circleci

後悔しない テクノロジー スタック選び

急成長を遂げるスタートアップ企業の 技術リーダーの秘密とは

Rob Zuber (CircleCI CTO) [著]

何年か前の休みの日に参加したパーティでのできごとです。主催者のひとりと話をしていると、ある参加者の話をしてくれました。起業をしようとしている人物で、ぜひ話してみるべきだと言うのです。

私は、スタートアップ企業が大好きです。あるいは、そのような挑戦をしようとする人が大好きだと言っても良いでしょう。実際に話をする前から、できる限りのサポートをしたいという気になっていたくらいです。ともかく、いよいよその人物に会って、話をすることができました。

話が始まると、その人は自身のアイデアを熱心に聞かせてくれました。1 人で 30 分ほどしゃべり続けていたでしょうか。45 分くらいだったかもしれません。息継ぎのためにやっと話を止めてくれたところで、私はそのアイデアが実にすばらしいものだと言いました。実際そのとおりだったからです。そのうえで、「現在のユーザー数はどれくらいでしょうか?」と聞いてみました。

> 「いえ、今はまだ事業計画を検討しているところです。」
>
> 「そうなんですか。その事業計画の検討には、今までにどれくらいの時間をかけてきたのでしょう?」
>
> 「18 か月くらいですね。」
>
> 「ソフトウェアの開発者はいるんですか?」
>
> 「ええ。CS (コンピューター サイエンス) を勉強していた友人が協力してくれています。」
>
> 「そのご友人は何を?」
>
> 「今は一緒に事業計画の検討をしています。」

その人との会話は結局、2 時間以上続きました。私からは、過去 20 年以上にわたりソフトウェア関連のスタートアップ企業で働くなかで学んできたことのすべてを伝えました。この起業家との会話ではっきりとわかったことがあります。それは、スタートアップ企業という存在がビジネスの世界でもてはやされているのとは対照的に、その中の日々については一般にほとんど理解されていないということです。特に、スタートアップの経営という観点から技術リーダーに求められる行動については、ほとんど知られていません。

フィードバックを貪欲に収集し、必要なら短時間で大きく変更できることが重要です。

スタートアップ企業は浮き沈みが激しく、そのリーダーに対しても大きなプレッシャーがのしかかります。スタートアップ企業のリーダーは、正しい意思決定を下すと共に、ノイズのなかから重要な情報を見極め、迅速に行動しなければなりません。このような立場は、非常に強いストレスにさらされるものです。意思決定の一つひとつが、社の命運を左右するかのように感じられるからです。でも、ご安心ください。重要なことに精力を傾け、そうでないことに労力を浪費しないようにするための鍵とも言うべき戦略がいくつか存在します。

スタートアップを成功に導くうえで私が最も重要であると考えている要素は、つまるところ "フィードバックを貪欲に収集し、必要なら短時間で大きく変更できること" です。では、そのようなことができるようになるには、どうすれば良いのでしょうか。変更にかかるコストを抑えることです。変更にかかるコストが小さければ、チャンスを掴める可能性が大きくなりますし、過去の誤った意思決定に縛られることも少なくなるでしょう。この eBook の次章以降では、変更のコストを小さくするためにやるべきことや、やめた方が良いことについて、私が厳選したヒントをご紹介します。

変更のコストとは?

現代のソフトウェア　エンジニアリングの創始者のひとり、グラディ・ブーチ氏は、アーキテクチャとはシステムの構造と動作の設計に関する一連の重要な意思決定の結果であって、その重要さは、変更のコストによって計測される、という趣旨の発言[1] をしています。

変更のコストは、技術リーダーが関与できる要素のうち、最も重要なものの 1 つです。変更の誘因はビジネスの発展と共に変化します。しかし、創業間もないプロダクトマーケットフィット (PMF) を達成するまでの期間には、非常に大きな変更が必要になることもあります。

かつて Odeo と呼ばれたポッドキャスト システムは、最終的に巨大 SNS、Twitter となりました。また、ゲーミング エンジン Tiny Speck は、幅広い人気を誇るメッセージング プラットフォームの Slack へと変化を遂げました。いずれも、もともとのビジネス モデルを根底から覆す変化です。プロダクトマーケットフィットを達成するには、大小を問わずさまざまな変化をしていくよりほかありません。システムの機能を固め、変更できる余地を少なくしていると、後で大きな変化を迫られた場合に、大きな代償を支払うことになります。逆に、既存のコードを捨ててもう一度やり直す気概があれば、変更のコストを低く抑えられます (この点について、さらに詳しくは、「**消してもかまわないコードを**」のセクションを参照してください)。

プロダクトマーケットフィットを達成するまでの間は、ビジネスの方向性を 90 度、ときには 180 度変えなければならないこともあり得ます。たとえば、変更が多々発生するその時期に、ビジネス　モデルをいくつものマイクロサービスに分かれた構造に固定してしまうと、変更のコストが大いに跳ね上がることになるでしょう。ですから、事業運営のあり方を大きく、抜本的に変えることができるようなかたちで最適化しておくことが賢い選択なのです。

何より大事なのは プロダクトマーケットフィットの達成

顧客からフィードバックを集めることは、どんなビジネスであれ創業当初にできることのなかでも最も重要な作業です。最新鋭のソフトウェア　ツールを使えば、創業初日からアイデアを形にしてユーザーに届けることができます。このようなことは、これまで常に可能だったわけではありません。

昨今は AWS、Heroku、サーバーレスの時代であり、ビジネスを簡単に立ち上げることができます。私がソフトウェア開発を始めた 1998 年とは比べるまでもなく、ビジネスの歴史のなかでも類のない便利な時代になりました。最新鋭のクラウド デプロイメント プラットフォームには、アイデアをかつてないほどの早さで具現化し、ユーザーにプロダクトとして提供することができるツールが揃っています。このようなツールを使えば、ユーザーがプロダクトを使っている間もプロダクト開発を進められるので、ユーザーの意見を迅速に反映することができます。

以上を考えると、スタートアップによく見られる手法のなかには、既に時代遅れになっているものがいくつかあります。そこで、ここからはその話をしていくことにしましょう。

私がソフトウェア開発の仕事を始めたのは 1998 年のことでした。当時は今とは大きく違っていました。最初に使っていたサーバーなど、私のデスクの下に置いていたくらいです。シード　ラウンドを終えた私は、データ　センターを立ち上げるべくバージニア州のハーンドンに飛びました。スーツケースにディスク ドライブを目一杯詰めこんで。1998 年当時にインターネット上でソフトウェアを運用するといったら、こういう方法にならざるを得なかったのです。

"ステルス モード" をやめよ

ここでステルス モードと表現しているのは、スタートアップ企業がプロダクトの開発を秘密裏に進め、それについてだれかに話をしたり、見せたりすることを避ける態度です。このような態度の背景には、だれかにアイデアを盗用されるという恐怖心があります。

この戦略がなぜ良くないかを説明する前に、まずはステルス モードの背景にあるこの恐怖心に一般に根拠がないという話をしておきたいと思います。そもそも、盗んだうえで開発までできるような優れたアイデアが聞こえてこないかと、ただ座って待っているような人はいません。アイデアを盗むために費やす時間とエネルギーがあるのなら、既に独自のアイデアがあって、その開発を進めていることでしょう。

しかし、それ以上に重要なのは、自分が開発を進めているアイデアについて、同じく開発に取り組んでいるチームが少なくともあと 3 チームはあるという点です。

大切なのは、秘密裏に開発を進めることではありません。開発をスピーディーに進め、一刻も早く具体的なプロダクトとして顧客に届けることなのです。秘密裏に開発を進めていれば、開発に割くことができる時間が増えるという考え方は間違っています。だれかがあなたよりも先にプロダクトを市場に出してしまう可能性が高まるにすぎません。

秘密裏に開発を進めるのはやめましょう。開発作業は公開しながら進めることです。長時間かけて完璧なものを作り上げようとするのはやめて、完璧ではなくても公開し、いち早くフィードバックを集めることが重要です。

では、ステルス モードの対極にあるものは何でしょうか。ランディング ページです。ランディング ページを使って、実行可能なコンセプトをごく小さな規模から試すわけです。朝食の前にビジネスのアイデアを 5 つほど提示しておけば、昼食までには最も強く関心が寄せられているものがどれかわかります。私の場合、時間を無駄に使わずに、自社の市場について価値ある情報が得られました。1 時間しか時間を使っておらず、コードも一切書いていないのに、ステルス モードを採用した場合よりも良いアイデアが得られたのです。

この eBook でこれから繰り返し出てくる表現ですが、**初期のスタートアップ企業に重要なのは、プロダクトマーケットフィットの達成だけ**です。

秘密裏にプロダクト開発を進めていては、プロダクトマーケットフィットの達成は見込めません。何かを隠している限り、そこから学びを得ることはできないからです。

事業計画は立てるな、MVP を作れ

「Rob Zuber が計画を立てるなと言っていた」などという誤解が生まれないように書いておきましょう。計画を立てること自体は良いことで、かつ重要でもあります。ただし、その計画を 1 文で表現できるようにしておくことです。必要以上に先のことまで計画するのはやめましょう。つまり、次のステップの分だけ計画を立て、成果物ができたら、フィードバックを集めるのです。そして、集まったフィードバックを基に計画を修正していく。その繰り返しです。

事前に詳細な長期計画を立てないというやり方は、無計画であると受け取られることも少なくありません。しかし、実際には、いつでもチャンスを掴めるよう最適な立ち位置を維持しようという計画があるわけです。また、仕事を進めるなかで実際の計画を発展させていこうという意図もあります。

ここで「事業計画は立てるな」と書いた本当の理由は、どんなに優れた事業計画であったとしても、間違いが避けられないからです。ビジネスにおいては、どこかで間違いを犯すものです。先ほど、同じアイデアに基づいて開発を進めるチームが (自分のチームのほかに) 3 つあるという話をしました。どのチームも、それぞれ何らかの点で間違いを犯します。しかし、間違った時点で自らの間違いに気付く人はいません。この段階で取れる唯一の戦略は、できる限り早く (理想的には、他の 3 チームよりも先に) 間違いに気付き、正解を見つけたうえで、その正解に基づいた開発を始めることです。

このアドバイスに意外なところはないでしょう。しかし、これが有効なのは初期のスタートアップ企業ばかりではありません。自社がどの段階にあるかを問わず、この開発、実践、フィードバックの収集というサイクルを回していくことが重要です。どこで間違ったかを把握し、正しい方向に是正していくためには、このフィードバックのサイクルを使う以外にありません。

どれほど感動的な言葉が並んだ事業計画を立てたところで、プロダクトマーケットフィットの達成には至らないのです。

ですから、事業計画を立てるのはやめましょう。まずは自分のアイデアを基に、何か指針が得られそうな、できるだけシンプルなプロダクトを考えてみることです。プロダクトの原案ができたら、その日の午後にでも試しに作ってみましょう。できたものはだれかに見せて、その反応を見ることも忘れないでください。そして、ユーザーのフィードバックを得るためなら、どんなことでもやりましょう。18 か月もの期間も、90 ページにも及ぶドキュメントも必要ありません。

> 唯一の戦略は、
> できる限り早く間違いに気付き、
> 正解を見つけたうえで、
> その正解に基づいた開発を始めることです。

しかるべきタイミングで正しい決断を下す: 安く済ますか、お金をかけるか?

スタートアップ企業が成長し、その規模が大きくなると、技術リーダーとしての業務が急速に変わり、コードの記述からチームの方向性の決定へと変化していくことに気がつくと思います。また、この変化は予想以上に早く起こります。早ければ、エンジニアを２～３名雇った時点で起こることもあります。

チームにとって良い意思決定を下すためには、また、後悔しない技術アーキテクチャを構築するためには、いったいどうすれば良いのでしょうか。

ここまでの話で、既に重要な原則を 2 つ紹介しました。1 つが変更のコストを考えること、もう 1 つが、スタートアップで最優先すべきはプロダクトマーケットフィットの達成であると理解することです。

ここで、意思決定にあたって検討すべき要素に 3 つめを加えましょう。タイミングです。

既に知られ過ぎた話で退屈だとは思いますが、スタートアップではタイミングがすべてです。もっとも、何百万回と聞いた話だからといって、それが真実でなくなるわけではありません。タイミング、具体的には "今日何に集中し、何を無視するか" を決断することは、重要性のきわめて大きな行為です。その重要性の大きさゆえに、ストレスも大きなものとなるでしょう。プロダクトマーケットフィットの達成を何よりも優先し、他のすべてを切り捨てていくと、混乱や問題が発生し、後からその対処を迫られることになります。また、ユーザーにプロダクトをすばやく届ける、新機能や修正パッチを迅速にリリースする、方向性が正しいかどうかに関するシグナルを入手するといったことを円滑に達成するためには、近道やスマートな手段を駆使する (ハックする) こともあるでしょう。

スマートな手段があれば、積極的に駆使しましょう。そして、混乱を受け入れましょう。これは非常に難しいことだとは思います。しかし、ビジネスの成功にはプロダクトマーケットフィットの達成が欠かせません。そして、そのプロダクトマーケットフィットの達成には、今日の時点で重要な問題への対応を冷酷なまでに優先し、その他の一切を無視することが一番の方法なのです。

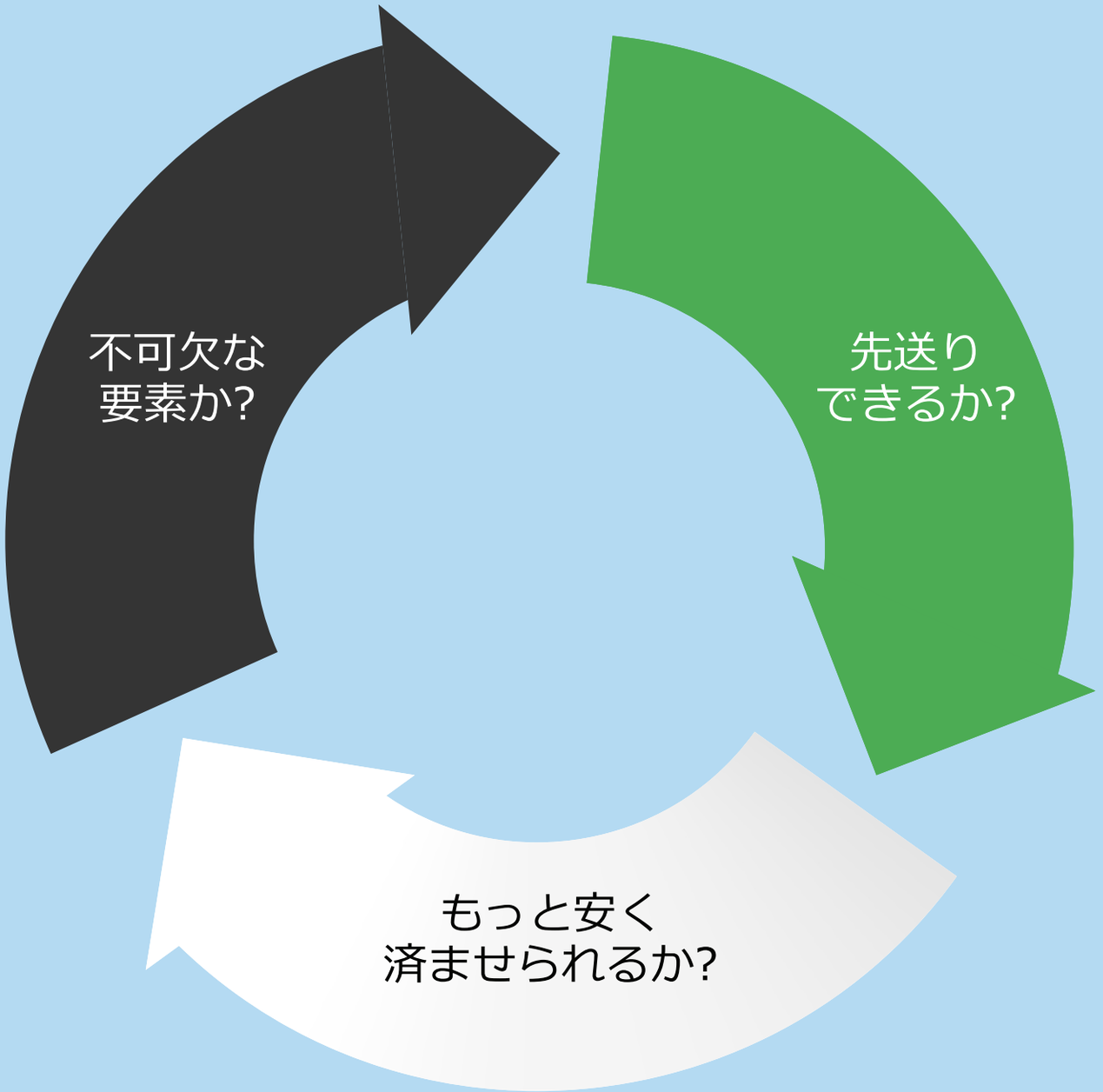
スマートな手段があれば、
積極的に駆使しましょう。
混乱を受け入れましょう。

達成に成功すれば、無視した時点に戻ってテクノロジー スタックを最適化する予算とリソースが手に入ります。

では成功しなかったとしたらどうでしょうか? 何も変わらないだけのことです。

これがエンジニアにとって難しいのはわかっているつもりです。これを読んでいるあなたの会社でも、自らの仕事に大きな誇りを持ち、細部に至るまで質の高い成果物を生み出せる非常に優秀な人材を雇っていることでしょう。

あなたの仕事は、エンジニアに対して、成果物が見事であったり、洗練されたものであったり、あるいは技術的に最先端であるということは必ずしも重要ではないという点を、うるさいくらいに何度も伝え続けることです。エンジニアに求められるのは、ビジネスにとって最善の選択となるソリューションを作り出すことだからです。


不可欠な
要素か?
先送り
できるか?
もっと安く
済ませられるか?

今は高価でも、後になれば安くなる輝かしいテクノロジーに背を向けよ

できるだけ先延ばしする、もっともっと先に

ここまでの話で、ある種の意思決定 (どれくらいの期間をステルス モードで進めるべきかなど) がまったく必要ないことがおわかりいただけたと思います。しかし、著しい成長を目指すスタートアップ企業の技術リーダーとしてはほかにも、複雑な要素が絡んだ興味深いながらも困難な意思決定を迫られ、的確な決断を下したいと思う場面が出てくることでしょう。たとえば、マイクロサービスを使うほうが良いかとか、Kubernetes クラスタのオーケストレーション方法をどうするかといったような話です。私がここで提案したいのは、こういった決断をできる限り先延ばしにしようということです。大きな決断を先延ばしにすることは、単に難しいタスクを回避する方法の 1 つというだけにとどまりません。重要な戦略の 1 つにもなり得るものです (私はむしろそう据えるべきだと考えており、これから説明していきます)。必要がない限り、決断を下すのはやめましょう。また、必要な決断であっても、できるだけ先延ばしにするようにしましょう。

初期のスタートアップ企業の技術リーダーは、好むと好まざるとにかかわらず、あいまいな状態に置かれることを覚悟しなければなりません。事業はどのようになるか、成功するかしないか、規模を拡大する必要があるか、必要があるのならどのように拡大するべきかなど、企業の存続に関するさまざまな疑問を前に、答えがわからない状況に置かれるのです。

> 必要がない限り、決断を下すのはやめましょう。
> また、必要な決断であっても、
> できるだけ先延ばしにするようにしましょう。

このような状況で求められる態度は次の 2 つです。

1. 後でも良いのなら、すぐに決断を下さない。
2. できるだけ既存のものを活用する。

これから、具体例を詳しく見ていきましょう。

時間が解決してくれる

初期の CircleCI アプリケーションはモノリスで、お客様のビルドとその関連データを取得し、複数ある LXC コンテナのうちの 1 つにプッシュしていました。スピンアップされるコンテナはどれも、だれか一人でもテスト環境に求めるであろうものがすべて揃った単一のイメージからインスタンス化されていました。今思えば、ひどいアイデアのように聞こえますが、当時はすばらしい方法だったのです。メンテナンスが簡単で、Rails　モノリスを構築していたことが多かったあのころのお客様のニーズにも対応していました。

時が経って顧客ベースが拡大するにつれて、テスト環境の面でのニーズも多様化していきました。基盤となるデータベースのアップグレード版、新たに登場した開発フレームワーク、さらには新しいオペレーティング システムも必要になりました。

CircleCI のもともとのコンテナ管理は、こうしたニーズの変化に簡単に適応できるようには設計されていませんでしたが、きわめてシンプルなものでした。そのため、この問題の解決に乗り出したとき、新しいアプローチの下でどこに手を入れるべきかはわかっており、作業は最小限で済みました。また、この問題に対応できない、かつての稚拙な一般化を再設計する必要もありませんでした。

私たちのもともとのアプローチは、非常にシンプルなものに思えることでしょう。実際そのとおりでした。しかし、会社は 5 年もの間にわたって成長を続けることができました。お客様のなかからその代替となるシステムを試すところが出てくるまでの 5 年間にわたり、このシンプルなシステムでお客様のニーズに対応し続けたのです。この 5 年の間に、Docker が生まれ、HashiCorp の Nomad が作り出されました。これらのツールを組み合わせることで、今日のお客様を支える柔軟でスケーラブルな環境を手に入れるための負担が大幅に軽減されました。

変化した市場に適応するためにシステムを再構築するにあたり、「どうすれば、よりうまく段階的な変化に対応できるか」を考えなければなりませんでした。それまでの 5 年間の経験は、ソリューションを設計するうえで、言葉では表しきれないほどの多大な価値をもたらしました。

私たちが設計上の決断を下す時点でその重要性に気がつくことはほとんどありません。まだ他の選択肢が存在していないからです。ですから、十分に長く待つ方法を見つけましょう。待っていれば、自ずと正しい選択肢が現れます。テクノロジーが牽引力を失えば、この業界は終わりです。仮に、2016 年にコンテナ オーケストレーション エンジンを選択していたら、その時に Kubernetes を採用していた確率は 20% ほどだったでしょう。同じ選択であっても、2018 年の初めであれば、ほぼ 80% の企業が Kubernetes に切り替えていましたから、それほど難しい選択ではありませんでした。

このように、先延ばしにするのが賢明なのです。ただし、深刻な事態を迎えるまで先延ばししてはいけません。

使い手の少ないテクノロジー スタックにご用心

自社の e コマース サイトを構築するにあたって独自のプログラミング言語を作り上げようと考えている人がいたら申し訳ありませんが、これからそうした方の幻想を打ち砕く話をお伝えしたいと思います。**スタートアップの歴史のなかで、テクノロジースタックが他社より優れていたという理由で競合に勝つことができた例はありません**。採用するテクノロジースタックが最新のもので、他のどの会社も使っていないからといって、優位が得られることはないのです。優位をもたらすものがあるとすれば、使い方が万人に知られているもの以外にありません。

Dan McKinley 氏のブログに、「**退屈なテクノロジーを選べ**」(英語) というすばらしい記事があり、そこにこのような記述があります。「ソフトウェアを選ぶ際に何の制約もないとしたら、さまざまな個別の問題に対して局所的に最適であると思われるものを何でも取り入れれば良いでしょう。しかし、実際には制約が存在します。その制約とは「運用」、ときに「認知面のオーバーヘッド」と呼ばれるものです。「この仕事にはどのツールが最善か?」という考え方の問題は、「最善」と「仕事」という言葉を近視眼的に捉えてしまう点にあります。あなたの「仕事」は、何よりも会社の事業に貢献することです。そうであれば、抱えている問題のできるだけ多くについて「悪い者の中でも一番ましな」位置づけにあるツールこそ、「最善」にほかなりません。」

CircleCI の創業者は当時、プログラミング言語に Clojure を採用しました。私は、Clojure が大好きです。開発者が 2 名しかいないチームであれば、優れた言語であると思います。しかし、9 年が経ってチームのエンジニアが 130 名に達し、さらに日々新たなエンジニアを採用するようになると、他の技術領域についてはさっぱりわからないものの Clojure がわかる数少ない人物か、オンボーディングに 3 倍時間がかかる人物のどちらかを選ばなければならないという問題が起こり始めました。3 倍時間がかかるというのは、採用したエンジニア一人ひとりに対して、知識ゼロからプログラミング言語を教えなければならないからです。

ビジネスを立ち上げ、費用対効果の高いかたちで顧客に価値を提供していくことを目指すのであれば、使用者の少ないプログラミング言語を新規採用者に教えたり、輝かしい新技術の運用面のオーバーヘッドに対応したりしなければならない事態は望ましくありません。プロダクトマーケットフィットの達成には無関係な作業だからです。

言い方を変えると、テクノロジースタック絡みの意思決定は技術に関するものであって、ユーザーにはまったく関係ありません。ユーザーにとっては、プロダクトこそ重要なのです。ユーザーに優れたプロダクトを提供することを目指しましょう。

では、マイクロサービスはどうでしょうか?

では、マイクロサービスという観点からプロダクトマーケットフィットを考えてみましょう。マイクロサービス アーキテクチャを採用すると、API やネットワークの垣根を越えてコードを分割することになるので、いっそう強固な分化が進んでしまいます。さらに、システムの一部を交換する必要が生じた場合の変更のコストが大幅に跳ね上がることにもなります。

ですから、マイクロサービスはまだ使わないようにしましょう。

自社のビジネスを見極めようとしている段階で一番不要なものを挙げるとすれば、それはマイクロサービスです。それどころか、マイクロサービスを今すぐに作ろうなどと考えていては、ビジネスを殺しかねません。組織の技術リーダーとしての仕事は、顧客に価値を提供するという側面に対して、テクノロジーを通じて最大限の影響を及ぼせるようにすることです。そして、技術リーダーとして重要な役割を果たすことができるポイントの 1 つが、アーキテクチャに関する (決断を下さないということを含めた) 意思決定です。つまり、この時点では物事をシンプルにすること、プロダクトを迅速にリリースし、フィードバックを集めるという流れを不必要に複雑化する可能性があるものをすべて除外することが望ましいのです。

> プロダクトマーケットフィットさえ実現すれば、
> それまでの技術的な意思決定のミスの
> すべてを挽回できるのです。

長期的に見て正しいテクノロジーを選択できているかどうかを心配する必要はありません。プロダクトマーケットフィットが重要なのは、今この瞬間です。プロダクトマーケットフィットに到達したときには、変更の誘因が変わります。その時点になったら、以前の意思決定のいくつかを再検討してみれば良いのです。

もっと強く背中を押してほしいという方は、プロダクトマーケットフィットさえ実現すれば、それまでの技術的な意思決定のミスのすべてを挽回できるということを覚えておいてください。プロダクトマーケットフィットを達成した後は、想像を超えるスピードで成功がもたらされます。逆に、技術面で優れた意思決定を下していたとしても、プロダクトマーケットフィットが欠けていてはまったくの無意味です。

勝利をもたらすのは常に、プラグマティックな考え方です。退屈に感じられてもシンプルなものを、数多の試練に耐えるものを作りましょう。

不確定性を力にする

自社のビジネスにとって重要な変化は、いつでも起こり得ます。しかし、どこで何が変わるかを事前に知るすべはありません。

世界中のソフトウェア　アーキテクトたちの知見や助言が詰まったエッセイ集『**ソフトウェア アーキテクトが知るべき 97 のこと**』の中で、ケブリン・ヘニー氏は、**不確定性について考えるうえでの重要なアプローチ**を紹介しています。

> 「2つの選択肢があるということは、設計の中に不確定性が潜んでいると感じなければならないというシグナルなのです。「不確定性が潜んでいるぞ」という感覚を磨くことで、詳細に踏み込むのを待つ箇所、あるいは設計の判断が大きく影響しすぎないように分割や抽象化をする箇所がどこかを見抜くことができます。最初に思いついたことをコードに埋め込んでしまうと、その判断に縛り付けられてしまいます。思いつきの判断に引きずられて、ソフトウェアの柔軟性が失われてしまうのです」

このような考え方は、明示的に意思決定を下す場面には最適ですが、何らかの選択をしていることを自覚していない場合はどうでしょうか。

その時点では選択という行為が認識されていなくても、たいていは後で明らかになるものです。このような場合には、「念のために抽象化しようとして一般化しすぎないこと」が大切です。抽象化は、実装のしかたが複数あることが確認できた箇所（つまり、変更のコストが大きな箇所）のうち、変更が発生する可能性が高いと思われる箇所のみ実施するようにしましょう。あらゆる部分を完璧に抽象化しておくことは、避けようのない変化に対する対策としては確かに良いものの、コストが非現実的なレベルにまで跳ね上がります。そのようなことをする代わりに、物事をできるだけシンプルにしておいて、時間が経ってから変更が必要になったときにも理解できるようにしておきましょう。単に懸念を種類に応じて分けておくだけでも、(変更が必要になった時点で作業こそ発生するものの) 変更作業にかかるコストを抑えることができます。

重要なのは、判断を誤った場合のコストを最小限に抑えるようにしたうえで、変化を注視していくことです。

そのためには、何かを作り上げる手間を省いてくれるものなら何でも活用するという態度が基本となります。既製品が使えない場合には、できる限りの抽象化に取り組みつつ、引き続き既存のツールやフレームワークが使えないかを検討しましょう。たとえば、自社のビジネスが定まっていない段階で、コードを 10 行ほど書くだけで AWS Lambda が使えるのなら、そうしましょう。仮に自分で 1 から作った方が安かったとしてもです。そうすることによって、問題が発生している部分だけを改修し、その結果を静観していれば済むようになります。やろうとしていることが市場の製品やサービスでは間に合わず、独自の何かを作成しなければならない局面がやってきたときには、きっと気付くはずです。そのときは、その問題を解決することに集中すれば良いでしょう。技術者としては、他人が作ったツールやライブラリをつなげているだけでは仕事をしているとは言えないという感覚になることがよくあると思います。しかし、あなたの仕事は、ビジネスに価値をもたらす意思決定を下すことにほかなりません。そのためには、刺激的なツールを新たに作り出すことに反対することだってあり得ます。

1 つ覚えておきましょう。コード自体に価値はありません。一度書いてしまえば、すぐさま負債に変わるからです。この段階においては、完璧な設計など存在しません。コードは少ないほど好ましいのです。捨てても惜しくないよう、できるだけ少量かつシンプルにしておきましょう。

消してもかまわないコードを

アーキテクチャは進歩していくにつれて、その混沌の度を増していきます。「このサイトをどうやってバックアップしたら良いものか」と途方に暮れる類のソフトウェア設計をしてしまう者の 1 人として、私もあまり多くのことを細かく検討するようなことはしていません。そのようなことをしても、「この問題を解消するために、昨日の時点で何ができたか?」と聞くようなものだからです。実際、ソフトウェアの発展の流れを少し後から振り返ってみると、運の要素が非常に大きく感じられるものです。

しかし、事前にいくつかの対策を講じることによって、規模を拡大すべきタイミングが来たときに望ましい状態を作り出しておくことは可能です。もっとも、そのような状態の実現を妨げるものも山ほどあるので注意する必要がありますが。このトピックに関して私が気に入っているのが、「Programming is Terrible」というブログの「**拡張しやすいコードではなく、削除しやすいコードを**」という記事 (英語) です。その一部を以下に引用します。

> 「[何行生み出したかではなく] "何行消費したか" という観点からコードを捉えれば、コードを数行削除する行為が、保守コストを下げる行為として浮かび上がってくるはずです。再利用可能なソフトウェアではなく、捨てられるソフトウェアを作ることを心がけるべきなのです。」(tef 氏)

この記事が指摘しているのは、将来捨ててしまって、代わりのものに取り替えてしまえるほどシンプルなコードを心がけていれば、優れたものを作り出す際の労力を大幅に節約できるということです。現代において上手に規模を拡大できる体制を構築するうえでは、このような実用主義的な考え方が非常に有用であると考えられます。

エンジニアリング分野では、モノリスとマイクロサービスのどちらにするか、コードが非常に巧妙で抽象化とカプセル化もできているか、それともいわゆるスパゲッティ コードになっているかなど、二者択一的な見方がされることが少なくありません。ところが、現実的にはその中間に落ち着くものです。そのことを覚悟し、割り切った考え方を心がけることが、いつの時点でも重要事項に集中し、重要性の大きな問題を解決できるチームの実現に役立ちます。

後になれば高くつく
早期に決めておくべきこととは

経験的に私が検討した方が良いと思っていることの１つに、決断当時にはそれほどお金がかからないものの、後になって変更しようとすると高くつく類の意思決定をどうするかという問題があります。何かに投資をしようと考えている（あるいは、今投資をしている）場合には、前もって変更のコストを考慮しておくことをお勧めします。

ここまでの内容を読んだ方なら、マイクロサービスを採用することはないはずです (既に書いたとおり、ビジネスの構築に集中せず、Docker や Kubernetes の細かな話に時間を割いてしまうのは、良くない意思決定です)。これに代わって私が検討をお勧めするのは "分離" です。ソフトウェアの分野では、カプセル化や **3 回ルール**（リンク先英語）ということがよく言われます。完璧な抽象化がどのようなものであるかを考えたらきりがありません。物事をシンプルかつ実用的にする過程では、ビジネスの特定の領域に紐付けられたものを他から分離したうえで、そのすべてを 1 つにまとめておくべきです。後になって十分理解が深まり、カプセル化するか抽象化するかを決断しなければならないタイミングがやってきた時点で、少なくとも "全部が入っている" 状態にはなっているからです。

そのうえで、境界を定めるわけです。

この境界は、シンプルなコード　ベースの枠内で定めるようにしましょう。現時点で開発しているものがモノリスであっても問題ありません。むしろ、その方が良いとすら言えます。いわれのない非難を受けているモノリスですが、現実にはすべてのものが１つにまとまっており、探しやすく、必要なときに取り出しやすいというメリットがあります。

このことを、コストという観点から考えてみましょう。定めた境界線を理解し、それを守りつつ、自社の発展と共に整理、調整していくというアプローチは、実は安上がりです。関数の名前やコード　ベースの中での位置を変えるコストの方が、サービスの垣根を越えて何かを移動するコストよりもずっと安いからです。ヘキサゴナル アーキテクチャ (ポート & アダプター アーキテクチャ) のようなものを考えてみましょう。1 つのコード ベースの中にはっきりとした境界線で明確に分離されたコードを記述するためのパターンが確立されています。そのため、ビジネスが拡大し、一部を取り出す必要が出てきた場合でも、その部分がどこにあるかが正確にわかるのです。

これとは対照的に、マイクロサービス アーキテクチャの場合、移行に際してある一部分を取り出そうと思ったら、その前に大混乱が起きないような別のモデルに切り替えなければならないことが多々あります。マイクロサービスのコストが高くなるのはこのときです。

ドメイン駆動設計

いわゆるドメイン駆動設計も、早期に始めれば安上がりで済むものの、後からやろうとするとコストがかさむプロセスです。今日生まれているビジネスは、多くが他のプラットフォームとの統合に対応しています。他のプラットフォームとの統合においては、そこからデータを取り込む作業が付きものです。データを取り込む際には、自社システムとのバウンダリー（境界）で、自社システムにとって都合の良い形式に変換することになります。

ここで私たちの事例をご紹介しましょう。CircleCI では、2011 年から GitHub を使用してきました。私たちのシステムの中には、2011 年に GitHub から取り込んだときの状態がほぼそのまま保たれているデータが残っています。そして、そのデータをさまざまな場所で扱うためのコードが大量に存在します。データをバウンダリーで変換するようにしておけば、物事が格段に円滑に進むようになるのです。

自社に何があるかを把握し、それをきちんとコントロールできるようにしておくことは、早いうちであればごく簡単ですが、後からやろうとすると非常に高くつくので注意しましょう。

早期から CI/CD に投資せよ

"Hello World" を出力するごく少量のコードを対象として CI/CD パイプラインを作ったことがある方は、その作業を非常に簡単なものに感じられることでしょう。しかし、コード ベースがきわめて大きくなり、何か非常に大きなものを取り出そうとしている場合にインフラストラクチャ全体を自動化するとなると、格段に難しくなるのが現実です。自社ソフトウェアのための CI/CD パイプラインを導入することは、早い段階で下せる意思決定のなかでもその影響が最も大きなものではないでしょうか。ビジネスが軌道に乗り、規模の拡大を始めたときに、あなた自身の業務だけでなくチームの業務が大幅に楽になるからです。なかでも、高速でアプリケーション開発と並行しながらでも簡単に導入できる CircleCI は、スタートアップ企業にとって最適な選択肢でしょう。運用は自動で、手作業によるオーバーヘッドが発生しないうえ、開発者の貴重な時間を管理に充てる必要もないので、チームに CI/CD のエキスパートが加わるようなものです。さらに、チームの拡大やコードベースの変化にも対応できるので、規模拡大やアーキテクチャの変更をシームレスに進めていくうえでも役立ちます。

変化にすばやく対応できるかどうかは、成果を出すことができるかと密接にかかわっています。チームがその機動性を高め、競合に先んじて成果を出すことができるよう、CircleCI をご活用ください。

時間がかかる、コストが高い、本質から逸れる... なかには常に高くつく決断も

自分で作らず、お金で買え

自社で使うメール サーバーを自ら構築することなどありうるでしょうか? 私には経験がありますが、これは私が電子メール サービスを提供する会社で働いていたからにすぎません。同じ理由で、CI/CD も自分で作ろうとしないことです。そちらは CircleCI が (うまく) やりますので、自らやる必要はまったくありません。

私たちはみな、巨人の肩に乗っているようなものです。そして、この状況はこれからも続きます。テクノロジーの歴史を振り返ってみると、1 サイクル前の時点で新たに生まれたテクノロジーを組み合わせることで、おもしろいものを作り上げるということが常に行われてきました。最初は、チップができました。次にソフトウェアができ、コンピューターが増えました。すると、今度はそこでさまざまなものが作られるようになりました。たとえば、地図アプリです。その地図アプリを基に、だれかがナビゲーション アプリを作りました。その後ライド シェア アプリが加わったわけですが、その開発者がナビゲーション アプリの機能を 1 から作る必要はありませんでした。世の中に既に存在していたからです。既に発明されているものを自社で 1 から作ろうとしても、競合他社まで同じことをしてくれるはずはありません。競合はその間も、あなたが獲得を目指すビジネスに集中しています。時間を浪費するだけで、待っているのは敗北という結果でしょう。

競合との差別化につながるものや、ユーザーに新たな独自の価値をもたらすものでないのなら、自分で作るべきではないのです (会社の事業として CI/CD を展開していなければ、**私のお勧めはこちら**です。ぜひご利用ください)。

技術面の意思決定について Zuber がお勧めする 14 のルール

1. スタートアップ企業のリーダーとして何より大事なのは、プロダクトマーケットフィットの達成。何かを選択する局面では、できるだけ早くユーザーにプロダクトを届けるという点に寄与するかどうかを基準にすること。
2. 技術に関する意思決定は、ビジネスとしての意思決定のためにある。エンジニアの最高の仕事とは、ビジネスにとって最善となる仕事であると心得るべし。
3. 「ステルス モード」をやめよ。
4. 事業計画は立てるな、MVP (Minimum Viable Product: 顧客に価値を提供できる最小限の製品) を作れ。
5. 決断は、できるだけ先延ばしに。
6. 時間が解決してくれると考えよう。
7. 使い手の少ないテクノロジー スタックにご用心。
8. プロダクトマーケットフィットさえ実現すれば、それまでの技術的な意思決定のミスのすべてを挽回できる。
9. 不確定性を力にせよ。
10. 消してもかまわないコードを書く。
11. 早期からドメイン駆動設計に取り組む。
12. 早期から CI/CD に注力する。
13. 自分で作らず、お金で買え。
14. 複雑化を防げ。

結び:
規模拡大は複雑化との戦い

自社のテクノロジーがどうあるべきかについて、強固な信念を抱いている方もいるかもしれません。自社の優先事項は自分がよくわかっているというわけです。しかし、チームが拡大するなかで、その優先事項をしっかりと明確にし、メンバーに伝えていくことを (必要だと思う回数の 50 倍は) しなければ、新たなメンバーを採用するたびに、自身の考えに基づいて行動する人が増えていきます。私が他社の CTO からよく聞く悩みの 1 つに、起業後に経験豊富なエンジニアを採用したときの問題があります。突然、オフィスで最も経験のある人物が自分でなくなってしまったような気がして、自分がエンジニアたちの行動に影響を及ぼす資格があるかどうかがわからなくなってしまうというのです。そうなるとやがて、会議の場にリーダーが何人も集まり、各種の事項に方向性がバラバラの意思決定を下す事態が起こり始めます。そして、後になって非常に大きな代償を支払うことになるのです。

物事をあらかじめはっきりさせておくこと。創業者であれ技術リーダーであれ、それがあなたの仕事です。あなたなら、自社のビジネスの背景を知っています。顧客に届けようとしている価値もわかっています。それを伝えることに集中しましょう。ポイントは、メンバーが指針として利用したり、日々の意思決定が正しくできているかどうかを確認したりできるようなかたちで伝えることです。

規模の拡大に伴ってテクノロジーが複雑化していかないように注意しましょう。そのためには、あらゆる場面でプロダクトマーケットフィットを追求し、必要に応じてすばやく、何度でも、多額の費用をかけずに方針を転換できるような意思決定を心がける必要があります。技術リーダーとして影響力の大きな決断に専念し続けることができれば、早期に短期間で下した戦略的決定に起因する問題を後から是正するための時間とお金が得られます。戦略を立て、重要事項に徹底的に集中しましょう。あとはほんの少しの運に恵まれれば、きっと成功を収めることができるはずです。