ACCELERATE

# State of DevOps 2019

DORA
DEVOPS RESEARCH & ASSESSMENT

Google Cloud

スポンサー:

Pivotal
Deloitte.
sumo logic
redgate
cloudbees
armory
Datical
New Relic

# 目次

目 次



DORA DEVOPS RESEARCH & ASSESSMENT | Google Cloud

# エグゼクティブ サマリー

「Accelerate State of DevOps Report」（DevOps の状況に関する報告書）は、世界中の 31,000 人を超えるプロフェッショナルを対象とした 6 年間の調査とそのデータを表したものです。これは、この種の調査としては最も大規模かつ長期にわたるもので、高いパフォーマンスを推進するプラクティスや能力についての独自の視点を提示します。これらの調査結果を見ると、どのようなプラクティスがテクノロジー提供のパフォーマンスを卓越したレベルにまで高め、ビジネスの力強い成果をもたらすかがわかります。

本調査では、テクノロジーを開発、提供するための最も効果的かつ効率的な方法に関する知見をデータから導き出すために、厳密な統計学的手法を使用しています。たとえば、業界の基準と比較したチームのベンチマーク評価には、クラスタ分析を使用しました。これにより、チームがローパフォーマー、ミディアム パフォーマー、ハイ パフォーマー、エリート パフォーマーのいずれに属するかが一目でわかります。

さらに、予測分析の結果から、チームのソフトウェア デリバリー パフォーマンスを改善し、最終的にエリート パフォーマーになるために必要となる具体的な能力を特定できます。

今年は、情報検索のサポート、使いやすいデプロイ ツールチェーン、柔軟なアーキテクチャ / コードのメンテナンス性 / 可視

性の高いシステムによる技術的負債の低減といったイニシアチブを通じて、組織がエンジニアリングの生産性向上をどのように支援できるかについても調査しました。

本調査はこれまで、ソフトウェア開発およびデリバリーに関する業界標準の 4 つの主要指標[1] が技術変革における組織的パフォーマンスを推進していることを示してきました。今年のレポートでは、速度を犠牲にすることなく安定性を向上させることは可能であるというこれまでの結果が再確認されました。さらに、 4 つの主要指標の改善を推進する能力とは何であるのかもつきとめました。これには、技術的プラクティス、クラウド導入、組織的プラクティス（変更承認プロセスなど）、文化が含まれます。

改善方法の指針を求める組織が進むべき現実的な方向はただ一つ。それは、まず基盤を確立し、次に自組織に固有の制約を特定して継続的改善に取り組む考えを取り入れるというものです。それらの制約が解消されたら、同じプロセスを繰り返します。また、こうした変革を実施するうえで最も効果的な戦略についてのアドバイスも示しました。

1 https://www.thoughtworks.com/radar/techniques/four-key-metrics

# 主な結果

1

**業界の改善は継続しており、特にエリート パフォーマーの間でそれが顕著である。**

最高レベルのパフォーマーの比率はほぼ 3 倍になっており、今や全チームの 20% を占めています。これは、エクセレンスを達成することは可能である - つまり、鍵となる能力を遂行すればそのメリットが得られることを示します。

2

**ソフトウェアを迅速、高信頼、かつ安全にデリバリーすることが、技術変革と組織的パフォーマンスの中核にある。**

ソフトウェアの速度、安定性、可用性が組織的パフォーマンス（収益性、生産性、顧客満足度など）に貢献しているという継続的なエビデンスが認められました。最高レベルのパフォーマーは、組織的パフォーマンスの目標を達成する可能性が 2 倍あります。

3

**組織で DevOps をスケールする最善策の主眼は、コミュニティを構築する構造的ソリューションにある。**

ハイ パフォーマーは、組織内で高いレベルと低いレベルの両方においてコミュニティ構造を作り出す戦略（実践共同体やサポートされた概念実証など）を好みます。そうすることで、組織再編成やプロダクトの変化に対してより持続可能で復元力の高い組織になると考えられます。

4

**クラウドは引き続きエリート パフォーマーにとっての差別化要因であり、高いパフォーマンスを推進している。**

クラウドの使用（定義は NIST Special Publication 800-145 に準拠）は、ソフトウェア デリバリー パフォーマンスと可用性につながるものと予測されます。最高レベルのパフォーマーは、ロー パフォーマーよりも、クラウド コンピューティングの 5 つの特徴のすべてを実装している割合が 24 倍高いという結果が示されました。

5

**生産性はワークライフ バランスの改善とバーンアウトの軽減を推進し、組織はスマートな投資によって生産性向上を支援することができる。**

組織は心理的安全性の文化を育てるとともに、ツール、情報検索、柔軟で拡張可能かつ可視性の高いシステムによる技術的負債の低減に対し、スマートに投資することで生産性向上を支援することができます。

6

**変更承認プロセスに対処する正しい方法があり、それは速度と安定性の改善とバーンアウトの軽減につながる。**

変更承認委員会などの負荷の高い変更承認プロセスは、速度と安定性にマイナスの影響を与えます。それに対して、明確に理解できる変更プロセスを確立すると、速度と安定性が推進され、バーンアウトが軽減します。

# 調査対象

DORA の調査は、業界で実践されているソフトウェア開発と DevOps のプラクティスに関する有益な情報を提供するものであり、実務に携わるプロフェッショナルから得た 31,000 件以上の調査回答を基にした 6 年に及ぶ科学的研究によって裏付けられています。今年のレポートには、世界中の幅広い業界からおよそ 1,000 人[2] の声が寄せられました。全体的に見ると、回答者の構成は個人および企業属性的な尺度から見て昨年とほぼ同じでした。ただし、チーム内の女性の割合が顕著に低下した点だけは昨年と異なります。

2 世界のソフトウェア プロフェッショナルの人口が 2,300 万人、信頼区間を 95% とすると、およそ 1,000 人の回答者を対象とした分析の誤差範囲は 3% です。

# 個人および企業属性

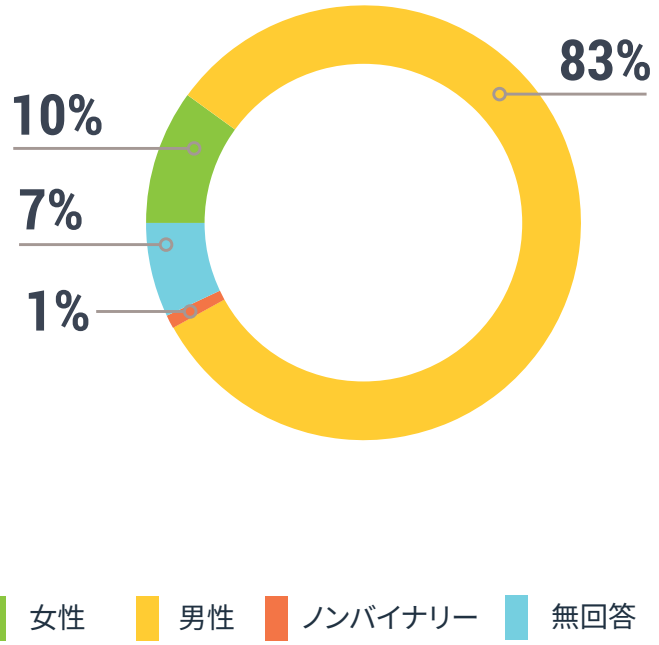
回答者の構成は、性別、障がい、過小評価グループといった主要な個人属性分類の全体にわたり、昨年と一致しています。調査回答者全体の性別構成はほぼ同じですが、チーム内の女性の割合は昨年より低下しました。

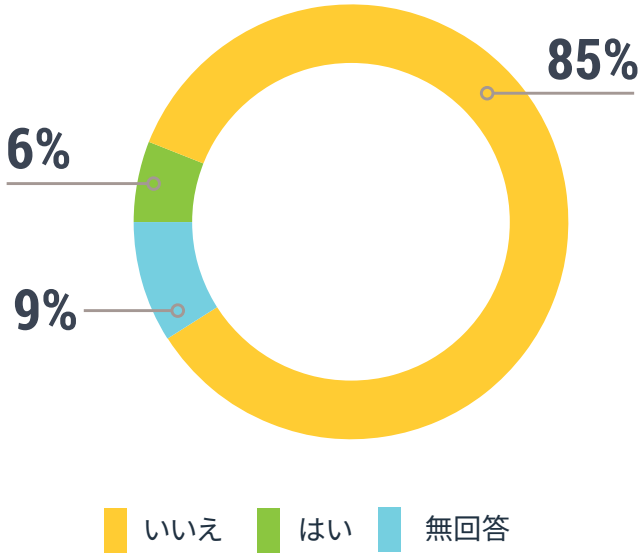
また、会社の規模、業界、地域といった主要な企業属性的分類の観点でも、回答者の構成は一致しています。回答者の大部分はテクノロジー業界でエンジニアまたはマネージャーとして働いています。回答者の所属する部署は、昨年までと同様に、コンサルタント、コーチ、セールス / マーケティング部門など多岐にわたります。業界についても変化はなく、金融サービスや政府機関などの高度に規制された組織から、医療、小売業まで、さまざまな業界に属する人から回答が集まりました。

# 性別

今年の調査回答の性別構成に変化はなく、男性 83%（昨年は 83%）、女性 10%（昨年は 12%）、ノンバイナリー 1% 未満（昨年は 1% 未満）でした。[3]

> 今年の調査回答では、チーム内の女性の割合は 16% にとどまり（中央値）、昨年の 25% から低下しました。




# 障がい

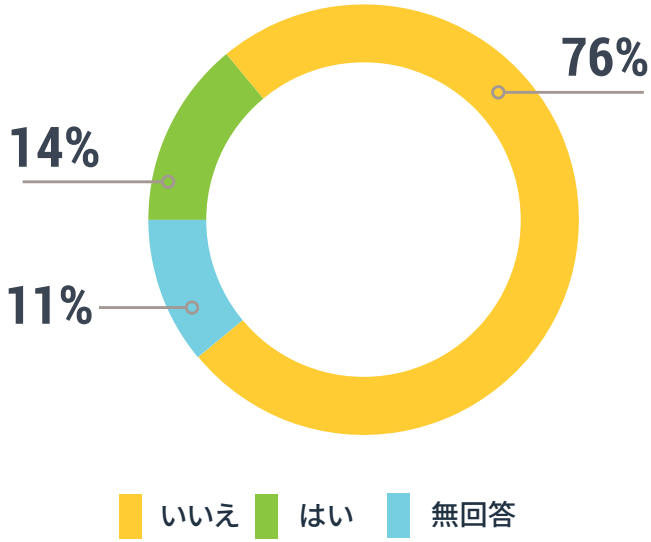
障がいの有無は、Washington Group Short Set のガイダンスに従い、6 つの側面から確認されました。今年は障がいに関する質問を始めてから 2 年目であり、障がいを持つ回答者の割合は 2018 年、2019 年とも 6% でした。[4]

3 これは Stack Overflow Developer Survey 2019 で報告された比率（男性 90%、女性 10%）とほぼ同じです。この調査にはノンバイナリーと無回答は含まれていませんでした。https://insights.stackoverflow.com/survey/2019

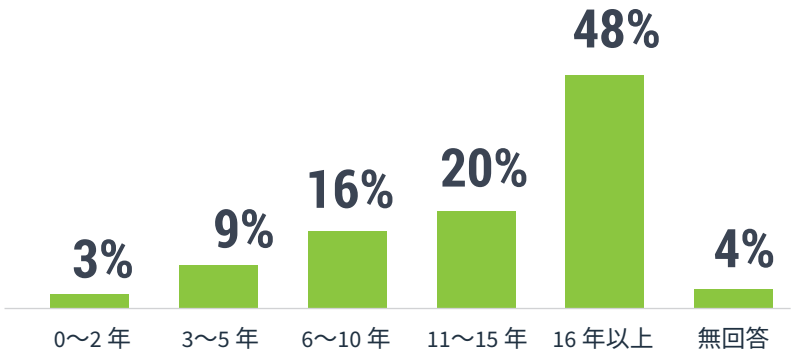
4 これは業界の他の調査で見られた比率と一致しています。たとえば、Stack Overflow Developer Survey 2019 では、全回答者の 6% がなんらかの障がいを持つと回答しています。https://insights.stackoverflow.com/survey/2019

# 過小評価グループ

過小評価グループに属しているかどうかは、人種、性別、その他の特性に基づいて判断されます。今年はこのデータの収集を始めてから 3 年目であり、2018 年は 13%、2019 年は 14% とほぼ一致していました。*




* 端数処理のため、100% を超えている




# 経験年数

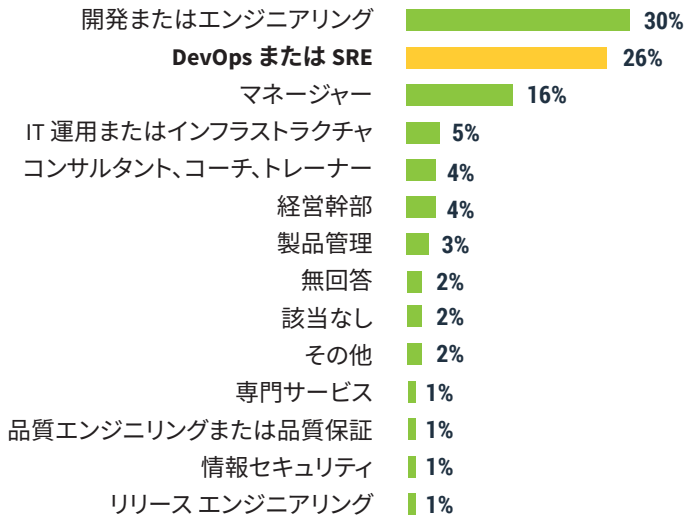
昨年と同様に、回答者の大部分は 16 年以上の経験を持ち（昨年は 50%）、その後に 11〜15 年が続きます（昨年も 20%）。

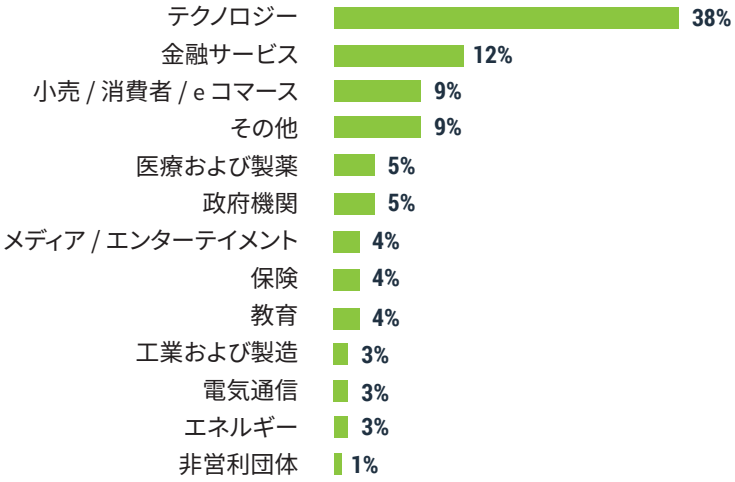
2019 年の全体的な個人属性的構成は 2018 年とほぼ一致しており、年度間のわずかな割合の相違は誤差範囲内に収まります。

企業属性

# 部署

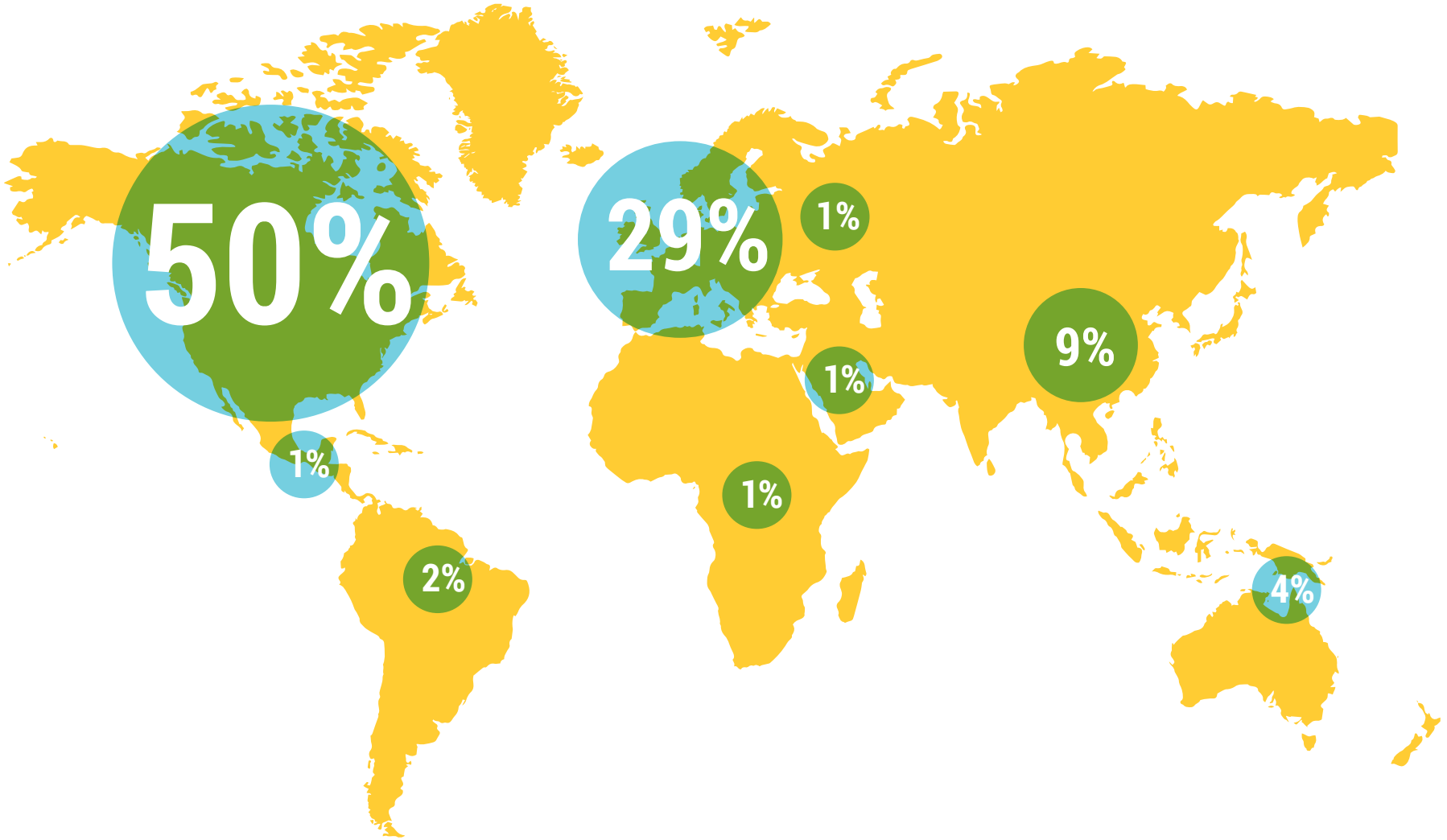
DevOps チームで働く回答者の割合は、調査の開始時より増加しています。2014 年は 16%、2015 年は 19%、2016 年は 22% で、その後 3 年間はおよそ 27% で安定しています（これは誤差範囲内です）。




# 業界

昨年と同様、ほとんどの回答者はテクノロジー業界で働いており、その後に金融サービス、小売、その他が続きます。



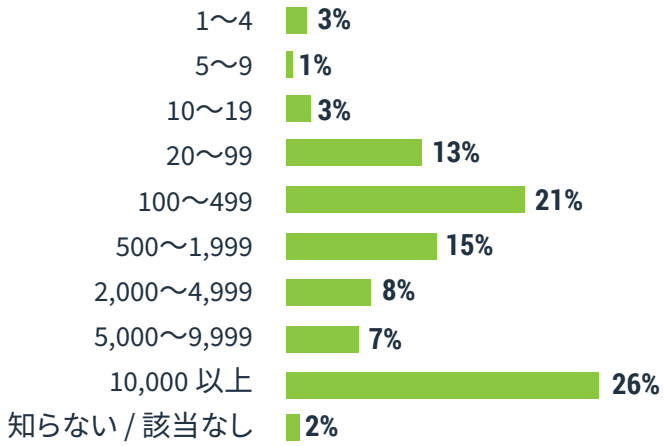

## 地域

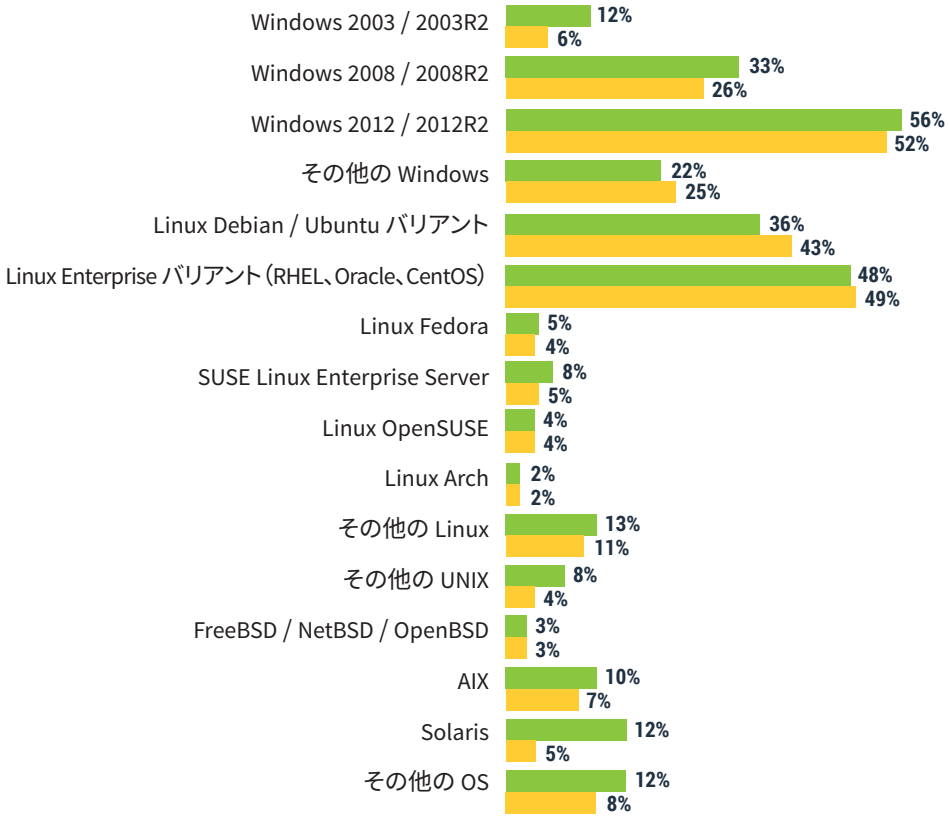
昨年と同様に、北米が全回答者の約半数を占め、その後に EU / 英国の 29% が続きます。アジアからの回答は、昨年の 18% から今年は 9% に低下しました。

企業属性

# 従業員数

回答者の 4 人に 1 人は従業員数 10,000 名を超える大企業に勤務しており、全回答者の 26% を占めます。回答者の半数は、従業員数 20〜1,999 名の会社に勤務しています。これらの分布は 2018 年のレポートとほぼ同じですが、従業員数 500〜1,999 名の会社に勤務する人からの回答は減少し（2018 年から 12% 減少）、従業員数 100〜499 名の会社に勤務する人からの回答は増加しました（2018 年から 7% 増加）。




# オペレーティング システム

オペレーティング システムの分布も昨年とおおむね一致していました。

2018
2019

# 比較に用いた手法

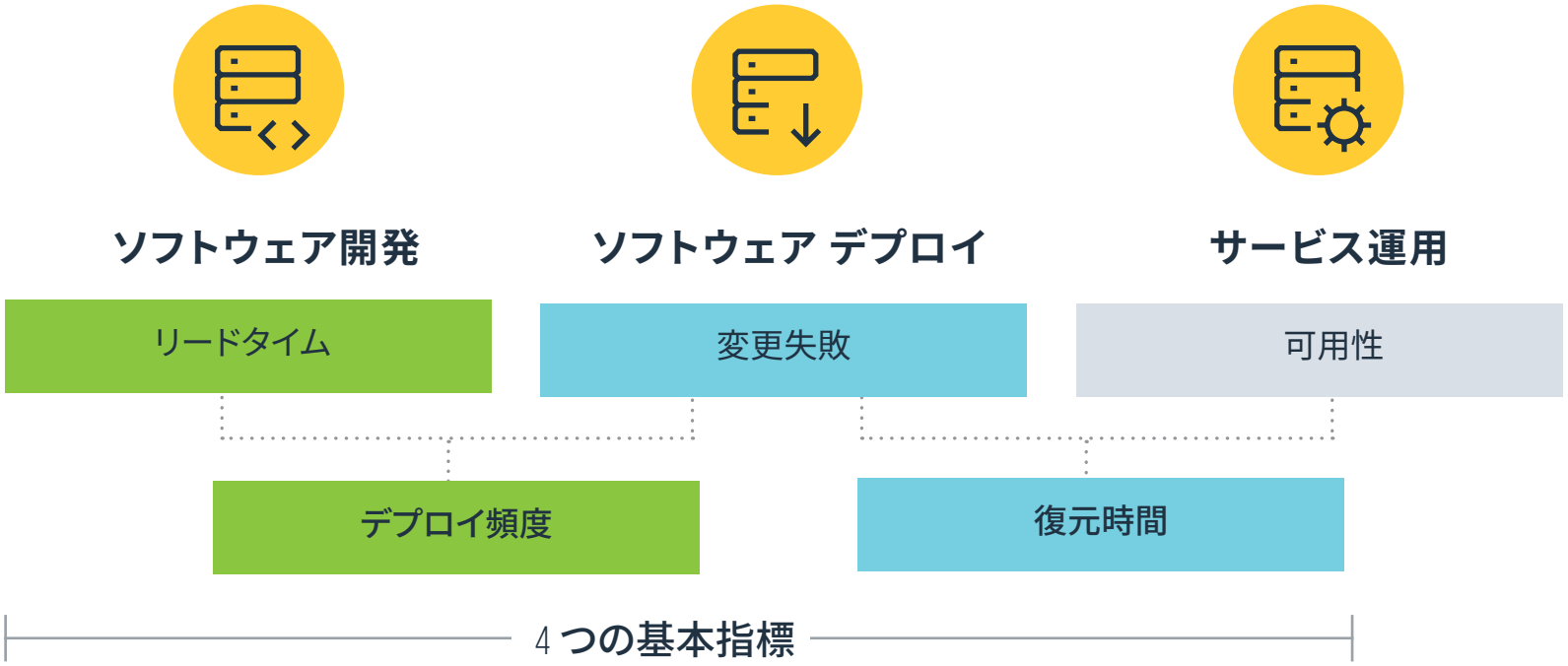
このセクションは、組織の DevOps のベンチマーク評価に役立ちます。本調査では、チームがソフトウェア システムをどのように開発、デリバリー、運用しているかを調べるために、厳密な統計学的手法を使用しています。エリート、ハイ、ミディアム、ローの各パフォーマーのベンチマークは、このレポートで実施された複数の重要な分析において自組織がどのレベルに位置するかを示します。また、前年度と比較した傾向も調べました。

# ソフトウェア デリバリーおよび運用パフォーマンス

組織は自らの目標を達成するうえで、ソフトウェアシステムをデリバリーおよび運用する能力への依存度をますます強めています。この重要な結果指標に関するパフォーマンスを比較するため、業界ではソフトウェア システムの開発およびデリバリーのプラクティスの有効性を測定する手段が求められています。 DORA では、過去 6 年間に 4 つの指標を開発し、検証してきました。これらの指標は、ソフトウェアデリバリーとパフォーマンスの俯瞰的なシステムビューを提供し、組織の目標達成能力を予測します。昨年、運用能力に焦点を当てた指標を 1 つ追加しましたが、この尺度は組織が優れた成果を上げるために有効であることがわかりました。本レポートでは、これら 5 つの尺度のことを**ソフトウェア デリバリーおよび運用（SDO）パフォーマンス**と呼んでいます。これはシステムレベルの成果に照準を合わせたもので、ソフトウェア指標によくある落とし穴を避けるのに役立ちます。こうした落とし穴はしばしば部門間の対抗意識をもたらし、局所的な最適化が図られて全体的な成果が犠牲になります。

開発およびデリバリー プロセスの有効性を捕捉する最初の 4 つの指標は、スループットと安定性の観点からまとめることができます。ソフトウェア デリバリー プロセスのスループットは、コード変更時のチェックインからリリースまでのリードタイムとデプロイ頻度を使用して測定されています。安定性は、復元時間（ユーザーに影響を与えるインシデントが検出されてからそれを修正するまでにかかる時間）と変更失敗率（リリース プロセスの品質を測定する尺度）を使用して測定されています。

## パフォーマンス指標



多くの専門家はこれらの指標を一連のトレードオフを表すものとして扱い、スループットの増加はソフトウェア デリバリー プロセスの信頼性とサービスの可用性にマイナスの影響を与えると信じています。しかし、本調査では 6 年連続して、速度と安定性は両立可能な成果であるという一貫した結果が得られています。2019 年のデータを使用したソフトウェア デリバリーに関連する 4 つの尺度のクラスタ分析では、4 種類のパフォーマンス プロファイルが明らかになり、スループットと安定性の尺度はそれらのプロファイル間で統計的な有意差がありました。[5] 昨年までと同様に、最高レベルのパフォーマーは 4 つの尺度の値すべてが有意に高く、ロー パフォーマーはすべての尺度の値が有意に低い結果となりました。

速度と安定性に加えて、可用性も運用パフォーマンスには重要です。ハイレベルにおける可用性は「テクノロジー チームと組織の、運用中のソフトウェアに関する約束や表明を果たす能力」を表します。その中でも特に、プロダクトまたはサービスが使用可能であり、エンドユーザーがアクセスできることを保証することを指します。[6]

可用性は、チームがどれだけ明確に可用性についての目標を定義しているか、現在の可用性を追跡しているか、過去の停止事象からの教訓を活かしているかを反映しており、チームのフィードバック ループが完全なものであることを確かめます。可用性の測定に使用される項目は、有効で信頼性の高い測定を可能にする枠組みとなります。

5 可用性は本調査のクラスタ分析には含まれていません。その理由は、パッケージ ソフトウェアやファームウェアなどのようにサービスの形で提供されないソフトウェア ソリューションの場合、可用性の尺度が同じように当てはまらないためです。

6 チームの可用性目標はサービスレベル契約（SLA）やサービスレベル目標（SLO）を使用して定義でき、そのパフォーマンスはサービスレベル インジケーター（SLI）を使用して測定できます。SLA、SLO、SLI の策定の詳細については、『Site Reliability Engineering: How Google Runs Production Systems』（2016 年、著者 Beyer 他）をご覧ください。

| ソフトウェア デリバリー パフォーマンスの側面* | エリート | ハイ | ミディアム | ロー |
|---|---|---|---|---|
| **デプロイ頻度**<br>担当している主要アプリケーションまたはサービスについて、どれくらいの頻度でコードを本番環境にデプロイまたはエンドユーザーにリリースしているか。 | オンデマンド（1 日に複数回のデプロイ） | 1 日に 1 回～1 週間に 1 回 | 1 週間に 1 回～1 か月に 1 回 | 1 か月に 1 回～半年に 1 回 |
| **変更のリードタイム**<br>担当している主要アプリケーションまたはサービスについて、変更のリードタイムはどれくらいか（コードがコミットされてから本番環境で正常に実行されるまでどれくらい時間がかかるか）。 | 1 日未満 | 1 日～1 週間 | 1 週間～1 か月 | 1 か月～半年 |
| **サービス復元時間**<br>担当している主要アプリケーションまたはサービスについて、ユーザーに影響を与えるサービス インシデントまたは不具合（予定外の停止やサービス障害など）が発生したときに、サービスの復元に通常どれくらいの時間がかかるか。 | 1 時間未満 | 1 日未満[a] | 1 日未満[a] | 1 週間～1 か月 |
| **変更失敗率**<br>担当している主要アプリケーションまたはサービスについて、本番環境に変更を加えた、またはユーザーへのリリースを実施した結果、サービスが低下し（たとえば、サービス障害やサービス停止が発生した）、その後修正（ホットフィックス、ロールバック、フィックス フォワード、パッチなど）を行う必要が生じた割合はどれくらいか。 | 0～15%[b,c] | 0～15%[b,d] | 0～15%[c,d] | 46～60% |

正規分布でないため、記載された値は中央値です。
特記された場合を除き、すべての差異について Tukey の事後分析で有意差が認められました。
a、b、c 平均値は、Tukey の事後分析で有意差が認められました。中央値については、基になる分布が原因で有意差は認められませんでした。
d 平均値は、Tukey の事後分析で有意差が認められませんでした。
* 4 つの指標を視覚的に表した図については、付録 A をご覧ください。

また、ソフトウェア デリバリー パフォーマンスが良好であるほど可用性が高いという昨年の結果も再確認されました。可用性の尺度はソフトウェア デリバリー パフォーマンスのプロファイルと有意に相関していました。エリート パフォーマーとハイ パフォーマーからは一貫して優れた可用性が報告され、エリート パフォーマーは強力な可用性プラクティスを持つ割合が 1.7 倍高いという結果が得られました。[7]

## 業界速度は早まっている

DevOps と技術変革に関して業界は「キャズムを乗り越えた」と報告するアナリストが増えており、DORA の今年の分析でもそれが裏付けられています。業界のペースは加速しており、速度と安定性を両立させることは可能です。クラウド技術への移行がこの加速を後押ししています。このことから、組織がステークホルダーに価値を提供できるようにする技術の重要性が改めて認識されます。

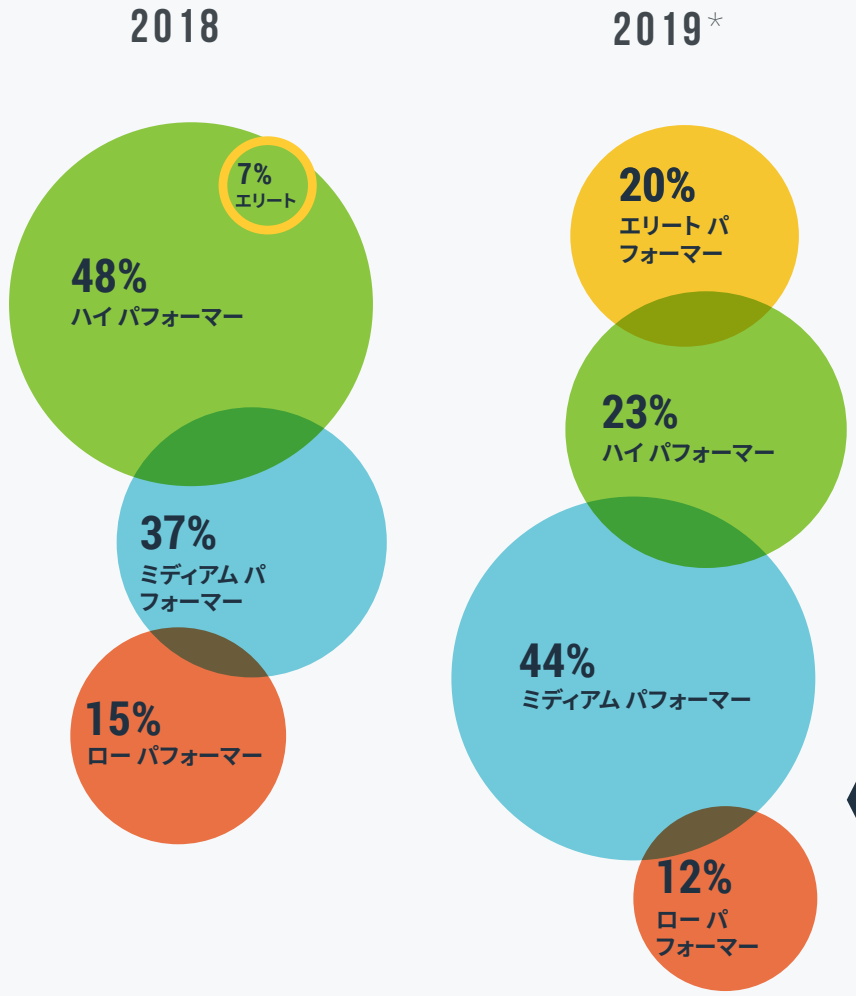
7 これらのプラクティスの中にクラウドだけにしか当てはまらないものは一つもないことを書き添えておきます。

# 業界および組織が SDO パフォーマンスに及ぼす影響

業界と組織の規模が SDO パフォーマンスに有意な影響を及ぼすかどうかを調べるため、(統制変数などによる)追加の分析を行いました。小売を除き、業界が影響を及ぼすというエビデンスは見つかりませんでした。このことは、金融サービスや政府機関などの高度に規制された業界も含め、あらゆる業種のあらゆる規模の組織が高いレベルのパフォーマンスを達成できることを示します。小売業界の分析結果からは、この業界では速度と安定性になんらかの利点があることが示唆されます。

従業員数が 5,000 名を超える大企業の組織は従業員数 5,000 名未満の会社に比べるとパフォーマンスが低いというエビデンスが見つかりました。これはおそらく、大きな組織に特有のいくつかの要因によるものと考えられます。特に、負荷の高いプロセスと統制、遅延と不安定性をもたらす密結合アーキテクチャは顕著な要因です。大企業がこのような結果をパフォーマンスが劣る言い訳とすることは望ましくありません。そうではなく、エクセレンスを達成することは可能であると認識し、継続的な改善プログラムに着手して、卓越したパフォーマンスを達成した他の大企業を参考に着想や指針を得ることを強くおすすめします。

各パフォーマンス クラスタの比率を 2018 年と 2019 年の間で比較しました。




* 端数処理のため、100% に達していない

## パフォーマンス クラスタ

昨年のレポートで初めてエリート パフォーマーが識別されましたが、このグループはハイ パフォーマーの一部にすぎませんでした。今年の分析では、4 つの明確に異なるグループが明らかになりました。同じ名称を使用したのは、エリート パフォーマーの速度と安定性の特性は今年も昨年も同じであり、これら 2 つのグループが同類であることを示すためです。

この比較からわかること:

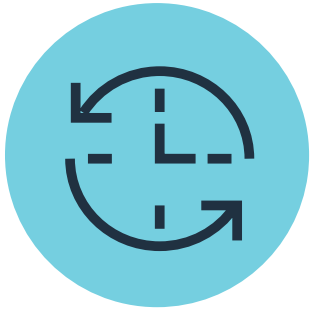
- エリート パフォーマーの比率がおよそ 3 倍になりました。これは、エクセレンスを達成することは可能であることを示します。必要なのは正しい実践、それに尽きます。
- ロー パフォーマーの比率は低下しました。これは、組織の技術変革が続く中、業界内で徐々にシフトが進行していることを反映しています。
- ミディアム パフォーマーの比率は増加しました。これには、ロー パフォーマーのパフォーマンスが改善されたケースと、複雑さの増加に悪戦苦闘したハイ パフォーマーがパフォーマンスを低下させたケースの両方が含まれます。

# エリート パフォーマー

エリート グループをロー パフォーマーと比較した結果は次のとおりです。

**208**
**倍多い**
コードのデプロイ頻度

**106**
**倍速い**
コミットからデプロイまでのリードタイム



**2,604**
**倍速い**
インシデントからの復元時間

**7**
**倍低い**
変更失敗率 (変更が失敗する可能性が 1/7)

スループット　安定性

# スループット

## デプロイ頻度

エリート グループは、オンデマンドのデプロイを日常的に行い、1 日に複数回デプロイを実施していると回答しており、これは過去数年一貫しています。それに対してロー パフォーマーは、デプロイ頻度が 1 か月に 1 回（年 12 回）から 6 か月に 1 回（年 2 回）の間と回答しており、昨年からパフォーマンスが低下しています。正規化された年間のデプロイ回数は、エリート パフォーマーでは年 1,460 回（4 回/日×365 日で計算）であるのに対し、ロー パフォーマーでは年 7 回（12 回と 2 回の平均）です。これらの数字を比較すると、エリートパフォーマーのコードデプロイ頻度はロー パフォーマーより 208 倍多いことになります。1 日に 4 回という見積もりは、ある製品のデプロイを 1 日に最大 50 回行うと述べている CapitalOne のような会社[8] や、Amazon、Google、Netflix といった 1 日のデプロイ回数が数千回（各社の本番環境を構成する数百のサービスを集計した場合）に上る会社と比較すると、かなり控えめであることも付記しておきます。

8 2017 年: https://www.informationweek.com/devops/capital-one-devops-at-its-core/d/d-id/1330515

## 変更リードタイム

同様に、エリート パフォーマーは変更リードタイムが 1 日未満と回答しています（変更リードタイムは、コードがコミットされてからそのコードが本番環境に正常にデプロイされるまでの時間で測定しました）。昨年のエリート パフォーマーは変更リードタイムが 1 時間未満と回答していることから、この指標は昨年より少し低下しています。エリート パフォーマーとは対照的に、ロー パフォーマーは 1 か月から半年のリードタイムを必要としていました。エリート パフォーマーのリードタイムが 24 時間（「1 日未満」の上限である控えめな見積もり）、ロー パフォーマーのリードタイムが 2,555 時間（1 か月 730 時間と 6 か月 4,380 時間の平均）とすると、エリート グループの変更リードタイムはロー パフォーマーよりも 106 倍速いことになります。

# 安定性

## サービス復元時間

エリート グループはサービス復元時間が 1 時間未満と回答したのに対し、ロー パフォーマーは 1 週間から 1 か月と回答しました。この指標の比較でも、ハイ パフォーマーでは 1 時間、ロー パフォーマーでは 1 週間（168 時間）と 1 か月（5,040 時間）の平均という控えめな時間を選びました。これらの数字に基づくと、エリート グループのサービス復元時間はロー パフォーマーより 2,604 倍速いことになります。すでに述べたように、サービス復元時間のパフォーマンスはエリート パフォーマーとロー パフォーマーのどちらにおいても、前年と変わりありませんでした。

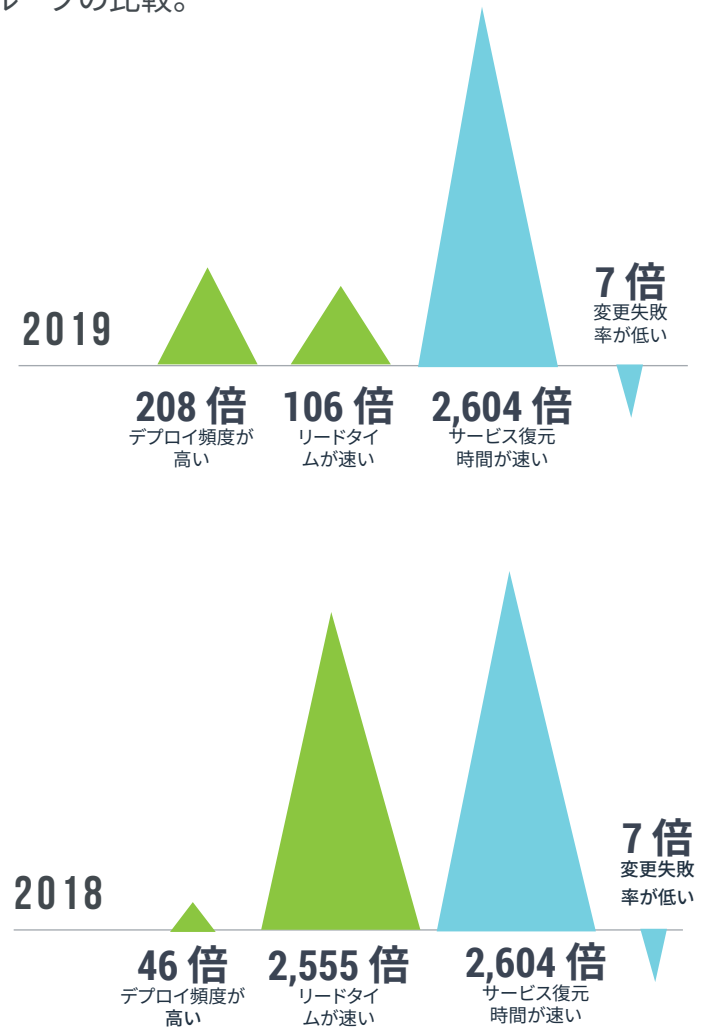
## 変更失敗率

エリート パフォーマーは変更失敗率が 0〜15% と回答したのに対し、ロー パフォーマーは 46〜60% と回答しました。それぞれの平均は、エリート パフォーマーが 7.5%、ロー パフォーマーが 53% です。これは、エリート パフォーマーの変更失敗率はロー パフォーマーより 7 倍良好であることを表しています。すでに述べたように、変更失敗率はエリート パフォーマーとロー パフォーマーのどちらにおいても、前年と変わりありませんでした。

ここで示した尺度はすべて相対的です。つまり、それぞれの年のパフォーマンスが最も高いグループと最も低いグループを比較したものです。各パフォーマンス指標のパフォーマンスが最も高いグループと最も低いグループの差を 2018 年と 2019 年の間で比較すると、変更リードタイム以外はすべて差が拡大しているか、同程度でした。デプロイ頻度の差の拡大は、ロー パフォーマーのパフォーマンスが低下したことを示します。これは、環境の複雑さが増し、その結果ソフトウェア デリバリーがより難しくなったことに起因している可能性があります。リードタイムにおいてロー パフォーマーとエリート パフォーマーの差が縮まったことは、エリート パフォーマーのパフォーマンスが低下したことを示します。エリート パフォーマーのリードタイムは、昨年の 1 時間未満から今年は 1 時間～1 日の範囲に増加しました。これは、より負荷の高いコードレビューと承認のプロセスが普及しつつあるという近年の傾向を反映している可能性があります。

# ソフトウェア デリバリー パフォーマンス

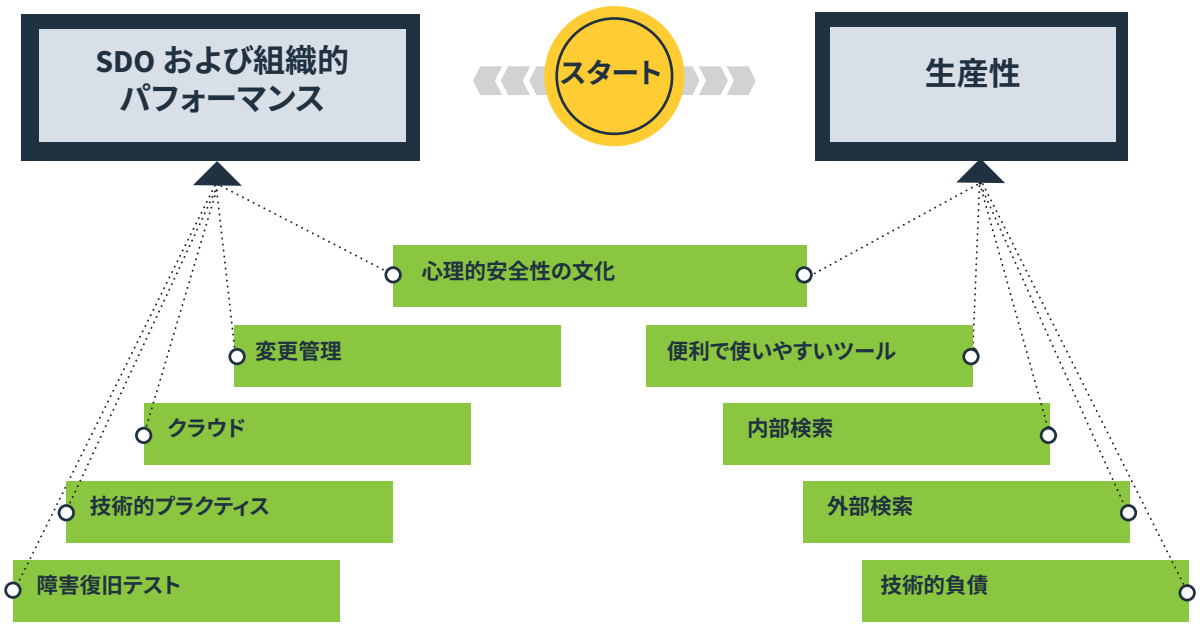
パフォーマンスの最も高いグループと最も低いグループの比較。



# 研究モデルの使用方法

今年のレポートでは、パフォーマンスと生産性の両面で改善を推進できるように、2 つの研究モデルが使用されています。なぜ研究モデルが 2 つあるのか不思議に思われるかもしれません。両者の相違点と共通点はどこにあるのでしょうか。さらに、最も重要なこととして、これらのモデルは意思決定や業務の方向付けにどのように役立つのでしょうか。

## 最初に目標を定める



**SDO パフォーマンスまたは組織的パフォーマンスの改善を目指す場合は、**パフォーマンスの構成概念を含むモデルを選びます。どのような能力に重点を置けばよいかについては、本レポートのこのセクションをご覧ください。

**生産性の改善を目指す場合は、**生産性の構成概念を含むモデルを選びます。どのような能力に重点を置けばよいかについては、本レポートのこのセクションをご覧ください。

## 研究モデルを使用して変革を導く方法

- 設定した目標を改善する能力（すなわち、改善する構成概念を指す矢印が付いているもの）を特定します。本レポートですでに見てきたように、これらは自組織に改善をもたらす能力の候補です（SDO および組織的パフォーマンスについては、過去 5 年の調査で他の能力も突き止められています）。[9]

- 変革を加速するには、まず確固たる基盤を確立してから、制約となっている能力に重点的に取り組む必要があります。たとえば、最も大きな遅延を引き起こしているのはどの能力か、最も大きな頭痛の種は何か、最も大きな問題はどこにあるか、などの答えを明確にします。まずこれらを解決するために、3〜5 人の専任の担当者を選抜します。まだ問題が残っていても心配いりません。現在最も大きな問題に取り組むことで、ボトルネックを解消し、相乗効果を見出して、不必要な作業を回避できます。

- この作業には他にも重要な成果があります。SDO および組織的パフォーマンスの改善を追求することには、バーンアウトやデプロイの苦痛の軽減（2016 年と 2017 年に調査）、セキュリティ結果の向上（2017 年と 2018 年に調査）、文化の育成（2014 年から 2019 年まで調査）というメリットがあります。生産性の改善に伴うメリットとしては、ワークライフ バランスの改善とバーンアウトの軽減が挙げられます。

## 研究モデルの読み方

本調査では構造方程式モデル（SEM）を使用しています。これは、関係の評価に使用される予測モデルです。各ボックスは本調査で測定した構成概念を表し、各矢印は構成概念間の関係を表します。他のボックス（構成概念）を内包する大きなボックスは、2 次構成概念です。他の構成概念を指す点線が付いた水色のボックスは、統制変数です（モデルの全体図については、31 ページと 57 ページをご覧ください）。太字の構成概念は、今年初めて調査された項目を表します。太い輪郭線で囲まれた構成概念は、チームと組織の共通目標（SDO パフォーマンスと組織的パフォーマンス、または生産性）です。目標を定めるとき、およびモデルを読むときは、これらを念頭に置いてください。

9 すべての State of DevOps Report は cloud.google.com/devops でご覧いただけます。

矢印が付いたすべての線を「予測させる」、「影響を与える」、「推進する」という動詞に読み替えると、モデルを理解しやすくなります。たとえば、2 次構成概念の「SDO パフォーマンス」は「ソフトウェア デリバリー パフォーマンス」と「可用性」という構成概念で構成されており、これらが組み合わさって組織的パフォーマンスを推進します。「障害復旧テスト」の構成概念は「可用性」を推進します。障害復旧テストは今年から調査対象に加わった構成概念であり、太字で記載されています。横に「-」記号が付加された矢印付きの線は、2 つの構成概念の間にマイナスの影響があることを示します。たとえば、「技術的負債」は「生産性」にマイナスの影響を与えます（生産性を低下させます）。

## 2 つの研究モデルには共通部分がある

両モデルに共通部分があるのは、SDO パフォーマンスと生産性という目標が多くの点で関連しているためです。これらの成果を上げるには、優れた方法、かつ組織と個人の双方に価値を提供するような方法でテクノロジーを構築し、提供する必要があります。ソフトウェア デリバリー業務を支援するために行うことがソフトウェアの開発やデリバリーを担当している個人の生産性に有益な影響を及ぼすことは、理に適っています。しかし、両モデルには類似する部分はあるものの、測定する成果は異なるため、分析は別々に実施することにしました。その結果、異なる 2 つの研究モデルが生まれました。

## 2 つの研究モデルの共通部分からわかること

- SDO パフォーマンスを追い求めてスマートな投資を行うと、バーンアウトを軽減できます。また、生産性の向上もバーンアウトの軽減につながる可能性があります。これは、業務の需要が増大し続ける中で、組織とテクノロジストの双方に明るい材料となります。良好なワークライフ バランスを保つことは、バーンアウトを軽減するための鍵です。
- 心理的安全性の文化は、SDO パフォーマンス、組織的パフォーマンス、生産性のすべてに寄与します。これは、健全な文化を育み成長させることは組織と個人にとって利益となることを示しています。
- コードのメンテナンス性、疎結合アーキテクチャ、モニタリングへの投資は、SDO パフォーマンスと生産性の向上につながります（前者は継続的デリバリーを通じて、後者は技術的負債の低減を通じて）。これは、優れたツールとシステムがいかに重要かということでもあります。

# SDO パフォーマンスと組織的パフォーマンスを改善するには

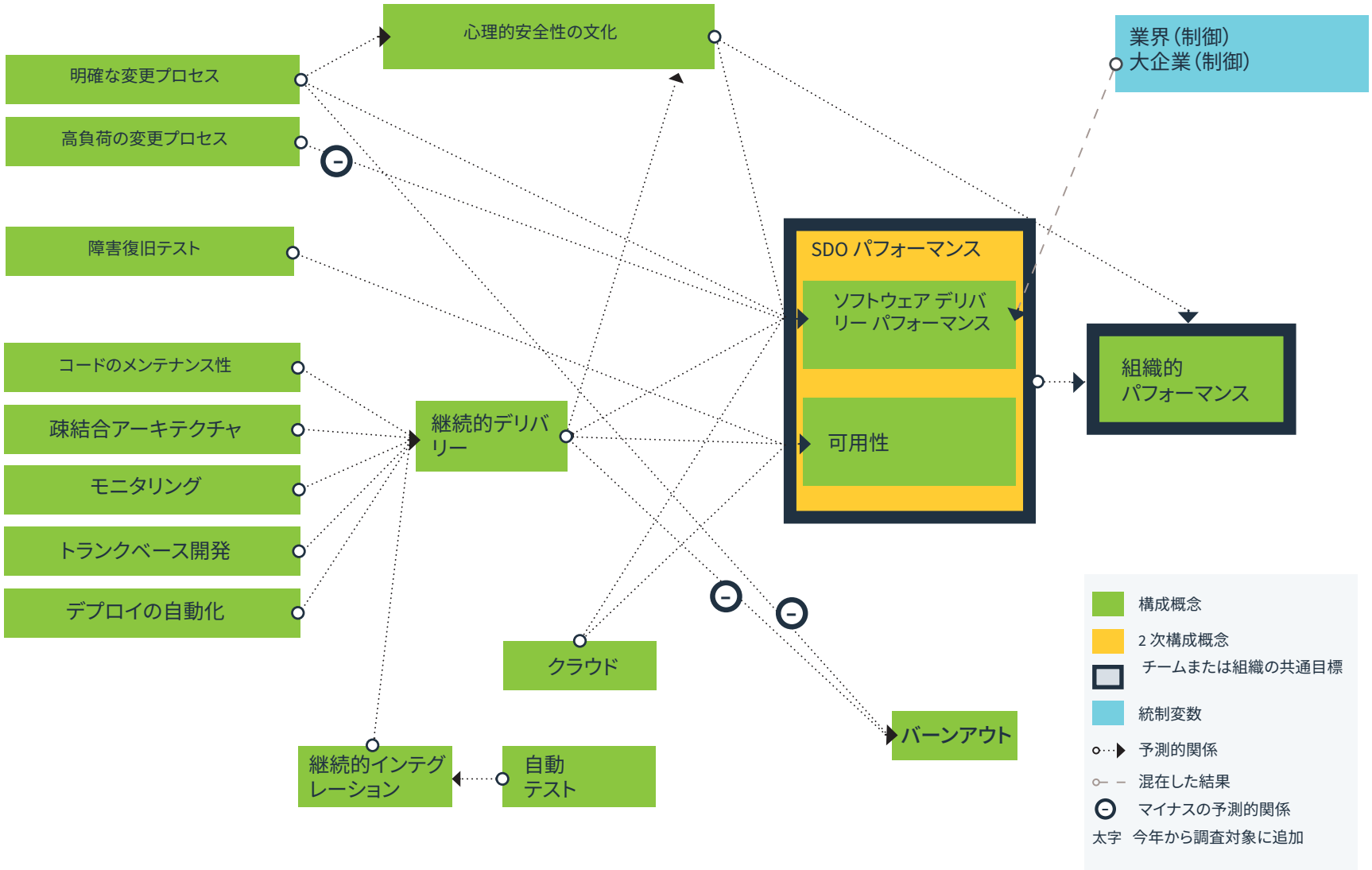
デジタル変革の重要な目標は、ソフトウェア デリバリー パフォーマンスを最適化することです。これは、テクノロジーを活用して顧客とステークホルダーに価値を提供することとも言えます。本調査はエビデンスに基づく指針を提示するものであり、読者が変革を加速するうえで重要な能力とプラクティスに注力することを可能にします。さらに、今年のレポートには実装戦略の要点も追加されています。これらの戦略を参考に、最大限の効果の実現に向けて組織の進路を定めてください。

# SDO パフォーマンスと組織的パフォーマンスの改善

まず、本調査で示された能力に的を絞ります。これらの能力は、テクノロジーの提供を改善して価値を与えるための予測的なガイドラインとなります。最初に、基盤を確立することから始めます。基盤とは、基本的な自動化（バージョン管理や自動テストなど）、モニタリング、明確な変更承認プロセス、健全な文化を指します。次に、何が制約になっているかを特定し、進行計画を策定します。この戦略は、変革に着手したばかりの組織だけでなく、最適化に数年間取り組んでいる組織にも有効です。現在障壁となっているものにリソースを集中した後、同じ工程を反復します。つまり、次の制約を特定して新たなターゲットを選びます。

改善する目標を定め、それに影響を与える能力を特定するには、31 ページのモデルを使用します。たとえば、ソフトウェア デリバリー パフォーマンスを目標にする場合、これに影響を与える能力は文化、明確な変更プロセス、継続的デリバリー、クラウドです。次に、最大の制約となっているものに取り組みます。

# ソフトウェア デリバリーおよび運用パフォーマンス



# 卓越しなければ未来はない

本調査の分析で、ソフトウェア デリバリーの速度と安定性において各業界で特別な成果が上がったというエビデンスは見つかりませんでした。ただし、小売だけは例外で、この業界では SDO パフォーマンスが有意に高いという結果になりました。大型商業施設の閉鎖が相次ぐ中、「リテール アポカリプス」（小売業の最後の日）という言葉も生まれた小売業界の激しい競争環境を考えると、これは意外なことではありません。収益性、生産性、顧客満足度に秀でた小売業者は生き残ります。エクセレンスに到達できなければ、倒産につながります。小売業者がこの競争の激しい変化の最前線にいる一方で、金融サービスなどの他の業界もそのすぐ後に続こうとしています。

技術変化のペースに遅れずついていくことは、このような競争環境で要求の厳しい顧客を満足させながら安定した収益を上げ、なおかつステークホルダーをも満足させなければならない組織にとって、必要不可欠です。小売業界はテクノロジーが価値を提供することを示す見本であり、小売業が歩んできた道から学ぶことは他の業界にとって有意義です。

- 顧客が購入を最適化できるように、顧客の購入習慣、嗜好、顧客とのやり取りを把握する A/B テストをウェブサイトやアプリに取り入れたのは、小売業者が最初でした。この技術能力にはより堅牢なテクノロジー ソリューションが必要ですが、これを実現できればプロダクト開発やマーケティングへの強力なフィードバック ループが形成されます。
- 小売業は利益率が低いケースが多く、すばやくスケーリングして変化に対応するために効率と自動化が求められます。
- 小売業者には需要の大きな変動に対処する能力が必要であり、これがなければ倒産に追い込まれる危険があります。たとえば、ブラック フライデーは小売業者の年間売り上げを大きく左右します。クラウドを活用すれば処理能力を容易に拡張できるため、小売業者はクラウドを「使用するかどうか」または「いつ使用し始めるか」に関する議論に時間を費やすことはありません。すでにその段階は過ぎています。
- 小売業者が高度に規制された環境で俊敏に動く方法を見つけ出したのは、必要にかられてのことでした。他の業界には導入の遅れを規制のせいにする余裕がありましたが、激しい競争環境にさらされている小売業者は、規制環境ですばやく安全にビジネスを進める方法を考え出さざるを得ませんでした。そして、それを見事に成し遂げています。いろいろ言ったところで、クレジット カード取引がまれにしか起こらなくても、それを処理しなければ商品を販売することはできません。

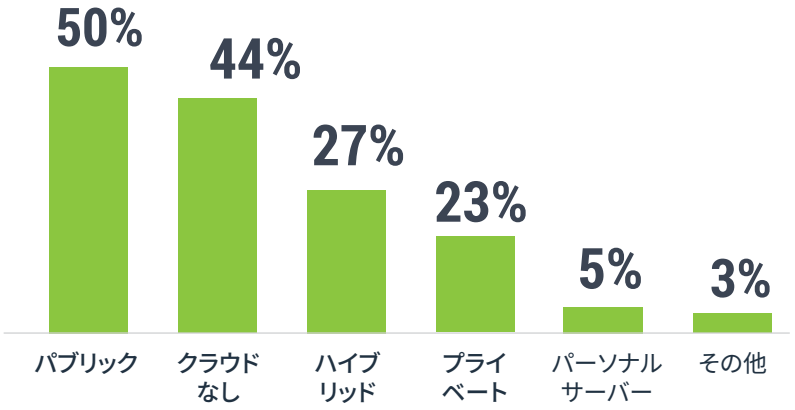
# クラウド

ビジネスには絶えず進化する性質があり、マルチクラウド ソリューションやハイブリッド クラウド ソリューションを選ぶ組織がますます増えています。その理由は、こうしたソリューションにはパフォーマンスの向上に加えて柔軟性、制御性、可用性というメリットもあるためです。[10] 本調査の回答では、マルチクラウドとハイブリッド クラウドの使用は昨年より増えていました。

また、各自の主要アプリケーションの作業がどこでホストされているかを尋ねたところ、こちらに対する回答からも、「ハイブリッド クラウドまたはマルチクラウド環境で作業する」ということが何を意味しているかについて明確な見解の一致がないことが見て取れました。昨年のレポートで述べたように、「ハイブリッド」が何を指すかはしばしば自己定義的です。回答者がハイブリッド環境で作業していると答えれば、ハイブリッド環境で作業していることになります。専門家がこれらの用語や業界の状況について論じようとするとき、これがフラストレーションになることは多く（定義も測定もできないものを比較することはできない）、業界アナリストの作成するレポートが一貫したものにならない原因となっています。

10 Transform Your Business with a Hybrid and Multicloud Strategy、執筆者 Tilak 氏、2019 年 3 月

## 主要サービスまたはアプリケーションのホスティング



# クラウド インフラストラクチャをどのように実装するかは重要である

「クラウドにある」という言葉に対して各自がそれぞれ異なる理解をしているとすれば、その利得を実際にどのように測定すればいいのでしょうか。DORA では、この制約に対処するため、**クラウド コンピューティングの基本的な特徴***（定義は米国国立標準技術研究所（NIST）の文書に準拠）に注目し、これをガイドとして使用しています。

本調査では、回答者の 80%[11] が、担当する主要アプリケーションまたはサービスがある種のクラウド プラットフォームでホストされていると回答しています。NIST フレームワークを使用して基本的プラクティスが SDO パフォーマンスに及ぼす影響を調べたところ、昨年に引き続き、本当に重要なのは、単にクラウド テクノロジーを使用するということではなく、クラウド サービスをどのように実装するかということでした。

* このレポートで太字で示した用語は、31 ページと 57 ページの研究モデルに含まれる構成概念に対応しています。

11 回答者の主要サービスまたはアプリケーションがどこでホストされているかを示す前述のグラフを参照してください。アンケートでは複数の選択肢を選べるようになっており、回答者の 80% がクラウドを含む選択肢を選びました。

エリート パフォーマーは、クラウドの基本的な特徴をすべて満たしている割合がロー パフォーマーより 24 倍高い結果となりました。[12] これは、クラウド コンピューティング技術をすでに導入していると回答したチームや経営陣が、速度と安定性という約束された成果が得られないことに不満を感じている理由の説明となります。つまり、クラウド コンピューティングを使用していると回答した人の多くは、実際には肝心な基本パターンを導入していないのです。クラウド インフラストラクチャを使用していると答えた回答者のうち、NIST が定義するクラウド コンピューティングの 5 つの基本的特徴をすべて満たしていると思う、または非常にそう思うと答えた人の割合は 29% にすぎませんでした。

12 これは昨年の結果と一致しています。昨年は、クラウドの基本的特徴をすべて満たしていると思う、または非常にそう思うと答えたエリート パフォーマーの割合はロー パフォーマーより 23 倍多いという結果でした。

# クラウド コンピューティングの 5 つの基本的な特徴

% そう思う、または非常にそう思うと答えた人の割合

57%

## オンデマンド セルフサービス

2018 年から 11% 増加

ユーザーがプロバイダと直接やり取りすることなく、必要に応じて自動的にコンピューティング リソースをプロビジョニングできる。

60%

## 幅広いネットワーク アクセス

2018 年から 14% 増加

スマートフォン、タブレット、ノートパソコン、ワークステーションなどの異種プラットフォームを通じてコンピューティング能力にアクセスできる。

58%

## リソースの共用

2018 年から 15% 増加

プロバイダ リソースがマルチテナント モデルでプールされており、物理リソースと仮想リソースがオンデマンドで動的に割り当てられる。ユーザーが国、州、データセンターなどの高い抽象レベルでリソースの場所を指定できる。

58%

## スピーディな拡張性

2018 年から 13% 増加

コンピューティング能力を弾力的にプロビジョニングおよび解放し、オンデマンドで迅速にスケールインまたはスケールアウトできる。ユーザーからは割り当て可能な能力が無尽蔵にあるように見え、いつでもどれだけの量でも調達できる。

62%

## サービスが計測可能であること

2018 年から 14% 増加

ストレージ、処理、帯域幅、有効なユーザー アカウントなどのサービスのタイプに基づいて、クラウド システムが自動的にリソースの使用を制御、最適化、報告する。

昨年の調査結果では、エリート パフォーマー チームはクラウドの 5 つの基本的特徴のすべてを実装している割合がより高いことがわかり、その結果は今年も再確認されました。これらの特徴は、実施可能な成功戦略を可能にするため（本調査はこれらの特徴が SDO パフォーマンスに影響を与えることを示しています）、クラウド コンピューティングを導入するとはどういうことなのかを定義する際に重要となります。パブリック、プライベート、ハイブリッドといったクラウドの種類にかかわらず、クラウドでの実行に重点を置くことで、チームまたは組織は速度、安定性、可用性というメリットを得ることができます。

# クラウドのスケーリング

クラウドを使用することで得られる明確な利点の一つにオンデマンド スケーリングがあります。動的スケーリングを活用するチームは、サービスの背後にあるインフラストラクチャをスケールしてユーザーの需要に柔軟に対応できます。サービスの利用状況をモニタリングし、必要に応じて自動的にインフラストラクチャをスケールできます。

クラウドで使用される抽象化は、インフラストラクチャに対する考え方や想像を一変させました。Kubernetes や Mesos などのコンテナ オーケストレーターを使用する場合、提供されるパッケージは、コンテナ イメージにデプロイ用の構成を加えたものです。典型的な Platform as a Service（PaaS）サービスは、パッケージング方法および実行用のコンテナ ランタイムとしてコンテナ イメージを中心としたデプロイモデルに一層重きを置くようになっています。この傾向は、Heroku、Google App Engine、Azure Container Instances、Cloud Foundry、Amazon Fargate などのプロダクトで見られます。サーバーレス（Function as a Service、FaaS とも呼ばれる）[13] はこれをさらに一歩押し進めたもので、デプロイを容易にします。開発者や運用者がスケーリングの抽象化、キャパシティ プランニング、メンテナンスを行う必要がなくなり、アプリケーション コード自体の実行のみに意識を集中できます。たとえば、AWS Lambda、Azure Functions、GCP Cloud Functions、Zeit などがこれに該当します。

クラウドにおける抽象化は次第に、デプロイの普遍的な標準としてクラウド プロバイダやプラットフォーム プロバイダ業界全体に広がりました。ネットワーク、仮想マシン、Identity and Access Management（IAM）、ストレージ、データベースはすべて、機械学習、モノのインターネット（IoT）、コンテナ ソリューション、言語ランタイム ソリューション、セキュリティ製品とともに、今やどのクラウド サービス プロバイダでも提供されているデファクト プロダクトになっています。DORA は今後も、コンテナ、ランタイム言語、アプリケーション パッケージを取り巻くパラダイムに対するプロダクトのサポート状況がどのように収束していくかを追跡し続けます。

13「サーバーレス」という用語は、「リッチ クライアント」アプリケーションを説明する際にも使用されています。ここでは、「サーバーレス」は Function as a Service を意味するものと限定します。詳細については、Mike Roberts 氏によるこの投稿をご覧ください。https://martinfowler.com/articles/serverless.html#WhatIsServerless

# クラウドのコスト

クラウドは、インフラストラクチャとデプロイメントに対するコストの考え方も変えました。もはや、測定単位はデータセンター全体ではなく、フルラックのサーバーですらありません。クラウド プロバイダのユーザーは、必要なときにいつでもスケールできる俊敏性を手にしながら、料金は使用した分だけ支払えば済みます。

クラウドのベスト プラクティスを取り入れることは、 SDO パフォーマンスにプラスの影響を与えるだけでなく、テクノロジーの運用にどれだけコストがかかっているかを把握できるという利点もあります。クラウドの基本的特徴をすべて満たしている回答者は、ソフトウェアの運用コストを正確に推定できる可能性が 2.6 倍高い結果となりました。また、運用に最もコストがかかっているアプリケーションを容易に特定できる可能性が 2 倍高く、ソフトウェア運用コストを予算の範囲内に抑えられる可能性も 1.65 倍でした。

# ブラック ボックス対透明性

クラウドを使用しているチームはなぜコストの推定と管理がうまくできるのでしょうか。それは、クラウド プロバイダが開発者や IT 運用担当者にインフラストラクチャの使用状況と支出を可視化する便利な機能を提供しているためと考えられます。このような高い可視性とコスト意識の向上はシステムの設計および構築方法に変革をもたらすとともに、インセンティブを一致させます。この新しい料金モデルに慣れていない人たちは、この変動性に最初は混乱や戸惑いを覚えるかもしれません。しかし、使用したコンピューティング リソースの分しか料金がかからないことから、効率的なシステムを設計すればチームにとってメリットになります。

それに対して、従来の環境におけるデータセンターはしばしば「ブラック ボックス」であり、処理コストとサイクルコストに関する情報を得るのは困難または不可能です。さらに、資本支出の性質上、インフラストラクチャを購入した後、設計によって積極的に効率化を図ってもメリットはありません。この点において、資本支出は固定費です。それは予測可能で直感的にわかりやすいものの、エンジニアリング チームに知らされることはほとんどなく、より効率的な設計を導入したとしても支払いを避けることはできません。

財務部門の人が次のように言うことがあります。「クラウドは短期的にはコスト節約につながらなかった。しかし、クラウドの情報透明性が高いことはわかっている。」なぜこのようなことが起こるのでしょうか。クラウドはコストに関する透明な情報をシステム オーナーに提供しますが、チャージバック モデルやそれに類似したメカニズムがない限り、ユーザーがそれらの費用を支払うことはありません。これにより大幅に変動する費用がまったくチェックされない事態につながる可能性があり、その結果クラウドコストが予測できなくなります。このようなシナリオでは、データセンターはコストが予測可能であるというだけの理由で、インフラストラクチャの料金を負担するチームがデータセンターの利用を選ぶ場合があります。たとえ、データセンターの可視性の欠如から、システム ユーザーがより効率的なシステムを構築しようとしないという欠点があっても、そのようなことは関係ありません。そこで提案したいのが、チャージバック モデルやそれに類似したメカニズムを使用することで組織間のインセンティブを一致させることです。これにより、システム オーナーがより効率的なシステムを構築するための可視性とそれを実施するためのモチベーションの両方を持つことができます。

ワークロードの中にはオンプレミス環境で実行する方が適しているものがありますが（定常的なものや予測可能なものなど）、クラウド インフラストラクチャを使用することには、 Infrastructure as Code を利用できるなど、オンプレミスにはない利点があります。

# 技術的プラクティス

## 最大限の効果を目指す

DevOps の導入を望む多くの組織は、導入工程のガイドとなる規範的手順またはベスト プラクティスを求めています。しかし、個々の組織は互いに異なり、どのプラクティスを採用すればよいかは組織の現状（テクノロジー、文化、プロセスの状態など）と短期的および長期的目標によって異なります。

これに対する解決策は、全体論的アプローチをとることです。このアプローチではまず、現在のソフトウェア デリバリー プロセスにおける制約を、短期的および長期的な成果を考慮しつつ、測定可能な観点から理解しようとします。次に、それらの成果を達成するために最良の方法を決定する権限をチームに与えます。結局のところ、彼らはそれぞれの業務や状況における専門家です。[14] このようなアプローチをとると、よりスケーラブルで柔軟なソリューションが見つかります。また、詳細な実施計画を細かく管理しなくても高レベルの成果のみに着眼することができ、組織の成長を促します。長期的な戦略をサポートする短期的な成果を設定してこれに重点的に取り組むことにより、チームは突発的な不測の問

14 このアプローチは、Mike Rother 氏のトヨタの研究に基づく著作に由来しています。詳細については、http://www-personal.umich.edu/~mrother/Homepage.html をご覧ください。

題に対応できるようになります。さらに、柔軟性と俊敏性に欠けた 3〜5 年計画を持ち、顧客需要の変化、テクノロジーの状況、セキュリティ上の新たな脅威に追いつけない同僚よりも、高いパフォーマンスを発揮できます。[15]

それ一つですべての改善事例に対処できる万能のアプローチはありませんが、DORA がこれまで組織の DevOps 導入を支援してきた中で、いくつかのテーマが明らかになっています。それらのテーマは特に、困難で複雑そうに見える制約がある中で変革を加速しようとしている会社に適しています。

## チームレベルと組織レベルの一致協力

能力の中には、チームレベルで向上させるのが一般的なものもあれば、(特に大きな組織や確固たる階層構造を持つ組織において) 組織レベルの取り組みが必要になるものもあります。チームレベルと組織レベルという 2 つの流れは、互いにサポートし合いながら並行して進めることができ、またそうすべきでもあります。

たとえば、自動テストに関するフィードバックが簡単かつ迅速に得られる継続的インテグレーションのプラットフォームは、組織内の複数のチームで使用すれば大きな戦力増強策になり得ます。同様に、チームのコードをデプロイする際に必ず他のチームのコードと組み合わせなければならない場合、チームレベルでデプロイを自動化してもほとんど効果はありません。これは、組織レベルで解決しなけれ

2019 年に調査した能力[16]

| | |
|---|---|
| **組織レベル** | • 疎結合アーキテクチャ<br>• 明確な変更プロセス<br>• コードのメンテナンス性 |
| **チームレベル** | • 継続的インテグレーション<br>• 自動テスト<br>• デプロイの自動化<br>• モニタリング<br>• トランクベース開発 |
| **チームレベルと組織レベルの両方** | • クラウド サービスの使用<br>• 障害復旧テスト |

15 チームにキャパシティとリソースを継続的に提供することは、このアプローチの成功にとって不可欠です。

16 SDO パフォーマンスの改善を推進する能力の完全なリストについては、次のレポートの付録 A をご覧ください。『Accelerate: The Science of Lean Software and DevOps』(2014〜2017 年の State of DevOps Report をまとめたもの)、「2018 Accelerate State of DevOps Report」、本レポート。

ばならない（その結果、通常は個々のチームでの作業も必要になる）アーキテクチャ上の問題があることを示します。

これらの能力については、続くセクションでチームレベルの能力と組織レベルの能力というレンズを通して詳細に考察します。

忘れてならないのは、ソフトウェアのデリバリー能力を改善することが目標であるということです。この目標を、**継続的デリバリー（CD）**と呼ばれるデリバリーとデプロイに関する技術的プラクティスを通じて達成します。CD は、リリースの実施に伴うリスクとコストを軽減します。ただし、組織の成功を望む場合、継続的デリバリーのための継続的デリバリーでは十分ではありません。収益性、生産性、顧客満足度といった組織的目標を視野に入れてこれを行う必要があります。

**継続的デリバリーをどのように測定したか**

- ソフトウェア デリバリーライフサイクルの全体を通して、本番環境またはエンドユーザーにオンデマンドでデプロイできる。
- システムの品質とデプロイ能力に関する迅速なフィードバックがチームメンバー全員に提供され、このフィードバックに基づいて行動することがメンバーの最優先事項である。

## チームレベルの技術的能力

昨年の調査では、**テストの自動化**が CD に大きな影響を与えることがわかりました。今年は、昨年の研究を踏まえたうえで、自動テストが**継続的インテグレーション（CI）**にプラスの影響を与えるという結果になりました。開発者は自動テストの結果を信頼し、テストスイートに合格した場合はコードが正常にデプロイ可能であると考え、合格しなかった場合は実際になんらかの問題があると考えます。問題を再現して修正する能力、テストからフィードバックを収集する能力、テスト品質を改善する能力、テストランを迅速に繰り返す能力も、自動テストに結び付いています。

これは昨年の調査とどこが違っていて、何を意味しているのでしょうか。

昨年は単に自動テストと CI の重要性を評価しただけで、両者の関係については調査しませんでした。今年は、自動テストが CI の改善を推進していることがわかりました。これは、自動テストスイートの構築にスマートに投資すると CI が改善することを意味します。

CI が CD を改善することも再確認されました。CI が効果をもたらすには、コードがコミットされるたびにソフトウェアが正常にビルドされ、一連のテストスイートが実行される必要があります。また、プロジェクトの自動ビルドと自動テストが毎日正常に行われる必要があります。

さらに、デプロイの自動化、トランクベース開発[17]、モニタリングが CD に影響を与えることも再確認されました。これらの能力は、デプロイの自動化で説明したように、組織レベルの取り組みに依存する場合があります。たとえば、チームは自分たちが担当するコードをモニタリングすることはできますが、アプリケーションとインフラストラクチャの両方がモニタリングされておらず、意思決定に使用されていない場合、完全なメリットは得られません。

17 DORA の調査によると、効果的なトランクベース開発は、アクティブなブランチの数が 3 つ未満で、マスターにマージするまでのブランチとフォークのライフタイムが 1 日未満という特徴を持ちます。

## 組織レベルの技術的能力

すばやく効果を得るためにチームレベルで実装、実施できる能力に対し、一部の能力は組織レベルの連携と支援があって初めて有効に機能します。この種の能力の例としては、アーキテクチャやポリシーといった複数のチームに関わる決定や設計を伴うもの (例: 変更管理) が挙げられます。

今年の調査では、**疎結合アーキテクチャ**が CD に対してプラスの影響を与えることが再確認されました。疎結合なアーキテクチャにおいては、デリバリー チームが、他のチームによる追加のサポート、サービス、リソース、承認に頼らずに、システムのテスト、デプロイ、変更を必要なときに自分たちだけで行うことができ、行ったり来たりのやり取りも少なくなります。これにより、チームは価値を速やかに提供できますが、より高いレベルでの連携が必要となります。

# オープンソース ソフトウェア開発

本調査は、組織的文脈におけるソフトウェアの開発とデリバリーに主眼を置いています。そこでは、チームがフルタイムでソフトウェアの開発とデリバリーに取り組み、メンバーは厳しいチェックイン スケジュールとリリース スケジュールの中で開発とリリースを連携させています。トランクに頻繁にチェックインするトランクベース開発と本番環境へのデプロイはパフォーマンスにおける成果を予測させるものであることが明らかになっています。

しかし、オープンソース ソフトウェア開発はどうなのでしょうか。

オープンソース プロジェクトは大部分がコミュニティ主導で、世界中のコミュニティ メンバーが各自の可能なスケジュールでパッチをプロジェクトに送信するため、タイムラインは複数存在し、期待も一定ではありません。オープンソース プロジェクトは世界中の人々や多くの組織 (あらゆるレベルのフリーランサー、ホビイスト、開発者が含まれる) との共同作業を支援する必要があるため、オープンソース プロジェクトのリリースは、継続的デリバリーのソフトウェア プラクティスとは異なるスタイルで切り出されます。通常は、重要なテストが完了した後の特定の時点でブランチからリリースが切り出されます。

DORA の調査結果は、一部のエリアではオープンソース開発にも及んでいます。

- コードを早くコミットするほど良い: 多くのオープンソース プロジェクトで、リベースを防ぐためパッチをより迅速にマージするほど開発者の動きも速くなることが認められています。
- 作業するバッチは小さい方が良い: 大きな「パッチ爆弾」は、変更のレビューに要する時間が長くなるため、小さくて読みやすいパッチセットと比較してプロジェクトへのマージが難しく、時間もかかります。

クローズソースのコードベースであるか、オープンソース プロジェクトであるかを問わず、短命のブランチ、小さくて読みやすいパッチ、変更の自動テストを使用する方が、全体的な生産性はより改善します。

この戦略を可能にするアーキテクチャ的アプローチとしては、範囲限定的なコンテキストと API を使用して大きな影響範囲を分離する方法が挙げられます。そうすれば、各ユニットがより小さく、ユニット間の結合が緩いものになります。サービス志向のアーキテクチャは、真のマイクロサービスアーキテクチャがそうであるように、このような成果を実現するはずです。[18] サービスのテストとデプロイを単独で行えるようにアーキテクチャを設計すると、チームのパフォーマンスが改善する助けとなります。

今日のシステムを実行するには大量のコードが必要です。たとえば、Facebook のコード行数は 6,200 万行（バックエンド コードを除く）、Android オペレーティング システムは 1,200 ～1,500 万行、標準的な iPhone アプリは 50,000 行です。[19] 組織を運営するために膨大なコードが必要となる現在、どのようなコード関連プラクティスが本当にパフォーマンスを推進するのか（あるいは、そもそもコード関連プラクティスにそのような効果があるのか）調査することには意味があります。本調査では、この構成概念のことを**コードのメンテナンス性**と呼んでいます。

分析の結果、コードのメンテナンス性は CD の成功に積極的に貢献することがわかりました。コードのメンテナンス性を良好に運用しているチームは、それに適したシステムやツールを持っています。そこには、他のチームが管理しているコードの変更、コードベースでの具体例の検索、他者のコードの再利用を開発者が簡単に行える機能があり、新しいバージョンの依存関係の追加、アップグレード、移行をコードを壊さずに行うこともできます。このようなシステムやツールが用意されていれば、CD に貢献するだけでなく、技術的負債の低減にも役立ち、ひいては生産性の向上にもつながります（これについては後のセクションで論じます）。

18 新しいシステムを早い段階でサービスに分解すること、または過度に細かいサービスに分けることは避けてください。どちらもデリバリー パフォーマンスを阻害する可能性があります。これは、Martin Fowler 氏が MonolithFirst（https://martinfowler.com/bliki/MonolithFirst.html）で取り上げています。

19 https://www.businessinsider.com/how-many-lines-of-code-it-takes-to-run-different-software-2017-2

コードのメンテナンス性の高い組織は、エンジニアに実際的な利点をもたらします。たとえば、依存関係を管理するのは大変です。ある依存関係を更新したことで、API 変更の破壊、中間的な依存関係の更新、矛盾する依存関係の発生（たとえば、菱形依存関係の問題）、機能性の破壊といった問題につながる穴が開く可能性があります。こうしたエラーを防ぐツールやコード変更の結果を視覚的に示すツールがあると、エンジニアはより良い設計を選ぶことができ、コードの品質も改善します。

ツールやコード編成については議論に発展しがちです。「ソフトウェアのパフォーマンスと生産性を向上させようとしているのか、それとも阻止しようとしているのか」という結果に焦点を合わせることが重要となります。

# 便利で使いやすいソフトウェア デプロイツール

開発者、テスター、システム管理者などの高度なユーザーはこれまで、ツールのユーザビリティを検討するときには無視されてきました。経営管理者は、関連技術のエキスパートであるテクノロジストは当然与えられたすべてのツールを理解できるものと決めてかかることがあります。これは珍しい考え方ではありません。第二次世界大戦中、パイロットは過度に複雑なコックピットを操縦する能力に基づいて選抜され、訓練されました。当時のユーザビリティのエキスパートは、航空機の操縦のような複雑な作業は十分に難しいことを認識するに至りました。コックピットを使いやすく理解しやすいように設計し、パイロットが航空機を安全に操縦することに集中できるようにすることが得策だったのです。

時が変わって現在、ユーザビリティのニーズは無視されています。これは、経営管理者がテクノロジストのニーズは通常のエンドユーザーのニーズと同じようなものと考えているためです。今日私たちは、エンジニアなどのパワーユーザーはしばしば特殊なユースケースと独自の設計ニーズを持つことを知っています。また、テクノロジストは、そのスキルセットや経歴（UX、情報セキュリティ、データベース エンジニアなど）が幅広く、能力も多種多様です。ツールを使いやすくアクセスしやすいものにすることは、ツールベンダーにとって重要な事柄です。

これを念頭に、今年の調査ではソフトウェアのデプロイに使用されているツールのユーザビリティについて調べました。ソフトウェアの開発とデプロイをサポートする技術的プラクティスは速度と安定性にとって重要だからです。

このデプロイツールの有用性と使いやすさは CI および CD と高い相関関係にありました。使用するツールが仕事に適しているほど成果は改善するため、これは理に適っています。また、これは生産性の推進にもつながることがわかりました。これについては 55 ページをご覧ください。

# 障害復旧テスト

ミッション クリティカルなソフトウェア システムを運用している組織にとって、障害復旧計画は不可欠です。しかし、計画を作成してもそれをテストしなければ、バックアップをとるだけでそこから定期的に復元する練習をしないのと同じようなもので、役に立ちません。本調査では、回答者にどのような種類の**障害復旧テスト**を実施しているかを尋ねました。

## パフォーマンス プロファイル別の障害復旧テストの種類

| | ロー | ミディアム | ハイ | エリート | 全体 |
|---|---|---|---|---|---|
| 実際のシステムでは実施しない机上訓練 | 35% | 26% | 27% | 30% | 28% |
| インフラストラクチャ（データセンターを含む）のフェイルオーバー | 27% | 43% | 34% | 38% | 38% |
| アプリケーションのフェイルオーバー | 25% | 46% | 41% | 49% | 43% |
| 本番環境を模したテストシステムを使用したシステム途絶のシミュレーション（ネットワーク リンクの機能低下やルーターの電源を落とすなどの障害注入を含む） | 18% | 22% | 23% | 29% | 23% |
| 本番環境システムを使用したシステム途絶のシミュレーション（ネットワーク リンクの機能低下やルーターの電源を落とすなどの障害注入を含む） | 18% | 11% | 12% | 13% | 12% |
| 定期的に本番環境システムを途絶させる自動化された仕組みおよびシステムの作成 | 9% | 8% | 7% | 9% | 8% |

必ずしもすべてのテストが本番環境システムで行われているわけではありません。これは次の 2 つの理由から不安材料となります。1 つ目は、本番環境システムを包括的に再現することは難しい（そして、法外な費用がかかる場合が多い）ためです。2 つ目は、本番環境システムを停止させるようなインシデントは、一見正常なパラメータ範囲内で動作しているように見えるコンポーネント間の相互作用が原因で発生することが多いためです。これはテスト環境で再現できない場合があります。システムが複雑化している中、これらの要因はますます重要になっています。

次の障害復旧テストについて、本番環境インフラストラクチャに対するテストをどの程度の頻度で実施しているかを尋ねました。

- 本番環境システムを使用したシステム途絶のシミュレーション（ネットワーク リンクの機能低下やルーターの電源を落とすなどの障害注入を含む）
- インフラストラクチャ（データセンターを含む）のフェイルオーバー
- アプリケーションのフェイルオーバー

これらのいずれかまたは複数の方法を使用して少なくとも年 1 回**障害復旧テスト**を実施していると答えた回答者は 40% にとどまりました。障害復旧テストを実施している組織は、サービス可用性（テクノロジー チームと組織の、運用中のソフトウェア プロダクトまたはサービスに関する約束や表明を果たす能力）のレベルがより高いという結果になりました。

Capital One の安定性および SRE 担当バイス プレジデントである Mike Garcia 氏は、次のように述べています。「今日は、最新のクラウド テクノロジーで迅速に提供する能力があることを実証するだけでは十分ではありません。特に銀行のような規制の厳しい業界では、お客様のニーズを満たす責任において自社の復元力のレベルを証明しなければならないという義務があります。... そのために、私たちは単に一括してフェイルオーバーできることを実証するだけでなく、その先へと進まなければなりませんでした... その考えは、より複雑なカオステスト手法を通じて高度な機能と自動的な復元力を継続的に示すというものです。」

部門や組織の垣根を越えて協力しながら障害復旧の訓練を行う場合、それによってもたらされる改善効果は単にシステムだけにとどまりません。このようなテストは多くのチームの協力が必要となり、多くの人が忘れている、または認識していない結び付きが存在することを明るみに出すため、テストするシステムを取り巻くプロセスやコミュニケーションの改善と強化につながり、それによってテストの効率と効果がより一層増します。

また、組織が障害復旧テストの訓練で得た知見に従って行動しているかどうかについても調べました。分析結果から、障害復旧の訓練から学んだことを基にアクション アイテムを作成、実施している組織はエリート パフォーマー グループに属する可能性が 1.4 倍高いことがわかりました。

## 障害復旧訓練から学ぶ

批判なしの事後分析は、失敗から学んで成長の糧とするために重要な側面です。これは文献で十分に立証されており、DORA のこれまでの調査でも裏付けられています。それによると、批判なしの事後分析は、学びの文化を育み、ソフトウェアのパフォーマンスと組織的パフォーマンスの成果を最適化する組織文化の構築に貢献します。

障害復旧訓練に関する素晴らしい論説として、Kripa Krishnan 氏と Tom Limoncelli 氏による「Weathering the Unexpected」があります。これは費用面を踏まえた障害復旧訓練の利点という観点だけでなく、実践半ばでの SRE における経過説明においても優れています。[20]

この ACM Queue 記事の中で、Kripa Krishnan 氏（以前に障害復旧テスト（DiRT）を担当していた Google のディレクター）は次のような見解を述べています。「DiRT 方式のイベントを成功させるには、まずシステムやプロセスの障害を学習の手段として受け入れる必要があります。仕事に失敗はつきものです。失敗が起こったときになすべきことは、複雑なシステムの障害について個人やチームを叱責することではなく、誤りを正すことです。」

20 https://queue.acm.org/detail.cfm?id=2371516

# 変更管理

ソフトウェアや本番環境システムに変更を加えることは、たいていの場合複雑で、簡単には進みません。この複雑さの大部分は 2 つの要因に起因します。それはチーム間の連携が必要であることと、規制要件です。後者は特に金融サービス、医療、政府機関に関係します。規制要件の実施に伴う複雑さはリーダーや実務担当者の手に及びませんが、変更管理においてチームの連携が果たす役割に影響を及ぼし、その役割を変えることは可能です。

たとえば、職掌分散（変更者以外の誰かが変更を承認しなければならないことを示す）が規制の枠組みによって義務付けられていることがよくあります。[21] 1 人の個人があるプロセスを最初から最後までコントロールすべきでないこと（この要件の意図）については同意しますが、高負荷のアプローチで見られる連携の負担をかけずに、負荷の低い安全な手段でこの目的を達成することは可能です。

21 たとえば、PCI データ セキュリティ スタンダード（PCI-DSS）や、米国連邦政府機関が使用する NIST リスク管理フレームワークなどがあります。

1 つのアプローチは、(ペア プログラミングの一環として) バージョン管理にコミットする前、またはマスターにマージする前に、コードレビューの一
環としてすべての変更をチームの他のメンバーが承認するという方法です。

これは、自動しきい値による変更の制限と組み合わせることができます。たとえば、変更をプッシュした結果、コンピューティングまたはストレージのコストがある一定のしきい値を超える場合、(ピアレビューで承認されたとしても) 開発者がその変更をプッシュできないようにするチェック処理を実装できます。このように低負荷で実施しやすいプロセスは、実務担当者に変更管理を改善する明白な機会を与えます。IT サービス マネジメント (ITSM) フレームワークに賛同する人たちは、正式な変更承認プロセスは安定性の向上につながると主張し、数十年もの間業界の標準であり続けていることを強調します。それに対して、リーンでアジャイルな理念に支えられた人たちは、より合理化された変更承認の方が迅速な

## 職掌分散の実施

コードレビューを使用して職掌分散を果たすためには、本番環境システムに対するすべての変更について、変更内容とその担当者、本人認証された承認イベントを変更管理システムに記録する必要があります。システムに対するすべての変更の信頼できる情報源としてバージョン管理を使用する場合は (「Infrastructure as Code」、「GitOps」と呼ばれるパラダイムの手法)、それぞれの変更を誰が承認したかを記録できるだけでかまいません。これには、レビューと提出のプロセスが高度に監査可能になるという追加の利点があります。前回の調査では、バージョン管理をシステム構成やスクリプトも含めて包括的に使用するとパフォーマンスが推進されることも示されました。

フィードバック、情報フローの向上、より良い成果の達成につながると主張しています。これらの仮説を検証するため、新たに 2 つの構成概念を作成し（1 つは低負荷で明確に理解できる変更承認プロセスを実施するもの、もう 1 つはより正式で**高負荷の変更承認プロセス**を実施するもの）、それぞれの構成概念のソフトウェア デリバリー パフォーマンスに及ぼす影響を調べました。

## 高負荷の変更プロセス

重要な変更に対して変更審議委員会（CAB）などの外部組織やシニア マネージャーの承認を必要とするフォーマルな変更管理プロセスは、ソフトウェア デリバリー パフォーマンスにマイナスの影響を与えることがわかりました。所属する組織がこの種のフォーマルな承認プロセスを導入している回答者は、ロー パフォーマーに属している割合が 2.6 倍高いという結果になりました。これは前回の調査結果を補強するもので、前回は高負荷の変更承認プロセスと変更失敗率との間にマイナスの相関関係がありました。[22]

ITSM フレームワークによって提案された高負荷の変更管理プロセスの背後にある動機は、リリースのリスクを軽減します。これについて調査するため、フォーマルな承認プロセスが変更失敗率の低下に関連しているかどうかを調べたところ、以前の調査と同様に、この仮説を支持するエビデンスは見つかりませんでした。[23] さらに、より多くの承認を導入することが、プロセスの遅延とより大規模で頻度の少ないバッチのリリース（これらは本番環境システムにより大きな影響を及ぼします）につながるかどうかも調べました。これはリスクレベルの上昇、ひいては変更失敗率の増加に関連している可能性が高いといえます。私たちの仮説はデータで裏付けられました。これは、リリース プロセスのリスクを軽減しようとしている組織にとって重要な意味を持ちます。ソフトウェアのリリースに関する問題に対処するとき、追加のプロセスを導入してより多くの承認を求めるという方法

22、23 Velasquez, N., Kim, G., Kersten, N., & Humble, J. (2014). State of DevOps Report: 2014. Puppet Labs.

で解決を図ろうとする場合がよくありますが、本調査の分析によると、このアプローチは事態のさらなる悪化を招きます。

外部の変更承認はパフォーマンスにマイナスの影響を与えるため、そのようなプロセスから脱却することを推奨します。その代わりに、開発プロセス中にピアレビューで承認する方法に「シフトレフト」してください。ピアレビューに加えて自動化を活用することで、デリバリー ライフサイクルの早期に不適切な変更を検出、防止、修正することもできます。継続的テスト、継続的インテグレーション、包括的なモニタリングと可観測性などの手法は、早期の自動化された検出、可視性、迅速なフィードバックを可能にします。このようにして、正式なレビューを待つ場合よりも速やかに誤りを修正できます。

## 明確な変更プロセス

最終的な目標は従来の正式な変更管理プロセスから脱却することですが、単に既存プロセスのやり取りを改善し、チームを効率的に舵取りするだけでも、ソフトウェア デリバリー パフォー

# 継続的デリバリー パラダイムにおける CAB の役割

継続的デリバリーによるリスク管理アプローチは従来の変更管理プロセスよりも優れていますが、それでも CAB には重要な役割があります。組織が複雑化するにつれて、チーム間のコミュニケーションと連携を改善することはますます重要になります。前回の調査では、ミッションを重視した高パフォーマンスの文化を育成するには情報フローを促進することが不可欠であることがわかりました。継続的パラダイムのプラクティスでは、CAB が個々の変更を承認することはありません。その代わりに、CAB はソフトウェア デリバリーのパフォーマンスを向上させようとするチームのプロセス改善の取り組みを支援することに尽力する必要があります。これは、ガイダンスやリソースを提供することでチームがパフォーマンスを推進する能力を実現できるよう手助けする、という形をとることができます。また、CAB は、ビジネスの高いレベルでのトレードオフやサインオフが必要となる重要なビジネス上の決定（市場投入とビジネスリスクとの比較検討など）に関与することもできます。

この CAB の新しい役割は戦略的です。詳細なコードレビューを実務担当者と自動化された手法に移管することで、リーダーや経営管理者の手が空き、その分をより戦略的な仕事に向けることができます。この門番からプロセス アーキテクトおよび情報ビーコンへの役割の変化は、社内の変更管理機構が向かうべき道であり、ソフトウェア デリバリー パフォーマンスに秀でた組織のプラクティスとも一致します。

マンスにプラスの影響が働きます。チームメンバーが変更の実装について承認を得るプロセスを明確に理解している場合、これはパフォーマンスの向上を促進する要因となります。これは、チームメンバーが通常行うすべての種類の変更について毎回「提出」から「承認」までどのようなステップが必要であるかを知っていて、承認プロセスを通じてタイムリーに変更許可を得られる自信を持っているということを意味します。**明確な変更プロセス**があると答えた回答者は、エリート パフォーマーに属している割合が 1.8 倍高いという結果になりました。

大規模な組織と仕事をするとき、変更管理は常に大きな制約の一つです。この制約を取り除くためには、複数レベルでの取り組みが必要となります。チームは、継続的インテグレーション、継続的テスト、ピアレビューによって不適切な変更をできるだけ早く検出するとともに、職掌分散も実現することができます。さらに、技術的な実務担当者だけが、本レポートで提案した変更管理ソリューションを構築、自動化する力を持ち、ソリューションを迅速、高信頼、繰り返し可能、監査可能なものにすることができます。

外部委員会を変更承認の門番とするフォーマルな承認プロセスから脱却し、ガバナンスと能力開発の役割へ移行させることが、あらゆるレベルのリーダーに求められています。結局のところ、組織ポリシーのある特定のレベルに影響を及ぼし、それを変更する権限を持つのはマネージャーだけです。技術的な実務担当者と組織のリーダーが協力した結果、ほんの数か月でパフォーマンス（スループット、安定性、可用性）が劇的に改善されたケースもあります。

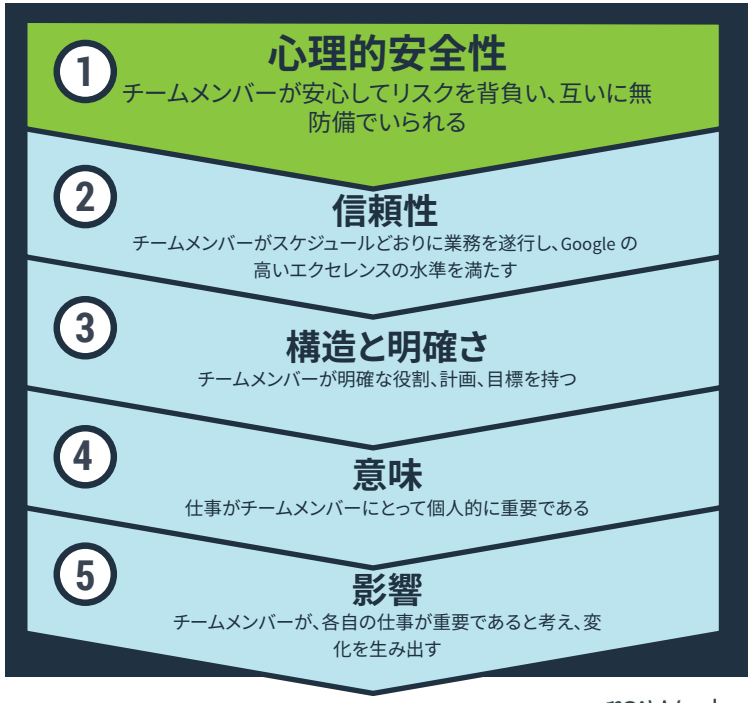
# 心理的安全性の文化

文化はしばしば、組織の取り組みを先導する実務担当者や推進者が DevOps と技術革新を成し遂げるための鍵と言われます。実際、 Davis 氏と Daniels 氏は著書『Effective DevOps』において、テクノロジーに関するスケーラブルな取り組みを成功させるための重要な要因として文化を挙げています。[24] DORA 独自の調査では、情報フロー、信頼、イノベーション、リスク共有を最適化する組織文化は SDO パフォーマンスを予測させるものであることを示しています。[25]

Google による 2 年間の大規模な調査[26] でも、パフォーマンスの高いチームは信頼と心理的安全性の文化、意味のある仕事、明確さを必要としているという同じような結果が得られています。このようなチーム環境があると、メンバーのリスクは予測された中程度のものとなり、メンバー間の率直なコミュニケーションが促進され、メンバーがより創造的になります。こうした結果が Google 以外にも当てはまるのか疑問を抱く人もいるでしょう。この種の文化は、企業の規模、ツール、エンジニアリング スキルがさまざまに異なる他の組織にも有益なのでしょうか。それともこれは、大企業に所属し、巨大なインフラストラクチャとある特定の種類のコードベースだけに支えられ、厳密で名高い Google のインタビューに合格したエンジニアだけに当てはまることなのでしょうか。

24 Davis, J. および Daniels, R. 共著 (2016 年)、『Effective DevOps: building a culture of collaboration, affinity, and tooling at scale』、O'Reilly Media, Inc.
25 この調査は、社会学者の Ron Westrum 博士が最初に提唱した組織文化フレームワークに基づいています。このモデルの概略については、「2018 Accelerate State of DevOps Report」をご覧ください。
26 https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/

この疑問を解き明かすため、本調査では文化の尺度を Project Aristotle チームが調査に使用した質問に合わせました。[27] DORA の分析では、この**心理的安全性の文化**はソフトウェア デリバリー パフォーマンス、組織的パフォーマンス、生産性を予測させるものであることが示されました。測定された成果は Project Aristotle と異なりますが、これらの結果は、信頼と心理的安全性の文化、意味のある仕事、明確さは Google 以外のチームにも大きな利益を与えることを示します。




re:Work

27 調査手段を提供していただいた Project Aristotle チームに深く感謝いたします。

# テクノロジーが組織的パフォーマンスに及ぼす影響

ソフトウェアを迅速かつ確実にデリバリーし、高いレベルの可用性を提供する能力は、強力なツールです。これにより、組織は新しい製品や機能のプロトタイプを短期間で簡単に作成し、既存ユーザーに影響を与えずにユーザーに対する効果をテストできます。また、コンプライアンスや規制の変更に遅れずに対応し、セキュリティ上不可欠なソフトウェア パッチやアップデートを迅速かつ確実に提供することもできます。これらを効果的に活用した場合、高いレベルの SDO パフォーマンスを達成できる組織は、技術変化や市場のシフトに的確に対応し、優れたプロダクトやサービスを生み出すことができます。これはさらに、商業的、非商業的の両面での望ましい組織的成果の達成につながります。**今年の分析結果は、エリート パフォーマーは組織的パフォーマンスの目標を達成する可能性が 2 倍あることを示しています。**

本調査で使用した組織的パフォーマンスの尺度は学術的文献から得たもので、次のような商業的[28] 目標と非商業的[29] 目標の両方を捕捉します。

- 収益性
- 生産性
- マーケット シェア
- 顧客の数
- 製品またはサービスの質
- 運用効率
- 顧客満足度
- 提供された製品またはサービスの質
- 組織または任務の目標の達成

昨年と同様に、2 次構成概念の SDO パフォーマンスは組織的パフォーマンスを予測させ、ソフトウェア デリバリー パフォーマンスまたは可用性を単独で使用した場合よりも高い予測性を示しました。

28 Widener, S. K. (2007). An empirical analysis of the levers of control framework. Accounting, Organizations and Society, 32(7-8), 757-788.

29 Cavalluzzo, K. S., & Ittner, C. D. (2004). Implementing performance measurement innovations: evidence from government. Accounting, Organizations and Society, 29(3-4), 243-267.

# 生産性を改善するには

チームと組織のもう 1 つの重要な目標は、変革の効果を高め、従業員からより多くの価値を引き出すために、生産性を改善することです。生産性は今年から調査を開始したテーマで、組織がツールと情報へのスマートな投資によって生産性向上をどのように支援できるか、技術的負債によって生産性がどのように阻害されるか、生産性は従業員のワークライフ バランスとバーンアウトにどのように影響するかを調べました。

# 生産性の改善

生産性が重要なのは誰もが認めるところです。生産性の高いエンジニアは自分の仕事をより効率的に行うことができ、多くの時間をドキュメントの作成やリファクタリングといった他の仕事に当てたり[30]、主要な業務により力を入れてさらなる機能の提供やインフラストラクチャの増強を果たしたりすることができます。[31]

しかし、生産性とはどのようなもので、どのようにして測定すればいいのでしょうか。生産性は、コード行数、ストーリー ポイント、解決したバグなどの単純な指標で捕捉することはできません。単純な指標を使用すると、チーム全体の目標を予期せず損なう結果となります。[32] たとえば、チームの他のメンバーを手助けすると業務速度に悪影響が及ぶため、たとえその手助けが組織の目標を達成するために重要であっても、メンバーが他者を手助けすることを拒む可能性があります。このトピックは長きにわたって議

30 著名な経済学者の Hal Varian 氏による生産性の考察が参考になります。https://www.wsj.com/articles/silicon-valley-doesnt-believe-u-s-productivity-is-down-1437100700

31 生産性向上によって節約された時間を他の仕事に使うことの利点（それらをコスト節約とみなし、不要な作業はなくすべきとする考え方と比較した場合の利点）については、2017 年の ROI ホワイトペーパー「Forecasting the Value of DevOps Transformations」で論じました。これは新しい考えではなく、経済学者がすでに研究しており、Varian 氏も上記の WSJ 記事で取り上げています。cloud.google.com/devops

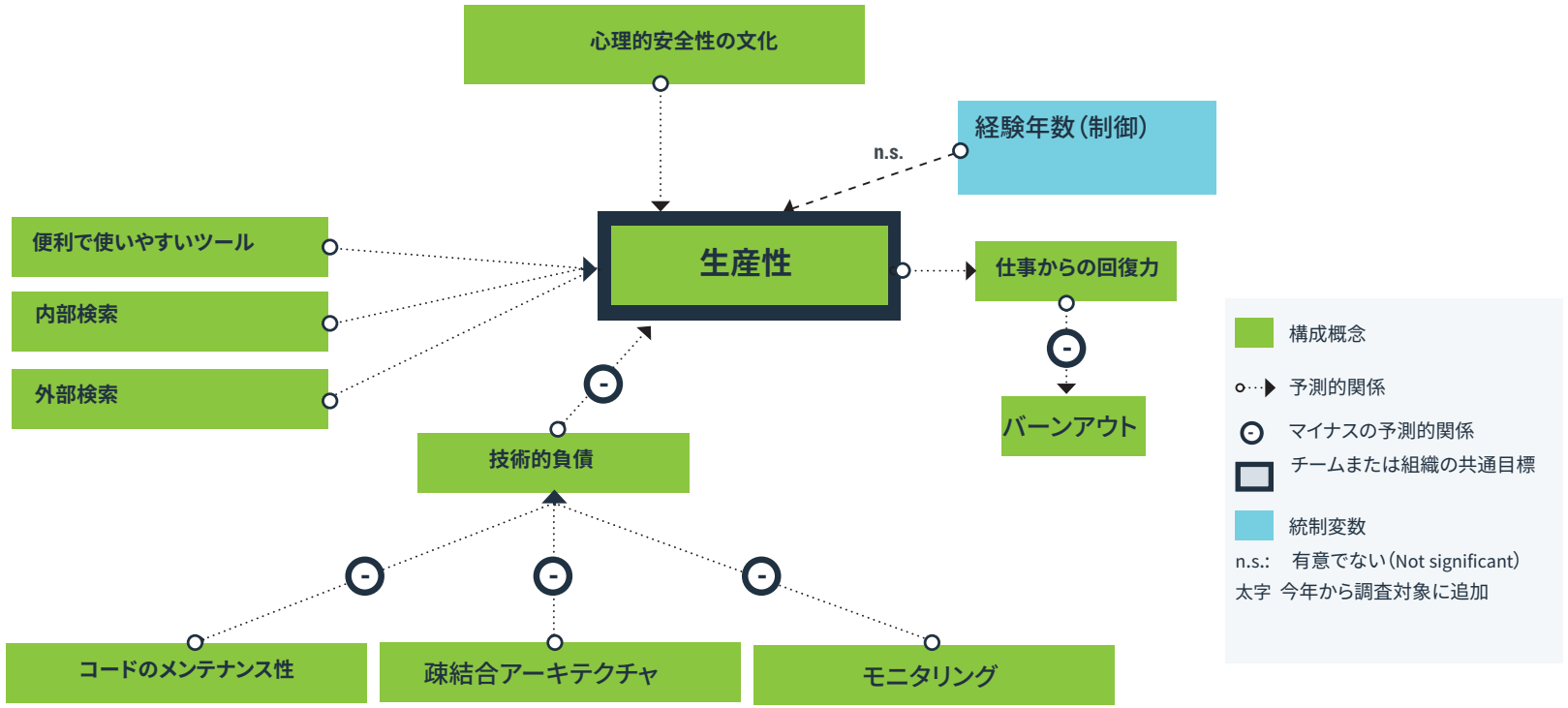
32 これはグッドハートの法則の一例ということもできます。この法則は、「計測結果が目標になると、その計測結果自体が役に立たなくなる」というものです。測定しやすいローカルな指標だけに目を向けることで、それらをチームの目標に設定してしまう場合があり、その結果、組織的成果を真に推進するグローバルな目標が失われます。詳細については、『Accelerate: The Science of Lean Software and DevOps』の第 2 章をご覧ください。

論が交わされており、ほとんどの研究者は次のような同じ結論に達しています。

**生産性とは、複雑で時間のかかる仕事を、集中を妨げるものと中断を最小限に抑えつつ完了させる能力である。**

私たちの多くはこれを「仕事の流れまたはリズムが良い」と表現します。

このモデルを使用するには、次の図から改善する目標を定め、それに影響を与える能力を特定します。たとえば、技術的負債の低減を目標とする場合、これに影響を与える能力はコードのメンテナンス性、疎結合アーキテクチャ、モニタリングです。



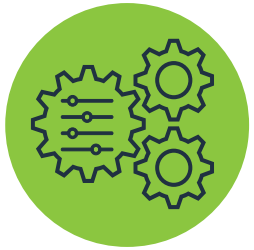
# 便利で使いやすいツール

**便利で使いやすいツール**は今や消費者向け技術にとってなくてはならないものですが、この明白な特徴は、自らが専門家であると決めてかかり、どのようなツールや技術でも使いこなせるテクノロジー プロフェッショナルの間では見過ごされることが往々にしてあります（あるいは、そのようなグループ向けのツールを購入する人が、テクノロジストにとってユーザビリティはさほど重要でないと考えているか、ユーザビリティ以外のコスト、ライセンス条件、ベンダー管理などの要素について最適化しようとしていることが原因である場合もあります）。実際、その逆のことも言えます。つまり、複雑なシステムを構築するときや、ビジネスに不可欠なインフラストラクチャを管理するときには、仕事がより難しくなるため、ツールの重要性が一層増します。

本調査で、ソフトウェアのデプロイに使用される CI / CD およびテスト自動化ツールチェーン内のツールに注目したのは、それらが DevOps の中核にあるためです。次の 2 つの特性が生産性を推進する要因であることがわかりました。

- ツールチェーンはどれくらい使いやすいか（操作や運用のわかりやすさと容易さを含む）
- ツールチェーンはジョブ関連の目標を達成するうえでどれくらい役に立つか

**パフォーマンスが最も高いエンジニア**



**1.5**

**倍高い**

使いやすいツールを持っている可能性

# パフォーマンス プロファイル別のツール使用率

| | ロー | ミディアム | ハイ | エリート |
|---|---|---|---|---|
| 独自ツール、オープンソース、商用オフザシェルフ（COTS）ソフトウェアの組み合わせ | 30% | 34% | 32% | 33% |
| 主にオープンソースとCOTS、大幅にカスタマイズ | 17% | 8% | 7% | 10% |
| 主にオープンソースとCOTS、カスタマイズはほとんどなし | 14% | 21% | 18% | 20% |
| 主に COTS パッケージ ソフトウェア | 8% | 12% | 8% | 4% |
| 主に社内で開発された、所属組織専用のもの | 20% | 6% | 5% | 6% |
| 主にオープンソース、大幅にカスタマイズ | 6% | 7% | 5% | 12% |
| 主にオープンソース、カスタマイズはほとんどなし | 5% | 12% | 24% | 15% |

これらのツールはどのようなものでしょうか。データをパフォーマンス プロファイル別に掘り下げてみたところ、興味深いパターンが確認されました。

- 完全に独自のソフトウェアの使用率が最も高かったのはロー パフォーマーで、最も低いのはハイ パフォーマーとエリート パフォーマーです。独自ソフトウェアは有益な場合もありますが、メンテナンスとサポートに多大なコストがかかります。最高レベルのパフォーマーがこのモデルから手を引いたのは自然なことと言えます。

- ほとんどカスタマイズされていない商用オフザシェルフ（COTS）ソフトウェアについては、使用率が比較的均一になっています。ハイ パフォーマーが COTS を使用していながら高いパフォーマンスを実現できることを不思議に思う方もいるかもしれません。特に、「software is eating the world（ソフトウェアが世界を飲み込んでいる）」という状況が続き、独自ソフトウェアの開発は必須であるにもかかわらずです。Martin Fowler 氏が指摘したように[33]、企業はどの

# パフォーマンス プロファイル別の自動化と統合

| | ロー | ミディアム | ハイ | エリート |
|---|---|---|---|---|
| 自動ビルド | 64% | 81% | 91% | 92% |
| 自動単体テスト | 57% | 66% | 84% | 87% |
| 自動受け入れテスト | 28% | 38% | 48% | 58% |
| 自動パフォーマンス テスト | 18% | 23% | 18% | 28% |
| 自動セキュリティ テスト | 15% | 28% | 25% | 31% |
| 自動のプロビジョニングとテスト環境へのデプロイ | 39% | 54% | 68% | 72% |
| 本番環境への自動デプロイ | 17% | 38% | 60% | 69% |
| チャットボット / Slack との統合 | 29% | 33% | 24% | 69% |
| 本番環境モニタリング ツールおよび可観測性ツールとの統合 | 13% | 23% | 41% | 57% |
| いずれも使用していない | 9% | 14% | 5% | 4% |

ソフトウェアが戦略的で、どのソフトウェアが単なるユーティリティなのかを慎重に検討する必要があります。ハイ パフォーマーは自社のユーティリティ ニーズに COTS ソリューションで対応し、カスタマイズを最小限に抑えることで、戦略的ソフトウェアの開発作業にかかるリソースを節約しています。

また、エリート パフォーマーはより頻繁に、ほぼすべてのディメンションでツールを自動化しツールチェーンに統合しているのがわかります。自動化は実装にコストがかかりすぎると思われるかもしれませんが（「自動化する時間も予算もない、自動化は単なる一機能ではない」という声はよく耳にします）、実のところ自動化は真に健全な投資と言えます。[34] 自動化によってエンジニアは手作業に費やす時間を減らすことができるため、新規開発、リファクタリング、設計、ドキュメント作成など、他の重要な仕事に時間をかけられるようになります。また、ツールチェーンに対するエンジニアの信頼度が高まるため、変化を推し進めるうえでのストレスを軽減できます。

33 執筆者 Martin Fowler 氏、MartinFowler.com、UtilityVsStrategicDichotomy。https://martinfowler.com/bliki/UtilityVsStrategicDichotomy.html
34 これはサイト信頼性エンジニアリング（SRE）のベスト プラクティス、「生産性のない業務であるトイルを削減する」というものです。

## 技術プロフェッショナルとツール

2017 年の調査では、ツールや実装について独自の決定を行う権限を与えられたチームは、ソフトウェア デリバリー パフォーマンスの向上に貢献していることがわかりました。今年の調査結果は、ハイ パフォーマーはその機会さえあれば便利で有用なツールを選択すること、そしてそうしたツールは生産性を高めることを示しています。

これは製品の設計において重要な意味を持ちます。実用性と使いやすさを兼ね備えた製品は技術プロフェッショナルに採用されやすく、それらを使用した場合の成果も優れています。業界のリーダーは、こうしたツールを優先的に使用する必要があります。デリバリーするプロダクトは機能が完全なだけでは十分でなく、採用されるだけの使いやすさを兼ね備えており、 DevOps 変革の中で価値を提供できなければなりません。

# 生産性、バーンアウト、業務のやりくり

一方、パフォーマンスの最も高いグループと最も低いグループでは、業務のやりくりの量が大きく異なるのかどうかも関心の対象でした。生産性とは結局のところ、「良い流れ」に乗りながら仕事をやり遂げる能力のことだからです。

これを確認するため、回答者に次のような質問をしました。

- 正式な役職名にかかわらず、いくつの役割をこなしているか、またはいくつの異なる種類の業務を担当しているか
- 1 日のうちにいくつのプロジェクトに対応したか
- 全体としていくつのプロジェクトに携わっているか

驚いたことに、これについてはロー、ミディアム、ハイ、エリートの各パフォーマーの間で大きな相違は見られませんでした。そのため、チームがソフトウェアをいかに効果的に開発しデリバリーしているかということが、回答者の担当している役割やプロジェクトの数に影響を与えるとは言えません。「この段階さえやり遂げてしまえば大きく改善する」などということはありません。そうではなく、業務は手順を踏んで持続可能にしていく必要があります。それはプロセス改善の取り組みと自動化によって成し遂げられるものです。これにより、トイルが削減され、業務は反復可能で一貫性があり、迅速かつスケーラブルで監査も可能なものになります。また、新しい創造的な仕事をするための時間も生まれます。

# 内部検索と外部検索

問題の解決やエラーのデバッグに役立つ正しい情報を見つけ出し、類似した解決策をすばやく簡単に見つけることは、作業を完了させ業務の流れを維持するための鍵となる場合があります。これは特に、かつてなく複雑なシステムで構成されている昨今の技術環境に当てはまります。情報ソースにアクセスできることは、生産性向上に資することがわかっています。こうした情報ソースには内部検索と外部検索という 2 つのカテゴリがあります。

- **内部検索:** ドキュメントやコードの作成に加え、企業のナレッジベース、コード リポジトリ、チケット管理システム、その他ドキュメントの有効な検索を支えるための投資は、エンジニアリングの生産性向上につながります。社内のナレッジソースを使用した人は、高い生産性を生む可能性が 1.73 倍高いという結果になりました。開発者、システム管理者、サポートスタッフが社内リソースを検索できるようにすることで、その業務固有の状況に適した回答を (「類似の案件を探す」ような機能を使うなどして) 見つけ出し、解決策をより早く講じることができるようになります。また、十分にサポートされ強化されている社内ナレッジベースは、さらなる情報の共有と収集の機会を生み出します。たとえば、自分の状況にほぼ当

### 内部検索の使用

**1.73**

**倍高い**

高い生産性を生む可能性

### 外部検索の使用

**1.67**

**倍高い**

高い生産性を生む可能性

てはまる解決策を見つけた人は、なんらかの質問をする可能性があり、それに対して社内コミュニティからの回答や議論が得られ、それがナレッジベースに追加情報として加えられることが考えられます。組織があらゆる種類の情報やデータを簡単に検索できるシステムに投資していれば、その文化は知識共有という「好循環」につながる可能性があります。また一部の組織は、機械学習の技術を活用して、行われた内部検索に対し解決策の候補を特定して提案しています。

- **外部検索:** これには検索エンジンや Stack Overflow などの外部ソースが含まれます。分析の結果、外部検索を業務に利用した人は、業務の中で自身が生産的であると感じる可能性が 1.67 倍高いことがわかりました。外部検索が重要なのは、こうした技術によって学習と成長（および重要な二次的効果である求人）のための強固なコミュニティが形成され、パブリック クラウドやオープンソース ツールの利用と導入にあたって助けとなるからです。つまり、強固なユーザー コミュニティと優れたエコシステムを備えた人気のある外部のツールやシステムを活用することで、技術プロフェッショナルはトラブルシューティングを世界中の技術者とともにできるようになりますが、独自の自社製ツールを使用している場合は、得られる解決策のヒントが組織内のエキスパートからのものに限られてしまいます。これはデータによっても裏付けられています。 2018 年の調査では、エリート パフォーマーはオープンソースを活用しており、その使用を増やすことも計画していました。今年の調査の結果、エリート パフォーマーのオープンソース ツール使用率が上がっており、ローパフォーマーは独自ツールの使用率が最も高くなっていました（59 ページを参照）。こうした技術的選択は、必然的に生産性への影響をもたらします。

# 技術的負債

**技術的負債**とは、1992 年に Ward Cunningham 氏[35] が提唱した概念であり、彼が「未熟な」コードと呼ぶものを十分に保守し続けられなかった場合に起こることを説明した言葉です。

> 未熟なコードでも正常に機能し、顧客には完全に容認されるかもしれないが、量が多すぎるとプログラムが習得不可能になり、結果的にプログラマーの技能が極度に専門化して、最終的にはプロダクトの柔軟性が失われる。最初のコードを出荷することは、借金をしに行くのと同じである。小さな負債は、書き直しによって即座に返済される限り、開発を加速させる。危険なのは、借金が返済されない場合である。品質の良くないコードの使用に費やされる時間は、その 1 分 1 分を借金の利息としてとらえることができる。技術部門全体が、統合されていない実装という借金の重みで立ち往生する羽目になる可能性がある。

現在の複雑なシステムにおいてこの技術的負債は、スクリプト、構成ファイル、インフラストラクチャに加え、アプリケーション コードにも発生することがあります。技術的負債には、次のようなものを含むコードやシステムがあります。

- 新しい機能を優先するため未修正のままになっている既知のバグ
- 不十分なテスト範囲
- 低品質なコードや設計不良に関連する問題
- すでに不要になっていながらクリーンアップされていないコードまたはアーチファクト
- 現在のチームに専門知識がないため、効果的なデバッグまたはメンテナンスができない実装
- 不完全な移行
- 旧式の技術
- 不完全なまたは古くなったドキュメント、あるいはコメントの欠如

35 http://c2.com/doc/oopsla92.html

技術的負債は生産性に悪影響を与えることがわかっており、高い技術的負債を抱える回答者は生産性が低い割合が 1.6 倍で、最高レベルのパフォーマーは技術的負債が低い割合が 1.4 倍となりました。

## 技術的負債への対処

Ward Cunningham 氏は、「(変化するシステムへの) 最良の対処法は、プロダクトへのより完全な習熟とその実装である」と述べています。エンジニアは変化を理解できれば、それを受け入れることができます。昨今の複雑なインフラストラクチャと分散したシステムにより、エンジニアは「システム全体の状態」についてのメンタルモデルを維持できなくなっています。またこうした複雑なシステムをサポートすることで、プロフェッショナルの技能はより専門化し、システム全体に対する完全な習熟を見込めなくなっています。[36]

エンジニアがメンタルモデルを構築できるようにするには、柔軟で拡張可能かつ可視性の高いシステムを構築して技術的負債を減らす必要があります。

36 システムを完全に把握するには、フルスタックの開発者が必要となります。明白な定義上の問題はさて置き、フルスタックとは何でしょうか。チップから CSS までという意味でしょうか。業界の人々の多くは、フルスタックの開発者などというものは不可能であり望ましくもないという意見で一致しています。



# 技術的負債の積極的な低減

技術的負債に対処するだけでなく、実際にそれを減らすにはどうすればよいでしょうか。1 つのアプローチはリファクタリングです。リファクタリングとは、「既存のコード本体を再構築し、その外部動作を変更することなく内部構造を変化させるための訓練された手法」のことで、Martin Fowler 氏はリファクタリングを日常業務の一部とすべきだと指摘しています。堅牢なリファクタリングのサポートが組み込まれた優れたツールも重要です。実際、大規模な組織の多くが、コードベース全体にわたってコードのリファクタリングを実行するためのツールに投資しています。たとえば Facebook では、自社の fastmod ツールをオープンソース化し、Google では ClangMR をオープンソース化しています。

# 心理的安全性の文化

**心理的安全性**、信頼、敬意を重視する文化は、生産性の向上に役立ちます。従業員が問題の解決に専念し、社内政治や言い争いではなく業務の遂行に集中することができるからです。これは他の研究者による調査結果とも一致しています。前述のセクションで説明したように、Google による調査では、この種の文化はより有能なチームを生み出す原動力となることがわかっています。

# 生産性とオープンソース プロジェクト

生産性に関するこの結果は、オープンソース プロジェクトにも当てはまります。オープンソース プロジェクトに貢献する方法についてのドキュメントが最新かつ単純で、他のオープンソース プロジェクトと一貫していれば、コントリビューターがプロジェクトをゼロから開始して生産性の発揮するに至るまでの時間を短縮できます。独自の古い貢献プロセスが規定された大規模なプロジェクトでは、参加までの手順が複雑すぎるため、新しいコントリビューターを得ることが困難になります。複雑な貢献プロセスに技術的負債が加われば、コントリビューター候補者はプロジェクトを「読み取り専用」とみなすでしょう。ここで紹介したのとまったく同じベスト プラクティスをオープンソース プロジェクトに適用すれば、新しいコントリビューターをはるかに短時間で流れに取り込むことができ、コミュニティを非常に生産的に拡大させることができます。

# 生産性向上の追加的メリット

生産性の向上がチームや組織にもたらすメリットは通常明らかで、より多くの仕事が成し遂げられるため、より多くの価値を生み出せるというものです。しかし、その仕事をしている人々にとってのメリットはどうでしょうか。

## 仕事からの回復力

研究によれば、生産性は**仕事からの回復力**にプラスの影響を与えることがわかっており、それはこの調査のデータでも裏付けられています。仕事からの回復力とは、仕事でのストレスに対処し、仕事をしていないときには仕事のことを切り離す能力のことです。これは「職場で仕事を終える」と呼ばれることもあり、研究によれば、自身を仕事から切り離せる人はより健康で、仕事関連のストレスに対応する能力も優れています。これについては逆もまた重要です。働きすぎだと感じていると、この切り離しが困難になり、バーンアウトや人生への満足度の低下につながります。[37]

## バーンアウト

**バーンアウト（燃え尽き症候群）**は、慢性的な職場のストレスが正しく管理されないことによって生じる状態として世界保健機関に認められており[38]、単なる疲労にとどまりません。バーンアウトは、極度の疲労感、シニシズム、職務における無能力が組み合わさったものです。これは病院、航空管制、テクノロジー関連の複雑で高リスクな職業で

37 Sonnentag, S., & Fritz, C. (2015). Recovery from job stress: The stressor-detachment model as an integrative framework. Journal of Organizational Behavior, 36(S1), S72-S103. https://onlinelibrary.wiley.com/doi/abs/10.1002/job.1924

38 World Health Organization, Burn-out an "occupational phenomenon": International Classification of Diseases. https://www.who.int/mental_health/evidence/burn-out/en/

よく見られ[39]、研究によれば、ストレスの多い仕事は受動喫煙[40] および肥満と同等に健康に有害なことがわかっています。[41] 極端なケースでは、バーンアウトによって家族の問題、臨床的うつ病、場合によっては自殺も引き起こされることがあります。

今年の調査では、仕事からの回復力によってバーンアウトを軽減できることがわかっており、この結果は他の研究結果を裏付けます。また、前年度の調査結果も再確認され、優れた技術的プラクティスと改善されたプロセス（明確な変更管理という形での改善）により、バーンアウトを軽減できることがわかりました。最高レベルのパフォーマーは、バーンアウトを感じると回答した割合が半分でした。別の言い方をすれば、ロー パフォーマーはバーンアウトを感じる可能性が 2 倍高いと言えます。

39 テクノロジーは高リスクの業界ではないと言われることもありますが、間違ったコードを作成することのストレスは非常に現実的です。世間に広く影響を与える可能性があるエラー、または人生を変えるような意味を持つエラー（病院や医療システムを支えるソフトウェアを記述する場合など）は、テクノロジー業界で働く多くの人々にとっての現実です。

40 Goh, J., Pfeffer, J., Zenios, S. A., & Rajpal, S. (2015). Workplace stressors & health outcomes: Health policy for the workplace. Behavioral Science & Policy, 1(1), 43-52.

41 Chandola, T., Brunner, E., & Marmot, M. (2006). Chronic stress at work and the metabolic syndrome: prospective study. BMJ, 332(7540), 521-525.



# 仕事からの回復をサポートするには

生産性は仕事からの回復力にプラスの影響を与えます。これはストレスやバーンアウトを軽減するための鍵となる可能性があります。これをサポートするにはどうすればよいでしょうか。

研究によれば、仕事からの回復力を高めるための鍵は 5 つあります。

1. **心理的デタッチメント**とは、職場の外で仕事について考えることをやめる能力のことです。これを促進するには、仕事のストレスを職場に留める効果がある電子バリアを検討してください。

2. リラックスは休暇中にだけ浸れる贅沢ではなく、生産性の鍵となる要素です。自分に回復の時間を与えることを習慣にしてみましょう。

3. 仕事以外のスキルに熟達することで、前向きな態度を促進し、ストレスを軽減して、健全な人間関係を育むことができます。仕事以外で何か楽しめるようなスキルを身につけてみましょう。

4. コントロールが効かないことは、ストレスの原因となります。仕事上の要求をコントロールできないという事実に対しては、仕事以外で自分がコントロールできる活動を積極的に行うことでバランスを取りましょう。

5. 仕事から離れることを奨励する文化を育みましょう。特に経営陣は、夜や週末には帰宅し、家では仕事をしないという社内の雰囲気を作ることができます。実務担当者は、勤務時間以外は職場との連絡を断つようにし、同僚にもそうするよう奨励することで、この文化をサポートできます。 *

* 心理的安全性の文化は、仕事からの回復力と強い相関性があり、仕事からの回復力を支えることができます。これはチームメンバーがストレスや仕事について気楽に話すことができ、帰宅するときには仕事から離れられるからです。本調査における文化の尺度では、特に「仕事から離れる時間をとることのサポートと奨励」については尋ねておらず、そうした文化は仕事からの回復力を高めることがすでに明らかになっているため、予測的関係については評価していません。

# 変革を実現するには：本当に役に立つのは何か

新しい仕事のやり方を組織全体に広げるにはどうすればよいか、と尋ねられることがよくあります。継続的に再構成や再編成を行うやりかたは生産性に短期的な悪影響を及ぼすため持続可能ではなく、大規模な変革を実施した組織のいくつかは再編成を一切行っていないことがわかっています。間違いなく成功する方法というものは存在しませんが、一つ確かなことは、DevOps 変革が受動的な現象ではないということです。今年の調査では、DevOps のベスト プラクティスを組織全体に広げるための最も一般的なアプローチを特定することを目指しました。

# 変革: 本当に役に立つのは何か

本調査では、回答者のチームや組織が DevOps とアジャイルのメソッドをどのように広めているかを尋ねました。

回答者には次のアプローチから 1 つまたは複数を選択してもらいました。*

- トレーニング センター（「道場」と呼ばれることもある）
- センター オブ エクセレンス
- 概念実証（その後停滞）
- テンプレートとしての概念実証
- シードとしての概念実証
- 実践共同体
- ビッグバン
- ボトムアップまたはグラスルーツ
- マッシュアップ

これらの項目は、業界全体で見られる一般的なアプローチに基づいて作成しました。リストの作成後、分野エキスパートの代表者チームがレビューを行い、そのチームのフィードバックを受けてさらに修正を加え、選択肢をより明確かつ包括的なものにしました。

* 個々のアプローチの詳細については、付録 B をご覧ください。

このデータを SDO パフォーマンス クラスタ全体にわたって検証し、それぞれの戦略の有効性を検討しました。これは完璧な手法ではありませんが、最高レベルと最低レベルのパフォーマーがどのように技術革新を推し進めているかを垣間見ることができます。

マッシュアップはこのサンプルでは共通して 40% と報告されていますが、いずれの特定の投資においても、十分な資金とリソースが不足しています。技術変革を導くための戦略がないと、組織は多くの場合、両面的な作戦を取って「イニシアチブによる死」に至るという誤りを犯してしまいます。これはあまりにも多くの分野でイニシアチブを特定した結果、最終的に重要な業務のリソース不足につながり、必然的にすべてが失敗するというものです。そうではなく、2〜3 のイニシアチブを選択し、そこにリソース（時間、資金、上級 / 最上級の実践者の支援）を注ぎ込んで成功を確保するのがベストです。[42]

ハイ パフォーマーは、組織内での高いレベルと低いレベルの両方においてコミュニティ構造を作り出す戦略を好みます。そうすることで、組織再編成や製品の変化に対してより持続可能で復元力の高い組織になると考えられます。最も多く採用された戦略は実践共同体とグラスルーツの 2 つで、その次がテンプレートとしての概念実証（PoC）（同じ PoC が繰り返されるパターン）と、シードとしての PoC でした。

ロー パフォーマーはトレーニング センター（「道場」とも呼ばれる）とセンター オブ エクセレンス（CoE）を好む傾向がありますが、これらの戦略はサイロ化と専門技能の分離を加速させてしまいます。PoC も試みられてはいるものの、通常は停滞し、成功には至っていません。

一部の戦略については、すべてのパフォーマンス プロファイルに共通のパターンが見られます。すべてのプロファイルで、複数の戦略を組み合わせて採用しサポートしていることが報告されました。

42 これは前に説明した継続的改善の制約モデルとよく似ています。短期的および長期的な目標を立て、その目標を達成するために最も大幅な改善が必要になる主要分野を特定し、それに従ってリソースを配分します。

ビッグバン戦略を重視していると回答したプロファイルはなく（この戦略を最もよく採用していたのはロー パフォーマーでした（19%））、これは結果的に最善の策だと言えるでしょう。経験的に言って、これは実践がきわめて難しいモデルであり、「フルリセット」が必要なほどのひどい状況でのみ試みるべきものです。ビッグバンにはすべての人が長期間にわたって参加する必要があり、数年以上に及ぶ作業にすべてのリソースが費やされます。これはこの方法がロー パフォーマーに最も多く見られることの説明になるかもしれません。

## CoE とトレーニング センターはなぜ推奨されないのか

センター オブ エクセレンス（CoE）は通常、専門知識を 1 つのグループに集中させてしまうため、推奨されません。これはさまざまな問題の原因となります。まず、現在 CoE は組織にとって関連する専門知識のボトルネックとなっており、組織内で専門知識への需要が高まっても、それに応じてスケールすることができません。次に、ともに継続的に学び成長していける同僚達の包括的なグループではなく、「エキスパート」の排他的グループが組織内に形成されてしまいます。これによって不正な規範や行動が排他的に助長され、健全な組織文化がなし崩しにされてしまうおそれがあります。そして最後に、エキスパートが自身の仕事に携われなくなってしまいます。一般的な「ベスト プラクティス」をエキスパートが推奨または確立することはできますが、一般的な学習から実際の業務の実装に至るまでの道程については、学習者に任されることになります。たとえば、エキスパートはアプリケーションをコンテナ化する方法についてのワークショップを開きますが、彼らが実際にアプリケーションをコンテナ化することはまず、あるいはまったくありません。この理論と実際のプラクティスとの乖離は、いずれ彼らの専門知識を脅かすことになります。

トレーニング センターには成功例も見受けられますが、トレーニング センターでは当初のプログラムと持続的な学習の両方を実施するための専用のリソースとプログラムが必要になります。多くの企業がトレーニング プログラムを効果的なものにするため膨大なリソースを確保しており、独立したクリエイティブ環境専用の建物や、トレーニング資料を作成し進捗状況を評価する専門のスタッフを抱えています。そしてその後には、組織全体で学習した内容が持続し伝搬されていることを確認するため、追加のリソースが必要になります。組織はトレーニング センターに参加したチームにサポートを提供し、彼らのスキルと習慣が通常の業務環境に戻ってからも持続しており、古い業務パターンが復活していないことを確認しなければなりません。こうしたリソースがない場合、組織はすべての投資が無駄になるリスクを負うことになります。チームが新しい技術やプロセスをセンターに学びに行ってそれを組織全体に広めることにはならず、新しい習慣がセンター内にとどまり、また新たなサイロが（一時的にせよ）生み出されることにつながります。また、CoE と同様の制限もあります。ワークショップやトレーニング資料がトレーニング センターのスタッフ（または他の実務から離れた「エキスパート」）のみによって作成されている場合、もし彼らが実際には一度もその業務を行うことがないとしたらどうでしょうか。

# パフォーマンス プロファイル別の DEVOPS 変革戦略のヒートマップ

| | ロー | ミディアム | ハイ | エリート |
|---|---|---|---|---|
| トレーニング センター | 27% | 21% | 18% | 14% |
| センター オブ エクセレンス | 34% | 34% | 20% | 24% |
| 概念実証（その後停滞） | 41% | 32% | 20% | 16% |
| テンプレートとしての概念実証 | 16% | 29% | 29% | 30% |
| シードとしての概念実証 | 21% | 24% | 29% | 30% |
| 実践共同体 | 24% | 51% | 47% | 57% |
| ビッグバン | 19% | 19% | 11% | 9% |
| ボトムアップまたはグラスルーツ | 29% | 39% | 46% | 46% |
| マッシュアップ | 46% | 42% | 34% | 38% |

## 有効なスケーリング戦略

本調査では、ハイ パフォーマーとエリート パフォーマーが最もよく使用している戦略を理解するために追加的なクラスタ分析を実施し、新しい 4 つのモデルパターンを特定しました。

- **コミュニティ ビルダー:** このグループは、実践共同体、グラスルーツ、PoC（前述の「テンプレートとしての PoC」と「シードとしての PoC」）に焦点を当てています。これは全体の 46% に認められます。
- **大学:** このグループは教育とトレーニングに焦点を当てており、その取り組みの大半はセンター オブ エクセレンス、実践共同体、トレーニング センターに向けられます。このパターンは全体の 9% しか認められません。これはこの戦略が、成功する場合もあるものの一般的ではなく、学習を組織全体に拡大するには多大な投資と計画が必要になることを示唆しています。
- **イマージェント:** このグループはグラスルーツと実践共同体の取り組みに的を絞っています。これは最も不干渉主義のグループのように思われ、全体の 23% に見られます。
- **実験者:** 実験者は 22% のケースに認められました。このグループはビッグバンと道場を除くすべての戦略、つまりコミュニティと創造に的を絞ったすべての活動において、高レベルの活動性を示しています。また、PoC（その後停滞）でも高レベルであり、彼らがこの活動を利用しつつハイ パフォーマーでいられるという事実は、彼らがこの戦略を利用してアイデアをすばやく試していることを意味します。

この 4 つのパターンを念頭に置いて、DevOps 変革を組織する方法の戦略を練ることができます。ハイ パフォーマーとエリート パフォーマーはスケーリングのための戦略の開発を始めており、組織はそうした戦略のいずれかを選択して、彼らの活動を模倣し自身のパフォーマンスを高めることができます。

# まとめ
# THOUGHTS

どの時代にも、固有のソフトウェア方法論のトレンドがあります。いずれもそれ以前の方法論に比べると良さそうに見えますが、後から見れば効果はなかったことがわかります。しかし、DevOps が価値をもたらすというエビデンスは継続的に認められており、本調査では 6 年連続で、組織が DevOps メソッドを使用してソフトウェアの開発とデリバリーを改善するために役立つ重要な能力とプラクティスを統計的に検証しました。

DevOps はトレンドではなく、いずれソフトウェアの開発と運用の標準的な方法となり、すべての人に生活の質の向上をもたらすものです。

今年の調査に協力してくださったすべての方々に感謝するとともに、皆様とその組織がより有能なチームと強力なソフトウェアを構築し、職場で仕事を終えられるようになるため、本調査が役立つことを祈念しています。

# 方法論

本調査の綿密な方法論は、単に生の数値を報告して、 SDO パフォーマンス、組織的パフォーマンス、技術的プラクティス、文化規範、生産性の間の予測的関係を確認するというだけにとどまりません。このセクションでは、分析手法に加え、調査回答者をどのように募ったか、そして質問、モデル、構成概念をどのように作成したかについても説明します。詳細については、本書のパート II である『Accelerate: The Science of Lean Software and DevOps』をご覧ください。

調査手法に関するご質問があれば、お気軽に dora-data@google.com までお問い合わせください。

## 調査デザイン

本調査では横断的な理論ベースのデザインを使用しています。この理論ベースデザインは、推論的または推論予測的と呼ばれており、昨今のビジネスやテクノロジー関連の調査において最もよく用いられている手法の一つです。推論的デザインが使用されるのは、純粋に実験的なデザインが不可能であり、現場での実験が望ましい場合です。たとえばビジネス関連の調査で、データ収集が無菌の実験室環境ではなく複雑な組織内で行われる場合などがこれに該当します。企業は調査チームが定義した対照群に適合するために利益を犠牲にする必要がありません。

## 目標母集団とサンプリング手法

本調査の目標母集団は、技術的業務と変革に直接携わっている、または深く関わっている実務担当者やリーダーで、特に DevOps に詳しい人々でした。こうした人々のリストはなく、すなわち彼らを説明することはできても、実際どこにいるか、どうすれば見つけられるか、何人いるのかといったことは不明なため、スノーボール サンプリングの手法で回答者を募ることにしました。つまり、メーリング リスト、オンライン プロモーション、ソーシャル メディアを通じて本調査を宣伝し、またそれを見た人々に自分のネットワークで調査のことを広めてくれるよう依頼して、サンプルを雪だるま式に増やしていったのです。本調査のサンプルは、 DevOps に詳しく、そのためおそらくすでに若干の DevOps に携わっている、組織やチームに限定されると考えられました。スノーボール サンプリングの制限を克服するための鍵は、幅広い初期サンプルを得ることです。これを実現するため、初期サンプルについては Google 独自の連絡担当者リストのほか、スポンサーの連絡担当者リストも活用しました。その結果、業界のトレンドとほぼ一致する個人および企業属性のサンプルが得られました。

## 潜在的構成概念の形成

仮説と構成概念は、可能な限り事前に検証済みの構成概念を利用して作成しました。新しい構成概念を作成する必要がある場合には、理論、定義、専門家の意見に基づいて作成しました。その後さらに、最終的な調査で収集されるデータの信頼性と有効性が高まるよう、意図と表現を明確化するための追加作業を行いました。[43] 構成概念の測定には、より高度な分析ができるように、リッカート式[44] の質問を使用しました。

43 本調査では、次の Churchill の方法論を採用しました。Churchill Jr, G. A. “A paradigm for developing better measures of marketing constructs,” Journal of Marketing Research 16:1, (1979), 64–73.

44 McLeod, S. A. (2008). Likert scale. www.simplypsychology.org/likert-scale.html から入手

## 統計分析手法

- クラスタ分析。ソフトウェア デリバリーのパフォーマンス プロファイルと、ハイ パフォーマーが使用しているスケーリングのアプローチについては、クラスタ分析を使用して特定しました。このアプローチでは、デプロイ頻度、リードタイム、サービス復元時間、変更失敗率といった本調査におけるスループットと安定性のパフォーマンス挙動の観点で、あるグループのメンバーは統計的に皆互いに類似し、他のグループのメンバーとは類似しなくなります。また、(a) 融合係数 (fusion coefficients) の変化、(b) 各クラスタに含まれる個人数 (個人数の非常に少ないクラスタを含むソリューションは除外)、(c) 一変量 F 検定に基づいて、ウォード法[45] を使用したソリューションを選択しました。[46] 本調査では階層式のクラスタ分析法を採用しました。これは、この手法の説明能力が高い (クラスタ内の親子関係を理解できる) ことと、クラスタ数を事前に規定するための業界的、理論的理由がなかったことによります。すなわち、必要なクラスタ数はデータによって決めることにしました。最後に、今回のデータセットはあまり大きくありませんでした (階層式クラスタリングは極端に大きなデータセットには適しません)。

- 測定モデル。分析を実施する前に、バリマックス回転を使用した主成分分析を含む探索的因子分析を使用して、構成概念を特定しました。[47] 収束性妥当性と弁別性妥当性[48] および信頼性[49] の統計的検定は、平均分散抽出 (AVE)、相関性、クロンバックのアルファ[50]、合成信頼性[51] を使用して確認しました。構成概念はこれらの検定に合格し、良好な精神測定特性を示しました。

- 構造方程式モデリング。構造方程式モデル (SEM)[52] は、相関ベースの SEM である部分的最小二乗 (PLS) 分析を使用して検定しました。PLS を分析に利用したのにはいくつかの理由があります。まず、データに正規性の前提が不要であること。次に、探索的、漸進的調査に適していること。最後に、この分析が (データのモデル適合度検定と比べて) 従属変数の予測に最適であることです。[53] 本調査では SmartPLS 3.2.8 を使用しました。統制変数として業界を使用した場合[54]、本文で説明したように、小売を除いて有意な影響は認められませんでした ($p < 0.05$)。統制変数として大企業 (従業員 5,000 名以上の組織) を使用した場合、有意な影響が認められました ($p < 0.001$)。統制変数として経験年数を使用した場合、有意な影響は認められませんでした。SEM の図に示されたパスはほとんどが $p < 0.001$ でしたが、次のパスは $p < 0.05$ でした。継続的デリバリー → バーンアウト、変更承認 → ソフトウェア デリバリー パフォーマンス、継続的インテグレーション → 継続的デリバリー、文化 → ソフトウェア デリバリー パフォーマンス (メインモデル)、外部検索 → 生産性、モニタリング → 技術的負債、コードのメンテナンス性 → 技術的負債 (生産性モデル)。

45 Ward, J.H. “Hierarchical Grouping to Optimize an Objective Function.” Journal of the American Statistical Association 58(1963): 236–244.
46 Urich,D., and B. McKelvey. “General Organizational Classification: An Empirical Test Using the United States and Japanese Electronic Industry.” Organization Science 1, no. 1 (1990): 99-118.
47 Straub, D., Boudreau, M. C., & Gefen, D. (2004). Validation guidelines for IS positivist research. Communications of the Association for Information systems, 13(1), 24.
48 http://www.socialresearchmethods.net/kb/convdisc.htm
49 http://www.socialresearchmethods.net/kb/reliable.php
50 Nunnally, J.C. PsychometricTheory. NewYork: McGraw-Hill, 1978.
51 Chin, W. W. (2010). How to write up and report PLS analyses. In Handbook of partial least squares (pp. 655-690). Springer, Berlin, Heidelberg.
52 http://www.statisticssolutions.com/structural-equation-modeling/
53 これらの方法論に関する考察は、次の資料で裏付けられています。Chin, W.W. (1998). Issues and opinions on structural equation modeling. MIS Quarterly, 22(2), vii-xvi; Gefen, D., Straub, D. W., & Rigdon, E. E. (2011). An update and extension to SEM guidelines for administrative and social science research. MIS Quarterly, 35(2), iii-xiv.; and Hulland (1999). Hulland, J. (1999). Use of partial east squares (PLS) in strategic management research: A review of four recent studies. Strategic Management Journal, 20(2), 195-204.
54 http://methods.sagepub.com/reference/encyc-of-research-design/n77.xml

# 謝辞

本調査の情報は、DevOps、アジャイル、その他の広範な技術コミュニティとの共同作業と対話によって得られたものです。ご自身の経験や体験談、成功談、課題や失敗などをオープンにお話しいただいたすべての同業者や同僚の皆様に深く感謝いたします。DORA ではこれらの情報をもとに文献のレビューを行い、毎年の調査で使用する質問を作成しています。

今年度のレポートの作成に当たって情報やアドバイスをいただいたことについて、以下の方々に感謝を申し上げます。謝辞はすべて、貢献の種類別に、アルファベット順に記載してあります。

レポートに関する全体的なガイダンスをいただいた、**Adrian Cockroft** 氏、**Sam Guckenheimer** 氏、**Gene Kim** 氏、**Kelsey Hightower** 氏に感謝いたします。皆様のアドバイザーとしてのご意見のおかげで、この調査における重要なアイデアと方向性を浮き彫りにすることができました。

調査に関するアドバイスと測定項目についての情報をいただいた Google の**エンジニアリング生産性研究**チーム、特に **Collin Green** 氏、**Ciera Jaspan** 氏、**Andrea Knight-Dolan** 氏、**Emerson Murphy-Hill** 氏に感謝いたします。また、心理的安全性に関する情報と調査手段を提供していただいた **Project Aristotle** チームにも深く感謝しています。

本調査は、各分野のエキスパートによる慎重なレビューなしには成し遂げることができませんでした。エキスパートの皆様がお忙しい中、補足トピックのレビューを行い、測定項目に関する情報を提供してくださったことに大変感謝しております。Angie Jones 氏、**Ashley McNamara** 氏、**Betsy Beyer** 氏、**Bridget Kromhout** 氏、Courtney Kissler 氏、**J. Paul Reed** 氏、Mike McGarr 氏、Nathen Harvey 氏、Seth Vargo 氏、Xavier Velasquez 氏に深く感謝の意を表します。

今年の調査とレポートのために分析をサポートしてくださった David Huh 氏に深く感謝いたします。

また、フィードバックによってレポートの改善を手助けしてくださった、技術的造詣の深い読者の皆様にも感謝申し上げます。**Nathen Harvey** 氏、**Tom Limoncelli** 氏、**Caitie McCaffrey** 氏、**Rachel Potvin** 氏、**Corey Quinn** 氏、**Xavier Velasquez** 氏、どうもありがとうございました。

今年のレポートを注意深く細部まで編集してくださった **Cheryl Coupé** 氏に対し、作成者一同から謝辞を述べさせていただきます。

レポートのレイアウトとデザインは、**Siobhán Doyle** 氏によるものです。

# 著者

**Nicole Forsgren** 博士は DORA チームのリーダーで、現在は Google Cloud に所属しています。過去 6 年間の State of DevOps Report で主任調査官として世界最大規模の DevOps 研究を率い、Shingo Publication Award を受賞した書籍『Accelerate: The Science of Lean Software and DevOps』の主執筆者でもあります。教授、システム管理者、パフォーマンス エンジニアの経験があります。Nicole の著作はいくつかのピアレビュー ジャーナルから発表されています。Nicole は、アリゾナ大学で経営情報システムの博士号を取得しています。

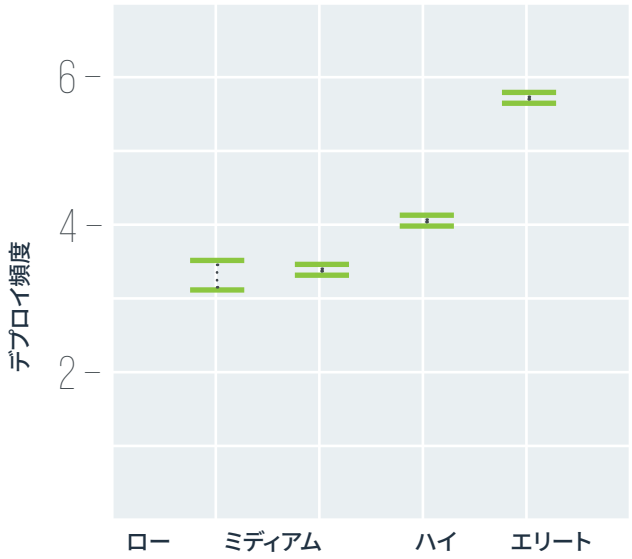
**Dustin Smith** 博士はヒューマン ファクター心理学者であり、Google でシニア ユーザー エクスペリエンス リサーチャーを務めています。Dustin は、ソフトウェア エンジニアリング、無料ゲーム、医療、軍事といったさまざまなコンテキストにおいて、人はどのように自身を取り巻くシステムや環境から影響を受けるかを研究しています。Google での彼の研究は、ソフトウェア開発者が開発作業中にどのようなところで幸福感を覚え、より生産的になるかを特定することに重点を置いています。Dustin は、ウィチタ州立大学でヒューマン ファクター心理学の博士号を取得しています。

**Jez Humble** 氏は、Shingo Publication Award を受賞した『Accelerate, The DevOps Handbook, Lean Enterprise』や Jolt Award を受賞した『Continuous Delivery』など、いくつかのソフトウェア関連書籍の共同執筆者です。3 つの大陸のさまざまな規模の会社でコーディング、インフラストラクチャ、製品開発を経験してきました。現在は Google Cloud でテクノロジー アドボケートとして勤務しながら、カリフォルニア大学バークレー校で教鞭をとっています。

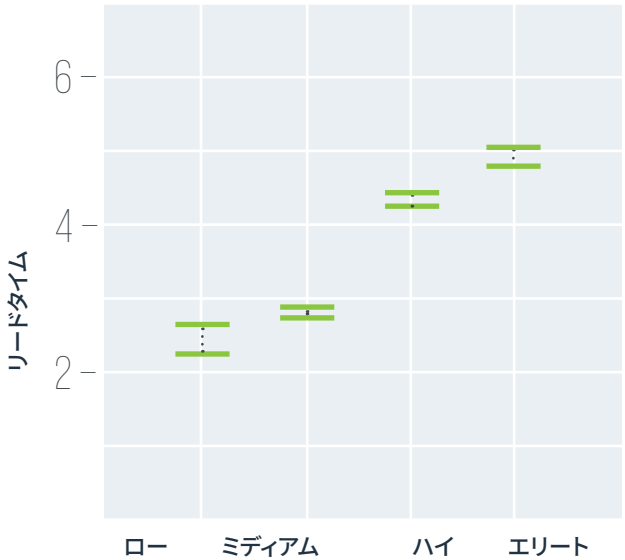
**Jessie Frazelle** 氏は独立コンサルタントで、さまざまなスタートアップ企業や Google、Microsoft、GitHub でエンジニアとして働いてきました。Docker や Kubernetes などのツールを日常的に使用し、各種 PaaS のエンドユーザーでもあったことから、彼女は数多くの開発プラクティスやインフラストラクチャ プラクティスを経験しました。物事をあらゆる観点から見たいと思い、ツールの開発から本番環境でのツールの使用まで幅広い職種を渡り歩いています。

# 付録 A

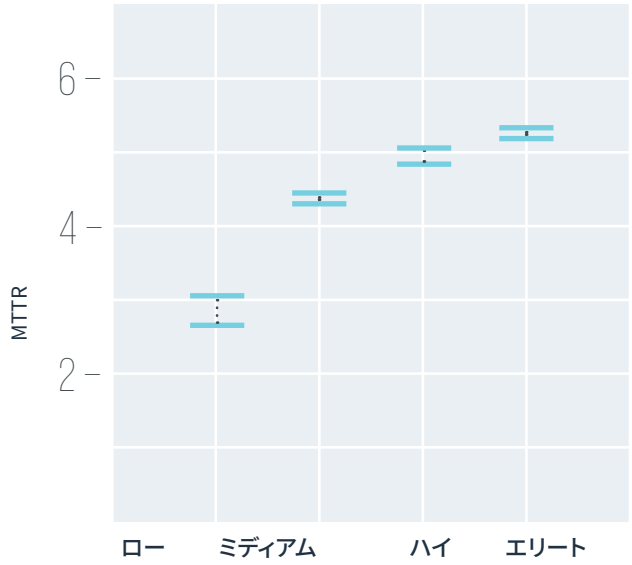
## デプロイ頻度



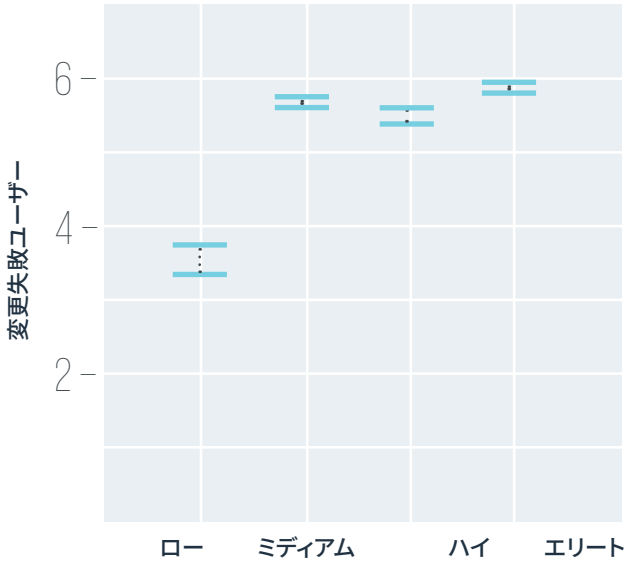

## 変更のリードタイム




## サービス復元時間




## 変更失敗率




**デプロイ頻度:**

1 = 半年に 1 回より少ない
2 = 1 か月に 1 回〜半年に 1 回
3 = 1 週間に 1 回〜1 か月に 1 回
4 = 1 日に 1 回〜1 週間に 1 回
5 = 1 時間に 1 回〜1 日に 1 回
6 = オンデマンド（1 日に複数回のデプロイ）

**変更のリードタイム**

1 = 半年より長い
2 = 1 か月〜半年
3 = 1 週間〜1 か月
4 = 1 日〜1 週間
5 = 1 日未満
6 = 1 時間未満

**サービス復元時間**

1 = 半年より長い
2 = 1 か月〜半年
3 = 1 週間〜1 か月
4 = 1 日〜1 週間
5 = 1 日未満
6 = 1 時間未満

**変更失敗率**

1 = 76%〜100%
2 = 61%〜75%
3 = 46%〜60%
4 = 31%〜45%
5 = 16%〜30%
6 = 0%〜15%

# 付録 B

## DevOps をスケールするための戦略

- **トレーニング センター（「道場」と呼ばれることもある）** - 人々が通常業務のルーチンから抜け出し、一定期間の間、新しいツールやテクノロジー、プラクティス、文化を学びます。その後、目標（希望？）を持って通常業務に復帰し、新しい仕事のやり方を実践するとともに、場合によってはそれを他の従業員にも広めます。

- **センター オブ エクセレンス** - すべての専門知識を集約し、従業員に指導やアドバイスを与えます。

- **概念実証（その後停滞）** - 中心的なチームに最良と思うやり方で自由に作らせる概念実証（PoC）プロジェクト。多くの場合、組織の規範（および正式なルール）から逸脱した方法をとります。ただし、 PoC の後、この取り組みは停滞します。

- **テンプレートとしての概念実証** - まず小規模な概念実証（PoC）プロジェクト（上記のとおり）から開始し、次に初回の結果をパターンとして他のグループで同じパターンを繰り返します。

- **シードとしての概念実証** - まず小規模な概念実証（PoC）から開始し、次に PoC の知識を他のグループに広めます。そのために、 PoC グループ（最初の PoC グループまたは後続 / 並列の PoC グループ）を解体し、メンバーを他のグループに送り込んで学んだ知識やプラクティスを共有します。これは「ローテーション」と呼ばれることもあります。ローテーションでは、 PoC メンバーを他のチームに組み入れ、新しいプラクティスや文化を広める教師の役割を与えます。加入したメンバーはこの新しいグループに無期限にとどまるか、新しいプラクティスを伝授するために必要な期間だけとどまります。

- **実践共同体** - ツール、言語、方法論に対する共通の興味を持つグループを組織内で育成し、知識や専門技術を互いに共有し合いながら、チームの垣根を越えて組織全体に広めます。

- **ビッグバン** - 通常はトップダウンの命令で、組織全体を同時に DevOps 方法論に移行します（ただし、方法論の定義は自ら選びます）。

- **ボトムアップまたはグラスルーツ** - 実務に近い複数の小さなチームが協力して変革を成し遂げた後、その成功に関する情報を、正式な組織的支援やリソースの助けを借りずに組織全体に広めます。

- **マッシュアップ** - 上記の複数のアプローチを実施します。多くの場合、一部だけを実施するか、成功を得るためのリソースまたは優先順位付けが不十分な状態で行います。