テクニカルライティング

読み手につたわる文章

mochikoAsTech　著

2024-05-25 版　　mochikoAsTech 発行

はじめに

2024 年 5 月 mochikoAsTech

この本を手に取ってくださったあなた、こんにちは、あるいははじめまして。『読み手につたわる文章 - テクニカルライティング』の筆者、mochikoAsTech です。

突然ですが……私たちは、毎日のようにテキストで誰かに何かを説明したり、説明してもらったりしています。

- Slack で他部署の人に仕様の疑問点をつたえて質問する
- 業務でつまづいている部分を日報に書いて上司に報告する
- ローカルで開発環境を作る手順を社内 Wiki に書いておく
- 全社イベントの日時と参加方法をメールで知らせる
- 家族に LINE で今週末の旅行スケジュールを送る
- 子供に買うものリストを見せておつかいを頼む

こんなふうに日常生活の中で何かをつたえる「手段」としての文章は、詩や小説といった文学作品の文章とは区別して**実用文**と呼ばれています。**実用**的な**文**章なので実用文ですね。

そしてこの実用文を書くときに、つたえたかったことを分かりやすく書くための技術として**テクニカルライティング**というものがあり、そのテクニカルライティングを専門とする**テクニカルライター**という職業もあります。[*1]

テクニカルライターは、もともとは家電や電化製品の取扱説明書を書く人たちでした。ですが近年はそこから広がって、IT 企業で技術ドキュメントや API リファレンスなどを書く専門職としてテクニカルライターが採用されるようになっています。

筆者はもともと Web 制作会社のインフラエンジニアでしたが、2019 年からは IT 企業でテクニカルライターをしています。転職をした当時はテクニカルライターという職業があること自体が業界の中でもほとんど知られていませんでした。そこから約

[*1] エンジニアリングの技術を学んでシステムを開発する人たちが「エンジニア」であるように、テクニカルライティングの技術を学んで実用文を分かりやすく書く人たちは「テクニカルライター」です。テクニカルライターは、厚生労働省の職業情報提供サイトにも載っています。`https://shigoto.mhlw.go.jp/User/Occupation/Detail/358`

5 年が経ってテクニカルライターを募集している IT 企業が段々と増えてきたことを感じます。

本書はテクニカルライターである筆者が、テクニカルライティング技術の中でも普段から特に気をつけているいくつかのコツを紹介するものです。「文章の書き方」を解説した書籍でよく見かける文法的な話だけではなく、実際に書いていて悩みがちな部分にスポットを当てて、できるだけ実践的な内容を紹介しています。[*2]

本書が「もっとうまく書けるようになりたい」と感じている方の一助となれれば嬉しく思います。

想定する読者層

本書は、こんな人に向けて書かれています。

- 文章がうまく書けるようになりたい人
- テキストでのコミュニケーションで誤解されることが多くて困っている人
- 相手の理解度にあわせた説明ができるようになりたい人
- 技術書や技術記事を書いている人
- 技術書の翻訳をしている人
- 英語ができないけど英語でドキュメントを書いている人
- ちゃんとドキュメントを残したい人
- ごちゃついた情報をスッキリ整理したい人

マッチしない読者層

本書は、こんな人が読むと恐らく「not for me だった…（私向けじゃなかった）」となります。

- 魅力的な小説や詩文を書けるようになりたい人
- 広告やキャッチコピーが書けるようになりたい人
- 法務文書を書くときのコツが知りたい人

[*2] なお本書には、2019 年 4 月に発刊した「技術をつたえるテクニック ～分かりやすい書き方・話し方～」の内容を一部改訂して再掲したものが含まれています。

本書のゴール

本書を読み終わると、あなたはこのような状態になっています。

- 読む前よりも実用文がうまく書ける
- テキストコミュニケーションで相手の誤解を生まなくなる
- 悩みすぎて筆が進まない状態から早めに脱出できる
- レビュイーが受け止めやすい形で文章のレビューができる
- 英語ができないのに英語を書かなければいけないときになんとか頑張れる

免責事項

本書に記載された社名、製品名およびサービス名は、各社の登録商標または商標です。

本書に記載されている内容は筆者の所属する組織の公式見解ではありません。

また本書はできるだけ正確を期すように努めましたが、筆者が内容を保証するものではありません。よって本書の記載内容に基づいて読者が行なった行為、及び読者が被った損害について筆者は何ら責任を負うものではありません。

不正確あるいは誤認と思われる箇所がありましたら、必要に応じて適宜改訂を行ないますので GitHub の Issue や Pull Request で筆者までお知らせいただけますと幸いです。

`https://github.com/mochikoAsTech/technical-writing-book`

目次

第１章

テクニカルライティングをはじめる前に

テクニカルライティングの具体的な手法を学ぶ前に、まずは「分かりやすい文章を書く」とはどういうことなのかを一緒に考えてみましょう。

1.1 テクニカルライティングとは

「はじめに」に書いたとおり、テクニカルライティングとは**実用文を分かりやすく書くための技術**です。

ですが日本で生まれ育った人のほとんどは、誰でもある程度の日本語は書けるはずです。誰にでもできる「文章を書くこと」と「テクニカルライティング」には、いったい何の違いがあるのでしょうか？

1.1.1 「文章を書くこと」と「テクニカルライティング」の違い

文章を書くこととテクニカルライティングの違いを知るため、まずは**実用文とは何なのか**を掘り下げていきましょう。実用文は、以下のような特徴[*1]を備えています。

- 対象となる物事について説明している
- 論理的である
- 誤解を生まない
- 簡潔で明快である
- 読み手の行動を促す
- 成果物が文章である

つまり実用文とは、**対象となる「何か」について論理的かつ簡潔に説明をしていてそれを読むことで何をどうすればいいのかが分かるような文章**ということですね。実用文では、文学作品とは違って「流れるような文体」や「人の心を揺さぶる表現」は必要ありません。

学生時代に宿題の読書感想文で 400 字詰めの原稿用紙 2 枚以上を求められ、大して書くこともないのに長く引き延ばそうとして苦しんだ経験がある人も多いと思います。ですが実用文では「長ければ長いほどいい」という尺度は存在しません。読んで得られる情報が同じなら端的な文章の方がいいからです。

[*1] 参考：日本語スタイルガイド（第 3 版）の「1.3.2 実用文の特徴」 https://jtca.org/learn-tc/publication/guide_jsg/

実用文は詩や小説とは異なる、という話は理解しやすいと思います。その一方で、一見すると実用文のようですが次のようにテクニカルライティングの技術だけでは書けないものもあります。

- 法務文書
  - 特殊な専門性に基づいて書かれる法務文書は、優先事項がテクニカルライティングとは異なります。法令遵守が最優先であり、読みやすさや簡潔さはさほど重視されません。
- 広告やキャッチコピー
  - 読み手に強烈な印象を残して購買に繋げることを重視するため、迂遠で難解な言い回しや抽象的な表現が正解になり得ます。

テクニカルライティングという名前だけを聞くと、なんとなく「利用規約をレビューする」「製品のよさをつたえるキャッチーな見出しを書く」といったことにも使えそうな気がしてしまいますが、これらのように目指すべき成果物が異なるものもあります。テクニカルライティングはあくまで**実用文を分かりやすく書く**ための技術であり、ありとあらゆる場面でこの書き方が常に正解という訳ではありません。

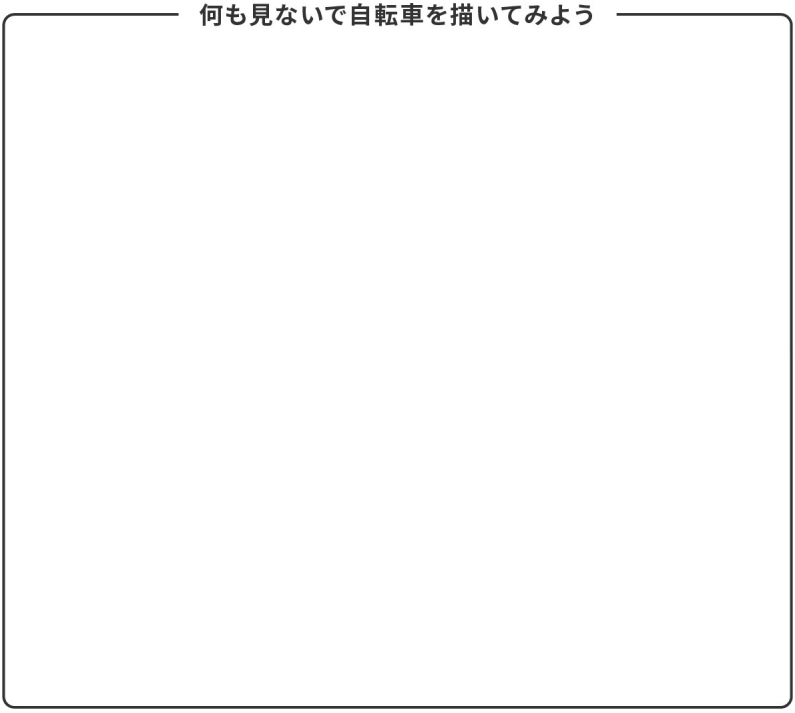
1.2 「知らない」と書けない

さて、話は変わりますが「自転車」を知っていますか？

何を唐突にと思われるかもしれませんが、ここからちょっと「自転車を描く」という体験にお付き合いください。

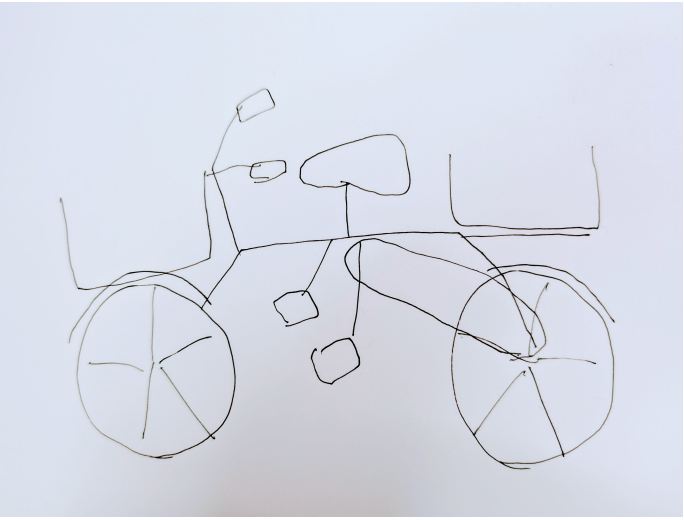
1.2.1 自転車の絵を描いてみよう

では自転車の絵を描いてみましょう。あなたがよく知っている自転車の姿を頭に思い浮かべて、下手でもいいのでペンを手に取ってやってみてください。（図 1.1）



▲図 1.1　自転車の絵を描いてみよう

筆者もあなたと一緒に自転車を描いてみます。自転車にはタイヤがありますよね……大きいタイヤが 2 つ……そしてその 2 つの間になんか太いパイプみたいなのがあったはず……サドルもありますね、座るところです。あとなんか漕ぐところがある……ペダル……？ あれ、ペダル？ ペダルってどうやって回していた？ 回る機構が必要なはず……どうなってましたっけ？ チェーン？ とタイヤが繋がってるはずですよね。チェーンって前輪と後輪どっちと繋がってましたっけ？ 後輪かな……あとハンドル！ ハンドルがあります！ カゴもあります……カゴってどこについてた？ あー、あとなんかタイヤの上に泥よけがあったかもしれません。我が家の自転車は前だけじゃなく後ろにもカゴが……あれ？ 自転車ってこんなでしたっけ？ なんか変だな。自転車って……どんな形でしたっけ？？（図 1.2）



▲図 1.2　筆者が描いた変な生き物みたいな自転車の絵

そしてこんな感じの自転車の絵が生まれました。うまく描けましたか？ 筆者は全然うまく描けませんでした……。

もともと自転車に詳しい人を除いて、恐らくはほとんどの人はこんな感じの「えー、自転車ってこんなだっけ？」という絵が出来上がったことと思います。

では正しい自転車の絵を見てみましょう。（図 1.3）

▲図 1.3　正しい自転車の絵

見慣れた自転車の姿ですね。私たちは「自転車がどんな見た目で、どんな乗り物かなんて知っている」と思っていたはずです。でも実際は、自転車がどんな構造でどういう理屈でできているのかという知識がないので、自転車の絵は思っていたよりうまく描けませんでした。

1.2.2　自転車を「知らない」と描けない

少し自転車の構造について解説していきます。

まず、自転車の骨組みは「ダイヤモンドフレーム」とも呼ばれており、どんな方向からの力に対しても剛性が高い、かつ低重心で操縦しやすい構造にするため、三角形を 2 つ組み合わせた斜めの菱形のような形をしています。このことを知っていれば、まず**中心部に菱形の骨組み**が描けます。

次に自転車に乗っているときの自分の様子を思い浮かべてみましょう。通常、あなたが椅子に座ったとき、膝はお尻の位置よりも前にあるでしょうか？ それともお尻の真下にあるでしょうか？ 人体の構造上、椅子に座ったとき、膝は 90 度に曲がった状態でお尻よりも前方にあるはずです。このことから、ペダルの根元にある丸い部分、**フロントギアはサドルよりも位置が少し前に来る**ということが分かります。

またハンドルと前輪は、フロントフォーク[*2]いう部品によって繋がっています。フロントフォークは前輪の中心部から真上に伸びるのではなく、乗っている人が舵取りをしやすいようサドル側に向かって少し斜めに生えています。このことを知っていると、**前輪からハンドルに向かって斜めに伸びる**フロントフォークが描けます

舵取りをする前輪は左右に動くので、そちらにチェーンを繋ぐと自転車の構造が複雑になってしまいます。そのためペダルを漕ぐ力は後輪に繋がる、つまり自転車は後輪駆動になっています。これを理解していると、**チェーンは前輪ではなく後輪と繋げて描くべき**だと分かります。

こんなふうに、自転車という対象物に対する知識があれば、パーツの位置をすべて暗記していなくても自ずと正しい自転車の絵が描けるようになります。**必要なものは絵心やセンスではなく知識**なのです。逆に言えば、対象物を知らないと上手な絵は描けません。

そして自分が対象物を思っていたより「知らない」ということに、我々は気付いていないことが多いのです。上手な絵は、対象物のことを知らないと描けません。それと同じように、分かりやすい実用文は、文章力のあるなし以前に**対象物をよく知らないと書けない**のです。

1.2.3 うまく書けないときに足りないのは文章力か技術力か

「割と知っている」「結構分かっている」はずの技術について本やブログを書こうとしたとき、書く前は簡単にできそうな気がするのに、実際にはじめてみるとなぜか書けない……と悩んだことがありませんか？ 既に使ったことがあり、分かっている技術であっても、理解の解像度が高くないと筆はなかなか進みません。

たとえば「ハッシュドビーフ作りの手順書を作ってください」という依頼があったとして、同じ「ハッシュドビーフを知っている人」でも次のように、理解度によって手順書をすらすら書けるかどうかや、書かれた手順の分かりやすさはかなり違ってくるはずです。

- 食べたことがあるけどそもそも自炊はしない、という理解度の人
- 食べたことがあるけど作ったことはなくて、「要は肉と野菜とルゥを買って煮ればいいんだろうな」という理解度の人
- 食べたことも作ったこともあり、「牛肉とタマネギときのこを買って、切って炒めて煮込んでルゥを入れる料理だ。赤ワインを入れることもある。こびりつくから後片付けのときは油汚れをお湯でよく流して洗わないといけないな」という理解度の人

[*2] fork は分岐という意味で、フロントフォークは名前のとおり前輪のタイヤを挟んで二股に分かれています。Git でブランチを分岐させる fork や、食器のフォークと同じフォークですね。

「分かっているはずの技術なのに、自分は言葉にするのが下手でなかなかうまくつたえられない」そんなとき実は不足しているのは出力の部分ではなく、入力や理解の部分なのかもしれません。「書こうとするとなぜか筆が進まず、頑張って書いてもふわっとした説明になってしまう……」と思ったら、そのもやもやっとした部分だけ理解の解像度が低くなっていることを疑って、改めて対象の理解に努めてみましょう。

私たちは対象物について「分かった！」の山と、「何も分かっていなかった……」の谷を越えて、少しずつ理解していきます。まずは自分が**対象物を知っているつもりでも意外と知らない**こと、そして**知らないものについて分かりやすい文章は書けない**ことを最初に知っておいてください。知っているつもりだけど実際は知らないままで体裁の整ったきれいな文章[*3]を書いても、それは分かりやすい文章にはなりません。

美味しい料理を作るには、材料がなければいけません。うまく料理が作れないとしたら、腕より前に材料が足りていない可能性があります。出力するためには先に入力が必要であり、そして材料が不足していることに気付けるのは大抵キッチンに立って料理を作り始めてからです。「書けると思ったのに……全然書けない……何も分かっていなかった……」という挫折はある意味必要なステップであり、これがなければ対象を理解した分かりやすい説明には辿りつけないのです。

1.3 まずはいっぱい読んでいっぱい書こう

「もっと文章をうまく書けるようになりたいです！ どんなことに気をつけて書いたら、文章力が上がりますか？」という相談を受けることがありますが、まだ経験の浅い方であれば細かいライティングのテクニックについて学ぶより先に、まずは**なんでもいいからいっぱい読む**といいと思います。

本屋さんで買える技術書も、インターネット上にある公式リファレンスも、個人の技術ブログや Zenn や Qiita も、いっぱいいっぱい読むと「この説明分かりやすいな」や「この言い方だと自分みたいな初心者には分かりにくいな」のように、参考にしたいお手本のような表現も、使わない方がいいと感じる表現もいっぱい自分の中に溜まっていきます。

レゴブロック[*4]と同じで、少ないパーツで何か作れと言われると難易度が高くなります。「この言い回し」とか「こういう表現」とか「こんな説明の仕方」というようないいパーツ、あるいは「こういう構成」や「こんな順番」といったいい組み合わせの例が自分の中にいっぱいあると、それだけ「いい文章」が組み立てやすくなります。

[*3] 「Amazon CR は、わずかなアクションで素早く安全に喫食可能なフルマネージド統合サービスです。CR を使用すればホットスタンバイ状態の H2O あるいは Raw として麺、醤油または味噌、また葱、肉などを単一のフローとして提供できます。」と言われて、誰がカップ麺のことだと分かるのだろうか……。 https://www.itmedia.co.jp/news/articles/2205/10/news123.html

[*4] レゴは LEGO Juris A/S の登録商標です。

そしてたくさん読んでパーツを集めるだけでなく、その溜め込んだパーツを実際に使ってどんどん組み立てて作ってみることが上達への近道です。つまり実用文をいっぱい書いてみましょう。書くことで「あ、こういう系統のパーツが全然足りてない！」ということに気付いて、また読む作業に対して意欲的になれます。

文章を書くことについてまだ自信がないのであれば、筆者からのアドバイスは「細かいことは気にせず、とにかくいっぱい読んでいっぱい書くといいよ」です。

1.4 完璧主義よりも完成主義

そしていっぱい書くときに大事なのは、「完成させること」です。せっかく色々書いたとしても、中途半端な書きかけがブログの下書きにいっぱい溜まっていたり、思いついた構想ばかりがメモ書きとして積まれていたり、という結局誰にも読んでもらえない状態ではうまくなりません。

完成度を求めた結果、いつまでも出来上がらない、完成させられない完璧主義はやめましょう。完成度にこだわりすぎず、**とにかく完成させて世に出すことを優先**しましょう。一度できあがって人目に触れる場所に出さないと、その次の改善するというステップにも進めません。

そのためには先に〆切を設定することが重要です。〆切もないのに何かを完成させられるほど我々の精神は成熟していません。あそこもここもおかしいし、本当は直したいし、もっとこういうことにも触れたかった、最初の構想と違って何もかも全然足りてないんだけど……一旦これを完成とする！ と開き直るためには、自分では動かせない〆切が必要なのです。技術書典のようなイベントに申し込んだり、自社の技術ブログで「こんなことを書きたい」と手を挙げて、ずるずると先延ばしにできないような状況を作りましょう。

それから**「間違ったことを書くのは怖いので○○についてもっと詳しくなったら書こう」と思っていると一生書けません**。なぜなら書きはじめてようやく「あ、この辺の知識が抜けている」という不足に気付くからです。人間は書くことで初めて「○○について詳しくなる」という階段を上り始めます。それに何かを誤解しているとき、人はその事実を自覚できません。目に見える形で自分の理解を外に示し、誰かに間違いを指摘してもらうことで、はじめて「自分が誤解していた」ということに気付けるのです。

ビルの 10 階に行きたいのなら、あなたに出来ることはまず 1 階から 2 階への階段を上り始めることです。「2 階とか 3 階とかそんな低い場所には居られない。俺はもっと高みに……10 階に行きたいんだ！」と意識の高いことを言われても、ヘリコプターでもない限り 2 階や 3 階を経ずに 10 階にはたどり着けません。それと同じで、100 点満点の完璧な文章が書ける自分になるためには、まずは 10 点や 20 点のひどい出来でも文章を何度も完成させるステップを踏まなければなりません。ただの一度も

三振をしたことがないままプロ野球選手になった人はいないはずです。

最初に高すぎる目標を立てて、「無駄なことや遠回りは極力したくない。このクオリティが出せるまでは自分は何も書かないし、書いたものも世に出さない！」と思っているといつまでも何も書けず、うまくもなりません。

まずは〆切を定めて、完成させて、世に出しましょう。

【コラム】どうやったらテクニカルライターになれますか？

「いまは他の仕事をしているんですが、書くことにも興味があって……IT 企業でテクニカルライターになるにはどうしたらいいですか？」という質問をいただくことがあります。そもそもテクニカルライターというポジション自体が職業としてまだあまり認知されていないので、そんな中で存在を知ってもらい、さらに「なりたい」と思ってもらえるのは大変有難いことです。

書いたものを見せてもらうのがいちばん早いので、そんなときは「過去に書いたドキュメントやブログなどはありますか？ Zenn でも Qiita でもなんでもいいです」と聞くようにしています。これは「プロ野球選手になるにはどうしたら？」と聞かれたときに、普段の練習メニューを教えてもらったり、過去の練習試合の動画を見せてもらったりするような感覚です。

そういうときに「特に何か書いてはいないので、まだ見せられるようなものは何もなくて……」と言われると、お気持ちは分かるものの、正直こちらから言えることがほとんどなくなってしまいます。

テクニカルライターになると、寝ても覚めても書くことに向き合う生活になります。筆者がこの職業を知ったときは「ドキュメントを書いて給料がもらえる？ そんな夢のような仕事がこの世に？」と思いましたが、ある人にとっては夢のように楽しいことが、他の人にとっては信じられないような苦行だったりします。好きな技術について頼まれてもいないのにあれこれ書いてしまう、という人にとってはテクニカルライターはきっと天職です。どんなことでもそうですが、長く楽しく続けられるのは「○○に興味があるのでこれからやろう」と思っている人ではなく、「面白そうだから○○をやってみた」と言える人です。[*5]

[*5] 大学教員８年目やってるとワナビーとモノづくり好きの区別がつくようになってくる→「へえ，○○がやりたくて大学に入ってきたんだ，でなんで今まではやってないの？」（次週）「え，どうして今週できなかったの？」｜落合陽一 https://note.com/ochyai/n/n781fb2209af4

第 2 章

エンジニアのためのテクニカルライティング

仕様書、設計書、手順書、オンボーディング資料、ドキュメント、技術ブログ、技術書など、エンジニアが技術に関する文章を書く場面はたくさんあります。分かりやすい文章を書くために、筆者が日々実践していることを紹介します。

2.1 必要ない人に無駄に文章を読ませない

小説であれば、読みはじめてすぐに「もういいや」と読むのをやめてしまうつまらない話よりも、最後の 1 文字まで読者を惹きつけて離さない面白い話の方がいいはずです。ですが実用文においては、読んだ人が「私は何をどうすればいいのか」という答えを見つけて、やるべき作業に早く進めた方がいいので、**最後まで読んでもらえないのはいいこと**なのです。

2.1.1 読者層を決めてから書こう

万人に最適な文章というものはありません。読む人が変われば、「分かりやすい文章」も変わってきます。

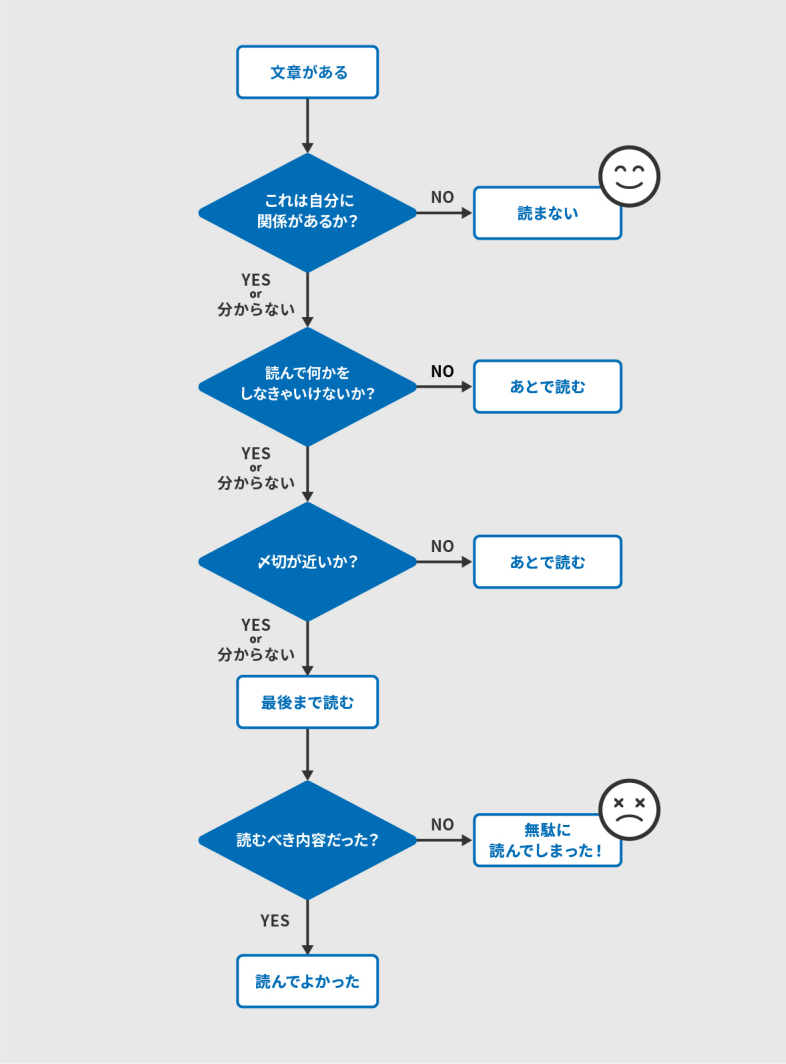
たとえば DNS に関する技術書を書くとしても、対象となる読者層が「インターネット？ ほぼ使ってないです。インスタは使ってますけど」というレベルの大学生なのか、それとも「A レコードは登録したことあるけどフルリゾルバは知らないです」というレベルのエンジニアなのかによって、書くべき内容や説明方法は大きく異なります。

あなたが**これから書く文章は誰に向けたものなのか**を最初にしっかり決めておかないと、のちのち「どこまでさかのぼって説明しないとだめなんだ……？」と頭を抱えることになったり、あるいは読者に「こんな簡単なことはもう知っている。もっと踏み込んだ内容が読めると期待していたのに」と不満を持たれたりします。

どれだけ美味しいめんつゆでも、麦茶だと思って飲んだ人からすると吹き出すくらいひどい味に感じられます。文章を読み終わったときに、文句を言いたくなったり低評価を付けたくなったりするいちばんの動機は「思っていたものと違った」です。読み終わってから残念な思いをさせないよう、書く前に想定の読者層をはっきりさせておきましょう。

2.1.2 関係ない人を早めにふるい落とそう

100 人の読み手がいて、そこに書いてある情報を必要としている人はそのうち 1 人しか居なかった場合に、100 人が最後まで読んでしまうと相当な労力が無駄になります。たとえばマネージャ以上の管理職に対して組織の目標設定を促すお知らせがあった場合、最初に「これはマネージャ以上の管理職に向けたお知らせです」と書いてあれば、その時点で管理職ではないほとんどの人が離脱できます。（図 2.1）



▲図 2.1　読む読まないの離脱フロー

一通り読んだ後で、ようやく自分には関係のない話だと気付いたときの「関係ないなら早くそう言ってよ！」という徒労感には誰しも覚えがあるはずです。このまま読み進むべきか、自分には関係ないので読まなくていいのかを判断できる情報を早い段

階で提示するようにしましょう。

みんなの時間を無駄に使わないことはコスト削減でもあります。組織全体の「読む」労力ができるだけ少なくなるよう、早めに**できるだけたくさんの人が該当する分岐**で読み手をふるい落としましょう。

2.1.3 想定する読者層とゴールを明示しておこう

関係ない人を早めに振い落として残念なミスマッチを防ぐ簡単な方法は、本文の前に「想定する読者層」と「ゴール」を書いておくことです。たとえば筆者が以前書いた「SSL をはじめよう」という書籍では、想定する読者層を次のように定義していました。

> 本書は、こんな人に向けて書かれています。
>
> * よく分からないままネットの手順通りにSSLを設定している人
> * 「サイトをHTTPS化したいな」と思っている人
> * 証明書の購入や設置の流れがいまいち分かっていない人
> * SSLとTLSの関係性がよく分からない人
> * SSL証明書が一体何を証明しているのか知らない人
> * これからシステムやプログラミングを学ぼうと思っている新人
> * ウェブ系で開発や運用をしているアプリケーションエンジニア
> * 「インフラがよく分からないこと」にコンプレックスのある人

想定読者を明示しておくことで、ここに当てはまらない人は本文を読む前に「自分が期待している内容ではないかもしれない。読もうかと思ったけどやめておこう」と判断できます。ここでさらにもう一歩踏み込んで「マッチしない読者層」も書いておくと、「あ、これは自分向けではないんだな」に気付いて、よりミスマッチが防ぎやすくなります。

> 本書は、こんな人が読むと恐らく「not for meだった… (私向けじゃなかった)」となります。
>
> * SSL/TLSの通信をC言語で実装したい人
> * 「プロフェッショナルSSL/TLS」を読んで完全に理解できた人

また「SSL をはじめよう」ではゴールを次のように定義していました。

> 本書を読み終わると、あなたはこのような状態になっています。
>
> * SSL証明書がどんな役割を果たしているのか説明できる

```
* 証明書を買うときの手順が分かっている
* 意図せず「保護されていない通信」と表示されてしまったときの対処法が分かる
* 障害が起きたときに原因を調査できる
* 読む前よりSSLが好きになっている
* SSL/TLSと併記されている「TLS」の意味が分かっている
```

このようにゴールを書いておくことで、読み手は「読むことで何が得られるのか」を事前に把握できます。ざっと概要だけ知りたいのか、手を動かして実践的な知識を得たいのか、文章を読む目的は人によって異なります。

最初に「想定する読者層」および「マッチしない読者層」、そして「ゴール」を書いておくことで、不幸なミスマッチをできるだけ防ぎましょう。これらを書いておくと、著者自身が方向性を見失いかけたときに「これって誰に向けた文章なんだっけ？」「読み終わったらどうなってて欲しいんだっけ？」と振り返る拠り所にもなります。

2.1.4 いつまでに何をしてほしいのかを書こう

ある日、会社であなたに「情報資産の棚卸しのお願い」というお知らせが届いたとしましょう。

お知らせには棚卸しの対象となる資産や、自分が会社から貸与されている資産の一覧を見る方法などが詳しく書いてありますが、結局いつまでに何をすればいいのかや、そもそも自分がこの棚卸しという作業をしなければいけない対象者なのか否かはまったく分かりません。色々と人に聞いたり調べたりした結果、もうすぐ年１回の棚卸しの時期がくるので事前告知として概要を先出ししていただけで、現時点はやるべきことやできることは何もない、ということが分かり、あなたには徒労感だけが残りました……。

仕事をしていると、こんなお知らせは実際によくありますよね。文章を書くときは、これを誰に読んでもらって、いつまでに何をして欲しいのかを最初の方に書きましょう。

このお知らせが届いた人全員に今すぐ何かをしてほしいのか、それともリンクを踏んでもらって貸与されている資産があった人にだけ何かをしてほしいのか、はたまた今やれることは何もないから時間のある人に事前告知として把握しておいてほしいのか、いったいこの文章を「どれくらいの優先度で読めばいいのか」を先に説明してあげましょう。

何の食べ物だか言わずにいきなり「食べて！ ほら食べて！」とスプーンを差し出されると、「え、怖い。なになになに？」となって、とても素直に口を開く気にはなれませんし、食べたところで猜疑心で味もよく分かりません。そんなときは「初めて作ったプリンが思いのほか美味しくできたので一口食べて感想を教えてほしい」というように、どういう意図で読み手に何をして欲しいと思っているのかを先に説明して

あげる必要があります。

文章も同じで「読んで！ さあ読んで！」と要求する前に、これを読んで**いつまでに何をして欲しいのかを提示してあげる**必要があります。

2.2 文書構造や文章量が適切だと分かりやすい

文章は、文章そのものの読みやすさに加えて、読む順番や量によっても分かりやすさが変わってきます。読み手に合わせた適切な文書構造や文章量にしましょう。

2.2.1 大枠からはじめて段々細かくしていこう

文章で何かを説明するときには、**先に大枠を理解してもらい、それから段々細かい内容にしていく**という順番を意識しましょう。

たとえばババ抜きの説明を書くのであれば、「トランプを使うゲーム」「2 人以上でやる」といった大枠（前提）をまず書きます。その上で、「同じ数字のカードは 2 枚を 1 組にして場に捨てられる」「カードの中に 1 枚だけジョーカーがある」「順番に隣の人のカードを引いていき、最後まで手元にジョーカーが残った人が負け」というゲームのルールを書きます。そして最後に「どれがジョーカーか悟られないようにポーカーフェイスを保つことが大事」のような補足情報を添えます。

この大枠からはじめるという順番を無視して、いきなり「ババ抜きから派生したゲームでジジ抜きもある」のような細かい話から書いてしまうと、読み手は全体像が分かっていないので混乱します。

また「大枠からはじめて段々細かく」という順番で書いておくことで、時間のない人は前半だけざっと読んで「あとは実際にやりながら覚えていこう」というように後半の補足情報を読み飛ばすことができます。

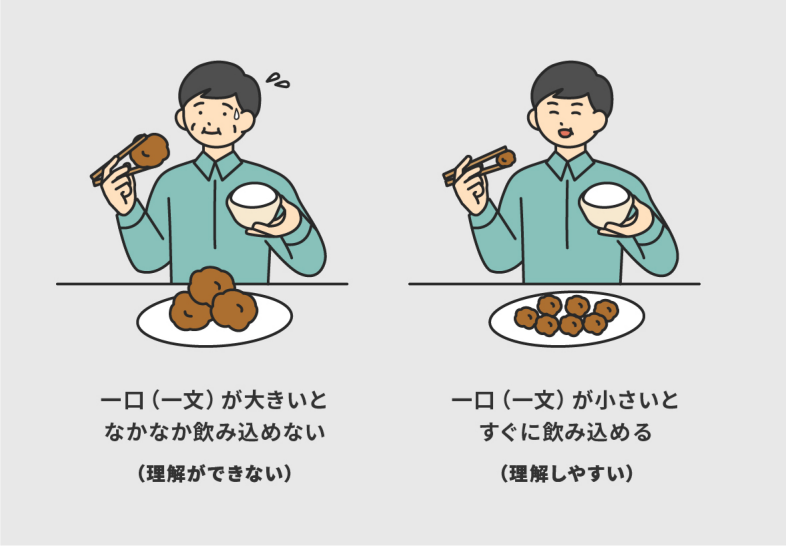
2.2.2 一文の長さは一口で食べられる量にしよう

この「長くて分かりにくい文章の例」を読んでみてください。

> まず、事前に確認しておきたいのですが先週金曜の16時の定例のミーティングで話に上がっていた方針で間違いはないと思うのですが、興味深い内容がありましたのでその際にもおつたえしたとおりその件につきまして理解したとおりの見解を共有させていただければという意図です。既存仕様と異なると考えられる理由としてはほとんどの機能がメインケースとサブケースを切り分けて動作していないので意図と異なる動きになりユーザーが混乱すると考えているため、リジェクトの可能性もあるので回避のために僭越ながら先方への回答は迅速に行ないたいと思っています。

我ながらすばらしくひどい例ができてしまいました。目が滑って全然頭に入ってきません。理解しようとして 3 回くらい読み返しましたが、読み返してもなお何を言っているのかさっぱり分かりません。

文章の「一文」は、食事の「一口（ひとくち）」と同じようなものです。句点（。）までの一文は、食事における一口分と同義なので、一口があまりに大きいと口の中いっぱいに食べ物を詰め込んでいるような状態になります。大きすぎる一口は、口の中で延々ともぐもぐ咀嚼する必要があり、いつまでたっても飲み込めません。（図 2.2）



▲図 2.2　一文を短くすれば理解しやすい

一文の長さは、一口で食べられるくらいの量（短さ）にしてください。具体的に言うと、以下のような接続詞が出てきたらそこを「です」や「ます」にして、文章を切りましょう。

- ～ですが
- ～ので
- ～し
- ～して
- ～ため

これくらいなら大丈夫かな、それとも長いだろうか……と迷ったときは、その一文

を声に出して読んでみましょう。前述のひどい例で試してみると分かりますが、長すぎる文章は一息で喋れません。途中で息継ぎをするはめになったら、その一文は長すぎると判断してください。

2.2.3 既知から未知に繋ごう

文章を書くとき、「大枠から詳細へ」の他に意識すべきもう 1 つの順番は「既知から未知へ」です。

技術ドキュメントを読んでいても、最初から知らない単語や知らない概念ばかりが次から次へと出てくると、「知らないことについて説明してくれているけど、その説明がまず分からない」という状態になります。まずは読み手がすでに知っていることから始めて、段々と知らないことに繋げていきましょう。

たとえばあなたが実家に帰ったとき、親から「こないだ、桜井さんちのりおちゃんが広場に行こうとして転んじゃって手当てしてあげたのよ」と言われても「まず誰だよ、桜井さん……どこだよ、広場って……」となり、肝心の「りおちゃんが転んだ」という情報が頭に入ってきません。

まずは「中学 3 年生のとき、あなたの同級生に桜井さんっていたでしょ？ 覚えてる？」という既知の情報から始めて、「ああ、桜井さんね。覚えてる」となったところで、「桜井さんって何年か前に結婚してお子さんも生まれたの。いま子供が夏休みだからって実家に帰ってきてるのよ」と新しい情報を分かりやすく提示し、最後に「お子さんの名前がりおちゃんって言うんだけど、その子がこの間、向かいにある公園の広場に行こうとしてうっかりうちの前で転んじゃったから手当てしたのよ」と繋げれば、情報がきちんと頭に入ってくるはずです。

未知の事柄の上にさらに新しい情報を重ねようとすると、ぐらぐらと不安定な積み木の上にさらに新しい積み木を載せようとしているような状態なので、知識が脳内で安定せずにどんがらがっしゃーんとすべて崩壊してしまいます。

まずは**知っていることからはじめて段々と知らないことへ**、認知の負荷は順を追って少しずつ高くしていきましょう。

2.2.4 1 つの段落では 1 つのことだけ書こう

人はみんな忘れる生き物です。2 時間観た映画がとても面白かったとしても、終わった後に主人公の名前が思い出せない、みたいなことはざらにあります。

しっかり分かってほしいからこそ、あれこれも色々書きたくなってしまうのですが、いま書いているこの段落でつたえたいことは何だったかを考えて、**1つの段落に書くことは1つだけ**に絞りましょう。言いたいことを絞り込みきれずに色々詰め込んでしまった文章を読むと、読み手は読み終わったときに「なんか色々書いてあったけ

ど、結局何が言いたいのかよく分からなかった……」という状態になります。

もし 1 つの段落で 2 つのことを混ぜて説明していたら、それは 2 つの段落に分ければいいだけです。

2.2.5 一度に把握できることは 7 つまで

コース料理の最後にデザートの案内をされたとき、「デザートは 10 種類ございます。チーズケーキ、ピスタチオとナッツのタルト、ガトーショコラ、梨とキャラメルのアイス、抹茶のティラミス、マンゴーのクレープ、ベリーのクラフティ、イチゴのブラマンジェ、メロンのカタラーナ、アールグレイと豆乳のシフォンからお選びいただけます」などと言われたら、どれにするか決めようにも全種類を把握できずに、「チーズケーキとガトーショコラと……アイスは……何と何でしたっけ？」と聞いてしまうはずです。

このように人間の短期記憶で覚えていられる情報には限りがあり、**一度に把握できるのは多くても7個**程度です。それ以上あると「いっぱいありすぎてよく分からない」という状態になるので、箇条書きや項目の数は 7 つ以内にしましょう。

数があまりに多い場合は、まず項目を階層化しましょう。前述のデザートの例なら「デザートは温かいものと冷たいものがそれぞれ 5 つずつあります」で一度分岐させてから、選んだ方の 5 種類を提示することで把握がしやすくなります。

2.2.6 最新のドキュメントが埋もれないにしよう

メンテナンスされていないものも含めて、ドキュメントがとにかく大量にあるという状態は、ドキュメントがまったくない状態よりも余計に読み手を混乱させることがあります。ドキュメントはあればあるほどいいというものではなく、メンテナンスを怠ればコードと同じようにドキュメントも負債化します。新しいものと古いものが混在しているアクセシビリティが低い状態では、折角書かれたドキュメントもその良さを発揮できません。

ドキュメントはできるだけ Git などでバージョン管理できる状態にして、「本当に必要なもの」と「古い情報だが取っておきたいもの」が同じ場所に雑然と置かれていることのないようにしましょう。

たとえば情報共有サービスの esa[*1]には、「アーカイブ」という機能があります。古い情報だが念のため取っておきたい記事はアーカイブしておくことで、その記事の検索順位を落とすことができます。

買ったばかりで状態のいい服と、穴が空いていたり毛玉がひどかったりで着られな

[*1] esa - 自律的なチームのための情報共有サービス `https://esa.io/`

い服がタンスにぐちゃぐちゃに詰まっていると、出掛ける前にさっと服を選ぶことができません。良い文章を書くことと同じくらい、その文章を**見つけやすくたどり着きやすい状態**にしておくことが大切です。

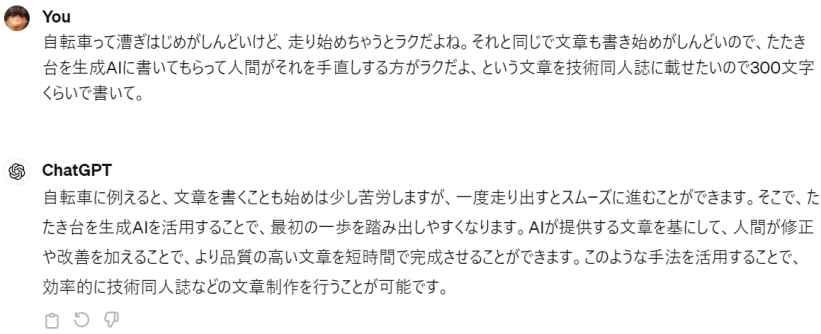
2.3 書く速度を速くするためにできること

書く速度が速いとそれだけ見直しやレビューに時間がかけられるので、最終的にできあがる文章の品質も上がります。「書くぞ！」と思ってから文章を書き上げるまでのスピードを速くするコツがいくつかあります。

2.3.1 書きはじめの遅さは生成 AI にサポートしてもらう

自転車は漕ぎはじめはペダルが重くてしんどいですが、速度が乗ってくれば少しの労力でぐいぐい進みます。文章も同じで、0 から書く部分がいちばん重労働で時間がかかります。大抵の人は 0 から作るよりも他人が作ったものにああだこうだ文句を付ける方が圧倒的に得意なので、書きはじめが遅い人はそこだけ生成 AI にやってもらう、という方法がお勧めです。

文章を書こうと思ってパソコンの前に座ったが、どうも書きあぐねている……というときは、雑な指示を ChatGPT に出したり、GitHub Copilot の提案に身を委ねたりして、低品質でいいので初案を作ってもらいましょう。（図 2.3）



▲図 2.3　雑な指示を出して ChatGPT に初案を書いてもらう

出来上がった初案を読んで「これ、私の言いたいこととはちょっと違うし、あれこれ間違っていてひどいな」と思ったらしめたものです。そうじゃなくてこうだよ、とサクサク手直しをしていきましょう。悩むばかりで 1 文字も進まないよりは、ずっと

速くあなたの文章ができあがるはずです。

2.3.2 長文を書きたいならまずは一文から

包丁すら握ったことがない人が、いきなりフレンチのフルコースを作ろうとすれば大抵は失敗します。それと同じで、短い文章すら書き慣れていないのにいきなり壮大なドキュメントを書こうとすると、うまくいかずにやたらと時間がかかってしまいます。

まずはトマトを切って塩を振っただけのものを毎日作ってみる。それがうまくできるようになったら、トマトときゅうりを切ってサラダにしてみる。それも慣れてきたらサラダとは別に肉を焼いてみたり、ご飯も炊いてみたりする。そこまできたらついに献立を考えて朝昼晩の食事を作ってみる、というように**意識してゆっくりと難易度を上げていく**必要があります。

高すぎる目標だけ立てて挫折してしまわないよう、次のようにまずは毎日一文を書くところからはじめて、それができたら一段落、次は一記事、最後に複数記事がまとまったドキュメントというように段階的にできるようになっていきましょう。

1. 句点（。）までの一文を書いてみる（50 文字以内）
2. 複数の文を繋げて一段落を書いてみる（150 文字程度）
3. 複数の段落で構成された一記事を書いてみる（1000 文字程度）
4. 複数の記事で構成されたドキュメントを書いてみる

2.3.3 全部並べていちばんよい一文を早く見つける

「ここの文章、いまいち分かりにくいから直したいんだけど、どうしようかな」と悩みながら書き足したり、消したり、少し言い回しを変えたり、前後を入れ替えたり……いやいややっぱりさっきの方がよかったかも、と元に戻したり……という流れに心当たりはありませんか？

こんなふうにいちばんよい一文を探してああでもないこうでもないと長時間悩んでしまう人には、**たくさんのパターンを並べて書いて、その中から最善案を見つける**という方法がお勧めです。

たとえば「メッセージ送信で文字数の上限が 100 文字から 200 文字に変更された」ということをつたえる文章を書きたいとき、初案として次のような一文を書いたとします。

```
メッセージ送信で文字数の上限が200文字に変わりました
```

初案を書いたあなたは、この初案だと「メッセージ送信」が「メッセージ」と「送信」という 2 つの名詞を組み合わせた合成語になってしまっているな、と気付きます。実際に「メッセージ送信」というボタンがある訳ではないので、ここはメッセージ送信じゃなくて助詞を入れた「メッセージの送信」にしよう、と思ったあなたは、次のようにさっと書き換えてみました。

```
メッセージの送信で文字数の上限が200文字に変わりました
```

これが長時間悩んでしまうダメなパターンです。書き換えてしまうと初案と第 2 案の比較ができません。まだ初案と第 2 案の 2 つくらいなら自分がどこを変えたのかや原型はなんだったのかを覚えていられますが、ここからあれこれ悩みながら書き足したり、消したり、少し言い回しを変えたり、前後を入れ替えたりしていると、「え、さっきいいなって思ったのはどれだったっけ？」と分からなくなってしまう未来が待っています。

いちばんよい一文を考えるときは、ヒストリーと差分が見えるように直前の案をコピーペーストしながら書き換えて、全部の案を並べていきましょう。このとき自分ひとりで考えるのもいいですが、誰かに付き合ってもらってオンラインミーティングで画面共有をしながら書いていくと、改善の螺旋階段を上がっていく速度が急激に速くなり、ものの 5 分くらいで「これにしよう！」といういちばんよい一文に辿りつけます。

```
1．メッセージ送信で文字数の上限が200文字に変わりました
2．メッセージの送信で文字数の上限が200文字に変わりました
3．メッセージを送信する際の文字数の上限が200文字に変わりました
4．メッセージとして送れる文字数の上限が200文字になりました
5．メッセージで送れる文字数の上限が200文字になりました
6．メッセージの送信で文字数の上限が200文字に緩和されました
7．メッセージの送信時に指定できる文字数の上限が200文字になりました
8．メッセージとして送信できる文字数の上限が200文字に変更されました
9．送信できるメッセージの文字数が最大100文字から200文字に変更されました
10．送信できるメッセージの文字数が最大100文字から200文字に変更されました
```

併せてそれぞれの案に数字を振っておくこともお勧めします。比較対象が 2 つくらいなら「上の方が分かりやすい」とか「私は下の方がいいと思う」で済みますが、3 つ以上になったときに「上からえーっと……4 つめのやつよりは、その下の下のやつがいいと思う」みたいになって混乱します。案を並べて書きながらそれぞれに数字を

振っておくことで、「4 がいいと思う」「4 よりは 6 の方が分かりやすい」のように、他の人につたえやすくなります。

【コラム】変わっていく語感を拾おう

いま、若い世代に「1 時間弱は何分ぐらいのことか？」と聞くと、「1 時間が 60 分で、1 時間とちょびっとだから 70 分くらい？」と答えるそうです。まさかと思って息子に聞いたら、たしかにこれとまったく同じように答えたのでちょっと驚きました。

その理屈でいくと 1 時間強は何分になるのかと聞いたところ、「1 時間とたくさんだから、85 分くらいとか？」らしいです。なるほど。同じ理屈で大さじ 1 杯弱についても「大さじ 1 杯プラスちょっと」だと思っていたとのこと。息子が作ってくれるだし巻き玉子が、いつも味が濃いめな理由が分かりました。

本来の意味はそれぞれつたえておいたけれど、こういう「口に出さずに脳内で起きている誤解」は周囲も気付かないので解くのが難しいものです。もちろん言葉の意味は辞書に載っていますが、その言葉から受け取る「こんな感じかな」という語感は、人や世代によって移り変わっていきます。

たとえば学校で先生に当てられて答えを言ったとき、途中で先生から「はい。結構です」と止められたら、「回答に満足しました。そこまでで十分ですよ」というプラスの意味で受け取るか、「もういいです。それ以上聞きたくありません」というマイナスの意味で受け取るかは、人や世代によってかなり異なると思います。

「誤解して受け取る側が悪いので、言葉の意味をちゃんと勉強してください」と憤る気持ちも分かりますが、誤解している人が 2 割、3 割と増えていってもなお、書き手が頑なにその存在を無視し続けるのはよくないと思っています。前述のようなケースであれば、誤解している人が若い世代に多いと分かった時点で、「1 時間弱というのは、1 時間より少し短いという意味です」というような補足を入れてあげるか、曖昧な言い方をやめて「45 分から 60 分」のように誤解されない説明にしましょう。

2.4 再利用しやすい文章にする

社内 Wiki で書いた文章を社外ブログに流用したり、登壇用に作った発表原稿をドキュメントで使ったり、文章は何かしらの形で再利用されることがあります。そんなときに再利用しやすい文章の書き方があります。

2.4.1 文章と修飾を分けて書こう

WordPress のような CMS や Microsoft Word などで書かれた文章は、文章そのものと「見出し」や「太字」といった修飾が分かちがたく一体化しているため、再利用する際の再加工に手間がかかります。

可能であれば Markdown[*2]や Re:VIEW[*3]などを使って、最初から**プレーンテキストで文章と修飾を分けて書いておく**ことで、別の環境に引っ越すときや文章を再利用するときに書き直す労力が少なくて済みます。

2.4.2 並列はナカグロ（・）で書かない

文章の中で並列を表そうとして「ホーム・検索・マイページ・ヘルプのタブは非表示にできません」のようにナカグロ（・）を使っていると、再利用されて箇条書きになったときに、次のような見た目になることがあります。

> 着せかえの設定時、以下の機能は非表示にしたり、見た目を変更したりできません。
>
> ・ホーム・検索・マイページ・ヘルプのタブ
> ・トレンドワード機能
> ・ID連携機能

並列を表す場合は最初から箇条書きにしておくか、文章の中で書く場合は読点（、）を使用することをお勧めします。

[*2] 文章を書くための軽量マークアップ言語のひとつ。オリジナルの Markdown の他に、拡張された Markdown 方言が存在する。たとえば GitHub のコメントなどで使える Markdown も方言の 1 つ。

[*3] Re:VIEW（リビュー）とは、Re:VIEW の記法に沿ってテキストの原稿を書き、コマンドをたたくと書籍として組版された PDF や EPUB が生まれてくる素晴らしいツール。本書も Re:VIEW で書かれています。詳しく知りたい方は「技術書をかこう！ ～はじめての Re:VIEW～ 第 3 版」という書籍がお勧めです。 https://techbooster.booth.pm/items/5231699

2.4.3 タイトルは「概要」か「○○の概要」か

たとえばShops APIというAPIのドキュメントを書くとき、概要のページのタイトルをただの「概要」にすべきか、それとも「Shops APIの概要」にすべきか、迷うことがあります。

概要のページはShops APIというカテゴリの下にあるので、ただの「概要」でも上位のカテゴリ名を見れば補完されて「ああ、Shops APIの概要なんだな」と分かります。それにいちいち全部のページのタイトルで「Shops APIの概要」や「Shops APIの開発ガイドライン」や「Shops APIの制限事項」というように同じ名前を連呼するのはいささか冗長にも感じます。

ですが、いちいち上位のカテゴリ名を見なければ、何の概要だか分からないというのは不便なものです。また、いまは概要のページがShops APIというカテゴリの下にあっても、今後ドキュメントの構造が変わってすべてのAPIが一目で把握できるように各APIの概要だけが1つの階層に集められてしまう可能性もあります。そのときにもしタイトルがただの「概要」になっていたら、すべてのタイトルを「○○の概要」に書き直さなければいけなくなります。

どちらも一長一短ではありますが、筆者はページのタイトルだけを見て分かるようにしてあげたいのと、他のドキュメントから当該ページへのリンクを張るときにリンクテキストが「概要」だけよりは「Shops APIの概要」になっていた方が分かりやすいと思うので、「Shops APIの概要」にする派です。

2.4.4 リンクテキストを「こちら」にしない

「詳しくはこちらをご覧ください」や「利用規約はこちら」のように、リンク先を表すリンクテキストを「こちら」にしているケースはよく見ます。

ですが、リンクテキストを「こちら」にしていると、再利用するためにコピーした文章をプレーンテキストとして貼り付けた結果、そもそもここでリンクしていた先はどこだったのかという情報が失われて、まったく分からなくなってしまうことがあります。

また「詳しくは[Shops APIのAPIリファレンス](/BankCodeAPI/reference/)をご覧ください」と書いてあれば、「Shops APIのAPIリファレンスにリンクしているはずなのに、リンク先が間違って金融機関コードAPIになっているようだ」と気付くことができますが、「詳しくは[こちら](/BankCodeAPI/reference/)をご覧ください」だと、そもそもどこにリンクすべきだったのかという仕様を把握している人以外はリンク先が間違っていることには気付けません。

読み手にとっても、リンクを実際に開いてみるまでどこに飛ばされるのかが分から

ないため、そのリンクを開くべきか開かなくてもよさそうかが判断できません。

リンクテキストは安易に「こちら」にせず、「詳しくは Shops API の開発ガイドをご覧ください」のように、リンク先のページやコンテンツの名前を書くようにしましょう。

2.5 技術文書に特有のコツ

技術に関する文章を書くときに特有のコツがいくつかあります。

2.5.1 正しい名前で呼ぼう

ソフトウェアやハードウェアの名称は、自分がなんとなく使っている通称や誤った表記ではなく、正しい名称で書くようにしましょう。(表 2.1)

▼表 2.1 通称や誤記ではなく正しい表記で書こう

通称や誤記	正しい名称
VSCode	Visual Studio Code
Github	GitHub
Word Press	WordPress
JAVAScript	JavaScript
iphone	iPhone

多少でも名前が間違っていると読者も混乱しますし、間違えられた側も決していい気分はしません[*4]。細かいことではありますが、名前が間違っていると「この人はこの技術をちゃんと理解しているのだろうか？」と思われてしまい、文章そのものに対する信頼度も下がります。特にスペースの有無や大文字小文字などは意識していても間違えやすいので、筆者は公式サイトや公式ドキュメントの表記をコピーペーストして使うようにしています。

また英数字の羅列だと覚えにくいけれど何の略なのか分かれば理解しやすくなる、という側面もありますので**はじめは正式名称で紹介して、以降は略称にする**という形もよいでしょう。読み方が分からずにひそかに悩んでしまう[*5]のも初心者あるあるで

[*4] 以前所属していた会社で MVP として壇上に呼ばれた際、社長に名前を間違えられて「表彰相手の名前くらいは把握しておいてもらえると嬉しい……」と思ったことがあります。相手に興味がなくてもいいのですが、それを悟らせても得るものは何もないので、せめて興味があるように見える最低限の準備は大事だなと思います。

[*5] k8s は Kubernetes の略でクバネティスと読むとか、nginx と書いてエンジンエックスと読むとか、誰かに教えてもらわないと筆者は想像もつかなかったです。密かに「んぎっくす…？」と思っていました。

すので、次のようにカタカナで読み仮名も添えるとなお親切です。

```
AWSではサーバはAmazon Elastic Compute Cloudの略で「EC2」(イーシーツー) と呼ばれて
います。
```

2.5.2 年月日や対象バージョンを書いておこう

書いたドキュメントは、本人が書いたことを忘れるくらい時間が経ってから突然参照されることがあります。その際、「いつ書かれたものか」という情報がないと、読み手は非常に古い情報をいま現在の仕様だと思って読んでしまう可能性がありますので、文章を書くときは必ず「その文章が書かれた年月日」を記載しておきましょう。

ブログであれば、その記事を投稿した年月日が自動で表示されるようにしておきましょう[*6]。技術書であれば奥付[*7]に書いておけばよいですが、それ以外に文中でも「今年の技術書典」や「5 月 26 日の技術書典」ではなく「2024 年 5 月 26 日 (日) の技術書典」のように、**数年経ってからその文章を読んでもいつのことを指しているのか分かるようにしておく**とさらによいでしょう。

またミドルウェアやソフトウェアであれば、**どのバージョンを対象とした内容なのか**も記載しておきましょう。

2.5.3 例示用の IP アドレスやドメイン名を使おう

たとえば「ブラウザで www.example.com を開くと、名前解決が行なわれてウェブサーバの 203.0.113.222 という IP アドレスが返ってきます」というように、技術の説明をしていると具体的な IP アドレスやドメイン名を書きたくなることがあります。このようなときは**例示用のIPアドレスやドメイン名を使いましょう**。

実はインターネットでは「例示やテストで使っていい IP アドレスやドメイン名」というものが定められています[*8]。

例として記載する URL、メールアドレスなどでは次のものをつかいましょう。

- 例示として使える IP アドレス

[*6] クラスメソッドさんの DevelopersIO は、1 年以上前の記事には「この記事は公開されてから 1 年以上経過しています。情報が古い可能性がありますので、ご注意ください。」という案内が表示されるところが素晴らしいなと思います。

[*7] 書籍や雑誌の巻末にある著者名・発行者・発行年月日などが書かれている部分。本書にもあります。

[*8] 例示用の IP アドレスは RFC5737、ドメイン名は RFC2606 や JPRS のサイトで確認できます。

  - 192.0.2.0/24（192.0.2.0〜192.0.2.255）
  - 198.51.100.0/24（198.51.100.0〜198.51.100.255）
  - 203.0.113.0/24（203.0.113.0〜203.0.113.255）
- 例示として使えるドメイン名
  - example.com
  - example.net
  - example.co.jp
  - example.jp

例示であっても自分の持ち物でない IP アドレスやドメイン名を勝手に使うことはトラブルの元になります[*9]。必ず例示用の IP アドレスやドメイン名を使いましょう。

2.5.4 リンクは「張る」ものか「貼る」ものか

リンクは「張る」ものでしょうか、それとも「貼る」ものでしょうか？[*10]

URL を誰かに教えてあげるためにコピーして貼り付けるときなら、「Slack のチャンネルに興味深い記事の URL を貼る」のように「貼る」を使うのがいいでしょう。これは他の人に「こういう記事があったよ」と知らせるために、URL の書かれた紙を壁に糊で貼って掲示しているイメージです。

一方、「商品一覧から詳細ページにリンクを張る」のような場合は「張る」を使います。これは蜘蛛がこちらからあちらへ糸を張るように、いま見ているページから別のページへと繋ぐ糸を張っているイメージです。

[*9] 実際にどんなトラブルになるのか？ は「DNS をはじめよう」という書籍の「4.4.4 <トラブル> test@test.co.jp を使って情報漏洩」で紹介しています。

[*10] リンクを「はる」は「張る」 – 毎日ことば plus https://salon.mainichi-kotoba.jp/archives/50189

第 3 章

英語を書いたり翻訳したり

英語が苦手な日本語話者（筆者のことです）が、なんとか頑張って英語をひねり出したときにやりがちな失敗を紹介します。

3.1 単語と単語の間にはスペースが 1 つ必要

たとえば `You can use this API to get weather information` という英文を見ると分かるように、単語と単語の間にはスペースが 1 つ入ります。英語が不得手な人でも、そこまではなんとなく認識できていると思います。

ですが、括弧やカンマやピリオドといった記号が入ると、この**単語と単語の間にはスペースが1つ入る**という原則を忘れがちです。たとえば、以下のような自己紹介文を書いてしまったことはありませんか？

```
mochiko(nickname)
Living in Japan(Tokyo)
Presented at Tech Seminar vol.22,vol.23
Loves : Flutter,Nuxt,Kubernetes
```

この自己紹介文は、どれも単語と単語の間にはスペースが 1 つ入るという原則を守れていません。重要なのは**記号はスペースの代わりにはならない**ということです。間違いに気付くため、括弧やカンマやピリオドといった記号を消してみましょう。

```
mochikonickname
Living in JapanTokyo
Presented at Tech Seminar vol22vol23
Loves  FlutterNuxtKubernetes
```

あちこちで単語が繋がってしまったり、逆にスペースが 2 つ続いてしまったりしています。では単語と単語の間に 1 つだけスペースを入れた上で記号を元に戻して、正しい英文に直してみましょう。

```
mochiko (nickname)
Living in Japan (Tokyo)
Presented at Tech Seminar vol. 22, vol. 23
Loves: Flutter, Nuxt, Kubernetes
```

できました！ 括弧書きの手前には半角スペースが入りますし、volume の略である `vol.` と数字の間や、区切り文字であるカンマの後ろにもスペースが入ります。逆に `Loves` とコロンの間にあったスペースはなくなりました。

これは日本語話者が英語を書いたときに非常にやりがちな失敗[*1]です。

3.2 単語や文の単位で翻訳すると失敗する

OSS のドキュメントなどで英語を日本語に訳すとき、前後の文脈や原文で言いたいことを汲まずに単語や文の単位で翻訳してしまうと、おかしな日本語になることがあります。たとえば、AWS が提供している日本語のドキュメント[*2]に、以前こんな一文がありました。

> インスタンスを削除するということは、実質的には、そのインスタンスを削除するということです。いったん終了したインスタンスに再接続することはできません。

削除するということは、実質的には削除するということ……進次郎構文[*3]みたいになっています。どうしたのでしょうか。

原文と思われる英語[*4]を確認してみると、どうやら Terminate と Delete という 2 つの英単語を、どちらも「削除」と訳したことで生まれてしまった悲しい日本語だったようです。

```
Terminating an instance effectively deletes it;
you can't reconnect to an instance after you've terminated it.
```

「インスタンス」とは AWS におけるサーバのことです[*5]。なので恐らく原文はこう言いたかったのだろうという意図を汲むと、「サーバの『終了』とは、単に電源を落とすシャットダウンではなく、サーバそのものを『削除』してしまうことを意味し

[*1] かくいう私もよくこの失敗をしていましたが、同僚が「日本人が英語でやりがちな失敗 / Common mistakes in English that Japanese people tend to make」というセッションで説明してくれてようやく理解できました。感謝！ https://www.youtube.com/watch?v=2nXUkXmFfZI&t=943s

[*2] チュートリアル: Amazon EC2 Linux インスタンスの開始方法 https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/EC2_GetStarted.html

[*3] 「今のままではいけないと思います。だからこそ、日本は今のままではいけないと思っている」のように前半と後半で同じことを繰り返す謎構文のこと。 https://dic.nicovideo.jp/a/%E9%80%B2%E6%AC%A1%E9%83%8E%E6%A7%8B%E6%96%87

[*4] Tutorial: Get started with Amazon EC2 Linux instances https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

[*5] AWS？ インスタンス？ その辺をもっと詳しく教えてくれ！ と思ったら「AWS をはじめよう 改訂第 2 版 ～AWS による環境構築を 1 から 10 まで～」という書籍をどうぞ。 https://booth.pm/ja/items/1032590

ます。そのため『終了』させたサーバには二度と接続できなくなります」という注意喚起の文章だったのではないかと思われます。

Terminate を英和辞典で引くと「終わらせる」「解除する」という意味が出てきます。「Terminating an instance」で「インスタンスを終わらせる」ということなら、確かに「インスタンスを削除する」と訳してもおかしくはありません。もう 1 つの Delete の意味も「削除する」「消す」なので、どちらも正しく訳した結果、「削除するということは、実質的には削除するということ」という文章が生まれてしまったのだと推察しています。

この注意喚起文によって何を食い止めたかったのか、という背景や意図を理解せず、単に単語や文の単位で「正しく」翻訳すると、こんなふうに実際は意味を為さない文が生まれてしまうことがあります。

3.3 前後の文脈やどこで使われるのかを知らずに翻訳はできない

とても簡単な英語を「正しく」翻訳しても、おかしくなってしまうことがあります。たとえば「Take Me Home」を「私を家に連れて行って」と訳すのは、英語がそこまで得意でない人が見ても恐らく正しいと思うはずです。

ですが、これがアプリでログアウトした後の画面に表示されていたらどうでしょう？（図 3.1）

▲図 3.1　ログアウト画面

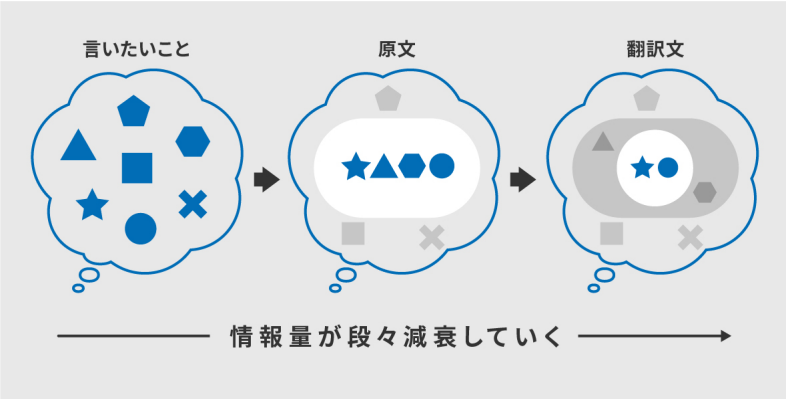
恐らく「Take Me Home」の訳は、「ホームに戻る」が適切だったはずです。その文章がどこに配置されるのか分からずに訳すと、やはり「正しい」のにとても違和感のある UI 文言になってしまうことがあります。[*6]

[*6] 実在した事例です。ただこういう翻訳を見ると、ひどい訳だと笑う前に、もはや日本が「丁寧に翻訳するコストをかけるほどの市場規模じゃない」と相手にされていない可能性を考えてしまいます。
https://twitter.com/mochikoAsTech/status/1779009150426767775

3.4　翻訳はもとの言いたいことに立ち返って考える

英語から日本語に翻訳するとき、あるいは英語から日本語に翻訳するときに、原文をしっかり読んでいても誤訳してしまうことがあります。

もともと原文を書いた人の頭の中に何か「言いたいこと」があって、そこから原文が生まれているので、原文になった時点で情報量は減っています。JPEG のように保存されるたび情報量は減衰していくので、元の「言いたいこと」に立ち返らずに、情報量の減った原文だけを読んで翻訳してしまうと、翻訳文はさらに情報量が減衰します。そして文章としての体を為すためにその減衰を補おうとして、もとの「言いたいこと」にはなかった想像や嘘が混ざることがあります。（図 3.2）



▲図 3.2　言いたいことを書いたり翻訳したりすると情報量が減衰していく

元々言いたかったことと、実際に発露した文章はイコールではなく、そこには必ず差分があります。翻訳するときは、原文を書いた人に意図や背景、元々何が言いたかったのかを確認して、できるだけ元の「言いたいこと」に立ち返って訳しましょう。

3.5 技術文書の翻訳に必須なのは英語力や文章力よりも技術力

前述のとおり、OSS のドキュメントや技術書などを英語から日本語に翻訳する場合、**元の「言いたいこと」や前後の文脈をどこまで汲めたかが翻訳文の分かりやすさに直結**します。そのため、こと IT の分野においては翻訳に必須な力は対象技術の知識、実際に触ったことがある深度の深い理解であって、日本語の文章力や、英語の文章力はその次だと筆者は思っています。もし技術力はあっても、英語が不得手だからとドキュメントや技術書の翻訳協力に尻込みをしていたら、ぜひ一歩踏み出してみてください。

翻訳する人は「文脈が分かっている」と非常に強いです。そもそも何の話か分かっているから、原文が多少説明不足でも元の「言いたいこと」を推察してピントの合った翻訳ができます。

ドキュメントや技術書を翻訳するとき、英語が読めても「何を言っているのか」が分からないと、文法があっているだけで正しいことを何もつたえない、きれいで誤った翻訳文になりがちという自省があります。

3.6　スペルミスが心配なら ATOK に頼ろう

筆者は自分が記憶している英単語のスペルや意味にまったく自信がないので、ATOK[*7]という有償の日本語入力システムに頼っています。

ATOK は、たとえば「ほうこくはいじょうです」という入力を、「報告は以上です」という漢字とかなが交ざった文に変換する部分を担うツールです。もともと Windows や Mac にはデフォルトで Microsoft IME などの IME[*8]が入っていますが、性能や精度はそこそこなので、平気で「報告は異常です」といった意図しない変換をかましてきます。ストレスなく文章を書くための投資だと思ってその辺りの変換が非常に賢い ATOK にお金を払うと、きっと期待以上のリターンが得られます。

ATOK は日本語を入力するときにも便利なのですが、英語を書くときにもしっかりサポートしてくれます。たとえば ATOK で「ろくがつ」と入力すると、変換結果に「June」（図 3.3）が出てきます。あるいは「ひつよう」や「ねせさりー」と入力すると、「necessary」が出てきます。

▲図 3.3　日本語を入力すると変換候補に英語も出てくる

[*7] Windows、Mac、iOS、Android に対応。ATOK Passport プレミアムなら月 600 円（税抜き）で最大 10 台までインストールできる。登録している単語や自分の入力のクセなどが複数台の PC とスマホですべて同期されるので非常に便利。 `https://atok.com/`

[*8] Input Method Editor の略。英語であれば物理キーボードに書かれた文字を直接叩くことで文字が入力できるが、日本語のようにすべての文字に対応するキーがある訳ではない言語では、この IME という入力システムを使って変換することで文字入力を行なっている。

また単語の意味や使い方を確認したいときは、調べたい言葉を選択して ctrl キーを 2 回叩くことで、すぐに広辞苑、大辞林、ウィズダム英和辞典、ウィズダム和英辞典といった辞書の最新版が引けます。（図 3.4）

▲図 3.4　単語を選んで ctrl キーを 2 回叩くと辞書が引ける

人間はミスをする生き物なので、自信のない分野は積極的にツールに頼りましょう。お金がかかるのはちょっと嫌なので毎回ネットで検索すればいいや、と思っていても、ちょっとした面倒くささを乗り越えて毎回調べるのは大変なので結局やらなくなってしまう可能性が高いです。生産性のためによいキーボードやよいチェアを選ぶように、ぜひよい IME を使いましょう。

3.7　例は「ex.」ではなく「e.g.」

日本だと例示（Example）の略で `ex.` と書く人が多いですが、ex は「前の彼女」を「ex-girlfriend」と言ったりするときの「前の」という意味なので、例を表すときは `e.g.` を使いましょう。

ちなみに `e.g.` はラテン語で「例えば」という意味の `exempli gratia` の略です。

第 4 章

よいレビューの仕方とされ方

自分が書いた文章をレビューしてもらったり、誰かの書いた文章をレビューしたりするときに、双方が相手に歩み寄って気持ちよくレビューが進められる方法を紹介します。

4.1 レビュアー（レビューする側）のコツ

文章のレビューという工程において、レビュアー（レビューする側）とレビュイー（レビューしてもらう側）の共通目的は「**これをもっとよい文章にして早く世に出すこと**」です。レビュアーの目的は、決して「間違いの多い低品質な文章だったとレビュイーに自覚させること」や「強い言葉でレビュアーを傷つけて反省させること」ではありません。文章を読んで指摘をするレビュアーがこんなことに気をつけると、指摘を受け止めるレビュイーとよいコミュニケーションがとれますよ、というコツを紹介していきます。

4.1.1 重要度合いを添えて指摘のコメントをしよう

レビューでの指摘は「文章をよりよいものにする」ものであると同時に、相手（レビュイー）のやることを増やすものでもあり、一歩間違えると公開を遅らせるブロッカーになり得ます。

レビュアーは軽い気持ちで指摘したのにレビュイーは「これを全部直さないと公開できないんだ……」と思い込んでしまい、必要以上にコストをかけて全部を直すことになった、というようなすれ違いは非常に勿体ないものです。

これを防ぐため、筆者がドキュメントや技術書のレビューをするときは、レビューコメントに以下のような「重要度」を添えています。

- `MUST FIX`
  - 仕様の明らかな誤りなど、重大な問題があるため絶対に直してほしいもの
- `Nice-to-have`
  - 絶対ではないが直した方がよくなるので、できれば直してほしいもの
- `Nitpicking`
  - 軽微な誤りや好みの問題なので、直さなくても構わないが気になったもの

重要度を添えることによって、レビュイーは「早く公開することが最優先なので MUST FIX だけ対応しよう」や「〆切まで時間があるし、読みやすくすること最優先にして全部丁寧に対応していこう」というように、どこまで対応するのかを判断しやすくなります。

自分がレビュイーのときは「指摘されたら迷わず即直す」という姿勢でもいいのですが、一方でどれだけ直しても、もう指摘するところが 1 つも見つからない「完璧で完全無欠な文章」というものが存在しないのも事実です。

どんないい文章でも、次の日に読み直せばさらに何かしら直せる箇所は見つかります。レビュイーが「完璧な文章以外はここを通さないぞ！」と思っていると、指摘す

る→直す→指摘する→直す→指摘する→直す……というループがいつまでも終わらず、よっていつまでもその文章を世に出すことができません。

レビュアーが指摘をするのは自由ですが、**指摘を取り込むか否かというオーナーシップはその文章を書いた人にあります**。そして文章は、世に出なければ多くの人の目に触れてもっと多くの指摘を受ける、という機会も得られないままです。「俺の意見を全部聞け！ 細かいものも含めて全部直すまで世に出すことは許さんぞ！」ではなく、MUST FIX 以外はレビュイーの判断を信じて、自分が必要以上のブロッカーにならないように気をつけましょう。

4.1.2 どう直すべきかを具体的に書こう

レビュアーは、自分がした指摘が**「具体的にどう直すべきか」が読み取れない指摘**になっていないか、あるいは文章をよくすることではなく相手を傷つけることを目的とした攻撃になっていないか気をつけましょう。

「具体的にどう直すべきか」が読み取れない指摘というのは、たとえば次のようなものです。

- 仕様理解が足りていないようです
- ここ、もうちょっと分かりやすくなりませんか？
- ○○の説明に怪しいところがあります
- この説明だと誤解させるので直してほしいです

これらの指摘は「直してほしいようだ」ということは分かりますが、具体的に仕様のどこを理解できていないのか、何がどう分かりにくかったのか、「説明の怪しいところ」とはいったいどこで正しくはどう書くべきなのか、読み手に何を誤解させそうだと感じたのか、などが分かりません。

本来は、文章を書いたレビュイーも、それを読んで指摘したレビュアーも、「もっとよい文章に早く辿り着きたい」、つまり今が 6 階に居るとしたら早く完成度 100% の 10 階に行きたい、という共通の目的があるはずです。であれば、レビュイーが「直すべきということはつたわってきたけど、具体的にどう直したらいいのか分からない……」と 6 階で延々と考え込んでしまって時間を無為に使うのは、レビュアーにとっても嬉しいことではないはずです。

「具体的にどう直すべきか」が読み取れない指摘は、家でビールを飲みながら野球を観戦している人の「お前、ほんとヘボピッチャーだな！」「なんで打てないんだよ、バカ！」みたいな野次に近いものがあります。レビュイーもレビュアーも、本来は同じグラウンドで共に勝利を追い求めるチームメイトのはずです。レビュアーは野次を飛ばす観客ではないので、ぜひ具体的な修正提案をしてあげましょう。

お勧めは変更を Pull request 上で提案できる Suggestion 機能[*1]を使うことです。「A is B」って書いてあるけれど A は C なのでこれは明確な誤りだな、と思ったら「A は B ではありません」とコメントするのではなく、次のような Suggestion を書いてあげることで、レビュイーに対してどう直すべきかが過不足なくつたわると共に、レビュイーはワンクリックで修正提案を取り込むことができます。

```
[MUST FIX]
AはBじゃなくてCです。

```suggestion
A is C
```
```

レビューでもらった修正提案（suggestion）をレビュイーが取り込む[*2]ときは、Pull request の Files changed のタブで `[Add suggestion to batch]` をクリックしていって、全部 Add し終わったら、最後に `[Commit suggestions]` をクリックします。

Suggestion を使うと現状に対して期待値[*3]を明確に提示できて、相手の修正にかかるコストも下げられるので、とってもお勧めです。逆にどう直すべきかという具体案は思いつかないけど、「分かりにくいよ！」という気持ちだけつたえたい、というときは、前述のように重要度が Nitpicking であることを書き添えた上で「こういうふうに自分は分かりにくいと感じたけど、具体的な代替案は思いつかなかった」と気持ちだけつたえてもいいでしょう。

4.1.3 レビューは「相手のやることを増やす」責任を持ってつたえよう

「分かりにくいです」と感想をつたえるだけのレビューや、直した方がより良くなるかもしれない箇所を増やすだけ指摘は簡単です。逆に「これで大丈夫！」と責任を持ってゴーサインを出す方がずっと難しいことです。

書いてある内容についてレビュアーがきちんと理解していれば「あとこれだけが足りない」「ここは明確に間違っているので○○に直しましょう」がはっきり言えます。

[*1] https://docs.github.com/ja/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/commenting-on-a-pull-request

[*2] https://docs.github.com/ja/github/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/incorporating-feedback-in-your-pull-request

[*3] 筆者の周囲では、変更前の現状を as-is、変更後のあるべき姿を to-be と表現することがよくあります。「こう直してほしい」という意図をつたえるときのラベルとしてとても便利です。

ですが、あまり理解していない状態で「レビュアーなんだからなにか言わないと……」と指摘をひねり出そうとすると、「できればこれも書いておいた方が分かりやすいんじゃないでしょうか」と無責任に宿題を増やすだけの人になってしまいます。

レビュイーがなんとかその宿題をこなして、再レビューに出すとまたレビュアーから新しい思いつきを言われて、一体何をクリアしたらこれを通過させられるのか分からない……という無限レビューに迷いこんでしまうと、レビュイーの心は簡単に折れてしまいます。[*4]

レビューをするとき、レビュアーは、その修正指示が**「早く世に出る」という価値を毀損してまで、いまやるべきことなのか**を自分にきちんと問いましょう。

レビューには「文章をよくする」といういい面だけでなく、「相手（レビュイー）のやることを増やす」「早く出るという価値を損なう」というように足を引っ張る面もあります。なんでもかんでも気軽に放言せず、私は今から相手のやることを増やすのだ！ 公開を遅らせてでもここは直した方が絶対にいいんだ！ という責任を持って指摘をつたえるか、前述のような重要度を添えて取り込む取り込まないを相手が選べるようにしてあげましょう。

4.1.4 大量に赤を入れることがレビュアーの存在意義ではない

稀に、レビューを頼まれたときに「たくさん赤を入れないと仕事をしたことにならない」と思ってしまうレビュアーがいます。そういう思い込みのままレビューをすると、「たくさん指摘をすること」が最優先になっているため、文章をよくするためのレビューのはずなのに、逆に誤りを混入させるような修正指示や、簡潔で分かりやすい文章を長ったらしく曖昧にさせるような逆効果の提案をしてしまいます。

大量に赤を入れることがレビュアーの存在意義ではありません。繰り返しになりますが、**よい文章を早く世に出すことがレビュアーとレビュイーの共通目的**であると心得ましょう。

レビューで指摘するところが少なかった、ということは、「レビュアーが仕事をしていない」ということを意味しません。レビュイーが書いた元の文章の出来がすごくよかったのかもしれないし、レビュアーの方が知識が不足していて誤りを見つけられていないだけかもしれません。

どちらにせよ、指摘することが少なかったときに、粗探しをして無理矢理指摘をひねり出す必要はありません。何か指摘をひねり出そうとせず、素直に「分かりやすく書けていたので修正して欲しいところはほとんどありませんでした。気づいたことはこれだけです」とつたえてレビュー結果をレビュイーに渡しましょう。

[*4] 作品を持ち込んでは編集者にダメ出しされることの繰り返しで、何をどう直したら編集会議を通過して掲載されるのか分からない……と心が折れてしまった漫画家や小説家の話を読むと、ジャンルは全然違うけどたぶん同じような状態なんだろうなと思っています。

4.1.5 指摘は基本的に「後出しじゃんけん」だと心得る

文章のあら探しをするのは簡単です。なんでこんな書き方になってるんだろう、何を思ってこういう説明にしたんだろう、と憤る前に、「色んな事情があって、当時はこうとしか書けなかったのかもしれない」「おそらくこれが書いたときの精一杯だったので、立場の違う今の自分が馬鹿にしても何もいいことはない」と落ち着きましょう。

他の人が書いた技術記事に対して「もっとこうすればよかった」「技術的に誤りがある」と指摘するのは、基本的に後出しじゃんけんです。あなたが 0 から書いていたら、もっと別の誤りを仕込んでいたかもしれないし、そもそも書けずに挫折していたかもしれません。レビュアーは初稿を書いた人よりも、そもそも優位な立場であることを自覚したうえで指摘をすべきです。

直すべきところだけでなく、「ここは素晴らしく分かりやすい説明だ！」と思ったところに、「この説明がすごく分かりやすい！」というポジティブなフィードバックも一緒にしてみましょう。それによってレビュイーは「あ、ここの書き方どうするか迷ってたけど、分かりやすいと思ってもらえたなら次もこの書き方を採用しよう」という判断ができます。

4.1.6 頼まれていないレビューはしない

人には間違えて素っ転んで痛い思いをする権利があります。そこにあるのが読み手にとって取り返しのつかないような危険な間違いで、誰がどう見ても MUST FIX なものであり、こちらから修正案も提供できる、というものでなければ、頼まれてもいない勝手なレビューはするべきではありません。

誰かが書いた文章を読んで「こんな粗悪な技術記事を出すなんて」と思ったのであれば、それとは関係なくあなたが正しい内容の記事を書いて、正しい技術知識を世に広めればいいだけの話です。0 から何かを書くことを放棄して、代わりに誰かが書いた 1 に対して文句をつけるだけの人になるのはやめましょう。

野球選手になって試合に出るまでの労力と、その試合を見て外野から「バッティングがなっちゃいない！ 下手くそ！」と野次を飛ばす労力であれば、どちらの方が大変だかは分かるはずです。野次を飛ばしたくなったら、あなたも選手になって試合に出ましょう。

4.1.7 方針を決めるのは別の機会にする

指摘された箇所の修正についてレビュアーとレビュイーの間で折り合いが付かないと、レビューの工程がお互いにとってつらく苦しいものになってしまうことがあります。

こういうときはこう書きましょう、というライティングのスタイルガイドやルールが存在していて、既に方針が決まっているものであれば意見は割れません。ですが、まだルールや方針が決まっていないものについて意見が割れ、さらにレビュアーの A 案もレビュイーの B 案もそれぞれメリットデメリットがあって一概に「こっちが正解だ！ 絶対にこっちの書き方の方がいい！」と言えないときは、どう直すのかなかなか両者間で折り合いがつきません。

方針を決めないと直せない→直せないといつまでもリリースできない→でも方針がなかなか決まらない、という状態は非常に苦しいです。そこでお勧めは「**方針を決めるのは別の機会にする**」という方法です。

組織の中で「レビューの観点を相談して方針を決める会」を毎月の繰り返し予定で入れておき、日々のレビュー指摘の中で「これ、そういえばどうするかという方針が決まってなかったね」というものが出てきたら、次回の議題に入れて「いったんこの文章は現行のままで行きましょう。次回の方針を決める会で方針が決まったら別途対応しましょう」とします。

これによってルールや方針が決まっていなくて意見が割れるものがあったとき、決まるまで進めなくて苦しくなっていたのが、次回の議題に積んでおいてさっと公開に進めるようになります。

方針が決まっていないのでそのうち話しましょうという口約束をして忘れないように気をつける、というやり方だとつい忘れてしまいます。しっかり議題に積んでおくことで、次の「方針を決める会」まではこの話題を忘れてもいい、というように安心して棚上げができるようになります。

4.2 レビュイー（レビューされる側）のコツ

レビュアーは様々な配慮をしながらレビューをしてくれています。できるだけレビューがスムーズに進むよう、レビュイーもレビュアーに歩み寄りましょう。

4.2.1 お願いしたい観点を添えてレビューを依頼しよう

どういう観点でレビューしてもらいたいのか、何についてどれくらいの粒度で指摘をして欲しいのか、という認識がレビュアーとレビュイーの間で合致していないと、「ざっと見て欲しいだけだったのに、頼んでもいない細かいところまで何度も指摘されて、時間もかかり嫌な思いをした」とか「すごくたくさん指摘するところがあってこちらも大変だったのに、言い訳ばかりされて結局全然直してもらえなかった」という地獄のような状態が生まれます。

レビューを頼むときは以下のように、どんな観点で何について指摘をして欲しいのか、あるいはどういう観点の指摘は不要なのか、といった情報をレビュアーへきちん

とつたえるようにしましょう。

- 技術仕様に誤りがないか
- 想定している読者層に向けて分かりやすい構成になっているか
- 日本語として明らかな「てにをは」の誤りがないか
- より分かりやすくするためにできることがないか
- 他の人が書いた章と文体が揃っているか
- 不足している情報がないか
- 説明に不親切なところがないか

逆にレビュアーはレビューを依頼された際、観点や粒度についての情報がなければ確認をしてからレビューを始めるようにしましょう。

4.2.2 指摘は素直に受け入れる

先日、「突然画力が伸びだした時、僕が発見した事」という note[*5]を読みました。内容をざっくりまとめると、絵を描いている途中で「あ、ここおかしいから直した方がいいな」と思う箇所を見つけたときに、人間の習性なのか素直にさっと直さずに「もう〆切まであんまり時間もないし」「ここまで割とうまく描けてるし、下手に直すといまより悪くなるかも」のように、なんとかして直さないで済む理由を見つけてそのままでいきたくなってしまう。けれどその気持ちをぐっと押し込めてすぐに直すようにしたら画力が伸びた、というものでした。

これを読んだ筆者は「絵もライティングもおんなじだな」と思いました。

うんうん唸りながらやっと書き上げたドキュメントや技術ブログに、指摘のコメントをもらうと、その瞬間、「指摘してもらってありがたいな」という気持ちと同時に、「いやいやこういうふうに書いた意図があるので、ここはこのまま直したくないです」という気持ちが湧き上がります。もちろん明らかな間違いであればすぐに直しますが、「そこはどっちでもよくない？ 好みの問題では？」というときは特に、指摘されたことを直すべきか、直さずにそのままでいくべきか、迷ってしまう人は多いのではないかと思います。

筆者も含め、レビュイーはレビュアーからの指摘を見ると、なんだかんだ言い訳をして取り込まない方向に持っていきたくなる習性がありますが、そういうときは次の 2 つに時間をかけるのをやめて、指摘されたところをさっと直すようにしてみてください。

1. 指摘されたことを直すか直さないか迷う

[*5] note 版 突然画力が伸びだした時、僕が発見した事｜安倍吉俊｜ note `https://note.com/abfly/n/n04a315114fcd`

2. なんとか直さず済ませるために相手を説得する

筆者はこの方法を試してみた結果、60% の完成度の文章を、80% や 100% にする速度が速くなって、前よりも文章を書くのがうまくなりました。

指摘された瞬間は「えー、今のままでいいじゃん！ 直したくないなー！」と心から思うんですが、直した結果、「やっぱり直さなきゃよかった」と思うことは基本的にありません。お風呂に入る前は面倒くさいと思っていても、入った後に「入らなきゃよかった」とは絶対にならないのとよく似ています。「直したくないな」と思っていても、直すとちゃんと前よりもいい文章になることの方が圧倒的に多いのです。

4.2.3 レビュアーが気付いたことに、書いたとき気付けなかったのは当然

レビュアーは文章を読むだけなので、リソースの 100% を「注意深く読むこと」に使えます。

一方、書く側は「情報全体を把握し」「どうつたえるか考え」「キーボードを叩いて文章を組み立てながら」「読み返し」「修正案を再び考え」「どちらがよいか判断する」というように、リソースを思考や判断、出力といった様々な作業に振り分けています。

わらわらと 10 人いる幼児の面倒を同時並行で見なければいけない保育士と、1 人だけにつきっきりになれる保育士だったら、それは後者の方が「襟元にカレーのシミがあるな」とか「昨日より少し元気がない」とか色んなことに気付けるはずです。

レビュイーはレビュアーから指摘を読んで、「俺はなぜ……これを書いたときに気付かなかったんだ……こんなにはっきり間違えているのに……」と己の目の節穴さに落ち込む必要はありません。使えるリソースが違うので、**レビュアーが気付いたことに、書いた時点で気付かなかったのはある意味当然**と言えます。

逆に言えば、書いた時点ですべての誤りに気づけるのであれば、レビューという工程自体が不要なはずです。

みんなの頭の中にふわふわとしたアイデアがあるだけで、「書かれた文章」がないと、その文章をもっと良くすることもできません。完璧でなくても下書きレベルでも、そこに文章があったからこそ、それを元にレビューができてもっといい文章に直すことができました。

レビュアーが誤りを拾えたということは、元をたどればレビュイーが最初の文章を書き上げたということであり、そしてその後のレビューという工程がしっかり機能しているということなので、お互いに「えらい！」「えらい！」と言い合ってみんなで喜びましょう。

レビュアーのときは誤りに気付くし、書いているときは気付かない。これはもうそういうものなのです。

第 5 章

どうやってテクニカルライティングを学ぶか

本書を読んでテクニカルライティングをさらに学びたくなったあなたに、お勧めの方法を紹介します。

5.1 テクニカルライティングの検定を受けてみよう

IT 企業におけるテクニカルライターの経歴を見ていると、ざっくり次の 2 つに分かれています。

- ライティング畑で取扱説明書を書いていた人や編集者をしていた人が技術側に踏み出すパターン
- エンジニア畑で開発をしていた人が書く側に踏み出すパターン

筆者は後者だったため、「いっぱい読んでいっぱい書いて、手探りで分かりやすい文章を書くコツを学んできた」という獣道ルートでテクニカルライターになっています。学問として体系立ててライティング技術を学んだ訳ではないので、仕事としては成り立っているものの、これで大丈夫なんだろうかというどこか不安な気持ちがありました。

そこで、テクニカルライティング技術を改めてしっかり学んでみるべく、2022 年に「TC 技術検定 3 級」というテクニカルライティングの検定を受けてみました。

5.1.1 TC 検定 3 級とは

検定の正式名称は「テクニカルコミュニケーション技術検定試験 3 級 テクニカルライティング試験」です。略称として「TC 技術検定 3 級 TW」と書かれることもあります。テクニカルライティングの頭文字で TW ですね。本書では、以後は TC 検定と呼びます。[*1]

試験の名称に入っている「テクニカルコミュニケーション」とはなんだろう？ テクニカルライティングと何が違うの？ と思われたかと思いますが、なんとこの「テクニカルコミュニケーションとは」という問題もこの検定の試験範囲に入っています。

しっかり試験勉強をした筆者の理解としては、「**『文章』だけならテクニカルライティング。そこにイラスト、写真、動画といったメディア、あるいは双方向的なコミュニケーションなどが加わってくるとテクニカルコミュニケーション**」ということのようです。要はテクニカルコミュニケーションは、テクニカルライティングを内包した上位概念です。

[*1] ちなみにこの TC 検定、本書で紹介する 3 級の上位試験として、2 級の「使用情報制作ディレクション試験（通称 DR）」と「使用情報制作実務試験（通称 MP）」もあるんですが、そこまで行くと取扱説明書ガチ勢向けになってきます。公開されている試験問題のサンプルを見てみたんですが「ウォーターサーバーの取扱説明書が題材。リスクアセスメントを行ない、警告図と共に安全表記を記載せよ」みたいな世界だったので、エンジニアのみなさんは一旦気にしなくてよさそうです。

5.1.2 TC 検定はいつどこで誰が受けられるのか

TC 検定には特に「実務経験何年以上」のような受験資格の条件はないので、誰でも受けられます。試験勉強を通してテクニカルライティング技術が学べるので、テクニカルライターとしては未経験でこれから勉強したい、という方にもおすすめです。

TC 検定は毎年夏と冬に 1 回ずつ、年 2 回の開催です。本書が発刊されたタイミング（2024 年 5 月）だと、次回の開催日は 2024 年 7 月 21 日（日）です。2025 年以降の開催日はまだ発表されていませんが、過去の開催日を見る限り、例年 2 月と 7 月に開催されているようです。（表 5.1）

▼表 5.1　TC 検定の日程

年	冬	夏
2024 年	2 月 18 日（日）	7 月 21 日（日）
2023 年	2 月 19 日（日）	7 月 23 日（日）
2022 年	2 月 27 日（日）	7 月 24 日（日）
2021 年	2 月 14 日（日）	7 月 18 日（日）

TC 検定の受験料は 16,720 円[*2]です。ウェブから申し込みできますが、会社が費用を出してくれるので会社名義の領収書が必要です、というような場合は、メールや電話で「領収書ください」と依頼することで、郵送で領収書が送ってもらえます。

試験は会場へ出向いて受けます。筆者が受験したときは、東京、大阪、名古屋、福岡、広島、石川の 6 会場から選んで申し込みができました。

5.1.3 TC 検定の難易度と勉強方法

TC 検定はシャーペンと消しゴムを持っていって、手書きで受ける形式です。試験は大きく 2 つに分かれています。まずは 4 つの選択肢から 1 つを選ぶ選択問題が 50 分、それが終わると 20 分くらいの休憩を挟んで、続いて自分で文章を書く記述問題が 50 分という流れです。

試験の難易度ですが、選択問題の問題数は 50 分で 50 問なので、1 分に 1 問のペースで解いていけば問題ありません。難易度はさておき、選択問題は迷った問題を 2 周見直すくらいの時間的余裕がありました。後半の記述問題は回答欄が多いので、50 分間ずっと懸命に文字を書き続けることになります。

TC 検定の教科書や参考書にあたる本は、「日本語スタイルガイド（第 3 版）[*3]」の

[*2] これは 2024 年 5 月時点の料金です。

[*3] https://jtca.org/learn-tc/publication/guide_jsg/

一冊だけです。出題範囲はすべてこれに含まれており、この本の巻末に試験範囲の説明や例題も載っています。この本がないと話にならないので、試験を受けるのであれば必ず買うべきです。

試験日の少し前に、試験の実施元であるテクニカルコミュニケーター協会が「受験対策セミナー」も開催します。このセミナーでは、実際の試験でどの辺りが出るとか、過去に問題としてここが出たことがある、というようなかなり具体的な説明があります。またセミナーの後半では練習問題を解いて、解説を聞くのですが、その練習問題に一定数「次回の試験で出る問題」が含まれています。そのため、個人的にはこの受験対策セミナーの受講料は、もうほぼ受験料とセットなのだと割り切って受けておいた方がいいと思います。

TC 検定に向けて勉強をした結果、他の人が書いたドキュメントをレビューしていて「こう直した方がいいのではないか」と思ったときに、「理由はうまくいえないんだけど、なんとなくこう書いた方が分かりやすい気がします」ではなく、「こういう理由でこう書いた方がいい」というように根拠を添えて修正提案ができるようになったのがいいところでした。

TC 検定で学んだことは、ドキュメントだけでなく企画書や仕様書や設計書、それから社内での報告書、手順書、告知、Slack でのやりとりなど、色んな場面で実用文を書くときに役立ちます。テクニカルライターでなくても、もうちょっと分かりやすい文章が書きたいな人や、文書の構造化や階層化の手法を学びたいという方にはお勧めです。

5.2 Technical Writing Meetup に参加しよう

筆者は Technical Writing Meetup という、テクニカルライティングをテーマにした Meetup を月 1 回のペースで開催しています。

- X（旧 Twitter）
  - `https://twitter.com/tw_meetup_jp`
- connpass
  - `https://tw-meetup.connpass.com/`
- Discord
  - `https://discord.gg/aBWYMYfftg`
- 過去のアーカイブ動画
  - `https://www.youtube.com/playlist?list=PLWuRMudM606T1fYlj0c0qRmSwRihmwiW5`

各社のテクニカルライターや UX ライターが集まって、毎月様々な知見を共有してくれていますのでぜひ参加してみてください。

あとがき

本書はテクニカルライティングをテーマにした本ですが、あとがきはあくまであとがきであってあとがきでしかないのでここからは自由にまいりましょう！ いえーい！ 書き終わったぞー！ 実用文だとかテクニカルライティングだとかを気にしないときの、私の文体を形成したのは野梨原花南のちょーシリーズ[*4]です。

さて、東京ディズニーランドには、コロナ禍真っ只中の 2021 年 4 月 1 日から始まった「ミッキーのマジカルミュージックワールド[*5]」という常設のステージショーがあります。開演からちょうど 3 年が経った 2024 年 4 月 1 日、このショーの演出や出演者の人数が予告なく大きく変わって、恐らく「これってもともとはこういうショーだったんだ！」という完全版のような状態になりました。

精神が舞浜で育った結果、ミッキーとミニーが顔を見合わせて頷く姿を見ると、すぐに涙ぐんでうんうん頷いてしまう筆者ですが、このショーは本当にお勧めです。やっぱり店の造りと美味しいご飯の組み合わせで最高の一品になるっていうか、舞台のよさと演者のよさが組み合わさって最高のショーが！ あー！ 観てくれー！

ところで話がかっ飛んでばかりですが、最近セカンドハウスを買いました。ベランダから海と山が見えて、部屋のお風呂で温泉が出るヴィンテージのリゾートマンションです[*6]。いわゆる「別荘」の響きとは裏腹にかなり安かったので、おうちオフィスの別拠点としてうっかり買ってしまいました。面白そうなことがあったときに、軽やかに飛び込める自分でありたいなと思います。

2024 年 5 月
mochikoAsTech

[*4] コバルト文庫の不朽の名作。最新シリーズの「ちょー東ゥ京」も 2023 年 11 月に完結したので、ちょーシリーズが好きだった人は読むといいよ！ 大人になってから読んでも面白いです。https://cobalt.shueisha.co.jp/contents/tyoutokyo/

[*5] https://www.tokyodisneyresort.jp/tdl/show/detail/895/

[*6] https://mochikoastech.hatenablog.com/entry/2024/05/05/131125

PDF 版のダウンロード

本書（紙の書籍）をお買い上げいただいた方は、下記の URL から PDF 版を無料でダウンロードできます。

- ダウンロード URL
  - `https://mochikoastech.booth.pm/items/5703302`
- パスワード
  - `****`

Special Thanks:

- 夫を要石にしそうなくらい私のことが好きなねこ
- おきゃんな鳴き方をするねこ
- 感情がすべて鼻息と一緒にもれるねこ
- 誰より丁寧に校正してくれる夫
- 一緒に魔法の杖をぶん回してくれる息子

レビュアー

- otapo
- statditto
- ガブリエル
- 高橋征義 @takahashim
- Shoco Sato
- Nkzn
- KUBOTA Yuji

参考文献

- センスは知識からはじまる - 水野学
  - https://publications.asahi.com/product/15849.html
- ずかん自転車一見ながら学習調べてなっとく - 森下昌市郎
  - https://direct.gihyo.jp/view/item/000000003092
- 日本語スタイルガイド (第 3 版) - 一般財団法人テクニカルコミュニケーター協会
  - https://jtca.org/learn-tc/publication/guide_jsg/
- 論理が伝わる 世界標準の「書く技術」 - 倉島保美
  - https://bookclub.kodansha.co.jp/product?item=0000194754
- ことばから誤解が生まれる 「伝わらない日本語」見本帳 - 飯間浩明
  - https://www.chuko.co.jp/ebook/2013/07/514091.html
- Software Design 2024 年 4 月号 - Software Design 編集部
  - https://gihyo.jp/magazine/SD/archive/2024/202404
- 毎日ことば plus – 毎日新聞
  - https://salon.mainichi-kotoba.jp/

著者紹介

mochiko / @mochikoAsTech

テクニカルライター。元 Web 制作会社のインフラエンジニア。ねこが好き。「分からない気持ち」に寄り添える技術者になれるように日々奮闘中。技術書典で頒布した「DNS をはじめよう 改訂第 2 版」「AWS をはじめよう 改訂第 2 版」「SSL をはじめよう」の「はじめようシリーズ 3 部作」は累計で 12,000 冊を突破。

- https://twitter.com/mochikoAsTech
- https://bsky.app/profile/mochikoastech.bsky.social
- https://mochikoastech.booth.pm/
- https://note.com/mochikoastech
- https://mochikoastech.hatenablog.com/
- https://www.amazon.co.jp/mochikoAsTech/e/B087NBL9VM

Hikaru Wakamatsu

表紙、章扉、目次デザインを担当。

Shinya Nagashio

挿絵デザインを担当。

テクニカルライティング
読み手につたわる文章

2024 年 5 月 25 日　技術書典 16 初版

著　者　　mochikoAsTech
デザイン　Hikaru Wakamatsu / Shinya Nagashio
発行所　　mochikoAsTech