DevOps を付け焼き刃にしないために

Andrew Agerbak、Kaj Burchardi、Steven Kok、Fabrice Lebegue、Christian Schm

アジャイル型ソフトウェア開発への移行を目指す多くの企業は、まるで「何カ月もかけて理想の芝の種を見つけ、何の手入れもしていない古い裏庭にその種を植えた」かのような状態に陥っている。長い時間をかけたにもかかわらず、庭の芝生は以前と比べ多少は見栄えが良くなった程度だ。期待した成果を得るためには、種を植える前に土中の木の根をすべて取り除き、土を入れ換え、水やりの方法を見直さなければならなかったのだ。

金融やヘルスケア、製造、消費財といった業界の大手企業の多くが、アジャイルを試みる必要性を感じており、そしてアジャイルから得られるメリットに大いに期待している。確かに、経営層が開発部門にアジャイルに取り組むように命じさえすれば、重要なソフトウェア開発が優先的に行われるようになり、アジャイルの特徴である短期間の開発サイクルによって、迅速な変更が可能となるはずだ。だが、そのような効果が即座に表れるとは限らないのが実態であり、企業は落胆する。新たにアジャイル開発を試した企業は、何かが不足しているのだろうか、と悩むことになる。

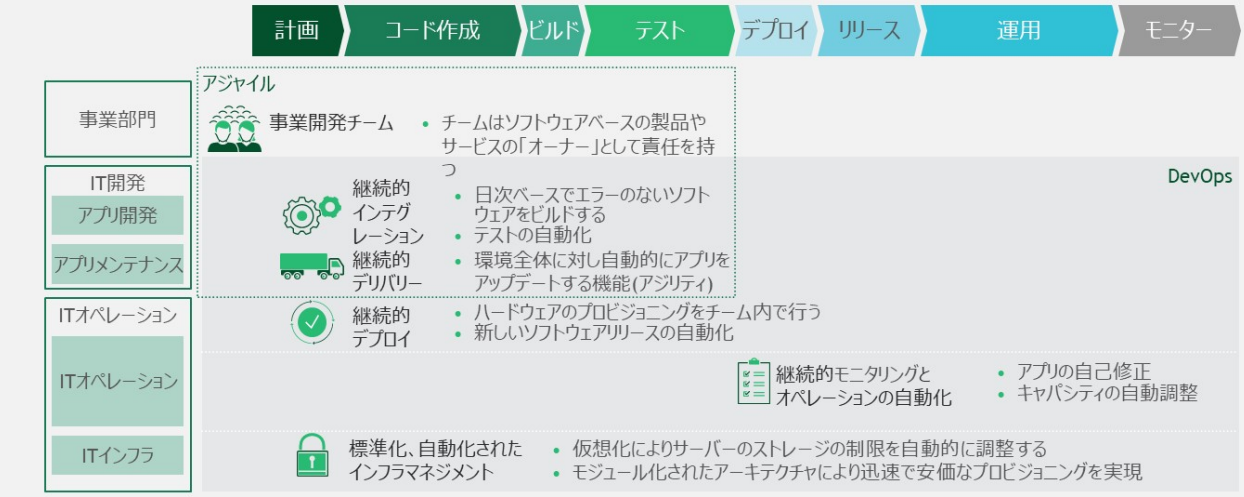
多くの場合、実際に何かが足りない。アジャイルは、ソフトウェア開発プロセスの初期段階から組織のカベを取り払うという点で優れているが、それだけで実現できることには限界がある。デジタル時代の勝者となるためには、ソフトウェア開発のライフサイクル全体も再考しなければならない。つまり、アジャイルだけではなく、DevOps だ。

DevOps は、テストやデプロイ（実装）計画など、非常に重要でありながら、従来のソフトウェア開発手法では後期段階になってからしか登場しない工程をコード作成の一環とすることができるアプローチである（図表 1）。複数の作業を同時進行させ、かつマルチファンクショナルなチームでの協業を重視する DevOps は、計画、コード作成、テスト、デプロイの各工程をそれぞれ別のチームが担当する従来型の手法、「ウォーターフォール型」モデルから脱却するものだ。

多くのソフトウェア企業では、DevOps の特徴でもある、継続的なソフトウェア開発とリリースを何らかの形で採用しているが、テクノロジーやインターネットサービス以外の業界では、このアプローチ（今も進化を続けており、その手法は最近では DevOps 2.0 または BizDevOps とも呼ばれる）はまだ

まだ新しい存在だ。それでも、特に金融業界では、伝統ある企業がテスト自動化やハードウェアの迅速なプロビジョニング（実運用に向けた準備、設定）など、DevOps の手法をいくつか導入している（BCG 論考「Leaner, Faster, and Better with DevOps（2017 年 3 月）」をご参照ください）。とはいえ、その取り組みの範囲はまだ限定的であり、DevOps のメリットを全面的に享受できるような、全社レベルでの変革を成し遂げているわけではない。

図表1: ソフトウェア開発サイクルにおけるアジャイルとDevOps



出所: BCG分析

DevOps を活用する

企業が DevOps を最大限に活用するには、コントロールとガバナンス、そして IT 組織の構造とオペレーティングモデルを変革する必要がある。

コントロールとガバナンスを再定義する　大手企業におけるソフトウェア開発は、品質を保証し、重大なエラーを防ぐために定められたコントロールガイドラインに沿って行われる。このガイドラインは、導入当時は有効であったかもしれないが、その後の変化には対応できていない。技術革新のペースが加速したいま、旧来のコントロール手法はもはや適切ではなく、障害である。

たとえば、開発チームが作業を開始する前に必要とするインフラのプロビジョニングについてのコントロールガイドラインが、仮想化が一般的となる以前に導入されたものである場合がこれにあたる。今では仮想化によって、はるかに低いコストで、よりシンプルに演算能力とストレージを利用できるにもかかわらず、企業によっては、承認を得てハードウェアのプロビジョニングをするために、依然として 1 カ月という時間と 50 通以上のメールでのやりとりを必要としている。

たしかに、新しいソフトウェアサービスをリリースする際に複数の承認が必要なケースもある。複数の機能が一つのシステムとして構成されている状況下で、ソフトウェアの更新頻度が年に数回しかないのであれば、重要な機能に関連するリリースについてはそうするべきかもしれない。だが現在、ユーザーが見る画面の色の変更などのわずかな調整に 20 人もの承認が必要であるはずがない。

ソフトウェアが、急速に変化する企業や顧客のニーズに対応する重要な手段であるいま、時代遅れのコントロールにより生じる遅延は、企業に重大な競争上の不利益をもたらす。仮に企業がサイバー攻撃の標的となれば、このような遅れは風評リスクの引き金となりかねず、ひいては企業の存続

を脅かすリスクにもなりうる（BCG 論考「Develop a Cybersecurity Strategy as If Your Organization's Existence Depends on It（2017 年 10 月）」をご参照ください）。

DevOps への移行において再考が必要なもう 1 つの分野はガバナンスだ。これには、予算編成に新たなアプローチを採用することも含まれる。アジャイルでは、予算はプロジェクト単位（定められた期間と、あらかじめ定義した一連の成果物に対して）で割り当てられるものではない。予算は、たとえば銀行なら資産運用口座の管理機能、小売企業なら受注配送システムなどといった、月単位、あるいは年単位で、いくつものチームが注力して取り組むべき重要な「プロダクト」に割り当てられる。DevOps では、このプロダクト単位の予算の枠組みに、アプリケーションのメンテナンスやインフラコストの一部を組み込む必要があるため、複雑さが増す。

クラウドソリューションに関する決定権も改めなければならない。過去には、意思決定権限が IT 部門にあることに異議を唱える者はいなかった。しかし、事業部門がクラウドベースのツールを使用して直接ソフトウェアの開発を行うケースが増えるにつれて、状況は変わりつつある。

ある企業では、デジタル製品開発チームが AWS（アマゾンウェブサービス）アカウントへの直接アクセスを求めたところ、標準化やセキュリティ上の懸念から IT 運用グループが抵抗したことで、意思決定権に関する疑問が浮上した。実のところ、この企業を含め、どの企業においても「決定権を誰が持つべきか」という疑問に対する唯一の正しい答えはない。だがこれは、ソフトウェア開発におけるさまざまな境界線を引き直す DevOps では、必ず検討することになる課題だ。

また、DevOps は、ソフトウェアのバグに対する考え方にも変化をもたらす。DevOps がまだ十分に浸透していない企業では、「開発者がオーナー」という概念はない。リリースしたソフトウェアに問題があった場合、IT サポートチームはチケットシステム（ServiceNow など）を通じて報告し、アプリケーション保守チームがチケットの管理者となる。しかし、コードの当初の開発者が問題を耳にすれば、問題の部分を修正しようとする場合もあるだろう。その結果、開発チームと保守チームの双方が同時に同じソフトウェアの修正に取り組むことになり、コードに乖離が生じたり、統合に追加の工数を要したり、脆弱性が生まれたりする可能性がでてくる。対照的に、DevOps の手法を実践している企業では、実装されたソフトウェアの問題は自動的に開発チームのバックログに登録され、開発チームが修正を行うことになる。この際、開発チーム以外が修正を検討することはなく、複数のチームが同じコードの修正を別々の目的で実施しようとするなどの混乱が生じる可能性もない。

究極的には、DevOps 実施後のガバナンスの良し悪しはリリースまでの期間、つまりサイクルタイムで判断できる。企業があるソフトウェアを企画してから、高品質、高信頼性のプロダクトをリリースするまでの時間を大幅に短縮できていれば、その企業の新しいガバナンスプロセスがしっかり機能していると判断できる。そして、その企業は今の時代に必要なケイパビリティを手にしたともいえる。

CIO と IT 部門の役割を再定義する　DevOps の導入を成功させるには、最高情報責任者（CIO）と IT 部門の役割を、微調整したり、ある部分は大幅に変更したりする必要がある。アジャイルモデルを取り入れる企業では、新しいソフトウェアの仕様設計や開発が、暗黙的に事業部門の責任となる。これにより、コードの一行一行に関する CIO の責任は軽くなるかもしれないが、それでも大半の場合、品質に関する責任のほとんどは CIO の肩にかかっている。なぜなら、CIO は引き続きソフトウェア開発ができる人材を採用・育成しなければならないからだ。また、開発人材がその能力を最大限に発揮できる運用環境（ガイドライン、サービス、プロセス、ツール、インフラ）を含む、より優れたデリバリ

ーモデルを導入する責任もある。さらに、ソフトウェア開発ライフサイクルにおいて、「フロントローディング」、すなわちより多くのタスクをより早いステージにて実施できるようにしなければならない。

ソフトウェア開発のライフサイクル上の多様な活動の工程を前倒し（左から右への時間軸を取った場合、左にシフト）するフロントローディングは、専門用語で「シフトレフト」と呼ばれる。DevOps を導入した場合、IT 運用部門に所属し、従来は後期工程を担当していたスタッフが、プロダクト開発チームへ異動し、コードの書き方についても意見を述べる立場となることもある。また、社内のデータアーキテクチャやサイバーセキュリティの責任者には、より早い段階でインプットが求められるようになる。このような工程、そして知見のシフトレフトは、作成されるすべてのコード（多くの場合複数の部署が関わる）を、素早く、必要なセキュリティレベルを満たした状態で市場に投入する 1 つの方法である。

DevOps に関連して特に重要な CIO の責任は、最適なインフラ環境の導入である。ビジネス志向の開発チームは、互換性を考える必要のない、インフラ非依存型のプラットフォームを必要としている。DevOps では、重要な責務として、IT 組織がこれを管理し、アプリケーション開発ツールキットを提供する。

ビデオストリーミングサービスを提供するグローバル企業のネットフリックス（Netflix）のケースは、DevOps の導入でどのようなメリットが得られるかを示す一例だ。ネットフリックスは、新たにリリースされるソフトウェアのパフォーマンス最大化と、ソフトウェア開発チームの効率性向上を任務とするセントラルエンジニアリングオペレーショングループ（一種の専門 IT チーム）が、DevOps を通じて得られるメリットを引き出している。

ネットフリックスの開発者は、共通のツール、サービス、インフラストラクチャ管理機能（ネットフリックスはこれらを「paved road（舗装された道路）」と呼ぶ）のおかげで、通常であればデコボコだらけの道を進まなければならないソフトウェアの新規開発をスムーズに進められる。「paved road」とエンジニアリングオペレーショングループによって、作成したコード（信頼性が高く、安全性も検証されている）を数分のうちに複数地域にリリースすることができる。

このように、ソフトウェアのデプロイまでのプロセスを円滑にし、そして加速させるには、IT 人材がこれまで必要とされなかった能力を習得する必要がある。たとえばエンタープライズアーキテクトは、IT アーキテクチャ戦略の策定、特にプラットフォームの選択に関して、一層深く取り組む必要がある。また、IT 組織は、マイクロサービスやコンテナといった手法を取り入れて、コード担当者たちが、再利用可能なソフトウェアを使って、よりスピーディーな開発ができるように、サポートする必要がある（コラム「DevOps を支えるテクノロジー」をご参照ください）。

また、IT 組織には、新しいスキルとそれを担う人材が必要だ。たとえば、品質について責任を持つクオリティエンジニアを雇うか、育成しなければならない。こうしたエンジニアをソフトウェア開発チームに組み込み、プロセスの早い段階で厳格なテストを実施する必要がある。IT 運用チームについても同様に、SRE[1]や IaaS[2]環境構築といった、新たな専門知識をもった人材を獲得するか育成する必要がある。このようなスキルがなければ継続的なデリバリーや継続的インテグレーション（コラムをご

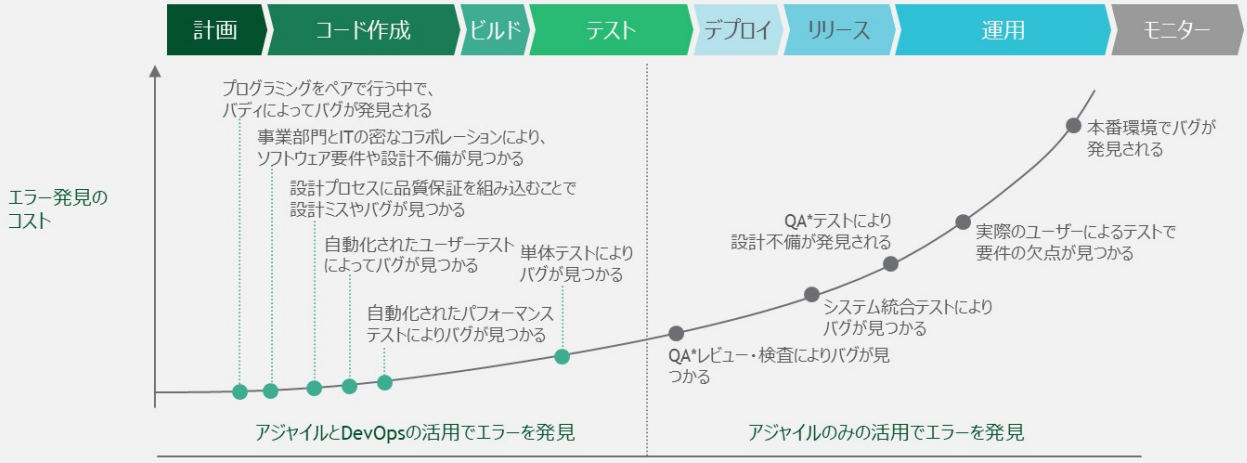
[1] Site Reliability Engineering、IT サービスの信頼性を向上させるためにプログラミングを駆使し、運用を行う手法
[2] Infrastructure as a Service

参照ください）は不可能であり、スピードと信頼性というアジャイルのメリットを享受するのは困難になる。

自動化によってオペレーティングモデルを作り直す　チームが機能を追加したり、危険なセキュリティホールを解消したりするにあたって、数週間、数カ月間かかる何重もの承認プロセスを経るのではなく、比較的緩やかな監督下で、（理想を言えば）数クリックでそれらを行うことができれば、そのメリットは極めて大きい。DevOps の柱の 1 つである自動化が、これを可能にする。とはいえ、自動化を取り巻く意思決定は複雑であり、企業が足掛かりを得るまでに時間を要することも少なくない。このため、自動化をいつ、どのように導入するかは、DevOps に移行するすべての企業にとって重要な判断事項となる。

導入は、テストの自動化から始めるのが好ましい。私たちの経験では、自動化されたテストの対象に多くの新しいコードが含まれることによるメリット自体が、DevOps への移行を後押しする。開発後期段階で発生する遅れとそれによって生じる追加コストという、開発において常に存在するリスクについて考えてみよう。従来のウォーターフォール型では、そして時にはアジャイルの場合でも、テストはコードの完成後に実施される。結果的に、コードをこれからリリースしようという段階で、重大な問題が発見される可能性があるのだ。対照的に、DevOps の枠組みでは、コードは細かい反復的プロセスと、高頻度のテストを通じて開発されるため、デプロイの直前にコードの問題点が見つかる可能性は低くなる（図表 2）。

図表2: DevOpsでは、バグは初期段階で発見され、修復コストが低い



注: QA= Quality Assurance(品質保証)
出所: Puppet「2016 State of DevOps Report」; BCG分析

テストの自動化は価値を生む取り組みだが、展開は段階的に行わねばならない。企業は、アジャイルへの移行をすでに始めている領域から導入すべきだ。そこで成功体験を積んだのち、自動化の範囲をより多くのコードに拡大するとよい。

Google など、デジタルサービスを主導する企業の中には、99%を大きく上回る信頼性を目標に掲げているところもある。これは、自動化されたテストを活用した場合でも、自分たちがリリースするソフトウェアが即座に、そしてすべての環境下で機能することを前提としているためだ。当然だが、Google は迅速なソフトウェアリリースとサービス改善を可能にしようと構築された企業である。デジタルネイティブでない企業が同じレベルの信頼性を目指す必要はないが、デジタルサービスに関して

は、Google や他のデジタル先進企業から学ぶことができるし、学ばねばならない。自社の障害が許されない業務において、必要不可欠なデジタルサービスを提供しようとする際には――たとえば銀行がスマホ支払い機能を提供するときなど――Google の基準に満たないようなソフトウェアをリリースしている場合ではないのだ。

大企業におけるレガシーシステム、たとえば給与計算や ERP といったシステムについては、その取り組みは必然的にやや異なるものとなる。レガシーシステムでも自動テストを行えるし、多くの場合、すでに実装している。しかし、アジャイル開発チームが期待するようなハイペースでのリリースサイクルに適応するには、テストはデータ処理や転送などを含むバッチ処理と実行時間帯を同期させて行うべきである。バッチ処理は夜間実行されるように設計されることが多いため、IT 組織は自動テストも夜間に実行するよう検討すべきだろう。

段階的に始めよう

企業は、現状のソフトウェア開発の慣行を無視して DevOps に全面移行するわけにはいかない。DevOps への移行には非常に多くの変更とトレーニングが必要なため、既存のシステムやプロダクトに大きな影響を及ぼしかねない。したがって、DevOps は段階的に導入するべきだ。

最初のステップは、他のアプリケーションへの依存度が低い、たとえばメーカーであれば調達ポータル、銀行であれば預金プラットフォームといったアプリケーションを対象にパイロットプロジェクトを実行して、DevOps モデルの学習と実践を重ね、そのモデルを調整することだろう。

パイロットでは、開発チームが、コードリポジトリ（コラムをご参照ください）の確立や、自動テストを導入するためのフレームワークなどの技術面のプランニングを進める。これが完成すれば、継続的インテグレーションと継続的デリバリーを開始できる。これらのプロセスにより、開発チームはバグや機能の不具合を手動でチェックする必要はなくなり、コードを書くことに集中できるようになる。

複雑なレガシーシステムを持つ企業の場合、継続的インテグレーションと継続的デリバリーはそれぞれ別個のフェーズとなる。対照的に、デジタルネイティブ企業の場合は、どちらもソフトウェア開発の中核であり、分けて考えるようなことはしない。すでに、ソフトウェアのリリースプロセスを強化する別の方法も検討しているかもしれない。だからこそ、デジタル先進企業の開発者は、コードを修正し、デプロイするのにかかる期間を数日や数時間、時には数分と見積もるのだ。

DevOps のパイロットを無期限に続ける必要はない。半年以内にはメリットが見えてくるはずだ。メリットは通常、1 週間あたりのソフトウェアリリース数の増加によって実現できるスピード／アジリティ（俊敏性）や、テスト範囲の拡大に伴う品質向上、手戻りに伴うコストの減少、自動化プロセス数の増加による効率性の向上という形で現れる。このような導入期間を経て、企業は DevOps の手法を別のソフトウェアやインフラのプラットフォーム、さらには別のテクノロジー領域に適用するためのロードマップを作成できるようになる。ロードマップには、使用すべき一連のツールと、DevOps を展開する順序を含める必要がある。

DevOps で顧客が求める機能を提供する

DevOps が必須となりつつある理由を示す、欧州の旅行会社の例を紹介する。

同社では、繁忙期のピーク中に、事業の中核である予約システムの価格変更ができなくなっていた。以前の更新時にシステムの脆弱性が明らかになったため、事業部門の責任者が、ピーク期間中のシステム改修を禁じる決定をしたのだ。

同社のような密なシステム結合環境におけるソフトウェアインフラや、それを前提とした開発運用方針は、決して珍しいものではない。しかし、このようなソフトウェア開発への慎重なアプローチによって、同社は 1 年でもっとも重要な時期に、競合他社の新価格や商品提案に対応できなくなってしまった。他社の旅行サイトでユカタン半島の人気リゾート地ベリーズへの 7 日間のツアーが突然 1,800 ドルに割引されたとしても、同社のサイトでは 2,100 ドルに据え置くしかなかったのだ。ダイナミックな価格変更を可能とするにはソフトウェアの修正が必要だが、同社のリリースプロセスが、変更できるタイミングに制限をかけていた。

ソフトウェア開発プロセスが事業にダメージを与えていることを認識した同社は、継続的インテグレーションを含む DevOps の手法を取り入れた。結果、ソフトウェアリリースの品質とシステム全体のレジリエンスが向上し、経営陣は繁忙期でもシステム改修を可能とする方針変更をした。その後、同社は業界のピークシーズン中、競合他社の動きにそれまでよりはるかに機敏に対応できるようになった。

遅かれ早かれ、大半の企業が同じような状況に直面することになる。つまり、競合他社が、セキュリティや品質を損なわずに、より低いコスト、より速いペースで何かを実現する。結果的に、他の企業はその差を埋めるために何らかの行動をしなければならなくなる。

そのための一つの方法が DevOps だ。DevOps の導入は組織とプロセスの変革を伴うが、それ自体は大半の顧客からは見えないところで行われる。しかし、DevOps がもたらすメリットは顧客の期待に沿ったものだろう。顧客が求める機能を俊敏に提供できない企業には、チャンスは二度と訪れないかもしれない。

DevOps を支える技術的基盤

IT 組織は、DevOps を実現するためにさまざまなツールやアプローチについて深く理解する必要がある。その中でも、もっとも重要な 7 つの手法について紹介する。

コンテナ型仮想化　ひとつのコンピューティング環境から別のコンピューティング環境に移行する際に、ソフトウェアを確実に運用できるようにする仮想化の一種。運用のために必要なすべての要素を新しいコードに紐づけることによって、ソフトウェア開発チームが、オペレーティングシステムと基盤となるインフラの間にある違いを無視することができるようになる。コンテナ化を容易にするオープンソーステクノロジーとして、Docker と Kubernetes の 2 つが挙げられる。

マイクロサービス　開発者が、非常に細かいレベルでアプリの機能にアクセスできるようにするためのプログラミングアーキテクチャ。マイクロサービスは、大きく、複雑なアプリの継続的なデリバリーとデプロイを容易にする。

コードレポジトリ　アプリのソースコードを含むデータベース。コードレポジトリが一元化され、アクティブに管理された場合、コードの安定性を向上させ、バージョンコントロールの問題を回避することができる。コードレポジトリのために使われるオープンソースツールとしては、Bitbucket や GitLab がある。

継続的インテグレーション　単一のソースコードのマネジメントや、総合的なテストの自動化を促進する開発手法。高度に自動化された方法によって、開発者が 1 日に数回の頻度でコモンレポジトリにコードを追加することができるようになる。これにより、すべての開発チームがアプリの最新版を使うことができる。オープンソースには、GitLab CI や Jenkins がある。

継続的デリバリー　ソフトウェアをいつでもステージング環境（最終的なテストを行う、本番に似せた環境）に移動できるようにする、ビルド中のソフトウェアのための原則。ツールとしては Bamboo や Jenkins が挙げられる。

継続的デプロイ　テスト済みのソフトウェアをリリースするための作業。時にはボタンを押すだけの場合もある。余計な付帯作業を大幅に減らし、問題を素早く解決する非常に貴重なツールとなる。

原典：Going All In With DevOps

Andrew Agerbak
ボストン コンサルティング グループ（BCG） ロンドン・オフィス ディレクター

Kaj Burchardi
BCG Platinion アムステルダム・オフィス　マネージング・ディレクター

Steven Kok
BCG ロンドン・オフィス　プロジェクトリーダー

Fabrice Lebegue
BCG Platinion モントリオール・オフィス　マネージング・ディレクター

Christian Schm
BCG ミュンヘン・オフィス　プリンシパル

監訳

高部 陽平
BCG 東京オフィス　マネージング・ディレクター&パートナー

山形 佳史
BCG 東京オフィス　マネージング・ディレクター&パートナー

2020 年 12 月発行

ボストン コンサルティング グループ（BCG）

BCG は、ビジネスや社会のリーダーとともに戦略課題の解決や成長機会の実現に取り組んでいます。BCG は 1963 年に戦略コンサルティングのパイオニアとして創設されました。今日、BCG の支援領域は、変革の推進、組織力の向上、競争優位性構築、収益改善をはじめとしてクライアントのトランスフォーメーション全般に広がっています。
BCG のグローバルで多様性に富むチームは、産業や経営トピックに関する深い専門知識と企業変革を促進する洞察を有します。これらに加え、テクノロジー、デジタルベンチャー、パーパスなどの各領域の専門組織も活用し、クライアントの経営課題に対しソリューションを提供します。経営トップから現場に至るまで、BCG ならではの協働を通じてクライアント組織に大きなインパクトを生み出しています。
日本では、1966 年に世界第 2 の拠点として東京に、2003 年に名古屋、2020 年には大阪、京都にオフィスを設立しました。
https://www.bcg.com/ja-jp/default.aspx