O'REILLY®

Google Cloud 監修

SRE エンタープライズロードマップ

SREを導入し継続する方法

James Brookbank 著
Steve McGhee 著
山口 能迪 訳

テクニカル レポート

Google Cloud

企業でSREを導入するには?

導入から実践までのノウハウが詰まっています。
これからSREを導入する企業の必携書!

SREに関する情報はsre.googleまで

SRE エンタープライズロードマップ

SRE を導入し継続する方法

James Brookbank、Steve McGhee 著
山口 能迪 訳

O'REILLY®

Enterprise Roadmap to SRE
by James Brookbank and Steve McGhee

Authorized translation of the English edition of Enterprise Roadmap to SRE, ISBN 9781098117733 © 2022 O'Reilly Media, Inc.

Acquisitions Editor: John Devins
Development Editor: Virginia Wilson
Production Editor: Christopher Faucher
Copyeditor: Tom Sullivan
Proofreader: Stephanie English
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

目次

序文

Googleが以前出版した2冊の書籍、"Site Reliability Engineering"（Betsy Beyer、Chris Jones、Niall Richard Murphy、Jennifer Petoff 編）と"The Site Reliability Workbook"（Betsy Beyer、Niall Richard Murphy、David K. Rensin、Kent Kawahara、Stephen Thorne 編）[†1]は、サービスライフサイクル全体に対する取り組みによって組織がソフトウェアシステムの構築、展開、監視、保守に成功できる方法と理由を示しています。

本レポートは、これらの書籍を基礎とし、大規模かつ複雑な組織（以下、大企業と呼ぶ）においてサイトリライアビリティエンジニアリング（SRE）を導入する際の課題について、もう少し深く掘り下げることを目的としています。過去数年にわたりSREが人気を博しているにもかかわらず、SREに対する熱意と採用のレベルとの間にギャップがあるというフィードバックを多数の大企業から受けています。

大企業にとって信頼性はますます大きな差別化要因となっているため、このギャップは解消すべき重要なポイントだと考えています。クラウドの導入とCOVID-19の大流行によって引き起こされた技術変化のペースと規模は、しばしばこの複雑性の増大に対処するためのさまざまな技術を必要とします。

これらのトピックは、本番システムの信頼性に携わっている（または依存している）方で、SREの採用についてもっと知る必要がある方にとって、より興味深いものとなるでしょう。これには、経営者やリーダーだけでなく、クラウドアーキテクト、サイトリライアビリティエンジニア（SRE）、プラットフォーム開発者などの個人的な貢献者も含

†1　翻訳注: それぞれ『**SRE サイトリライアビリティエンジニアリング**』（2017年、オライリー・ジャパン、ISBN978-4-87311-791-1）と『**サイトリライアビリティサークブック**』（2020年、オライリー・ジャパン、ISBN978-4-87311-913-7）として日本語訳版が刊行されています。

まれます。職種に関係なく、技術システムを設計、実装、または保守している方なら、きっとお役に立てることがあるはずです。

第1章
エンタープライズSRE ことはじめ

既存の大企業にSREを導入することは困難なことのように思われますが、ここではそのような場合に役立つ提案を集めました。まずは既存の環境を評価し、期待値を設定し、SREを評価しながら正しい方向に進むようにし、それが組織内でどのように機能するかを確認します。

進化は革命に優る

大企業の特徴の1つは、これまでのIT/経営情報システム（MIS）の方法論や原則の歴史が常に存在することです。ここでは、いくつかの一般的な方法論について詳しく説明します。現状に関わらず、SREの導入で最も成功するのは、既存のフレームワークと真っ向から戦うのではなく、進化させ、補完することを選択した場合です。また、SREは、あらゆる技術採用プロセスにおいて、歴史が重要であるという事実と無縁ではありません（Wikipediaの**経路依存性**のページを参照してください）。つまり、企業のような複雑なシステムでは、同じ変更を異なる場所から適用すると、収束するのではなく、乖離した結果になることを意味します。まずは、SREの実践が一般的な異なるフレームワークとともに成功する例をいくつか紹介します。

SREの実践はITILフレームワークと共存できる

ITIL（Information Technology Infrastructure Library）は、ITサービスマネジメント（ITSM）などのIT活動に関する詳細なプラクティスの集合体です。すべての企業が

ITILを使用しているわけではありませんが、組織内でITILをある程度採用している場合、SREとITILの実践にはかなりの重複があることを心してください。また、ITILはあくまでフレームワークであるため、各企業の実装はライブラリとは大きく異なる可能性があります。

> **キーポイント:**ITILはITサービスの構築と運用方法について書かれた5冊の本で、数千ページにも及びますが、これらのトピックの多くは信頼性に関係しないため、意図的にSREではカバーしていません。ITILはフレームワークであり、SREはプラクティスの集合体なので、両者に互換性はありますが、例えば「保証（warranty）」「有用性（utility）」など、用語の翻訳には困難が予想されます。また、SREは変更管理やサービスオーナーシップなどの分野で強い考え方を持っているため、結果が一致していてもやり方を調整する必要があることを肝に銘じてください。

SREに共通するアンチパターンがいくつかあり、これを調和させるのは困難です。変更諮問委員会（Change Advisory Board、CAB）は、変更管理のための一般的なパターンです。継続的なデリバリーに対するSREのアプローチは、この組織を合理化し、戦略的にすることを意味します。詳細は、GoogleのDevOps Research & Assessment（DORA）の記事で、**変更承認の合理化**について説明されています。同様に、ネットワーク オペレーション センター（NOC）についても、イベント駆動型モデルから、自動化と有効化を中心としたより積極的なアプローチに移行する必要があります。どちらの場合も、すぐに置き換えようとするのではなく、モデルを進化させることに重点を置いてください。

DevOps、アジャイル、リーン

DevOpsには多数の**定義**があります。ここでは簡単のために、**アジャイル**（Scaled Agile Framework [SAFe]、Disciplined Agile Delivery [DAD]、Large Scrum [LeSS]）や**リーン**（Six Sigma、カンバン）といった他の方法論の関連部分を含むと仮定することにします。Googleの**DORAの調査**によると、SREとDevOpsは補完関係にあり、組織内でDevOpsをある程度導入していれば、通常は有益です。ITILと同様に、SREとDevOpsのプラクティスは重複する部分があり、特定の実装では**The DevOps Hand-**

Book [†1]から大きな差異が生じる可能性があることを想定してください。

具体的なSREの実践については後で詳しく説明しますが、一般的にDevOpsによく関連する多くの機能（バージョン管理、ピアレビューなど）も、一般的にはSRE導入の前提条件と考えられています。これらの能力をDevOpsやSREの取り組みを通じて構築するかどうかはあなた次第ですが、採用を確実に成功させるためにはやはり必要なことです。

> **キーポイント**:DevOpsとSREの取り組みを両立させるためには、現実的であることが必要です。大規模な変化を成功裏に収めるには、反復的かつ段階的に行うことが必要です。完璧な「全体像」を得るために不必要な時間と労力を費やすのではなく、個々の活動を分解し、人々が実践できるようにすることに焦点を当てることが重要です。

DevOpsとSREは補完関係にありますが、両者を調和させることが困難な領域もあります。例えば、DevとOpsの各部署を機能横断的なDevOpsチームに置き換えることを決定している場合があります。このような状況において、SREのような専門的な職務を再び導入することは、慎重に検討する必要があります。

今いる場所から始める

現在使用している方法論やフレームワークが何であれ、現在の状況を理解し、自分自身に正直になることが重要です。**SRE本にあるように**、「希望は戦略ではない!」のです。もし、現在の環境に不足や改善の余地がないと思うのであれば、なぜSREを導入しようとしているのか、自問自答する必要があります。同様に、現在使用しているテクノロジーや従業員の中には、SREのビジョンに沿わないものもあるかもしれません。変更を加える前に、時間をかけてこれを理解する必要があります。

期待することと目指す姿を概説する

次に、どのような成果を期待するかを理解することが重要です。SRE には技術的、

†1　翻訳注: 翻訳版があります。『The DevOps ハンドブック 理論・原則・実践のすべて』(2017年、日経BP、ISBN978-4-82228-548-7)

文化的な要素が多く含まれますが、それらはすべて信頼性目標を達成するという共通の目標を持っています。それらの要素が既存のフレームワークとどのように相互作用するかを定義するために、有意義な時間と労力を費やすことになるでしょう。単に「信頼性を向上させる」というだけではうまくいきません。同様に、信頼性とは関係のない成果（コスト、速度など）を期待している場合は、SREの手法を全体的なビジョンに適合させるために、多少の余分な作業を覚悟しなければなりません。

SREは「人」から始まる

人やプラクティスは、時間の経過とともに変化するプロセスやテクノロジーを取り入れ、適用させることが可能です。トレーニングや人材採用から始めれば、技術やプロセスはいつでも追加したり削除したりすることができます。能力開発は段階的なプロセスであるため、人材採用で成功に導こうとしないことです。人材採用は、トレーニングに取って代わるものではなく、トレーニングを補強するものと考えてください。**SREを成功させるには、生成的な文化 が必要であり、これを確保することが重要であることを覚えておいてください。**

自分らしさを大切にする

特定の組織で SRE を導入するベストプラクティスは 1 つではありません。正しい方法は、あなたが成功した方法だけです。私たちは、複数の組織で何がうまくいき、何がうまくいかなかったかについて、かなり研究の恩恵を受けていますが、そこで見つかっていない新たな間違いをあなたが経験することは避けられません。このような組織の経験を真の学習ツールとして利用し、組織に改善の好循環を構築してください。

第2章
なぜ信頼性のためにSREというアプローチをとるのか？

信頼性というのは、新しい概念ではありません。大企業は常に、信頼性、品質、あるいは稼働率を、改善すべき性質として位置づけてきました。では、SREは何が違うのでしょうか。なぜ今日このようなことが起きているのか、何が違うのか、そしてなぜそれが企業にとって重要なのでしょうか。

信頼性を製品の主要な差別化要因にする

なぜ大企業はSREチームの結成や信頼性の追求を望むのでしょうか。また、どのような成果を期待しているのでしょうか。技術やプロセスには流行り廃りがありますが、、それが定着するためには、実質的なビジネス価値が必要なのです。信頼性とセキュリティについて考えてみましょう。どちらも当初は明確な製品差別化要因にはなりませんが、想定される要件です。問題が発生し、製品への期待や信頼が高まったときに初めて、この2つの要件が顕著になります。例えば、何年も前は、セキュリティの悪用やハッキングは比較的まれであったため、セキュリティ要件は存在していましたが、消費者や企業向けの製品のマーケティング資料には記載されていませんでした。しかし現在では、脆弱性はより一般的なものとなっており、セキュリティは製品の差別化要因のひとつと考えられています。

信頼性（より一般的には可用性または稼働時間）は、主にサービスレベル合意書（SLA）および同様の契約や期待値設定の細則の文脈で語られる傾向があります。しかし、実際に信頼性を意識するのは、顧客満足度（CSAT）スコアやDowndetectorのような第三者サイト、そして私たちの生活やビジネスの多くがインターネット上に移行して

いる全体的な傾向を通じてです。COVID-19の流行時には、多くのSaaS（Software as a Service）製品がビジネスを大きく拡大させ、信頼性への期待値を飛躍的に高める必要がありました。

一般的に理解されている信頼性の代名詞である可用性とは別に、耐久性、データレジデンシー[†1]、負荷時の速度や性能、一貫性、結果の質といった性質も、消費者やインターネットサービスの顧客が暗黙のうちに期待している、似たような信頼性の性質として考えることができます。

信頼性は、実は製品に強く求められる機能であることを理解すれば、製品の最も望まれる機能であるとさえ言い切れるかもしれません。なぜなら、その製品が使えなければ、その機能を何一つ生かすことができないからです。性能や品質で不満が出れば、満足度は上がりません。ピーク時、重要な瞬間に使えないのであれば、利用する価値がまったくないかもしれません。

Google検索は「常に稼働している」ことで有名であり、その存在はありとあらゆる場所で感じられます。Google検索の可用性は、スピード、品質、使いやすさ、ユーザー体験と並んで、競合他社と比較する際の重要な差別化要因となっています。これは偶然ではなく、Googleが10年以上にわたって行ってきた意図的な選択と投資なのです。

いつ信頼性を重視すべきか

スタートアップが信頼性への投資を検討するとき、時期尚早と見るかもしれません。特に、Googleのような大組織がとっている包括的なな対策を考えると、なおさらです。スタートアップはまず、耐久性や回復力のあるサービスではなく、実用最小限の製品のみを構築することを目的としているため、これは完全に妥当なことです。しかし、実用可能性を確立したら、セキュリティやその他の「水平」な取り組み（国際化、アクセシビリティなど）と並行して、できるだけ早く信頼性を製品ロードマップに統合してください。

高コストな、信頼性技術の自前での開発は、スタートアップやアーリーステージのビジネスには時期尚早でしょう。しかし、セキュリティと同様、信頼性の分野でも、オープンソース ソフトウェアやサードパーティ製のサービスやツールなど、多くの製品がコモディティ化しつつあります。セキュリティと同様、信頼性の分野でも、オープンソー

†1 翻訳注: データの格納場所のこと。参照https://www.atlassian.com/ja/software/data-residency

ス ソフトウェアやサードパーティが提供するサービスやツールによって、コモディティ化が進んでいます。大規模で複雑な既存システムへの統合や、信頼性の問題に対する対応が遅れることがないよう、早い段階からこれらを活用することが必要です。信頼性と準備のためには、先を見越した検討が重要です。また、Google のような企業は多くの信頼性システムを自社で構築していますが、これは必ずしも費用対効果の高い方法ではないことも知っておく必要があります。外部のサービスやツールを活用することは、確立されたベストプラクティスです。特にセキュリティ（「暗号は自分で作らない」）や信頼性の分野では、無数のエッジケースや副作用が考えられるため、構築よりも購入が推奨されます。信頼性ベンダーの分野は、現時点ではセキュリティほど成熟しておらず規模も大きくないですが、成長してきているので、発展途上の企業にとっては、大きな違いをもたらす可能性があります。

マッキンゼーの「成長の3つの地平線（ホライズン）」モデル（図2-1参照）は、投資計画を立てる上で参考になります。これは、自社の将来について考えるための3つの方法を説明しています。

- ホライズン1は、現在すでに重要視されている分野です。
- ホライズン2は、あなたが期待する新たな成長分野です。
- ホライズン3は、現在研究開発段階にある、長期的な潜在成長分野です。

各ホライズンに対して様々な投資レベルを検討することで、無限にあると思われる信頼性空間の作業を分割できるのです。

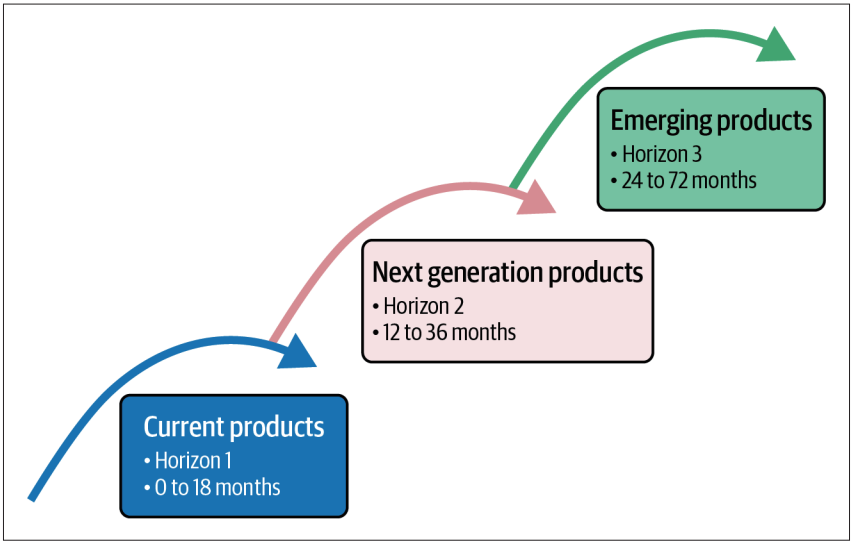

図2-1: 成長の3つの地平線 出典 "The Three Horizons of Growth" Mehrdad Baghai、Stephen Coley、David White著 (Basic Books)。

最後に、ホライズン3の信頼性向上のための取り組みには、企業が提供する事業を拡大する方法が含まれます。破壊的な機会に対応するため、または競合の脅威に対抗するために、新しい能力と新しいビジネスモデルを実現する必要があります。ホライズン3に投資する企業は、プラットフォームとアーキテクチャを単一のビジネスモデルに縛られることなく、制御と品質基準を維持しながら、さまざまな形状のシステムを生成、変形させることができるようにする必要があります。信頼性は高くても、硬直したシステムでは十分ではありません。中央集権的な承認委員会やトップダウンのアーキテクチャーの義務化といった取り組みは、ホライズン3で必要とされるイノベーションを阻害するものです。

そのため、ホライズン1にSREを適用することで、ビジネスに即効性を持たせることができます。ホライズン2にSREを中核的な基盤として配置することで、その潜在的な成功は確実なものとなります。しかし、ホライズン3はまだ実現可能性が確立されていないため、SREを始めるには最適な場所とは言えません。

なぜ今SREなのか

なぜSREは1970年代に発明され、普及しなかったのでしょうか。なぜ2010年にはなかったのでしょうか。インターネットを利用したサービスの複雑さは、近年明らかに増しており、中でもクラウドコンピューティングの台頭は顕著です。クラウドは、コンピュータサイエンスの深い学問分野である分散システムの商業的な後継者だと考えています。このコンピュータサイエンスの一分野が、個人消費者（例: Google、Facebook、Apple）や企業（例: Salesforce）に関係するようになったのはここ十数年のことで、クラウドプロバイダはもちろん、サービスプロバイダ（Akamai、Stripe）がその原理を使ってスケーラブルなインターネットシステムを効果的かつ広範囲に提供できるようになったのもこの頃からです。

「**ウェアハウス スケール コンピューティング**」の導入は、企業がサービスを構築、提供、運用、拡張する方法に変化をもたらしています。このモデルは、空間を建てたり借りたりして、コンピュータシステムを購入する資本支出（CapEx）モデルから、計算処理サービスの一部をオンデマンドで借りる運用支出（OpEx）モデルへと、企業のコスト処理方法を明らかに変化させています。しかし、それ以上に、システム設計やアーキテクチャ、変化する障害モードへの対応にも影響があります。

従来のインフラは、ビルやピラミッドのように、大きな強固な土台を下から上に積み上げていくモデルでした。基盤が故障すると、その上にあるすべてのものに災難が降りかかります。これを私たちはコンポーネントベースの信頼性モデル、あるいはユニオンモデルと呼び、システム内のすべてのコンポーネントが使用可能であることがシステムの動作の前提となっています。

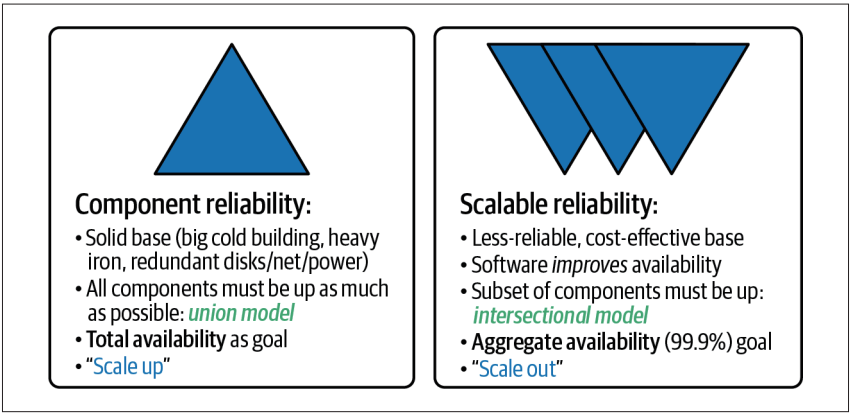

図2-2:　信頼性のピラミッド

クラウドコンピューティングで使用される新しいモデルは、確率的信頼性モデルまたは交差モデルであり、故障を想定したアーキテクチャの選択により、システムが動作するためには全体の一部分のみが利用可能であればよいのです。

このコンセプトはまったく目新しいものではなく、理解するのも難しいものではありませんが、クラウドサービスを採用する企業にとっては、特に「リフト＆シフト」のような、新旧モデル間の公平性を売りにした提案の場合、明白ではないことも確かです。旧モデルを新プラットフォームで稼働させることは十分に可能ですが、多くのことを考慮しなければならず、準備不足の人はしばしば困惑してしまいます。例えば、従来のIT部門は仮想マシン（VM）の稼働率に誇りを持つかもしれませんが、クラウドのVMはより揮発的で、意図的に、しばしば非常に頻繁に作成・破棄されることが想定されます。

大企業は、Google、Facebook、Appleのような現代企業に成功例を見いだし、2つの大きなメリットを見ています。（1）大規模なイノベーションと、（2）大規模な信頼性の高いシステムです。これらの企業は、新しいシステムを構築できるだけでなく、それらを常に利用可能で、高速かつ安定的に維持することができます。この組み合わせは、市場の需要に迅速に対応し、市場全体に広範なソリューションを提供することができるため、ビジネスにとって非常に強力です。

Google の後光を越えて

もちろん、Google は常に巨大だったわけではありません。実際、初期の Google は、より伝統的な方法でサーバー群（フリート）を管理していました。Google が異彩を放ったのは、そのフリートを垂直スケーリングから水平スケーリングに早くから移行したことです。つまり、より大きく、より強力なコンピュータを購入するモデルから、より多くの、より安価なコンピュータを購入するモデルへと移行したのです。この変化は、初期のコロケーション施設の通路を歩けば一目瞭然です。第一次ドットコムブームの頃、多くのテナントが見栄えのする高価なハードウェアをケージに入れていたのに対し、Google はその代わりに膨大な量のコモディティハードウェアを置き、いつでも単一のマシンが故障することを想定し、その故障をソフトウェアで対処していたのです。

この移行における重要な要因は、この水平シフトに伴う運用コストを意図的にスケールさせないという選択をしたことです。つまり、水平方向に拡張する場合、管理下にあるマシンの台数に比例してスケールする運用チームの人員を配置することは、経済的に意味がなかったのです。

この技術的、財政的な選択が、組織的な選択を促したのであり、間違いなくこれが SRE の起源です。Google は、ごく初期のウェブスケール企業であったため、他の多くの企業よりも先にこの選択をしたに過ぎません。当時のほとんどの企業は、Google がインターネット全体の検索結果を保存、インデックス、提供するといった規模の処理量やストレージ量を考慮する必要がなかったのです。

現在、多くの企業が、当時 Google が直面したのと同じようなスケーリングの課題に直面していると考えています。ただし、現在これらの企業は、自社のデータセンターを大量のコモディティ ハードウェアで満たす必要がなく、パブリック クラウドという利点を享受しています。Google が SRE 機能を進化させることでこのアプローチの変化を克服できたという事実は、SRE がこれらの企業が同じハードルを克服するのに役立つことを意味するはずだと私たちは考えています。

従来の IT 運用の人員配置と比較して、SRE の人員配置について Google で学んだ重要なポイントは、サブリニアスケーリングということです。これは、システムを運用するチームの規模は、システムの基本単位と同じ割合で大きくなってはいけないという意味です。システムの使用量が 2 倍になったとしても、2 倍の運用担当者は必要ないはずです。Google は、マシンの台数ではなく、クラスタ、サービス、プラットフォームなど、

より高次の指標でスケーリングすることを選択しました。より高い抽象性に焦点を当てることで、チームはより多くのことを、より少ない負担で行うことができます。これらの抽象化は、以前の抽象化をモデルとしたシステムを操作していた人たちによって構築、拡張される傾向があります。

複雑化するとSREの必要性が高まりますが、サービスの新規立ち上げよりも人員配置をゆっくり増やすべきです。これは**サブリニアスケーリング**と呼ばれる概念です。これは実は、チームのトイル（toil）の削減[†2]という原則に直接関係しています。システムを健全に保つために反復的な作業を行うチームは、システムが成長し、増殖すると、本質的にこれらの作業をより多く行わなければならなくなります。SREの管理者は、これを積極的に防止し、追跡する必要があります。これは、もしチームがあまりに多くのトイルを野放しにすると、滑りやすい坂道となり、下降スパイラルに陥って、持続不可能なチームとシステム障害が増加する結果となります。

なぜ伝統的なオペレーションを増やさないのか

Googleの垂直統合から水平統合への移行と、それに伴うSREの開発によってもたらされた変化の経験を活用することで、組織は規模の拡大のメリットを早期に享受し、かつコストを削減することができます。複雑化した業務に対応するチームの規模拡大のコストを削減するために、より安価な労働力を探すという方法もあります（例：「スタッフ増強」または「アウトソーシング最適化」）。これは、規模や複雑さに対処する際に、あまりにも一般的なアプローチです。その結果、システムの成長が阻害され、障害が発生し、時間とともにコストが増加するという予期せぬ事態に陥ることがよくあります。このような予期せぬコストは、システムの停止やブランドへのダメージだけでなく、実行速度や規模の縮小、新製品の市場投入までの時間、さらには競合他社に追い越されることによる逸失利益など、さまざまな形で発生する可能性があります。組織は、運用と信頼性の投資にかかるコストを最適化する方法を選択する前に、全体像を考慮する必要があります。

そのため、SREチームに適切な人材を配置することが重要です。博士号取得者ばかりを雇う必要はありませんが、手を抜くのもよくありません。運用担当者の単価だけに

†2　翻訳注：トイルの削減については"Site Reliability Engineering"（2016年、O'Reilly Media、ISBN978-1491929124）の第5章を参照してください。（日本語版でも同様です）

注目するのではなく、システム全体のコストに目を向けるようにしましょう。例えば、食品工場では、桃を缶詰にするために何百万ドルもする機械を使っています。それなら、「未熟練労働者を雇えばいいじゃないか。そのほうが、100万ドルの機械よりずっと安いだろう」と思うかもしれません。一見すると、その方が安くつくように思えます。しかし、未熟練労働者を雇うシステム全体のコストを考えると、実は割高なのです。したがって、100万ドルのマシンを使うというシステム全体の方が、未熟練労働者を雇うよりも結局は安くて良いということになるのです。SREやプラットフォーム エンジニアに任せれば、あなたのために缶詰ロボットを作ってくれます。自分が昔はそうしていたからといって、缶詰を手作業で詰めることを彼らに強要してはいけません。

SREのような優れた実績のあるな手法を採用するには、同様に優れたスタッフがいなければ、より困難です。では、既存のチームには見込みがないのでしょうか。そんなことはありません。既存の人材を進化させることは十分に可能であり、大いに推奨できます。チームは、SREの経験を持つ外部の専門家やチーム全体を雇いたいと思うかもしれませんが、これは間違いかもしれません。同様に、スタッフの増員やオフショアによって（長期的に）SREの能力を獲得することを期待しても、期待するような結果は得られないと思われます。SREは従来の運用チームよりも単価が高い傾向があり、SREチームの人員配置に関して、低価格で高い価値を得る方法を見出そうとすると、失敗する傾向があります。信頼性が重視される組織であれば、このコストを合理化することができるはずで、後の節でその方法を探ります。

運用をサイロ化したコストセンターとして扱うのは、よくある間違いです。収益と総所有コストという全体像を考慮し、局所的なコストの最適化を避け、短期的なコスト削減に注力するだけでは、長期的には企業のコストがはるかに高くなることを認識する必要があります。例えば、故障シナリオを検討し、それが収益やブランドに与える影響を予測することで、チームは、緩和策や予防策のロードマップを完備した長期保険として位置づけることができます。理想的には、このチームは単なる「保険」ではなく、顧客にイノベーションを提供する能力を向上させる（信頼できる!）原動力にもなるのです。ホライズン2の目標を検討し、そのためのプラットフォームを計画しましょう。今日の問題を解決するだけでなく、将来を見据えた計画を立てましょう。

既存のスタッフを変革することの利点を考えてみましょう。適切なインセンティブと機会、そして十分な時間があれば、組織は規範を変革し、近代化された従業員の役割と責任を円満に採用することができ、また、不必要な離職を最小限に抑えることができ

ます。なぜなら、組織にとって最も価値のある資産は、間違いなく、常に「人」だからです。スタッフのスキルセットを評価する際に、自社のコアビジネスを本当に理解しているかどうかを過小評価してはいけません。

第3章
SREの原則

具体的な実践方法について話す前に、法律の分野でも法源と法解釈が重要であるのと同様に、原則に書かれた**文字通りの意味とその精神が意味すること**を明確に理解しておくことが重要です。原則そのものだけでは不十分で、SREの精神は原則の中にあるのです。また、実践は網羅的になりえず、むしろ実践は原則から生み出されたものであり、時間の経過に伴って変わりますし、組織によっても異なります。

原則とは、変革の基盤となる基本的な真理であり、意思決定の指針となるものです。ビジネス目標を達成する方法は複数存在するため、文言上は守れても精神的には守れないような網羅的なルールを設定するよりも、核となる原則に集中することを奨励する方がよいでしょう。Google では、新しいサービスをどのように設計し構築するかについて、複数の社内ポリシーがありますが、私たちが常に遵守しようとしている**基本原則**は、「ユーザーに焦点を絞れば、他のものはみな後からついてくる」です。

個人を無力化するような一連の指令に縛られるのではなく、あらゆるレベルで人々がリーダーシップを発揮できるようにすることに注力すべきです。特に、事業部門や管理職は、変革のシナリオに納得し、それぞれの専門分野において細かな指示に変更を加えることに積極的にならなければなりません。彼らのような影響力のある人々はは、納得してもらえれば最大の資産となり、納得してもらえなければ最大のハードルとなります。

原則と同様に、優れたポリシーは、タスクではなく結果に焦点を当てます。しかしながら、それらはより規範的なガイダンスです。ポリシーは、ビジネスにおける官僚主義に対抗するものではなく、それを活用するための手段なのです。ポリシーとポリシーの枠組みは、皆が納得したガードレールの中で、人々が安全に活動できるようにするもの

であるべきです。また、正しい方向に行動を誘導するために、適切なデフォルト値が含まれている必要があります。

アンチパターン: 物事がどのように実装されるかについて、前もって壮大な計画や設計を持つこと。

本来は、多くの時間をかけて学習する必要があり、一貫した原則に導かれたフィードバックループ（好循環）を構築することをお勧めします。

ここでは、SREの本から各原則の簡単な概要を説明し、それをどのようにあなたの組織に反映させるかを説明します。詳細については、「Site Reliability Engineering」の参照章を読むことをお勧めします。

リスクの受容（ SRE本 第3章 ）

これは、最も困難な最初のステップの一つです。よく、信頼性と速度のトレードオフという言い方をしますが、これは必ずしも正しいとは言えません。企業にとって信頼性を考える上で最も有益なのは、指数関数的なコストです。例えば、99.9から99.99というように9の数を増やすごとに、ソフトウェア、ハードウェア、人など、コストが1桁ずつ増えていきます。これが投資対効果をもたらすかどうかを考えることで、ビジネス要件に適合させることができます。また、障害の種類も非常に重要です。例えば、24時間365日稼働するサービスはSREに適しています（一日8時間、週5日稼働する社内システムとは対照的です）。また、積極的にメンテナンスされていないサービスでは、SREはあまり役に立ちません。なぜなら、これらのサービスでは、継続的な改善をあまり適用しないからです。特に、意図的に多くのリリースを行わず、新しいコードも書かないようにしている場合は、その傾向が強くなります。

アンチパターン: 信頼性100%のサービス

本来は、多くの時間をかけて学習する必要があり、一貫した原則に導かれたフィードバックループ（好循環）を構築することをお勧めします。

アンチパターン：「平常時」の運用で99.999%を達成する

月次のメトリクスや保守のためのウィンドウは、災害の影響を見えなくできます。

サービスレベル目標（SRE本 第4章）

SLOやSLAにこだわる前に、サービスレベル指標（SLI）から始め、システム上のデータを使用して正確な SLI を作成し、SLO/SLA の交渉をサポートすることができるようにします。既存のビジネス・コミットメントによってSLO/SLIの正確さと妥当性が左右されないようにしてください。メトリクスを用いて変化を促すか、変化によってメトリクスを動かすかを選択する必要があります。上辺だけをよく見せようとしたり、選り好みをしないでください。あなたの理論をサポートする都合の良いデータポイントを使用するよりも、あなたの顧客が何を望んでいるかを理解するために時間を費やしてください。要するに、証拠（SLI/SLO）があなたの結論（SLA）を後押しするようにするのです。SREは99.9%以上のサービスに集中させ、99.9%未満のサービスは（必要となるまでは）SREなしで維持させるようにしましょう。もしサービスがSLO/SLIから利益を得られないなら、おそらくSREからも利益を得られないということは、いくら強調してもし過ぎることはないでしょう。最後に、SLO違反が発生した場合にソフトウェアやプロセスの変更を行うことができなければ、SREの効果もあまり期待できないでしょう。

アンチパターン: SLO = SLA

常にSLOはSLAよりも厳しいものにしましょう。（例 SLO 99.95%、SLA 99.9%）

アンチパターン: SLI = OKR (objectives and key results)/KPI (key performance indicator)

ここで**グッドハートの法則**が適用されます。ある計測結果が目標になったとき、それは良い計測でなくなります。

トイルの削減（SRE本 第5章）

これは、SREが成功するために必要な生成的な文化と密接に結びついているため、おそらく最も重要な原則の1つでしょう。ほとんどの場合、企業のリーダーは物事のスピードを上げたいと考え、すべてのリソースが100%忙しい状態にすることでこれを実現します。もしあなたが、チームが間違ったことを早くやるのではなく、正しいことを早くやるようにしたいと心から思っているのなら、忙しい仕事（私たちは「トイル」と呼

んでいます）を50%未満にすることを目指すでしょう。これが規模に応じた信頼性（とスピード）の秘訣です。これを技術的負債と同じように考えてはいけません。つまり、貯め込んで後で返済すればいいようなものや、四半期に一度「トイル週間」として取り組むようなものとして扱ってはいけません。トイルがチームを圧倒してしまうと、他のすべてのSRE活動が停止してしまいます。あなたの組織にとってのトイルとは何かを決める必要があり、それは上から決めるのではなく、SRE実務者が決めなければなりません。また、トイルの定義は時間の経過とともに変化します（これも実践者が決めることです）。

アンチパターン：補足的な原則としてのトイル

トイルの削減を無視すると、SREの導入に大きな影響を及ぼします。トイルの削減のための時間がなければ、SREを導入する時間もないでしょう。

アンチパターン：トイルの削減が皆で共有されず、1人/1チームの仕事になっている

その仕事に最も近い人たちが、その仕事を解決する必要があります。もし、この仕事をオフロードしようとすれば、間違った行動を促すことになります。

アンチパターン：トイル削減週間

四半期に一度、「トイル削減週間」を設けたくなるのはよくあることですが、これではうまくいきません。トイルをなくすには、もっと体系的で継続的なアプローチが必要です。

分散システムのモニタリング（SRE本 第6章）

オブザーバビリティは、それ自体が専門的な分野であり、他の開発プラクティスと同様に十分な配慮と検討が必要です。現実的には、ほとんどの企業では、チームがより良い仕事をするために、さまざまなシステムに投資することが予想されます。「一枚のガラス」はうまく機能しませんし、何百ものツールが重なり合っている状態も好ましくありません。SREのユニークなユーザー ジャーニーと、システム間の論理的接続を診断し解決するために複数のツールを使用する必要性を理解することで、自分に合ったバランスを見つけるようにしてください。オブザーバビリティシステムは、投資と熟考

に値する内部製品として扱い、システムは常に変化しているため、「完璧な」ダッシュボードよりも使い勝手の良いツールを重視しましょう。過剰なアラートは過小なアラートと同じくらい悪いということを心に留めておきましょう。アクションが期待されない限り、アラートは人間の手に渡るべきではありません。このアラート学習サイクルを構築することは、学習を加速するための重要な方法であり、これを誤ると SRE は急速に疲弊してしまいます。

アンチパターン: 無視されるアラート

無視されたアラートでメールの受信箱がいっぱいになると、ノイズが多すぎるため、誰も重要度の高いアラートに反応しなくなります。

アンチパターン: SRE を置き換える「NoOps」ツール

ツールは SRE を補強するものであり、SREに取って代わるものではありません。運用を完全に排除しようとすることは不可能であり、SRE チームとの関係を悪化させることになるでしょう。

アンチパターン: 原因に基づくアラート

いろいろなことを記録することができますが、常に原因ではなく、症状に対してアラートを発してください。

Googleにおける自動化の進化（ SRE本 第7章 ）

自動化が最も重要なのは、信頼性が極めて高いレベル（99.99%以上）の場合で、このような目標値だと、人間が介入しなければならない場合、ほとんど必ずSLO違反が発生するためです。エラーバジェットの縮小に伴い、介入のバランスは、グレースフルフェイル、リトライなどのアプローチによるプロアクティブなメンテナンスにますます移行しています。自動化のための自動化もよくある問題で、悪いプロセスを修正するために時間をかけることは非常に重要ですが、チーム文化に組み込むのは非常に困難です。自動化は常に、システムの他の部分と同じように簡単にメンテナンスできる必要があります。

アンチパターン: プロセスの品質や適合性に関係なく、自動化を既定している

最高のコードとは、書かれていないコードなのです！プレイブックは、頻度の低いプロセスのための良い中間的な解決策です。

アンチパターン:「本当に重要な」部分への不必要な人手の介入

人間が関与するのは、真に意思決定が必要で、その権限が与えられている場合のみとすべきです。

リリースエンジニアリング（SRE本 第8章）

リリースエンジニアリングは、DevOpsチームがすでに取り組んでいる継続的インテグレーション／継続的デリバリー（CI/CD）プラクティスと広範囲に重なります。トップダウンのプラクティスを押し付けるのではなく、その作業を活用しましょう。また、プラットフォームチーム（規模によっては1チーム）に十分な投資を行うようにします。テストチームをプロセスの早い段階で参加させ、すべての段階でのテストについて考えるなど、リリース面ではできるだけテストを前の段階までシフトしましょう。開発者に過度の負担をかけず、リリースサイクルの各パートが価値あるものとして扱われ、他のパートと連携していることを確認しましょう。リリースパイプラインは、ほとんどのSREの問題の原因（つまり解決策）です。オンコールやメンテナンスのスタッフはしっかりと連携する必要があります。

アンチパターン: DevOps/SREチームがあらゆるもののリリースを行う

それは職種名が違うだけの運用作業者です。

アンチパターン: リリースエンジニアリングはCI/CDを導入しなければならない

継続的デリバリーは、それ自体が規律であり、プラットフォームと開発チームはその基盤を構築する必要があります（SREが支援します）。

簡潔性（SRE本 第9章）

チームの認知的負荷は重要であり、チームの任務の拡大や縮小に伴い、時間の経過とともに変化します。認知負荷に合わせて、チームの合併や分割ができるようにしま

しょう。本質的な複雑さとは、多くのものが理解しにくいということです。したがって、偶発的な複雑さを減らすインセンティブを与え、本質的に複雑なものを、例えば**ドメイン駆動設計（DDD）**のような、より小さく管理しやすい塊に分割してみてください。DevOpsから再利用するもう一つの重要な概念は、**ローコンテキストとハイコンテキストの比較**です。プレイブック、ドキュメント、**DiRT（Disaster Recovery Testing）演習**などのSREのコンセプトは、すべて物事をローコンテキスト化するための重要な要素です。より少ないコードとより少ない製品機能は、製品のインセンティブに反する可能性が高いので、信頼性への影響を考慮する際には、この点を必ず確認してください。

アンチパターン: シンプルであれば、理解できる

エグゼクティブダッシュボードは、すべてを有意義に表示できるわけではありません。無理に表示させようとしないでください。

アンチパターン: 年次レビューに基づく静的なチーム

ダイナミックなチーム編成は年に1回以上必要です。

これらの原則をどのように既存の組織に当てはめるか

これらの原則がすでにあなたの組織に完全に合致している可能性は極めて低いですが、それでいいのです。あなたのSREのバージョンがGoogleのものと完全に一致する必要はありません。原則だけは一致させてください。ただし、何を追求するかを意図的に選択し、既存の原則とのミスマッチをチェックし、この機会に意味のない指標がないか再確認してください（Eric Ries著 "**The Lean Startup**"（Crown Business）[†1]で説明されている「成功劇場」を参照してください）。もし、自信がないのであれば、確認し、変更する必要があると考えましょう。原則にヘッジをかけないようにしましょう。もし、何かができないと思うのであれば、実行を遅らせる方が、うまくいっているふりをするよりも良いのです。

†1　翻訳注: 日本語版があります。「**リーンスタートアップ**」（2012年、日経BP、ISBN978-4-82224-897-0）

組織破壊のミスを防ぐ

変更は、非常に異なる潜在的な影響を持つ可能性があります。新しい原則を採用する際に、必ずしもうまくいかない変更があるのは避けられないことです。変更の影響は、通常、元に戻す能力よりも重要ではありません。つまり、元に戻すのが最も困難な変更が、最大の苦痛をもたらすことがよくあるのです。たとえそれが間違いであったとしても、そこから学ぶことができるように、元に戻すのが簡単な変更に焦点を当てましょう。例えば、「組織変更」がうまくいかなかった場合、いつでも別の組織変更を行うことができますが、解雇した人間を戻すことはできません。

アンチパターン: コーディングができない運用担当者は全員クビにする

倫理的、法的な意味合いは別として、この決断を覆すことはできません。

アンチパターン: すべての開発者にrootや本番環境のアクセスを許可する

最小権限のような優れたセキュリティと運用の実践は、これまで以上に自動化に適用されます。

アンチパターン: 業務で最も重要なシステムを選ぶ

マラソンのトレーニングプログラムでは、初日に26マイルを走ることから始めることはないでしょう。

SRE採用の旅をフェイルセーフにする環境作り

失敗を想定し、そこから学び、改善するようにしましょう。**やっかいなこと**をするときは、必ず専門家を参加させましょう。しかし、**複雑なこと**をするときは、失敗に報酬を与えるか、失敗の予算を確保しましょう。ほとんどの組織では、失敗に純粋に報いることは難しいので、失敗の予算がより適切である場合もあります。これは、平均値や中央値ではなく、うまくいったことの上位n%で評価されることを意味します。このような行動は、リーダーシップチームの中でロールモデルとなるようにすることが重要です。

アンチパターン: 成功する限り、どんなリスクもサポートする

本当のリスク予算は、ある程度の失敗を受け入れるということです。

優先順位の相違に注意

リーダー層全員から絶対的な支持を得ている可能性があります。しかし、より可能性が高いのは、人々は、信頼性を求めながらも、変化とコストに対する正当な懸念を抱いているということです。図3-1に示すように、変化のJカーブを認識することを推奨します。これは、最初はある程度簡単に成功した後に、インパクトのある変化を実現するためのカーブが難しくなることを意味します。たとえば、新しい自動化システムを導入しても、大きな成果が得られる前に、一歩後退したように感じられることがあります。ルーフショット（屋根）とムーンショット（月着陸）の両方を行うことで、成功への準備をするようにしましょう。劇的な改善を目指すことは可能ですが、最初は控えめにしましょう。

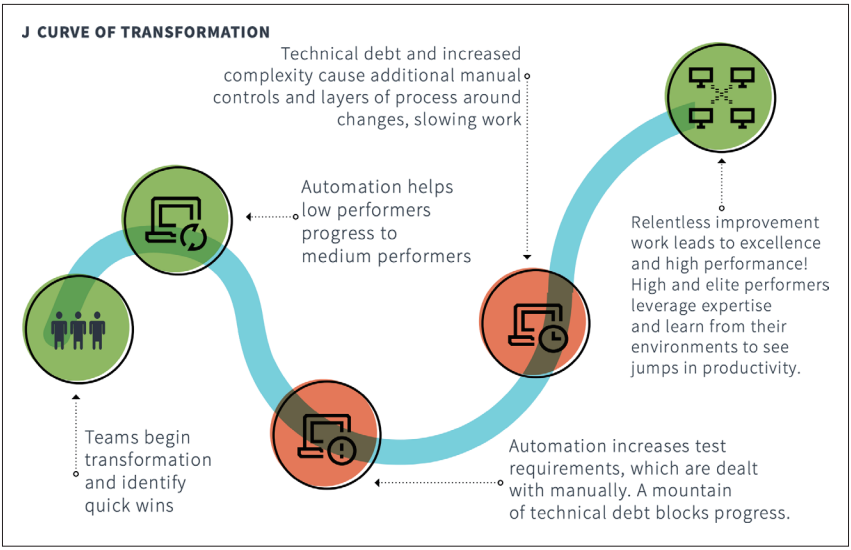

図3-1: DORA 2018 State of DevOps Report に掲載された変革のJカーブ

アンチパターン: あきらめるのが早すぎる。例えば、SREを6か月間試した後、すぐに成果が出ずにやめてしまう。

これは、前もってすべてを達成する必要があるということではなく、数四半期後に正しい方向へ進むための明確なシナリオが必要だということです。

これらの原則に賛同してもらい、必要な同意と支援を得るにはどうしたらよいか

John KotterやBJ Foggが言及したような、一般的な大企業の変革の原則を考慮し、SREを成功に導くように設定していることを確認してください。リーダーシップチームが、あなたのしようとしていることを十分に信じられなくても大丈夫です。しかし、最低限、変化を起こす緊急性とそれを実行する動機が十分にあることを確認する必要があります。

技術分野では、問題の発生を未然に防ぐのではなく、問題の解決に報酬を与えることが多く、SREの原則と実践の採用もこのような運用形態の犠牲となる可能性があります。SREの取り組みが数カ月後に「効果が不十分」という理由で断念されないように、SRE の導入による継続的な価値を可視化することが必要です。そのためには、投資対効果を実証するための適切な指標を見つけることです。例えば、小売業ではブラックフライデーの売上を最大化することに焦点を当て、ヘルスケアでは継続的なコンプライアンスと可用性に焦点を当て、金融では取引システムのスループットや分析パイプラインを完了する速度に焦点を当てるかもしれません。

第4章
SREのプラクティス

SREチームを設立し、原則を理解したら、次はプラクティスを開発する番です。チームのプラクティスは、メンバーが何ができるか、何を知っているか、どんなツールを持っているか、そしてこれらすべてを使って何をするのが快適か、から構成されます。

チームが行うことは、最初は彼らの憲章とその環境に基づいています。多くの場合、「開発チームがやっていないことすべて」がデフォルトになりますが、これは危険なことです。チームを運用業務のサブセットに集中させることで、時間の経過とともに、互いに積み重なる能力のはずみ車を生み出すことができます。もし、未定義のスコープで深いところへ放り込まれたら、苦労とフラストレーションが生じるのは間違いないでしょう。もう1つのよくあるパターンは、すでに負担が大きいチームにSREの仕事を追加することです。

チームが知っていることは、自主的に、あるいは中央で組織される教育によって拡大することができます。チームは、定期的に社員同士で教育を行うセッションを開催するよう奨励されるべきです。例えば、毎週1回、制作に関するどんな質問でも、新しいチームメンバーでもベテランメンバーでも歓迎される時間です。もしその質問に誰かが答えることができれば、教育セッションを開催することができます。誰も答えを知らない場合は、共同で調査することになります。私たちの経験では、このようなセッションはチーム全員にとって非常に価値のあるものです。経験の浅いメンバーは新しいことを学び、先輩は自分の知識を広める機会を得、そして誰も知らなかった新しい発見があることも多いのです。また、「**不運の輪（Wheel of Misfortune）**」と呼ばれるテーブルトップ演習は、緊急時の役割分担や緊急事態への対応について話し合うもので、ストレスのない環境で本番環境に触れることに慣れるために非常に有効な方法です。最近発生した

障害を再現することは、簡単に始められる方法です。もし、あるチームメンバーがダンジョンマスターの役を演じ、実際に起こった証拠を提示することができれば、他のチームメンバーは、自分ならどうするか、または、直接ツールを使って、そのイベント中のシステムを調査することができます。

また、チームは、自分たちが運用しているシステムに関する知識を開発チームから得るよう奨励されるべきです。これは、既存のシステムをよりよく理解するためのよい訓練であるだけでなく、新しい機器を直接導入したり、性能向上などのシステムの変更について議論し計画したり、スケーラビリティや一貫性に関する懸念に対処する機会でもあります。このような会話は、チーム間の信頼関係を構築する上で非常に価値のあるものになる傾向があります。

また、サードパーティ製の新しいツールの導入や、オープンソースソフトウェアのツール、あるいはチームが独自に作成したツールによって、チームの能力を拡張することも可能です。

何から始めるか

チームに能力を追加する場合、どこから手をつければいいのでしょうか。信頼性とSREの問題空間は広大であり、すべての能力が同時に適切であるとは限りません。私たちは、チームが次に取り組むべきことを学べるような一連のプラクティスから始めることを提案します。抽象的には、PDCA（Plan-Do-Check-Act）と呼ばれるモデルを参照しています。システムが現在どのように動作しているかに基づいて次のステップに進むことで、次のステップは常に適切なものになります。この章の後半で、これらの能力のプラットフォームをどのように構築し、どこから始めればよいかを説明します。この初期の能力のセットははずみ車を形成するため、チームは次に何を構築し、何を採用すべきかを推測する必要がなくなり、システムの観察から自然に発展していきます。

どこへ行くのか

目標を適切に設定することが重要です。すべてのシステムが「ファイブナイン（99.999%）」で超高信頼である必要はありません。サービスやアプリを信頼性の必要性に応じて分類し、それに応じて投資のレベルを設定することをお勧めします。前回も述べましたが、「9が一桁増えるたびに、その前の桁の10倍のコストがかかる」、つまり

「99.99%のコストは99.9%のコストの10倍」であることを忘れないでください。この文は正確に証明することは困難ですが、原理は真実です。したがって、やみくもに、あるいは検討もせずに大雑把に目標を設定すると、コストがかかり、努力が水の泡になる可能性があります。また、信頼性目標を必要としないシステムに過剰な信頼性目標を押し付けることは、優秀な人材を失う良い原因になります。地球低軌道に乗れればいいのであれば、月を目指してはいけません。

成功への道は、目標に向かって少しずつ前進する「ルーフショット」であるようにしましょう。一度の大きなプロジェクトや革命で達成できるとは思わないでください。漸進的な改善こそが、行うべきことなのです。

チーム内で新しいプラクティスを展開する際には、得られる利益を必ず記録してください。このような成果は、チーム内だけでなく、利害関係者や他の同僚チームにも宣伝する必要があります。仲間からの評価は非常に重要で、チームの立ち上げでメンバーを褒める、ステージに上がって惨事を回避した方法を語る、ニュースレターでヒヤリハットを公表する、予防策がなかったら起きていたかもしれないことを組織全体に訴える、などがあります。特に過去にそうしてこなかった環境では、このような仕事を称えることが重要です。口頭や書面による賞賛は、金銭的なボーナスや贈り物と組み合わせることもできます。小さな贈り物でも、大きな効果が期待できます。

どうやって行くのか

長期的（例えば3年）な詳細な計画を持とうとしないようにしましょう。そのかわり、進行方向を知ることに重点を置いてください。指針となる北極星を知り、一段階のステップを達成するたびに、次の段階となるステップを生み出していきます。方向性が定まったら、新しいモデルにそぐわない既存のチームやプロセスを「爆破」する必要はありません。そのかわり、正しい方向に「舵を切る」ようにしましょう。

これは、目的地はわかっているけれど、途中の不都合に備えるという「**戦場の霧**」のようなアプローチだと考えています。特に初期の段階では、短期的な計画と機敏な対応が不可欠です。短期間で成果を上げ、すぐに効果を実証することで、立ち上げ間もないプログラムとチームの士気に大きなプラス効果をもたらします。今日の問題を解決する達成可能な目標を設定し、複数のチームが使用できる汎用的で再利用可能な機能の構築を開始します。このような機能を提供するプラットフォームを構築することで、

投資の効果を拡大することができます。この章では、プラットフォームと能力という概念について、後ほど詳しく説明します。

組織内のすべての製品開発チームが、そのニーズと現在の能力において同等であるとは限りません。SREを企業に導入する際には、そのエンゲージメントモデルを柔軟に設定するよう努める必要があります。製品開発チームの現状を把握することで、今日の問題を解決すると同時に、組織全体の規範やベストプラクティスを導入することができます。SREチームが軌道に乗ると、多くのチームが彼らの助けを求めているため、サービス提供が過剰に感じられることがあります。明確なエンゲージメント「メニュー」を作成することで、単発のエンゲージメントや持続不可能なモデルを回避することができます。エンゲージメントモデルには、組み込み型、コンサルティング型、インフラ型など、いくつかの種類があります。これらは、**GoogleのCustomer Reliability Engineering（CRE）チームのブログポスト**や、**SRE本の第32章**で紹介されているモデルでよく説明されています。

SREを導入する場合、組織体制を早期に明確化することが重要です。私たちは、SREリーダーが経営陣と同じテーブルにつくような、独立した組織を推奨しています。SREのリーダーを製品開発部門から切り離すことで、SREチームは信頼性という中核的な目標に集中しやすくなり、速度や機能の実現に意欲的なチームから直接的なプレッシャーを受けることがなくなります。しかし、SREは企業の他の部門と密接に連携することが重要であるため、「運用」サイロを孤立させないよう注意してください。開発チームは、このような共通のSREチームに投資し、このチームから得られる価値が、開発チーム内でSRE機能を構築するよりも高くなるようにする必要があります。

SREを可能にするものは何か

SREを可能にするものは何でしょうか。SLOやポストモーテムのような一連のプラクティスだけなのでしょうか。そうではありません。それらは実は、そもそもSREを機能させた文化の産物なのです。したがって、SREの導入を成功させるには、単にプラクティスを模倣するだけでなく、適合する文化を採用しなければならないのです。

この文化は、チーム自体の信頼と安全性に根ざしています。チームは、主要なシステムをコントロールするというプレッシャーの強い立場に置かれたとき、心理的に安全であると感じなければなりません。仲間やリーダーに対して、報復を恐れることなく

「ノー」と言えること。自分の時間が大切にされ、自分の意見が聞き入れられ、自分の貢献が認められていると感じなければなりません。そして何より、SREは開発組織内の担当者よりも「特別」あるいは「劣っている」と感じさせてはいけません。これは、歴史的に否定されてきた開発部門対運用部門のモデルに基づいているため、よくある落とし穴です。

よく知られている例は、責任の所在を明らかにしない事後分析です。何が悪かったのか」を書き出すことで、チームは、技術的または手続き的な障害につながる要因を共同で特定することができます。人為的なミスが発生した場合、「ヒューマンエラー」としたくなることがよくありますが、これはあまり意味がなく、システムを改善するための効果的な方法ではないことが分かっています。そのかわりに、SREは「非難のないこと（blamelessness）」を推進します。簡単に言うと、「人間がミスをしにくいシステムであること」です。オペレーターの入力を検証するために自動化とチェックを行い、合意と協力を促進するためにピアレビュー（peer review）[†1]を奨励する必要があります。自分がミスを犯したときに事後報告書に人々が自由に名前を載せるような状況であれば、非難のない振り返り（blemeless portmortems）をしているとわかります。誰にでも起こりうる簡単なミスであれば、恥も外聞もなく、降格もなく、業績評価も下がらないとわかっていれば、自分がミスをした状況について報告書に自由に名前を書き込むことができるのです。もしあなたが、事後報告で「あるエンジニア」や「担当者1」と書かれているのを見たなら、これは責任のない良いやり方だと思うかもしれませんが、実はこれは、直接対処しなければならない根本的な文化的問題が原因である可能性があります。もし名前が編集され、紙の上で「あるエンジニア」または「担当者1」に置き換えられても、事後報告の文脈の外でエンジニアに非難が集中するなら、非難する文化に対処できていないことになります。ログや文書から名前を明示的に削除するプロセスを自動化することは、文化的な問題を解決するものではなく、文書を読みにくく、理解しにくくするだけです。表面的に名前を消すのではなく、その下にある文化に対処して、非難がないようにするのです。

悪い文化の兆候として、スイカのメトリクスがあります。つまり外見は緑だが、中身は赤い、ということです。これは、チームの努力を反映した測定基準で、良く見えるように工夫されているが、実際には本当の欠点が隠されているものです。これはグッド

†1　翻訳注: ソースコードや設定の変更を修正者以外の第三者に確認してもらうレビューのこと

ハートの法則と同じで、「目標になるような測定は、良い測定でなくなる」というものです。例えば、サポートチケットの枚数や全体の平均解決時間（MTTR）に注目すると、意図的に、あるいは善意の人が自分の間違いに気づかず、悪用されることがよくあります。チームの活動を測定することは、その活動を目標にすることであり、顧客の成果を測定することではありません。そのかわり、チームは、顧客の幸福、システムの安定性、開発速度といったものを直接代表するような、独自の成功指標を定義できるようにすべきです。

SREは、単に「20%の時間」に行う役割ではなく、組織内で専用の肩書きと役職を持つべきです。また、異動要件や昇進の見込みを明示したジョブラダー[†2]が必要です。レベル分けと給与は、チーム間で公平であるべきです。異動があっても、大きな影響を及ぼしてはいけません。

確立されたSREチームが成功しているかどうかを知る良い方法は、SREへの異動とSREからの異動を見ることです。異動が日常的で、官僚主義や制限のようなものがないことを確認することで、人々がSREに「行き詰まり」を感じているのか、それとも望ましい役割なのかがすぐに分かるでしょう。開発部門からSREへの自主的な異動率を観察することで、それがうまくいっているかどうかを知ることができます。

SREは、自分の時間には価値があることを知らなければなりません。特に、業務が「通常時間」を超えている場合はそうです。Googleでは、SREが通常勤務時間外に対応しなければならない場合（いわゆる「オンコール」）、時間的な補償を行うことを規定しています。Googleのいくつかのチームでは、オンコール対応エンジニアに対して、オンコール時間に対する一定の割合で、金銭的な補償か休暇のどちらかを選択できるようにしており、多くの場合、事前に合意した上限が設定されています。チームによって提供される以上の要求があってはならないので、オンコールプールが十分なサイズであることを確認することが重要です。よくある間違いは、オンコールプールを SRE だけから構成することです。これは人為的な制限です。また、オンコールプールはオプトインベースで行われるべきです。チームが自分たちの時間が乱用されていると感じると、すぐに下降線をたどってしまうからです。

もう一つの文化的な接点は、計画や目標設定です。SREは本番環境の問題に最も近いところにいるので、何が最も重要か、何が燃えているか、何が最も苦痛を与えてい

†2 翻訳注: 外資系大手企業で一般的なジョブ型雇用において、特定の職種の中での各職位を定義したものを指します。

るかということをよく理解している傾向があります。SREチームが自分たちで優先順位やロードマップを設定できるようにすることで、そのチームに力を与え、より効果的に、より幸せになることができます。経営陣は、期待される成果について、合意された共通の理解を深めるという実践に従わなければなりません。ビジネスがより速く進む必要があるのでしょうか。ユーザーはより早く結果を得る必要があるのでしょうか。この反面、よくあるパターンがテイラーイズム[†3]です。つまりリーダーが独自に詳細な計画とタスクを設定し、優先順位をつけ、それを労働者に割り当てるというモデルです。

能力のプラットフォームの構築

SREチームは、パートナーチームに機能を提供するためのプラットフォームを構築することができ、理想的には、その貢献度を時間とともに組織全体に拡大していくことができます。共有サービス、プラクティス、規範、コードにレジリエンス機構を導入すれば、自動化、コード、共有ライブラリ、パイプライン、手順、規範、文書、プレイブック、さらには、人の頭の中だけに存在する文書化されていない特別な知識からなる共有プラットフォームを構築することができるようになります。各チームが独自のベストプラクティスを構築するのではなく、プラットフォーム上に構築することができるのです。製品はプラットフォーム上でゼロから構築する（いわゆる「デジタルネイティブ」）ことも、プラットフォーム上に移植することもできます。時間が経つにつれてプラットフォームの機能が向上し、チームがその運用特性に自信を持ち、快適に使用できるようになれば、より重要なワークロードをプラットフォームに移植することができます。このように能力をプラットフォームに組み込むモデルを採用することで、SREチームは、多くのサービスに能力をまとめて適用することで、その影響力を拡大することができます。プラットフォームは内部製品であり、サービスチームを顧客として扱い、機能要求を受け付け、不具合を追跡するように管理する必要があります（図 4-1 を参照）。

†3　翻訳注:「**科学的管理法**」と呼ばれています。

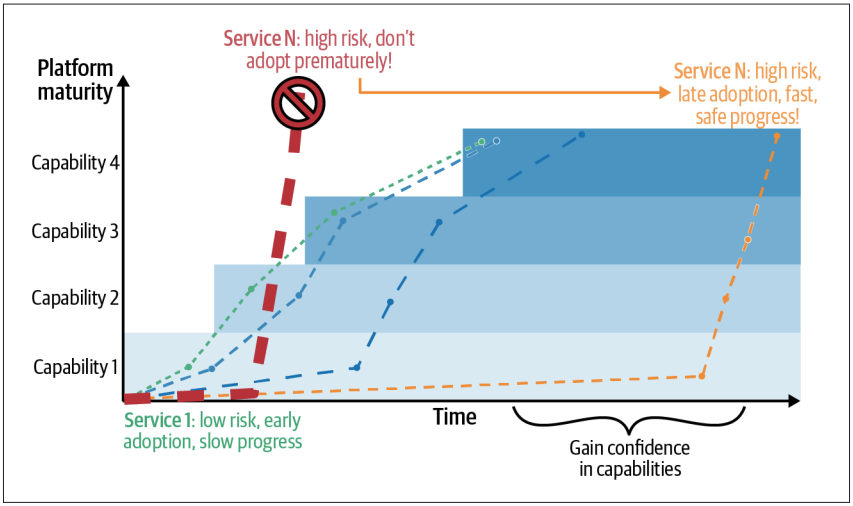

図4-1: 機能のプラットフォーム

チームがプラットフォームを構築する際、「最初に何を作るか？」という問題が発生します。リスクの低いサービスを最初に採用することで、そのリストをMVP（Minimum Viable Product、実用最小限の製品）にすることで最小限に抑えることができます。時間が経てば、さらに多くの機能を追加することができます。でも、次はどれを作ればいいのでしょうか。考え方は2つあります。開発者とあなたの環境です。つまり、「メッセージ バスが必要だ！」といった開発者からの要望と、「スケーラブルなサービス発見システムが必要だ、さもなければこれはうまくいかない」といった開発者が必要とすることがわかっているものを構築するのです。

環境面では、以下のようなものが挙げられます。

- ソフトウェア開発ライフサイクル（SDLC）の強化や、より多くのコードをより速く、より安全に提供するためのDevOpsの改善
- リライアビリティエンジニアリングの改善：侵入したエラーによるリスクの最小化

リライアビリティエンジニアリングの改善には、チーム内で「好循環」を展開することをお勧めします。何を改善したらよいかわからない場合は、障害の状況を見ながら、次のようなことをするとよいでしょう。

- SLOを導入する

- インシデント対応を正式化する
- 責任の所在を明確にした事後報告とレビューの実施
- リスクモデリングによる優先順位付けを行う
- エラーバジェットやその他のリスク許容度に基づいて、信頼性バックログを燃やし尽くす

このサイクルをはずみ車として、新しい機能を生み出すのです。たとえば、デプロイメントによってバグが発生し、サーバーがクラッシュするような障害が発生した場合、カナリアリリース、実験フレームワーク、またはその他の漸進的なロールアウトを使用して、爆発の半径の縮小などのリスクを軽減する方法を開発する必要があります。同様に、メモリリークが発生した場合は、プリデプロイメント パイプラインに新しい形式の負荷テストを追加することもできます。これらはいずれも、プラットフォームに追加される機能であり、プラットフォーム上で動作する各サービスに利益と保護を提供することができます。汎用的な緩和策がその価値を示すように、単発の修正はまれになっていきます。

リーダーシップ

もちろん、そのようなプラットフォームを構築するためには、機能開発に充てられるはずのエンジニアリングの時間を割く必要があります。そこで必要なのが、上流工程への影響力です。開発者の才能を機能と安定性の両方に使う場合、トレードオフを行わなければなりません。このトレードオフを行う人が、全体像を把握し、適切なインセンティブを持つようにすることが重要です。信頼性に関する戦略的な意思決定の場に席を置く、組織内の上級管理職である最高信頼性責任者（Chief Reliability Officer）の役割を目にする機会が増えています（Mark Schwartz著の「A Seat at the Table」（IT Revolution Press）のファンにはお馴染みのコンセプトかもしれませんね）。この役職は、SREを成功させるための一般的な職務ですが、一般的な役職名ではなく、既存の役員が兼任していることが多いようです。

機能しているかどうかを知る

信頼性を理解し、その価値を認めているような、うまく機能している組織は、いくつかの顕著な特徴を備えています。まず、信頼性に問題がある場合、機能提供の速度を

落としたり、停止させたりすることができることです。スピードとリリースが唯一の目標である場合、信頼性やその他の非機能的な要求が常に犠牲になります。信頼性の確保が機能より優先順位を下げられることはないでしょうか。プロジェクトが提案されたものの、「時間がない」という理由で終了しないことはありませんか。ここで重要な注意点は、コードデリバリー パイプラインを遅くしていると見なすべきではないということです。アクセルは踏み続けることです。

もう一つの成功の指標は、個人のヒロイズムが賞賛されなくなり、むしろ積極的に奨励されなくなったときです。システムの成功が少数の人々の肩に支えられているとき、チームは持続不可能な英雄主義の文化を作り出し、それは崩壊するに違いありません。ヒーローは自分の知識を独占しようとし、その知識を使う必要性を組織的に防ごうとする意欲を失います。これは、Gene Kim、Kevin Behr、George Spafford著「The Phoenix Project」(IT Revolution Press) [†4]に登場するBrentという人物に似ています。Brentは非効率的であるだけでなく、危険でもあります。チームは、チームの責任を果たしつつ、個人のヒロイズムを積極的に阻止しなければなりません。なぜなら、ヒロイズムは、短期的な思考では合理的なアプローチに感じられるからです。

また、信頼性向上への取り組みが、障害前に事前計画の一部として予算が確保されていることも、チームがうまく機能していることの証です。一方、機能不全のチームでは、信頼性向上への投資が、障害や一連の障害を処理するために使われていることが見受けられます。このような投資は必要なことかもしれませんが、一過性のものとして扱われ、「状況が良くなったら」放棄されたり、回収されたりするのではなく、長期的に維持される必要があります。

このことをさらに説明するために、組織の信頼性へのアプローチを2つのモードに単純化して考えてみましょう。「平時」と「戦時」です。それぞれ、「物事はうまくいっている」状況と「誰もが知っているように、すべてが崩れ落ちそうになっている」状況です。この2つのモードを区別して考えることで、投資に関する選択ができるようになるのです。戦時中は、プラットフォームの隠れた機能、インフラ、プロセス、トレーニングなどに多くの時間とお金を費やします。平時には、そのような作業を放棄するわけではありませんが、投資額が大幅に減ることは確かです。

しかし、会社が戦時下にあることを誰が決めるのでしょうか。その決定はどのように

†4 翻訳注: 日本語版である「The DevOps 逆転だ!」(2014年、日経BP、ISBN978-4-82228-535-7) が出版されています。

なされるのでしょうか。パニックや人員削減を招かない方法で、どのように社内に伝達するのでしょうか。一つの方法として、優先順位の高いコード（例えば、コード・イエローやコード・レッド）を使用する方法があります。この2つは、仕事の優先順位を決める際にチームを支援する組織的な慣習です。コードイエローは、技術的な問題が1四半期以内にビジネス上の緊急事態になることを意味します。コード・レッドは、数日以内に同じことが起こるか、すでに存在する脅威に対して使用されることを意味します。これらのコードは、明確に定義された基準を持ち、リーダーシップチーム全員が理解し、同意する必要があります。意図した効果を得るためには、その宣言がリーダーによって承認される必要があります。このようなコードの結果、チームの優先順位が変更され、（コード・レッドの場合のように）既存の作業が中止される可能性があり、多額の支出が承認され、他のチームを直接支援することができるようになるはずです。優先順位の高いコードを宣言すると企業にとってコストがかかるため、明確な成果があることを確認する必要があります。これらの成果は、最初から終了基準として定義され、完了時にははっきりと明示されるべきです。これがないと、チームはコードの宣言による疲労を経験し、適切な対応ができなくなります。

信頼性への投資という選択

このような信頼性のリーダーの権限で、それほど劇的でない変化は何でしょうか。それは、方針と支出です。組織全体の方針を決めるには、ボトムアップで進めると一貫性がなくなる傾向があります。それよりも、チームから提案された方針を吟味し、検証し、承認し、普及させるリーダー的役割を果たす方がはるかに効果的です。同様に、人材派遣、ハードウェア、ソフトウェア、出張、サービスなどに会社の資金を使う場合も、階層的な方法で行われることが多いようです。

先に述べたような仕組みを作る前に、組織における信頼性の価値を考えなければなりません。そのためには、信頼性をコストセンターとしてではなく、投資として、さらには製品の差別化要因として考えなければなりません。信頼性こそ、隠れた最も重要な製品機能であるということです。使えない、遅い、エラーが多いといった製品は、その機能にかかわらず、顧客にとってはるかに有益ではありません。この方向性の設定は、特にこれが新しい方向性である場合は、一貫したトーンを設定するために、幹部レベルで行われなければなりません。

信頼性は、コード品質のような、より理解しやすい概念の代用品になり得るというの

が、この単純な論拠の一つです。もしシステムがユーザーから見える障害を発生させる場合、コード品質の問題に直接取り組む前に、段階的な変更などの信頼性プラクティスを適用することで、エンドユーザーからはシステムの障害が少なくなったように見えるようにできます。例えば、1%の顧客に不具合のある変更を展開することで、そのうちの99%の顧客は問題を経験しないことになります。これにより、システムは実際の100倍良く見えるようになり、サポートコストと風評被害を減らせます。意思決定を行う信頼性をより強い製品への投資として設定することで、より大きな効果をもたらす長期的な計画を立てることができるのです。従来のモデルでは、ITはコストセンターとして扱われ、そのコストを削減することにのみ焦点が当てられていました。結局のところ、サービスが安くても、それが稼働していなければ意味がありません。コスト削減を適用することは可能ですが、信頼性の目標を達成した後に検討すべきです。信頼性目標を維持するためのコストが高すぎる場合は、その目標を明示的に再定義し、例えば「9の数を減らす」というように、その結果として生じるトレードオフを評価することができます。

これらの目標を達成するためには、理事会、意思決定者グループ、経営幹部を説得する必要があります。また、長期にわたってチームを維持し、リソースを提供し、チームメンバーを訓練し、さらに成長させるためには、彼らの賛同が必要になります。これは、長期的な投資と見なされ、他の予算の隠れた項目としてではなく、それに応じて明確に資金を提供する必要があります。

アンチパターン: オデュッセウス[†5]を無視する信頼性に関してよくあるアンチパターンは、障害やその他の「悪いニュース」が予想される場合でも、計画サイクルに影響を与えることです。悪い知らせを前にすると、リーダーは「何かしなければ」という気持ちになりがちで、「計画を守る」ことにインパクトが感じられないことがよくあります。しかし、障害が起こることを想定した計画であれば、システムの理解に大きな変化がない限り、「計画を守る」ことはまさに正しい行動なのです。ここで、オデュッセウスの盟約という言葉を使うと分かりやすいでしょう。これは、リーダー（オデュッセウス）がチーム（船員）に対して、「計画を守れ（マストに縛られたままセイレーンの前を通過する）」と指示するものです。オデュッセウスがマストに縛られたままセイレーンの前を通り過ぎる。そして、（「もうやめてくれ」と叫びながらも）チームがその計画を守り通したとき、彼は彼らを

†5 翻訳注: オデュッセウス（Ulysses）はギリシャ神話の英雄で、智将として知られています。

祝福するのです。彼らは、短期的な思考に誘惑されませんでした。彼らは長期的な影響を考慮した計画を立て、混乱が始まる前に明確な計画を立てる時間があったのです。

チームにその場しのぎの決断をさせることは、優れた計画を無視し、感情やエゴに左右される選択をすることになりがちです。この典型的な例は、リーダーが「障害の中に入って」、十分な状況を把握しないまま、また有能なチームがすでにコントロールしているにもかかわらず、指揮を引き継いでしまうことです。これは、企業文化の結果であることが多いのです。HiPPOs（最も高給取りの意見に基づく意思決定）の文化は、インシデント管理と信頼性全般に劇的な悪影響を及ぼしかねません。そのかわりに、オデュッセウスに耳を傾け、計画に固執し、船を放棄しないことです。これは、インシデント対応だけでなく、エラーバジェットの枯渇や、「本当に悪い」インシデントに直面したときのSLOの追跡などにも当てはまります。エラーバジェットの枯渇に直面して機能リリースを中止する計画なのに、毎回「この重要な機能」のために例外を設けると、あなたのリーダーシップは大きく損なわれてしまうでしょう。これを改善するために有効なプラクティスが、「銀の弾丸」の導入です。これは、リーダーに3つの銀の弾丸を付与し、想定した計画を変更する必要がどうしても出た時に弾丸の回数だけ上書きできるようにするものです。このように人為的な希少性を導入することで、リーダーは明確なトレードオフを要求されます。同様に、たった一つの悪い出来事でSLOが消滅してしまった場合でも、それを無視してはいけません。チームを集めて、この出来事によってシステムに対する理解がどのように変化したかを分析します。このような障害は、これまで考慮されたことがなかったのでしょうか。障害発生時の対応は不十分だったのでしょうか。

アンチパターン：両方を同時に行うもう一つのアンチパターンは、古いモデルと新しいモデルをそのまま混ぜようとすることです。これは、チームをおかしな方向に引きずり込むので、避けるべきでしょう。例えば、ITILの問題管理の場合、中央のチームは問題管理者を通じてすべての問題の原因と解決時間を削減することが期待されています。これに対し、SREでは、エンベデッドエンジニアがポストモーテムやレビューを通じて、自ら問題解決を推進することが期待されています。結果（障害の減少、短縮）は一致していますが、手法やペルソナは大きく

異なっています。両方を同時に行おうとすると、結局は混乱に陥り、両方のアプローチから得られる意図した成果が互いに衝突し、苦しくなってしまうのです。このようなSREと非SREの悪い組み合わせは、実際の薬でも飲み合わせが悪いと悪影響があるのと同じように、「有毒な組み合わせ」と呼んでいます。それぞれを単独で使用することは有益ですが、この2つを一緒に使用すると、意図しない悪い結果を引き起こします。多くの場合、既存のスタッフを参加させようとしたり、報告の継続性を図ったりするために、両方を使う背後に善意を見出すことができます。しかし、その魅力は、障害の長期化、労力の増加、信頼性の低下といった、悪化した結果にはるかに及ばないのです。

人材確保と維持

人員配置と役割の定義もまた、アンチパターンとなり得ます。SRE チームを構築する際、既存のチームに秩序を与えるために外部から SRE を採用したくなることがあります。この場合、採用された SRE は既存のチームや技術のニュアンスを理解できず、新しい業務で実際に妥当かどうかわからないまま、以前使用していた方法を適用することになり、無駄な労力を費やすことになりがちです。

そのかわりに、既存のチームをSREチームに成長させることをお勧めします。単に名前を変えるだけでは効果的ではありませんが、体系的な学習経路と成長・発展するための環境を提供すれば、確実に効果があります。もちろん、移行が失敗するケースもあります。成功するための環境が整っておらず、「本を読めば」すぐにシニアSREになれると期待されると、不満が募って他の職を探す可能性があります。同様に、エンジニアの中には変化の理由が分からず、インセンティブが得られないなど、新しい役割を採用することに強い抵抗を示す人もいます。有給の教育、学ぶための時間と余裕、そしてなぜその変化が必要なのかをチームが理解できるような背景を提供することで、チームをSREの役割にうまく移行させることができます。これには時間と労力、そして忍耐が必要です。うまくいかない場合、特に移行について、また個人にとって何がうまくいったかいかなかったかについて、終了時のインタビューを行うことが重要です。計画の不備が見つかったり、意図したとおりに実行されていないことがわかったりするかもしれません。最後に、より複雑でインパクトのある仕事をチームに依頼する場合、これは文字通り価値の高い仕事であり、チームにはそれに対する報酬が必要であること

に留意してください。つまり、チームがSREのように振る舞い始めたら、SRE のような報酬を支払うべきであり、さもなければ彼らは報酬が良い他のところに移ってしまうでしょう。もしあなたがチームに高い価値のあるスキルを身につけさせるために報酬を支払ったのに、彼らがそのスキルを使うために他の場所に行ってしまったら、あなた自身の責任にしかなりません。

技術向上

既存のスタッフをSREに成長させ、移行させる場合、技術向上計画を立てることが重要です。この計画には、「何を」「どのように」の両方が含まれます。つまり、その役割に必要なスキルは何か、また、スタッフがそれらのスキルを習得できるようにするためにはどうすればよいか、ということです。スキルギャップ分析や調査などのツールは、職務に必要な基礎的スキルについての評価を確認する上で、本質的に有用です。これらのスキルは、SREの文献ではあまり言及されていませんが、SRE が組織全体に貢献するために不可欠なものです。例えば、従来の運用チームは、バージョン管理、ユニットテスト、ソフトウェアデザインパターンなどのソフトウェアエンジニアリングの基礎に馴染みがないことが少なくありません。このようなベースラインを技術向上計画の一部とし、それぞれの学習者のプロファイルに合わせて調整することは、チーム内でスキルのクリティカルマスを確立するだけでなく、個人が新しい役割にスムーズに移行できるようにする（つまりチームの離職率を下げる）ために非常に重要です。

第5章
積極的な成功体験の育成

SREが組織にとって価値あるものであると判断し、投資を決意したなら、その投資を成功させることが重要です。システムに変化をもたらすのは常に難しいことですが、その変化を定着させるのはさらに難しいことです。ここでは、SREを組織で機能させ続けるためのヒントをいくつか紹介します。

大きく考え、小さく行動する

「測定できなければ、管理できない」という言葉は、エドワード・デミングとよく結びつけられます。しかし、完全な引用は「測定できなければ管理できないなどというのは間違いであり、コストのかかる神話である」というものです。SREは、その核心において、メトリクス駆動の方法論です。しかし、いくらSLOやSLIを作成しても、SREの導入がうまくいっているか、企業戦略に合致しているかどうかを理解することはできません。これは、継続的な実験と学習を通じて見つける必要があります。

これまでの章では、「大きく考える」ことを求めてきましたが、成功を育むためには「小さく行動する」ことが必要です。どんな種類の大規模な変化も、**反復的かつ漸進的に**達成されるものであり、SREもこの課題と無縁ではありません。この点については、明白な注意点もあります。あまりに短いタイムラインでは、意味のある変化を起こすことができないので、バランスを取る必要があります。

Googleでは、社内で共有された目標と主要成果（**OKR**）を使用して、チームを調整し、達成方法が必ずしも明確でない場合に目標を設定しています。あなたの組織には、これを行うための独自のプロセスがあるかもしれませんが、SREチームのメトリクス（ト

イル、アラート、ソフトウェアエンジニアリングの影響、キャパシティ計画など）の明示的な反復と定期的なレビューを含めるように拡張する必要があります。SREプラクティスの採用が非線形であるということは、進捗には常に後退が伴うということであり、これもプロセスの通常の部分として扱われるべきです。

文化は戦略を糧に成長する

Googleが前提としていることの1つは、SREのストーリーに書かれていない重要な部分であり、根底にある生成的なGoogleの文化です。Googleはまた、これらの属性について説明するために**行った調査**を共有しました[†1]。その結果、誰がチームにいるかということよりも、チームメンバーがどのように交流し、仕事を構成し、貢献をどのように見ているかということのほうが重要であることがわかりました。

Googleでは、成功するチームには、他のチームと異なる5つの重要な力学があることを学びました。

心理的安全性

このチームで、不安や恥ずかしさを感じることなく、リスクを取ることができるか

頼りがい

質の高い仕事を時間内に行うために、お互いに信頼し合えるか

構造と明瞭さ

チーム内の目標、役割、実行計画が明確になっているか

仕事の意味

私たち一人ひとりにとって、個人的に大切なことに取り組んでいるか

仕事の影響

自分たちのしている仕事が重要だと、根本的に信じているか

SREの導入で見られる典型的な懸念の多くは、コストへの影響、特殊な業界への懸

†1　翻訳注：日本語訳版があります。https://rework.withgoogle.com/jp/guides/understanding-team-effectiveness/#introduction

念、技術的負債などのようなものです。しかし、この発見で最も良かったのは、他の良いものと同様に、これら5つの力学は本質的に無料であるということです。業界や状況に関係なく、これらの項目を優先事項とすることを検討してください。Googleの高パフォーマンスのチームは、SREを効果的にこの文化的基盤からの**創発的**行動にし、これらの文化的規範に基づいてSREを成功させました。

文化を避けることも待つこともできない

SREの導入を成功させるには文化が重要であるという話を聞くと、大抵の場合、ムッとします。SREを導入する前に文化が特定の段階まで到達するのを待つ必要があるということを暗示しているからです。よく使われることわざを借りれば、文化を変え始めるのに最適な時期は20年前だったのかもしれませんが、2番目に良い時期は、今なのです。また、信頼性のフィードバックに対処できる企業文化を作らないことは、信頼性の問題以外にも大きな**影響**を及ぼします。

SREを育てるとはどういうことか

SREを育成し、成長させるためには、いくつかの重要な活動を検討する必要があります。

1. サブリニアスケーリング

このことは既に述べましたが、これは「より少ない人数でより多くのことを行う」ということではなく、自動化と継続的改善の文化を利用して、信頼性問題へのアプローチ方法を変えるということであることを明確にすることが重要です。そのため、ソフトウェア**組立ライン**の既存のステップに人を増やそうという誘惑に負けず、SREを利用してそれらのステップを自動化または排除してください。

2. 持続可能で幸福なチームの構築と維持

技術業界では、「**プロジェクトからプロダクト**」へのアプローチに移行していますが、個人を気まぐれに活動の間を移動する代替可能なリソースとして扱うことは、まだ非常に一般的です。これは、私たちの文化的なアドバイスと真っ向から対立するものです。このようなことをしても、SREで成功するとは思わないでください。

3. SREは静的なものではない - 本来は動的な役割であり、時間とともに成長することを認識すること

トイルを削減し、自動化を導入するという進化のプロセスの一部として、SRE は組織内で進化していくことになります。それでも、予算や計画を立てることはできますが、特定のタスクや固定されたチームサイズではなく、結果を出すことを目指してください。これは、多くのトップダウン型の計画活動と相反するため、最初は奇妙に感じるでしょう。しかし、SREが**ダイナミックにチームを再編成**するとき、それは通常、成功の兆しであると言えます。

4. 信頼性マインドセットのレベルと組織内の目標を評価する

SREを高いレベルで導入するには、想像以上に長い時間がかかります。**Googleでは**、製品の信頼性を戦略的なレベルにまで高めるには、3年から5年かかると考えられています。このレベルを維持するための絶え間ない努力を考えると、古い習慣に逆戻りすることもよくあることです。そのため、この新しい考え方を継続的に評価し、調整するために時間とエネルギーを費やす必要があります。

SREの手入れと餌やり

SREを開始した後は、生まれたばかりの組織を世話し、養っていきたいと思うでしょう。SRE の実践が発展するにつれ、以下のことを考慮する必要があります。

足場となるチームをより大きな組織に成長させる

最大の問題や、誰もが手をつけることを恐れているような企業の巨大な核となるモノリスから始めるべきではありません。チームや理念、実践をサポートする環境を構築するためには、ある程度の素早い勝利が必要です。逆に、おもちゃのようなサービスから始めてはいけません。SREは、重要な信頼性のニーズがある場合にのみ価値があります。一度足場を固めたら、安全に拡張するために継続的に学習する必要があります。重要度の低いサービスを大量に引き受けることは魅力的に見えるかもしれませんが、その誘惑には負けないでください。SREの価値は、信頼性の高いサービスにあります。その他のサービスは、「あなたが作るなら、あなたが動かすべし」モデルに従ってください。

SREの組織構造：独立したSRE組織と組み込みチームの比較

Googleは、SRE専門の組織を創設してから常にその組織を維持し、そうすることには信頼性の文化、リリースの優先順位付け、人材採用など、かなりの利点があると考えています。しばしば、「サイロを壊す」というDevOpsのアプローチと比較するよう求められることがあります。それぞれのSREの管理チェーンは、決してサイロであってはならないということを理解することが重要です。SREは、組み込み型の個人からほんの触り程度のコンサルタントまで、様々な方法で開発チームと連携できます。とはいえ、専任の組織体制がなくてもSREの導入は成功するかもしれませんが、シニアリーダーの広範囲なサポートが必要になることは覚悟しておいてください。

昇進・研修・報酬

SREは開発者であり、少なくとも組織内の他の開発者と同等の報酬とインセンティブを期待すべきです。昇進率もまた、他のチームと同等かどうかを確認するための大切な指標です。給与と昇進率の両方を定期的に比較し、格差をなくす必要があります。この給与体系では、SRE を不当に扱うことができる（例: 長時間働かされる）と思われるようなものにしないようにしてください。また、SRE は有意義で影響のある仕事ができることがより強く期待されるようになることにも留意してください。

オンコールへ入ることは、恐ろしく、とても疲れる仕事なので、入念な準備とトレーニングが必要です。また、オンコールに対する報酬を有意義な方法でチームに提供することも重要です。直接的な報酬に制限がある場合は、間接的な方法（代休など）で工夫しましょう。

コミュニケーションとコミュニティーの構築

SREの育成には、正式なトレーニングコース、社内技術発表会、読書会など、さまざまな活動が含まれます。その多くは、実験のために時間やリソースを提供することで行われる間接的な仕事になります（例: 20%ルール）。コミュニティの構築には、自律性と権限委譲が重要であり、これは（受動的ではなく）能動的なリーダーシップによって行われる必要があります。つまり、明確なリーダーシップのビジョンまたは目指すべき目標を設定し、組織内で目に見える形でエンパワーメントのロールモデルを示すことです。どのような種類の変革においても、コミュニケーションの量を過小評価しがちですし、SRE は本心からではないメッセージを見抜くことにも長けています。

SRE導入の効果を測る

SLO、SLI、エラーバジェット、ダッシュボードなど、SREを導入する過程で多くの成果物を取得することはよくあることです。これらはすべて組織の代理指標ですが、信頼性がどのように変化しているかを常に全体的に把握できるわけではありません。そのためには、もっと型にはまらない視点を検討する必要があるかもしれません。純粋に物事がうまくいっていれば、好循環は時間とともにかなり落ち着いてくるはずです。次から次へと発生するインシデントの火消しをしているというよりも、積極的に火事を予防しているという感覚がでてきます。これは特に、忙しさによって自らの価値を示すことに慣れきっている組織にとっては、不安なことかもしれません。このとき、忙しさを取り戻すために戦術的な最適化を行おうという誘惑に負けないようにしましょう。SREは、失敗を経験し、能力を高めるにつれて、自然にSLOとエラーバジェットを改善し始めるでしょう。

船の舵取り

より積極的なアプローチに移行することで、より多くの時間をビジョンに費やすことができるようになります。組織内のさまざまなサービスが実際に必要とする信頼性レベルを正確に把握することができるようになります。このデータを使って最適化する場所を決定し、新たな期待成果を設定します。例えば、社内システムの一部はビジネスクリティカルと印が付けられていましたが、SREはそのシステムが99.9%のSLOで十分なことを知っています。他のシステムにはより高い信頼性が必要かもしれません。SREが成功したかどうかを判断する確実な方法は、他のチームがSREの恩恵を受けることに興味を持ち始めたときです。

第6章
Googleを超えて

本レポートで紹介した視点の締めくくりとして、過去数年間に様々な形でSREを導入してきた異なる業界のSREリーダー 3名に話を聞きました。それぞれの業界や組織でSREをうまく機能させるための洞察に加え、導入がどのように機能したか、何が違っていたのかについて、ユニークなストーリーを語ってくれました。

ヘルスケア // Joseph

Joseph Bironas氏は、GoogleのSREリーダー時代から、いくつかの医療機関でSRE導入を主導してきました。そのため、医療分野へのSRE の導入が他のハイテク企業やスタートアップ企業の文化とはどのように異なるのか、業界からの視点を提供することができました。生命に関わる業務の性質上、信頼性は多くの場合において最重要事項です。しかし、ヘルスケア業界は、組織モデル、文化、予算、規制要件など、さまざまな特殊な課題に直面しています。

医療機器製造やFDA（アメリカ食品医薬品局）規制分野など、非常に厳しい条件での取引を行う企業で働いた後、Josephは、信頼性は要件として理解されてはいるものの、SREの費用対効果はこの業界ではほとんど理解されていないことに気がつきました。その結果、SREやインフラチームは、ITコストセンターに引きずり込まれ、その範囲が大幅に拡大する「なんでも屋エンジニアリング」になってしまうことがあるのです。

SREチームがITコストセンターの下で管理されることの何が問題なのかと思われるかもしれません。ITILのような広範なITフレームワークにしたがって管理をしている企業では、ITILの単なる一部分であるSREに価値を見出すことは難しく、例えばハー

ドウェア調達のような、SREが意見を言う必要のないようなことも扱われます。さらに言えば、社内ITと事業用ITのすべてを管理するCIOは、ソフトウェアシステムの信頼性を判断するのに最適な立場にはないのです。むしろ、ソフトウェアに特化したリーダー、例えばエンジニアリング担当の上級副社長やCTOに任せるほうが理にかなっています。

この業界の組織は、DevOpsのプラクティスをまだ導入していないのにSREの導入を希望するという急勾配に直面することがよくあります。例えば、彼らは月に一度ソフトウェアをリリースしており、規制や組織全体のコンプライアンス管理に関する避けることのできない複雑さのために、CI/CD の自動化はほとんど行っていませんでした。多くの医療機関は、単純に、迅速にデプロイしたくないのです。一部の顧客にとって、あまりに迅速にデプロイすることは、不十分なテストや不十分な安全性を意味するからです。

SREへの移行など、変革への意欲は業界内でも大きく異なりますが、これはおそらくリーダーシップの優先順位やスタイルが異なるためでしょう。Josephは、あるチームが現場にデザイナーを送り込み、要件を収集し、新しいワークフローを構築し、より良い製品によってケアに革命を起こすことができたというシナリオを紹介しました。別のシナリオでは、あるチームは、現在よりも良くなることだけによって動機づけされ、同じレベルの投資は必要ありませんでした。また、別のシナリオでは、あるチームは惰性に悩まされ、変革や投資を行う前にトップダウンの命令を待っていました。Josephの経験によると、より進歩的なリーダーは、信頼性に対する顧客の要求に敏感である傾向があるようです。

スタートアップの文化とは対照的に、これらのチームの中には、変革が非常に遅いところもあります。あるチームは、18か月で「何か」（例えばSREの導入）を達成できるか懐疑的でした。これは、スタートアップ企業にとっては永遠に感じる時間です。このようなペースで組織に大きな変化をもたらすことを考える場合、計画された投資対効果を理解するためのモデルが必要です。J カーブ（**ルーフショットとムーンショット**を参照）について知っておくことは、本当のリターンを得る前に、その谷間で努力を放棄することを避けるために重要です。Josephは、四半期ごとにチームと進捗状況を確認することを推奨しています。彼は、SREへの移行をインシデント対応から始め、インシデントレビューで継続的な学習のサイクルを構築してから（例えば6か月間）、SLOに集中することを推奨しています。密かに失敗するわけにはいかない「真の投資」を行うには、経

営陣のスポンサーシップを求めたり、トップレベルのOKRを導入したり、組織の中での努力を「本物」にするためのあらゆることに注力するとよいでしょう。また、このサイクルから学ぶだけでなく、学んだことを実行に移すことも重要です。

ヘルスケア業界でよくあるもう一つの間違いは、チームが従来の運用業務にどっぷりと浸かっていて「IT消費のXX%をソフトウェアで削減できるなんて、想像もつかないですよね。」と考えている場合に、「SREのソフトウェア面」を見落としてしまうことです。このSREの中核となる価値は、多くの場合、リーダーにとってなじみのない概念であり、一部の凝り固まったシステム管理者や運用担当者によって意図的に抵抗されたり、追いやられたりすることさえあります。この点を無視すると、SREが非常に非効率に見えてしまいます。ソフトウェアエンジニアリングはまた、困難で高価です。たとえSREに関連した商用ツールを購入したとしても（長年にわたる膨大な数の貢献者にもかかわらず不完全です）、統合作業から逃れることはできません。そしてそうした統合作業は主にソフトウェアエンジニアリングの努力によるものです。

信頼性のための予算も問題になることがあります。Josephは、「この業界には（SREを構築した当時の Google のような）広告収益曲線がない」と指摘しています。このことは、Googleが行ったように専門化して投資する能力に影響を与え、商用ソリューションに依存するようになります。ビジネスの予算編成や計画はまだウォーターフォール方式であることが多く、SREの仕事には困難が伴います。新しいソリューションを探求し、理解し、設計するために必要な時間は、ウォーターフォール方式の仕事の進め方にはあまり適していません。

その結果、これらのことはあらゆる業界にも当てはまりうることがわかったそうです。Josephは、不完全な努力でさえ価値のある出発点になることがあることを示す話を共有しました。彼が関わったある企業の場合、経営陣はエラーバジェットを劇的にシンプルにすることを望んでいました。クリティカルユーザージャーニー（CUJ）に適したSLOを選択する代わりに、すべてに対してひとつのSLO（可用性99.95%）を宣言したのです。この目標はわかりやすかったのですが、エンジニアリングチーム全体のSLOの概念を麻痺させるものでした。ステートフルなアプリケーションもステートレスなアプリケーションも、バッチもリアルタイムも、すべて同じSLOを持つことになり、結局役に立たず、この手法への信頼も損なわれてしまいました。その結果、エラーバジェットが意味をなさなくなり、エラーバジェットを使おうとしたプロセスも同様に機能不全に陥りました。

しかし、結局、今まで測っていなかったものを測るようになったという事実に価値があるのです。SLOは、今まで疑問に思っていなかったことをお互いに問いかけるきっかけになりました。これは、目の前にあるデータに基づいて、お互いに話し合い、良い決断を下す手助けをすることが重要であることを物語っています。

小売業 // KipとRandy

The Home Depot（THD）のCommodore "Kip" Primous氏とRandall Lee氏は、大規模小売業者がSREを導入した方法、その成功例、および現在の課題について洞察を述べました。THDはGoogle Cloud Platform（GCP）の初期の大口顧客で、最近出版されたSREの本に詳述されている原則に従ってSREを採用することが、彼らのクラウド導入の一部でした。6年後、彼らが構築しようと期待していたことと、現在の状況は全く異なっています。

Kipは、「ドットコム」事業部の信頼性エンジニアリング（RE）マネージャーとして入社し、THDのeコマースサイトの「ブラウズスタック」を担当しました。RandyはKipより先にTHDに入社し、より良いプラットフォームを通じてレジリエンスを向上させる、というSREの共通目的を共有していました。当初は自社でクラウドとデータセンターを構築することを検討していましたが、様々なクラウドサービスプロバイダーを評価し、GCPにたどり着きました。クラウドへの移行を成功させるには、SREや"DevOps 2.0"を導入することを通して、自分たちの仕事のやり方も同時に変えるしかありませんでした。

元々、THDの目標は、巨大でモノリシックなコマースサービスからの脱却でしたた。プロジェクト オーロラは、THDの副社長が資金提供を含めて直接管掌し、規模の経済の達成、運用チームの縮小、契約社員数百人を少数の正社員へ転換することを目指しました。また、信頼性を向上させ、組織内の他のチーム（連携がうまくいっていない可能性がある）への依存を減らすという一般的な意図もありました。このドットコム・チームは、"インターネット・スピード"で運用することを望んでいました。

連携が重要でした。クラウドに移行する前は、すべてのデプロイがスペースシャトルの打ち上げのようなもので、何年もの努力の上に多くの調整が必要でした。THD/dot-comの中では、現在のDevOpsのモデルが順調に進んでいるとチームは感じていました。信頼性エンジニアリングを導入することで、チームは新しいプラットフォームと新

しい権限で、明確な作業の新しいパターンのに従うことができ、スタック内の信頼性に関連するあらゆる作業を行う権限を得たと感じました。クラウドネイティブエンジニアを多く採用し、可能な限り自動化しました。既存のDevOpsチームを制限していた境界線を飛び越えることができたのです。

2015年から2017年まで、SREは新しいクラウドインフラで、これまでと異なる最新のツールやハードウェアを使って作業していたため、迅速かつ独立して動くことができました。そして2018年、エンタープライズのチームが追いつき、SREはGCPに取り組む唯一のチームではなくなりました。中央集権的なエンタープライズのチームが従来のモデルを更新したことで、誰もが安堵し、両者は歩み寄ることができました。例えば、エフェメラルな仮想マシンの新しい世界では、個々のマシンのパッチを追跡する必要がないことを合意することなどです。、両チームが建設的な話し合いを重ねることで、ドットコムの信頼性エンジニアリングチームは新しく設立された集中管理型のエンタープライズチームと協力し、より大企業に適したプロセスを確立して、会社のセキュリティガイドラインをより適切に遵守できるようになりました。さらに、GCPプラットフォーム管理の大部分（課金、権限管理、クォータ管理など）を、信頼性エンジニアリングチームからエンタープライズチームに移管することができたのです。

THDがSREの道を歩む中で、KipとRandyはいくつかのパターンと、他の業界にも通じるであろう教訓を得ました。他のチームにSREの概念を採用してもらうプロセスは何年もかかりました。コンプライアンスの自動化の改善し、コストの改善、アクセス制御、そしてサイバーセキュリティの順番で推進を繰り返したのです。その都度、多くの議論と教育が必要でした。障害やダウンタイムがほとんどない静かな時期には、社外で発生する事象から切迫感を煽られることがあります。Equifaxの障害、AkamaiやFacebookのDNSの問題などが発生すると、誰もが慌てふためき、信頼性向上のための新たなサイクルが始まるかもしれません。

THD社内でSREの導入を成功させるためには、経営陣のスポンサーシップが不可欠でした。SREモデルによるドットコムのクラウドへの移行が最初に成功した後、SREの役割は社内で高パフォーマンスの同義語になりました。他の多くのチームもこのモデルをコピーしようとし、中にはSLOの要件に関係なくSREの導入を強制されたチームもありました。しかし、すべてのチームがドットコム チームのように幸運ではなく、まっさらな分野でクラウドネイティブにスタートできたわけではありません。そのため、チームは信頼性エンジニアリングチームがもたらす価値を認識するのに苦労し、役割と責任

に対する期待を取り違えることもありました。このような曖昧さがあると、ある組織の「非公式」な信頼性エンジニアと交流したときに問題が発生します。この信頼性エンジニアは、元のチームと同じレベル、同じ原則に則って仕事をしていないかもしれません。例えば、トイルを自動化する計画もなく「ただボタンを押すだけ」の人もいます。このような経験はそのチームに良くない後味を残し、将来他の信頼性エンジニアリングチームと一緒に仕事をすることに興味をなくしてしまいます。

また、Kipは、数年ごとにSREに触発された新しい取り組みがなければ、信頼性水準が低下すると警告しています。信頼性エンジニアリングは壁を壊しましたが、その壁は再び築かれつつあります。「信頼性は私の問題ではない、信頼性エンジニアの問題だ!」と考えるチームがいますが、これは間違ったメッセージです。Randyは、うまく機能している信頼性エンジニアリングチームは、明確に定義された役割と責任を明確に加えて、信頼性エンジニアリングの実践と原則の継続的な強化および教育がなければ、後退してしまう可能性がある、と付け加えています。

現在、THDは信頼性エンジニアを「さらに強化」する状況にありますが、SREの原則に従わない変更は、実はアンチパターンとなりえます。SREはあらゆる問題に適用できる万能薬ではありませんが、チームがSREで成功を収めることは難しいことですし、SREをすべての場所へ適用したくもないはずです。最近、Kipは配送センターにあるベンダーがサポートする物理ハードウェアで SRE を実践するように言われましたが、これは SRE に自然に適合するものではありません。これらのシステムの信頼性を向上させる機会は常にありますが、非クラウドネイティブの環境で信頼性エンジニアリングの多くの手法を適用することはより困難です。おそらく、ビジネスの一部の領域では、SREではなく、バリューストリームマッピングやリーン生産方式などのプラクティスがより効果的な方法であると思われます。このような力学を避けるためには、SREを「プッシュ」モデルではなく、「プル」モデルとして適用することがより理にかなっているかもしれません。つまりSREをチームに押し付けるのではなく、サービスとして提供するだけにとどめ、必要とするチームに来てもらうようにするのです。

KipとRandyの最大のアドバイスは、幹部教育に力を入れること、そしてチャンピオンの価値を認識することです。トップダウンのサポートがなければ、意味のある変革のための予算を確保することは困難です。製品開発チームから予算を得ようとすると、そのチームは「税金を払いたくない」という力学が働きます。税金を払うときには、SREが自分たちの製品とその目標に向かって、直接作業をすることだけを望んでいます。

THDでは、もともとドットコム以降の信頼性エンジニアリングチームの創設と成長を唱導したシニアリーダーがいました。現在、THDは、SREの原則を適用する能力の程度が異なる多くの信頼性エンジニアリングチームがさまざまなプロジェクトに取り組んでいるという奇妙な状態にあります。RandyとKipは、よりシニアな人物がいれば、THDの状況が改善されるだろうと期待しています。信頼性担当副社長がすべての信頼性エンジニアリングの役割を担えば、規模の経済が生まれるかもしれません。信頼性エンジニアリングの中心的な組織がなければ、SREの役割は、異なる組織のSREが全く異なることを行い、異なる基準や原則に従うというところまで変質してしまう可能性があるのです。

まとめ

このレポートによって、あなたの会社がどのようにSREを導入するのか、どこに課題があるのか、そのヒントが得られることを願っています。SREの原則を明確に定義し、それを実践と能力に結びつけ、チーム内での成長と育成を優先させることで、成功の可能性はより高くなると考えています。また、企業内でSREを実践するプロセスを経験したチームの例や、彼らが直面し克服した具体的な課題も紹介しました。

このレポートが皆様のSRE導入の一助となり、すべての人にとってより信頼性の高いテクノロジー体験につながることを期待しています。そして、この採用を通じて、運用チームがより持続可能になり、サービスがよりスケーラブルになり、そして開発速度が向上することを願っています。

「クエリは流れ、ページャーは沈黙を守らんことを。」

著者について

James Brookbankは、Googleのクラウドソリューションアーキテクトです。ソリューションアーキテクトは、複雑な技術的問題を解決し、専門的なアーキテクチャのガイダンスを提供することで、Googleのお客様がクラウドをより簡単に利用できるよう支援します。Googleに入社する以前は、ITインフラストラクチャと金融サービスを中心に、数多くの大企業に勤務していました。

Steve McGheeはリライアビリティアドボケイトであり、世界クラスの信頼性の高いサービスを構築・運用するための最良の方法をチームが理解できるよう支援しています。それ以前は、GoogleのSREとして10年以上、検索、YouTube、Android、クラウドなどのグローバルシステムの拡張方法を学びました。カリフォルニア、日本、英国で複数のエンジニアリングチームをマネージメントしていました。また、カリフォルニアに拠点を置く企業で、クラウドへの移行を支援した時期もありました。