技術をつたえるテクニック

技術書執筆から登壇までをサポート

mochikoAsTech　著

2019-04-14 版　　mochikoAsTech 発行

はじめに

2019 年 4 月 mochikoAsTech

本著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。「分かりやすくつたえる技術」筆者、mochikoAsTech です。

突然ですが私は「教えたがり」です。どれくらい教えたがりかというと、ただ好きだからというだけで「DNS をはじめよう」や「AWS をはじめよう」という初心者に向けてインフラを教える本を書いてしまうくらいの教えたがりです。同僚が「困ったなー、分からないなー」という顔をしていると「ふははは、解説しよう！！」としゃしゃり出ていきたくなるし、「ちょっと分からないことがあって教えてもらえますか？」などと言われた日には鼻息荒く「いつでもいいですよ！ なんなら今でも！」と言ってしまいます。

有難いことにそんな筆者が書いた「DNS をはじめよう」や「AWS をはじめよう」は「優しい先輩が隣で解説してくれてるみたいですごく分かりやすい」という評価をいただき、なんと半年ちょっとで 4,500 冊も買ってもらうことができました。もっとインフラを分かるようになりたい！ と思っていたたくさんの人に私の好きな技術を詰め込んだ早口な説明が届いて、少しでもお役に立てたのだと思うととても嬉しいです。

自分で言うのもおこがましいのですが、筆者は分かりやすくつたえるための文章を書いたり、相手の理解度にあわせて説明をすることが比較的得意です。前職でハンズオンセミナーをやったところ延べ 200 人以上が参加して、終了後の満足度アンケートで平均 4.9 点（5 点満点）をいただけた程度には得意です。

これらの経験から「どうやったらあんなふうに分かりやすく説明できるの？」と聞かれることが多かったため、本著では「DNS をはじめようや AWS をはじめようを書くとき、こんなことに気を付けて書いたら分かりやすいって言ってもらえたよ！」という技術を分かりやすくつたえるテクニックを詰め込みました。

筆者は文章を紡ぐ専門家ではなく、辞書の編纂者でもなく、ただの本好きなひとりのエンジニアです。個人的な「こんなことに気をつけて文章を書いたり教えたりしています」という経験則がどれくらいお役に立てるか分かりませんが、よりよい技術書が生まれてくる際のお手伝いが少しでもできたら嬉しく思います。

想定する読者層

本著は、こんな人に向けて書かれています。

- 技術を分かりやすく伝えられるようになりたい人
- 技術的なことを説明するのが下手だと思う人
- 社内の技術勉強会で講師をやる人
- エンジニア向けの技術書を書く人
- 技術ドキュメントの校正や添削をする人
- 技術系ポッドキャストで上手くしゃべれるようになりたい人
- 新卒研修や後輩の指導を任された先輩
- ゆるふわ愛され教わり上手になりたい後輩

本著の特徴

本著では次のどちらにも対応して「技術を分かりやすくつたえるテクニック」をまとめています。

- 第 1 章：技術を文章で分かりやすくつたえる
  - 技術書やドキュメントを書くときの「もっと分かりやすくてつたわる文章が書きたい！」という気持ちに応える章
- 第 2 章：技術を登壇で分かりやすくつたえる
  - 勉強会や研修で登壇するときの「もっと技術を分かりやすく説明できるようになりたい！」という気持ちに応える章

ありがちな「悪い文章の例」を挙げて、どこが分かりにくさの原因なのかを説明した上で「改善した文章の例」を提示していますので、業務や勉強会などで学んだことをすぐに実践できます。

また教える側だけでなく教わる側にもスポットライトを当てて、技術を学んでより成長するための聞き手としてのテクニックも記載しています。

- 第 3 章：教わり上手をはじめよう
  -「短期間でいっぱい成長するにはどんなことに気をつけて勉強すればいいの？」という気持ちに応える章

本著のゴール

本著を読み終わると、あなたはこのような状態になっています。

- 読者に伝わる文章が書けるようになっている
- 相手の理解度にあわせて分かりやすく説明できるようになっている
- 教えるときはどんなことに注意すべきか理解している
- 原稿を書くことや登壇が前より怖くなくなっている
- 「知らない」「分からない」を素直に言えるようになっている

免責事項

本著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。

また本著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。よって本著の記載内容に基づいて読者が行った行為、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/startWriting`

目次

第１章

技術を文章で分かりやすくつたえる

勉強会で講師役を務める、ドキュメントを書く、イベントで登壇する、後輩に実装方法を教える、そして技術書を書く。私たちは色んな場面で技術を「つたえて」います。

この章では技術書やドキュメントを通して「文章で分かりやすくつたえる」際に、こんなところに気をつけるとより分かりやすくなりますよ、というお話をします。

1.1 読者層をはっきりさせよう

万人に最適な説明はありません。つたえる相手が変われば最適なつたえ方も変わります。

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、書くべき内容やつたえ方は大きく異なります。

誰に向けて書いている文章なのか？ を最初にはっきりさせておかないと、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…！」と破綻したり、あるいは読者に「こんな簡単なことはもう知ってるからもっと踏み込んだ内容が読めると思ったのに」と不満を持たれたりします。

読み終わってから「思っていた内容と違った…」という残念な思いをさせないよう、ミスマッチを防ぐ次のような対策をしておきましょう。

1.1.1 想定する読者層とゴールを書いておこう

技術書やドキュメントの場合、本文の前に「想定する読者層」と「ゴール」を書いておきましょう。たとえば「AWS をはじめよう」では想定する読者層を次のように定義していました。

> 本著は、こんな人に向けて書かれています。
>
> * AWS が何なのかよく分かっていない人
> * ブログやポートフォリオサイトを独自ドメインで作ってみたい人
> * JavaScript や HTML や CSS なら書けるけどサーバは分からなくて苦手という人
> * プログラミングの勉強がしたいけど環境構築でつまづいて嫌になってしまった人
> * 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の人
> * インフラやサーバになんとなく苦手意識のある人
> * AWS、EC2、RDS、ELB、Auto Scaling、Route53 などの単語に興味がある人
> * クラウドってなんだろう？ サーバってなんだろう？ という人

想定する読者層を書くことで、ここに当てはまらない人は「自分向けの本ではないな」と判断できるのでミスマッチが減らせます。同様に「AWS 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。

本著を読み終わると、あなたはこのような状態になっています。

* WordPress のおしゃれなサイトができあがっている
* 使うも壊すも自由な勉強用の Linux サーバ環境が 1 台手に入る
* クラウドがなんなのか？ や、そのメリットデメリットが説明できるようになっている
* 読む前より AWS やサーバや黒い画面が怖くなくなっている

このようにゴールを書くことで「読むことで何が得られるのか」を把握できます。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、読む目的は人によって異なります。読者層と同様、ゴールを提示しておけばやはりミスマッチを減らせます。

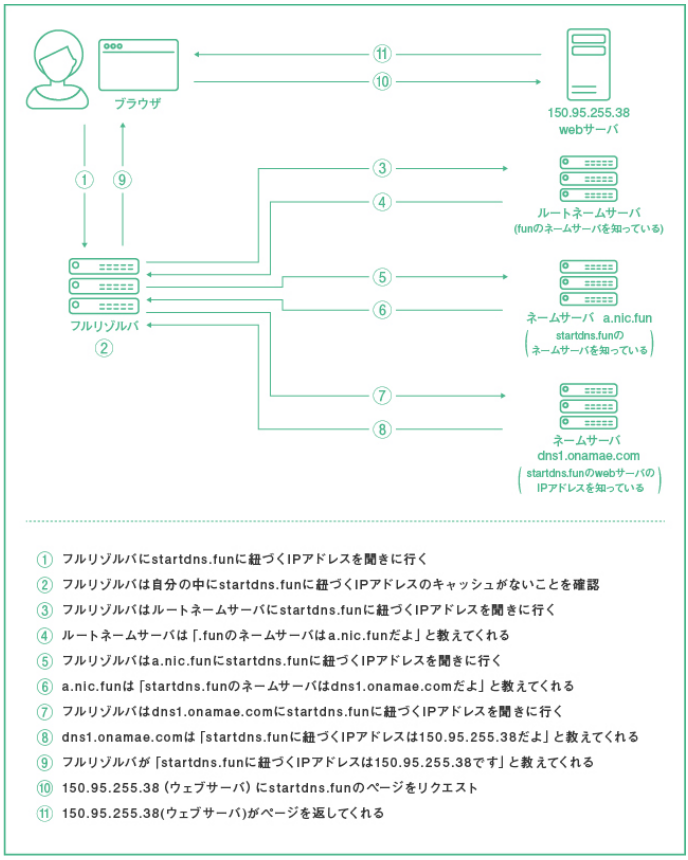
1.2 「すんなり入ってくる」ための工夫

1.2.1 分かりやすい例は理解を促す

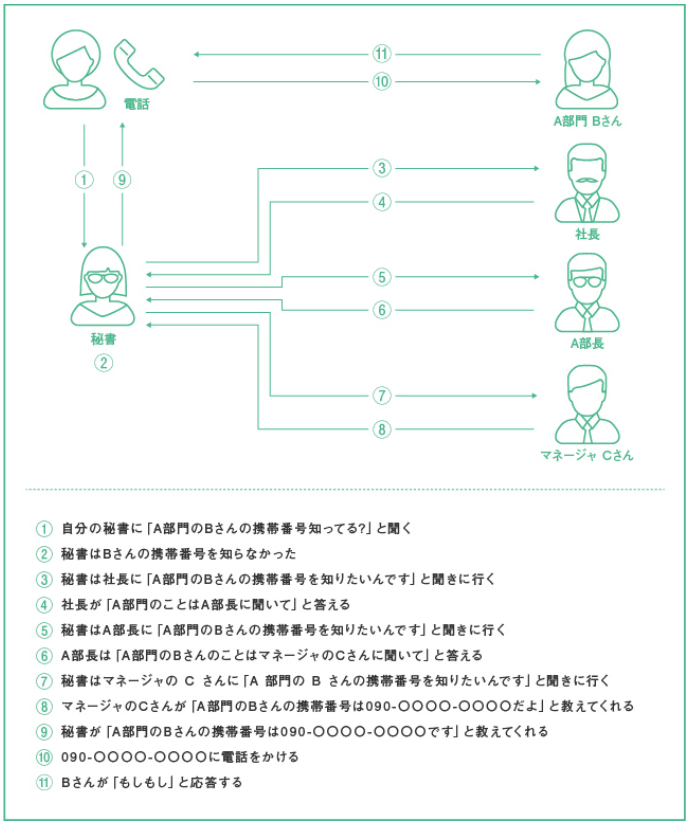
難しい事象を説明するときは、たとえば「dig コマンドはスプラトゥーンのブキチのように解説が長い[*1]」というように、読者にとって身近な物事に置き換えたたとえ話を出すとぐっと分かりやすくなります。

次のように技術的な事象とたとえ話の図を並べるのもよい方法です。

[*1] 気になる方は「DNS をはじめよう」の P102 を参照ください。https://mochikoastech.booth.pm/ にダウンロード版があります。



▲図 1.1　ウェブページが表示されるまでの名前解決の仕組み



▲図 1.2　B さんに電話をかけるまでの流れ

結城浩さんの数学ガール[*2]という本には「例示は理解の試金石」というフレーズがたびたび登場します。よい例を作れれば自分はその事象をきちんと理解できているし、作れなければ理解が足りていない。例示を作れるか否かが理解度を判定する材料になる、ということです。

確かに思い返してみると、技術を誰かに説明しようとしても上手い例が出てこないとき、あるいは作った例に矛盾が生じてしまうようなときは、まだまだ自分自身の理解が足りていない、いわゆるエンジニア用語の「チョットデキル」よりずっと手前の

[*2] https://www.hyuki.com/girl/

「完全に理解した！」[*3]という時期でした。

自分の理解が足りないのに人に分かりやすく説明する文章は書けません。例を書いてみることで自分の理解度を試してみましょう。

1.2.2 なんで？ のコップが準備できてから説明しよう

文章で何かをつたえたいときには、まずは読者が一度失敗したり、疑問を抱いたりして「なんで上手くいかないんだろう？」「なんでこんなことするんだろう？」という気持ちにならないことには、いくらつたえても知識が上滑りしてしまいます。ですのでいきなり説明から入らず、**疑問を抱かせてから説明する**という流れを意識しましょう。

たとえば極寒の屋上で 1 時間吹きっさらしになった後だからこそ、暖かい毛布のありがたみが分かるのであって、ずっと真夏のビーチで過ごしている人に「すごく暖かいんだよ！」と毛布の良さを説いたところで「言ってることは分かるけど…」という感じで実感を伴った理解には至りません。それと同じで、まだ何にも困ってないのに「解決策だよ！ これがベストなやり方だよ！」と立て板に水の説明をされても、聞かされた知識があまり染み込んでこないのです。

中身を求めている空のコップがなければ、いくら知識を注いでもだーだーとこぼれるばかりで入っていきません。ですので「なんで？」という空のコップを用意してもらって、それから知識を注いで満たすという順番が大切なのです。

1.3 一度にたくさんのことはつたわらない

技術を文章で分かりやすくつたえよう！ と思ったとき、やってしまいがちな失敗に「一度に多くのことをつたえすぎる」というものがあります。

本当につたえたかったことがその他大勢に紛れて見えなくなってしまわないよう、次のことに注意しましょう。

1.3.1 つたえることは 1 つにしぼる

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「主人公のあの人格好良かったよねー…えーっと主人公…あー、名前なんだっけ…？」みたいなことはざらです。

あれもこれも書きたくなってしまうのですが、いま書いているこの 1 つの段落でつたえたいことは何か？ 1 冊の本を通してつたえたい大きなメインテーマは何か？ を

[*3] Linux の開発者が「ワタシハリナックスチョットデキル」と書かれた T シャツを着ていたのが起源のジョーク。 https://twitter.com/ito_yusaku/status/1042604780718157824

考えて、つたえたいことは常に 1 つだけにしぼりましょう。ここでしぼりきれないと読者は「結局何が言いたかったの？」という気持ちになります。

1.3.2 できるだけシンプルに。余計な情報は出さない

「A は B でもあり、C のときもあるけれど、まれに D のケースもある。E の可能性も捨てきれない。ですがいったんここでは A は B、とだけ覚えておいてもらえれば OK です」

こういう説明、誰しも一度は読んだり聞いたりしたことがあると思いますがお勧めしません。覚えなくていいならそもそも書くべきではありません。余計な情報はなくしてできるだけ説明はシンプルにしましょう。

実際に C や D や E のような例外があるとしても、それは「A は B」をきちんと理解できた後で改めて説明すればいいことです。前述のように冗長な説明をするくらいなら、この時点でははっきり「A は B です」とだけ書くようにしましょう。

1.4 事実は時代とともに変わる

事実は時代とともに変わります。たとえば大昔、Linux の環境で nslookup コマンドをたたくと「このコマンドは非推奨だし、将来的には廃止されるから今後は別のコマンドを使ってね」という警告メッセージが表示された時期がありました。その時点では「nslookup コマンドは非推奨なので使わない方がいい」は確かに事実でした。

ですがその後、Bind9.9.0a3 がリリースされたタイミングでその警告メッセージは消え、リリースノートには「nslookup を非推奨として扱うのはもうやめるね。非推奨の警告も消したよ」と書かれました。ですので、現在は「nslookup コマンドは非推奨なので使わない方がいい」は事実ではありません。[*4]

このように「○○は使わない方がいい」「○○では××はできない」などの事実は、ミドルウェアやソフトウェアのバージョンアップに伴って状況が変化し、事実でなくなってしまうことが往々にしてあります。

状況の変化は避けられませんので、変化に対応するため次のように対策を取っておきましょう。

[*4] この nslookup にまつわる話の詳細は「DNS をはじめよう」という本の P103 に載っています。気になる方は `https://mochikoastech.booth.pm/` からダウンロード版をどうぞ。

1.4.1 年月日やバージョンを書いておこう

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。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「4 月 14 日の技術書典」ではなく「2019 年 4 月 14 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、どのバージョンを想定した内容なのかも記載しておきましょう。

1.4.2 一次ソースに当たろう

前述の「nslookup コマンドは非推奨なので使わない方がいい」のような後から状況が変わったケースでは、ネットで個人ブログや Qiita の記事が出てきても、個々の記事には「書いた時点」の情報が断片的に載っているだけなので、現在までの経緯を見渡した正確な情報にはなかなかたどり着けません。

特に本を書いて人に技術をつたえるときは「検索して出てきたブログに○○と書いてあった」「以前、他のエンジニアに○○と教えてもらった」のような伝聞をそのままソースにするのではなく、きちんと一次ソースに当たるか自分で動作確認をしましょう。絶対に間違えないことは無理であっても、間違えないよう努力することは教える側の誠意だと思います。

どうしても一次ソースが見つからない場合はいっそ書かない、あるいは「一般的にこう言われているが正確なところは調べても分からなかった」と正直に書くべきです。筆者も Bind のリリースノート（英語）を追いかけていって該当の記述を見つけるまでは「この nslookup の話書きたいけどソースがないと書けない…！」と必死でした。

1.5 読みやすさを支える正しく統一された表記

どんなに内容がよいとしても表記にばらつきがあったり、誤字脱字が多かったりすると読者は「読みにくいな」「内容はちゃんと合ってるのかな…？」と品質に不安を抱いてしまいます。

読みやすさを下支えする次のような点に注意して文章を書いてみましょう。

[*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。本著にもあります。

1.5.1 正しい名前で書こう

ソフトウェアなどの名称は自分がなんとなく使っている略語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 1.1）

▼表 1.1 略称や誤記ではなく正しい表記で書く例

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

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「別にいいですよ」と言いつつ決していい気分はしません。[*6]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、次のようにはじめは正式名称で紹介して、以降は略語にする形もよいでしょう。読み方が分からずにひそかに悩んでしまう[*7]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。
```

細かな表記が間違っていると全体の信頼度も下がります。正しい表記を心がけましょう。

[*6] 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笑顔の裏で「全社員とはいわないけど、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…」と思ったあの日。相手に興味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。

[*7] 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。

1.5.2 ひとつのものはひとつの名前で書こう

たとえばひとつの説明の中で同じものを「ターミナル」と書いたり「RLogin」と書いたりして表記にばらつきがあると、読者は「RLogin…？ ああさっきインストールしたターミナルのことか」と脳内で辞書を引くことになります。それに「敢えて別の名前で呼んでいるってことは実は違うものを指しているのかな？」と不安になってしまいます。

このように表記ゆれは読者の脳内メモリを無駄に食います。本当につたえたいことに集中してもらえるよう、同じ単語を 2 通り以上の表記で表すのはやめましょう。

一般的に見ても色んな名前で呼ばれているような場合は「A や B という名前で呼ばれることもあるけれど、本著では統一して C と呼びます」のように明示しておくと、他の文献を読んだ際に理解がつながるのでさらに親切です。

1.5.3 英語？ カタカナ？ 最後の伸ばし棒はある？ どこまでひらく？

「サーバ」と「サーバー」と「Server」のように、声に出して読んだときは同じでも、文字だと英語で書くのかカタカナで書くのか、最後の伸ばし棒はあるのかないのか、先頭は大文字なのか小文字なのか表記のゆれる単語もあります。

またサーバは「立てる」なのか、「建てる」なのか[*8]のようにどの漢字を用いるか、という問題。さらに「たとえば」と「例えば」、「はじめて」と「初めて」のようにどこまでの漢字を平仮名に「ひらく」のか、という問題もあります。

これらは「こちらの方が読みやすい」「自分がこうすべきだと思う」といった考え次第ですので、最初にどの方針で書くのか決めてしまいましょう。

1.5.4 表記ゆれのチェックはツールでやろう

名前や書き方の方針が決まったら、表記ゆれを防ぐためにはただ頑張るよりもツールを使った方が確実です。たとえば Re:VIEW[*9]を使って Visual Studio Code[*10]で

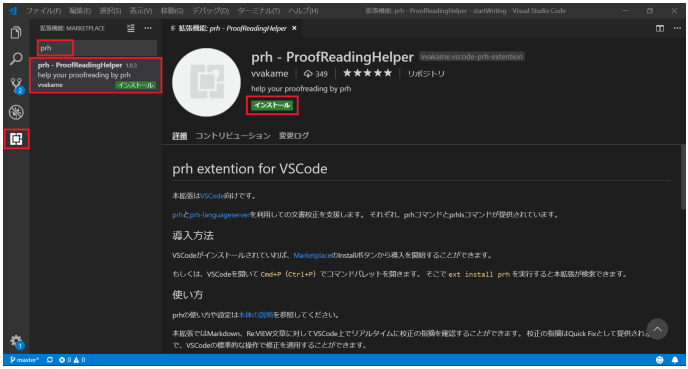
[*8] ちなみに筆者は「立てる」派です。「AWS をはじめよう」という本の P100 にあるコラムでその理由を説明しています。https://mochikoastech.booth.pm/

[*9] Re:VIEW（リビュー）は簡易なマークアップを付与したテキストの原稿を書くと、コマンド 1 つで書籍として組版された PDF や EPUB が生まれてくる夢のようなツール。本著も Re:VIEW とデザイナーさんのパワーで生まれています。詳しく知りたい方は「技術書をかこう！ ～はじめての Re:VIEW～ 改訂版」という書籍がおすすめ。 https://techbooster.booth.pm/items/586727

[*10] Windows でも Mac でも無料で使える Microsoft のコードエディタ。本著も Visual Studio Code で書いています。 https://azure.microsoft.com/ja-jp/products/visual-studio-code/

原稿を書く場合は vscode-prh-extention[*11] という拡張機能が便利です。

Visual Studio Code の拡張機能で「prh」を検索すると、「prh - ProofReadingHelper」という拡張機能が表示されるので「インストール」をクリックします。（図 1.3）



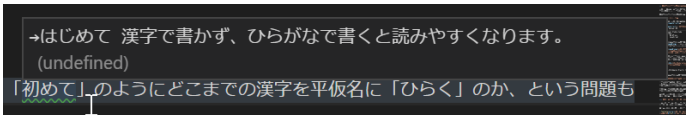
▲図 1.3　Visual Studio Code の拡張機能を prh で検索してインストール

あとは原稿の.re ファイルと同じフォルダに「prh.yml」というファイルを用意して、そこに期待値（expected）や誤りのパターン（pattern）といったルールを書いておきましょう。

```
- expected: はじめて
  pattern:  初めて
  prh: 漢字で書かず、ひらがなで書くと読みやすくなります。
```

すると次のように Visual Studio Code 上で文章の下に波線が出て表記ゆれを指摘してくれます。（図 1.4）

[*11] 技術書典の運営でお馴染みわかめさん謹製。 https://github.com/prh/vscode-prh-extention

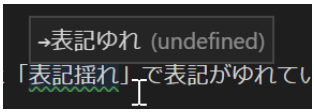


▲図 1.4　「初めて」に波線が出て表記ゆれを指摘してくれる

TechBooster が無償で公開してくれている Re:VIEW 用の書籍テンプレート[*12]を使えば、原稿が入っている articles フォルダの中にすでに prh.yml も配置されています。ちなみに筆者もここまで書きながら「表記ゆれ」と「表記揺れ」で表記がゆれていたので、「ゆれ」は平仮名にしよう！ と決めて prh.yml に次のように書き足しました。

```
  - expected: 表記ゆれ
    pattern:  表記揺れ
```

ルールを書き足したことで、次のように表示されて表記ゆれにすぐ気づけるようになりました。(図 1.5)



▲図 1.5　prh.yml にルールを追加したら「表記揺れ」にも波線が出るようになった

1.6　迷子にならない文章にしよう

読者はあなたが書いた道案内に従って「新しい技術」という道を歩いていきます。ですので道案内には書かれていない分かれ道が現れたり、道が忽然と消えてしまったりしたら、どうすればいいのか分からずに迷子になって立ち尽くしてしまいます。

読者が迷子にならないように次のことを心がけましょう。

[*12] TechBooster というサークルが公開している Re:VIEW 用の書籍テンプレート。 https://github.com/TechBooster/ReVIEW-Template

1.6.1 はじめて出てきた言葉は説明しよう

いきなり知らない言葉が出てきたのに、その言葉を知っている前提で話が進むと読者は困ってしまいます。書きながら初出を見つけるセンサーを働かせて、その言葉がはじめて出てきたときはきちんと説明をしましょう。

1.6.2 言葉を定義しよう

「商業出版」のように一般的な言葉であっても、人によって「何を持ってして商業出版と言うのか？」という定義が異なる言葉もあります。たとえば「書店に並ぶ本はすべて商業出版だ！」という人もいれば、「Amazon で売られていれば商業出版だ」あるいは「出版社が発行すれば書店や Amazon に並ばずとも商業出版だ！」という人もいるでしょう。

技術用語に限らず、一般的な言葉であっても人によって解釈が異なる場合は「本著では商業出版を○○と定義します」のように説明しておくとよいでしょう。

1.6.3 「それ」ってどれのことですか？

「それを再びクリックして」「そのボタンを」のように「それ」「あれ」「この」といった指示語を乱用すると、読者は「それってなんだろう？」と迷うことになります。

次の悪い例を見てみましょう。

> 準備が出来たらセキュリティグループ名と概要を記入して「ルールの追加」をクリックします。
> もしそれを行った際にエラーが発生したら…

「それ」は何を指しているのでしょうか？ 次のいずれとも読めるため読者を迷わせてしまいます。

- セキュリティグループ名と概要を記入すること
- 「ルールの追加」をクリックすること
- セキュリティグループ名と概要を記入してから「ルールの追加」をクリックすること

前述の場合は「それ」という指示語ではなく、もう一度対象を書くことでつたえたいことがより明確になります。

> 準備が出来たらセキュリティグループ名と概要を記入して「ルールの追加」をクリックします。もし「ルールの追加」をクリックした際にエラーが発生したら…

何度も同じ言葉を繰り返すとかえって読みにくくなることもありますので程度の問題ですが、むやみに指示語を使って読者を迷子にさせていないか、「それ」や「この」などの指示語で検索して文章を読み返してみましょう。

1.6.4 幾通りもの解釈ができると迷ってしまう

次の文章を読んでみてください。

> 面接の場でペアプロをやったら楽しかった話をしたら聞いていた人が「わかる」と頷いた。

これは果たして次のうちどちらの意味でしょうか？

- 「面接の場でペアプロをやった。楽しかった」という話を、後日他の人に話したら「わかる」と頷かれた
- 「ペアプロをやったら楽しかった」という話を、面接会場で話したら面接官が「わかる」と頷いた

幾通りもの解釈ができる文章は読者の誤解を招き、理解を誤った方へ導いてしまいます。つたえたいのが後者の意味であるなら次のように直してみましょう。

> 「ペアプロをやったら楽しかった」と面接の場で話したら、聞いていた面接官が「わかる」と頷いた。

「面接の場で」を「話したら」の直前に持ってきたことで、ペアプロの話をどこで話したのかが明確になりました。

- 「ペアプロをやったら楽しかった」と話した
- 面接の場で話した

のように「話した」を修飾する言葉が複数ある場合、**長い修飾語や重要な修飾語を前にして、短い修飾語や重要でない修飾語を後に**した方が分かりやすくなります。

また「聞いていた人」という抽象的な表現から「面接官」に変えたことで、話の聞き手が誰なのかもはっきりしました。このように前後を入れ替えたり、動詞に対応する主語を明確にしたりして、読者を迷わせないようにしましょう。

1.7　できるだけ簡潔にしよう

突然ですが質問です！ 技術を説明する 100 文字の文章と 10,000 文字の文章があります。どちらでもまったく同じ内容がつたわるとしたら、どちらを読みたいですか？

うわ、びっくりした！

お隣のわかばちゃん[*13]がちょうど通りがかったタイミングで答えてくれました。そうですね。楽しむことが目的の小説であれば長短は個人の好みによりますが、技術の習得や問題解決が目的の技術書であれば、わかばちゃんの言うとおりできるだけ「早く」「楽に」読める方がいいですよね。怠惰なのはエンジニアの美徳のひとつです。

丁寧に説明しようとするのはいいことですが、結果として冗長になっていないか次のような点をチェックしてみましょう。

1.7.1　「ということ」「することができる」は必要？

次の文章を読んでみてください。

> 筆者が文章を書くときに心がけているのは、できるだけ余計なものをそぎ落として簡潔にならないかな、ということです。

なんということでしょう！「簡潔にしようと心がけている」と言っている割に文章が冗長です。たとえばこの文章は前後を入れ替えると簡潔で分かりやすくなります。

[*13] 「マンガでわかる Docker」や「わかばちゃんと学ぶ Git 使い方入門」などでおなじみのわかばちゃん。本著を頒布する技術書典 6 ではお隣のあ 03 が「湊川あいの、わかば家。」なので、通りすがりにちょっぴり参加してくれました。「マンガでわかる Ruby」も楽しみです！　※湊川あいさんから直接許可をいただき掲載しています

筆者が文章を書くときは「できるだけ余計なものをそぎ落として簡潔にしよう」と心がけています。

同様に「行動に移すときがきたということを知った」よりも「行動に移すときがきたと知った」、「説明することができる」よりも「説明できる」の方がずっと簡潔です。**「ということ」や「することができる」と書いたときはそれが本当に必要なのか見直してみましょう。**

1.7.2 コップからあふれる長い文章は切ろう

文章を読むとき、「。」で一区切りがつくまで、私たちはその文章をコップに溜め続けています。

コップに一口分の文章を注がれ、それを飲んでコップを空にする、を繰り返すことで読者は文章の意味するところを理解し記憶の引き出しにしまい込みます。

ですが「。」で一区切りがつかず、延々とコップに文章を注ぎ続けるとそのうちコップから文章がこぼれてしまいます。あふれてこぼれてしまった分は飲むこともできません。長すぎる文章は記憶の引き出しにしまい込む前に一部を消失してしまうのです。

長すぎる文章の例を読んでみましょう。

データセンターは「物理的な攻撃や侵入」からサーバを守るための設備を整えており、堅牢さはデータセンターによって異なりますが、たとえば「所在地を一般に公開しない」「建物自体に侵入経路となる窓がない」「入るときと出るときで体重が違うと出られない」といったような防犯対策が一例として挙げられ、この入退出時の体重チェックは盗んだハードディスクを持ち出せないようにするためのものです。

長い…！ 実際に声に出して読んでみると分かりますが、この長さだとひといきでは読めず途中で息が切れてしまいます。あくまで目安ですが**ひとつながりの文章で 50 文字を超えたら「。」で区切りましょう。**前述の文章を適切な長さで区切ると次のようになります。

データセンターは「物理的な攻撃や侵入」からサーバを守るための設備を整えています。
堅牢さはデータセンターによって異なりますが、たとえば次のような防犯対策が例としてあげられます。

* 所在地を一般に公開しない
* 建物自体に侵入経路となる窓がない
* 入るときと出るときで体重が違うと出られない

ちなみに入退出時の体重チェックは、盗んだハードディスクを持ち出せないようにするためのものです。

文章を適切な長さで区切り、かつ列挙していたものを箇条書きにしたことでぐっと読みやすくなりました。ひとつながりの文章をコップからあふれるほど長くしすぎないようにしましょう。

1.7.3 自信がないからといってぼかさない

「恐らく」「と思われる」「らしい」「など」「やや」「おおよそ」「少し」のように、つたえたいことをぼかして不明確にしてしまう言葉は安易に使わないようにしましょう。

そうした言葉が自然と入ってしまう箇所は「一次ソースの確認が出来ていない」「実は筆者もしっかり理解できていない」というように、書いている側が自信を持って説明できない部分であることが多いです。筆者も身に覚えがあります。

断言されておらず、ぼかされていると「断言されていないということは何か例外的なケースがあるのかな？」と読者に余計な想像をさせてしまいます。自信が持てないときはつたえたいことをぼかしてごまかすのではなく、さらに調べるか、こういう事情で断言できないという理由を併せて書いておきましょう。

1.8 推敲は文章の品質を上げる

文章は書いたら書きっぱなしで終わりではありません。次のような視点で読み返して文章の品質をあげていきましょう。

1.8.1 文章はたくさん撫でるとつやつやになる

文章は猫と同じでたくさん撫でるとつやつやになります。

書いた文章を頭からしっぽの先まで撫でるように読んでいくと、必ずざらっとしたところやささくれたところが見つかるので、そこを直してやってまた頭からしっぽまで撫でて…を繰り返しましょう。一度もざらっとせずにしっぽの先までたどり着けたら文章の毛づくろいは完了です。

但しこの方法だとどうしても頭の方をたくさん撫でることになるので、時間が足りないと頭だけつやつやでしっぽはぼさぼさになってしまいます。全身よい毛並みの文章にできるよう、最初にスケジュールを引く時点で推敲のための時間をしっかり確保しておきましょう。

1.8.2 仮想の読者とリアルな読者

筆者が「渾身の出来！」と思っても、想定する読者層が「分かりやすい！」と思ってくれなければただの独りよがりになってしまいます。

文章を読むときはいったん自分が書き手であることは忘れて、脳内に用意した仮想の読者に読んでもらうようにしましょう。筆者の場合は「エンジニアとして勉強を始めたころの自分」を仮想読者として脳内に保っています。

1.8.3 読者の歩幅に合わせよう

筆者は大きな駅でよく迷子になります。「大江戸線はこちら」と書いてある看板の矢印に従って角を曲がっても、そこに次の案内はなく「大江戸線！ どこ！！」となるのです。同じことは文章でも起こります。説明という飛び石の間隔が読者の歩幅に合っていないと、読者は次の石がどこにあるのか分からなくて立ち尽くしてしまうのです。

たとえば初心者に向けて説明するときに「httpd.conf を編集したら Apache を再起動して NameVirtualHost を on にしましょう」という説明だけだと、読者は「え、どうやって on にするの…？」と立ち尽くしてしまいます。次のように説明という**飛び石の間隔を狭めて読者の歩幅に合わせて**あげましょう。カッコの中は脳内にいる仮想読者の声です。

- まず vi コマンドで httpd.conf を開きます。この画面が表示されましたか？
  –（はい、表示されました）
- i で編集モードにします。左下に「-- INSERT --」と表示されました？
  –（はい、表示されました）
- NameVirtualHost on と追記して、エスケープで閲覧モードに戻ります。左下の「-- INSERT --」は消えましたか？
  –（はい、消えました）
- :wq で保存します。vi の画面は消えましたか？
  –（はい、消えました）
- では Apache を再起動します。apachectl restart と書いて Enter キーを押してください。何のメッセージも表示されなければ問題なく Apache が再起動しています
  –（よかった。再起動できたみたいだ）

逆に上級者向けであれば、必要以上に飛び石が多いと冗長に感じられます。対象としている読者層の歩幅に合わせてあげましょう。

1.8.4 リアルな読者には未知の視点

どんなに想像力があっても脳内の仮想読者には限界があります。推敲の後半には同僚や後輩といったリアル読者をつかまえて原稿を読んでもらいましょう。

たとえば「AWS をはじめよう」では RLogin というターミナルソフトを使ってサーバにログインする…という下りがあるのですが、ここを実際に試してもらったところ「画面の文字をコピーしたり、画面にペーストしたりするにはどうしたらいいの？ ctrl+p しても何も貼り付けられない…」という声があがりました。

これは筆者にとっては完全に盲点でした。ターミナルで文字をコピーしたければ該当箇所をマウスで選択すればいいし、ターミナル上で右クリックすればペーストされる、というのは筆者にとってもはや「当たり前」のことだったので、そこでつまづくという**読者の視点が欠けていた**のです。

このように**リアルな読者は必ず未知の視点を持っています**。脳内の仮想読者だけでなくリアル読者に読んでもらうことで、自分では気づけなかった説明不足がきっと見つかるはずです。

第 2 章

技術を登壇で分かりやすくつたえる

　人前に出て話すと緊張して手汗が止まらず声が震える…そんなあなたに向けて、この章では勉強会やイベントなどで登壇するときのコツをお話しします。

2.1 段取り八分現場二分

「段取り八分現場二分」[*1]という言葉があるように、登壇を成功させるためには当日の働きよりも事前の準備の方が肝要です。

2.1.1 準備時間は登壇する時間の 30 倍

研修や勉強会などで発表をするときには、どれくらいの準備時間が必要なのでしょうか？

たとえば人事に依頼されて新卒向けに何か 1 時間教えることになったとします。その場合、どれくらいの準備時間を見積もっておけばよいのか、なかなか目安が分からないと思います。慣れないうちは少なく見積もりすぎて「予定よりもずっとたくさんの工数がかかってしまった…準備だけでこんなに時間がかかるなんて向いてないのかも…」と落ち込んでしまうこともあるでしょう。

今までいろんな勉強会を開催したり、イベントで登壇したりした、筆者個人の体感としては**1 時間の登壇には 30 時間の準備時間が必要**です。1 日 8 時間働くとして**3.5 人日くらい**は見積もっておこう、ということですね。**登壇する時間に対して 30 倍の準備時間が必要**なので、たとえば 5 分のライトニングトーク[*2]をするには 2 時間半の準備時間を確保しないといけません。

「え、いくらなんでもウソでしょ？ そんなにかかるの？！」と思われる方が殆どだと思います。でも本当です。実際にそれくらいかかります。とはいえ上司に「1 時間話すので準備に 3.5 人日かかります」とは言いづらいと思うのですが、残念なことに準備時間を削ってしまうと、すべき準備がきちんと出来ずに結果ぐだぐだな登壇となってしまいます。そうなれば発表者も「あんなに大変だったのに…もう二度とやりたくない…」と落ち込むし、聞く側も時間を無駄にして、折角勉強会を開催したのに誰も幸せになりません。

「勉強会開催して！」とか「もっとアウトプットして！」と発破をかける側の皆さんは、ぜひ登壇の 30 倍の準備時間を工数としてちゃんと確保してあげてください。

[*1] 「段取り八分仕事二分」や「段取り八分仕上げ二分」とも言われているようです。

[*2] 5 分だけ喋ってすぐ終わる、一瞬の稲妻（Lightning）のような電光石火のトークのこと。IT 系のイベントやカンファレンスなどでは、メインセッションが終わった後に LT の枠が用意されていることが多いです。

2.1.2 絶対必要リハーサル

もしあなたが「人前での発表がうまくいかない」とお悩みであれば、うまくいかない原因はきっとたったひとつ、リハーサル[*3]をしないから、もしくはリハーサルの回数が足りていないからです。

登壇する際はできるだけ本番と同じ環境を用意して何度もリハーサルをしましょう。リハーサルは誰かに聞いてもらってもいいし、少人数用の会議室やカラオケなどで自分ひとりだけでやっても構いません。但し「HDMI のケーブルを差し込んでもなぜかモニタに映らない」「プロジェクターで投影すると文字の色が薄くて見えない」など、その場所でやってみて行ってはじめて気づく問題もありますので、必ず 1 回は「実際に登壇する場所」でリハーサルをしておくべきです。筆者もはじめての内容で登壇するときは必ずリハーサルを最低 2 回はしています。

リハーサルはパワーポイントの資料を投影して印刷した発表原稿を手に持ち、しっかり声を出して当日と同じようにやってみます。実際に声に出しながらやってみると、必ず発表原稿や投影資料でおかしなところが見つかるので、そこを直してもう 1 回通しでやってみましょう。持ち時間に対して長すぎてオーバーしてしまうとか、逆に想定より早く終わってしまうといった問題もリハーサルをやれば分かります。

この「直したらもう 1 回通しリハーサルをする」を繰り返す作業は、文章を頭からしっぽの先まで撫でて直していく毛づくろい作業に似ています。

一度もつまずかずに最後まで通しでリハーサルが出来たら準備完了です。

2.2 前に立って話すときのテクニック

2.2.1 今いる場所を確認してゴールを指し示そう

たとえば筆者がアプリケーションエンジニア向けの勉強会で、前に立つなりいきなり「今日は named.conf について話します。named.conf の書き方は…」と話し始めたら、きっと聞き手の大半は「え？ なんのこと？」となるでしょう。このように聞き手がボールをキャッチできる体勢になっていないのに、いきなりボールを投げてはいけません。

前述の例であれば、いきなり「今日は named.conf について話します」と話し出すのではなく、次のようにしてみましょう。

[*3] ところでこの項のキャプションが少女革命ウテナのオマージュだと気づいてほしい。

```
皆さんは普段アプリケーションエンジニアとしてコードを書いていると思いますが、DNS の仕組
みについては知っていますか？
では名前解決をするための DNS サーバを立てたことがある人はいますか？…無い人が殆どです
よね。
今日は「1 人で DNS サーバを立てられるようになる」をゴールに据えて、DNS サーバのインス
トールと設定ファイルについて話をしていきます。
では最初に DNS サーバの設定ファイル、named.conf について説明しますね。
```

技術書やドキュメントの冒頭で「対象読者層」と「ゴール」を示すのと同様、前に立って話をするときも次の 2 点を確認した上で話を始めるようにしましょう。

- みんなが今いる場所
  - DNS の仕組みは何となく知ってるけど、DNS サーバを立てたことはない
- ゴール
  - 1 人で DNS サーバを立てられるようになる

今いる場所については、教える側と聞き手でお互いに認識があっていれば確認しなくても構いません。

大規模なカンファレンスなどでどうしても初心者からベテランまで幅広い層を相手に話をしなければならないこともあります。そうした場合は話の冒頭で「対象となる層」や「話のゴール」をつたえておきましょう。最初に対象層をつたえてもらえば、聞き手も聴講するしないをその場で判断することができます。

2.2.2 演説じゃなくて対話をしよう

仲のいい A さんと自分のふたりっきりで話をするときはどんな濃い話題でも気兼ねなく話せます。ですが B さんが増えて 3 人になると、A さんと B さんどちらにも通じる話題を選ばなければならず、話の濃度は少し下がります。そうして人数が増えるにつれて「全員に通じる話題」は段々無難なものになっていき、話の濃度はますます下がっていきます。

このように会話に参加する人数が増えていくと、最終的には対話は難しくなり、一方的な演説になってしまいがちです。そして「仲良しの A さんが自分に向かって濃い話をしている」と思えば話を聞く方も身が入りますが、「たくさんいる中の誰かがみんなに向かって演説している」と思うと集中力は途切れがちです。

ですのでつたえたいことがあるなら 1 対多で演説をするのではなく、たとえ大勢を前に話していても**ひとりひとりの反応を見ながら 1 対 1 で話をする**ように意識しましょう。こちらの目を見て、にっこりしながら何か話している、何かつたえようとしている、それだけで聞き手は話を聞いてくれる姿勢になります。

発声した声が放物線を描いて相手のところでボールのように飛んでいくイメージを持ちましょう。不特定多数の誰かに向かって大量のボールを適当に投げるのでは

なく、会場を見回して聞き手と目をあわせながらひとつひとつ「このボールはあの人に…よし、キャッチした」「今度のボールはこっちの人に…よし、キャッチした」というようにつたえたいことを声にのせて投げるのです。

一方的な演説は相手の心に届きません。大勢を前に話していても対話を心がけましょう。

2.2.3 「あー」「えー」よりも沈黙を

早口や小声は聞き取りづらいので、できるだけ意識してゆっくりと大きな声で話しましょう。緊張しているときは自分で思っている以上に早口や小さい声になってしまいます。

また言葉と言葉の間でどうしても「あー」や「えー」と言いたくなりますが、言葉が続かないとき、それからドキドキして息が続かないときは 20 秒くらい沈黙しても問題ありません。不思議なことに沈黙があった方がずっと話し上手に見えるものです。

2.2.4 動くものは目で追ってしまう

人は動くものに目がいくので、特に集中して聞いて欲しいときは立ち上がったり、うなづいたり、手をあげてみたり、その手をきゅっと握ってみたり、投影スライドの右前から左前に移動したりしてみましょう。歩き回りながら聞き手に向かって「どの辺が分からなかった？」と話しかけてみるのもよいです。

2.2.5 話す速度は一定にせず緩急をつけよう

ずっと同じ場所にいると目につかないので動きましょう！ というのと同じ理屈で、ずっと同じ速度で話していると聞き手は集中力が途切れて段々眠くなってきます。大事なポイントはゆっくりと念を押すように話したり、細かいところは敢えて早口にして笑いを誘ったり、話す速度も緩急をつけることで衆目を集められます。

2.2.6 「大丈夫？」と聞かないで

聞き手の質問を引き出そうと話しかけるのはとてもよいのですが、そのとき「大丈夫ですか？」「分かりましたか？」と聞くと、聞かれた人はつい「大丈夫です！」「分かりました！」と応えてしまいます。質問するときは「ここまでの話でいまいちわかってないところをひとつだけ挙げるとしたら？」のように、聞き手が疑問を切り出しやすい聞き方にしましょう。

筆者の体感としては、聞き手はみんな、登壇者が思っている以上に色んな疑問が頭に浮かんでいて、でも恥ずかしくてそれを聞けないことでひそかに苦しんでいます。

聞き手の様子に目をこらし、耳を澄ませましょう。

2.2.7　1 匹 1 匹にあわせて猫じゃらしを振ろう

もしあなたがよい登壇者ならば同じ資料、同じ内容、同じ所要時間であっても 2 人に向かって話すのと 250 人を前に話すのでは疲れ方はまったく異なるはずです。疲れが異なる理由は、前述のようにひとりひとりの反応を見て関心を引きながら話そうとしているからです。2 匹を相手に猫じゃらしを振るのと、250 匹を相手に猫じゃらしを降り続けるのでは後者の方が疲れて当然ですよね。

もし大勢を相手に話しても大して疲れていないとしたら、猫じゃらしが適当になっているのかも知れません。対話ではなく一方的な演説になっていないか、自分の話し方を振り返ってみましょう。

2.3　つたえるときの心のあり方

技術を学んで少しずつ色んなことが分かるようになってくると、逆に「分からない人の気持ち」が分からなくなってきます。

技術をつたえる側と聞き手の間で心理的安全性に満ちた関係が築けるよう、ここでは聞き手の気持ちを想像してみましょう。

2.3.1　聞き手は恥におびえている

聞き手は「こんなことも知らないのかと思われたくない」「技術がないことを悟られたら怒られるのではないか」「間違えて恥をさらしたくない」と常におびえています。

登壇するとき、本題に入る前にまずは「この人は私が分からないと言っても怒らない人だ」という安心感を持ってもらいましょう。そうしないと彼らはあなたから「知らない」「分からない」を隠すことに全力を注いでしまい、質問など出てこないし、何を説明しても入っていかなくなってしまいます。

そのためにも心のどこかに少しでも「こいつこんなことも知らないのか」「こんなに分かりやすく説明してやってるのになんで分からないんだ」と馬鹿にする気持ちがあってはいけません。どんなに取り繕ってもその気持ちはにじみ出して聞き手を萎縮させます。

もし説明を分かってもらえないとしたら、きっとつたえ方や話す順番にさらなる工夫が必要なのです。あるいはもっと手前の段階から説明が必要なのかも知れません。いずれにせよ、他責的に考えるのはやめましょう。否は聞き手ではなくつたえる側にあります。

【コラム】真面目な後輩に説明を聞いてもらえないのはなぜ？

ある日、「サーバに入ってこのファイルを落としてきて！ 14 時までによろしくね」と頼まれた新米エンジニアの A さんが「サーバに入れない…どうしよう…」と困っていました。

先輩エンジニアの B さんは、設定を見てすぐに「ログイン時に使う秘密鍵を指定していないから入れないんだな」と分かったものの、サーバにログインするときの SSH の仕組み自体を理解しておいて欲しかったので、A さんに「そもそも SSH ってね…」と説明を始めました。

すると A さんは説明を聞いてはいるもののどこか上の空で「いいから早く答えだけ教えて！」という気持ちが隠しきれない様子です。A さんは普段から真面目で、技術的なことも熱心に吸収しようとしているタイプなのにどうしてだろう？ と B さんは不思議に思いました。

みなさんは**B さんが A さんに説明を聞いてもらえなかった理由**が分かりますか？

B さんは新米エンジニア A さんのコップに「SSH の説明」を注ごうとしたのですが、このとき A さんのコップは「14 時までにサーバに入ってファイルを落としてきてって頼まれたのに！ 間に合わなかったらどうしよう！ 早くやらなきゃ！」という気持ちでいっぱいでした。すでに中身がいっぱいのコップに上から新しい知識を注いでも、あふれるばかりで新しい知識はちっとも入っていきません。

ですのでまずは「サーバに入れない」という問題をさっと解決して、ファイルを落としてくるところまで終わらせてコップを空にしてあげましょう。その上で「間に合ってよかったね。ところでなんでサーバに入れなかったのか、仕組みを理解しておくと次からは自分で対処できると思うからちょっと説明してもいい？」と切り出せば、B さんの説明はしっかり A さんのコップに入っていったはずです。

これはめちゃくちゃ怒っているクレーマーに「お客様それは違います！」と勘違いを指摘しても火に油で全然聞き入れて貰えないのと同じですね。まずは「なるほど、なるほど。不快な思いをされたんですね、なるほど」とひたすら聞いてコップを空にさせてから「恐らくですが認識に齟齬が…」と切り出さないと、こちらのつたえたいことは相手のコップに入っていきません。

説明をあんまり真面目に聞いてもらえない…と感じたときは、相手のコップが空で中身を入れられる状態なのか？ を先に確認してから話すようにしてみましょう。

第 3 章

教わり上手をはじめよう

技術を分かりやすくつたえるテクニックをまとめた本著ですが、実は教わる側にもコツがあります。

教わってもよく分からないときは「この人教え方下手だな…」とげんなりする前に、あなたがよい聞き手であるかを振り返ってみましょう。

3.1 知らない・分からないを正直に言うことこそ最初の一歩

これはエンジニアに限らずですが、何かを学んで圧倒的に成長するためには、

- お！ こいつはデキるやつだな！ と思われたい
- 間違えたことを指摘されて恥をかきたくない
- こんなことも知らないのか、とがっかりされたくない

という気持ちはさっさと海にぶん投げて、**「知らない」「わからない」を正直に言えるようになる**ことが何より大切です。

【コラム】誰より成長したのは文学部出身のエンジニア

筆者が新卒で IT 系の会社に入ったとき、エンジニアの同期が 10 名弱いたのですが、その殆どが大学や大学院でエンジニアリングの勉強をしてきたメンバーでした。そしてその中にひとりだけ「文学部出身で一切経験ありません」という男性がいました。ここでは彼を K さんとします。

入社後の技術研修で、教育担当の先輩社員にたとえば「cron って知ってる？」と聞かれたとします。そんなとき当時の筆者はプライドの塊だったので「cron か…大学の授業で聞いた気もするし、全然知らない訳じゃないし…とりあえず知ってるって言っとこうかな…」などと保身の方向で頭がフル回転していたのですが、K さんはそんな私の思惑も知らずに隣でさらっと「知らないッス」と言ってのけます。

「知らない」を言うことにものすごく抵抗感のあった筆者は、それだけで「K さん知らないって言った！！ えっ、ちょっと！ 叱られない？！」と慌てたのですが、先輩社員はただ「教える前に今の理解度を確認しておく」ために聞いただけなのでもちろん怒るはずもありません。「そっか、cron っていうのはね…」と K さんに説明をはじめました。

先輩からしっかり教えてもらった K さんの cron に対する理解度は 0 から 10 になりました。元々 2 くらい知っていた筆者と比較すると、その時点で K さんの方が cron について詳しくなっています。たった 1 回「知らない」を言えたか言えなかったかだけで、筆者と K さんの間には 8 もの差がついてしまったのです。その調子で K さんは「知らない」「わからない」を臆せずにばんばん聞いていって、入社して 1 年が経ったとき、エンジニアとして一

番成長していたのは K さんでした。

この話でつたえたいのは**「知らない」「分からない」を言って損することは何もない**ということです。

K さんは「知らない」と言ったことで 10 の知識を得ます。近くで聞いていた筆者も知ってるつもりだったけど実際は 2 しか理解していなかったので、K さんのおかげで残り 8 の知識を得ました。そして教える側の先輩社員も、相手の理解度に合わせて教えて理解してもらわないと実務に入ったときに困るので、知らなかったら「知らない」と言ってもらった方がよほど有り難いです。会社だって理解度が 0 や 2 の状態で研修が終わるより、新入社員みんなが 10 の知識を持ってくれた方が嬉しいです。

K さんが「知らない」を言ったことで誰も損をしていません。むしろ周囲も「知らない」が言いやすくなるので好循環です。

医者に対して「ここが痛い」を隠してもいいことはないように、**エンジニアとして働く上で「知らない」「分からない」を周囲に隠してもいいことはありません**。恥を捨てて「知らない」「分からない」を晒せれば、それが学びの第一歩になります。

3.2 質問はオープンな場でしよう

「何か分からないこととか質問はありますか？」と聞かれたときには手を挙げないけど、終わった後に登壇者のところへ個別にそっと聞きに行くなんてことをしていませんか？

クローズドな場で質問すると、登壇者はあなたひとりにしか回答をつたえることができません。後から何人もに同じ質問をされると、登壇者としては「個別の質問はコスパが悪いからみんながいるときに聞いて！」という気持ちになることもあります。

前述のコラムであったように、K さんが臆せず質問してくれたことによって筆者がおこぼれで学べたことはたくさんありました。あなたがオープンな場で最初に質問をすれば次の人も続きやすくなります。あなたがレベルの低い質問をすれば、周りも「なんだ…自分だけが特にレベル低いってわけじゃないんだ」とほっとして次から質問しやすくなります。あなたが分からなかったことを聞けば、同じことで悩んでいた他の人も「同じところでつまづいてる人がいたんだ」と言い出しやすくなります。オープンな場で質問することにはメリットしかありません。

とにかく「質問する奴は偉い」[*1]のです。

【コラム】失敗するとしっぱいねこが生まれる

失敗すると恥ずかしくて落ち込むものですが、あなたが失敗したら「しっぱいねこ」が生まれると想像してみましょう。灰色でつるつるした毛並みの可愛いねこです。あなたが間違った理解をしていてトンチンカンな質問をしたとか、Git で誤ったブランチを Master にマージしたとか、リリースをしくじって切り戻しになったとか、何かしら失敗をするとしっぱいねこがポンッと生まれて「いやー、しっぱいしっぱいー」と言いながらてくてく歩いてきます。大きな失敗をすると大きなしっぱいねこが生まれます。小さなしっぱいだと小さなしっぱいねこです。どちらもかわいいですね。

しっぱいねこは仔猫を生みます。たいていの場合は同じ「しっぱいねこ」が生まれますが、稀に「せいこうねこ」というレアなねこが生まれることがあります。

そしてこの「せいこうねこ」は必ずしっぱいねこからしか生まれません。[b]

[*2]せいこうねこはこがね色でふわふわした毛並みのすごく可愛いねこです。せいこうねこは「やったー、せいこうー」と言いながらしっぽをぱたぱた振ってくれます。

あなたがしっぱいねこを生めば生むほど、せいこうねこが生まれる確率は上がっていきます。逆に恥をかく失敗を恐れて何もしないと、しっぱいねこが生まれないのでせいこうねこも絶対に生まれません。

失敗したら「失敗した…恥ずかしい…地面に埋まりたい…」と落ち込むのではなく、「よいしっぱいねこが生まれた！」としっぱいねこをぎゅっと抱きしめて喜んでみましょう。成功は失敗の後にしか生まれないのです。

[b] ようやくゲットした色違いポケモンを育て屋に預けて卵を産ませても、普通のポケモンしか生まれないのと同じですね。

[*1] https://twitter.com/motcho_tw/status/870589211832795136

3.3 地蔵にならず反応を返そう

勉強会などで講師役をやりたがらない人に「人前で話すことの何がいや？」と聞くと、いちばん多いのは「リアクションがなくてつらい」です。

それと根っこは同じかと思いますが、

- ちゃんとみんなの興味があることを話せているか、退屈させているんじゃないかと不安
- 自分なんかがつまらない、間違った、しょうもない話をしてみんなの時間を浪費するのは申し訳ない
- 「それ違うだろ！」と内心で思われてるんじゃないかと思って目線がつらい

というように、聞き手のリアクションがないことによって疑心暗鬼になった挙句、話すのが嫌になっていることが分かります。

確かにひとりで喋っていて誰も何も反応してくれないと

- みんなどう思ってるんだろう？
- 退屈だな、つまんないな、と思ってるんじゃないかな？
- 内心、低レベルな発表だなとか、それ間違ってるよって思われてるのでは？

とどんどん悪い方へ想像が転がっていってしまうものです。

リアクションが何にもないのってすごくつらいですよね。ボールを投げても投げても返ってこない。一度でも人前で話した経験があれば、あの冷や汗がでるような、吸っても吸っても酸素が入ってこないような焦る感じは分かると思います。リアクションがないのは本当につらい！

だというのになぜ人間は「聞く側」に回った瞬間にすべて忘れて地蔵になってしまうのでしょう？

反応がないのは暗に「つまらない」と言っているようなものです。という訳で聞く側も聞く姿勢を正してよい聞き手を実践しましょう。みんな自分が発表するときは緊張するのに、その緊張することを頑張ってやっている人が目の前にいて応援しないとは何事だ！ という話です。合コンさしすせそのように「さすがー」「知らなかったー」「すごーい」「センスいいー」「そうなんだー」と合いの手を挟みましょう。

大事なことなのでもう 1 回書きますが、リアクションがないのは登壇者にとって本当につらいことです。みんながアウトプットをしていくには受け止める側の熟練度も大切です。地蔵はやめてばんばんリアクションをしていきましょう。

3.4　つたえる側を体験すれば教わり上手になれる

「一人暮らしをはじめたら今まで何の気なしに食べていた料理の有り難みが分かった」という話はよく聞きます。それと同じで「教える側」を体験すると、準備の大変さやつたえることの難しさがよく分かります。「先輩は教えるのが下手だ」「説明されても全然理解できない」と思ったら、次の勉強会は自分がやりますと手を挙げてみましょう。

新しい技術を習得したければ、誰かに教わるよりも、誰かに教えるつもりで勉強した方が身につきます。そして一度技術を分かりやすくつたえようと四苦八苦すると、再び教わる側になったときに相手の意図や親切をくみ取れるようになります。登壇者と聞き手がお互いを思いやって良い循環が生まれるやさしい世界がいいですね。

あとがき

書き終わったら書くー！！！
数ある技術本の中から「技術をつたえるテクニック」を手に取ってくださったあなたに感謝します。

2018 年 10 月
mochikoAsTech

Special Thanks:

- ミルクティー色で折れ耳のゆるふわ愛され猫
- Ayaka Chikamoto
- Miho Sunada
- Masao Takado
- 湊川あい

レビュアー

- Takeshi Matsuba
- 深澤俊

参考文献・ウェブサイト

本著を書くにあたって、今一度「技術を分かりやすく伝える技術」について勉強をすべく、次の本やウェブサイトを改めて読み返しました。するとそこには私の中で芯となっている「わかりやすい文章とはかくあるべき」という教えがいくつもいくつもあり、微細な内容は忘れていたくらいなので読んだ当時の記憶はもはやないのですが「ああ、私の中にある考えのオリジナルはここにいたんだね」という気持ちになりました。

正直、分かりやすい文章の書き方は「これらを読んでください」で済む話なので、これらの本やウェブサイトを認識しながらこの分野の本を書こうとするのは恥ずかしい思いもありました。

ですが筆者は結城浩さんの「人生の中で、そのときにしか書けない文章というものがあります。もっと経験を積んでから書くのではない。もっと勉強してから書くのでもない。いまの私にしか書けない文章。いまの私が書かなければ、世界中の誰も（未来の私でも）書けない文章があるのです。」というツイート[*3]の考え方がとても好きでした。この考えを胸に、この先も恐れずに「しっぱいねこ」をたくさん生んでいこうと思います。

- 新版 日本語の作文技術　本多勝一 https://www.amazon.co.jp/dp/4022618450/
- 数学文章作法 基礎編・推敲編　結城浩 http://www.hyuki.com/mw/
- 教えるときの心がけ　結城浩 https://www.hyuki.com/writing/teach.html

[*3] 結城浩の連ツイ いまのあなたにしか書けない文章を、書こう！ https://rentwi.hyuki.net/?962345433212203010

著者紹介

mochiko / @mochikoAsTech

無職。元 Web 制作会社のシステムエンジニア。モバイルサイトのエンジニア、SIer とソーシャルゲームの広報を経て、2013 年よりサーバホスティングサービスの構築と運用を担当したのち、再び Web アプリケーションエンジニアとしてシステム開発に従事。「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 4,5 で頒布した「DNS をはじめよう」と「AWS をはじめよう」は累計販売数 4,500 冊を突破。

- `https://twitter.com/mochikoAsTech`
- `https://mochikoastech.booth.pm/`

Hikaru Wakamatsu

表紙デザインを担当。

Shinya Nagashio

挿絵デザインを担当。

技術をつたえるテクニック
技術書執筆から登壇までをサポート

2019 年 4 月 14 日　技術書典 6 版 v1.0.0

著　者　　mochikoAsTech
デザイン　Hikaru Wakamatsu / Shinya Nagashio
発行所　　mochikoAsTech
印刷所　　日光企画