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伝えて質問する
- 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
- 子供に LINE で今週末のお出かけスケジュールを送る
- 薬剤師が処方薬の飲み方や注意事項を紙で見せながら説明する

こんなふうに他の人に何かを理解してもらうことや、情報を齟齬なく伝達することを目的として書かれた実用的な文章のことを**実用文**と呼びます。そしてこの実用文を、相手の理解度や状況に合わせて分かりやすく書く技術が**テクニカルライティング**です。

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、分かりやすい実用文を書くことを生業としている人たちは**テクニカルライター**です。

厚生労働省による職業情報提供サイト[*1]では、テクニカルライターは別名マニュアルライターとも紹介されています。その名前のとおり、もともとは家電や電化製品を買うと付いてくるマニュアルや取扱説明書を書いている人たちがテクニカルライターでした。近年はそこから転じて、IT 系の企業において開発ドキュメントや API リファレンスなどを書くテクニカルライターや、ウェブサービスの UI 文言などを担当する UX ライターといったポジションの募集も増えてきています。

さらに IT 系の中でも、一概にテクニカルライターと言ってもフロントエンドエン

[*1] 職業情報提供サイト - 厚生労働省 https://shigoto.mhlw.go.jp/User/Occupation/Detail/358

ジニアやサポートエンジニア、QA エンジニアのような「技術職」の 1 つとして扱われ、がっつり開発者向けのドキュメントを書くポジションもあれば、サービスを使うエンドユーザー向けのマニュアルを書くため技術知識はそこまで問われないというポジションもあります。またガジェット系の新製品の記事を書くライターが、テックにまつわることを書くライターという意味でテクニカルライターという名前で募集されているのを見かけることもあります。

筆者はもともと Web 制作会社でインフラエンジニアをしてましたが、2020 年から IT 系の企業でこのテクニカルライターというポジションに就いて仕事をしています。ジョブチェンジをした当時はテクニカルライターを抱えている企業は片手で数えるほどでしたが、やはりドキュメントやマニュアルを作る部分で課題を抱えている企業が多かったのか、近年はスタートアップでも「1 人目エンジニア」を採用するときと同じ感覚で「1 人目テクニカルライター」を探している、という話をよく聞くようになってきました。

テクニカルライターがいれば、ドキュメントを書く部分はお任せしてエンジニアは開発に専念できますが、きっとまだそうでない会社の方が多いはずです。エンジニアのみなさん、「ちゃんとドキュメントに書き残すべきだ」という気持ちはあっても、どうしても開発だけが先行してドキュメントは置いてけぼりになってしまったり、せっかく書いたのにどうもイマイチ分かりにくくて結局質問しにくる人が絶えなかったり、そんなつらい思いをしていませんか？

最初に書いたとおり、私たちは、毎日のようにテキストを介してコミュニケーションをしています。そんな中で、もし文章でうまく伝えられずやりとりに齟齬が起きていたら、それは相当なストレスであるはずです。文章がうまくかけるようになりたい！ ごちゃつきがちな情報をスッキリ分類整理して伝えられるようになりたい！ 相手の理解度に合わせていい感じに文章で説明できるようになりたい！ 本書は、そんなあなたに向けて書かれた本です。

いわゆる「文章の書き方」を解説した入門書でよく見かける文法の話だけではなく、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お伝えしていきたいと思います。[*2]

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

[*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。

- 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
- どこに何が書いてあるのか一目で分かるような階層構造を作りたい人

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

本書のゴール

本書を読み終わると、このような状態になっています。

- 読む前よりも文章がうまく書ける
- 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

はじめに **3**
想定する読者層 4
マッチしない読者層 5
本書のゴール 5
免責事項 5

第 1 章 「分かりやすい文章を書く」とはどういうことか？ **11**
1.1 テクニカルライティングとは 12
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 . 12
1.2 「知らない」と書けない 14
1.2.1 自転車の絵を描いてみよう 14
1.2.2 自転車を「知らない」と描けない 16
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 . . . 17
1.3 初心者はいっぱい読んでいっぱい書こう 18
1.4 完璧主義よりも完成主義 19
【コラム】どうやったらテクニカルライターになれますか？ . . . 20

第 2 章 エンジニアのためのテクニカルライティング **23**
2.1 必要ない人に無駄に文章を読ませない 23
2.1.1 読者層を決めてから書こう 23
2.1.2 関係ない人を早めにふるい落とそう 24
2.1.3 想定する読者層とゴールを明示しておこう 26
2.1.4 いつまでに何をしてほしいのかを書こう 27
2.2 文書構造や文章量が適切だと分かりやすい 28
2.2.1 一文の長さは一口で食べられる量にしよう 28
2.2.2 大枠からはじめて段々細かくしていこう 29
2.2.3 既知から未知に繋ごう 30
2.2.4 1 つの段落では 1 つのことだけ書こう 31

2.2.5 一度に把握できることは 7 つまで 31
2.2.6 最新のドキュメントが埋もれないにしよう 31
2.3 書く速度を速くするためにできること 32
2.3.1 まずは「分かりやすい一文」を書けるようになろう 32
2.3.2 全部並べていちばん良い文を早く見つける 32
2.3.3 漕ぎ始めを生成 AI にサポートしてもらおう 35
2.4 読み手に配慮しよう 35
2.4.1 気持ちが伝われば文法はどうでもいい？ 35
2.4.2 分かったと分からないの両方の気持ちが必要 36
2.4.3 意味のない空白や意味のないスペースを残さない 36
2.4.4 単語で終わらせない 36
2.4.5 語順を入れ替えよう 36
2.4.6 修飾語はかかる言葉に近づけよう 36
2.4.7 同じものは同じ名前で呼ぼう 37
2.4.8 箇条書きを挟んだ文章を作らない 37
2.4.9 主語と述語を対応させよう 37
2.4.10 「自分」という主語に注意しよう 37
2.4.11 ひらがなに開こう 37
2.4.12 一文の中で同じことを二度言わない 38
2.4.13 いただくはだいたい「くださる」にできる 38
2.4.14 「しましょう」は Let's do it together か You should do it か 38
2.4.15 「教えてあげる」ことに酔わないこと 38
2.4.16 変わっていく「語感」を捨て置かずに拾おう 38
2.5 再利用しやすい文章にする 39
2.5.1 文章と修飾を分けて書こう 39
2.5.2 再利用しやすいテキストにしよう 39
2.5.3 並列をナカグロで書くと、後の変更で箇条書きにしたとき見た目が変になる 39
2.5.4 タイトルは「概要」にすべきか、「○○の概要」にすべきか 39
2.6 技術文書に特有のコツ 39
2.6.1 正しい名前で呼ぼう 39
2.6.2 年月日や対象バージョンを書いておこう 40
2.6.3 例示用の IP アドレスやドメインを使おう 41
2.6.4 リンクテキストを「こちら」にしない 42
2.6.5 リンクは「張る」ものか「貼る」ものか 42
2.6.6 時刻の表記は JST か 42
2.6.7 その「文字数」って何文字ですか？ 42

2.6.8　以上と以下か、より大きいと未満か 42

第 3 章　英語を書くとか翻訳するとか　43
3.1　単語と単語の間にはスペースが 1 つ必要 43
3.2　単語や文の単位で翻訳すると危ない 44
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 45
3.4　翻訳はもとの言いたいことに立ち返って考える 46
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 47
3.6　スペルミスが心配なら ATOK に頼ろう 48
3.7　例は「ex.」ではなく「e.g.」 49
3.8　機械翻訳があれば英語がまったく分からなくても訳せるか 49
3.9　翻訳しづらい文章をやめよう 50

第 4 章　誤解を生まないテキストコミュニケーション　51
4.1　まずは「何の話か」を先に言う 51
4.2　背景も添えて話そう 51
4.3　「聞いてほしい」のぶつかり合い 51
4.4　Slack でのコミュニケーション 51
4.5　チケットの書き方 51
4.6　チケットのサイズはできるだけ小さくする 52
4.6.1　素早く近づける 52
4.7　見せて選んでもらう 52
4.8　いっぱい書いて Working Out Loud な働き方をしよう 52

第 5 章　レビューする、レビューされる　53
5.1　レビュアー（レビューする側）の心がけ 53
5.1.1　重要度合いを添えて指摘のコメントをしよう 53
5.1.2　どう直すべきかを具体的に書こう 54
5.1.3　レビューは『相手のやることを増やす』責任を持って伝えよう 56
5.1.4　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 . . 56
5.1.5　指摘は基本的に「後出しじゃんけん」だと心得よ 57
5.1.6　頼まれていないレビューはしない 57
5.1.7　方針を決めるのは別の機会にする 58
5.2　レビュイー（レビューされる側）の心がけ 59
5.2.1　お願いしたい観点を添えてレビューを依頼しよう 59
5.2.2　指摘は素直に受け入れる 59
5.2.3　レビュアーが気付いたことに、書いたとき気付かないのは当然 60

第 6 章　どうやってテクニカルライティングを学ぶか　63
6.1　テクニカルライティングの検定を受けてみよう 63
6.1.1　TC 検定 3 級とは 63
6.1.2　TC 検定はいつどこで誰が受けられるのか 64
6.1.3　勉強方法は？ 66
6.1.4　合否はすぐ分かる？ 67
6.1.5　合格率は？ 67
6.1.6　受けてみて得たものはあった？ 67
6.2　色んな本を読もう 68
6.3　「書き方」の指標を見つける 68
6.4　Technical Writing Meetup に参加しよう 68

あとがき　69
PDF 版のダウンロード 70
Special Thanks: 70
レビュアー 70
参考文献 70

著者紹介　71

第１章

「分かりやすい文章を書く」とはどういうことか？

　具体的な「分かりやすい文章の書き方」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。

ですが日本語ネイティブであれば、もともと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、論理的かつ簡潔明快に説明をしており、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。

文章を書くというと、どうしても人の心を打つようなエモい文章や技巧的な文章、凝った小難しい文章を書ける高い文章力が必要なのではという方向で考えてしまいがちです。ですが小説や詩文とは違って、実用文においては前述のとおり「論理的かつ簡潔明快」が正義なので「流れるような文体」や「文学の香り」といったものは必要ありません。

また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なんとか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が、実用文においては「長ければ長いほどいい」という尺度は存在しません。読んだ結果、得られる情報が同じならば、無駄に長々しい文章よりも、短い文章でサクッと理解できた方が圧倒的にいいからです。

実用文は小説や詩文とは異なる、という話は理解しやすいと思います。その一方

[*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/

で、一見すると実用文のようですが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。それが次の 2 つです。

- 法務文書
  - 簡潔で明快な表現よりも、論理的な整合性を優先して書かれるため、テクニカルライティングとは正解が異なります。
- 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、意図的に間違った言い回しや、誤読させうる表現を用いるなど、テクニカルライティングでは「やってはいけない」とされていることが正解になり得ます。

テクニカルライターという職業の名前だけを聞くと、なんとなく「利用規約をレビューしてもらう」「この商品をうまく説明するキャッチーな見出しを書いてもらう」といったことも頼めるような気がしてしまいますが、法務文書や広告は他の実用文と善し悪しの物差しが異なります。そのため、テクニカルライターがよかれと思って文章を直したことでかえって法的な誤りを仕込んでしまったり、簡潔な言い方にしたことで印象に残らない広告にしてしまったりする可能性があります。

食を楽しみたい人にとっては美味しい食事が「いい食事」ですが、体を絞って筋肉をつけたい人にとっては脂質や糖質を抑えてタンパク質を効率よく取れる食事が「いい食事」です。つまりどんなシチュエーションでも、どんな人にとっても 100 点満点な「いい文章」というものは存在せず、**「いい文章」の定義も目的によって異なる**ということです。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

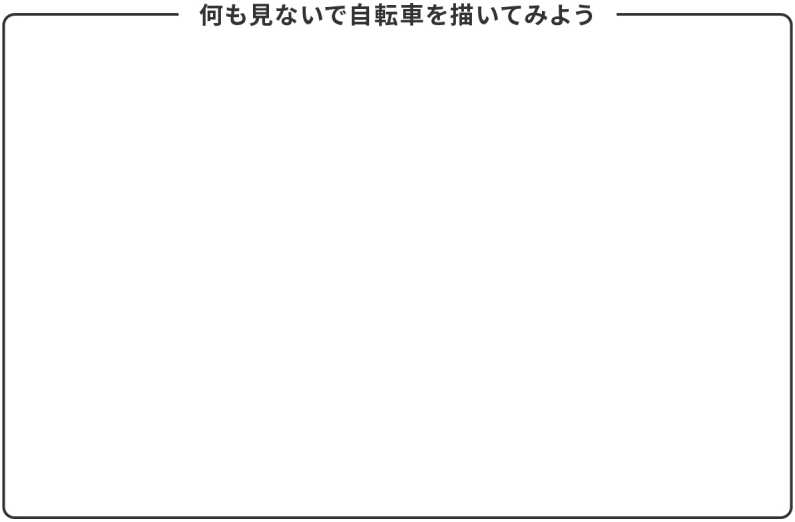
さて、話は変わりますが「自転車」を知っていますか？

何を唐突にと思われるかもしれませんが、自転車を知っているかと問われたら、恐らくあなたを含め、大半の人が「知っている」と答えるはずです。

殆どの人は自転車というものを知っています。乗ったこともあるし、日々見かけるし、子供乗せ自転車[*2]とかロードバイクとか色んな種類があることも知っています。自転車は英語だと bicycle だということもきっと知っているはずです。

1.2.1 自転車の絵を描いてみよう

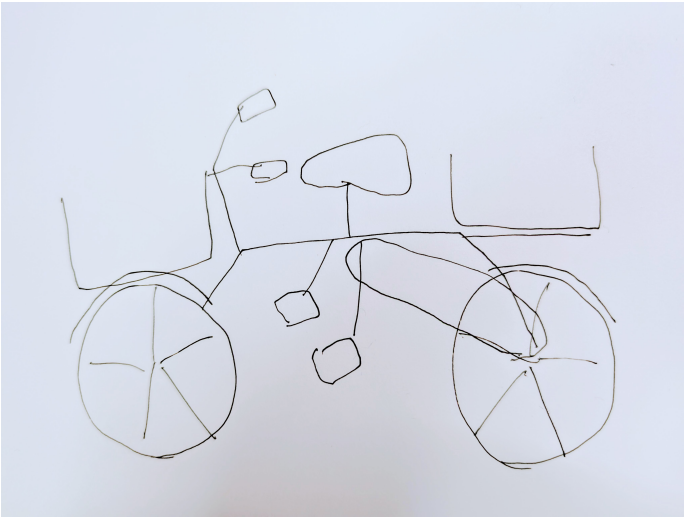
ではちょっと、自転車の絵を描いてみましょう。あなたがよく知っているはずの、自転車の姿を頭に思い浮かべて、下手でもいいので自転車の絵を描いてみてください。（図 1.1）



▲図 1.1　自転車の絵を描いてみよう

[*2] 電動アシストの子供乗せ自転車は、うっかり充電が切れると漕ぐのがめちゃくちゃ重くてつらいことも知っている……知っているのに、なぜ人は充電を忘れてしまうのか……。

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ってたでしたっけ？ 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だっけ？ 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

ばばーん！ 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……。

もともと自転車の構造をよく理解している人や、絵を描くことを生業にしていて普段から自転車を含め色んなものをよく観察している人でなければ、恐らくはこんな感じの「えー、自転車ってこんなだっけ？」という絵が生まれたことと思います。

さて、正しい自転車の絵がこちらです。（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。さて「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」と私たちは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**はずだ、ということが分かります。

またハンドルと前輪は、フロントフォークという部品によって繋がっています。こ

の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、乗っている人が操作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。さらにこのフロントフォークは、名前のとおり食器のフォークのように先の方が少し持ち上がるように曲がっており、それによって舵取りをしやすくしています。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、書く前はラクに書ける気がするのに、実際書き始めるとなぜ書けないのか……と悩んだことがありませんか？[*3]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が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

[*3] 筆者はよくあります！ 本書も最初はさらさら書けると思っていたのに、全然うまく書けなくて苦しみました。

いう理解度の人

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伝えられない、と思っている人にとって、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[*4]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 初心者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？」と相談された場合、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**ことをお勧めしています。

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「こういう説明の仕方」といったい

[*4] 「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

いパーツ、あるいは「こういう構成」や「こういう順番」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。

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、技術ブログや技術記事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初心者であれば、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、いつまでもできあがらない、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**〆切を先に得る**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、そもそも自分ではそれが誤解だと気付いていないので、目に見える形で自分の理解を外に出して誰かに指摘してもらわない限り「自分が誤解している」ということにも気づけません。

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。ただの一度も空振り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。

まずは〆切を手に入れて、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 系の会社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 でもなんでもいいんですが……」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殆どなくなってしまいます。

たとえば毎日毎日ラーメンを食べる仕事があったとして、その仕事に向いているのはどんな人かと言われたら、恐らく「仕事でもないのに毎日食べてしまうくらいラーメンが好きな人」だと思います。そのため「ラーメンって美味しそうだけど、今まで殆ど食べたことがない。だけど毎日ラーメンを食べる仕事に就きたいと思っている」という希望に対して、私からは「取りあえず頻繁に食べてみて、結構いけそうか、それとも毎日食べるのは思ってたよりしんどいのか試してみたらどうでしょう」と勧めています。

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、適性があるのは「○○に興味があるのでこれからやろうと思う」と言っている人ではなく、「面白そうだから○○をやってみた」と言う人です。[*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、技術記事など、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、文章の書き方も変わります。

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。

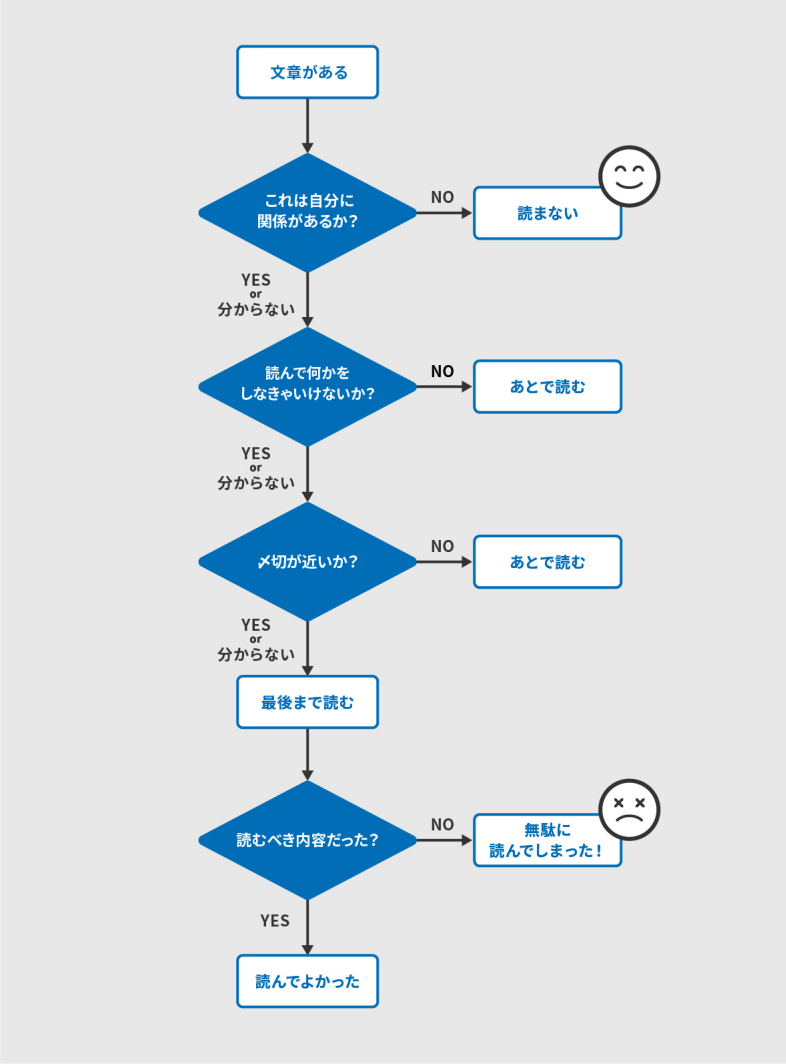
あなたがこれから書く文章は誰に向けたものなのか、を最初にしっかり決めておかないと、のちのち「どこまでさかのぼって説明しないとだめなんだ……？ HTTP ス

テータスコードの概念からか……？」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ので、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殆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防ぎましょう。これらを書いておくと、著者自身が文章の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、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きました。

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、「どういう気持ちで何をして欲しいのか」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。

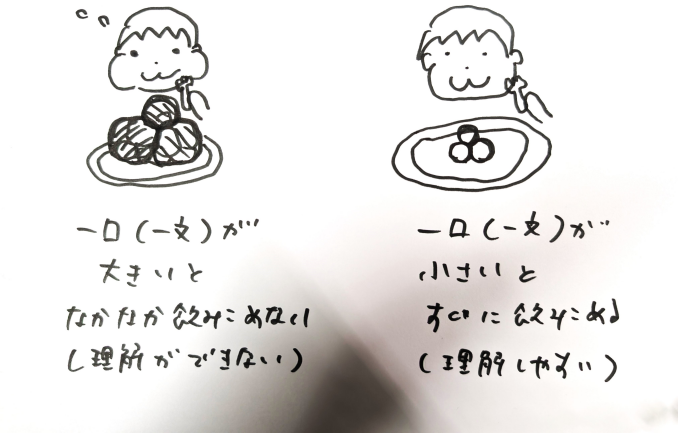
2.2.1 一文の長さは一口で食べられる量にしよう

まずはこの「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伝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殆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 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。

2.2.2　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」「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、先に細かい部分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。

2.2.3 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て、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。

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。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

ランチ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？」などと何度か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全体が把握しやすくなります。

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然と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することで、本当に必要なものと古い情報だが取っておきたいというものが同じ場所に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]では、「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、検索した際にんその記事の検索順位を落とすことができます。

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

2.3.1 まずは「分かりやすい一文」を書けるようになろう

タマネギのみじん切りができないのに、いきなりハンバーグに挑戦するのはやめろ段階的に難しくなっていくので、段階的にできるようになっていこう

一文を考える一段落を考える一ページを考える一章を考える一冊を考えるドキュメント全体を考える段々と難しくなっていく

2.3.2 全部並べていちばん良い文を早く見つける

早くたくさんのパターンを書いて、最善手を見つける 100 点を思いつくまで書かないのではなく、思いつくままにたくさんのパターンを書いていく改善案を思いついたら直前の案をコピペして別の案として書くとにかくナンバリングしながら全部の案を並べていく 1 番良い物を見つける

「いちばん良い一行を見つける近道」

どういうことかというと、こんなシチュエーション、心当たりがありませんか？

「ここの説明、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…

いちばん良い一行を探して、気づいたらあーだこーだと気づいたら長時間悩んでしまっていた！

心当たり、ありますかね？

今日は、こういう「いちばん良い一行」を探すときは、こういうふうにやるといいよ、という方法を紹介します。実践的に、実際やってみるのが分かりやすいと思うので、お題を出しますね。

ちょっと前なんですけど、LINE API Status っていうサイトをオープンしました。

これは AWS のサービスヘルスチェックダッシュボードとか、Twitter の API Status みたいな感じで、開発者が「あれ？ このサービス落ちてる？」とか、「この

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

API、応答しないけどいまなんか起きてる？」っていうのを確認したいときに、落ちてるのか落ちてないのか、状況を一目で確認できるサイトです。

我々は思いました。LINE API Status サイトがオープンした！ めでたい！ LINE API を使っている開発者のみなさんに、ぜひともお知らせしたい！

そうだ、LINE Developers サイトでニュースを出そう！ と思いました。

で、ニュースのタイトルはどうしようかと悩んだんですね。

初案がこれでした。障害状況やステータスを確認できる LINE API Status を公開しました

障害が起きてるってことがすぐ分かるし、何も起きてなくて LINE API が安定稼働してるっていうステータスも分かる、そういう LINE API Status っていうサイトがオープンしたんだなぁ、なるほど。つたわる気がする。いいタイトルでは？ よしよしって思いました。

でも、前回テクニカルライターのお仕事を紹介したときにも話したんですが、LINE Developers サイトって日本語と英語で同時公開してるのんですね。日本語書き終わった後に、チーム内で英訳するんですけど、タイトルを英訳してみるとなんか変だなってなったんですよ。

それがこちらです。

日：障害状況やステータスを確認できる LINE API Status を公開しました英：LINE API Status site for checking outage status and status opened

障害状況やステータス、日本語だとまあそこまで違和感なかったんですが、英語にすると Outage status and status なので、同じこと 2 回言ってるわーと。それに気づいてから改めて日本語のタイトルを読んでみると、確かに「障害が起きてる」っていうのも API のステータス、つまり状況のひとつなので、「障害状況やステータス」って言っちゃうと、同じこと 2 回言っててておかしいなーと。

で、英訳してくれたライターが気を利かせて、こうしてくれたんですね。「Outage status and status」って 2 回言わなくていいだろ、日本語はそのままで、英語だけ「and status」削っておいたぞ、と。なるほど配慮。ありがとう。

良いかな！ このまま行っても！ とちょっと思いました。

でもすかさず、レビューしてくれた別のテクニカルライターから日本語に突っ込みが入ります。

> 悩ましいのですが、障害「状況」と「ステータス」が同意なので、「障害の概要やステータス」ではいかがでしょうか？ もう少し座りの良い言い回しがあれば良いのですが。。

おっしゃるとおりです。英語だけ「and status」削ればいいって話じゃなくて、日本語のタイトルをちゃんと直すべきですね。よし、腰を据えて良いタイトルを模索しよう！

ということで、よりよい一行を見つける旅に出ることになるんですが、こういうと

きにやってはいけないことがあります！

初案の、障害状況やステータスを確認できる LINE API Status を公開しましたから、「状況や」のところをささっと消して、さっきレビュアーに提案してもらった障害の概要やステータスを確認できる LINE API Status を公開しましたに書き換える。よし、第 2 案できた！

これ、だめです。何がだめかというと、初案と第 2 案の比較ができないんです。

こっちが初案で、こっちが訂正後の第 2 案です。

まだ初案と第 2 案の 2 つくらいなら、どこを変えたか、原型はなんだったか覚えていられると思うんですが、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、いやいややっぱりさっきの方がよかったかも、みたいなことをやってると、

え、さっきのってどれだっけ…？ っていう未来が待っています。やっぱりさっきのやつがよかった気がするけど、「さっきの」がどんなんだったかもう思い出せない。

よりよい一行を求めて旅に出て、段々良くなっていったとしても、そのヒストリーと差分が見えないと、善し悪しの比較もできなくなっちゃうんですね。

「いちばん良い一行」を探すときにお勧めの方法

という訳で今日の本題、「いちばん良い一行」を探すときはこういうふうにやるといいよ、というこういうときにお勧めの方法はこちら！

直前の案をコピペしながら、全部の案を並べていく、です。できればナンバリングもしましょう。2 つくらいなら「上の方がいい」とか「下の方がいい」で済みますが、こんな感じで案が増えていったときに「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〜」みたいになって辛いです。

こんな感じで、書きながらナンバリングしておけば、「4 よりは 6 の方が分かりやすい」みたいに「どれ」っていうのをはっきり指せるので便利です。

この全部の案を並べていく方法、自分ひとりで考えるときも便利ですが、誰かに「ちょっと 5 分付き合ってください」みたいに声をかけて、Zoom で画面共有とかしながら頭突き合わせてあーだこーだ考えていくと、改善の螺旋階段を上がってく速度が急激に速くなって、5 分くらいで「えっ最高じゃん？ 最高に分かりやすい。これが最高では？ はー、最高！ 分かりやすい！」みたいな一文にすぐ辿りつけます。

1. 障害状況やステータスを確認できる LINE API Status を公開しました 2. 障害の概要やステータスを確認できる LINE API Status を公開しました 3. 障害の概要や安定稼働を確認できる LINE API Status を公開しました 4. 安定稼働・障害状況を確認できる LINE API Status を公開しました 5. 現在の稼働状況を確認できる LINE API Status を公開しました 6. サービスの稼働状況を確認できる LINE API Status を公開しました 7. サービスの稼働状況や障害状況を確認できる LINE API Status を公開しました

これ、実際に矢崎さんとあーだこーだ喋りながら考えたときのリストそのままなん

ですが、このときは最終的に「6 だな！！」ってなって、6 でリリースされました。こんな裏側を経て、公開されたニュースがこちらですね。

このタイトルの裏側に、そんなエピソードがあったんだと思うと、ニュースも可愛く見えてきますね。

やっぱりニュースひとつとっても、自分ひとりで書いて、誰のチェックも受けずに出すっていうことはなくて、さっきみたいにレビューしてくれる別のテクニカルライター、それから機能やサービスの担当者、あと問い合わせを受けるサポートのチーム、そういう色んなところと連携して「こういう背景から生まれた機能なので、ここをもっと強調したい」とか「この書き方だと、恐らくこういう懸念から問い合わせが増えちゃうので、そこを解消できるようにこういう書き方にしてほしい」とか、色んな会話をしながら、タイトルひとつとっても色々考えながら、こうやってひとつひとつのドキュメントやニュースを出しています。

という裏話でした。

ということで、まとめです。

いちばん良い一行を探すときは「もとの一行」に直接手を入れずに、コピペしながら全部の案を並べていきましょう。ナンバリングするとなおよしです。テキストエディタでやれば、最初から行番号出てるので、自分で番号振らなくてよくて楽ですね。

以上、いちばん良い一行を見つける近道、というお話でした。

2.3.3 漕ぎ始めを生成 AI にサポートしてもらおう

使えるなら ChatGPT でも GitHub Copilot でもなんでも使う。自転車はこぎはじめがしんどい。文章も同じで、初案を書く作業がしんどい既にある文章を読んで、あーだこーだ文句を付けながら手直ししてく方がラク

ゴミみたいな初案を自分で作れるならそれでもいいけど、「こんな話を書きたい」というのを言って ChatGPT にゴミみたいな初案を作ってもらうのもあり漕ぎ始めだけやってもらうと、スピードがのった状態で手直しから入れるからラク

2.4 読み手に配慮しよう

2.4.1 気持ちが伝われば文法はどうでもいい？

配信などを見るとき、人は多少画質が荒くてもなんとか我慢して見られますが、音が悪いと聞くに堪えなくてすぐに閉じてしまいます。それと同じで、書き手が伝えようとしていた内容がどれだけ重要で素晴らしいものだったとしても、その文章が文法的に誤ったひどいものだと読むに堪えないので最後まで読んでもらえません。

「気持ちが伝われば文法だのなんだの細かいルールはいいじゃないか」という主張

も分かりますが、音の悪い配信は生理的に聞くに堪えないのと同じで、あまりに文法が間違っていると「情報を伝える」という目的を阻害するほど読みにくくなってしまいます。

「気持ちが伝わればいい」と思っていても、最低限の文法は守らないとまず「伝わらない」ので、読み手に配慮しましょう。

2.4.2 分かったと分からないの両方の気持ちが必要

分からないから説明を読むが、分からないと読めない

「返ってきたパケットが TLS/SSL record ではないってことですよ」「え、どういうこと…？（しばらく調べる）あー！ 返ってきたパケットが TLS/SSL record ではないってことか！」すごい！ さっきまで分からなかった文章が分かるぞ！ 分からないから説明読むんだけど、説明は分かんないと読めなくて、分かると読めるんだよな。

分かんない人の気持ちも分からなきゃいけないし、分かんない人に教えられるだけの理解度もいる分からないときの気持ちや、何が分からないのかを書き残しておこう。

2.4.3 意味のない空白や意味のないスペースを残さない

「なぜここに空白？ 敢えて？」となるので、意味のない謎の改行やスペースを残さない「なんでここはこうなの？」と聞かれたら全部理由が言えるようにしておこう

2.4.4 単語で終わらせない

「確認」じゃなくて「確認した」なのか「自分が確認する」なのか「相手に確認してもらう」なのか「事前告知」じゃなくて「事前告知をした」なのか「事前告知をする」なのか最後まで書く

「説明がうまい人」とかどういう人のことか？「話したいこと」を「話したい順番」で好きに話す人手はなく、「相手の聞きたいこと」を「相手の関心度が高い順番」で話して、早く疑問を解消してあげる人

2.4.5 語順を入れ替えよう

より分かりやすい語順がないか考えてみよう。

2.4.6 修飾語はかかる言葉に近づけよう

修飾語は近づけよう。

誤解の少ない文章にする特効薬は、一文を短くすること。複数の修飾が延々と続くような説明はやめて、一文ごとに句点で終わらせる。

2.4.7 同じものは同じ名前で呼ぼう

いままでのものが「A」に変わったら、「いままでのもの」をなんて呼ぶのか考えよう

同じものを安否確認サービスと安否確認システムと緊急時安否確認アラートみたいに色んな名前で呼ばない。検索したときに、完全一致じゃないと引っかからない検索システムでも引っかかるように。あとバラバラの名前だと修正時に漏れる。

2.4.8 箇条書きを挟んだ文章を作らない

私がやりたいことは

- A
- B
- C

の 3 つです。
みたいに書かない。

2.4.9 主語と述語を対応させよう

主語と述語のねじれを見つけよう。

2.4.10 「自分」という主語に注意しよう

カメラマンから「写真を撮るときは鏡に映った自分の目を見てください」と指示されたら、あなたは誰の目を見ますか？ この指示文は、次の 2 つの解釈が可能です。

- 写真を撮るときは、鏡に映ったカメラマンの目を見てほしい
- 写真を撮るときは、鏡に映った自分自身の目を見てほしい

自分という一人称が指すものは、「私」であることも「相手」であることもあります。

2.4.11 ひらがなに開こう

一方で、漢字にしないと意味が即座に取れずに返って分かりにくいものもある。

2.4.12 一文の中で同じことを二度言わない

「俺が避けるべきだと思う実装は、こういうのやああいうのは避けるべき」みたいに、文章として崩壊していないか確認しよう。

2.4.13 いただくはだいたい「くださる」にできる

2.4.14 「しましょう」は Let's do it together か You should do it か

2.4.15 「教えてあげる」ことに酔わないこと

「教えてあげる」「話を聞いてもらう」のは基本的に気持ちがいいことなので、酔わないように気をつける。色々アウトプットをしているように見えて、実際はここ 10 年同じことを繰り返し言っているようだと、苦しみながら新しいものを学ぶという楽しい時間を逃しているかもしれない。

2.4.16 変わっていく「語感」を捨て置かずに拾おう

若い世代に「1 時間弱」は何分ぐらいか？ と聞くと、「1 時間が 60 分、60 分とちょびっとだから 70 分くらい？」と答えるらしい。まさか、と思って息子に聞いたら、その通りに答えた。

その理屈でいくと「1 時間強」は何分くらいになるのか？ と聞いたところ、「60 分と結構たくさんくらいなので、85 分くらいとか？」らしい。なるほど。同じ理屈で大さじ 1 杯弱も「大さじ 1 杯＋ちょっと」だと思っていたとのこと。

本来の解釈は伝えたが、こういう「口に出さない誤解」は周囲も誤解に気付かないので解くのが難しい。

言葉の本来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。

学校で先生に当てられて答えを言ったとき、先生から「はい。結構です」と止められたら、「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、人に寄って異なると思います。

正しい意味はこれなんだ！ 誤解した意味で受け取る側が悪い！ 日本語をちゃんと勉強しろ！ と怒る気持ちも分かりますが、誤解する人が 2 割、3 割を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ありません。

誤解する人が多いと分かった時点で、「1 時間弱というのは本来は 1 時間より少し

少ないという意味です」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〜60 分」のように誤解されない言い方に直してあげましょう。

2.5 再利用しやすい文章にする

2.5.1 文章と修飾を分けて書こう

Markdown 記法のように、文章そのものと、文章の修飾を分けて書こう。Word とか CMS みたいに、文章そのものと修飾が分かちがたく 1 つになっていると後で再加工が辛い。

2.5.2 再利用しやすいテキストにしよう

再利用しやすい、再加工しやすいテキストにしよう。箇条書きに・をつかっていると、「マークダウンに変換しよう」と思ったとき、箇条書きでないところ「りんご・バナナ・いちご」も誤変換されてしまう。

2.5.3 並列をナカグロで書くと、後の変更で箇条書きにしたとき見た目が変になる

・生命・身体・健康に影響を与えるもの・投資・資産運用に関連するもの・クラウドファンディング・寄付・投げ銭に類するもの・ヒーリング・セラピーに類するもの・自己啓発に類するもの・メンタルヘルスに関連するもの・宗教・スピリチュアルに関連するもの・政治に関連するもの・アカウントと関連のないプランを提供するもの

2.5.4 タイトルは「概要」にすべきか、「○○の概要」にすべきか

上位のタイトルを見れば補完されて分かるけど、場所を動かす可能性もある。私はタイトルだけで分かるようにしたいので「○○の概要」派です

2.6 技術文書に特有のコツ

2.6.1 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[*2]。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[*3]のも初心者あるあるですので、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。
```

2.6.2　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、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

[*2] 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。

[*3] 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。

ましょう[*4]。技術書であれば奥付[*5]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。

2.6.3 例示用の IP アドレスやドメインを使おう

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ドメインや IP アドレスを使いましょう。

実はインターネットでは「例示やテストで使っていいドメインや IP アドレス」というものが定められています[*6]。

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。

- 例示として使えるドメイン
  - example.com
  - example.net
  - example.co.jp
  - example.jp
- 例示として使える IP アドレス
  - 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）

例示であっても自分の持ち物でないドメインや IP アドレスを勝手に使うことはトラブルの元になります[*7]。必ず例示用のドメインや IP アドレスを使いましょう。

[*4] 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。

[*5] 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。

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

[*7] 実際にどんなトラブルになるのか？ は「DNS をはじめよう」の P110「＜トラブル＞ test@test.co.jp を使って情報漏洩」で紹介しています。

2.6.4 リンクテキストを「こちら」にしない

可能なら URL は全部書くコピペしたときに、テキストとして貼り付けると情報が失われることがあるあとリンク先が間違っていたときに、リンクテキストが書いてあることで間違いに気づけるリンクを開く前に、なにに飛ばされるのか判別できないので、見るべきか見なくていいのか分からない

たとえば利用規約など同意必須の内容を「こちら」をリンクにして参照させていた場合、何かで A タグが外れたり、プレーンテキストとしてコピペして別の場所で使われた際に意図せずリンク情報が消えることがある。URL をそのまま書いていれば、少なくとも自力で URL を開いて参照できる。

2.6.5 リンクは「張る」ものか「貼る」ものか

蜘蛛がこちらからあちらへ糸を張るように、他の情報と繋ぐためのリンクなので、個人的にはリンクは「張る」を使っています。

URL をコピーペーストするイメージで「貼る」を使いたくなるかもしれませんが、Slack でリンクを教えてあげるときは「URL 貼っときますね」だし、「商品一覧から詳細ページにリンクを張る」ときは「張る」だと思っています。

`https://twitter.com/mainichi_kotoba/status/1769229909283778901`

2.6.6 時刻の表記は JST か

時刻の表記は JST？ UTC？

2.6.7 その「文字数」って何文字ですか？

文字数の上限を示すときは、カウント方法をちゃんと書こう。半角は 1 文字分？ 全角なら 2 文字分？ それとも 1 文字は 1 文字？ 絵文字は何文字？

2.6.8 以上と以下か、より大きいと未満か

上限や下限の数字を示すときは、「以上」なのか「より大きい」なのか、「以下」なのか「未満」なのかを明示しよう。

第 3 章

英語を書くとか翻訳するとか

英語ができない日本語話者のエンジニアが、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と、書いた文章が変な言い回しになっていないか確認する方法を紹介します。

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.js,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  FlutterNuxt.jsKubernetes
```

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.js, 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よくやる失敗[*1]です。

3.2 単語や文の単位で翻訳すると危ない

OSS などで英語を日本語訳するとき、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すると、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。

[*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」というイベント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

原文と思われる英語[*4]を当たってみると、どうやら Terminate と Delete を両方とも「削除」と訳したことで生まれてしまった悲しい翻訳文だったようです。

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

「インスタンス」は AWS におけるサーバのことです。なので恐らく原文はこう言いたかったのだろう、という意図を汲んで日本語にすると、「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ます。そのため『終了』させたサーバには二度と接続できないので注意してください」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。[*5]

Terminate を英和辞典で引くと「終わらせる」「解除する」という訳が出てきます。「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 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」という進次郎構文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）

[*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～」をどうぞ。 https://booth.pm/ja/items/1032590

▲図 3.1　ログアウト画面

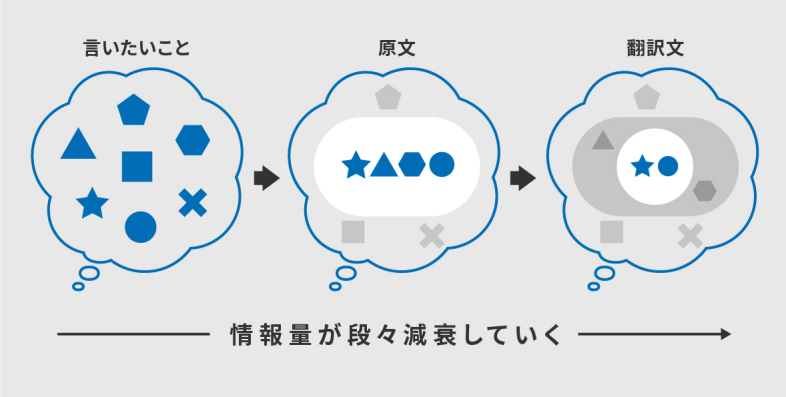
恐らく「Take Me Home」の訳は、「ホームに戻る」が適切だったはずです。その文章がどこに配置されるのか分からずに訳すと、やはり「正しい」のにとても違和感のある UI 文言になってしまうことがあります。[*6]

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も誤訳してしまうことがあります。

もともと頭の中に何か「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、文章として保存されるたびに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すると、原文から翻訳文になるとき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）

[*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ってうーん……。 https://twitter.com/mochikoAsTech/status/1779009150426767775



▲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や、元の言いたかったことを質問して、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 や技術書など英語から日本語に翻訳する場合、元の「言いたいこと」や前後の文脈を汲めるかどうかが重要です。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個人的には考えています。もし技術力はあっても、英語が不得手だからと OSS 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伝えない、きれいで誤った翻訳文になってしまいます。

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を使っています。

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 にお金を払うと思っていた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わけではない言語では、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 回押すことで、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最新の辞書が引けます。（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.` と書く人が多いけど、ex だと「前の彼女」を「ex-girlfriend」って言ったりするときの「前の」の意味なので e.g. の方がよさそう。

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

3.8　機械翻訳があれば英語がまったく分からなくても訳せるか

ChatGPT も DeepL も、しれっと真逆の意味に翻訳したり、間の文章を一文落としてきたり、逆に 2 回繰り返して訳してきたりする。

機械翻訳はあくまで電動自転車みたいなもので、自力ではまったく自転車が漕げな

いという人が乗ると事故る。なので乗ってはいけない。

3.9　翻訳しづらい文章をやめよう

　なんか言ってるけど、実際のところ何も言っていない、翻訳しづらい文章はやめよう。

第 4 章

誤解を生まないテキストコミュニケーション

4.1　まずは「何の話か」を先に言う

「〇日ヒマ？」は、ヒマと言ったらどうなるのか分からないので返事がしにくい何が欲しいのかの後に、どういう背景でこれを申し出ているのか、自分がどういう立場なのかを添えると、よりよい案を出してもらえることがある

4.2　背景も添えて話そう

「真鯛がほしい」だけだと「ありません」になるけど、「なぜならば今晩鯛しゃぶにしたいから」を添えると「真鯛はないけど、いい金目鯛があるよ」と言われるかも

4.3　「聞いてほしい」のぶつかり合い

こちらの言いたいことを聞いてもらうには、先に相手の懸念を解消してあげないといけない、というケースがあります。

4.4　Slack でのコミュニケーション

4.5　チケットの書き方

このチケットでやること、このチケットのスコープ外とすること、やることになった背景、スケジュール、誰が OK したらリリースできるのか、リリースのタイミングを合わせなければいけない他の何かがあるか

4.6 チケットのサイズはできるだけ小さくする

チケットクローズやリリースは健康によいので、できるだけサイズを小さくして 1 工程が終わったら即閉じられるようにしておこう。

4.6.1 素早く近づける

言いたいことと、「これを言いたいのかな？」と相手が察することを素早く近づける

4.7 見せて選んでもらう

実物を見ないでした議論は、実物を見るとあっさりひっくり返る質問は「どんなのがいいですか？」とオープンで聞かず、現物を用意して「A と B どっちがいいですか？」

4.8 いっぱい書いて Working Out Loud な働き方をしよう

作業が途中でも、みんなの目に触れる times でどんどんアウトプットしていく。実況中継しながら作業したり、分からないことを「これってどういうことだ？」と口に出していく。（文字に、だが）

質問が必要になったら質問すればいいと思うかもしれないが、「やらなくてもいい無駄な工程をやっている」などは、そもそも自分で無駄だと気付いていないので「これって無駄な工程ですか？」という質問も出てこない。よって「あ、こいつ無駄な方向に歩いていこうとしている」が目に見えないと「そっち行かなくていいよ」というアドバイスがもらえない。

https://blog.studysapuri.jp/entry/2018/11/14/working-out-loud

第 5 章

レビューする、レビューされる

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れる方法を紹介します。

5.1　レビュアー（レビューする側）の心がけ

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」のはずです。レビュアーの本来の目的は、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。

5.1.1　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、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、レビューに対する悪いイメージを抱く原因隣、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ような「重要度」を添えています。

- MUST FIX

  - 仕様や名称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。

自分がレビューのときは、前述のとおり「指摘されたら迷わず即直す」という姿勢でいいのですが、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、その文章を書いたレビュイーにあります。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。

5.1.2 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、共通して「もっと良い文章に早く辿り着きたい」、つまり今は 6 階に居るけど、早く完成度 100% の 10 階に行きたい、という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伝わってきたけど、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 じゃないです」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伝わると共に、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）を取り込む[*2]には、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。

これだと as-is に対して、to-be 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！」という気持ちだけ伝えたい、というときは、先ほ

[*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

どのレベル感で Nitpicking であることを書き添えた上で「こういう風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」と気持ちだけ伝えてもいいでしょう。

5.1.3 レビューは『相手のやることを増やす』責任を持って伝えよう

「分かりにくいです」と感想を伝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」がはっきり言えるけれど、理解していない状態で「なにか言わないと」と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はレビュイーの心を折ってしまいます。[*3]

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伝えるか、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。

5.1.4 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」と思ってしまう人がいます。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。よい文章を早く世

[*3] 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しまいます。

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。ライターが書いた元の文章の出来がもともと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「気づいたことはこれだけでした」と伝えてレビュー結果をレビュイーに渡しましょう。

5.1.5 指摘は基本的に「後出しじゃんけん」だと心得よ

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。

モンティ・ホール問題[*4]と同じで、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、基本的に後だしじゃんけんです。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、そもそも有利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！」というフィードバックの言葉を添えれば、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。

5.1.6 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。

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。

[*4] 3 枚の扉のうち 1 枚だけある「当たり」の扉を見つけるときに云々。モンティー・ホール問題とは

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」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。野次を飛ばしたいなら、あなたも選手になって試合に出ましょう。

5.1.7 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あって、その中で既に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！」とは言えないときは、どう直すのか両者間で折り合いが付かず、その結果いつまでもリリースができなくて苦しくなってしまう、といケースです。

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。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。

チームの中で「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 Pull request 内のそのやりとりへのリンクも貼っておきます。そして、Pull request 上は「いったん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」として、さっとリリースへ進みます。

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リリースに進めるようになりました。

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なあなあになって忘れてしまいます。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。

5.2 レビュイー（レビューされる側）の心がけ

5.2.1 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伝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おはの誤りがないか
- 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。

5.2.2 指摘は素直に受け入れる

絵を描いている途中で「あ、ここおかしいな。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」のような、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るんだけれど、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note[*5]がありました。

これを読んだ筆者は「絵もライティングもおんなじだな」と感じました。

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

[*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

トをもらうと、その瞬間、湧き上がる「指摘してもらってありがたいな」という気持ちと、「いやいやこういう風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。みなさんも、覚えがあるんじゃないかな、と思います。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？」みたいな場合に、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、迷ったことはありませんか？ もっと言うと、指摘してくれた人に「こういう意図でこう書いたから、ここはこのまま行きたいです」とか「ここを直すとまた時間がかかっちゃうから、今はこのまま行きたいです」とか、言ったことはないでしょうか？

レビュイーは指摘を聞くと、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りますが、そういうときは指摘を素直に受け入れてさっと直すと大抵その方が文章がずっとよくなります。

指摘されたことを直すか直さないか迷うのはやめて、基本すぐに全部直すようにしてみましょう。

どういうことかというと、この 2 つに時間をかけるのをやめて、指摘されたらさっと直すようにします。

1. 指摘されたことを直すか直さないか迷う
2. なんとか直さず済ませるために相手を説得する

仮に「いやいやこういう意図があってね、あなたの指摘は誤解に基づく指摘なのよ」と思ったとしても、少なくとも 1 人が「この文章を読んで誤解した」というのは事実なので、このまま直さず行けば他の人も同じように誤解してしまう可能性があります。なので、指摘されたことを直すか直さないか迷うのはやめて即直してみましょう。

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% とか 100% に持っていく速度が速くなって、前よりも文章を書くのうまくなりました。

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なと思っていても、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良い文章になることの方が圧倒的に多いのです。

5.2.3 レビュアーが気付いたことに、書いたとき気付かないのは当然

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」「どう伝えるか考え」「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い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、文章をもっと良くすることはできません。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、レビュイーが最初の文章を書き上げたということ、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 6 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について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。

6.1　テクニカルライティングの検定を受けてみよう

IT 系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。

6.1.1　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 検定と呼びたいと思います。

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。試験勉強をした私の理解としては、「文章」だけならテクニカルライティングですが、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ね。

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。

それから 3 級、2 級ときて 1 級もあるにはあるんですが、内容を検討中で開始時期未定……のまま何年も経過しているそうです。なので、筆者のように「テクニカルライティング技術を改めて学んでみたい」というレベルであれば、いったん 3 級だけでよさそうです。

6.1.2 TC 検定はいつどこで誰が受けられるのか

この TC 検定をどこが実施しているのかというと、「一般財団法人テクニカルコミュニケーター協会」です。この日本語スタイルガイド（第 3 版）を出している協会です。

`https://www.amazon.co.jp/dp/4902820102`

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、これから勉強したい、という方にもおすすめです。

IT 系の資格試験は日にちを選ばずに好きなタイミングで受けられるものも多いですが、このテクニカルライティング試験は毎年夏と冬に 1 回ずつ、年 2 回だけの開催です。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 6.1）

▼表 6.1　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）

申し込み受付はだいたい開催の 2 ヶ月前から始まって、開催の 1 ヶ月以上前には〆切となります。ぎりぎりになってから受けたいと思っても、間に合わないので注意してください。

TC 検定の受験料は 15,840 円です。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。直近の試験は、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できました。ちなみに会場には、事前に郵送されてくる受験票に証明写真を貼って持っていかねばなりません。

試験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。

試験は大きく分けて 2 つあります。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。

で、公開されている選択問題の例題がこちらです。適当に 2 つ、抜粋してきました。

問 2 は、主語と述語の対応が不適切なもの、アが主語と述語が噛み合っていないので、アですね。問 5 は、二級品はもうそういう熟語だし、四角いも形容詞、十数日は概数、イだけ「3 日以内」で、3 を任意の数字、例えば 8 日以内とか、14 日以内とかに置き換えても通用するので、イは算用数字にできます。

と、こんな感じの問題がでます。公開されている例題は、こういう「もともと日本語の読み書きが得意な人なら、感覚でいけるだろ！」という問題が多いんですけど、実際の試験はこのレベルよりもっと難しかったです。

次、後半の記述問題！　こっちもね、50 分間で回答欄が 30 個くらいあるんですよ。みなさん、最後に 50 分間、一生懸命文字を書き続けたのはいつですか？　こっちも例題があって、見てもらうと、この例題は割と書く量が少ないんですけど、実際の試験だとかなり書く量が多かったです。

結構な量の回答欄を埋めないといけないのでめちゃくちゃ手が痛くなるし、この文

とこの文を入れ替えたいと思っても、手書きなのでコピペでシュッと入れ替えられないんですよ！ あと枠の中に全然文字が収まらない！ こっちの記述問題の方は、私は全部書き上げるのがぎりぎりで、清書するような時間の余裕はなかったです。終わった後、2 日くらいお手々が痛くて、もうキーボードを叩く以外の方法で手を出力デバイスとして使うのはむりーって思いました。

難易度についてまとめると、個人的には、総じて「思ってたより難しかった〜」という印象でした。

今回、チームからは私を含めて 3 人が受験したんですが、全員「もうだめだ…また冬にがんばる…」みたいになってて、「これはよゆーでしょ？」という感じでは全然なかったです。

6.1.3 勉強方法は？

次、勉強方法は？

テクニカルライティング試験の教科書、参考書にあたる本はこれ一冊、「日本語スタイルガイド（第 3 版）」だけです。

https://www.amazon.co.jp/dp/4902820102

出題範囲はすべてこれに含まれているし、実はこの本の巻末に試験範囲の説明や、例題も載っています。なので、試験勉強がそのままテクニカルライティング技術について学ぶという形になります。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。

「日本語スタイルガイド（第 3 版）」に掲載されている例題は、テクニカルコミュニケーター協会のウェブサイトでも公開されています。私はこれを印刷して、何回か解いてみました。

▼択一選択問題 https://www.jtca.org/publication/jsg_book_reidai.pdf ▼記述問題 https://www.jtca.org/certificate_exam/jsg_book_reidai.pdf

それから試験の少し前に、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。セミナーの受講料は 8,800 円です。こちらは「一般財団法人テクニカルコミュニケーター協会」の法人会員または個人会員だと半額になるそうです。

このセミナー、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、など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。

なので、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だと思って、セミナーは受けた方がいいのではと思います。私はぎりぎりにならないと頑張れないタイプなので、このセミナーを受けてから勉強を頑張り始めました。

6.1.4 合否はすぐ分かる？

次、合否はすぐ分かるか。

そんな感じで一生懸命勉強して、日曜の午後を丸々費やして試験を受けるんですが、結果はすぐには分かりません。試験からおおよそ 1 ヶ月後にウェブと郵送で合格発表が行われるので、それまでじりじりしながら待ちます。郵送で届く結果に、選択問題と記述問題、それぞれ自分が 100 点満点中、何点だったかは書いてあります。でも問題文は回収されるし、正しい答えも特に発表されないので、結果が届く前に自己採点はできません。

なので私もいま合否が分かってません！ 8 月末に結果が届いて落ちてたら、来年 2 月にみなさんとまた試験会場で会えると思いますw

6.1.5 合格率は？

次、合格率。

3 級の問題は、合格率は 5 割を目指して作問しているそうで、前回（2022 年 2 月）の試験の合格率は 58% でした。

合格ラインは明らかにされていないので、はっきりとは言えないんですが、過去、合格された方のブログ記事を見ていると、選択問題と記述問題が両方ともだいたい 70 点を超えていれば合格している、という傾向だなと感じました。

6.1.6 受けてみて得たものはあった？

最後、受けてみて得たものはあったか？

さっき、結構難しかったのであんまり自信ないなーという話をしたんですが、じゃあ不合格だったら受けなきゃよかったかというと全然そんなことはなくて、受かっても落ちても学んだことは消えないのでヨシ！ という感じでした。

実際、受験勉強していたこの 1 ヶ月くらいは、他の人が書いたドキュメントをレビューするときに「あ、ここ緑の本で出てきたところだ！ あの書き方が使えるぞ！」みたいな進研ゼミ現象が結構起きてました。それからレビューで「こう直した方がいいのでは？」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する〜」ではなく、ちゃんとこういう理由でこう書いた方がいい、という根拠を付けて修正提案を説明できるようになったのも、いいところだなと思います。

テクニカルライティング技術は、ドキュメントだけじゃなくて、企画書とか仕様書とか設計書とか、あとは社内での報告書、手順書、告知、Slack でのやりとりなどなど、色んな実用文で役に立ちます。なので、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ーとか、文書の構造化とか階層化の手法を学びたいなーという方にはとってもおすすめです。

試験の詳細は LINE Technical Writing Meetup vol. 15 というイベントでお話ししましたので、なにそれ？ テクニカルライティングの試験なんてあるの？ どんな試験なの？ という方はこちらのスライドと動画をご覧ください。

https://speakerdeck.com/line_developers/my-story-about-taking-the-technical-writing-exam

https://www.youtube.com/watch?v=N9OS4BtDHik&t=856s

公開されている分析データによると、今回の合格率は 71% だったようです。ひとつ前の回（2022 年 2 月）は合格率 58% だったので、私の「想定より難しかった…」という体感とは異なり、実際は従来より問題が易しめだったのかもしれません。（あるいは受験者のレベルが軒並み高かったとか）

そして 2022 年 9 月 2 日 (金) に、郵送で合否判定通知書と合格証書も届きました。合否判定通知書に載っていた点数は選択問題が 92 点、記述問題が 87 点だったので、なんとか面目を保てる点数でほっとしました。（毎日「どうもどうも！ テクニカルライターです！」という顔で仕事をしているのに落ちたらまじで洒落にならんと思っていた）

6.2 色んな本を読もう

近年、技術の 1 ジャンルとして確立されてきたのか、テクニカルライティング技術を題材にした本がたくさん出版されるようになりました。

6.3 「書き方」の指標を見つける

文化庁が 2022 年 1 月に出した「公用文作成の考え方」という資料がある。

「公用文作成の考え方」について（建議）https://www.bunka.go.jp/seisaku/bunkashingikai/kok

公用文作成の考え方（建議）（PDF）https://www.bunka.go.jp/seisaku/bunkashingikai/kokug

6.4 Technical Writing Meetup に参加しよう

テクニカルライティングをテーマにした Meetup を開催しているので、ぜひオンラインで参加しよう。

過去のアーカイブ動画も見られるよ。

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！ 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[*1]です。

さて、東京ディズニーランドにおいて、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[*2]」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、部屋のお風呂でじゃぐちをひねると温泉が出るヴィンテージのリゾートマンションです。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。わいわい。たのしいね。

2024 年 5 月
mochikoAsTech

[*1] 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よ！ https://cobalt.shueisha.co.jp/contents/tyoutokyo/

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

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:

- 「だめだよ」と言われると「かわいいのに？？」という顔をするねこ

レビュアー

- やさしい誰か

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`

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