることを身につけることが、モダンなものを覚えるよりも必要なことだからです<sup>†1</sup>。そして、ある程度時間が経つとモダンなことを学ぶ参考書が多数出版され、インターネットに情報が溢れてきます。

<sup>†1</sup> 筆者は1つの技術を覚えるだけでも多くの時間を必要とすることも要因としてあります。さらに、モダンなことは情報が少なく、学習にもより多くの時間がかかるため、そこまでの余力がありません・・・。

## 4.8 偽りのフルスタックエンジニア

---

本節は、フルスタックエンジニアの方々に対して、不快感を与えててしまうかもしれないためご注意ください。

フルスタックエンジニアという呼び方は、日本国内では2010年代初頭あたりから広まり始めたと記憶しています。フルスタックエンジニアの範囲は、解説しているサイトなどによって異なりますが、開発に関わる技術要素（ハードウェア、クラウド～ミドルウェア～バックエンド～フロントエンドに至るまで！）の全てを扱えて、ときには上流工程～下流工程を担当できるITエンジニアであると説明されることもあります。近年では「1年目からフルスタックエンジニアになる」という採用のキャッチコピーや「私はフルスタックエンジニアです」と名乗る人、「フルスタックエンジニアになりたい」と希望する初学者が急増しているように感じます。

本節では、フルスタックエンジニアという偽りに惑わされないように、筆者なりの警鐘を鳴らします。

### フルスタックは何でも屋ではない

冒頭で述べた通り、フルスタックエンジニアは、どのような技術要素でも扱え、ときには提案や設計のような上流工程でも対応できる「何でも屋」として扱われることに違和感を抱いています。

米国では「フルスタックエンジニア」に関して「Full Stack Developer」や「Full Stack Web Developer」と呼んでいることが多い印象を受けます<sup>†1</sup>。見てわかる通り、Developer（開発者）です。また、技術要素もフロントエンド・バックエンド・データベースを主軸に、クラウド分野を加える場合が多い印象です。「Full Stack Web Developer」のように「Web」がついているのも、2000年代中頃のWeb 2.0などからWebアプリケーション開発が盛んになり、Web向けの開発者の需要が増加した背景を考えると理解できます。

米国情報が正しいと断定するのは早計ですが、国内のフルスタックエンジニアの説明よりは幾分納得のできる範囲にあるように思えます。フルスタックは「何でも屋」ではなく、フロントエンド・バックエンド・データベースを主軸とした技術要素を扱える開発者といえるでしょう。

## フルスタックエンジニアとは何者なのか？

では、国内でいわれるフルスタックエンジニアとは何者なのでしょうか？フロントエンド・バックエンド・データベースだけを見てみても、多くのプログラミング言語とプロジェクトが存在し、これらを扱うだけでも非常に難しいことは想像に難くありません。

しかし、それにクラウドやインフラ、時にはマネジメントまで含むあらゆる技術に精通した各分野のスペシャリストであるかのように、フルスタックエンジニアを名乗る人たちを見ることができます。おそらく、彼らは所属する組織やプロジェクトでこれらの技術要素に触れたことがきっかけで、フルスタックを名乗るようになったのかもしれません。

問題は「技術要素に触れた」というところです。開発者であっても、サーバーやクラウドの設定を変更することはあるでしょう。インフラエンジニアであっても、コードを読むことはあるでしょう。これらは「触れただけ」であり、スペシャリストになった訳ではありません。触れただけでフルスタックを名乗れば、それは「偽りのフルスタックエンジニア」となるでしょう。

国内で言われるような「真のフルスタックエンジニア」を定義するとすれば「必要となった技術を短期間で学び、仕事で実際に使いこなす能力を持つITエンジニア」のことだと筆者は考えています。そのような人々は、長い経験と努力<sup>†2</sup>により身につけた力であり、存在はごく少数でしょう。まして1年や2年で身につけられるものではありません。

真のフルスタックエンジニアが素晴らしいことは確かです。彼らはコミュニケーションコストを削減し、全てのコンポーネントが調和したようなシステムを短期間に構築します。その結果において人的コストも低減します。これらは、スタートアップ企業から大企業の一部門まで、多くの場面で大きなメリットをもたらします。

しかし、私たちはフルスタックエンジニアを目指すべきではありません。真のフルスタックエンジニアは、非常に長い経験と豊富な知識の上に成り立つものです。偽りのフルスタックエンジニアを目指すと、浅い経験と知識を持った器用貧乏なITエンジニアが誕生するでしょう。

私たちが目指すべきは、プロジェクトの役割に応じた技術要素を学び、使いこなせるITエンジニアです。

†1 筆者が独自にインターネットを使い確認したもののため客観的に示すデータはありません。

†2 「努力」という表現は誤っているかもしれません。真のフルスタックエンジニアになるような人々の多くは、趣味か遊びの延長のように楽しみながら仕事や技術を学ぶ常軌を逸した存在です。

## 4.9 私たちは失敗する

---

私たちはよく失敗します。「実装の間違い」「ケーブルの挿し間違い」「手順を誤る」など、これらの失敗は大きな問題となり私たちを困らせます。これらを抑止するツールもあります。IDE<sup>†1</sup>の補完機能やLinterは初步的な間違いを減少させ、AI支援はロジックの間違いを見つけてくれます。LANケーブルのタグや機器に設定されたDescription、既存の操作・運用手順書は、多くの場合正しい情報を教えてくれます。それでもなお、私たちは失敗をします。

筆者は、開発中や検証中の失敗は、学びの機会であり、理解を深めたり誤った認識を正すことができるため、どんどん失敗すべきだと考えています。多くの失敗を経験すべきですが、本番環境での失敗は避けなければなりません。本番環境に関する設計や作業の失敗は、私たちだけでなく多くの人たちに影響を及ぼし、不幸をもたらします。

失敗を避けるための、いくつかの方法を紹介します。

### 継続的な確認

確認を怠ってはいけません。確認は個人で完結できるものもあれば、他者と連携が必要な場合もあります。

- 技術的に理解していないことを調べる
- 既存資料、前提条件、準備物、スケジュールを確認する
- 暗黙的なルールがないか確認する

これらの確認は、多くの時間と手間が必要な場合がほとんどですが、失敗を回避することに役立ちます。「定時で帰りたい」や「面倒くさい」などの気持ちは一旦忘れましょう。確認を怠ることは、私たちの失敗の可能性を常に高めます。

### 時間を優先しない

タスクの中には、限られた時間の中で遂行しなければならないこともあります。多くの人が関連する現場作業では、10分単位などで区切られることも珍しいことではありません。このような状況では、時間に間に合わせようと焦ったり、他に影響がある確信のない

対応をすることがあります。これらの焦りや確信の持てないことは、失敗につながります。間に合わない、他に影響があることの「可能性を感じた」時点で関係者へ伝えて相談しましょう。私たちに時間的猶予を与えてくれるかもしれません。

## 慣れは存在しない

経験した技術や操作を「慣れていると誤解」することがあります。本番環境の作業において、慣れは存在しないと思わなければいけません。1つの操作をするとき、常に失敗に繋がっている可能性を秘めているのです。マウスやキーボードを触る際「1クリック」「1つのキーを押下する」といった瞬間でも、慣れていると感じてはいけません。

## 対岸の火事から学ぶ

自身に関与していないプロジェクトや自身の責務範囲外の他者の失敗を知った際、関係のない「対岸の火事」として、他人事のように無視することができます。ときには、それらの不幸を笑う人たちもいます。現在は「対岸の火事」かもしれませんが「将来出会う火事」かもしれません。

このような「対岸の火事」の原因と解決方法を調べましょう。火事が落ち着いた後、当事者に聞けば教えてくれるはずです。それは、私たちが「将来出会う火事」を減少させてくれます。

## 休む

影響の大きなタスクを行うときは、事前に体と心を休めなければなりません。心身の疲労は、集中力を低下させ、思考力を奪い、思いもしない失敗をもたらします。そのタスクに取りかかる前に、スケジュールが許す範囲で休む時間を取らなければいけません。たとえ30分などの短い時間だったとしても、心身を休ませましょう。

どれほど経験が豊富になっても、失敗を100%回避することはできません。しかし、本番環境での失敗は可能な限り回避しなければなりません。本番環境の失敗は、多くの人を不幸にし、失敗から生まれた問題を解決するために、多くの時間が必要になります。

そして、私たちの現在の評価を大きく下げ、未来の評価にも影響します。

†1 IDE (Integrated Development Environment)。プログラミングを行う時に使用する統合開発環境を指しています。

## 4.10 未経験者と経験者の違い

---

本書の読者は、主にITエンジニアとして経験を積んでいる人たちが多いでしょう。一般的に「経験者」とは「実務で物事を体験したことがある人」を指します。「未経験者」とは、それらを「体験したことがない人」です。

私たちは、ITエンジニアとしてある程度の経験があるかもしれません、初めての技術やプロダクトに触れる際には、いつも未経験者です。これらの新しい領域を学び、仕事で使えるようになり「経験者」へと成長します。

それでは「ITエンジニアとしての経験者」の線引きはどこでしょうか？筆者は、一般的な解釈では不十分だと考えています。筆者が考える「経験者」とは、以下の3つの要素を備えている人です。

- 業務に従事し物事を体験したことがあること
- 責務範囲と影響範囲を理解できること
- 責任をもって取り組むことができること

これらの要素の1つでも満たせていなければ、未経験者と同様の問題に直面するか、それ以上の深刻な問題を引き起こす可能性があります。いくつかの事例を紹介しましょう。

### 1. 部分知識だけの経験者

Linuxの経験が豊富なインフラエンジニアが、とあるベンダーの物理サーバーに、Linuxをインストールするタスクを頼まれました。多くの台数があったため、彼はKickstartと独自の設定スクリプトを使用してタスクを完了しました。しかし、ベンダーの提供する管理パッケージをインストールしていませんでした。彼はLinuxの経験は豊富でしたが、物理機器やそのベンダー製品の経験が不足していました。

### 2. 指示に頼った経験者

とあるソフトウェアのバージョンアップのタスクがありました。担当した開発エンジニアは、実績のある手順書に従いバージョンアップを実施し、動作確認を終わらせました。バージョンアップ後の環境には不要なファイルが含まれていました。これらの不要なファイルは、開発中に誤って混入したものでした。彼は手順書の内容は理解していましたが、

ソフトウェアのファイル構成などを把握していませんでした。

### 3. ルーチンだけの経験者

とあるシステムのコンバート作業がありました。P2V<sup>†1</sup>によるコンバート方法を採用しており、担当するITエンジニアもP2V経験を持つ人がアサインされました。彼は、経験に則りコンバートの検証をしましたが、エラーを解決することもできず、多くの問題が未解決のままでした。彼が経験豊富だったのは、特定の環境下で手順通りに実施すれば、成功するものばかりでした。

彼らは、一般的には経験者と言われる人物でしたが、ITエンジニアの経験者として必要な3つの要素が欠落していました。

これらの問題を回避する技術的な方法論がいくつかあります。以下はその一例です。

1. 幕等性を保つツールを導入する
2. DevOpsなどの仕組みとベストプラクティスを導入する
3. 環境差異を吸収できる高機能なツールを使用する

本節で伝えたいのは、これらツールや方法論の重要性ではありません。これらは、私たちの経験不足による問題をある程度回避するために役立ちますが、費用がかかったり組織全体での取り組みが必要な場合が多いです。また、これらが採用されている環境であっても、私たち自身の意識が変わらなければ、同様の新たな問題が発生する可能性は存在し続けます。

仕事に取り組む際、自身の「責務範囲と影響範囲を理解し、責任をもって取り組む」意識があるでしょうか？実務で体験したから「経験者である」とすることは危険です。私たちが問題を回避するツールや仕組みは多く存在しますが、それらがあったとしても「範囲の理解と責任を持つこと」を怠れば、多くの問題を発生させることになります。

体験しただけの経験者は溢れるほどいます。範囲を把握し責任を持つ経験者はいつも不足しています。

<sup>†1</sup> P2V (Physical to Virtual)。物理的なコンピュータを、オンプレやクラウドなどの仮想マシンに変換する方法論もしくは変換する技術を指します。

第

# V

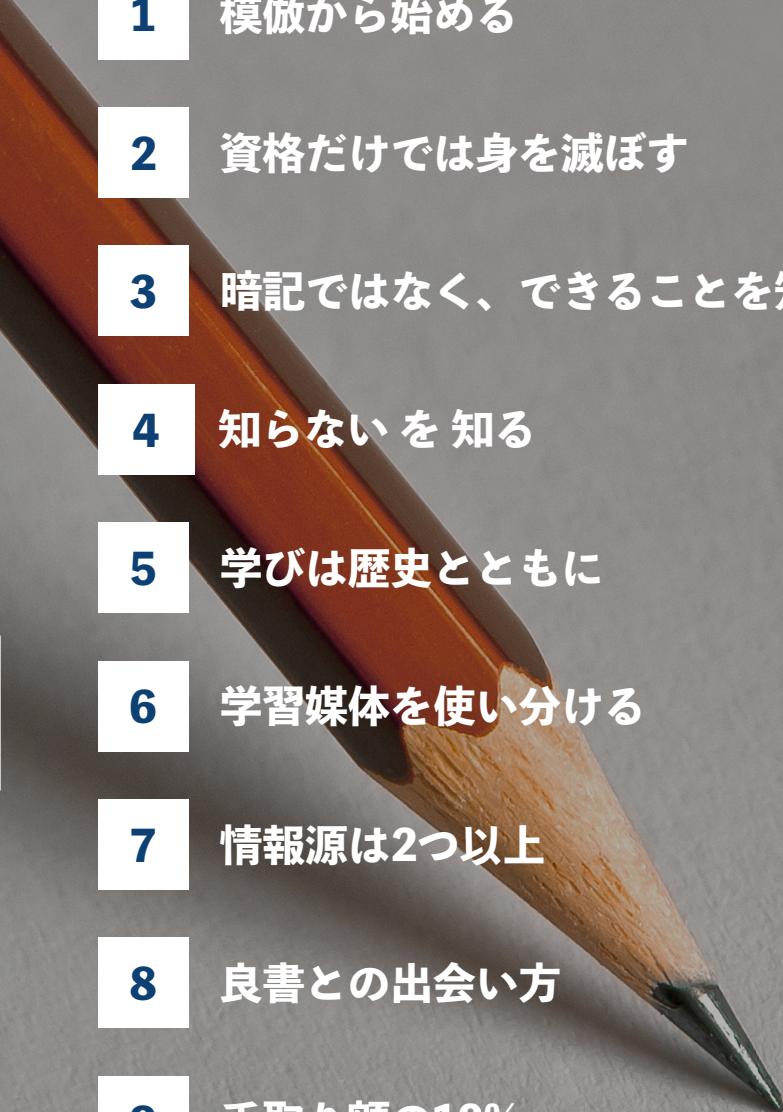
章

## 無能のための学習

本章は、無能なITエンジニアとして生きていくための  
学習についての教訓になります。

近年は、ITエンジニア向けの学習オンラインサービスや  
参考書が数多く提供され、最新の技術情報についても  
インターネットを用いて情報を入手できるようになりました。

私たちは、学習する目的を定め、必要となる学習方法を  
選択しなければなりません。

- 
- 1 模倣から始める
  - 2 資格だけでは身を滅ぼす
  - 3 暗記ではなく、できることを知る
  - 4 知らないを知る
  - 5 学びは歴史とともに
  - 6 学習媒体を使い分ける
  - 7 情報源は2つ以上
  - 8 良書との出会い方
  - 9 手取り額の10%
  - 10 検索エンジンへの聞き方

## 5.1 模倣から始める

---

ITエンジニアの仕事では、ときおり独創性や独自性が必要になることがあります。例えば、ベストプラクティスではない設計や実装方法、もしくは環境に依存した問題の回避策が必要になったときです。また、働き続けるために自分しかできない技術などのテクニックを身につけ、自分の居場所を作ろうとすることもあるでしょう。

しかし、仕事は独自性や独創性の追求ではなく「問題を解決すること」です<sup>†1</sup>。私たちが問題を解決するためには、自身の中だけで探す前に先人たちが実践してきたことを「模倣」することが大切だと考えています。模倣を「他人の真似」として忌避する人もいますが、それぞれが自身の経験から得たことから、良いことであると考え実行しており、私たちに多くの気づきを与えてくれます。

模倣を行う際に、気をつけておくべきことを紹介します。

### 何を模倣するか

模倣する対象を明確にしなければなりません。特定の処理のロジックなのか、設計技法なのか、交渉術や話し方なのかなど、模倣できる項目は多岐にわたります。全体を模倣するのではなく、模倣する対象を可能な限り細分化しましょう。模倣対象を具体的に特定せずに取り組むと、良い側面だけでなく悪い側面も模倣する可能性が高くなります。

### 誰から模倣するか

素晴らしい人物を見つけると、その人を神のように崇めて、すべてを真似しようとすることがあります。しかし、誰しもが得意な部分と苦手な部分を持ち合わせています。模倣は、特定の人物に依存せず、模倣対象を得意とする人物からすべきです。また、模倣は人だけとは限りません。文献・書籍・設計書・既存のコードなども含まれます。そのためには、誰が何を得意としているのかを観察し続けましょう。

### いつ模倣するか

素晴らしい模倣先を見つけたとしても、タイミングや状況によって有益さは異なります。例えば「話し方」に関しては「提案時の自信がある話し方」と「謝罪時の許しを得る

ための話し方」では異なるものです。「技術」に関しては、数年前の素晴らしいアイデアが現在も素晴らしいとは限りません。

模倣しようとするものが、現在の自身にとって合っていることなのかを考える必要があります。

## 模倣を実行した結果

私たちは、模倣を実行した際に、失敗することがあります。失敗したときに行なうことは2つです。

1つ目は「何を」「誰から」「いつ」のいずれかの判断が誤っていた可能性を振り返ることです。誤りがある場合は、修正するか別の模倣対象や模倣先を見つけることができます。

2つ目は、失敗した要因が自身の考え方や性格、人生観に関連している可能性を振り返ることです。これは、筆者が特に失敗した要因です。優秀な人たちから、学習方法を模倣しようと躍起になっていたことがあります。彼らの学習方法は、膨大な経験や純粋に楽しんでいたことで可能なものでした。筆者にとって、そのような模倣は不可能であることに早く気づくべきでした。

模倣した結果を振り返ることで、次に模倣を実行するときに同じ過ちを繰り返すことは少なくなります。

模倣には、私たちが技術的な素晴らしいものを発見したり、理解を深める以外のメリットもあります。それは「人」に対しても素晴らしい発見や理解を深める機会を得られることがあります。

私たちは、自分が苦手なものを得意とする人に対しては容易に興味を持ちますが、自分が得意なものを苦手とする人には興味を持ちません。しかし、模倣の対象は特定の人に依存しません。それぞれの「良い部分を模倣」しようとするため、以前は興味を持たなかつた人たちにも目を向け、模倣すべきものを見つけ出す機会を得られます。

自分自身の居場所を見つけるために、しばしば自分だけにできることを求めがちになります。何もない状態から独創性や独自性を求めるのも素晴らしいことですが、模倣を通じて学び、それを実行することで、最終的にはそれが自身の独創性と独自性になることも同様に素晴らしいことだと考えています。

†1 詳細は「第1章 無能のための考え方」の「仕事は問題を解決すること」を参照ください。

## 5.2 資格だけでは身を滅ぼす

---

誤解のないよう先に述べておくと「資格を持っていた方が良いか？悪いか？」と問われれば、筆者は「持っていた方が良い」と一貫して答えます。

本節で述べるのは、資格取得のメリットとデメリットを把握せずに、資格取得だけに全力を注ぐのは危険であるということです。

IT関連の資格には、国が認定する「国家資格」と民間の企業や団体が認定する「民間資格（ベンダー資格）」の2つの種類があります<sup>†1</sup>。本節で述べることは、どちらの資格にも共通していることです。

ITエンジニアの仕事には、医師や弁護士のように資格が絶対必要なわけではありませんが、多くのITエンジニアが資格取得を目指します。資格取得には、所属する組織と個人の双方にとってメリットがあるためです。

### 組織のメリット

- 案件への参画やプロダクトの取扱いに資格保持者が必要な場合がある
- 組織として技術力を保有していることの対外的アピールになる
- 従業員の評価や採用の基準として使用できる

### 個人のメリット

- 体系的な知識を身につけることができる
- 自分の能力を示し説得力を高められる
- 資格手当を受けたり、昇格することができる

私たちが資格取得を目指すときは「個人のメリット」によるものですが、「個人のデメリット」も意識しておく必要があります。筆者の考える個人のデメリットについていくつか紹介しましょう。

### 資格取得に時間要する

資格取得のためには、少なからず学習時間が必要です。一定以上の点数を取るために、確実にその知識を暗記したり、論述問題に対してのテクニックなどを身につける必要があるためです。また、民間資格の多くは有効期限があるため、資格の更新の学習にも時間を要することがあります。

## 仕事に直結しない知識も必要

資格取得には、幅広い範囲を体系的に学ぶ必要があります。自身の仕事に直接関連しない知識や現在はほとんど使われないような古い技術も学ばなければなりません。仕事へ直結する部分に焦点を当てる判断ができれば良いのですが、網羅的に学習をすると仕事との関連性がわからず、知識の理解度に差が出てきます。

## 能力を過大評価される

IT系の資格の多くは選択式のものが多いです。高難易度の資格を除けば、問題パターンや解答を暗記することで、その技術を経験したことがなかったり、本来必要となる理解に至っていない場合でも取得できてしまうことがあります。しかし、周囲からの評価は資格を「持っている」か「持っていない」だけである場合があります。本来の資格保有者の能力を求められ、柔軟に対応できれば良いですが、そうでない場合は、多大な苦労をするか仕事が失敗します。

私たちは、これらのメリット・デメリットを評価して取得資格を目指す必要があります。筆者の考える資格取得を目指しても良いと思えるのは以下のような場合です。

- 業務命令として資格取得を命じられた
- 参画したい案件に資格が必要
- 自分の能力を示したい（暗記ではなく実力）
- 仕事に直結するレベルで有用と判断できる
- 資格取得が趣味（自信に繋がる、純粋に楽しいなど）

筆者は、学習すること自体がそれほど好きではありません<sup>†2</sup>。自身の理解力や暗記力が他の人と比べ劣るため、資格取得には多大な時間が必要なことを知っています。また、可能な限り仕事へ直接関連することに時間を使いたいこともあり、積極的に資格取得を目指しません。しかし、網羅的に知識を把握したい際や、仕事で必要な知識のために資格向けの参考書を読むこともあります。

皆さんは資格取得を目指していますか？なぜ、資格を取得するのですか？

†1 学位を資格とする場合もありますが、本節では学位に関しては資格に含めていません。社会人となり、仕事の中で取得する可能性のあるIT関連の資格を対象としています。

†2 筆者の学生時代から続いている「単に勉強が嫌い」という個人的なものです。

## 5.3 暗記ではなく、できることを知る

---

私たちは、暗記を主軸とした学習方法を学びすぎました。学生時代は、方程式・単語・年代などを暗記することで、テストで高得点を取ることができました。そして、社会人としてITエンジニアになってからも、暗記だけで資格を取得する人が存在するほどです。

確かに、暗記した知識は役立つものです。暗記した情報は、本やインターネットで調べるよりも、はるかに短い時間で情報を引き出し、使うことができます。

しかし、暗記には多くの時間が必要であり、筆者も含めほぼ全ての人間は、ITに関する全てを暗記することは不可能でしょう<sup>†1</sup>。仕事で触れているITの知識だけでも全て暗記することは難しいはずです。

それでも、私たちは仕事を遂行できています。日常的に必要な情報は、反復により自然と記憶され、一時的に必要となる情報は都度調べることで補完しているからです。

まわりくどい言い方になりましたが、仕事を遂行する上で必要な学習は「何ができるか」を知ることです。「何ができるか」を知っておけば「できるようにする方法」や「できないことの原因」などを調べることができます。

筆者も経験していますが「暗記症候群<sup>†2</sup>」に陥る人は意外と多いと感じます。特に、プログラミング言語やLinuxのコマンドなど、目に見えないことを学び始めると、多くの人がこの症状を発症します。彼らは「何ができるか」を理解しようとする前に「暗記すること」を優先し、多くの時間を費やします。

プログラミングを学ぶ場合、組み込み関数の綴りや引数の渡し方に至るまで暗記を試み、コマンドであれば参考書に記載されているオプションの暗記を試みます。それらは、暗記をしなくとも、IDE<sup>†3</sup>が自動補完してくれ、manコマンドが参考書よりも多くのオプションと詳しい説明をしてくれます。

彼らに共通する特徴は、スマートフォンやパソコンなど、GUIを備えた操作方法を暗記しようとも思わなかったのに、文字になった途端に暗記を試みることです。GUIは、視覚的に見えるものは「できそうなこと」を示し、見えないものは「できないだろう」という制約を与えることで、ある種の安心感を得るためにでしょう。

一方、視覚的に文字情報しかない領域については「何ができるか」や「できること」を判別することが極端に難しくなります。そのため、参考書などでは、例題として「できること」を示し説明しますが、それを暗記することで安心感を得ようとしていると考えています。このような発想の結果、暗記が学習の主軸になってしまいます。

もちろん、最低限必要な暗記は存在します。プログラミング言語の場合、その言語の基本的な文法を覚える必要があります。Linuxの場合は、基本的なOSの知識やディレクトリ構造などを把握する必要があります。しかし、これらの暗記は「できることを知る」ために必要な土台<sup>†4</sup>であり、それ以上の部分の学習は暗記を主軸とするべきではありません。

筆者は、ITの技術を学習する際に必要なサイクルは、以下の4つだと考えています。

1. 基本的な仕様を知る
2. できることを知る
3. できることを試す（実際にできる確信を得る）
4. 余力があれば、できないことも試す

「1.基本的な仕様を知る」と「2.できることを知る」は、インターネットや参考書に記載されているため容易に入手できます。「3.できることを試す」では、うまくいかないことやエラーが発生することがあります。知らないことも新たにでてきます。その時は「できる」まで調べましょう。多くの時間を必要とするかもしれません、暗記に時間を使うことよりも有用なことを知ることができる時間になるはずです。

「できることを知っている」ことは、他の細かなことを暗記しているよりも、私たちの役に立ちます。

†1 本当に不可能だろうか？と調べたところ、ハイパーサイメシア（超記憶症候群）と呼ばれる見たもの全てを詳細に記憶できるフィクションのような能力をもった人たちが実在するようです。羨ましいと思う反面、忘れられないのは辛い人生でもあると思います。

†2 筆者が作った造語です。学習したり見たものを、まず暗記することを優先する人たちを指しています。

†3 IDE (Integrated Development Environment)。プログラミングを行う時に使用する統合開発環境を指しています。

†4 これらの土台は「できることを知る」学習を行う上で頻繁に登場します。暗記をしようと思わなくとも、反復により自然に覚えることが多いでしょう。

## 5.4 知らないを知る

---

「できることを知る」ことは大切なことです、それも私たちにとっては難しいことかもしれません。「できることを知る」ために、多くの情報を読み込み、実際に試す時間が必要だからです。この問題に対処する方法は「知らないことを知る」ことです。これは「できることを知る」よりも短時間で簡単にできることです。

私たちは、初めて触れた技術を当たり前のように使っています。なぜ、そんなことができたのでしょうか？それは、誰しもが知らないことに気づき調べたからです。「知らないことを知っていた」から調べることができたのです。知らないことを自覚できなければ、調べるという行動になっていなかつたはずです

幸いなことに、私たちの周りには「知らないこと」に気づかせてくれる機会が溢れています。顧客は予想外の要望や問題を持ち出し、実現方法について質問してくれます。周囲にいる技術好きなITエンジニアたちは、試した技術の結果や所感を嬉々として説明してくれます。上司たちは、利益になりそうな未知のプロダクトの活用や習得を求めてきます。

しかし、私たちは、これらの外部からの要因だけでなく、自身が何を知らないのかも積極的に知っていくべきです。

筆者が「知らないことを知る」ために注視していることを紹介しましょう

### 自身の仕事に関するこ

私たちの仕事の多くは、ITの技術に関するものです。同じ技術に長い間携わっていたとしても、知らないことが発生する機会は多くあります。以下のものは時間の経過や時代の変化に伴い、知らないことをもたらす代表的なものでしょう。

- バージョン変更に伴う仕様の変更や新機能の追加
- 新しいプログラミング言語、プロダクト、方法論の登場
- ベストプラクティスやバッドプラクティスの考え方の変化
- 脆弱性などセキュリティの脅威

これらは、現時点では知らないことが何なのかを知る術はありませんが、将来におい

て、知る必要性が高いものです。私たちは、ニュースレターやリリースノート、それらに関するニュースサイトで調べることができます。また、私たちよりも詳しい人たちに助言を求めるすることもできます。

## プロジェクト全体

プロジェクトの中にも、私たちの知らないものが溢れています。

- 他者が書いた要件定義・設計書の意図
- 自身以外のスケジュールや課題事項
- 他のチームの技術要素や進捗状況

これらは、知らなくても問題がない場合もあります。他の役割の人たちが考えたり、管理しているからです。しかし、問題が発生した場合や特定の課題に取り組む際に、知らないことを自覚できていれば、円滑なコミュニケーションにより早期の解決に繋がる可能性が高くなります。筆者は、プロジェクトに参画したとき、全体で管理されるファイルや成果物を眺めるようにしています。それらのすべてを理解することはできませんが、調べたり相談する際に役立つことがあります。

## 日々の生活の中

いつの時代にもバズワード<sup>†1</sup>が頻繁に登場しています。バズワードは、ビジネス面を対象とした言葉も多いのですが、その中身に着目するとITの技術や私たちの仕事に関連することも多いでしょう。例えばDX（デジタルトランスフォーメーション）は、必ずしも高度なITの技術を必要とするわけではありませんが、多数の高度なITの技術を使うこともあります。これらは、ニュースサイト・雑誌・新聞・SNSなどで知ることができます。

「知らないことを知る」際に大切なことは「全てを知ろう」という考え方を捨てることです。「言葉だけ知っている」「概要だけ知っている」けれど「他のことは知らない」で問題ありません。

これはリファレンスや参考書などを読む際も同じです。目次をチェックしたり、内容を流し読みしますが、詳細まで読むのは必要な部分のみです。他のことは「知らない」ことを知りたいれば、必要になった際に調べることができます。

<sup>†1</sup> 専門性や説得力のあるように聞こえるが、曖昧な定義のまま広まった言葉のことです。

## 5.5 学びは歴史とともに

---

私たちの使っているシステムで使われているITの技術の歴史の始まりはいつでしょうか？これを明確に説明することは難しいかもしれません。現在の私たちが使っているシステムには、さまざまな技術や手法が結びついているためです。

しかし、それぞれの技術や手法ごとにその「始まり」が存在し、それが現在までの「歴史」になっていることは確かです。

以下に私たちが日常的に使用している技術に大きな影響を与えたいくつかの「始まり」を挙げてみましょう<sup>†1</sup>。

- フォン・ノイマン型コンピュータの始まりは、1945年<sup>†2</sup>
- UNIXの始まりは、1969年<sup>†3</sup>
- TCP/IPの始まりは、1974年<sup>†4</sup>

これらの始まりは、現代の技術に大きな影響をもたらしたものの一例ですが、私たちが使っているプログラミング言語やプロダクトにおいても、それぞれに「始まり」と「歴史」があります。私たちの身近なものである「クラウド」についても触れてみましょう。

クラウドコンピューティングの概念の始まりは、1990年代にサン・マイクロシステムズの「ネットワークこそがコンピュータである」というマーケティングスローガンだと筆者は考えています<sup>†5</sup>。このスローガンは、現代のクラウドコンピューティングへの道を示唆していました。

その後の2000年代中盤以降、AmazonやGoogleなどがクラウドサービスを提供し始め、現在では私たちに無くてはならないほどの技術として確立し、利便性を得られるようになりました。クラウドコンピューティングの概念は1990年代には存在していましたが、それがサービスとなるまでには時間がかかりました。当時のネットワークやコンピュータに関する技術やそれらを利用する人たちの理解がまだ成熟していなかったためです。

現在使われているクラウドサービスにおいても、登場の初期段階では安定性やセキュリティ面などにおいて、ネガティブな見解が筆者の周りでも少なからずありました。しかし、利用者の理解度は向上し不可欠な存在となりました。現在も新しいクラウドサービスが生まれ続けています。

私たちが、技術や手法の学習を行う際に、このような「始まり」や「歴史」を学ぶことには2つの大きな意味があります。

## 需要を知ることができる

1つ目に、これらの歴史を学ぶということは、需要の根本的な理由やその遷移に対応した歴史を垣間見ることができることです。新しいITの技術は、提唱から普及（成熟）までに少なくとも数年～10年以上の時間を要すると筆者は感じています。普及しているということは、ITエンジニアや顧客の立場から見て、それだけ需要が続いているためです。

これらの需要がなぜ存在し、どのように変化してきたのかを理解することは、学習内容に対する納得感を高め、同僚や顧客に対する説明の助けにもなるでしょう。

詳細な歴史や時代背景を掘り下げるには時間がかかりますが、大まかな概要を調べることは比較的短時間でできるはずです。

## 生きた歴史に学ぶことができる

2つ目に、ITの歴史を長く経験してきた人たちから実体験に基づく昔話を聞く機会が得られることです。先輩や年上の上司だけでなく、年下の人たちからも得られるかもしれません。ITエンジニアの中には、昔話が大好きな人が多くいます。私たちが歴史に興味を持てば、笑いながら昔話をしてくれるでしょう。

彼らはプロの語り手ではないため、無駄な話が多く要領を得ないストーリーかもしれません。また、彼らはお酒が入ったときに流暢に話すことが多いため、面倒に感じるかもしれません。しかし、それらの昔話は、彼らが実際に経験し、強烈に記憶している話です。現代のあり方に繋がっている話も多く、私たちが今後直面する問題の一助になるかもしれません。

歴史を通じて、私たちがより理解できるように手助けをしてもらいましょう。

歴史を通じて、私たちが将来直面するかもしれない問題に備える手助けをしてもらいましょう。

†1 1つの技術をとっても「始まり」についての定義は難しいものです。本節では、表記している年代に可能な限り誤解を与えないよう、書籍を参考にしました。それぞれの書籍を補足として記載しています。

†2 ポール・E・セルージ. モダン・コンピューティングの歴史. 株式会社 未来社. 2008年10月. 40ページ

†3 ピーター H.サルス. UNIXの1/4世紀. 株式会社アスキー. 2000年12月. 17ページ

†4 アンドリュー・S・タネンダウム,ニック・フィームスター,デイビッド・J・ウェザロール. コンピュータネットワーク 第6版. 株式会社日経 BP. 2023年3月. 57ページ

†5 筆者はこのスローガンと年代を以下の書籍で知りました。

ニコラス G.カーナ. クラウド化する世界 ビジネスマネジメント構築の大転換. 株式会社 翔泳社. 2008年10月. 134ページ

しかし、インターネット上では、「The Network is the Computer（ネットワークこそがコンピュータである）」は1980年代に生まれた言葉と解説されていることが多いようです。

## 5.6 学習媒体を使い分ける

学習を行う際、仕事のために行うのか、趣味として楽しむために行うのかなど、目的を明確にすることが大切です。併せて、目的達成のための学習媒体の選択にも気をつけなればなりません。筆者が学習に使う主な媒体をまとめてみました（表5-6-1）。

学習媒体	信頼性	網羅性	情報鮮度	手軽さ
匿名の情報	★★★★☆☆	★☆☆☆☆☆	★★☆☆☆☆	★★★★★★
公式リファレンス	★★★★★★	★★★★★★	★★★★☆☆	★★☆☆☆☆
自社のナレッジ	★★★★☆☆	★★☆☆☆☆	★★☆☆☆☆	★★★★★★
市販の参考書	★★★★☆☆	★★★★☆☆	★★☆☆☆☆	★★★★☆☆
IT系雑誌	★★★★☆☆	★★☆☆☆☆	★★★★☆☆	★★★★★★
IT系ニュースサイト	★★★★☆☆	★☆☆☆☆☆	★★★★★★	★★★★★★
開発者ブログ	★★★★☆☆	★★☆☆☆☆	★★★★☆☆	★★★★☆☆
開発者コミュニティ	★★★★★★	★★☆☆☆☆	★★★★★★	★☆☆☆☆☆

表5-6-1 筆者が学習に使う媒体

この表は、各学習媒体に対する信頼性・網羅性・情報鮮度・手軽さという要件に基づく評価を示しており、★マークの数が多いほど、要件を満たしています<sup>†1</sup>。以下は各要件の詳細です。

- 信頼性：情報がどの程度正確で信用できるか
- 網羅性：入手できる情報の多さ
- 情報鮮度：情報の新しさ
- 手軽さ：情報の入手性と理解の容易さ

これらの中で、筆者の利用頻度が特に高い「匿名の情報」「公式リファレンス」「自社のナレッジ」について詳しく説明します。

## 匿名の情報

匿名の情報は、匿名のITエンジニアなどが自身のウェブサイト、技術共有コミュニティ、動画サイトなどで提供している情報を指します。インターネットを検索すれば容易にアクセスでき、情報を簡単に入手できます。しかし、匿名の個人が提供する情報は、専門家が提供しているとは限らないため「信頼性」は低く、妥当性の評価がされていなかったり、前提条件などの記載がないものもあり「信頼性」「網羅性」が低い傾向にあります。

## 公式リファレンス

公式リファレンスは、技術仕様の策定やプロダクトの開発元など、組織から公開されているマニュアルやナレッジの全てを指します。この媒体は、使用するための前提条件や使用方法が詳細に記載されており「信頼性」「網羅性」は非常に高いです。また、情報の更新があるたびに反映されるため「情報鮮度」も高くなります。ただし、情報量が多く、英語での記述が多いなど、必要な情報を見つけ出し、理解をするためには一定の労力が必要となるため「手軽さ」は低い傾向にあります。

## 自社のナレッジ

自社のナレッジは、自社の業務として蓄積された設計書、問題への対応履歴などの媒体を指します。自社の情報のためアクセスは容易であり、業務の内容から蓄積されているため「入手性」「信頼性」も高い傾向にあります。しかし、情報の更新が追いつかず古いままで放置されたり、前提情報が不足することが多く<sup>†2</sup>「網羅性」「情報鮮度」は低い傾向があります<sup>†3</sup>。

いずれの媒体も、どれかを使い、どれかを使わなくて良いわけではありません。学習の目的達成のためには、複数の媒体を使うことが多いでしょう。

重要なのは1つの媒体に依存するのではなく、媒体の得意な部分と苦手な部分を認識し使い分けていくことです。

<sup>†1</sup> この評価は筆者の経験に基づくものであり、他の人にとっては異なる評価になる可能性があります。絶対的な正しさではなく、各自が自身の立場に応じて媒体を評価する際の基準を持つことが大切です。

<sup>†2</sup> 自社内において「当たり前」とされる技術情報や運用については、その知識を暗黙の前提として、省略されることが多いと感じています。

<sup>†3</sup> 情報の更新頻度や詳細度については、組織文化やそのナレッジ提供者による影響が大きいため、かなりのバラつきがあることを考慮する必要があります。

## 5.7 情報源は2つ以上

---

初めて新しい技術を学ぶ際には、学習を行うための情報源があります。参考書やインターネット、または仕事上の指導など、さまざまな情報源があります。気を付けることは、どの情報にもバイアスが含まれていることです。バイアスという言葉は、一般的に否定的な印象を抱くかもしれません、筆者は「良いバイアス」と「悪いバイアス」が存在していると考えています。

- 良いバイアス：経験に基づいて正しい選択や判断を支援できるなど
- 悪いバイアス：主観や偏見によって非合理的な思い込みをもたらすなど

バイアスを客観的に把握し、それらがもたらす悪影響を最小限に抑える必要があります。簡単な方法として、異なる情報源から情報を収集することが有用だと考えています。

例えば、技術的な学習の際、公式のリファレンスと市販の参考書は異なる情報源です。公式のリファレンスには、仕様や使い方についての事実が説明されています。バイアスが無いように見えますが、デメリットなどについては省略されることがあります。これは公式リファレンスが正確性を重視するためです。市販の参考書には、技術的な仕様に加えて、著者の経験に基づく「この場合は、この方法を避けるべき」などの情報が含まれる場合があります。

良いバイアスか悪いバイアスかは状況によって異なりますが、それぞれにバイアスが含まれています。公式のリファレンスと市販の参考書、またはこれらの情報と私たちが実際に動かした結果を比較することで、バイアスを把握することができます。

また、同じ媒体であったとしても、複数の情報源を持つことは有用です。

### 市販の参考書から学ぶ場合

初めての技術を学ぶ際「入門向け」の参考書を購入することが多いでしょう。しかし、同じ技術の入門向けの参考書であっても、さまざまなバイアスが存在します。

- 著者の意図と経験
- 出版社や編集者の影響
- 時代的背景

書籍は「著者の思考やスタイル」が明確に表れる傾向があります。入門向けの参考書で

あっても、書籍の構成や内容の説明・注釈の仕方などは、著者によって異なってきます。

筆者は、初めてのことを参考書で学習する際に「入門向け」と「中級者向け」をそれぞれ1冊ずつ購入することが多いです。参考書の意図として、読者に求める技術レベルも伝えたいレベルも異なるためです。しかし、入門向けを読んだ後に、その著者の見解と自分の見解に違和感を覚えた場合や、時代背景による異なる見解を得たい場合は「別の著者の入門向け」や「古い中古の入門向け」の参考書を購入することもあります。このように複数の情報源は、それぞれのバイアスを把握することに役立ちます。

## 人から学ぶ場合

人から学ぶ場合も、複数の人から学ぶことでバイアスを把握することができます。例えば「前にこのようにしたから、今回も同じようにして。簡単だから。」といった依頼を受けることがあります。仕事の学習の機会として大切ですが、依頼側は普段行っていることや、滞りなく対処できたことについては詳細な説明を省略する傾向があります。こうした状況は生存者バイアスが関与しており、失敗の要因を見過ごしていることがあります。

- 当時の関係者の感情や関係性
- 依頼者自身の経験値
- 依頼者のポジション（社内的立ち位置、プロジェクト内の役割など）

これらは、設計書・議事録・過去のメールなどからは読み取りにくいものです。このようなときは、依頼内容と同じ業務を行っていたメンバーや当時関わっていたメンバーなどに尋ねることが有用です。また、依頼者本人に不明点を詳細に尋ねることも有用です。同じ人物であっても、生存者バイアスを認識してもらうことにより、異なる情報源と捉えることができます。

本節では「参考書」と「人」という媒体の観点でしたが、私たちが情報源を持つ際には、以下のことに気をつける必要があります。

- 2つ以上の異なる媒体の情報源を持てているか
- 2つ以上の同じ媒体の情報源を持てているか
- それぞれのバイアスを客観的に評価できているか

ほとんどの情報には何らかのバイアスが隠れています。