Qiita Tech Book

vol.1 | 2025 Qiita執筆部

Qiita Tech Book vol.1

Qiita 執筆部　著

2025-04-16 版　　発行

まえがき

この度は、私たちの技術同人誌「Qiita Tech Book vol.1」を手に取っていただき、誠にありがとうございます。このイベントで皆様とお会いできたこと、そして私たちの成果物を直接お渡しできることを、メンバー一同、たいへん嬉しく思っています。

Qiita は「エンジニアを最高に幸せにする」というミッションのもと、日本のエンジニアリング発展を目指して活動しています。今回の技術同人誌は、私たちが日ごろ培ってきた技術力や、大切にしているチームの文化を皆様に紹介したいと考え、制作いたしました。私たちのことを少しでも知っていただき、親近感を持っていただけたら嬉しいです。

本書は大きく 2 つのパートに分かれています。最初の三章では、私たちが大切にしている文化について紹介しています。具体的には、エンジニアリング、アウトプット文化、勉強会の開催など、私たちが大切にしている考え方や取り組みについて紹介します。このパートを通じて、私たちがどのように考え、どのように働いているのかを感じていただければ幸いです。

後半の三章では、私たちが普段取り組んでいる技術について、具体的な事例を交えながら解説しています。具体的には、Qiita の記事で使用されている Markdown の処理、ダークテーマ導入までの流れ、Qiita の設計改善など、幅広い技術領域における開発事例を紹介します。私たちの経験が、読者の皆様の技術的な取り組みの参考になれば幸いです。

今回の同人誌制作は、私たちにとってはじめての試みであり、多くの挑戦がありました。執筆、編集、デザインなど、すべての工程において試行錯誤を繰り返しましたが、チームメンバー全員の協力と熱意によって、ようやく完成させることができました。この場をお借りして、執筆に協力してくれたメンバー、素敵な表紙を提供してくれたデザイナー、そして細部にわたって校正してくれた方々に、心から感謝申し上げます。

私たちは今後も、技術に関する情報を積極的に発信していきたいと考えています。今回の同人誌をきっかけに、私たちに興味を持っていただけたら嬉しいです。

最後に、本書が皆様にとって有益な情報源となり、少しでも楽しんでいただけることを願っています。今後も Qiita 株式会社は「エンジニアを最高に幸せにする」というミッションの実現に向けて、よりよいサービスを提供できるよう努めていきます。引き続き、Qiita をよろしくお願い致します。

目次

第 1 章

Qiita 株式会社のエンジニアリング文化

1.1　はじめに

第 1 章「Qiita 株式会社のエンジニアリング文化」 をお読みいただきありがとうございます。Qiita 株式会社の清野です。本章では、当社が重要視するエンジニアの振る舞い、それが独自のエンジニアリング文化として形成され、日々の開発にどう影響しているかをご紹介します。

Qiita は「エンジニアを最高に幸せにする」というミッションのもと、日本のエンジニアリング発展を目指しており、高い技術力に加え、プロフェッショナルなエンジニアの振る舞いが不可欠です。ユーザーに安心と誇りを提供するため、エンジニアは常に高い意識と誠実さを心がけています。

本章は、組織づくりや改善に関心のある方、Qiita の開発組織に興味のある方に向けて、当社がエンジニアに求める価値観とその組織文化への浸透について解説します。

まず Qiita が重視するエンジニアの振る舞いとその背景を明確にし、具体的な行動指針をご紹介します。そして、これらの価値観や指針がどのように当社のエンジニアリング文化として根付き、日々の業務で実践されているのかを解説します。

本章を通じて Qiita のエンジニアリング文化を感じていただき、皆様の組織づくりにお役立ていただければ幸いです。

1.2　重要視するエンジニアの振る舞い

Qiita 株式会社では、ユーザーに誇りと感じてもらえるエンジニアとしての振る舞いを重要視しています。それは技術力だけでなく、プロフェッショナルとしての自覚と責任感に基づいた行動です。

重要視していること

Qiita 株式会社がエンジニアの振る舞いとして特に重要視しているのは、「**Qiita を使っているユーザーに誇りと感じてもらえる**」という点です。これは単に機能が動けばよいということではありません。ユーザーが Qiita を通して最新の技術情報に触れ、日々の開発業務をより効率的に、より楽しく開発を進められることを願う姿勢。そしてそれを実現するため、エンジニアとして何ができるのかを常に考える姿勢を求めています。

具体的に求めている姿勢は次のとおりです。

- 質の高いコードを書こうとする姿勢
- 将来を見据えた保守性の高い設計にしようとする姿勢
- 最新技術への積極的な取り組む姿勢
- ユーザーからのフィードバックを真摯に受け止め改善に繋げる姿勢
- チーム内外との円滑なコミュニケーションと協力を大切にする姿勢

さらに、Qiita はエンジニアのためのプラットフォームであるため、エンジニア自身がロールモデルとなるような振る舞いも期待しています。技術情報の積極的な発信も、ユーザーに誇りと感じてもらえるエンジニアの重要な側面です。

なぜその振る舞いを求めているのか

「**Qiita を使っているユーザーに誇りと感じてもらえるエンジニアとしての振る舞い**」を求めるのは、ユーザーからの信頼を得るためです。尊敬できないエンジニアのサービスをユーザーは信頼しません。

多くのエンジニアに利用される Qiita では、開発エンジニアの質がユーザーの信頼に直結します。杜撰なコードやユーザーを無視する姿勢、技術的成長の停滞は、ユーザー離れにつながります。

私たちは Qiita を単なる情報収集ツールではなく、成長とコミュニティの場にしたいと考えています。そのため、エンジニア一人ひとりの高い倫理観とプロフェッショナル意識が不可欠です。

ユーザーに使い続けてもらうことこそが Qiita の存在意義であり、エンジニアの使命です。常に自己研鑽を怠らず、ユーザーの期待を超える価値を提供し続ける必要があります。

1.3　求めている振る舞いの紹介

それでは早速ですが、Qiita 株式会社でエンジニアに求めている振る舞いを紹介します。以下は社内メンバーに展開されている文章をそのまま記載したものです。

これらの振る舞いをすべて完璧に実践することは難しいですが、Qiita のエンジニアとして目指すべきラインとして理解し、できることから改善していくことを求めています。

Qiita の開発に関わるすべての領域に精通している

Qiita はインフラからフロントエンドまですべてのエンジニアが使うサービスです。そのため、Qiita のエンジニアはできるだけすべての領域に精通しているべきです。「インフラはインフラエンジニアが、マークアップはデザイナーが」という考えではなく、すべての領域を一人で開発できることを目指し、他の領域の手を借りる場合は自身の努力不足と考えましょう。

最先端を牽引する

Qiita はエンジニアが最先端の情報に追従できるサービスであるため、そのシステムも最先端を牽引すべきです。常に新しい情報を追い、スペシャリティを発揮しながら最先端を牽引し続けましょう。

誰に見られても恥ずかしくないコードを書いている

Qiita はエンジニアが使うサービスである以上、利用するエンジニアに落胆されるようなコードは絶対に書くべきではありません。自分が誇りに思えるコードを書きましょう。レビューで指摘されないと判断できるまでは提出せず、自分のコードにこだわりましょう。指摘されると分かっているコードをレビューに出すのは避けるべきです。

過去、現在、未来、誰が読んでもわかりやすいログを残している

Qiita は今後も多くの人が継承し、開発し続けるサービスです。どの時代でも、常に次世代のエンジニアに現在何が起こったのかをログとして残し続けることが、未来への道標となります。Qiita Team、PR、Issue、コミットログなど、あらゆる手段を用いてログを残しましょう。口頭での議論もログとして記録しましょう。また、過去のログを常に確認し、過去に何が起こったのかを理解した上で開発しましょう。

リリースするものすべてに責任をもつ

Qiita が長く続くサービスであるために、リリースは終わりではなく始まりです。機能をリリースしたり、新しいものを導入した後も、チームとして責任を持って改善し続けるべきです。これは開発に限らず、ユーザーへの発信や、それに対するユーザーからの声に対しても真摯に向き合い改善を続けることも含みます。

責任のあるレビューを行っている

Qiita におけるエンジニアリングにおいて、レビューは実装ミスを実装者だけに負わせないための重要なプロセスです。レビューするコードに不備があれば自分の責任と考え、時には実装者と議論しながらよりよいコードを作り上げましょう。実装者や他のレビュアーに判断を任せるような態度は改めましょう。

サービスを動かし続けることに責任を持っている

Qiita が停止することはもっとも深刻な事態であり、日本のエンジニアリングの停滞と同義といえます。アラートを受け取ったら、可能な限り迅速に問題を解決するために対応しましょう。

Qiita におけるエンジニアリングに沿っていない行動を指摘することができる

誰でもミスや判断の誤りをする可能性があります。そのような場面に遭遇したら、お互いにフィードバックし合い、よりよいエンジニアを目指しましょう。

1.4　文化としての根付かせ方

Qiita 株式会社ではエンジニアに求める振る舞いを単なる理想として掲げるのではなく、日々の業務や制度の中に落とし込み、組織文化として根付かせるためのようさまざまな取り組みを行っています。ここでは、その一部を紹介します。

職種として特定の領域に絞らずに業務を行う

Qiita 株式会社では、「フロントエンドエンジニア」「インフラエンジニア」のように職種で担当領域を限定していません。基本的にすべてのメンバーがインフラ、バックエンド、フロントエンドの業務に携わる機会があります。もちろん機能開発組織や基盤開発組

織など、組織ごとに特定の領域をメインに担当することはありますが、完全に限定されることはありません。また、メンバーの組織異動も積極的に行い、全員が幅広い知識を習得できる体制を整えています。

メンバー同士のお互いへのフィードバックとして使用

前提として「お互いにフィードバックすること」自体を Qiita 株式会社のエンジニアに求める重要な振る舞いと定義しています。役職にかかわらず、お互いにフィードバックをすることを推奨しています。

Qiita 株式会社ではエンジニアに求める振る舞いを明確にした上で、それをメンバー同士がフィードバックをする際の基準として活用しています。日々のコードレビューやチームの振り返りなどの場で、この行動指針に照らし合わせながら具体的な行動や成果について建設的な意見交換が行われています。

たとえばコードレビューでは「**誰に見られても恥ずかしくないコードを書いているか**」という観点から、コードの品質だけでなく、可読性や保守性についても議論されます。また、Issue や PR の内容はその時点での理解だけでなく、将来的に背景を知らない人が読んでも理解できるドキュメントとして作成できているかという点もレビューや指摘の対象となります。

OJT で文化の定着を目指す

新入社員には OJT を通じて Qiita のエンジニアリング文化を丁寧に伝えています。先輩エンジニアが技術だけでなく、大切にしている価値観や行動指針も時間をかけて教育します。

たとえば、コードレビュー依頼時には「過去、現在、未来、誰が読んでもわかりやすいログを残している」という指針を意識させ、コミットログや Issue、PR の書き方、Slack での作業ログ共有方法などを指導します。

OJT 期間中は定期的な面談で新入社員の状況を確認しサポートします。このように、OJT を通じて早期に文化理解を深めてもらい、スムーズなチーム合流と活躍を支援しています。

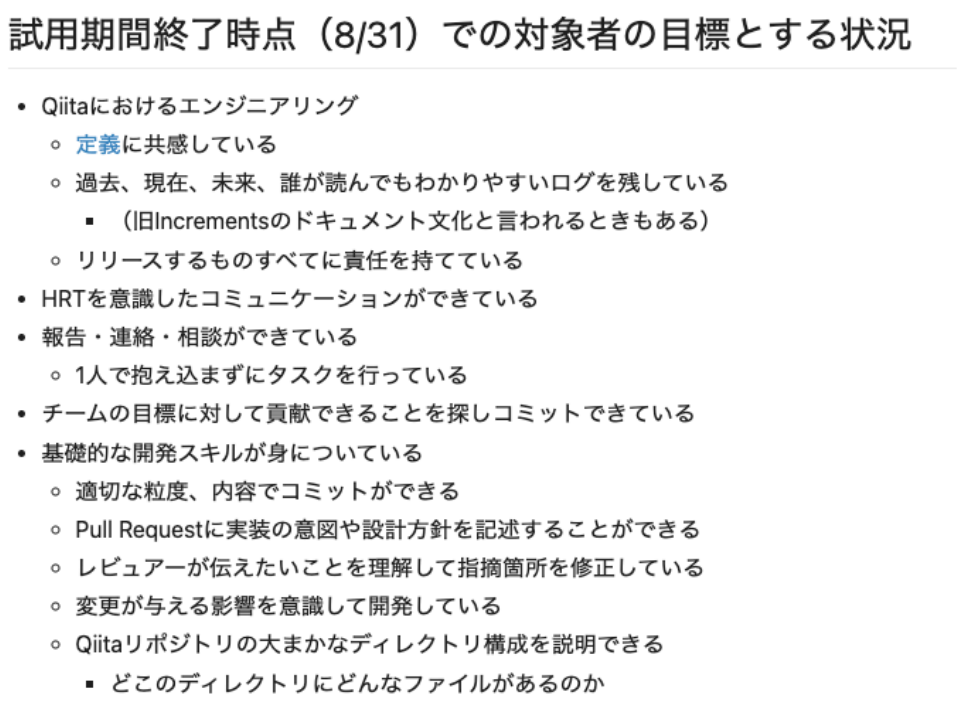

図 1.1: 新入社員のオンボーディングプロセスドキュメント（一部抜粋）

1.5　まとめ

本章では、Qiita が重要視するエンジニアの振る舞いと、その組織文化への根付かせ方について解説しました。

もっとも大切なのは「**Qiita を使っているユーザーに誇りと感じてもらえるエンジニアとしての振る舞い**」です。これは単なる技術力ではなく、プロフェッショナル意識とユーザー視点での行動を意味します。ユーザーが信頼し、使い続けたいと思えるサービス提供こそが Qiita の存在意義です。

この価値観を浸透させるため、組織体制やフィードバック、OJT などさまざまな取り組みを行っています。

Qiita のエンジニアリング文化は常に進化していますが、「ユーザーに誇りを持ってもらえるサービスを提供する」という根本的な想いは変わりません。

今後も Qiita は「エンジニアを最高に幸せにする」というミッションのもと、エンジニアリング文化を大切にし、よりよいサービス提供に努めます。引き続きご期待ください。

第 2 章

Qiita のアウトプット文化

2.1　はじめに

第 2 章「Qiita のアウトプット文化」を読んでいただき、ありがとうございます。Qiita 株式会社でデザイングループのマネージャーをしている出口裕貴です。普段は、デザイン組織のマネジメントや PdM として Qiita の開発に関わりながら、社内外問わずアウトプット活動にも積極的に取り組んでいます。

Qiita 株式会社では、記事投稿やイベント登壇、社内勉強会など、社内外問わずアウトプットが積極的に行われています。Qiita というエンジニア向けの記事投稿サービスを運営している会社として、「アウトプットすること」を自然に行っています。ですが、それはたまたま「アウトプット好き」が集まったからというわけではありません。「どうすればアウトプットのハードルを下げられるか」を試行錯誤しながら、少しずつ根付かせた結果が今のアウトプット文化だと感じています。

本章では、そんな Qiita 株式会社のアウトプット文化について紹介します。「アウトプットが続かない」「何をアウトプットしていいかわからない」と悩んでいる方や組織としてアウトプット文化を醸成したいと考えている方に向けて、Qiita 株式会社のリアルな取り組みや工夫をお伝えできればと思っています。ぜひ、ご自身やチームのアウトプットの参考にしてください。

2.2　アウトプットとは

Qiita のアウトプット文化の説明に入る前に、「そもそもアウトプットとは何か」について紹介します。

アウトプットと聞くと、「技術記事を書くこと」や「カンファレンスで発表すること」を思い浮かべる人が多いのではないでしょうか。しかし、Qiita 株式会社ではアウトプットをもっと広い意味で捉えています。

アウトプットの定義

アウトプットとは、「**自分の経験や学びを、他の誰かの役に立つ形で共有すること**」だと私たちは考えています。その手段は人それぞれで、特別なことをしなくてもかまいません。

- 技術記事を書く
- 勉強会・イベントに登壇する
- 社内 LT 会で話す
- 業務フローを社内 Wiki にまとめ、マニュアル化する
- 日報や Slack で学んだことをシェアする

こういったものはすべて「アウトプット」です。業務で得たナレッジはもちろん、業務に直接関係ない挑戦や勉強の過程を共有することもアウトプットです。

「**自分の経験や学びを、他の誰かの役に立つ形で共有すること**」。それが Qiita 株式会社におけるアウトプットの定義です。

アウトプットの価値

「アウトプットはよいこと」だと、よくいわれますが、なぜよいことなのでしょうか。自分自身の成長はもちろん、チームや組織全体にもさまざまな価値をもたらすからです。ここでは、「学びの定着や循環」と「対外的なアピール」の 2 つの視点から、考えてみましょう。

学びの定着や循環

アウトプットは、単に「誰かのために発信する」という行為ではありません。自分自身の学びを深め、成長を実感するための有効な手段でもあります。たとえば、学んだことを文章にまとめる中で、思考が整理されます。また、人に説明することで、自分の理解がより深まることもあります。こうした経験は、多くの人が感じたことがあるのではないでしょうか。

さらに、過去に書いた記事や資料を振り返ると、「当時はこんなことに苦労していたんだな」と、自分の成長を実感できることも少なくありません。そのためアウトプットは、まさに自分自身の成長した証を積み重ねていく行為ともいえます。

加えて、アウトプットを続けていると、思わぬチャンスが巡ってくることもあります。記事をきっかけに社内外の人から声をかけていただいたり、登壇や執筆の依頼が来たり、自分の発信が新しい機会を呼び込むこともあるのです。

このように個人にとってのアウトプットは、**学びを深め、成長を記録し、新たな機会を切り開く力** になります。

対外的なアピール

アウトプットは、単に「情報を外に発信する」という行為にとどまりません。組織全体に知識を共有し、学びを循環させ、組織そのものを成長させるための大切な手段でもあります。たとえば、個人が学んだことや経験したことを社内でオープンに共有することで、ノウハウや情報が特定の人に閉じることなく、チームや他のメンバーに広がっていきます。結果として、同じような課題に直面したときに「**誰かのアウトプットが役立つ**」状況が生まれ、チーム全体の生産性や対応力が底上げされていきます。

また、組織として社外に情報を発信することは、企業のブランディングにもつながります。自社がどんなことに取り組んでいるのか、どんな技術や知見を持っているのかをオープンにすることで、社外の人々に「この会社、面白そうだな」「一緒に働いてみたいな」と感じてもらえる機会が増えます。

さらに、アウトプットを続けていると、社外のコミュニティとの接点が広がることにもつながります。他社とのコラボレーションや新しい情報のインプットが自然に増え、組織全体の視野を広げたり、さらなるイノベーションのきっかけになったりすることも少なくありません。

このように組織にとってのアウトプットは、**ナレッジを循環させ、学びを加速し、組織の価値や可能性を広げる力** になります。

2.3　Qiita におけるアウトプット文化

アウトプット文化は、Qiita 株式会社のミッションやバリュー、そして事業にも深く結びついています。ここからは、Qiita 株式会社でアウトプットをどのように捉え、文化としてアウトプット日常的に続けていくために工夫していることをご紹介します。

Qiita にとって、なぜアウトプットが重要なのか

Qiita 株式会社のミッションは、「**エンジニアを最高に幸せにする**」です。私たちが日々学び続けた知見や経験をアウトプットし、他のエンジニアの学びや課題解決を支えることが、ミッションの実現に繋がると考えています。

また、私たちは「**Mastery - 自分の専門性を高めよう**」というバリューを掲げています。アウトプットは、その専門性を磨くための大切なプロセスです。そのため、学んだことを自ら発信し、チームや社外と学び合いながらスキルを高め続けることが、バリューの実現につながると考えています。

このように、Qiita 株式会社のミッションとバリューを実現するためにも、アウトプットは欠かせない存在なのです。

さらに、Qiita を使ったアウトプットは「**ドッグフーディング**」としての役割も果たしています。Qiita のユーザーとして、自ら記事を書き、UX や機能に対するフィードバックをすることで、サービスの改善に役立てています。

アウトプット文化を支える仕組み・取り組み

アウトプット文化は「やりたい」「やろう」というだけでは定着しません。Qiita 株式会社では、文化としてアウトプットを日常的に続けていくために、いくつかの仕組みやルール、環境づくりを工夫しています。ここでは、Qiita 株式会社でアウトプット文化を支えている具体的な仕組みや取り組みについて紹介します。

業務時間を使ったアウトプットの許容する

アウトプットは、Qiita 株式会社の業務そのものと深く関わっています。特に、Qiita への記事投稿は単なる情報発信ではなく、ドッグフーディングの側面もあるため、自己管理の範囲内であれば業務時間を使ったアウトプットを許容しています。

また、この取り組みは、「アウトプットしたいけれど、時間が取れない」という課題に対する解決策の 1 つにもなっています。業務時間内でアウトプットの時間を確保できることで、「忙しくて書けない」というハードルを下げ、アウトプットを無理なく習慣化しやすい環境が整えられています。

もちろん、他の業務とのバランスは個人の裁量に委ねていますが、「**アウトプットは Qiita のミッションやバリューを実現するために不可欠なものだ**」という認識が組織全体に浸透しているため、自然と「書いてみよう」「発信してみよう」という空気が生まれています。

社内勉強会

Qiita 株式会社では、定期的に社内勉強会を開催しています。この勉強会は、テーマや目的ごとに、エンジニアやデザイナーが中心となって企画・運営しており、それぞれの専門領域に合わせた学びを深める場として機能しています。

また、勉強会を通じて得た知識や気づきを、そのままアウトプットにつなげることができるため、「何を書けばいいかわからない」といった悩みの解消にもつながっています。たとえば、勉強会で発表した内容を記事にまとめたり、チーム内で共有した学びを Qiita に投稿したりと、アウトプットのネタが自然に生まれるサイクルを作っています。

こうした勉強会は、チーム内での知識共有やスキルの向上を促進するだけでなく、**気軽にアウトプットできる場** にもなっており、アウトプット文化の定着に大きく貢献してい

ます。

アウトプットチャンネル

Qiita 株式会社の Slack には「**#アウトプットプロジェクト**」というチャンネルがあります。このチャンネルでは、記事を投稿した時やイベントに登壇した時、ハッカソンに参加した時など、メンバーが行ったアウトプットを気軽に報告し合っています。

このチャンネルがあることで、アウトプットが社内で可視化され、自然と称賛やフィードバックが集まりやすくなっています。「いいね」「すごい」「頑張ったね」といったリアクションが気軽につくことで、アウトプットすることへの心理的なハードルも下がり、アウトプット文化の定着に貢献しています。

また、「批判されたらどうしよう」という不安に対しても、ポジティブなリアクションが中心となることで、安心してアウトプットできる環境づくりを意識しています。

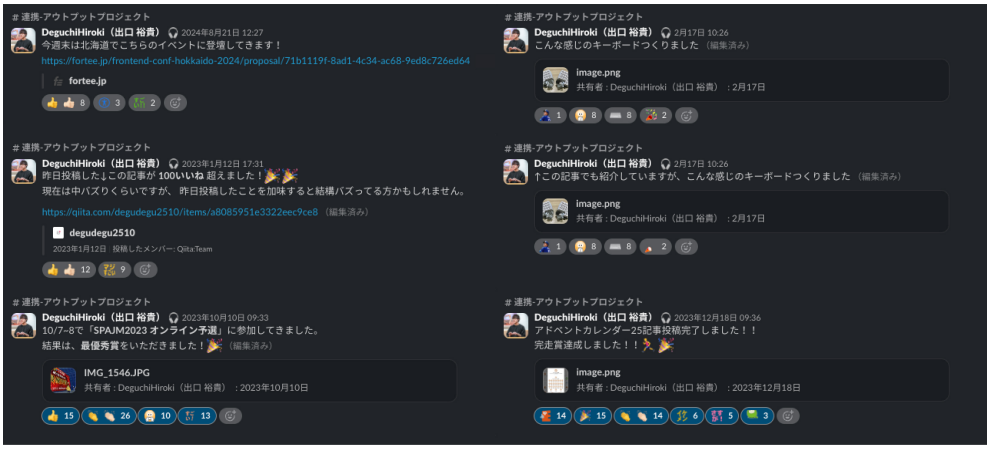

図 2.1: アウトプットチャンネルでアウトプットを報告している様子

社内表彰プログラム

Qiita 株式会社では、「**社内表彰プログラム**」というアウトプットを称える制度を設けています。この取り組みでは、Qiita の機能である「Contribution」を指標にして、積極的にアウトプットしているメンバーを表彰しています。そして、一定の Contribution に達したメンバーには、Qiita のオリジナルグッズが贈られます。

また、この取り組みは、毎月開催される Qiita 株式会社員が全員参加するミーティングで、表彰者を発表しています。メンバーのアウトプットを全体に共有することで、アウトプットに対するモチベーションが高まり、アウトプット文化の定着に貢献しています。

■クリエイターの表彰基準（エンジニア・デザイナー）

Contribution	2000 達成	1000 達成	500 達成	250 達成
景品	Qiita ロゴパーカー	Qiita ロゴ T シャツ	Qiitan ぬいぐるみ	Qiitan バッジ

■ビジネスの表彰基準

Contribution	300 達成	100 達成	50 達成	250 達成
景品	Qiita ロゴパーカー	Qiita ロゴ T シャツ	Qiitan ぬいぐるみ	Qiitan バッジ

2.4 まとめ

ここまで、Qiita 株式会社のアウトプット文化について紹介してきました。

Qiita 株式会社ではアウトプットを単なる情報発信ではなく、**自分たちの学びを深め、チームや組織の成長を促し、プロダクトそのものをより良くするための重要な行動** だと考えています。また、ミッションである「**エンジニアを最高に幸せにする**」、そしてバリューである「**Mastery - 自分の専門性を高めよう**」を実現するためにも、アウトプットは欠かせない文化として根付いています。

しかし、この文化は最初からあったものではありません。誰もが直面する「何を書けばいいかわからない」「批判されたらどうしよう」「忙しくて時間が取れない」などの課題を 1 つひとつ解決しながら、業務時間でのアウトプット許容し、社内勉強会の開催や社内表彰プログラムなどの仕組みを整え、少しずつアウトプットが「特別なこと」ではなく「**日常の一部**」として浸透していきました。

アウトプット文化は、一日にしてなるものではありません。**小さなアウトプットの積み重ねが、やがて個人やチームの成長を促し、組織全体の学びと進化につながる**。それが、Qiita 株式会社が実践してきた文化づくりのプロセスです。

この記事が、「アウトプットをもっと続けたいけど、うまくいかない」「自分やチームにアウトプット文化を根付かせたい」と考えている方々のヒントやきっかけになれば幸いです。

第 3 章

Qiita の社内勉強会

3.1　はじめに

Qiita 株式会社でエンジニアをしている山田航です。私は新卒入社から現在までの約 4 年間、社内勉強会に積極的に参加してきました。また、勉強会を自ら主催することもありました。新卒の頃は技術的な知識に自信がなく不安を感じていましたが、先輩社員のサポートや勉強会での学びを通じて、徐々に成長を実感できるようになりました。この経験から、勉強会は成長の大きなきっかけになると確信しています。近年、技術の変化が加速しており、社内でも継続的な学習の重要性が高まっています。私自身、勉強会を通じて多くのことを学び、成長できた経験から、そのノウハウを共有したいと考えました。

この章では、これまでの経験を通じて得た社内勉強会に関する知見を皆さんにお届けします。成長したい方や勉強を習慣化したい方、組織の技術力を底上げしたい方に、ぜひ読んでいただきたい内容となっています。「勉強会を始めたいけれど、なかなか人が集まらない」「勉強会を始めたものの、継続するのが難しい」といった悩みを抱えている方もいるのではないでしょうか。この章では、これらの課題に対する具体的な解決策も紹介します。

本章では、まず、私が経験した 4 つの社内勉強会の事例を紹介します。次に、これらの事例から得られた知見をもとに、個人と組織、そして社外にもたらすメリットについて詳しく解説します。さらに、勉強会を成功させるための具体的な準備や運営のコツ、参加者のモチベーションを高めるための工夫など、実践的なノウハウを紹介します。この章を通じて、社内勉強会の効果的な運営方法を学び、実際に活用できる知識を得ていただければ幸いです。

3.2 Qiita 株式会社でこれまでに開催された社内勉強会

Qiita 株式会社では、これまでに数多くの社内勉強会が開催されてきました。この節では、その中から特に印象に残っている 4 つの勉強会を紹介します。

Case 1: 新機能勉強会

新機能勉強会は、私が新卒 2 年目のときに主催した勉強会です。この勉強会では、プログラミング言語やライブラリの新機能について、毎週 2 人がそれぞれ 15 分ずつ事前に調べてきた内容を発表し、その後質疑応答を行う形式です。最初のテーマは Ruby 3 の新機能で、続いて React 18 の新機能についての勉強会を開催しました。参加者は 9 人で、約 1 年間続きました。

この勉強会を主催したきっかけは、自分自身が成長に伸び悩んでおり、成長が必要だと感じていたことにあります。そして、習慣化が苦手だったため、自らを勉強せざるを得ない環境に置く必要があると考え、勉強会を開催することにしました。

また、勉強会を開始した翌年に、プログラミング言語 Ruby のカンファレンスである『RubyKaigi』にはじめて参加しました。RubyKaigi では、Ruby の新機能に関する説明が多く、内容も難しいものでしたが、勉強会のおかげでその内容を理解できました。

Case 2: 検索システム勉強会

検索システム勉強会は、書籍『検索システム』[*1] の発売をきっかけに始まりました。この勉強会では、毎週 1 時間、当番の人が 1 章分を事前にまとめて、その内容を説明します。テーマが「検索システム」という少し難しいものであったこともあり、参加者は 4 人という少人数でしたが、そのおかげで少数精鋭のメンバーで深い議論ができました。

私は参加募集を知り、もともと検索システムに興味があったので参加することにしました。この勉強会を通じて、本の内容が Qiita の検索システムではどのように実装されているかを知るきっかけになりました。また、この検索システム勉強会の後、Qiita の質問機能に検索機能を追加するプロジェクトが立ち上がりました。そのプロジェクトを私が担当することになったのですが、勉強会で得た知識を活かし Qiita の機能改善に貢献できました。

[*1] 打田智子・他（2022）『検索システム — 実務者のための開発改善ガイドブック』ラムダノート.

Case 3: お勉茶会

「お勉茶会」というのは、「お茶会」と「勉強会」を掛け合わせた名前の会です。この会はデザイナーの勉強会であり、毎週 2 人がそれぞれ 15 分ずつ事前に調べてきた内容を発表し、その後で質疑応答を行う形式になっています。最初は Qiita のアクセシビリティを改善していくためにアクセシビリティに関する知識を身に付けることが目的でしたが、その後はアクセシビリティに限らず自由なテーマで発表が行われています。

この「お勉茶会」は現在も続いており、もうすぐ 1 年になります。続いている秘訣は、参加のハードルを下げることにあります。「お勉茶会」という名前も「お茶会」のように気軽に参加できる雰囲気を意図しています。また、業務が忙しいときは延期するなど、柔軟に対応しています。

私はエンジニアなのでこの会には参加していませんが、エンジニア以外のメンバーも積極的に勉強会を行っていることを紹介したかったので、挙げさせていただきました。ちなみに、「お勉茶会」という名前ですが、実際にお茶を飲んでいるわけではないそうです。

Case 4: パタヘネ読書会

パタヘネ読書会は、書籍『コンピュータの構成と設計』[*2]（通称パタヘネ）を読むための会です。この読書会は毎週 1 時間、各自が自分のペースでパタヘネを読み進める「もくもく会」スタイルで行われます。ただし、途中でわからないことがあった場合は、参加している他のメンバーに自由に質問ができる環境が整っています。

この読書会の特徴として、業務と直接関係がないため、勤務時間外に勉強会を実施しています。参加するメンバーはそれぞれ、自分の目標を設定して取り組みました。たとえば、「CTF で解ける問題を増やす」や「プログラムの効率化に関する知識を得る」など、個々のニーズに合わせた目標が挙げられました。

3.3　社内勉強会のメリット

社内勉強会にはさまざまなメリットがあります。この節では、個人や組織が得られるメリット、そして、得られた知識を社外にアウトプットするメリットも紹介します。

[*2] デイビッド・A・パターソン著, ジョン・L・ヘネシー著, 成田 光彰訳（2021）『コンピュータの構成と設計 MIPS Edition 第 6 版』日経 BP.

個人が得られるメリット

勉強することで個人が成長できるのはもちろんのこと、社内勉強会には多くの利点があります。まず、勉強会を定期的に行うことで、継続的に学ぶ習慣が身につきます。定期的に勉強する環境が整っているため、自分が担当する週にはしっかりと準備する必要があります。実際に、私が勉強会を開催したのも、この習慣を身につけるためでした。

一方で、個人で勉強する場合、すべてを自分で調べて理解する必要があり、それが負担になることもあります。この過程で自分で調べる能力が身につくこともありますが、勉強会では自分が担当する回以外は他のメンバーが準備をしてくれるため、勉強にかける時間を抑えつつ、効率よく学ぶことができます。

さらに、勉強会には一緒に勉強する人がいるため気軽に質問ができ、1 人で勉強していたら知り得なかった知識まで身につけることが可能です。これにより、深い理解と新たな知識の獲得が期待できます。

組織が得られるメリット

社内勉強会は組織にも多くのメリットをもたらします。まず、チーム全体の技術力を底上げできます。特に、チームの課題として特定の技術に対する知識不足が挙げられる場合などに非常に有効です。先述の「お勉茶会」も、アクセシビリティの知識を身に付けることを目的として始まりました。

また、勉強会を通じて共通言語が形成されます。例としてレビューで良くない設計を指摘する場合を考えてみます。相手が何も知らない状態だと、なぜその設計が良くないかを伝えるのは難しいです。それが、設計の勉強会を行うことで、「単一責任の原則に違反している」といった具体的な指摘がスムーズに伝わるようになります。これらの取り組みにより、開発パフォーマンスの改善が期待できます。

得られた知識を社外にアウトプットするメリット

社内勉強会で得られた知識を外部にアウトプットすることには、さらなるメリットがあります。たとえば、Qiita に記事として投稿することや、社外の勉強会で発表するといった方法があります。実際に、新機能勉強会の開催後、参加したメンバーによって Ruby の新機能に関する多くの記事が Qiita に投稿されました。このように、得られた知見を外部にアピールすることで、企業の技術的な取り組みに対する認知が広まり、企業の採用活動やブランディングにも貢献できます。第 2 章でアウトプット文化について記載されているので、詳しくはそちらを読んでみてください。

3.4　社内勉強会を開催するコツ

勉強会を開催するためには、準備が必要です。私が過去に社内勉強会を主催した際にもさまざまな準備をしました。当時は Qiita で行われた勉強会を参考に試行錯誤しましたが、今振り返ってみると次の 3 つは特に重要だったと感じています。

- 勉強会の内容・形式を決める
- メンバーを集める
- 承認を得る

それぞれについて、どのように行ったか説明していきます。

勉強会の内容・形式を決める

まず、内容が決まるまでにやるべきこととして、参加の目的を明確にすることが重要です。自分が学びたいことや、チームで学んでほしいことを考慮し、どのような人に参加してほしいかを具体的に考えましょう。そして、それに基づいて、目的が達成できる内容であり、かつ多くの人が参加してくれそうなテーマを選ぶことが大切です。

次に、形式を決める際に考慮すべき点についてです。勉強会の頻度としては、週に 1 回、30 分程度の時間を設けることをお勧めします。間が開くと内容を忘れがちになり、逆に 1 時間は参加者にとって疲れる可能性があります。また、参加のハードルを下げる工夫も重要です。たとえば、1 人あたり 15 分の発表にするなど、負担を軽減する方法を考えるとよいでしょう。

さらに、参加を任意にするか全員参加にするかについても、チームの状況に合わせて検討しましょう。特に、共通の課題感や認識が揃っている場合には、全員参加の形式を取ることで、チーム全体の理解を深めることができます。一方で、任意参加にすることで、各自が自発的に学ぶ意欲を高める選択肢も考慮に入れてみてください。

メンバーを集める

特に任意参加の場合において、どのようにメンバーを集めるかが重要となります。実際、メンバーが集まるかどうかは不安に感じるところです。しかし、メンバーを集めるためのコツがあるので、参考にしてみてください。

まず、仲のよいメンバーや信頼できる人たちに対して、「こういう勉強会を企画しているのですが、開催したら参加しませんか」のように事前に根回しをしておくとよいでしょう。また、テーマについて詳しい知識を持っている人がいる場合は、その人に対して事前に参加してほしい旨を伝えておくことも効果的です。ある程度の仲間を増やした上で、勉

強会の形式をブラッシュアップし、全体に対して募集をかけるとスムーズです。

さらに、勉強会の内容や形式を明確にしておくことで、参加者の関心を引きやすくなります。もし、参加人数が期待どおりに増えない場合には、参加のハードルを下げることもひとつの手です。具体的には、参加しやすい時間帯や内容に調整するなど、柔軟な対応を考慮してみてください。

承認を得る

社内勉強会を勤務時間内に行う場合は、上司の承認を得る必要があることが多いです。勉強会に対して理解のある会社であれば承認は得やすいのですが、理解がない場合は承認を得るために苦労する可能性が高いです。

承認を得るために特に重要なのは、「どのような成果が得られるのか」を明確に説明することです。その勉強会を通じて参加者がどのような知識を身に付け、会社にどのような効果があるのかを具体的に示すことが求められます。勉強会の形式を工夫して、効果を高めることもできます。たとえば、参加者が実際に手を動かして、自分たちが開発しているプロダクトで試してみるといった工夫が考えられます。

その他、説得力を増すためのテクニックとして、開催期間を明確にしておくことや、参加希望者の人数をアピールすることが挙げられます。また、提案の際は及び腰になりやすいので、「勉強会を開催したい気持ち」をしっかりアピールすることも重要です。

3.5　おわりに

この章では、Qiita の社内勉強会やそのメリット、開催するコツについて紹介しました。ぜひ社内勉強会に参加したり、自分で主催してみてもらえたら嬉しいです。あなたの一歩が、職場に新しい風を吹き込み、豊かな学びの文化を育むきっかけになることを心から願っています。

第 4 章

qiita-markdown について

4.1　はじめに

Qiita 株式会社でエンジニアをしている花田拓矢です。Qiita にて 2022 年 2 月リリースの Markdown パーサーの変更、2022 年 7 月リリースのエディタのアップデートの開発を担当しました。

Qiita を支える重要な要素のひとつが、記事や質問の記述に用いられる Markdown 記法です。そして、その Markdown 記法を支えているのが、OSS として公開されている qiita-markdown[*1]です。この章では、qiita-markdown の概要から、そのコンセプト、拡張性、Qiita での利用例までをご紹介します。

4.2　qiita-markdown とは

qiita-markdown は、Qiita で利用されている Markdown 処理を行うための Ruby gem であり、Markdown で記述されたテキストを HTML に変換する役割を担っています。

Qiita では、記事や質問の作成はもちろん、コメント、回答、Organization の概要、リリースノート、プロフィールなど、多岐にわたる場所で Markdown が利用されています。これらの場所で一貫した Markdown の解釈と表示を実現するために、qiita-markdown が使われています。

[*1] https://github.com/increments/qiita-markdown

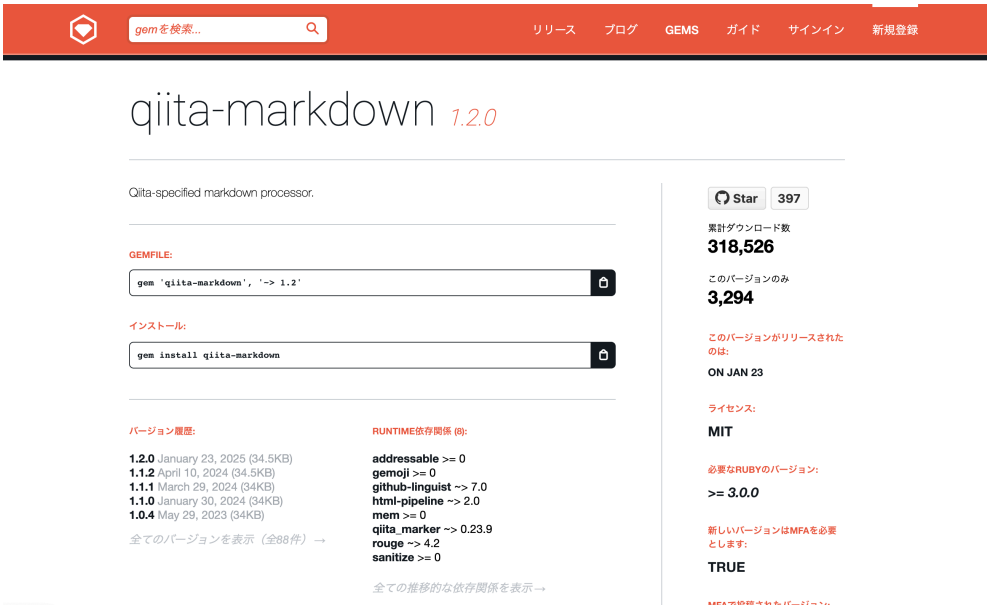

図 4.1: qiita-markdown | RubyGems.org

4.3 コンセプト

qiita-markdown の中核をなすのは、Processor と Filter という 2 つの概念です。

Processor の役割

Processor は、Markdown テキスト全体の処理を統括する役割を担います。内部に Filter のリストを保持しており、これらの Filter を登録された順番に実行していくことで、Markdown テキストを段階的に HTML へと変換します。

Processor は、Markdown テキストを受け取ると、最初の Filter にそのテキストを渡します。最初の Filter は受け取ったテキストに対して特定の変換処理を行い、結果を次の Filter へと渡します。このバケツリレーのような処理が、Filter のリストの最後まで繰り返されます。各 Filter は、たとえばシンタックスハイライト、絵文字の変換、Qiita 独自記法の変換など、特定の変換タスクに特化しています。Processor は、これらの Filter が連携して動作するように、全体の流れを制御する役割を担っています。

Filter の役割

Filter の役割は、Markdown テキストに対する個別の変換処理を行うことです。各 Filter は、特定の変換タスクに特化したモジュールであり、それぞれが独立した処理を行います。これらの Filter は、Processor によって登録された順番に、1 つずつ実行されていきます。

各 Filter は、前の Filter から渡されたテキストや HTML を受け取り、自身の変換処理を適用した後、変換結果を次の Filter へと渡します。この連続的な処理を通じて、Markdown テキストは段階的に変換され、最終的な HTML として出力される形になります。

具体的な Processor、Filter を見る

`Qiita::Markdown::Processor`[*2]を例に処理の流れを見ていきます。`Qiita::Markdown::Processor` はデフォルトでは次の Filter が設定されており、順番に処理が行われていきます。

リスト 4.1: Qiita::Markdown::Processor のデフォルトの Filter

```
module Qiita
  module Markdown
    class Processor < BaseProcessor
      # 略

      def self.default_filters
        [
          Filters::QiitaMarker,
          Filters::HeadingAnchor,
          Filters::UserInputSanitizer,
          Filters::ImageLink,
          Filters::Footnote,
          Filters::CodeBlock,
          Filters::CustomBlock,
          Filters::Checkbox,
          Filters::Toc,
          Filters::Emoji,
          Filters::SyntaxHighlight,
          Filters::Mention,
          Filters::GroupMention,
          Filters::ExternalLink,
          Filters::InlineCodeColor,
```

[*2] https://github.com/increments/qiita-markdown/blob/v1.2.0/lib/qiita/markdown/processor.rb

```
          Filters::FinalSanitizer,
        ]
      end
    end
  end
end
```

これらの Filter のうち、重要な役割をもつ 2 つを見ていきます。

Qiita::Markdown::Filters::QiitaMarker

`Qiita::Markdown::Filters::QiitaMarker` は、入力された Markdown テキストを HTML に変換するための Filter です。

この Filter は、Markdown テキストを HTML に変換することで、後続の Filter が HTML を加工できるようにします。

また、`Qiita::Markdown::Filters::QiitaMarker` は、内部で qiita_marker[*3] という Markdown パーサーを利用しています。qiita_marker は、GitHub Flavored Markdown[*4]をベースに、Qiita 独自の記法を解析するための拡張がされています。

ここで詳しくは紹介しませんが、qiita_marker の機能の行番号の付与により、Qiita のエディタでのスクロール同期が実現されています。詳しくは次の Qiita 記事をご覧ください。

Qiita のエディタの「スクロール位置の同期」はどのように実現されているか - Qiita
https://qiita.com/ohakutsu/items/2246dd476bf76e2a034e

Qiita::Markdown::Filters::UserInputSanitizer

`Qiita::Markdown::Filters::UserInputSanitizer` はユーザーが入力した Markdown テキストに含まれる HTML タグをサニタイズすることを目的とした Filter です。

Qiita では、ユーザーが記事やコメント内で HTML タグを直接記述することが許可されています。これにより、ユーザーはより柔軟な表現ができますが、同時にセキュリティ上のリスクも生じます。たとえば、悪意のあるユーザーが JavaScript コードを埋め込んだり、サイトのレイアウトを崩したりする可能性があります。

このようなリスクを軽減するために、許可された HTML タグのみを通過させ、それ以

[*3] https://github.com/increments/qiita_marker
[*4] https://github.github.com/gfm/

外のタグを削除します。

4.4 qiita-markdown の拡張性

qiita-markdown の最大の特徴は、その高い拡張性にあります。Filter を入れ替えるだけで、Markdown の解釈や変換処理を自由に変更できます。

Filter を入れ替えるだけで拡張可能な仕組み

qiita-markdown では、Filter を Processor に登録することで、Markdown の変換処理に組み込むことができます。この仕組みにより、次のような独自の Filter を作成することで、任意の機能を追加できます。

- 画像を遅延読み込み
- 特定のサービスの埋め込み対応
- PlantUML や Mermaid のサポート
- 特定のリンクをダウンロードリンクへ変換

リスト 4.2: 画像を遅延読み込みする Filter の例

```
class LazyLoadImageFilter < HTML::Pipeline::Filter
  def call
    doc.search('img').each do |img|
      img['loading'] = 'lazy'
    end

    doc
  end
end
```

Qiita での活用例

qiita-markdown の拡張性を活かし、Qiita では用途に応じて柔軟に Markdown の処理をカスタマイズしています。

Qiita では、記事、記事の目次（ToC）、コメント、質問、回答、回答コメント、Organization の概要に加え、Qiita 運営が管理する記事投稿キャンペーンの説明、利用規約、プライバシーポリシーなど、多岐にわたる箇所で Markdown が利用されています。しかし、これらのすべてで記事と同様の Markdown 機能が提供されているわけではありません。

たとえば、利用規約やプライバシーポリシーでは、X（Twitter）のウィジェット表示、リンクカード、PlantUML や Mermaid によるダイアグラム表示は許可されていません。このように、Markdown が使われる各シーンの要件に合わせて、qiita-markdown の Processor と Filter を使い分けています。記事の作成にはすべての機能を含む Processor を使用し、利用規約などの特定のページではセキュリティやページ固有の要件を考慮して、必要な Filter のみを含むカスタム Processor を使用しています。

このアプローチにより、各利用シーンで必要とされる機能のみを提供し、不適切な記法やセキュリティリスクを排除しています。たとえば、XSS 攻撃のリスクがある特定の HTML タグを許可しない、または特定の外部リソースの埋め込みを制限するなどのカスタマイズが可能です。

このように、qiita-markdown の柔軟なアーキテクチャは、Qiita 内外の多様なニーズに応じた Markdown 処理のカスタマイズを可能にし、ユーザー体験とセキュリティの両方に対応できるようになっています。

4.5 まとめ

qiita-markdown は、Qiita を支える重要な要素の Markdown を支える OSS であり、その柔軟性と拡張性は、自由なカスタマイズを可能にしています。この記事を通じて、Qiita の Markdown に少しでも興味を持っていただけたら幸いです。

もし qiita-markdown に興味を持っていただき、よいと思っていただけたら、GitHub リポジトリへスターをしていただけると嬉しいです。

第 5 章

ダークテーマの導入

5.1　はじめに

デザイナーの綿貫佳祐です。Qiita では 2023 年 9 月 8 日よりダークテーマの UI を提供しており、最初期の実装を私が担当しました。サービスローンチから約 12 年と、長い時間を経てからダークテーマを導入した、数少ない実例としてお届けします。

5.2　ダークテーマの要望と、提供を決定するまでの葛藤

かねてよりダークテーマの要望は多くいただいていました。少なくとも 2017 年には要望をいただいており、それをもとにした議論もなされています。社内でも実施したい意見ばかりだったのですが、対応工数の多さと ROI の不明瞭さからなかなか実現には至りませんでした。

図 5.1: 当時の議論の一部

しかし 2021 年 9 月に転機が訪れます。CX 向上グループという、目標を数値で追わず顧客体験の向上だけに焦点を当てる部署[*1]が発足したのです。私はその部署の統括をしており、体験向上にあたってダークテーマの導入は欠かせないと判断しました。

ただし、一度にすべてのページに導入しようとすると膨大な工数がかかります。一部のページだけへの導入や順次導入であればまだ実現可能性が高く、そういった方法でも要望は叶えられるのか、Qiita Discussions[*2]上で意見を募りました。

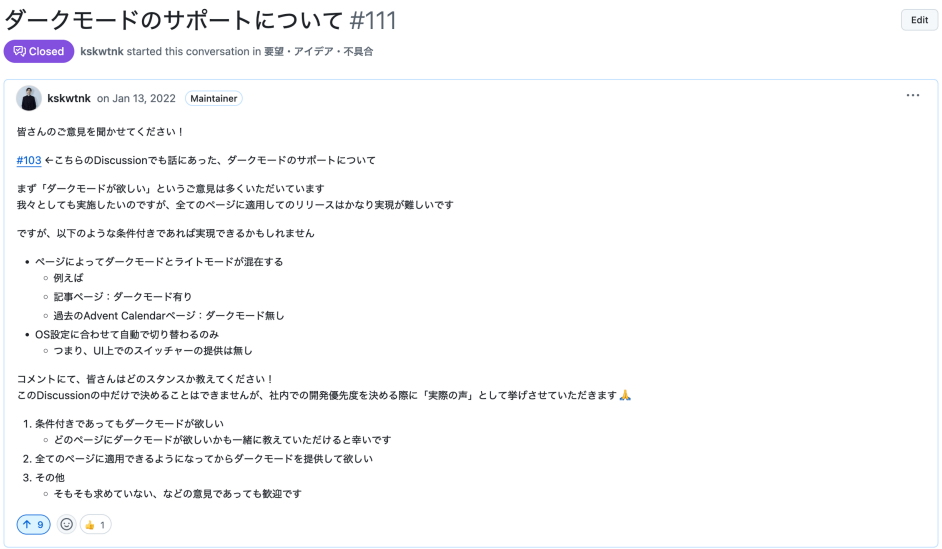

図 5.2: Qiita Discussions での呼びかけ

結果「記事ページやフィードページなど主要なページだけでもよいから対応して欲しい」という方が多いことが分かりました。そのため、ベータ版として一部のページだけから提供し、徐々に導入範囲を増やしていくことに決定しました。

5.3 ベータ版機能として提供

重要なのは、できるだけ既存コードから変更箇所を増やさないことでした。Qiita では歴史的経緯から Sass や CSS in JS が混在していて、ページ種別によってスタイリングの手法が異なっています。その状態でさらに変更や分岐を増やしては、将来に禍根を残すこと必至です。ですから、できるだけ小さな変更でダークテーマを実現すると決めました。

[*1] CX 向上グループの役割や活動についてはこちらの記事でも紹介しています。https://qiita.com/kskwtnk/items/1d814c4d23e20829234c

[*2] GitHub Discussions を利用した、Qiita 運営とユーザーの皆様とのコミュニケーションの場です。

ダークテーマを考える際、本来はライトテーマの UI から見直すべきです。 単純に今ある UI の背景を暗くして文字を明るくするだけでは、エレベーションやコントラストが変わってしまうかもしれません。その場合、ライトテーマとダークテーマとで視覚的な情報の優先度が変わってしまうともいえます。

また、背景色が違えば前面にある色の見え方も変わります。そのため、同じような色に見せるためには手動での調整[*3]が必要です。 しかし前述のとおり変更箇所をできるだけ小さくしたかったため、この時期にはライトテーマの UI やカラーパレットにはほぼ手を入れていません。一部、ページによってバラバラだった色、たとえば背景色が少し暗かったり明るかったりブレがあったのを統一した程度です。

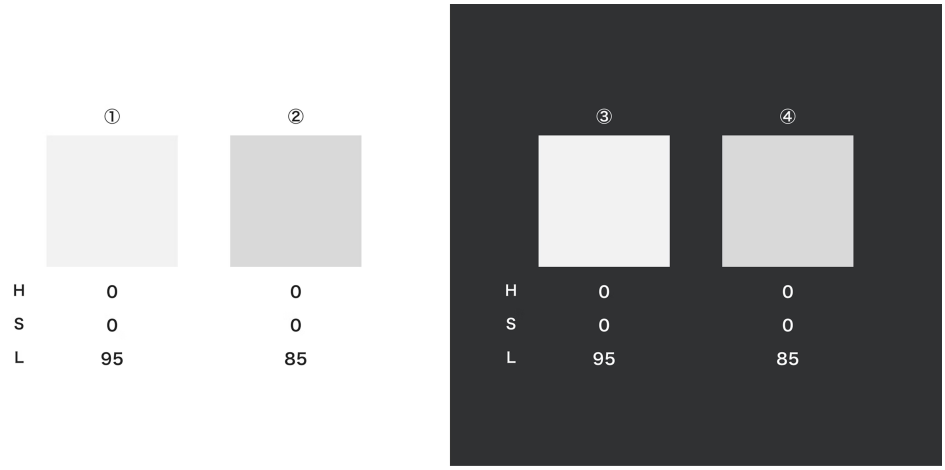

図 5.3: 背景色の明暗の違いによる色の見え方の違い

ライトテーマとダークテーマを切り替える機能についても、ベータ版では省略しています。OS がライトテーマなら Qiita もライトテーマ、OS がダークテーマなら Qiita もダークテーマという簡単な作りにしていました。サイト内で切り替えられるようにするとか、アカウントの設定を保持できるようにすると、どうしてもスコープが広がってしまいます。ですが、CSS カスタムプロパティを複数定義するだけであれば、比較的簡単に、追加の実装は不要だったためこのように判断しました。CSS の設定については、簡単なイメージではありますが次にコードを載せておきます。

すべては、できるだけ早く提供するための選択です。

[*3] 人間の目は不思議なもので、機械的に HSL の値を X ずつ変える、といった調整ではなかなか自然に見えません。

リスト 5.1: CSS カスタムプロパティによるライト・ダーク切り替えのイメージ

```
:root {
  --color-gray-0: #fff;
  --color-gray-10: #f5f6f6;
  --color-gray-20: #edeeee;
  /* ... */

  --color-background: var(--color-gray-10);
  --color-surface: var(--color-gray-0);
  --color-text: rgb(0 0 0 / 87%);
}

@media (prefers-color-scheme: dark) {
  :root{
    --color-background: var(--color-gray-110);
    --color-surface: var(--color-gray-100);
    --color-text: rgb(255 255 255 / 87%);
  }
}
```

初回リリースは記事ページだけで、次に各種フィードページ、次にユーザーページとタグページ、とかなり狭い範囲ごとにリリースをしました。全ページを一度にリリースすると余りにも遅くなってしまう他、不具合が出た際の影響が大きくなるのを避けるためです。実際新規にリリースしたページのうち、一部のセクションが黒背景に黒文字になってしまい、まったく読めなくしてしまったことがありました。細心の注意を払いつつ実装・検証をしていたのですが、CSS の変更範囲を把握しきれず。結果論ではありますが、一度に全ページに適用していたらより多くのご迷惑をおかけしたと考えると、狭い範囲ごとにリリースして良かったです。

5.4 正式機能として提供

ベータ版として提供する間、いただいたフィードバックをもとに順次修正を施していました。見た目の良し悪しだけでなく「ダークテーマが表示される直前に一瞬ライトテーマが表示され、画面がちらつく」といった報告もいただき、CSS 関連の設定を一部見直すなども実施しました。一部のページしか対応できていない間や、上記のちらつき問題を修正できる前は「明暗のコントラストが増えてむしろ目が破壊される」といったご意見もいただきました。多くの要望があって対応したにもかかわらず、感謝や喜びの意見はあまりいただけず、正直にいうと精神的につらい時期でした（もちろん、まったくもってそのとおりなご意見であるため、私に非があるのですが）。

しかし徐々に改善が進み、半年ほど経ったのちに、全体に提供しても問題無いクオリティに達したと判断しました。2023 年 9 月 8 日に正式版機能として、ログインしているすべてのユーザーに向けて提供を開始しました。

正式版として提供した際も、一部表示回数が少ないページはダークテーマ未対応でしたが、少し経ってすべてのページをダークテーマに対応させました。また、この頃には Qiita 内の設定としてテーマ変更をできるようにしています。

5.5　アクセシビリティの向上

正式版として提供しましたが、もともとのカラーパレットから変更していなかったのもあり、コントラスト比が低い箇所も存在してしまいました。ただしそれはダークテーマ固有の問題ではなく、Qiita 全体の問題でもあった[*4]ため、正式版をリリースして落ち着いたのちにサイト全体のカラーパレットを見直しました[*5]。

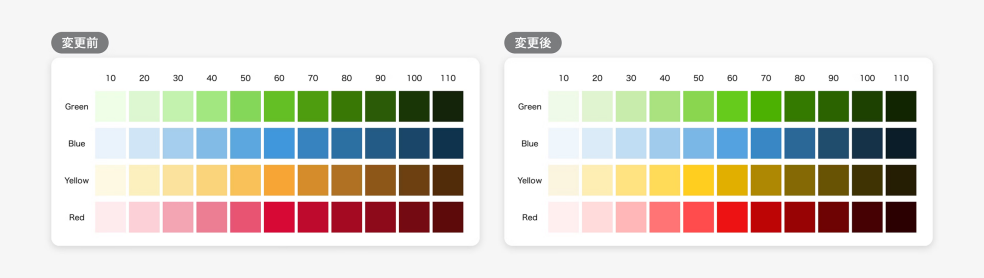

図 5.4: 新旧のカラーパレット

見直す際の方針として、ブランドカラーはカラーパレットには使わないことと、両テーマで同じカラーパレットを使うことを挙げていました。ブランドカラーはどうしてもコントラスト比が低くなってしまうのと、テーマ別に完全にカラーパレットを分けると管理が大変になる、という理由です。

実際にパレットを刷新する際は、両テーマの surface 上でコントラスト比の基準を満たしているかをチェックしていました。また、色の種類を Base, Container, Text, Border に分類するなど、ルール設定も新たにしました。これによりテーマを切り替えながら、都度コントラスト比を気にしながら UI 制作をせずとも、常に表示崩れや低コントラスト問題が起きないようになりました。

[*4] たとえばロゴの色である`#55c500`をボタンやリンクテキストなどに使用していましたが、コントラスト比が低く改善する必要がありました。

[*5] 詳細はこちらの記事で紹介しています。 https://qiita.com/degudegu2510/items/d3095fd83dff9e61f54f

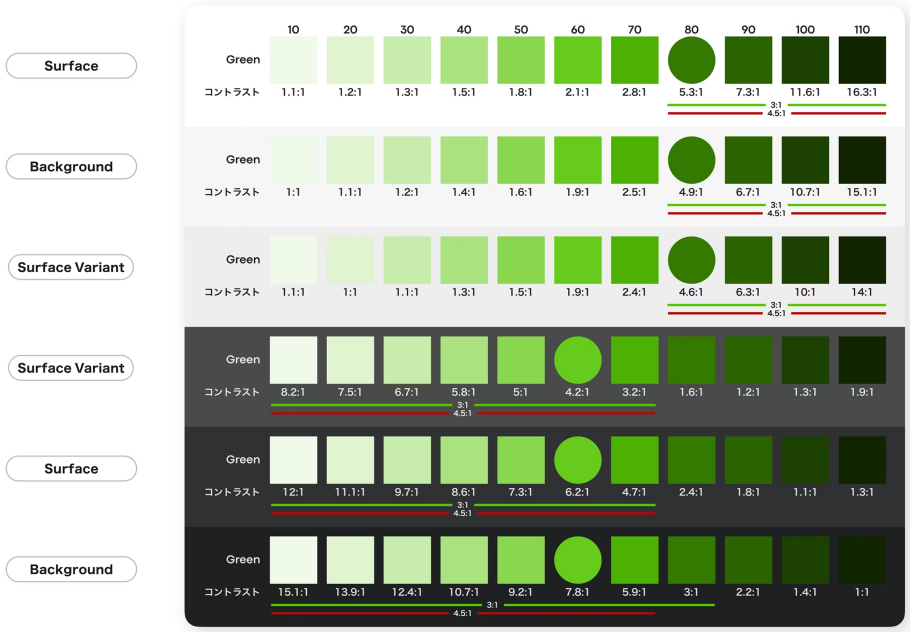

図 5.5: スケールのうち、4 色以上がコントラスト比を満たしているかのチェック

5.6 まとめ

2025 年現在、すべてのページはダークテーマに対応しており、新たに制作する UI もダークテーマが前提となっています。これにより、夜中の閲覧がしやすいなど、多くのユーザーからご好評の声をいただけています。

12 年以上経ったコードに後からダークテーマを適用するのは非常に骨の折れる作業でした。しかし、しっかりと向き合い再設計したおかげで、現在は苦労せずに両テーマの UI を制作できています。

Qiita の裏側を知っていただけるだけでも幸いですし、実装方法や導入方法に悩んでいる方の助けになれればなお幸いです。

第 6 章

15 年ものの Rails アプリケーションを持続させるための取り組み

こんにちは。 Qiita の共通基盤開発グループの 千葉 （@tomoasleep） です。自分は普段は Qiita の開発に参加しながら、（領域を問わずに） 開発基盤の改善などに携わっていています。

ところで、 Qiita は 2011 年に開発を開始し、2025 年現在で 14 歳、来年には 15 歳になろうとしています。この記事ではそんな Qiita を持続させ、成長させ続けるために行っている取り組みについて （まだ途上なものも含まれますが） 紹介します。

6.1　Qiita というアプリケーションを取り巻く変遷

Qiita は最初期は Rails, CoffeeScript, jQuery 等を利用したシンプルな Web アプリケーションでした。そこから 10 数年が経ったことで、さまざまなことが変わりました。

まず、サービスや機能が増え、開発チームの規模も大きくなり、要件が複雑な開発も増えました。そして、Rails や Web アプリケーションを取り巻く技術も大きく変わりました。特にフロントエンドに関しては進化が著しく、バックエンドが返す HTML に jQuery 等でちょっとした動きを与える程度のものだったのが、React 等などのフレームワークを利用し複雑な View をフロントエンド自らが組み立てるようになり、Web アプリケーションの配信に関わる技術や関心ことも JavaScript が中心になってきています。また、Web アプリケーションに求められる品質も、エンジニアが学ぶべき技術も、これらの発展に合わせ高度化、多様化しつつあります。

こうして、アプリケーションの要件もとりまく技術も複雑になってきて、開発も、「ごく少数のメンバーが全容を理解して開発する」という体制から「さまざまな専門性を持ったメンバーで分担やカバーし合う」という体制に変わってきています。この体制でもアプリケーションをうまく持続させ、より早く成長させられるようにうまく複雑さをコントロー

ルする、というのが、この変化についていくための課題といえます。

こうした課題に対して、 Qiita では主に次の形で取り組んでいます。

- 既存の技術を活用する形での分離
  - フロントエンドとバックエンドの分離
  - モジュラーモノリスの段階的導入
- ライブラリ等のアップデートを安定して高速に行うためのチーム、基盤の改善

6.2 フロントエンドとバックエンドの分離

Qiita では、 Web アプリケーションをとりまく環境の変化に合わせ、次のとおりに変更してきました。

- Rails は引き続き採用しつつも、MVC の Controller の責務を （GraphQL-Ruby を利用した） GraphQL API に移動
- View を （React on Rails を利用し） 徐々に React 化、 TypeScript 化

こう変更することで、フロントエンド技術の進化を取り入れることはある程度できるようになった一方、 Rails の上で動くフロントエンド、という形を取っているために次の課題を抱えています。

- フロントエンド、バックエンドの責務が分離しきれてないことによる問題
  - Rails or JS のどちらに何を実装するかで一部混乱が起こっている
  - フロントエンド変更をする際に、 Rails 側の変更が必要なことが多々発生し、結果として、バックエンドの知識が必要になってしまう
- 採用している技術の難しさの問題
  - Rails サーバー上で JavaScript を動かして SSR する、というマイナーな手法のため、運用上の問題が発生したり、複雑化してしまうところも
  - プレーンな React を利用しているため、メタフレームワーク（Next.js, React Router 等） が持っているような機能を再発明しなければいけないことがある
    - コード分割、SSR の最適化、パフォーマンス最適化の設定など

これらの問題を解消するために、メタフレームワーク（Next.js, React Router 等）への移行を想定した、完全なフロントエンドとバックエンドの分離をゴールとしたビジョンを立てて、そこに向かう改善を進めています。現在進めている改善は、非常に細かいアクションの積み重ねで、具体的に紹介すると細かすぎてページに収まりきらないのですが、少しだけ具体化すると次のようなものになります。

- 技術選定を徹底させる
  - 同じ役割をもつ選択肢が複数ある場合（例: Rails View と React）に片方に統一する
- 導入した技術が、ビジョンに沿って最大効果を発揮するように、レールを引き直す
  - 例: GraphQL の「クライアント側が必要なデータを取得できる」特性を活かすため、Fragment Colocation を導入し、コンポーネント主体の Fragment 定義に移行する
  - 例: 不必要な場面で使われすぎている Redux の利用を削減する

6.3 モジュラーモノリスの段階的導入

Qiita は複数のサービスのほぼすべての機能を 1 つのモノリシックな Rails アプリケーションとして開発してきました。長年の開発により、Qiita のコードは巨大になり、全体の構造の理解が難しくなり、開発する際にどのコードを変更すればいいのかも把握するのが難しくなってきました。これらの課題を改善するべく、Qiita のフロントエンド、バックエンドの両方のコードでモジュラーモノリス[*1] アーキテクチャの導入を進めています。

モジュラーモノリスとは、モノリスのように単一のアプリケーションでありつつも、アプリケーション内部を独立性の高いモジュールに分割するアーキテクチャです。マイクロサービスのように独立したサービスとして分割するのではなく、モノリスの中で独立性を高めるという点が特徴です。このため、モジュール間の境界線をあとから変更しやすいという利点があります。

具体的に Qiita では、バックエンドでは packwerk[*2], packs-rails[*3] といったライブラリを活用し、 packs/ ディレクトリに切り出して移動しています。Qiita だと、Qiita Conference[*4] などのランディングページ、振り返りレポート[*5]、アドベントカレンダー[*6] などのイベント系の機能、あとは（gem に切り出せそうな）社内ライブラリ的実装などを最初の切り出しのターゲットに選びました。独立性が高い部分でまずは切り出しを試行し、そこでうまくいったら、徐々に他の部分にも適用していく、というスタンスで進めています。

フロントエンドでも同様の切り出しを行っています。(フロントエンドでは、このようなディレクトリ構成は Package by Feature [*7] という名前で呼ばれることが多いです)

[*1] https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith
[*2] https://github.com/Shopify/packwerk
[*3] https://github.com/rubyatscale/packs-rails
[*4] https://qiita.com/official-campaigns/conference/2025
[*5] https://qiita.com/tomoasleep/yearly-summary/2024
[*6] https://qiita.com/advent-calendar/2024
[*7] https://medium.com/@vitorbritto/the-package-by-feature-approach-c62a197a8a3d

図 6.1: 実際に切り出された package

また、モジュールとして分かれていることで、その単位でリファクタリングや技術的な試行錯誤が行いやすいというメリットもあります。切り出したモジュール内で「フロントエンドとバックエンドの分離」を試行し、その知見を全体に適用していく、という活用も行っています。

6.4 ライブラリ等のアップデートを支える基盤の改善

設計面の改善もそうなのですが、サービスを長く持続させ続けるためには、ライブラリや技術のキャッチアップ、アップデートが必要不可欠です。

ECS を利用した Docker 化、Immutable Infrastructure 化によるアップデート容易な環境の整備

Qiita は Chef で構築した EC2 サーバーに対して、Capistrano でデプロイを行うというインフラになっていました。 ただ、言語等のアップデート時にサーバーの再構築が必要になるなど、運用保守の妨げになっていることがありました。

これを ECS を使った宣言的な Immutable Infrastructure[*8] であるインフラ基盤に移行しました。設定の変更の手順が単純化され、アップデートなども Dockerfile の変更だけで済むようになったため、非常に簡単にアップデートが行えるようになりました。

[*8] https://glossary.cncf.io/ja/immutable-infrastructure/

Dependabot 対応の Dashboard 化による対応の可視化

Qiita では Dependabot を用いたライブラリのアップデートを行っていて、これを各エンジニアがレビューし入れていくというフローで進めています。ある程度は各々の自主的な取り組みに委ねている部分もありますが、 GitHub Project や Findy Team+ といったツールで、これらの進捗を可視化し、対応で困っている箇所での相談しやすくしています。

テストの自動化によるアップデートの安全性の確保

継続的なアップデートを行う上で、テストの自動化は非常に重要です。Qiita では、自動テストを当たり前に書く文化が根付いていて、これにより（特にバックエンドに関しては） 高いテストカバレッジを保ち、アップデートによる影響を検知しやすくしています。

ただ、ロジックに関する変更に関してはこれらのテストが機能しているのですが、CSS 等の見た目に関するテストや、複雑なユーザーインタラクションに関する E2E テストに関しては十分とはいえない部分もあります。今後は主にフロントエンドに関するリファクタリングが増えていくこともあり、こうした変更に耐えうるようなテストを増やしていくことが課題となっています。

6.5　持続可能な開発を行うための文化

アプリケーションを長く開発し続けるためには、設計のテコ入れだけではなく、日々の開発でも持続可能性を意識した取り組みが必要です。Qiita では、次のような文化が、持続可能な開発の支えとなっています。

Why を残すためのドキュメント文化

Qiita では、GitLab Handbook[*9] を参考に、業務に関わる情報を Qiita Team 上のハンドブックに残したり、技術選定や設計方針を Qiita Team 上に残すフローが文化として根付いています。合わせて、Qiita Team や Pull Request に設計に関する情報を残す際に、「なぜその変更が必要なのか、どのような問題を解決するための変更なのか」といった背景を残すことを重視しています。今回の取組を進める際にも、Qiita Team 上に設計方針を残し、それを参照しながら進めています。

これが、長期にわたって一貫した方針で設計していく上で非常に重要な役割を果たして

[*9] https://handbook.gitlab.com/

います。現在行っているアーキテクチャの改善も、方針を決める際に、過去のドキュメントを参照し、その背景と意図を理解して過去の設計を活用した上での方針立てを行うことができています。

コードの品質を保つための Linter, 規約

Qiita では、コードの品質を保つために、初期から RuboCop, ESLint などの Linter を導入し、Linter によってコード規約を守らせるようにしています。これにより、個々人のコードスタイルの差異を減らし、（ある程度） 一貫したコードを保つことができています。

また、最近では GitHub Copilot や Gemini Code Assist などによる AI によるコードレビューも活用しています。これらにコード規約やベストプラクティスのガイドラインを与えることで、それに沿うようにレビューさせることで、 Linter だけでは表現が難しいガイドラインを浸透させることに貢献しています。

ユーザー体験向上として、設計の地層化に対処する

長年の運用や改善を進めていくと生じがちな問題として、アプリケーションの設計の地層化があります。頻繁に開発されるページとそうでないページで、設計の差異が生じ開発体験の一貫性が損なわれる、という問題です。

ただ、これは開発体験だけではなく、細かい差異がユーザー体験の一貫性にも影響が与えていることがあり、こういった地層化を減らすことは設計としてもユーザー体験としても重要と位置づけています。こういった考えのもと、定期的に地層化の解消に取り組んで、古い設計のページが少ない状態を保てています。

6.6 おわりに

Qiita は 15 年近い長い歴史をもつアプリケーションですが、開発を持続させるためには、外部の新しい技術や考えを取り入れていっています。また、日常的に新しい状態にしていく開発文化も重要で、開発チームとしての総合力が問われるところでもあります。今後も Qiita がユーザーに便利であり続けるために引き続き改善を進めていくのでよろしくお願いします。

著者紹介

第 1 章 清野 隼史 / @getty104

今回紹介させていただいたエンジニアリング文化は突発的に生まれたものではなく、先人たちが残してきた思いや取り組みの集大成だと思っています。これからの Qiita の取り組みからもこの文化を感じ取っていただけるように頑張ります！

第 2 章 出口 裕貴 / @degudegu2510

最近、快適な作業環境を求めて、自作キーボード「**killer whale**」を作ってみました。打鍵感にこだわった 1 台で、この本の執筆を通してようやく手に馴染んできたと思います。これから何十年と使っていく道具には、やっぱりこだわりたいものです。

第 3 章 山田 航 / @wataru86

第 3 章の執筆と編集長を務めさせていただきました。少しでも楽しんでもらえたなら嬉しいです。技術同人誌を作る際はできるだけ余裕を持ったスケジュールで作ることをおすすめします。想像の 10 倍くらい大変でした。

第 4 章 花田 拓矢 / @ohakutsu

第 4 章の執筆を担当しました。この本が人生初の本の執筆だったので頭を抱えることが多かったですが、なんとか納得のいく形にできました。もしこの本を読んで面白いと感じたら、ぜひ SNS で感想を投稿してもらえると嬉しいです。

第 5 章 綿貫 佳祐 / @kskwtnk

ダークテーマ UI については、意思決定から一旦の着地までに 2 年以上かかったプロジェクトで、こうして振り返ってみると感慨もひとしおです。余談ですが、執筆にあたり京極夏彦さんのようにすべてのページを句点で終わろうとしたら、急に難しくなりました。

第 6 章 千葉 知也 / @tomoasleep

近年のやっていきを書きました。今後も Qiita をプロダクトも技術もいい感じにしていくので、vol.2 出す頃には新たなやっていきが載っていると思います。次回作にご期待ください！

表紙デザイン 加藤 璃子 / @ktrk2002

表紙をデザインさせていただきました。普段は Qiita 内のイベントバナーや UI 等を作成しています。この本は、Qiita や Qiita の中の人について深く知れる本になっています。表紙も含めて、楽しんでお読みいただけると嬉しいです！

Qiita Tech Book vol.1

2025 年 4 月 16 日　初版第 1 刷　発行
著　者　Qiita 執筆部

Qiita