#jassthokuriku

組織に自動テストを書く文化を根付かせる戦略

(2023早春バージョン)

和田 卓人 (@t_wada)

自己紹介

Hatena t-wada

t_wada

twada

📷🤷‍♂️ 📹🤷‍♂️

🐦🤷‍♂️ #jassthokuriku

技術書の出版に関わっています

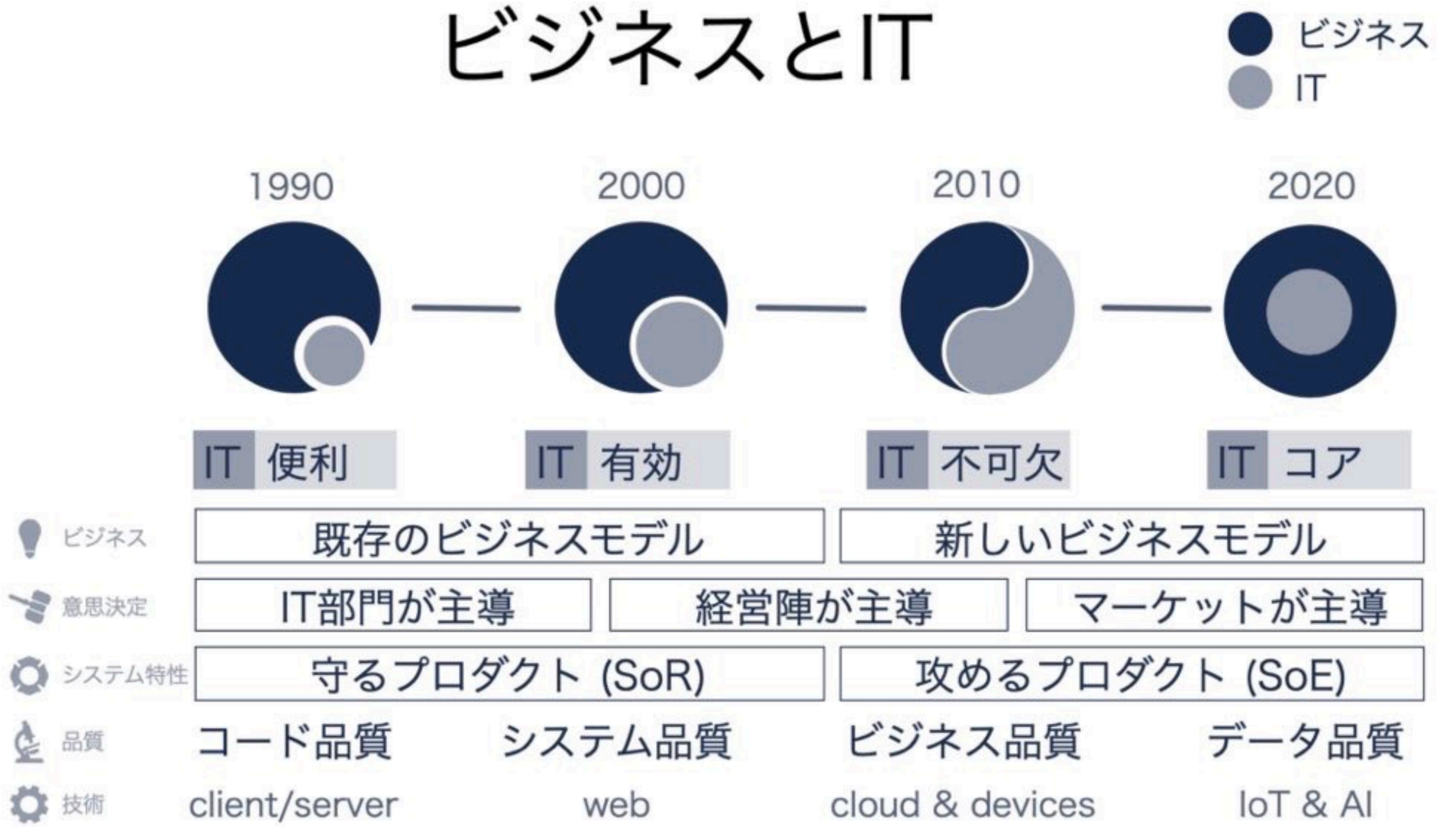
WEB+DB PRESS にコラムを連載しています

https://t-wada.hatenablog.jp/entry/wdpress-savanna-column

よろしくお願いします

illustrated by @mty_mno

前提: ITは事業のコアになった



https://www.evangelism.jp/resources/business-with-it/

PMBOK第7版の大改訂: 予測型から適応型へ

PMBOK第6版

PMBOKガイド

- イントロダクション
- プロジェクト環境
- プロジェクトマネージャの役割

- 統合
- スコープ
- スケジュール
- コスト
- 品質
- リソース
- コミュニケーション
- リスク
- プロキュアメント（調達）
- ステークホルダー

PMスタンダード

- 立ち上げプロセス
- 計画プロセス
- 実行プロセス
- 監視・コントロール
- 集結

※PMBOK 7thのp.xiiiをベースに作成

PMBOK第7版

PMスタンダード

- イントロダクション
- 価値提供システム
- **12のプロジェクトマネジメント・プリンシプル（原則）**
  - スチュワードシップ
  - 協力的なプロジェクトチームの環境を作る
  - ステークホルダーを効果的に連携する
  - 価値に集中する
  - システムの相互作用を認識し、評価し、対応する
  - リーダーシップを行動で示す
  - 文脈に基づいたテーラリング（カスタマイズ）
  - 品質をプロセスと成果物に組み込む
  - 複雑性に適応する
  - リスクへの対応を最適化する
  - 適応力とレジリエンスを高める
  - 未来の状態を達成するために変化できる

PMBOKガイド

- **8のパフォーマンス・ドメイン**
  - ステークホルダー
  - チーム
  - 開発アプローチとライフサイクル
  - 計画
  - プロジェクトワーク
  - デリバリー
  - 測定
  - 不確実性
- テーラリング
- モデル、メソッド、ツール

https://note.com/miz_kushida/n/n103a7da460c5

予測型から適応型へ

決められたものを
決められたときまでに
破綻なく作る

↓

だれも答えがわからないものを
模索しながら作り続ける
しかも答えが刻々と変わっていく

わからないなりにやりようはある

日経 BP 中田さんのレポート

2017年10月

シリコンバレーの「何が」凄いのか

– スタートアップ取材から見えた共通点 –

日経コンピュータ
NIKKEI COMPUTER

シリコンバレー支局
中田　敦

日経BP社
Nikkei Business Publications, Inc.

https://www.slideshare.net/atsnakada/ss-80520040

体験したことのない製品やサービスを作る方法論

体験したことの無い
製品・サービスを作るには？

- 要件定義の存在しない世界
  - 「顧客に言われたとおり作る」ことが不可能
  - 顧客も「欲しいものが分からない」から
- 単純な聞き方では絶対に分からない
  - （例）自動運転車の中で何がしたい？
    - ユーザーはそもそも自動運転車の車中を創造できない
  - （例） AIをどうやって活用しますか？
    - AIを使ったことが無い人が分かるはずがない

ITpro http://itpro.nikkeibp.co.jp/

体験したことの無い
体験を作るには？

- プロトタイプをとにかく早く作る
- それを顧客にテストしてもらう
- 顧客のフィードバックを得る
  - 実物を体験すれば、さすがに何かは言える
- 良くない点を製品を改善する
- 失敗を改善するサイクルを繰り返す
- つまり「リーンスタートアップ」
  - 実現手法が、アジャイル開発やデザイン思考

ITpro http

もう一つの柱であるデザイン思考は
この講演ではスコープ外とさせてください

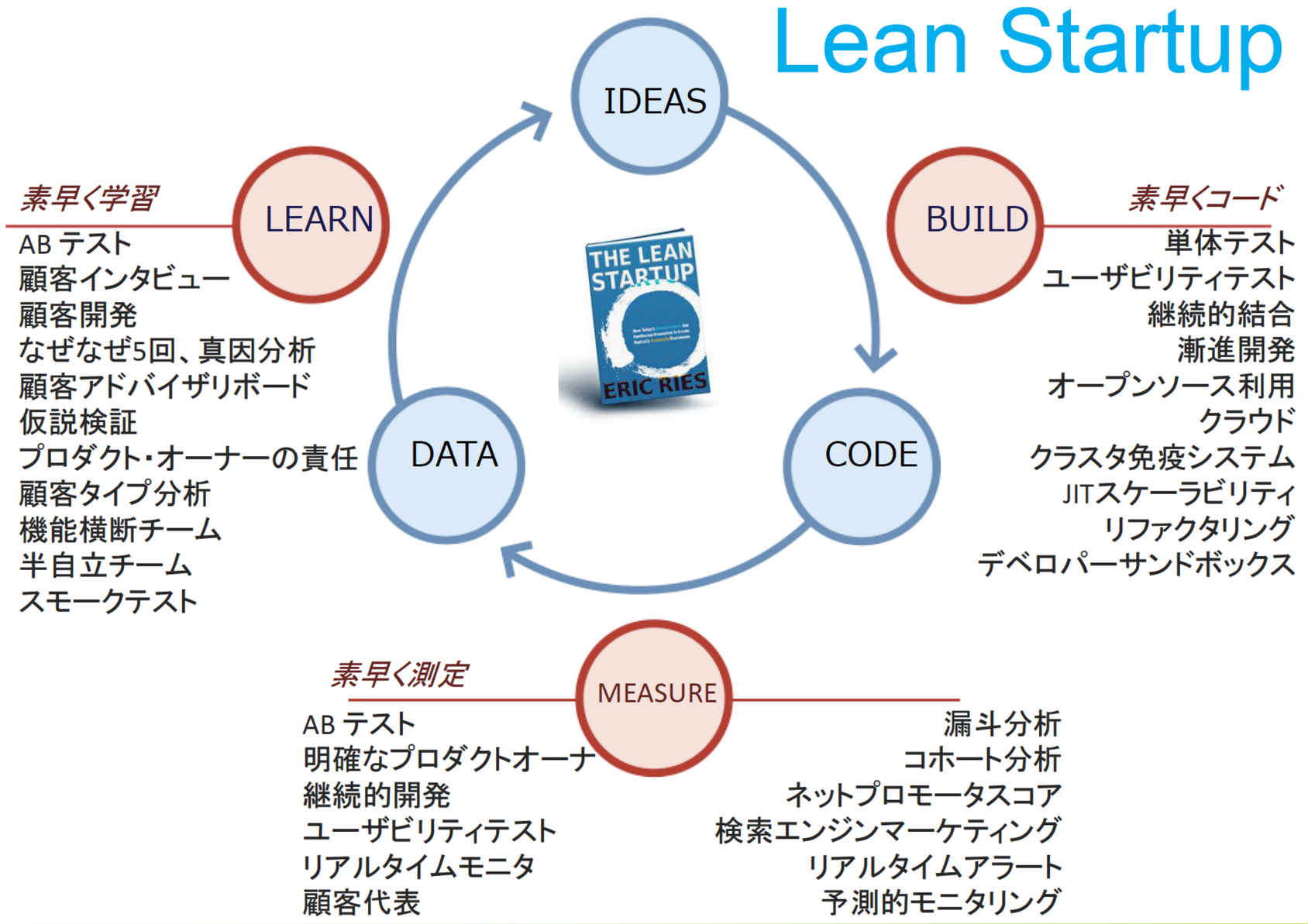
リリースした製品は「必ず間違っている」

シリコンバレーの強みは技術＋デザイン

- シリコンバレー企業の最大の特徴は？
  - リリースした製品は「必ず間違っている」
  - 「リリース後の改善力」こそが強み
    - ユーザーの「痛み」を引き出すためには、質問力を備えたデザイナーが必要**＝デザイン思考**
    - ユーザーの要望に素早く応えられるソフトウエア開発者が必要**＝アジャイル開発**
    - 質問に的確に答えられる「優れたユーザー」
      - シリコンバレー住人は質問慣れしている
- だからシリコンバレーで作りたい

ITpro http://itpro.nikkeibp.co.jp/

リーンスタートアップ



http://blogs.itmedia.co.jp/hiranabe/2011/05/lean-startup-introduction-in-japanese-1.html

アジリティを支える技術はDevOpsへ体系化された

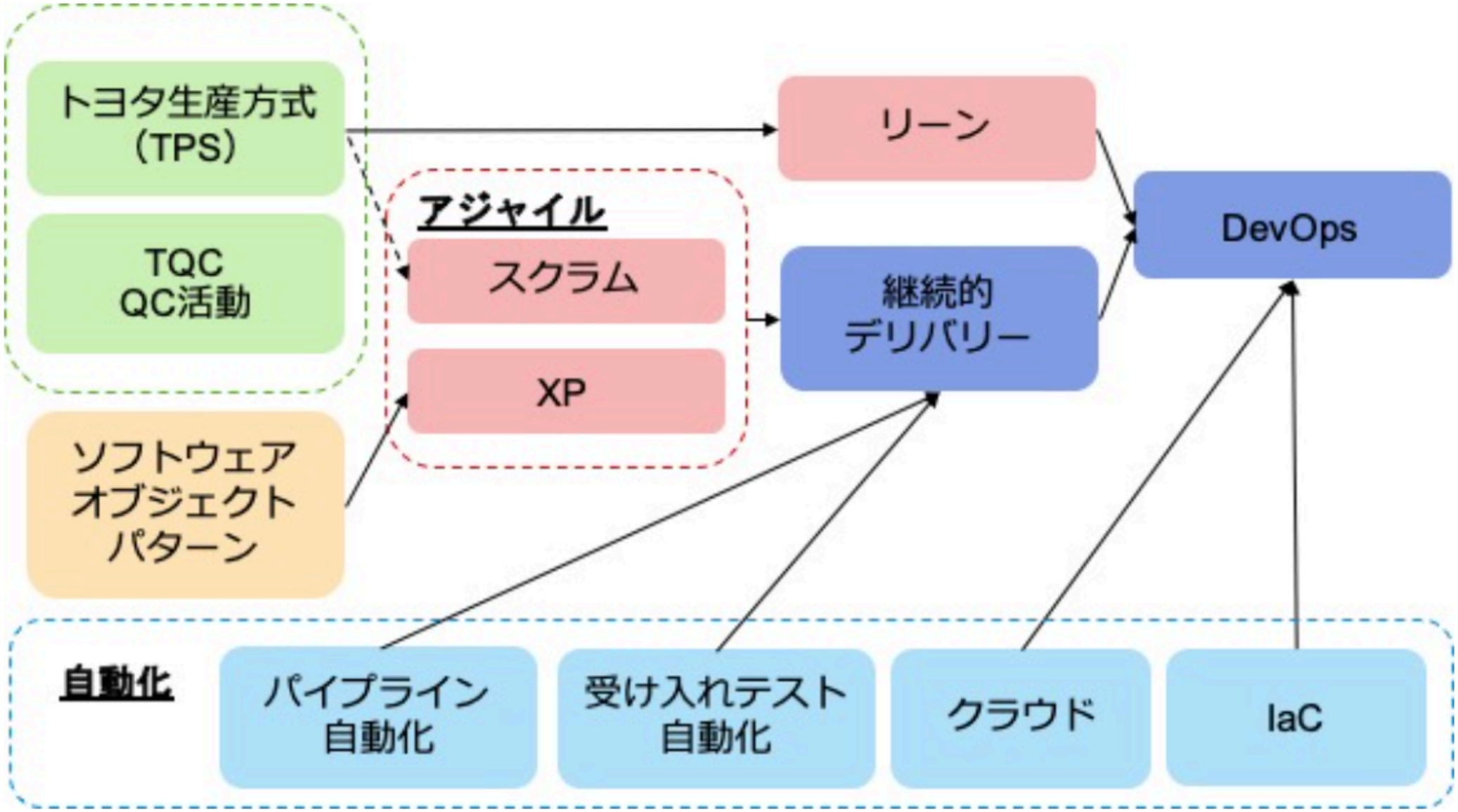


図1 アジャイルからDevOpsへの流れ

https://kokotatata.hatenablog.com/entry/2020/06/01/163652

The State of DevOps Reports

The State of DevOps Reports

2022 State of DevOps Report

(registration required)

Read PDF →

2021 State of DevOps Report

Read PDF →

2019 State of DevOps Report

Read PDF →

2018 State of DevOps Report

2017 State of DevOps Report

(in partnership with Puppet)

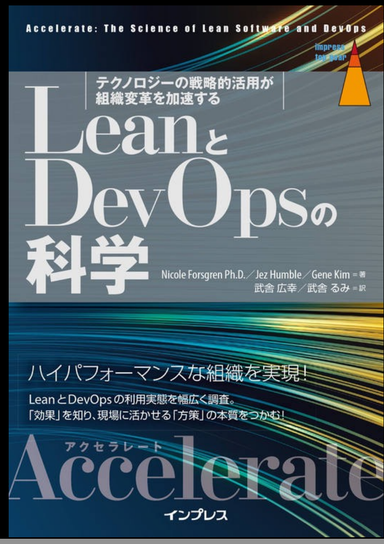
2016 State of DevOps Report

(in partnership with Puppet)

https://www.devops-research.com/research.html

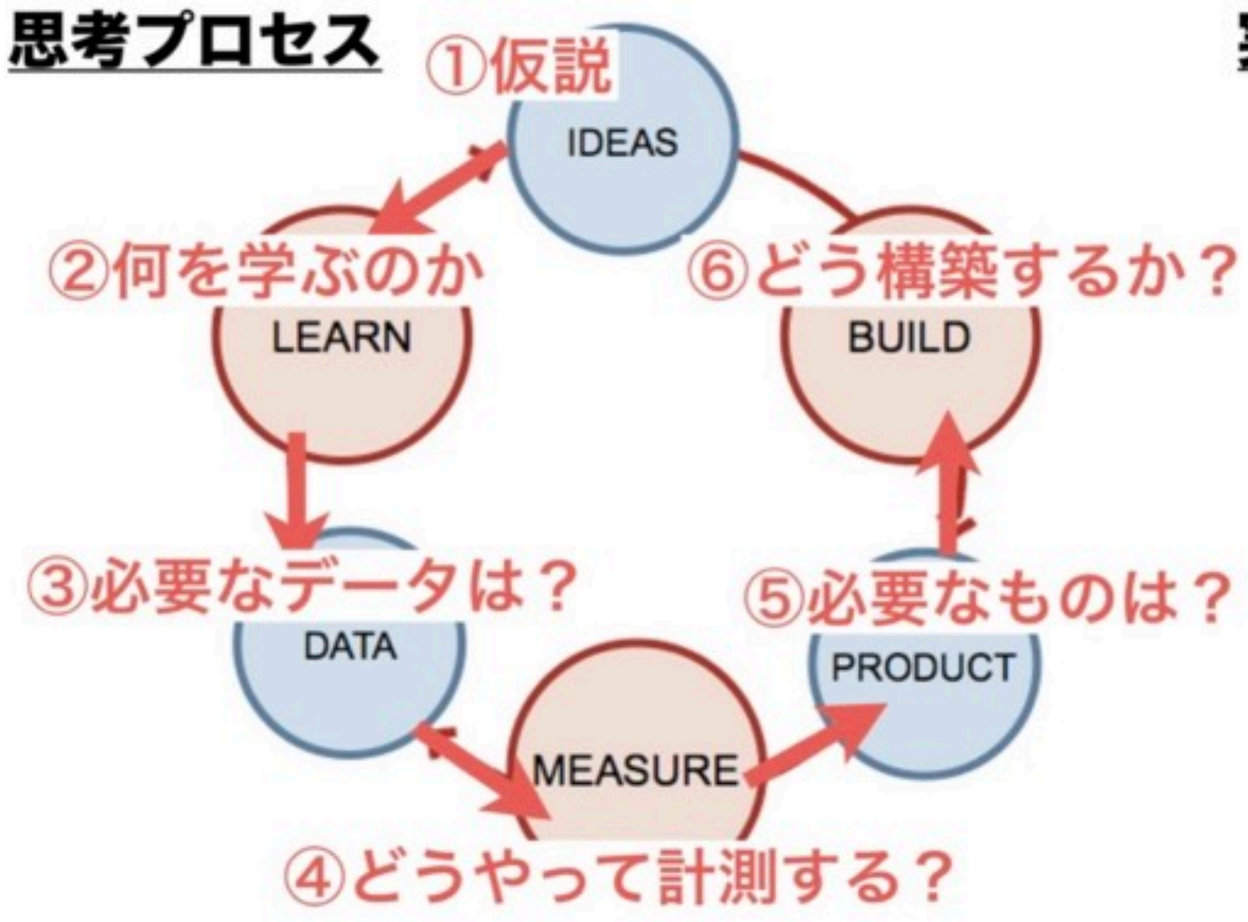
変わる世界: 4つのキーメトリクス

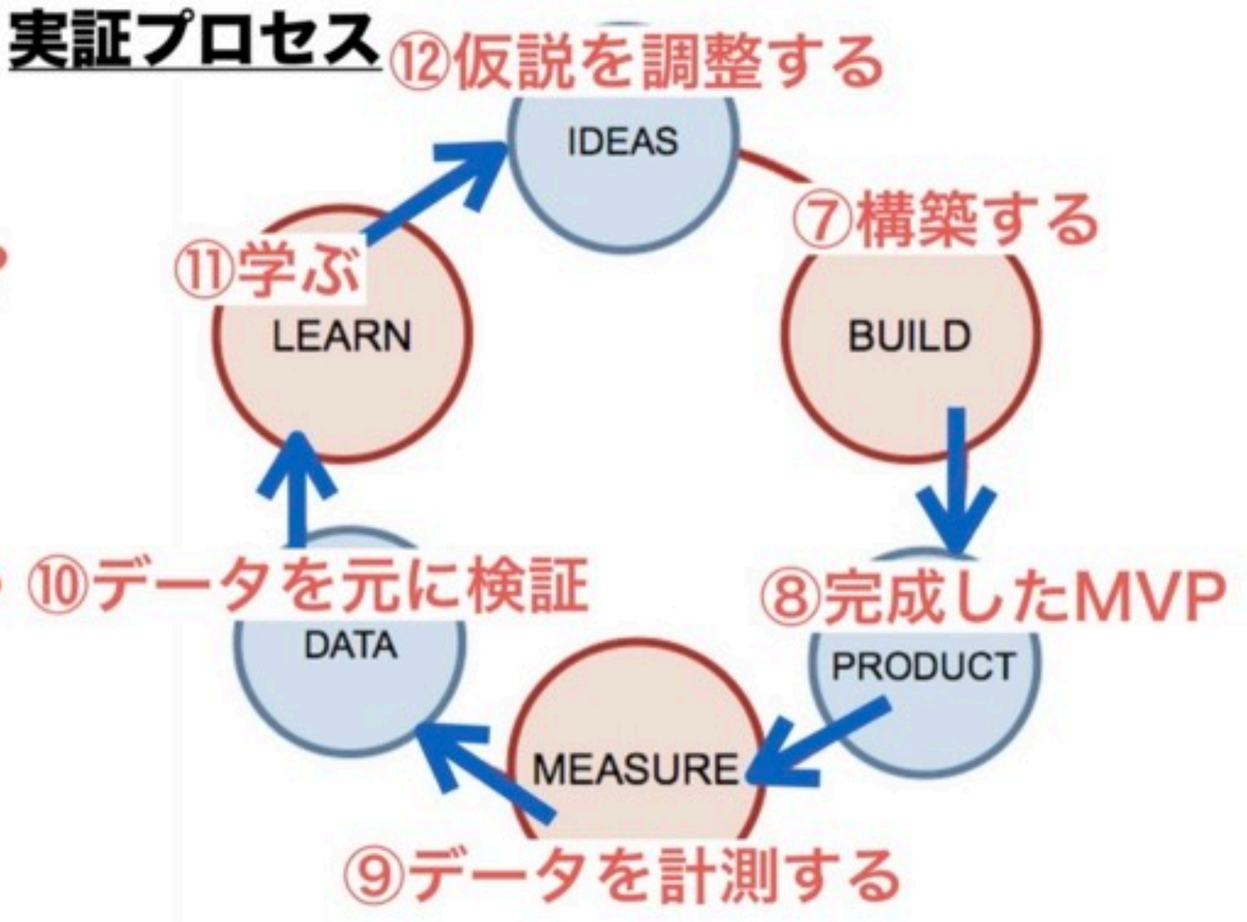
- リードタイム
- デプロイ頻度
- MTTR(平均修復時間)
- 変更失敗率



https://www.amazon.co.jp/dp/4295004901/

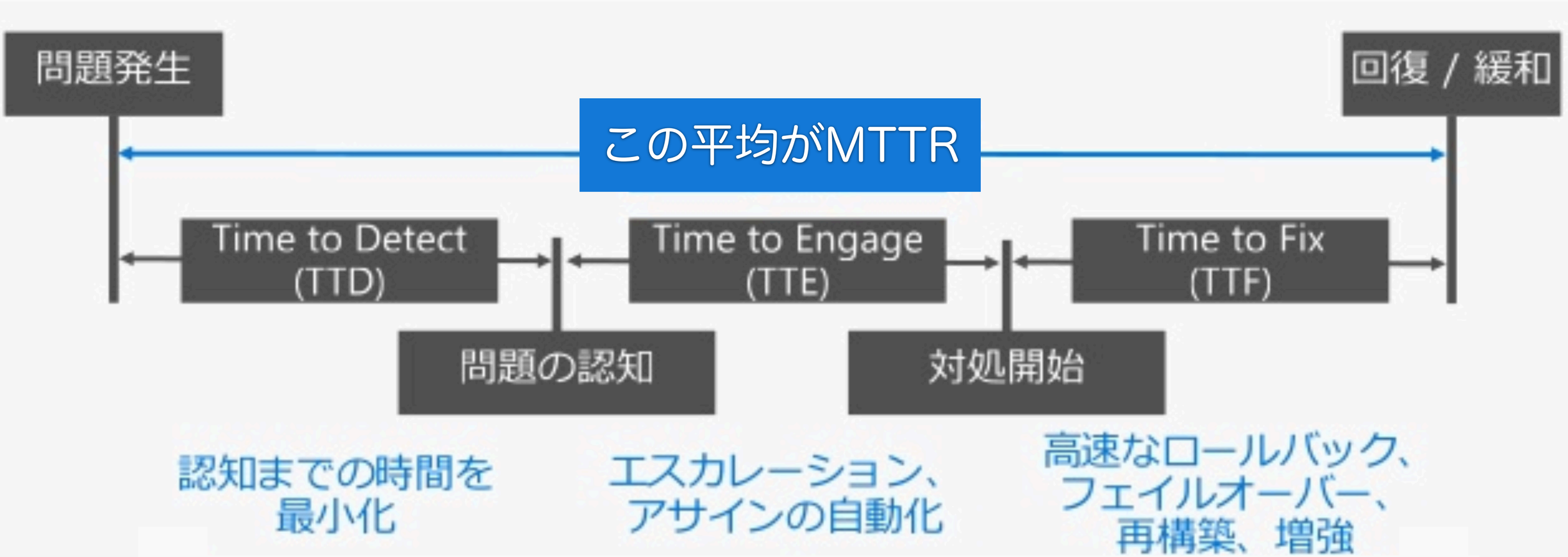
リードタイムとデプロイ頻度: 仮説検証プロセスを迅速に回す





http://i2key.hateblo.jp/entry/2017/05/15/082655

MTTR: Mean time to Recovery/Repair



https://www.slideshare.net/decode2017/do05-sre-site-reliability-engineering-in-azure-on-azure (一部改変)

https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

4つのキーメトリクス: 2019年の調査（3万1千件以上の回答をクラスター分析）

2019

	エリート	ハイパフォーマー	ミディアム パフォーマー	ローパフォーマー
リードタイム	1日未満	1日から1週間	1週間から1ヵ月	1ヵ月から半年
デプロイ頻度	オンデマンド (1日複数回)	1日1回から週1回	週1回から月1回	月1回から半年に1回
MTTR	1時間未満	1日未満	1日未満	1週間から1ヵ月
変更失敗率	0 - 15%	0 - 15%	0 - 15%	46 - 60%

- 開発速度と品質はトレードオフの関係ではない
- 組織間の差はかなり大きく、さらに開いている（2016 - 2019）
- 圧倒的な差は継続的デリバリやDevOpsへの組織的な投資の差

https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

4つのキーメトリクス: 2019年の調査におけるエリートクラスタとローパフォーマークラスタの差

2019

	エリート	エリートとローパフォーマーの差	ローパフォーマー
リードタイム	1日未満	106倍	1ヵ月から半年
デプロイ頻度	オンデマンド (1日複数回)	208倍	月1回から半年に1回
MTTR	1時間未満	1/2604	1週間から1ヵ月
変更失敗率	0 - 15%	1/7	46 - 60%

- 開発速度と品質はトレードオフの関係ではない
- 組織間の差はかなり大きく、さらに開いている（2016 - 2019）
- 圧倒的な差は継続的デリバリやDevOpsへの組織的な投資の差

https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

Google Cloud

2021

Accelerate

State of DevOps 2021

Sponsored by armory circleci redgate Deloitte.

sysdig GitLab Liquibase PagerDuty CD.FOUNDATION

https://cloud.google.com/devops/state-of-devops/

4つのキーメトリクス: 2021年の調査（3万2千件以上の回答をクラスター分析）

2021

	エリート	ハイパフォーマー	ミディアム パフォーマー	ローパフォーマー
リードタイム	1時間未満	1日から1週間	1ヵ月から半年	半年以上
デプロイ頻度	オンデマンド (1日複数回)	週1回から月1回	月1回から半年に1回	半年に1回未満
MTTR	1時間未満	1日未満	1日から1週間	半年以上
変更失敗率	0 - 15%	16 - 30%	16 - 30%	16 - 30%

- 開発速度と品質はトレードオフの関係ではない
- 組織間の差はかなり大きく、さらに開いている（2016 - 2021）
- 圧倒的な差は継続的デリバリやDevOpsへの組織的な投資の差

4つのキーメトリクス: 2021年の調査におけるエリートクラスタとローパフォーマークラスタの差

2021

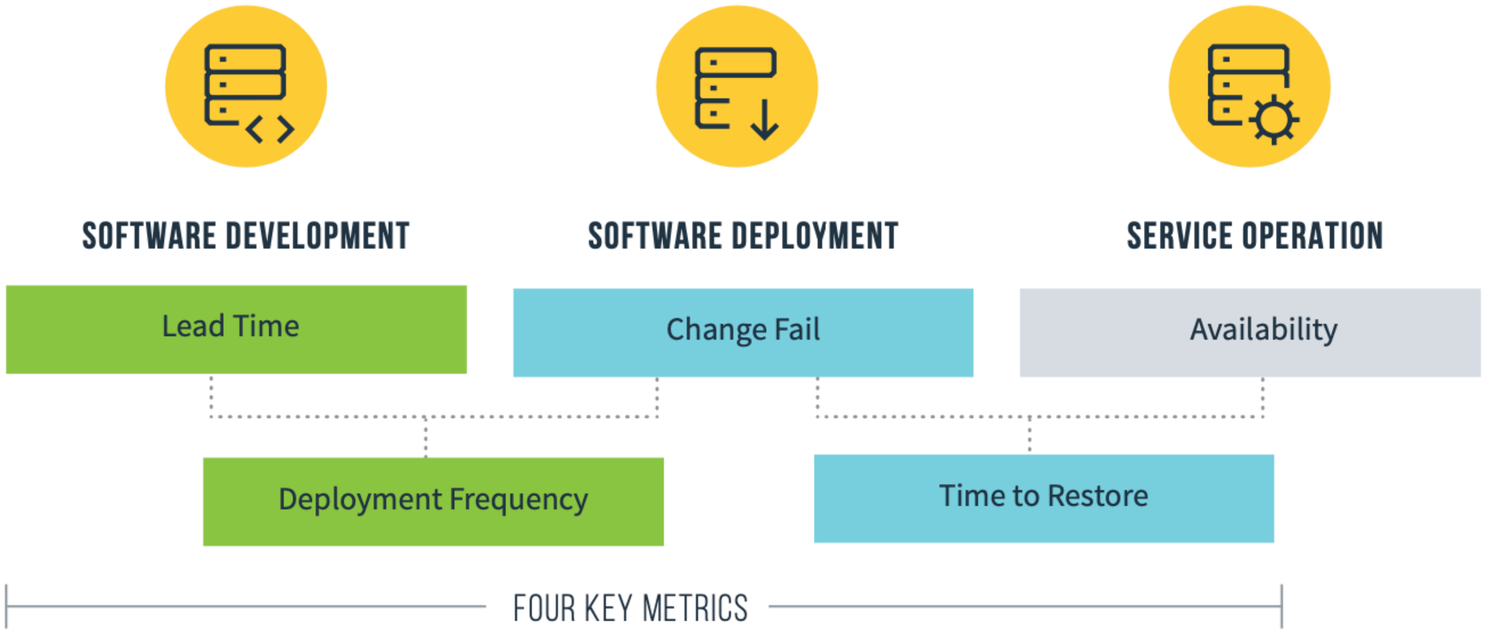
	エリート	エリートとローパフォーマーの差	ローパフォーマー
リードタイム	1時間未満	6570倍	半年以上
デプロイ頻度	オンデマンド (1日複数回)	973倍	半年に1回未満
MTTR	1時間未満	1/6570	半年以上
変更失敗率	0 - 15%	1/3	16 - 30%

- 開発速度と品質はトレードオフの関係ではない
- 組織間の差はかなり大きく、さらに開いている（2016 - 2021）
- 圧倒的な差は継続的デリバリやDevOpsへの組織的な投資の差

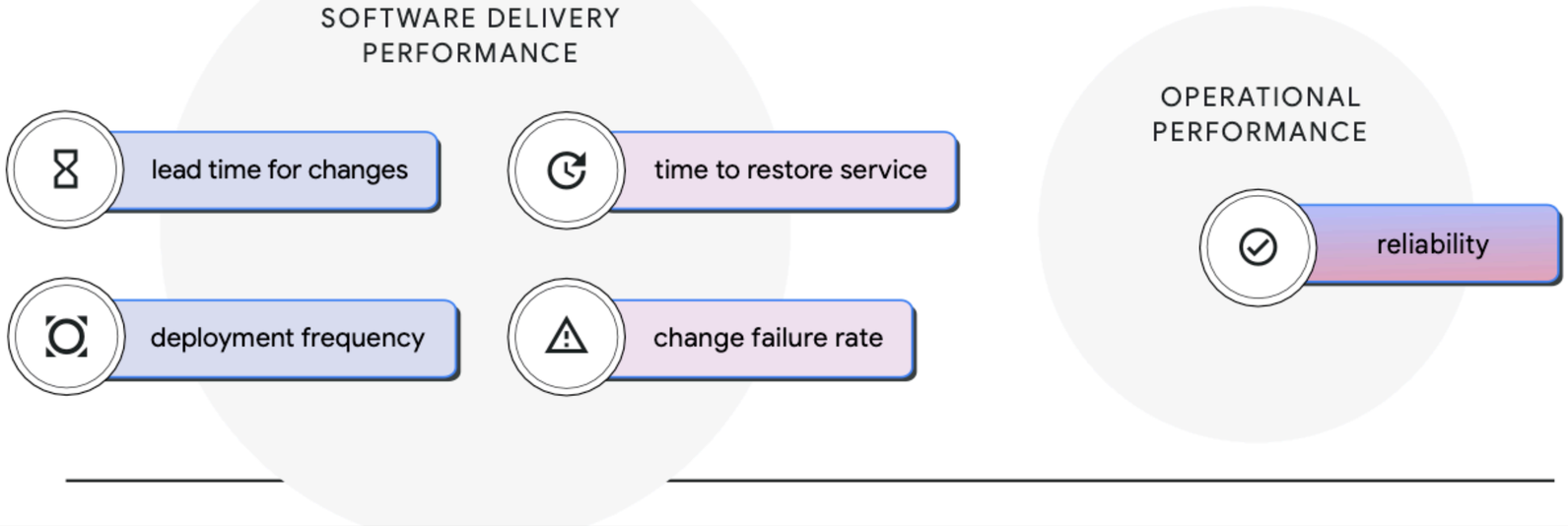
https://cloud.google.com/devops/state-of-devops/

5つめのメトリクス: Reliability

2019



2021



ソフトウェアデリバリの能力は、組織に競争上の優位性をもたらすことが立証された

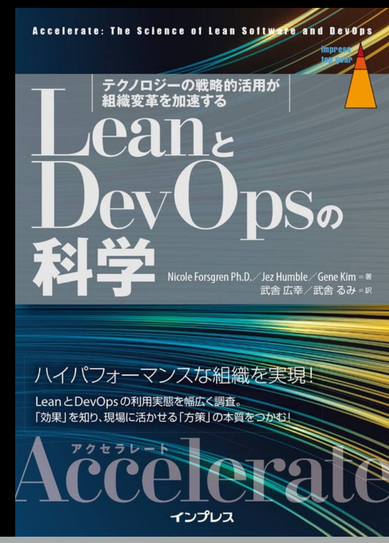
組織のパフォーマンスの測定指標は収益性、市場占有率、生産性の3つ

これら3つは過去の調査研究[Widener 2007]で複数回にわたり有効性が立証されており、投資利益率(ROI)との高い相関があり、景気の影響を受けない

ここ数年の分析で、ハイパフォーマーのパフォーマンスの測定結果がローパフォーマーのそれを上回る傾向にあり、両社の対比は一貫して2倍を超えている

これにより、組織のソフトウェアデリバリのケイパビリティ(能力、機能)は、組織に競争上の優位性をもたらすことが立証された

『LeanとDevOpsの科学』 p.31



https://www.amazon.co.jp/dp/4295004901/

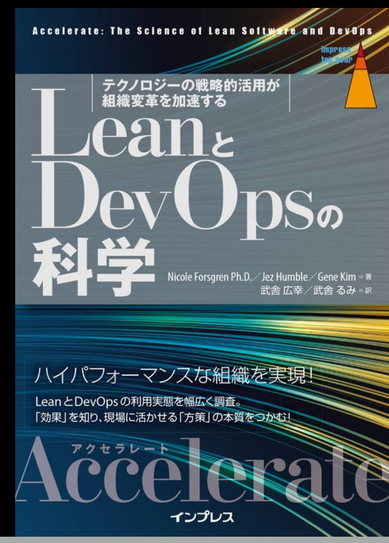
でもうちの(会社|業務|ソフトウェア)は特殊だから……

システムのタイプとパフォーマンスには相関がない

我々はパッケージソフトウェア、SoR、組み込みソフトウェアを使っているチームはパフォーマンスが低く、SoEや新種のシステムを使っているチームはパフォーマンスが高いと予測していたからである。そしてその予測が外れていたことを立証するデータが得られた。

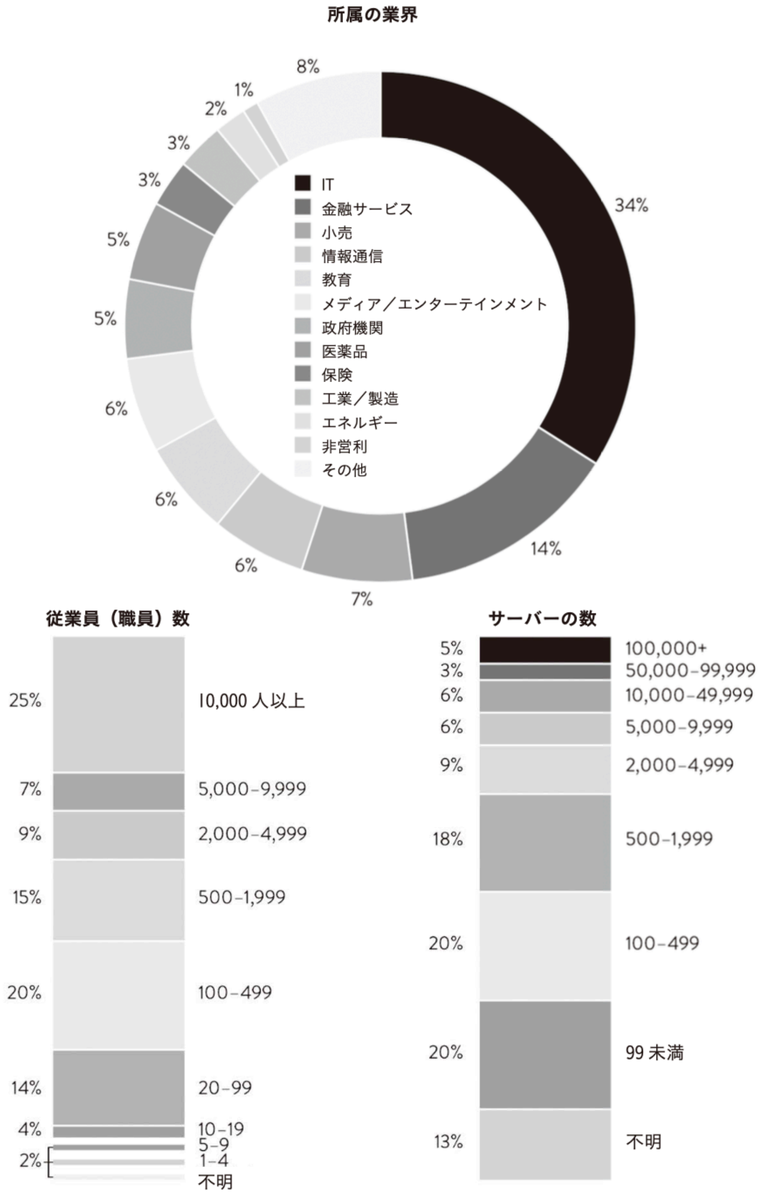
これによって、アーキテクチャの実装の詳細よりも
（このあと説明する２つの）アーキテクチャが持つ特性に
注目することの重要性が浮き彫りになった。

『LeanとDevOpsの科学』 p.74



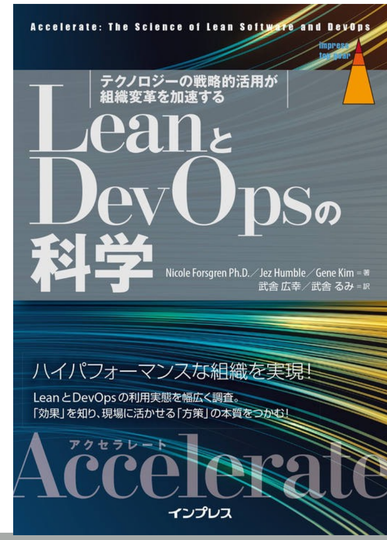
https://www.amazon.co.jp/dp/4295004901/

システムのタイプとパフォーマンスには相関がない



図B.1　企業特性: 組織の規模、所属業界、サーバーの数（2017年）

- 図B.1は2017年度のデータの企業特性である。注目すべき点は、この年度のすべてのグループに「ハイパフォーマー」「ミディアムパフォーマー」「ローパフォーマー」の該当者がいたということである。つまり、大企業にもスタートアップにも規制の厳しい業界（金融、医療、情報通信など）にも、「ハイパフォーマー」「ミディアムパフォーマー」「ローパフォーマー」の該当者がいたということであり、重要なのは所属の業界でも組織の規模でもないということである。たとえば、規制の厳しい業界に属する大規模な組織でも、ソフトウェアの開発とデリバリのパフォーマンスを非常に高いレベルにまで向上させうるのであり、関連のケイパビリティを強化して顧客にも組織全体にも価値を提供しうるのである。



『LeanとDevOpsの科学』 pp.255-256

https://www.amazon.co.jp/dp/4295004901/

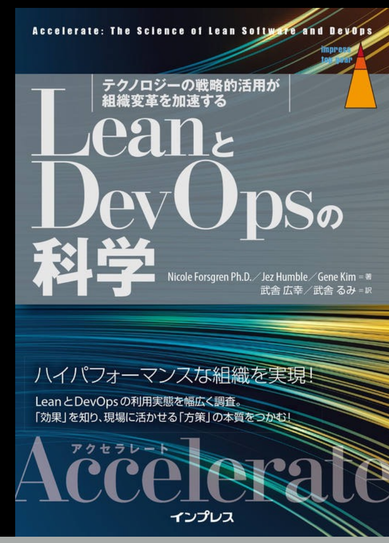
テスト容易性とデプロイ容易性（デプロイ独立性）

アーキテクチャの特性の中には大きな影響を及ぼしうるものが2つある。我々の調査では、次の2つの事項に「同意できる」と回答した組織であればハイパフォーマーである可能性が高い、という結果が出たのである。

- テストの大半を、統合環境を必要とせずに実施できる
- アプリケーションを、それが依存する他のアプリケーションやサービスからは独立した形で、デプロイまたはリリースできる

アーキテクチャに関わるこの2つの特性を、我々は「テスト容易性」と「デプロイ容易性」と呼んでいるが、この2つがパフォーマンスを向上させる上で重要だと思われる。

『LeanとDevOpsの科学』 p.75



https://www.amazon.co.jp/dp/4295004901/

テスラの変更頻度とアーキテクチャ

1日60個以上の新しい部品がリリースされる

（スライドを示して）多くの自動車会社は、2年から3年に1度、マイナーモデルチェンジのイベントを行い、製造を更新しています。テスラは毎日、1回以上マイナーモデルチェンジイベントを実施しています。

毎日、60個の新しい部品が生産導入され、販売されています。そして毎日61個以上の部品が削除されます。商品の総数は、毎日減少します。これらの商品には、エレクトロニクス、ソフトウェア、およびサプライヤーから提供された部品も含まれます。

通常、部品のチェンジにだいたい2年から7年かけていると思います。テスラでは、ヘッドライト、テールライト、あと充電口などを2日で変更します。2日で変更というのは、設計から製造、そしてテスト、リリースまで2日で行ってしまうということです。

テスラの「Model 3」や「Model Y」などの充電口は3時間でアップデートが行われます。そして、ソフトウェアや組み込みも含めて1日60個以上の新しい部品がリリースされます。

https://logmi.jp/tech/articles/327160

テスラの変更頻度とアーキテクチャ

Joe Justice,
WIKISPEED
Model

AEROSHELL
エアロシェル

DRIVE
TRAIN
MODULE
駆動系モジュール

内装モジュール
INTERIOR
MODULE

REAR CRUSH
STRUCTURE
リア・クラッシュ
ストラクチャー

CHASSIS
シャシー

PEDAL PLATE
ペダルプレート

カンパニー・アズ・ア・サービス・チーム
Company As a Service Teams

コーヒー
Coffee
Team

施設紹介
Facilities
Team

グルメ
Gourmet
Team

資金調達
Funding
Team

SUSPENSION
MODULE
サスペンション

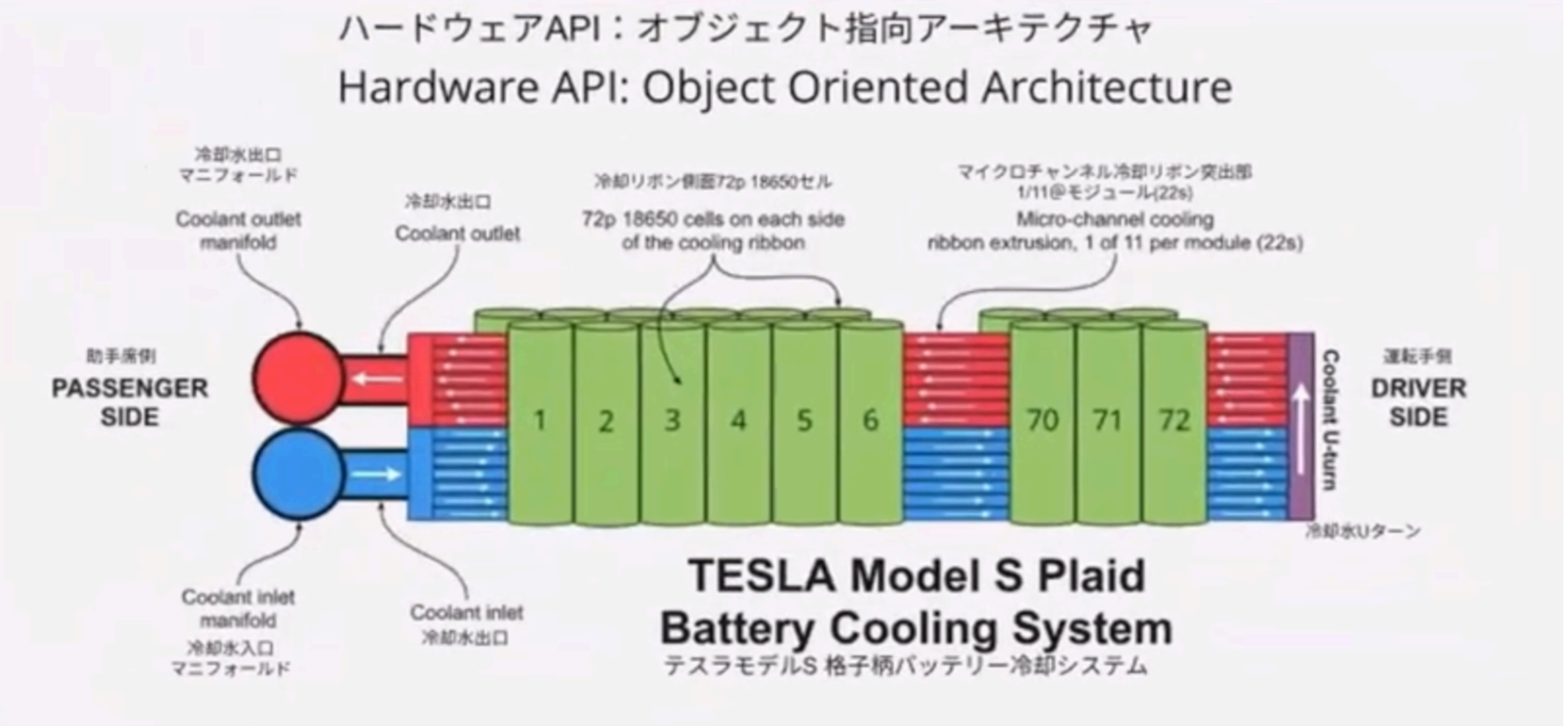
FRONT CRUSH
STRUCTURE
フロントクラッシュ
ストラクチャー

https://logmi.jp/tech/articles/327160

テスラの変更頻度とアーキテクチャ

（スライドを示して）「Model S Plaid」のバッテリーパックの図です。

この冷却水の出口と入口が互換性があるので、このモデルを変えたとしても、いつでも新しいものに更新できます。Web開発でも同様だと思いますが、アダプターをつけて、インプット・アウトプットに互換性を持たせれば新しいものがどんどん更新できます。モジュラー・アーキテクチャは、お互いを待つことなく数百のチームが並行して作業するのに有効です。

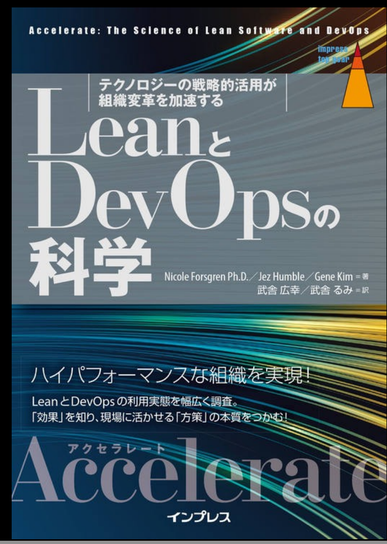


厳しい現実

ローパフォーマーとなる傾向が強いのは、次のように回答したチームであった

- 「 構築中のソフトウェア(あるいは利用する必要のある一群のサービス)は、他社(外注先など)が開発したカスタムソフトウェアである」

『LeanとDevOpsの科学』 p.73

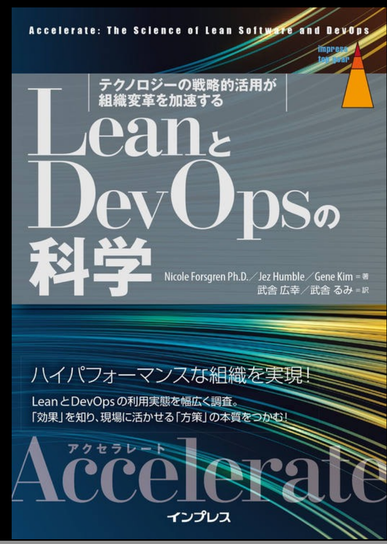


https://www.amazon.co.jp/dp/4295004901/

厳しい現実

「ソフトウェアデリバリのパフォーマンスは組織全体の業績に重要な影響を及ぼす」という事実は、自組織の事業にとって戦略上重要なソフトウェアの開発能力を自組織の中核的要素として位置付けずに外部委託することへの強力な反証となる。

『LeanとDevOpsの科学』 p.33



https://www.amazon.co.jp/dp/4295004901/

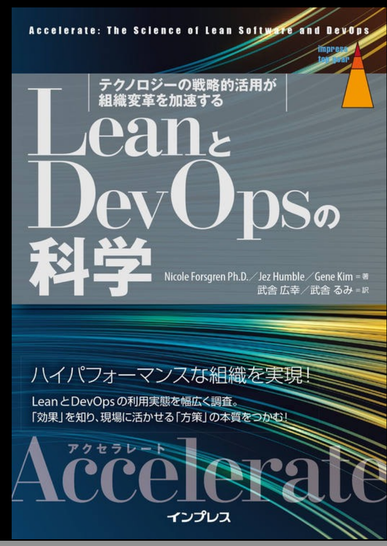
継続的デリバリ

継続的デリバリの5つの基本原則

「継続的デリバリ」とは、機能の追加、構成の変更、バグの修正、各種試行など、さまざまな変更を、安全かつ迅速かつ持続可能な形で本番環境に組み込んだりユーザーに提供したりする作業を促進する一群のケイパビリティから成る手法である。次の5つの基本原則を柱とする。

- 「品質」の概念を生産工程の最初から組み込む
- 作業は（小さい）バッチ処理で進める
- 反復作業はコンピュータに任せ人間は問題解決に当たる
- 徹底した改善努力を継続的に行う
- 全員が責任を担う

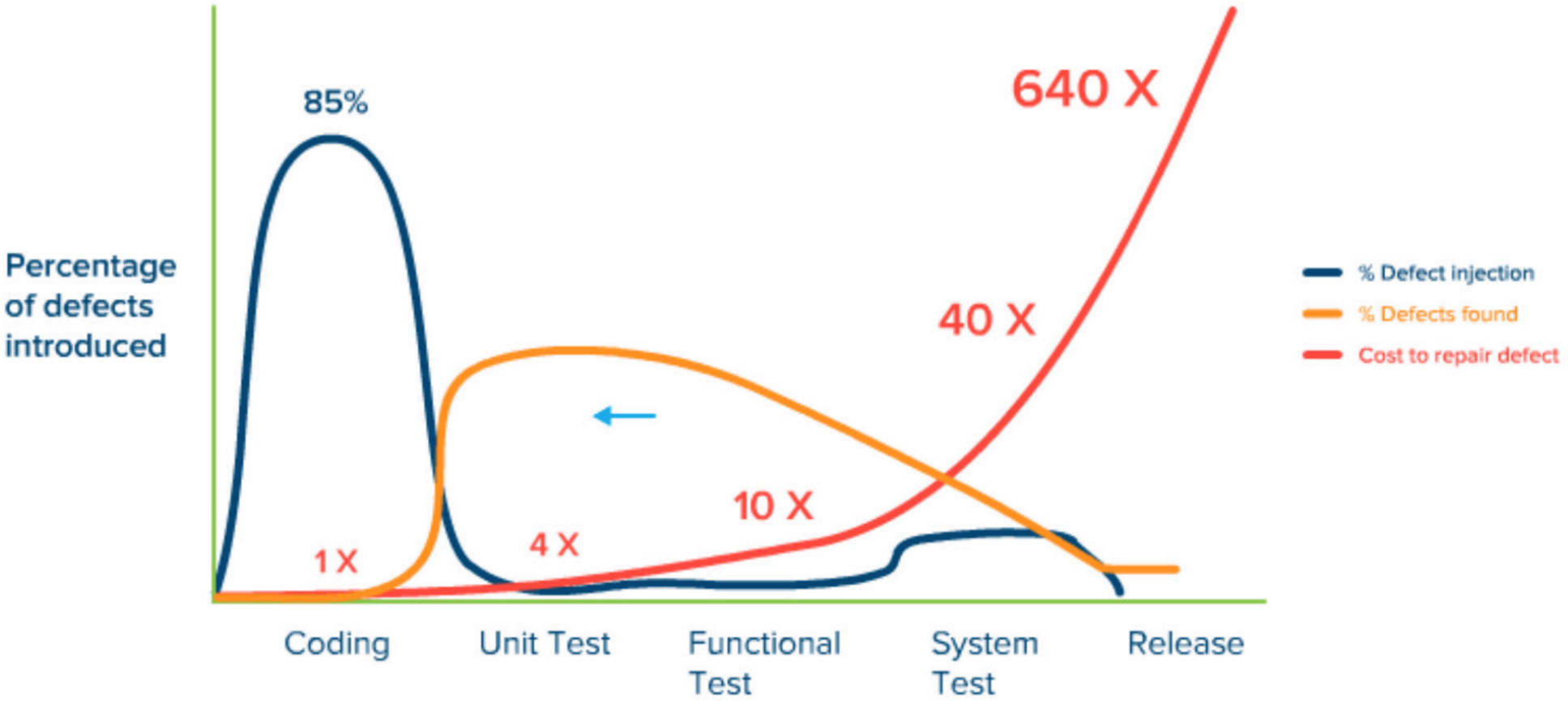
『LeanとDevOpsの科学』 pp.53-54



https://www.amazon.co.jp/dp/4295004901/

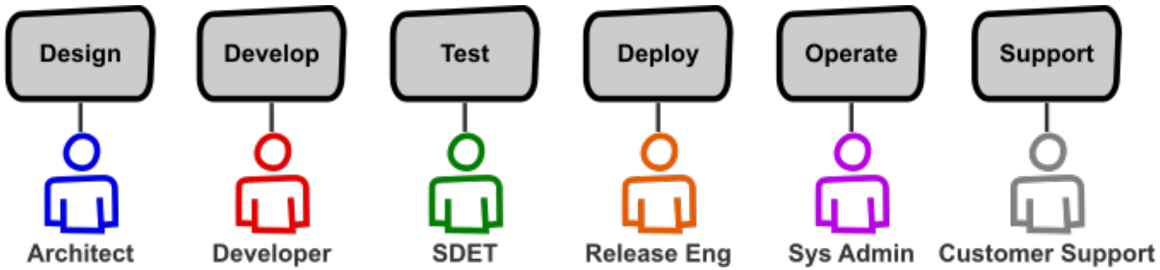
問題発見の時期をシフトレフトし継続的に取り組む

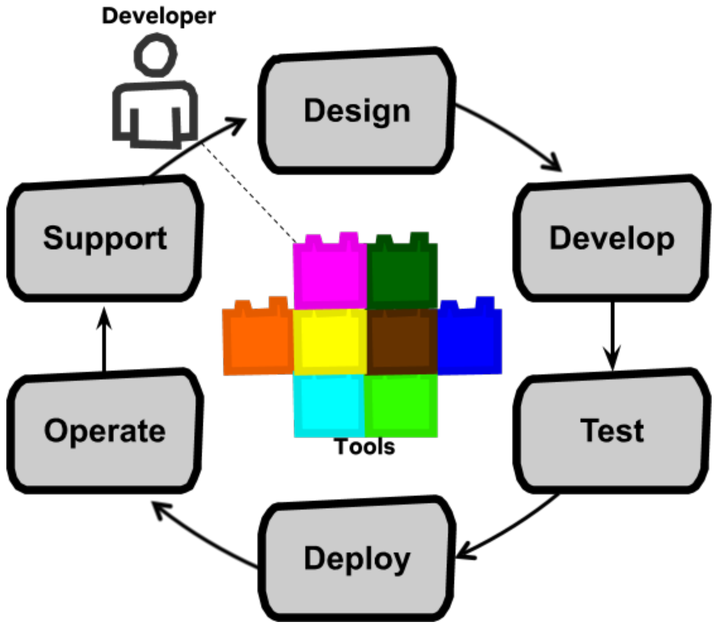
つまりテスト自動化を始めとした技術基盤が欠かせない



Capers Jones, Applied Software Measurement: Global Analysis of Productivity and Quality

全員が責任を担う: 分業からフルサイクルへ



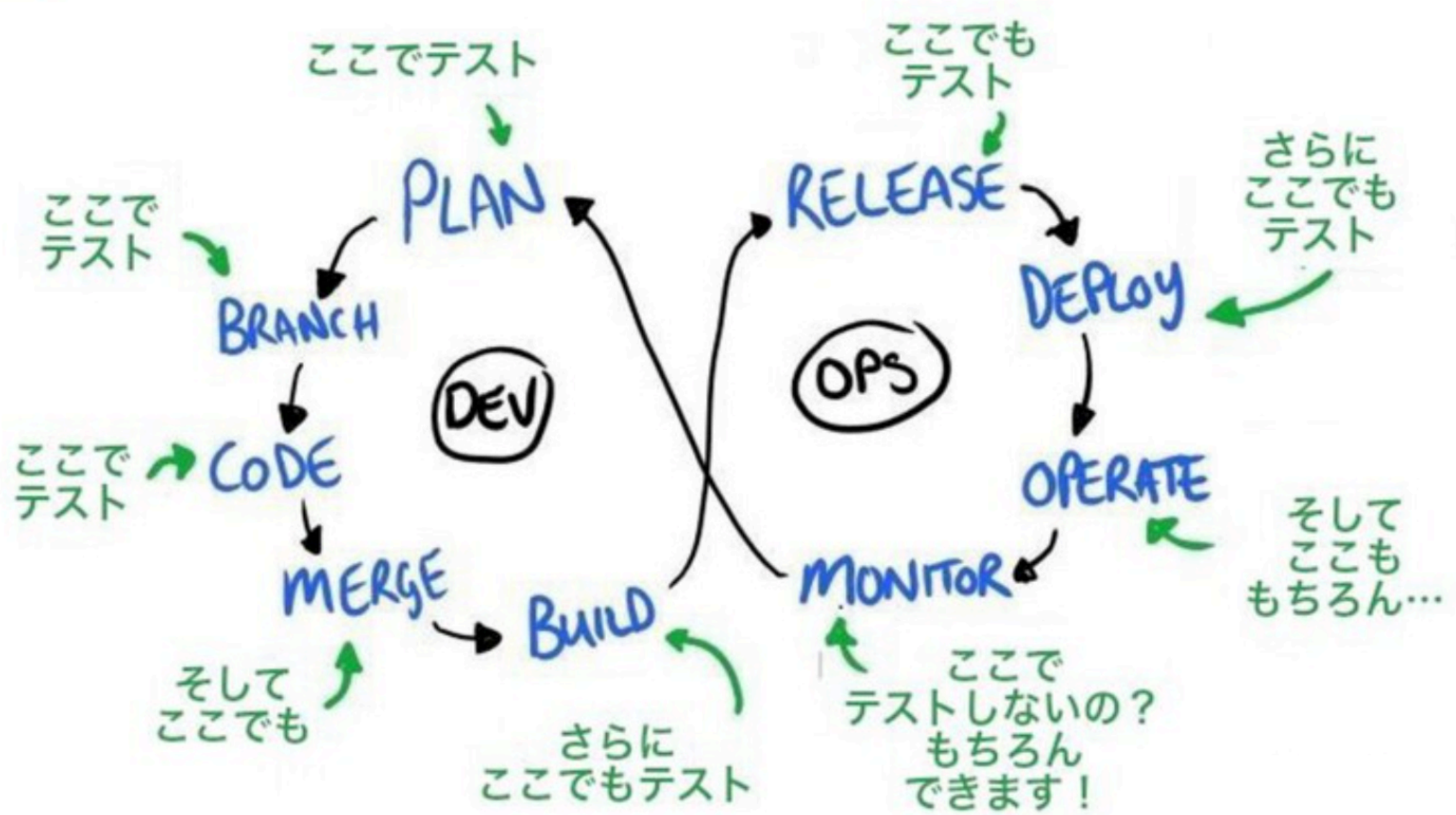


https://techblog.cartaholdings.co.jp/entry/2019/02/04/171325

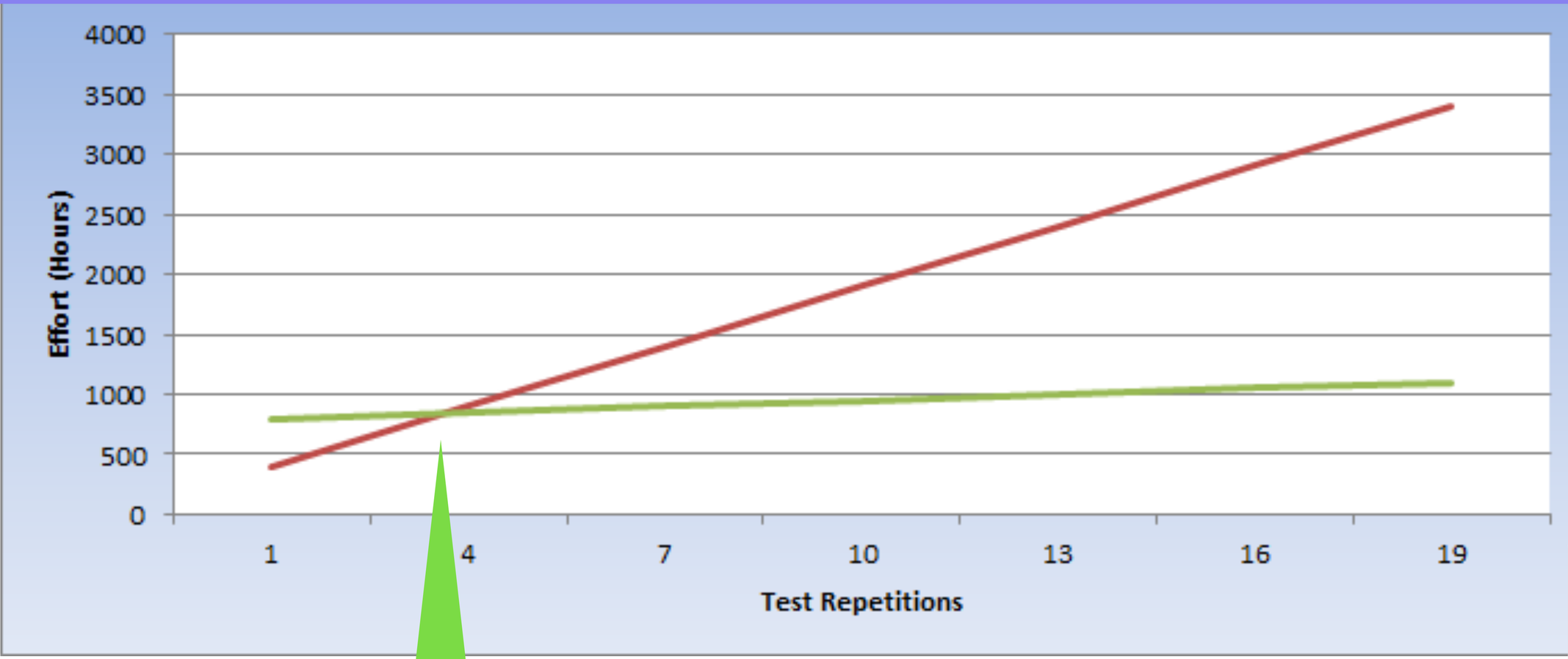
継続的デリバリ実践の3つの技術的基盤

- 包括的な構成管理
- 継続的インテグレーション
- 継続的テスト

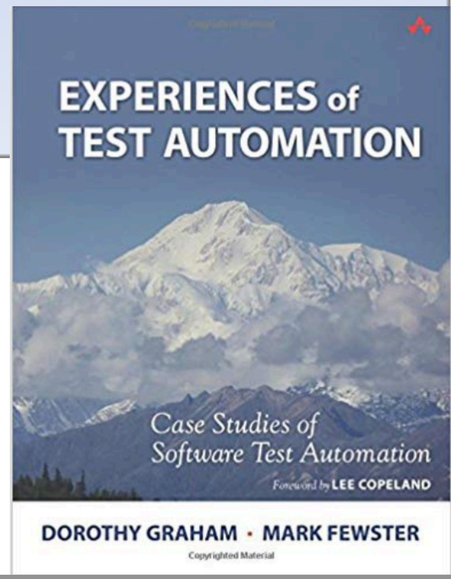
継続的テスト


ここでテスト
PLAN
ここで
テスト
BRANCH
ここで
テスト
CODE
MERGE
そして
ここでも
BUILD
さらに
ここでもテスト
DEV
OPS
ここでも
テスト
RELEASE
さらに
ここでも
テスト
DEPLOY
OPERATE
そして
ここも
もちろん…
MONITOR
ここで
テストしないの？
もちろん
できます！

常にテスト → 自動化の損益分岐点はあっという間に来る



およそ4回で
手動テストと自動化されたテストの
コストが逆転する



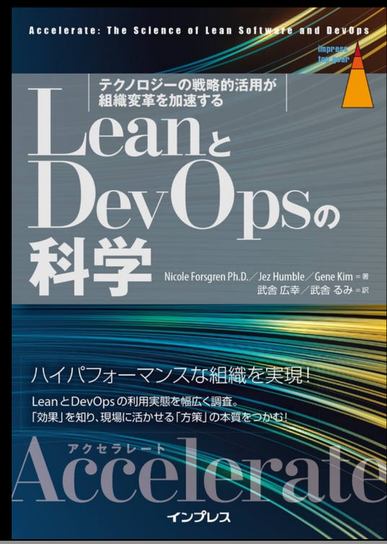
https://www.amazon.co.jp/dp/0321754069/

テスト自動化と企業の業績の因果関係

テストの自動化において、ITパフォーマンスの予測尺度となりうることが判明したのは次の2つ

1. 信頼性の高い自動テストを備えること
2. 開発者主体で受け入れテストを作成・管理し、
   手元の開発環境で簡単に再現・修正できること

『LeanとDevOpsの科学』 p.65（※訳を一部変更）



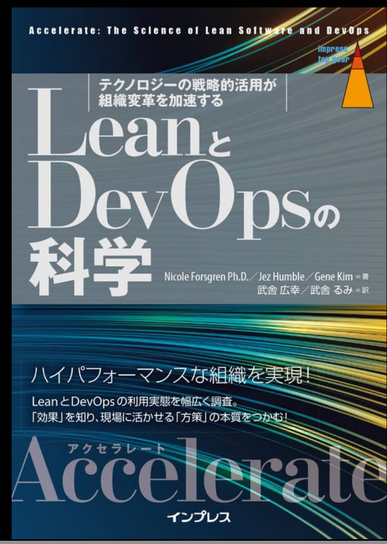
https://www.amazon.co.jp/dp/4295004901/

テスト自動化と企業の業績の因果関係

テストの自動化において、ITパフォーマンスの予測尺度となりうることが判明したのは次の2つ

1. 信頼性の高い自動テストを備えること
2. 開発者主体で受け入れテストを作成・管理し、
   手元の開発環境で簡単に再現・修正できること

『LeanとDevOpsの科学』 p.65（※訳を一部変更）

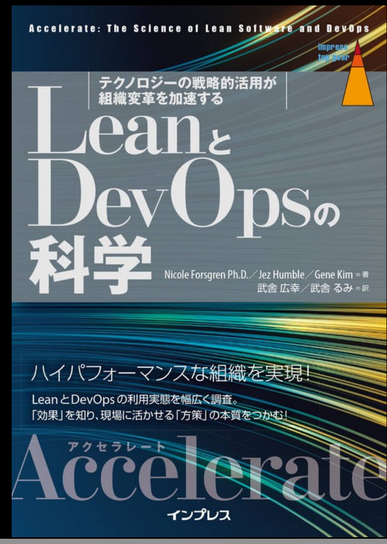


https://www.amazon.co.jp/dp/4295004901/

信頼性の高い自動テストを備えること

テストに合格したソフトウェアであればリリース可能、不合格であれば重大な不具合がある、とチームが確信できるようなテストを実施していること

1. 誤検知（偽陽性: false positive）や見逃し（偽陰性: false negative）が多く、信頼性に欠けるテストスイートがあまりにも多すぎる
2. 信頼度の高いテストスイートを作り上げる継続的な努力と投資は価値がある



『LeanとDevOpsの科学』 p.65（※訳を一部変更）

https://www.amazon.co.jp/dp/4295004901/

偽陽性と偽陰性

表1　偽陽性と偽陰性

テスト結果が ＼ プロダクトコードが	正しい	誤っている
成功	期待どおり	偽陰性
失敗	偽陽性	期待どおり

信頼不能性（flakiness）が1%に接近すると、テストは価値を失い始める

テストの信頼不能性が増加の一途をたどるようなら、生産性を失うよりもっとまずいことを経験することになるだろう。つまり、テストへの信頼の喪失である。チームがテストスイートへの信頼を失うまでに、そう多くの信頼不能テストの調査は要しない。テストへの信頼の喪失が起こると、エンジニアはテストの失敗に反応することをやめ、テストスイートの提供する価値が全部削がれてしまう。我々の経験が示唆するのは、信頼不能性が1%に接近すると、テストは価値を失い始めるということだ。Googleでは、信頼不能テストの割合は約0.15%あたりをうろついており、これは毎日数千の信頼不能テストが起こっているということだ。我々は、エンジニアリングを行う時間を信頼不能テストの修正のために積極的に投資するなどして、信頼不能テストを制御下にとどめるよう奮闘している。

『Googleのソフトウェアエンジニアリング』 p.255

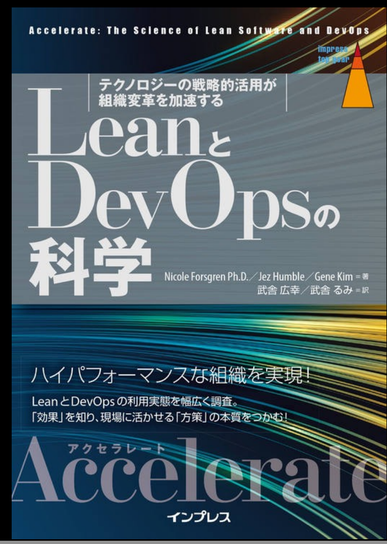
https://www.oreilly.co.jp/books/9784873119656/

テスト自動化と企業の業績の因果関係

テストの自動化において、ITパフォーマンスの予測尺度となりうることが判明したのは次の2つ

1. 信頼性の高い自動テストを備えること
2. 開発者主体で受け入れテストを作成・管理し、
   手元の開発環境で簡単に再現・修正できること

『LeanとDevOpsの科学』 p.65（※訳を一部変更）

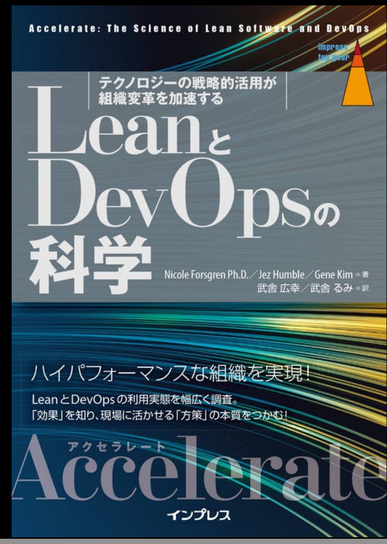


https://www.amazon.co.jp/dp/4295004901/

厳しい現実

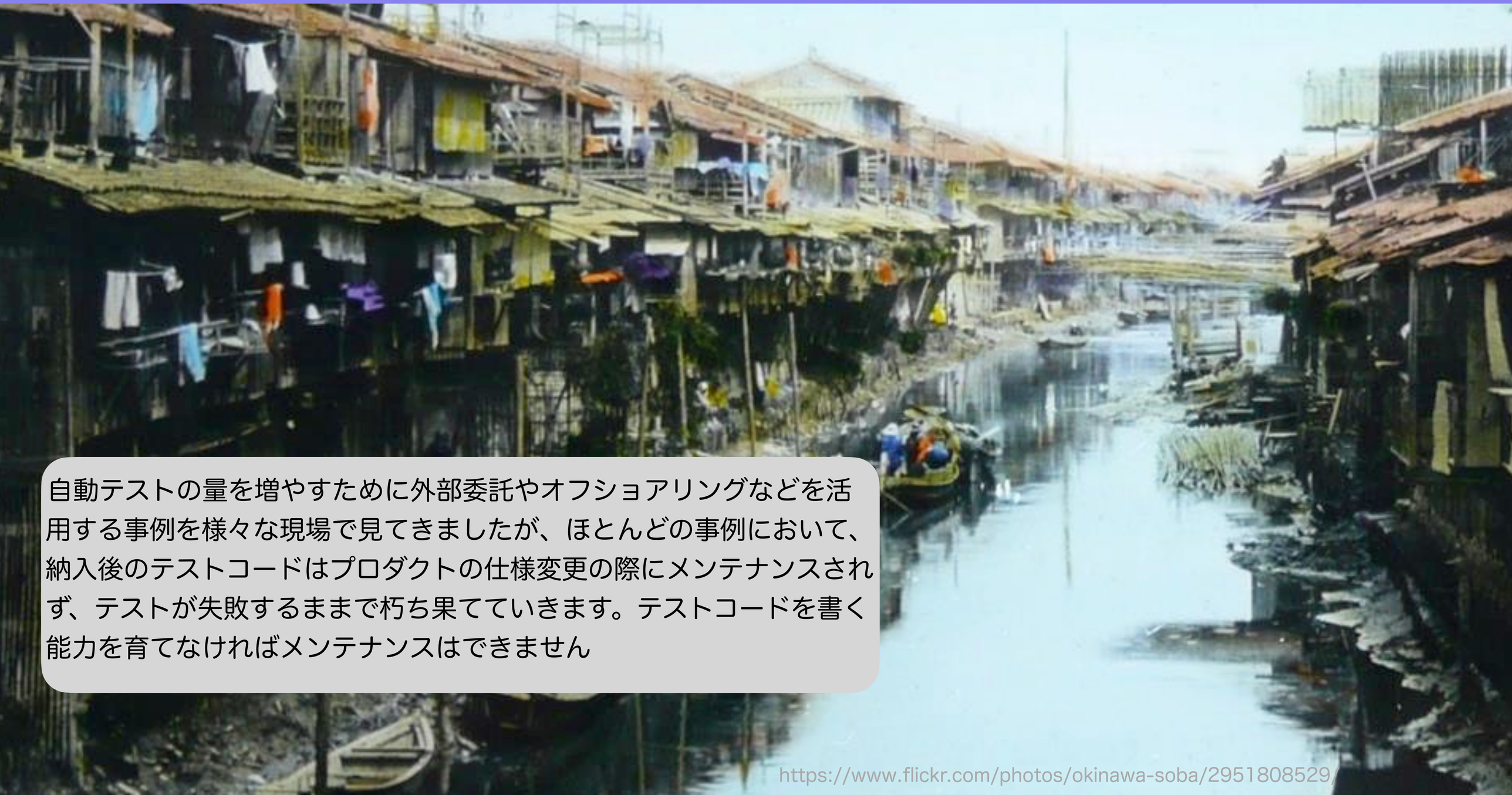
我々の分析結果の中でも興味深いのは「主としてQAチームか外注先が作成・管理している自動テストは、ITパフォーマンスと相関関係にない」という点である

『LeanとDevOpsの科学』 p.65



https://www.amazon.co.jp/dp/4295004901/

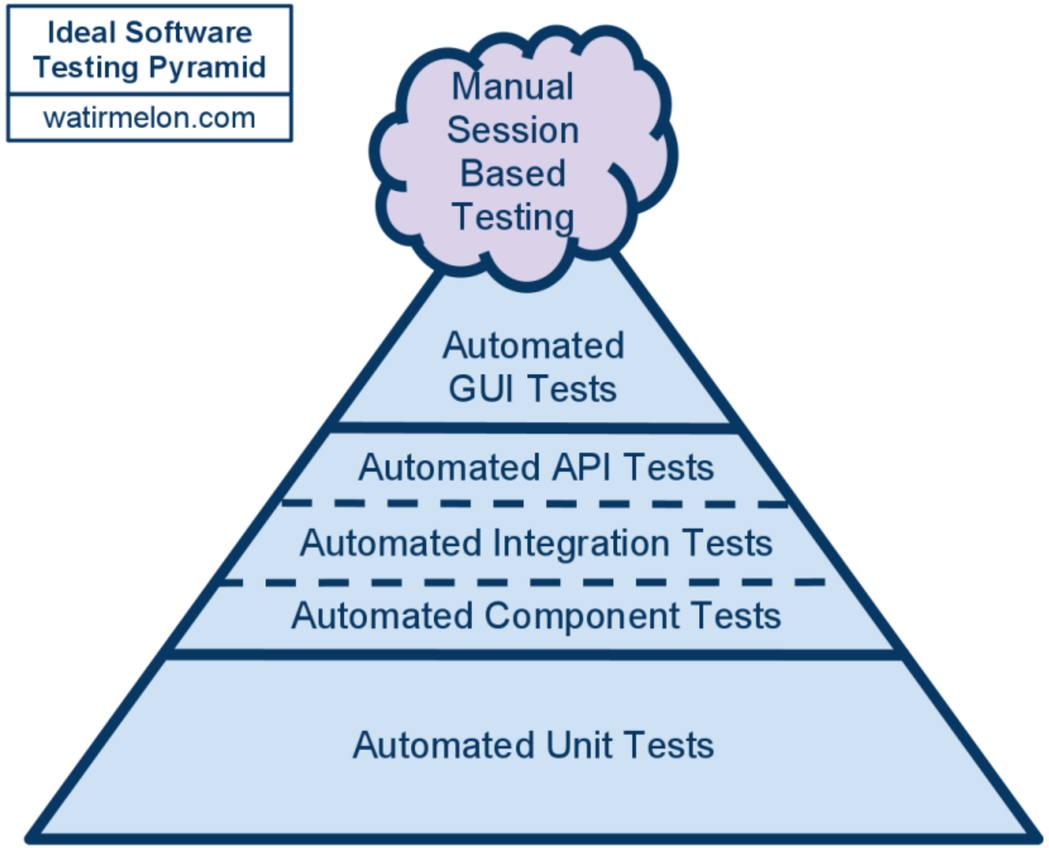
アンチパターン: 買ってきた自動テスト

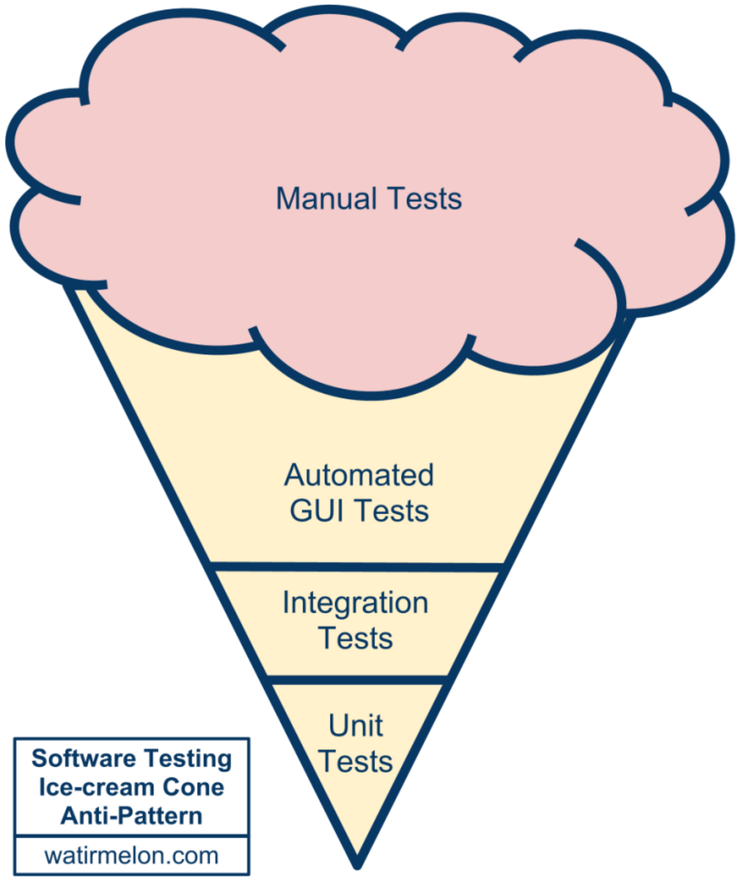


自動テストの量を増やすために外部委託やオフショアリングなどを活用する事例を様々な現場で見てきましたが、ほとんどの事例において、納入後のテストコードはプロダクトの仕様変更の際にメンテナンスされず、テストが失敗するままで朽ち果てていきます。テストコードを書く能力を育てなければメンテナンスはできません

https://www.flickr.com/photos/okinawa-soba/2951808529/

アンチパターン: アイスクリームコーン





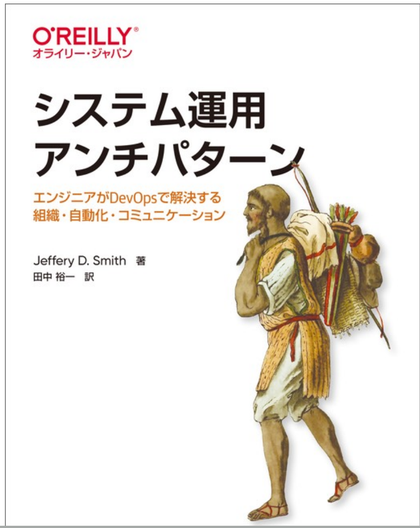
https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

アンチパターン: アイスクリームコーン

エンドツーエンドテストを重視してしまう背景

エンドツーエンドテストが多くなってしまう理由の1つは、テストの大部分を担当しているチームが開発チームではないことです。自動テストを行っている（すべてではないにしても）多くのQAチームは、UIテストに集中します。なぜならそれが彼らが慣れているアプリケーションやデータとのやりとりのしかただからです。

『システム運用アンチパターン』 p.83



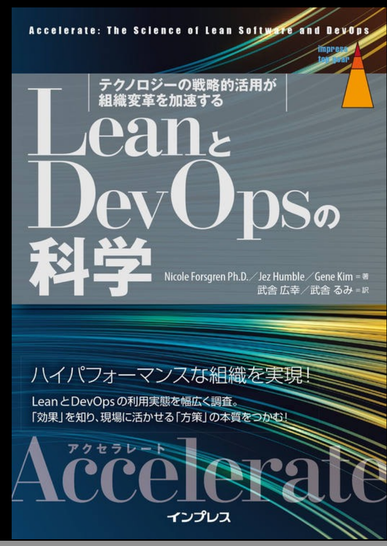
https://www.oreilly.co.jp/books/9784873119847/

開発者主体で受け入れテストを作成・管理し、手元の開発環境で簡単に再現・修正できること

開発者が受け入れテストの作成・管理に関与すると、2つの重要な効果が生じる

1. 開発者がテストを作成するとコードがよりテスト可能なものになる
2. 自動テストに対する責任を開発者が負うと、テストに対する意識が高まり、その管理や修正により注力することになる

『LeanとDevOpsの科学』 p.65（※訳を一部変更）



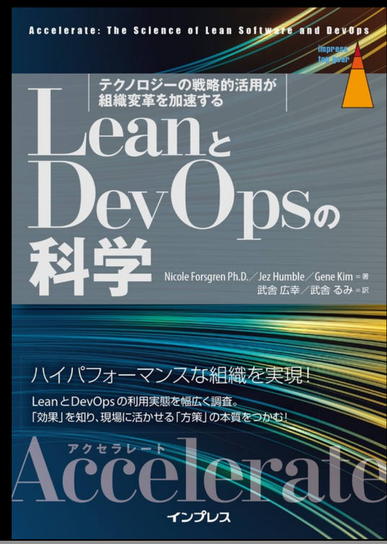
https://www.amazon.co.jp/dp/4295004901/

テストエンジニアの活躍の場は……？

テストエンジニアの役割

テスターは不要だと言っているわけではない。テスターはソフトウェアデリバリのライフサイクルにおいて不可欠な役割を果たす。探索型テスト、ユーザビリティテスト、承認（受け入れ）テストなどを手動で行い、開発者と協力して自動化テストスイートの作成と改良を助けるのである。

『LeanとDevOpsの科学』 p.66



https://www.amazon.co.jp/dp/4295004901/

アジャイルテストの4象限

https://www.shoeisha.co.jp/book/detail/9784798119977

探索的テスト: 未知から既知へ

The Pragmatic Programmers

Explore It!

Reduce Risk and
Increase Confidence with
Exploratory Testing

Elisabeth Hendrickson

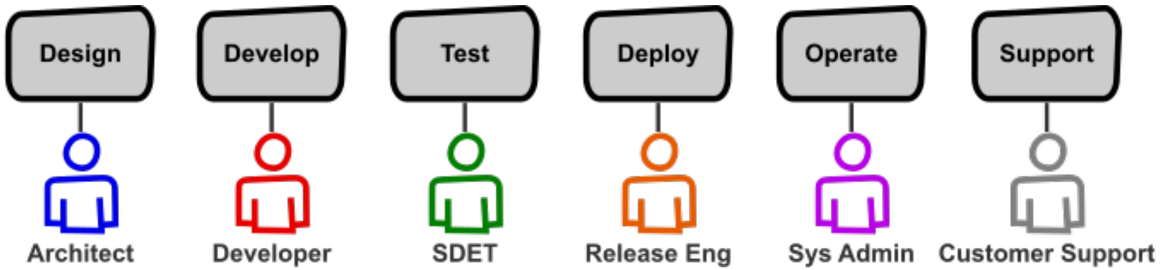
Edited by Jacquelyn Carter

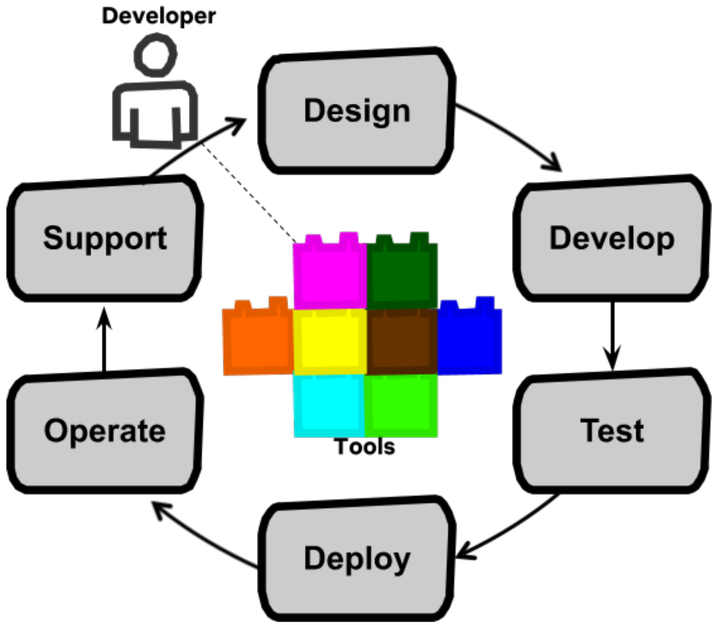
https://pragprog.com/titles/ehxta/explore-it/

The Testing Manifesto: チームの一員として

https://nihonbuson.hatenadiary.jp/entry/TestingManifesto

【再掲】全員が責任を担う: 分業からフルサイクルへ





https://techblog.cartaholdings.co.jp/entry/2019/02/04/171325

自動テストの動機

アンチパターン: コスト削減を主目的にする

自動テストを書く主目的や指標を「コスト削減」にすると、短期的には自動テストの学習コスト、中期的には保守コストによって思ったようなコスト削減効果が得られず、手動テストに戻るという判断をしてしまいがちです
(自動テスト以外の技術施策でもコスト削減を主目的にすると失敗しがちです)

https://www.flickr.com/photos/m4dgroup/6001102681/

変更容易性の高いソフトウェアによるアジリティの獲得

ところてん
@tokoroten

なんとなく、DXという言葉に対しての引っかかりが理解できてきた気がする
「デジタル」という言葉に「アジリティ」という意味が含まれていないので、
「変更容易性の高いソフトウェアによるアジリティの獲得」というDXの本質がソフトウェアエンジニア以外には伝わってない感じなんだな

- デジタルを活用したダブルループ学習によるアジリティの獲得
  - デジタルネイティブな組織は、その中に学習し続け、自らを軌道修正していく仕組みを構築（DevOps、スクラム、リーンスタートアップ、）
  - コンピュータシステムを市場環境の変化に合わせて柔軟に変化させていく
  - ITシステムの改善速度を上げていくことで、市場競争力を確保

https://twitter.com/tokoroten/status/1385826623451111424

アジリティの本質: あらゆるレベルで躊躇なく変化する

すべてをアジャイルな形で機能させるには、優れた設計に向けたプラクティスを実践する必要があります。というのも、優れた設計によって変更が容易になるためです。そして変更が容易である場合、あらゆるレベルで躊躇なく調整が可能になるのです。

それこそがアジリティーというものなのです。

『達人プログラマー 第2版』p.336

自動テストの動機: 常に変化を可能にするため

自動テストにより、バグが野に放たれてユーザーに影響を及ぼすことを防げる。バグは、捕捉するのが開発サイクルの後の方になればなるほど、コストが高くつくようになる。そして多くの場合、コストの増加は指数関数的である†1。しかし、「バグを捕捉すること」は、テストの動機の一部にすぎない。ソフトウェアのテストを望む理由として同等に重要なものは、変化を可能とする能力を備えておくため、という理由である。新機能追加、コードの健全性に主眼を置いたリファクタリングの実施、比較的大規模な再設計への着手と、どれを行っていようが、自動テストは間違いを素早く捕捉でき、そのおかげで信頼性を保ちつつソフトウェアを変更できるようになる。

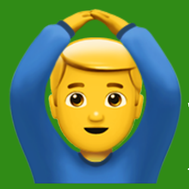
より高速に反復できる企業こそが、変化する技術、市場の状況、顧客の好みに対し、より迅速に適応できる。しっかりとしたテストのプラクティスがあれば、変化は恐れるに足らない。変化を、ソフトウェア開発の本質的特性として受け入れることができるのだ。システムをより大幅かつ高速に変化させたいと望むほど、システムに対し高速にテストを行う方法の必要性が増す。

『Googleのソフトウェアエンジニアリング』 p.243

https://www.oreilly.co.jp/books/9784873119656/

なぜ自動テストを書くのか


コストを削減するため


素早く躊躇なく変化し続ける力を得るため

自動テスト文化とは

自動テスト文化とは

大量のテストスイートの作成と保守には大変な労力が要る。コードベースが大きくなるにつれて、テストスイートもまた大きくなるだろう。テストスイートは、不安定さや遅延等の問題に直面し始めることだろう。それらの問題への対処を怠ると、テストスイートは役立たずになってしまう。テストの価値は、エンジニアがテストに寄せる信頼に由来していることを心に留めておくべきだ。テストが生産性を吸い込む底なし沼となり、トイル[†2]と不確実性を絶えず誘発するようになれば、エンジニアはテストへの信頼を失くし、ワークアラウンドを探し始めるだろう。駄目なテストスイートは、テストスイートが全くない場合よりたちが悪い。

『Googleのソフトウェアエンジニアリング』 pp.244-245

要約すると、健全な自動テスト文化は、テストを書く作業の分担共有を皆に促す。そのような文化は、定期的なテスト実行をも担保する。最後に、そしてともすれば最も重要な点として、そのような文化は、テストのプロセスに存在する高い信頼性を維持するために、破綻したテストの迅速な修正を重視している。

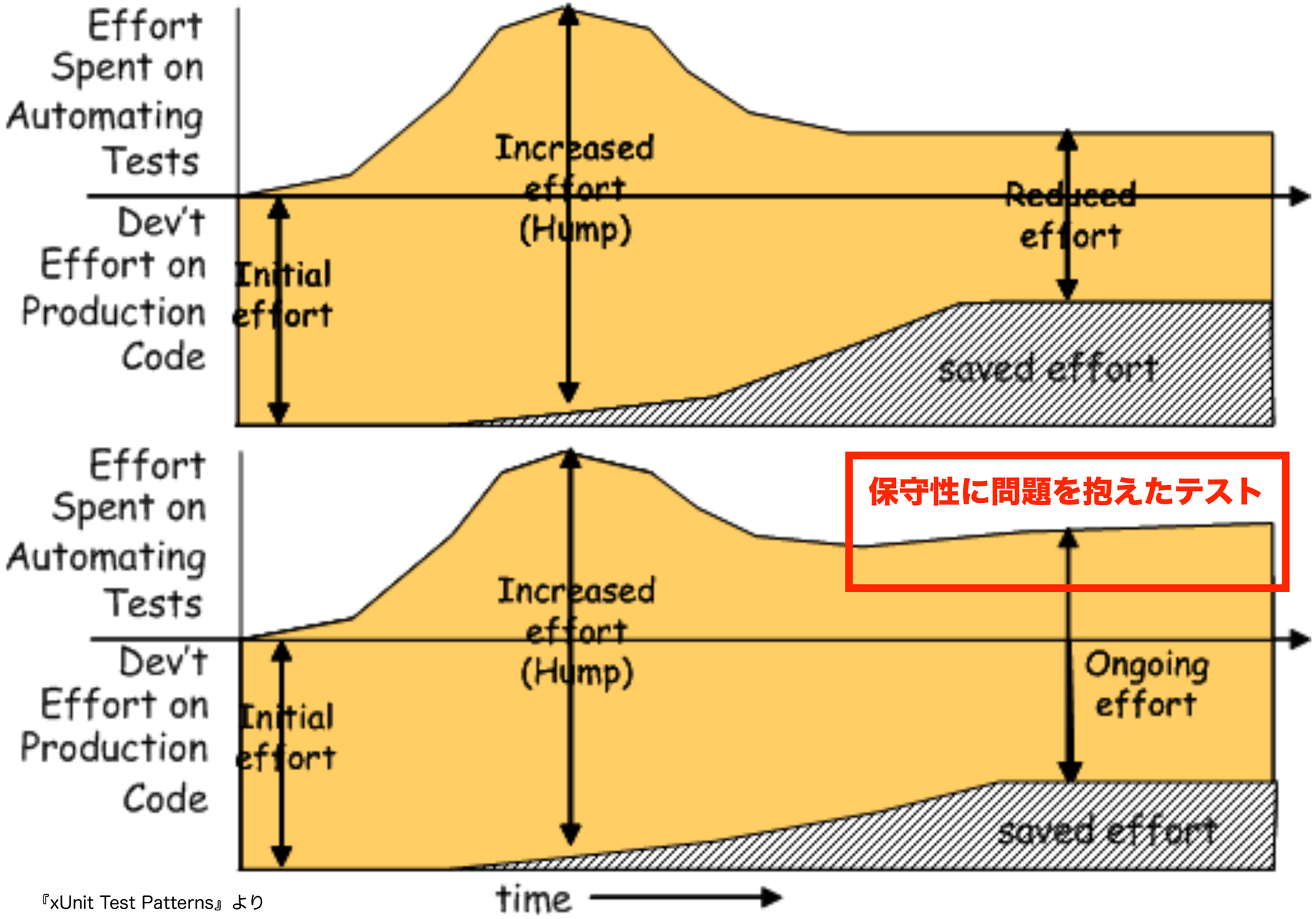
『Googleのソフトウェアエンジニアリング』 p.249

https://www.oreilly.co.jp/books/9784873119656/

テストコードのメンテナンスと向き合う

理想

現実



『xUnit Test Patterns』より

https://www.amazon.co.jp/dp/0131495054

自動テスト文化とは

時間の経過とともに、テストはGoogleのエンジニアリング文化のなくてはならない部分となってきた。全社で、エンジニアにとってのテストの価値を補強する無数の手段がある。訓練、穏やかな勧奨、メンター制度、そしてご存知の通り若干友好的な競争すら通じて、テストは皆の仕事であるという明確な期待水準を我々は作ってきた。

何故、テストを書くことを命令して強制するところから始めなかったのだろうか。

テスト小グループは、テストに関する命令を出すようシニア幹部陣に求めることを検討したが、すぐにそれは行わないことに決めた。コードの開発方法についての強制は、どんなものであろうが、Googleの文化に真っ向から逆らうものであると同時に、おそらく進歩を鈍化させるものだ。それはどんな思想が強制されるかに関係ない。成功する思想は広まるものであるというのが我々の信念であり、したがって専念すべき点は、成功を実証してみせることとなった。

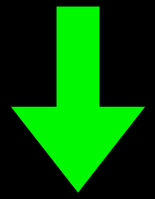
エンジニアが自身でテストを書くことを決めていた場合、それが意味するのは、テストの思想をエンジニアが全て受け入れ、おそらくは正義を為し続けていくということなのだった。しかも、誰にも強制されなかったとしてもそうするのだ。

『Googleのソフトウェアエンジニアリング』 p.268

https://www.oreilly.co.jp/books/9784873119656/

自動テスト文化とは

- テスト自動化に夢を見ない
- 自動テストのメンテナンスに全員が腹落ちする
- 自動テストのメンテナンスコストを下げる努力を怠らない



自動テスト文化とは、自動テストの重要性と保守性（理解容易性、変更容易性）を組織、チームが理解し、改善努力を継続的に行うこと

テストにコストがかかることの解決方法は、テストをやめることではありません。うまくなることです。

-- Sandi Metz

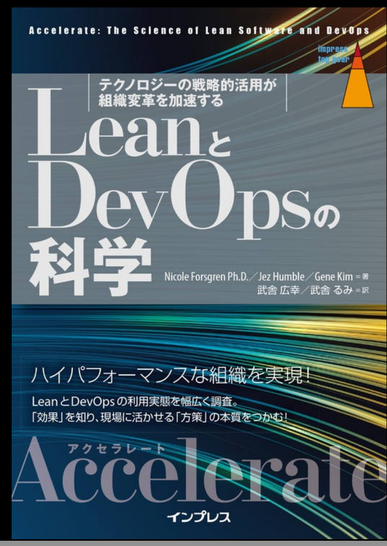
『オブジェクト指向設計実践ガイド』 p.239

https://gihyo.jp/book/2016/978-4-7741-8361-9

【再掲】継続的デリバリの5つの基本原則

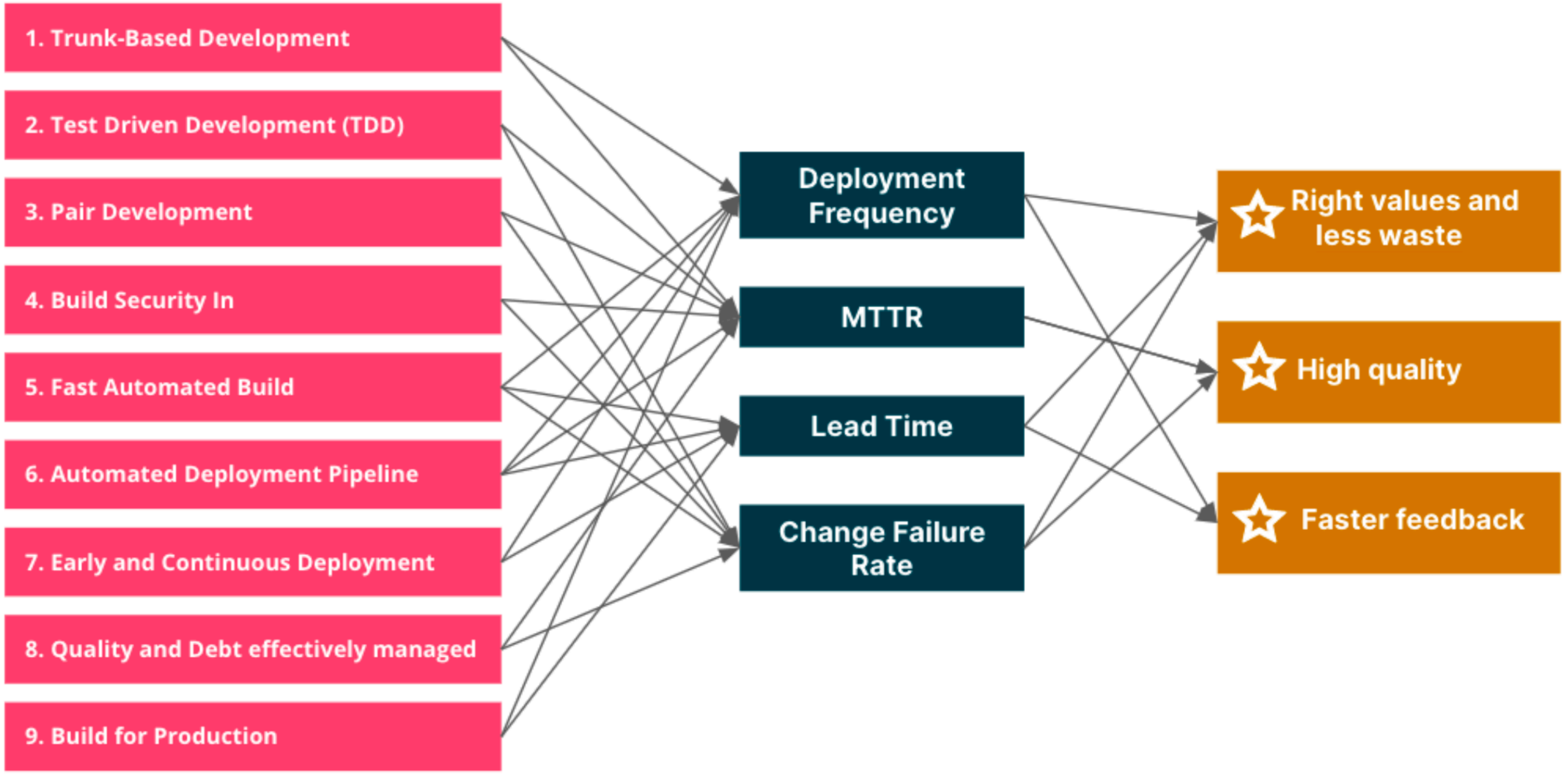
- 「品質」の概念を生産工程の最初から組み込む
- 作業は（小さい）バッチ処理で進める
- 反復作業はコンピュータに任せ人間は問題解決に当たる
- 徹底した改善努力を継続的に行う
- 全員が責任を担う

『LeanとDevOpsの科学』 pp.53-54



https://www.amazon.co.jp/dp/4295004901/

North Star を目指す



https://www.thoughtworks.com/insights/articles/engineering-productivity-governance-and-improvement-in-software-

ご清聴ありがとうございました

- 不確実性の高い時代においては、計画も設計も実装も常に変化し続けられる力が重要となる
- 継続的デリバリを構成するケイパビリティ（能力）が企業の業績（収益性、市場占有率、生産性）に因果関係があることが研究によって立証された
- 企業の規模や属する業界、システムのタイプはデリバリのパフォーマンスに関係なく、システムのアーキテクチャが備える2つの特性（テスト容易性、デプロイ独立性）が関係していることが判明した
- 自組織の事業にとって戦略上重要なソフトウェアの開発能力を自組織の中核的要素として位置付けずに外部委託すると、有意にパフォーマンスが下がることも判明した
- 自動テストを開発者主体で書き、実行結果の信頼性が高い状態を保つことがデリバリのパフォーマンスと因果関係がある
- テストエンジニアはチームに参加し、プロセス全体の品質に主体的に関与することが望まれる
- 自動テストを書く理由は、コスト削減ではなく、アジリティ（あらゆるレベルで素早く躊躇なく変化し続ける力）を得るため
- 自動テスト文化とは、自動テストの重要性と保守性（理解容易性、変更容易性）を組織、チームが理解し、改善努力を継続的に行うこと

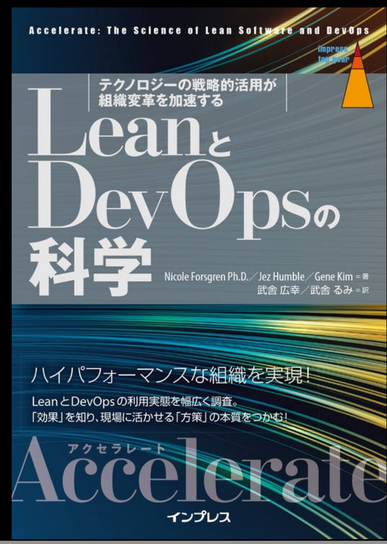
Q&A用
スライド

技術から文化へ

技術「が」文化「に」影響を与える

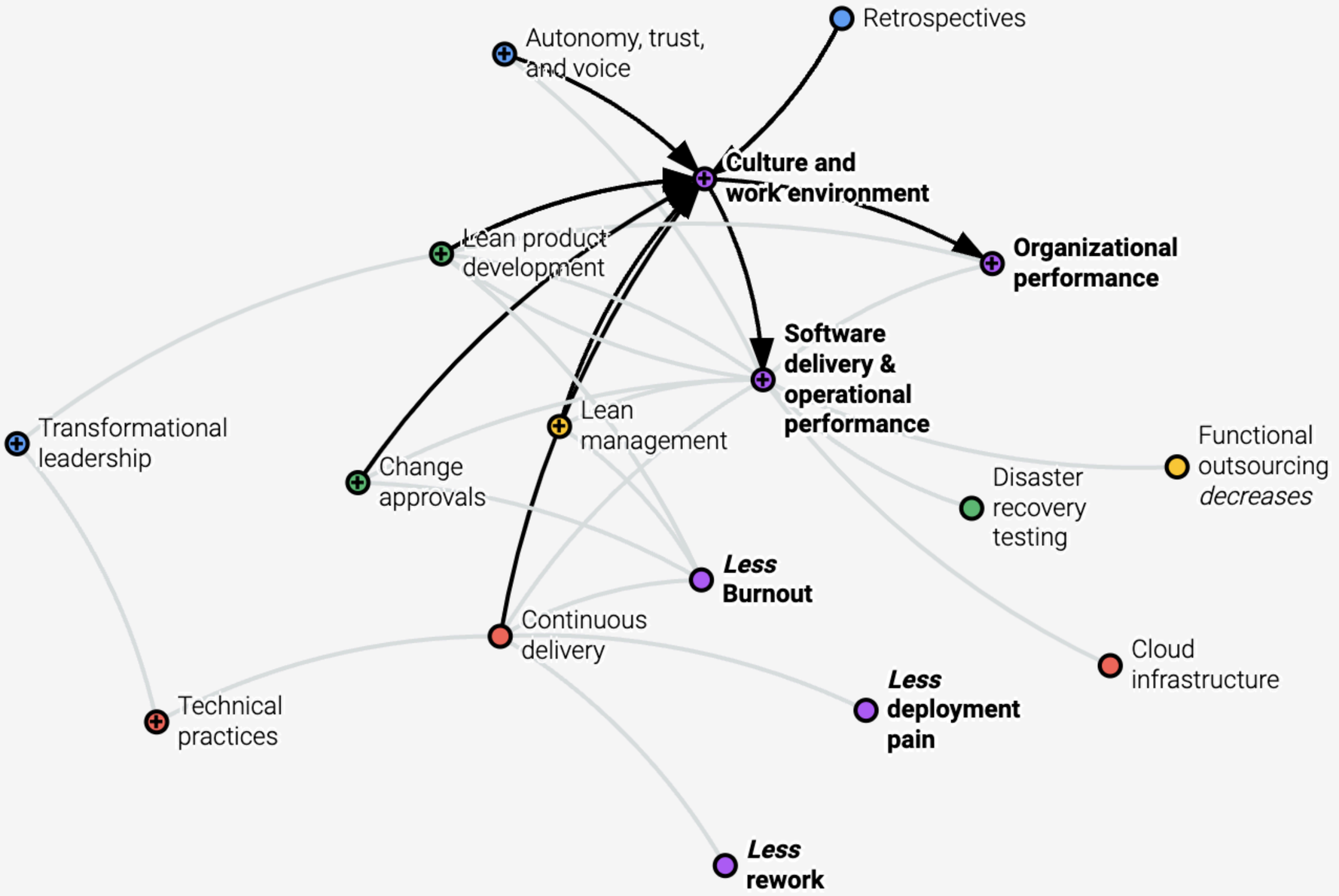
我々は「継続的デリバリを実践すると、組織文化に好影響を与えることができる」との仮説を立てた。そして分析の結果、それが真であることを立証できた

『LeanとDevOpsの科学』 p.57

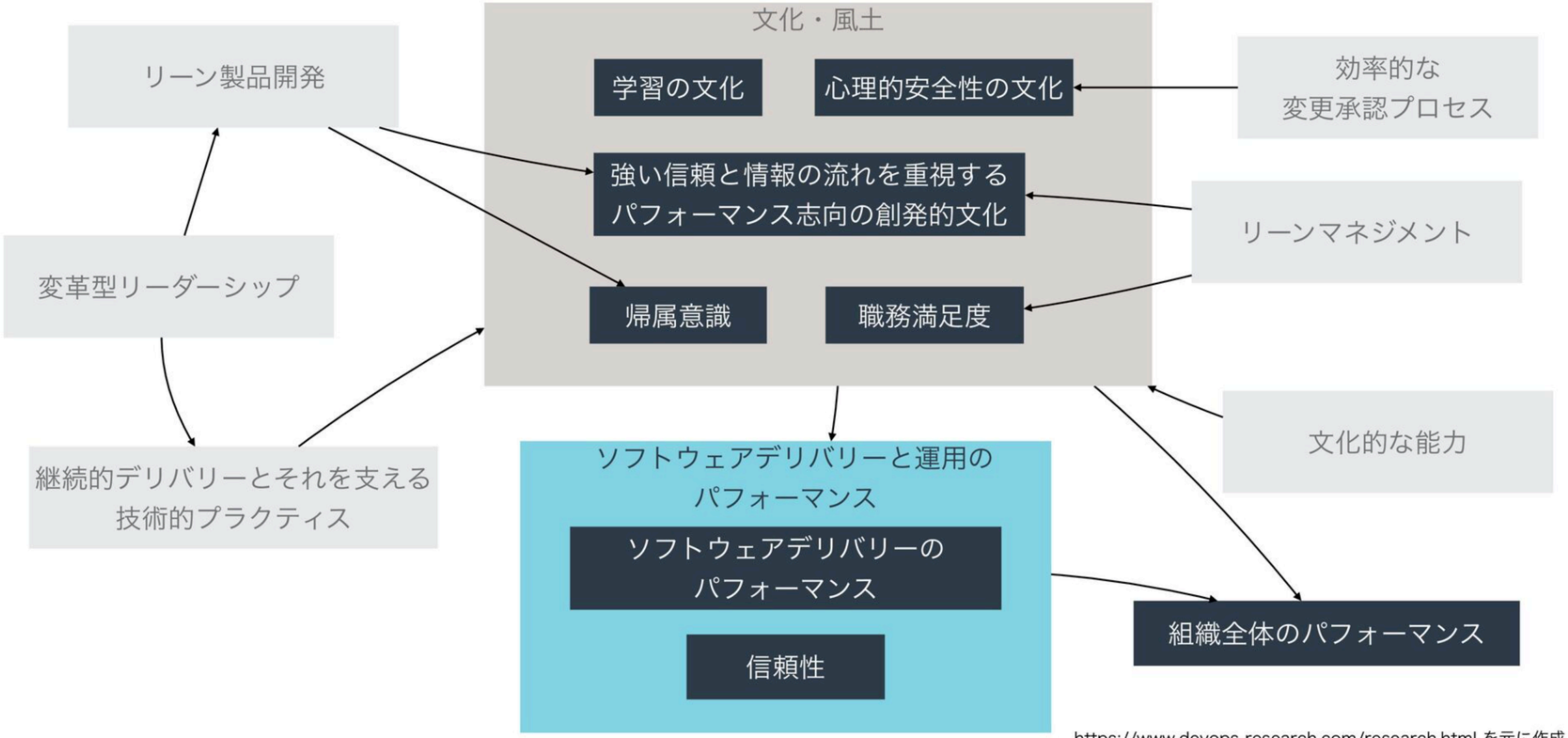


https://www.amazon.co.jp/dp/4295004901/

文化に影響を及ぼす要素

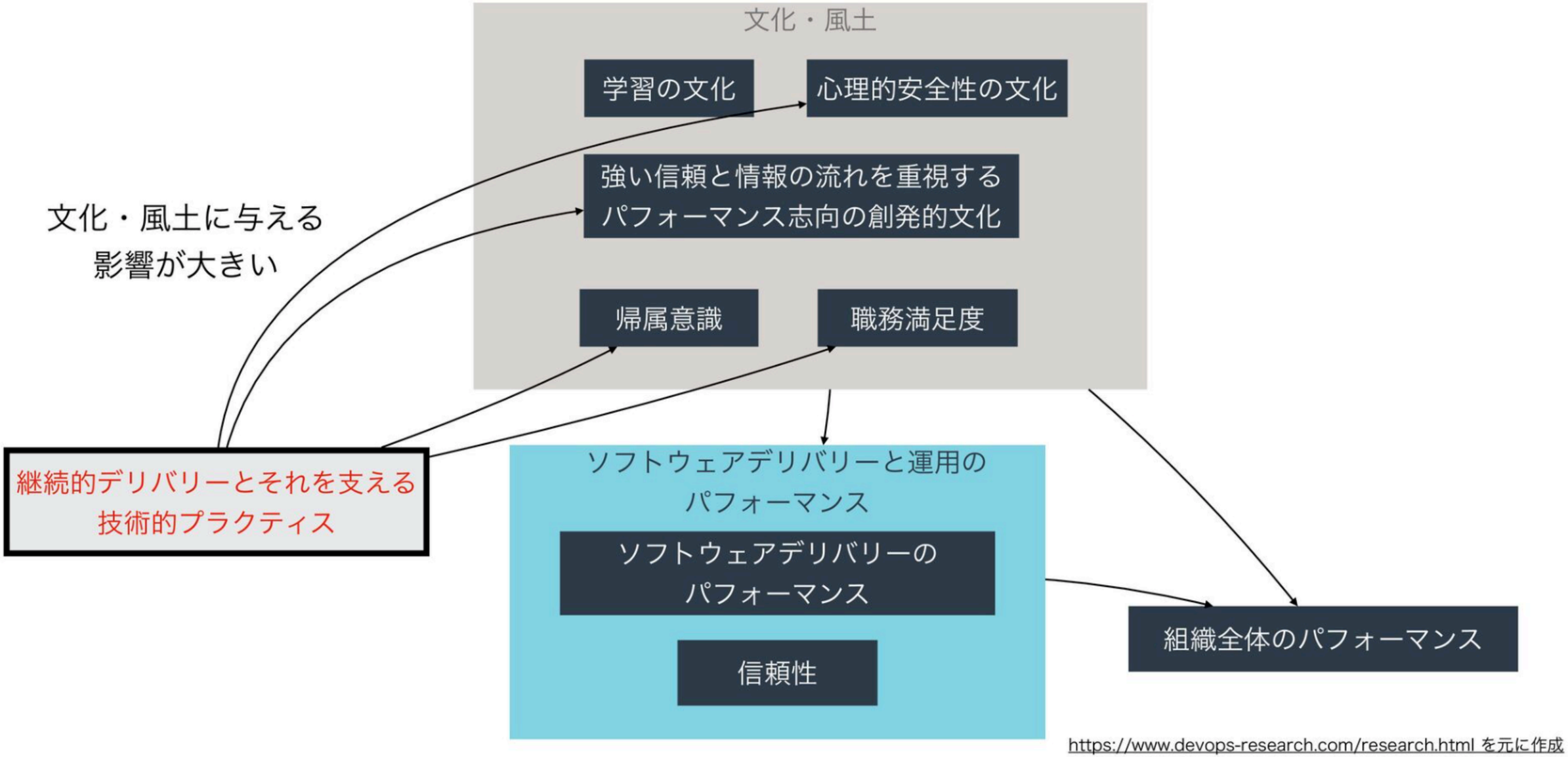


文化・風土およびパフォーマンスに影響を与えることが認められた要素



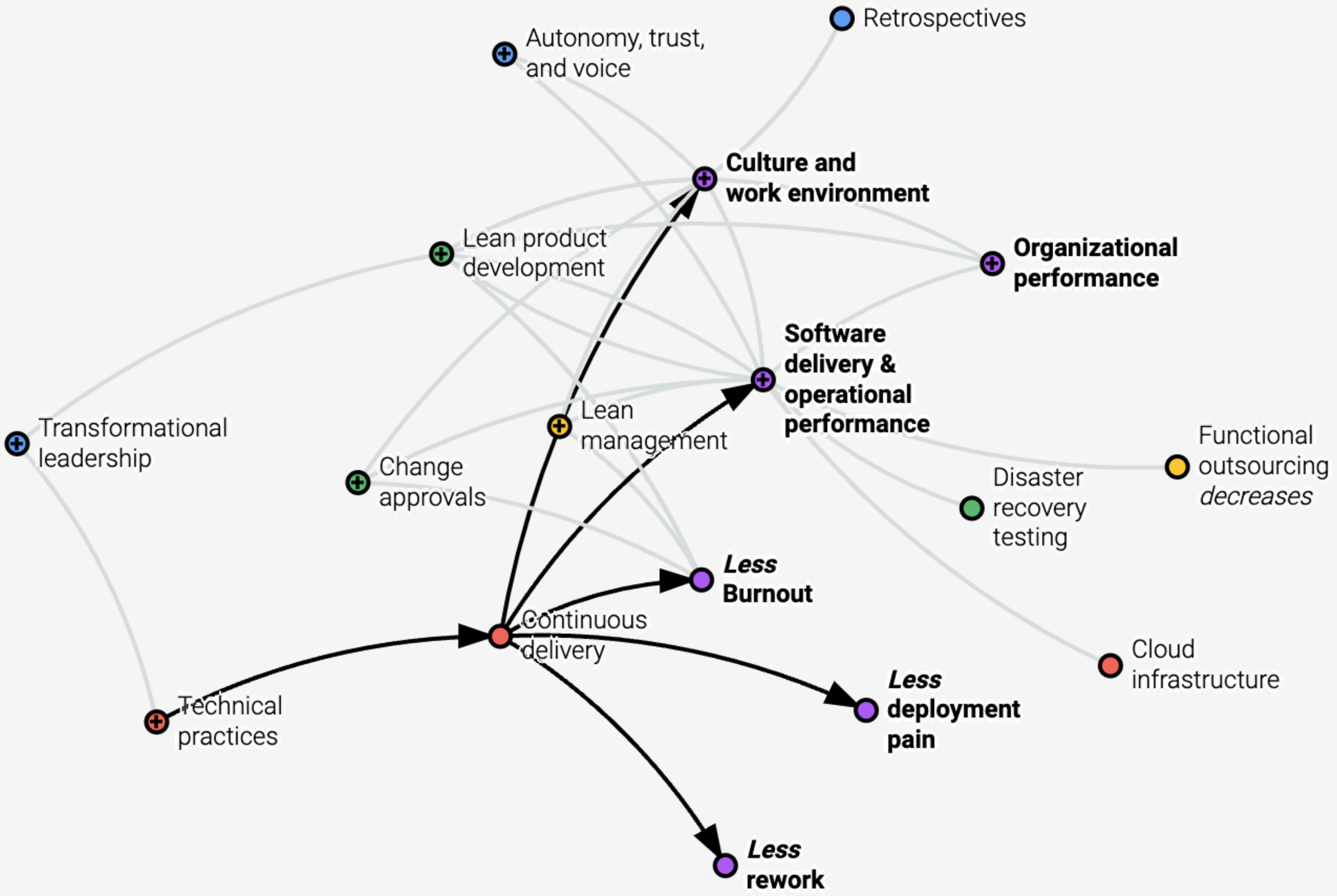
https://www.devops-research.com/research.html を元に作成

ハイパフォーマンスな組織を実現するための文化を育てる



https://www.devops-research.com/research.html を元に作成

継続的デリバリを支える技術プラクティスが文化につながる



継続的デリバリを支える技術プラクティスが文化につながる

テストの自動化

デプロイの自動化

バージョン管理

継続的インテグレーション

継続的テスト

コードのメンテナンス性

疎結合のアーキテクチャ

テストデータ管理

データベース変更管理

包括的なモニタリングと可観測性の実現

プロアクティブな通知

チームによるツール選定

セキュリティのレフトシフト

トランクベースの開発

https://speakerdeck.com/snoozer05/ready-player-one-yunikonqi-ye-falsehimitu-nixue-berukoto