組織にテストを書く文化を根付かせる戦略と戦術

（2020秋バージョン）

Sep 28, 2020 @ JaSST'20 Niigata 基調講演

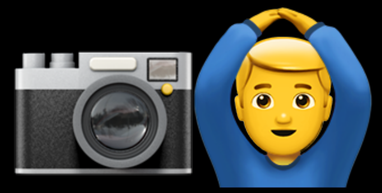
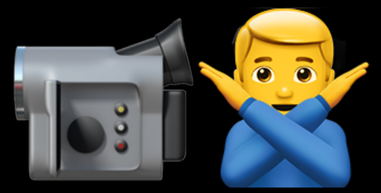
和田 卓人 (@t_wada)

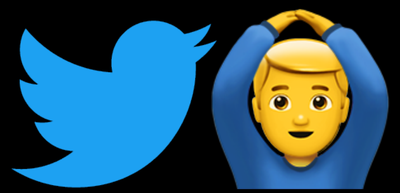
自己紹介

t-wada

t_wada

twada



#jasst
#jasstniigata

技術書の出版に関わっています

よろしくお願いします

テスト書いてないとかお前それ
@t_wadaの前でも
同じこと言えんの?

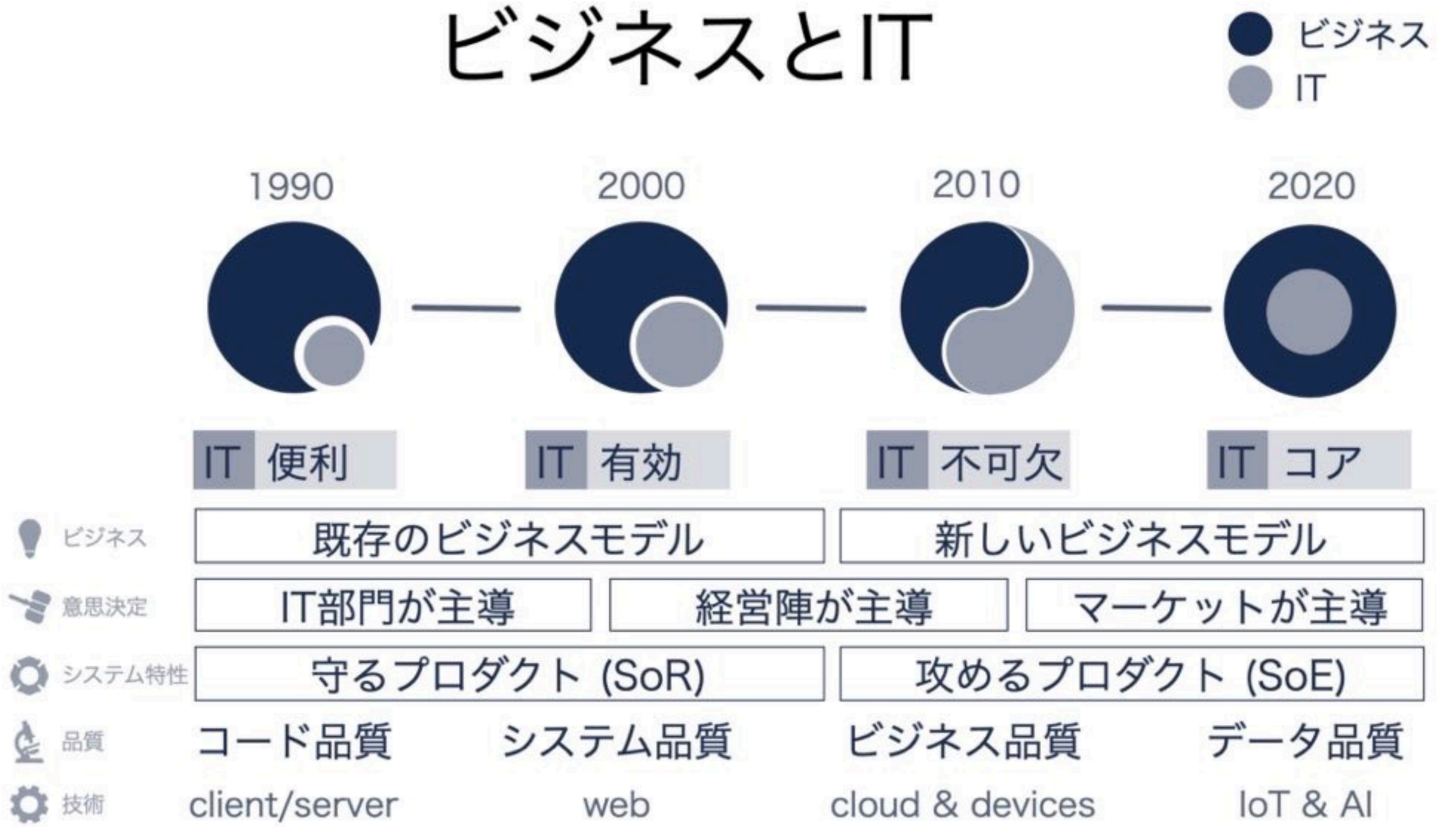
📷🤦‍♂️ 📹🤷‍♂️

🐦🤦‍♂️ **#jasst**
#jasstniigata

組織にテストを書く
文化を根付かせる

戦略編

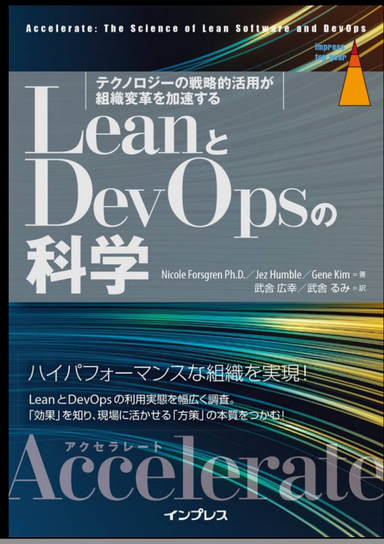
前提: ITは事業のコアになった



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

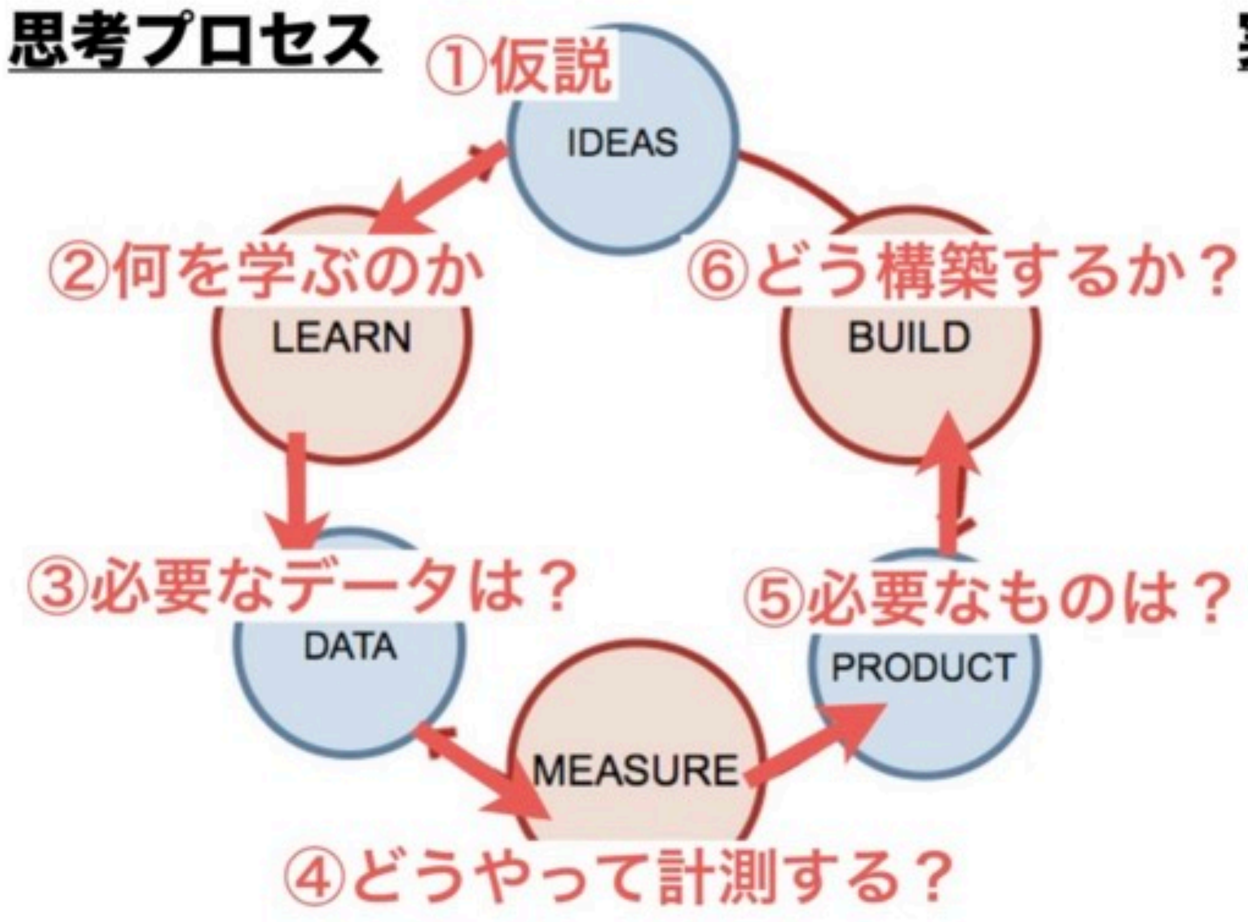
変わる世界: 4つのキーメトリクス

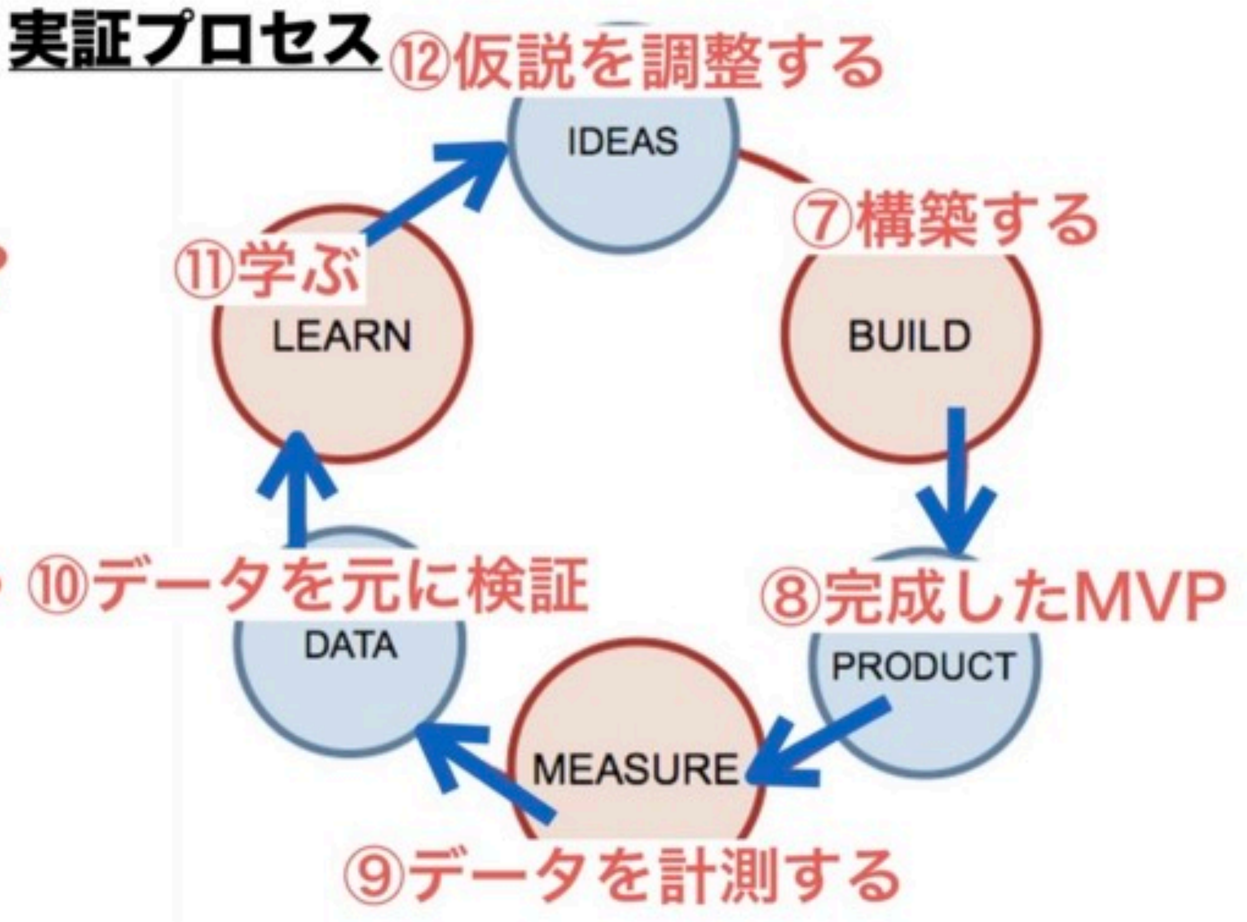
- リードタイム
- デプロイ頻度
- MTTR(平均修復時間)
- 変更失敗率



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

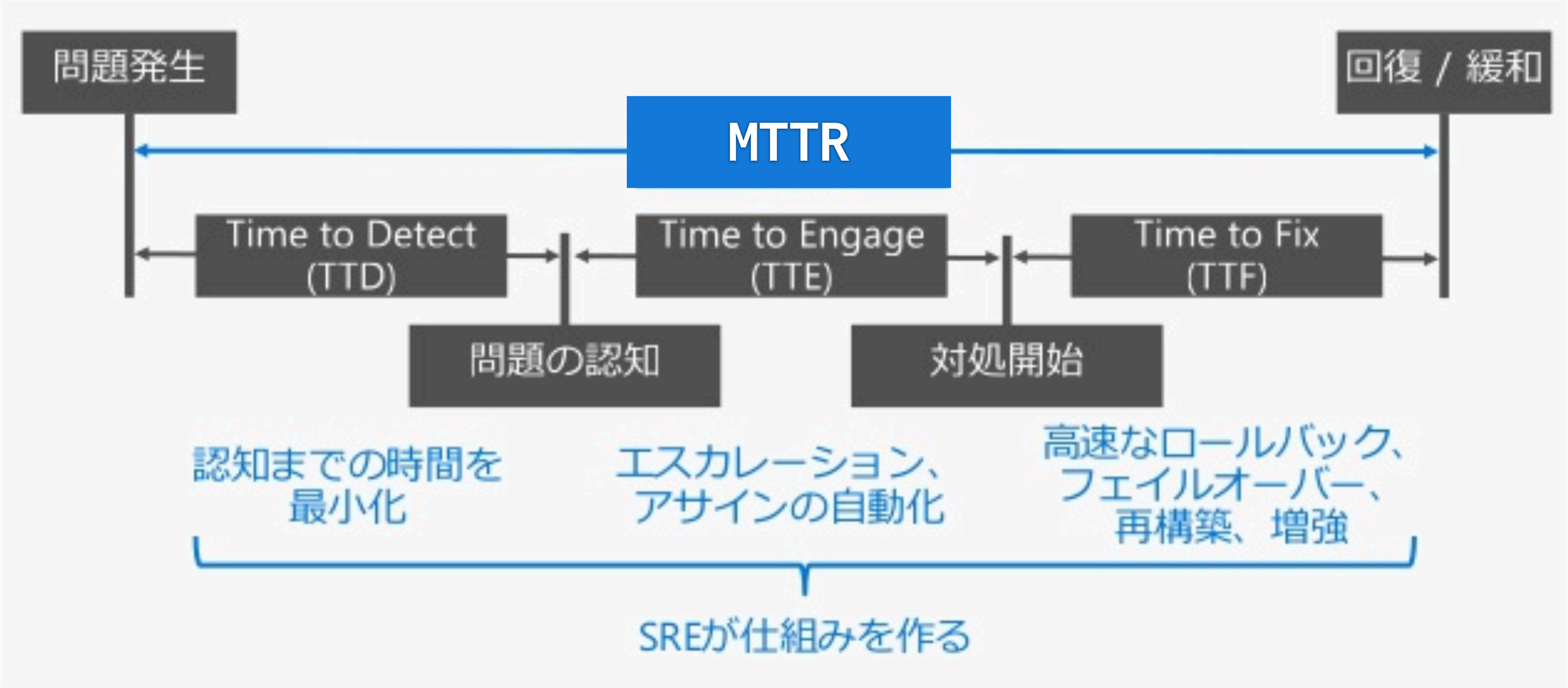
リードタイムとデプロイ頻度: 仮説検証プロセスを迅速に回す





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

MTTR: Mean time to Recovery/Repair

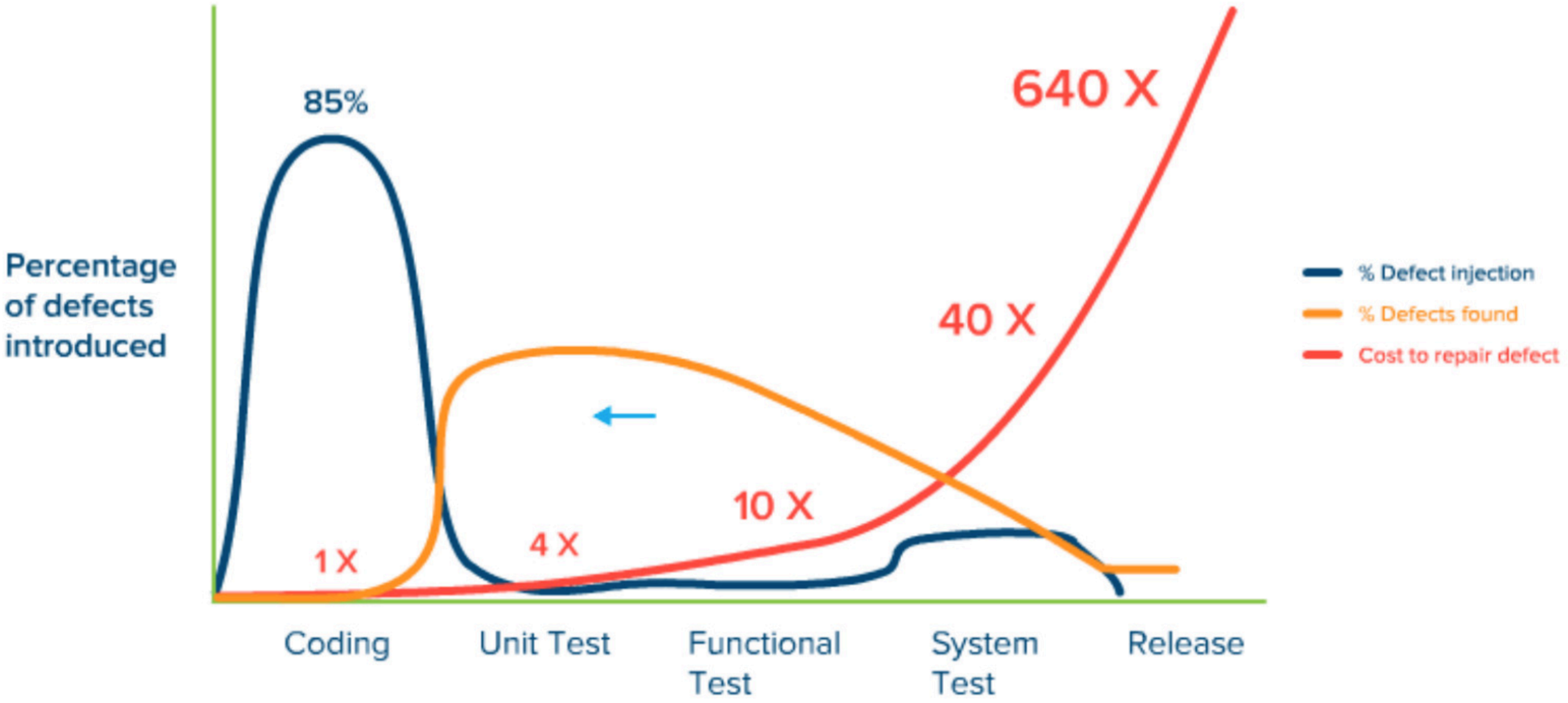


https://www.slideshare.net/decode2017/do05-sre-site-reliability-engineering-in-azure-on-azure (一部改変)

MTBFを捨てるのではなく品質をプロセスで作り込む

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

4つのキーメトリクス: 2019年の調査

	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6ヵ月
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	1ヵ月から6ヵ月
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://www.slideshare.net/DevOpsWebinars/2019-accelerate-state-of-devops-survey-results-are-in

2019年におけるエリートとローパフォーマーの差

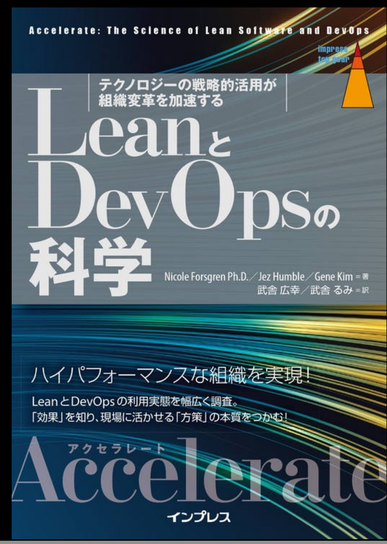
- リードタイム: 106倍
- デプロイ頻度: 208倍
- MTTR: 2604倍
- 変更失敗率: 7倍

https://www.slideshare.net/DevOpsWebinars/2019-accelerate-state-of-devops-survey-results-are-in

4つのキーメトリクス: 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/

なぜテスト自動化なのか

 コストを削減したいから

 ITが事業のコアであり、アジリティが競争力だから

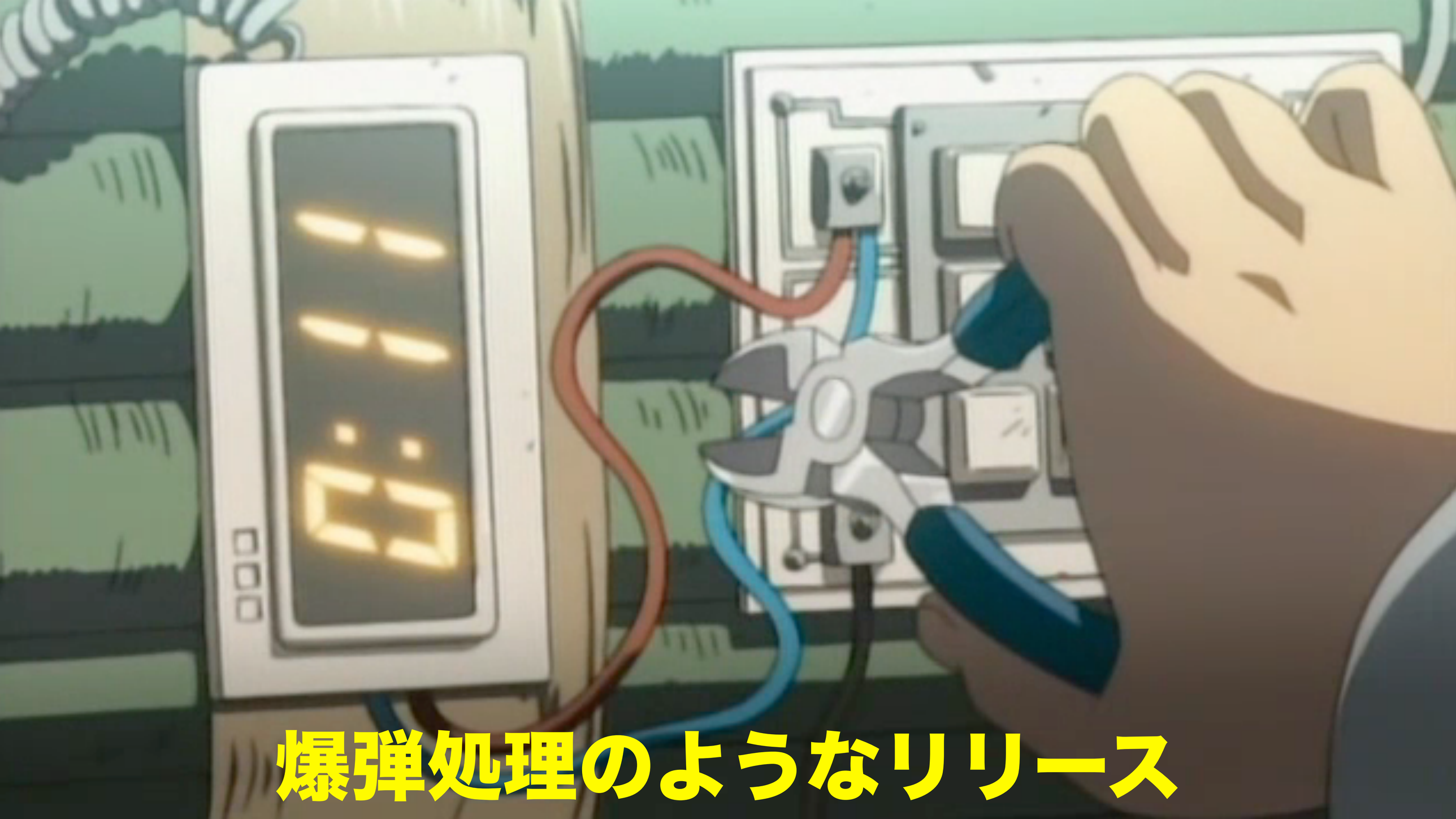
でも……

現場からの中継です

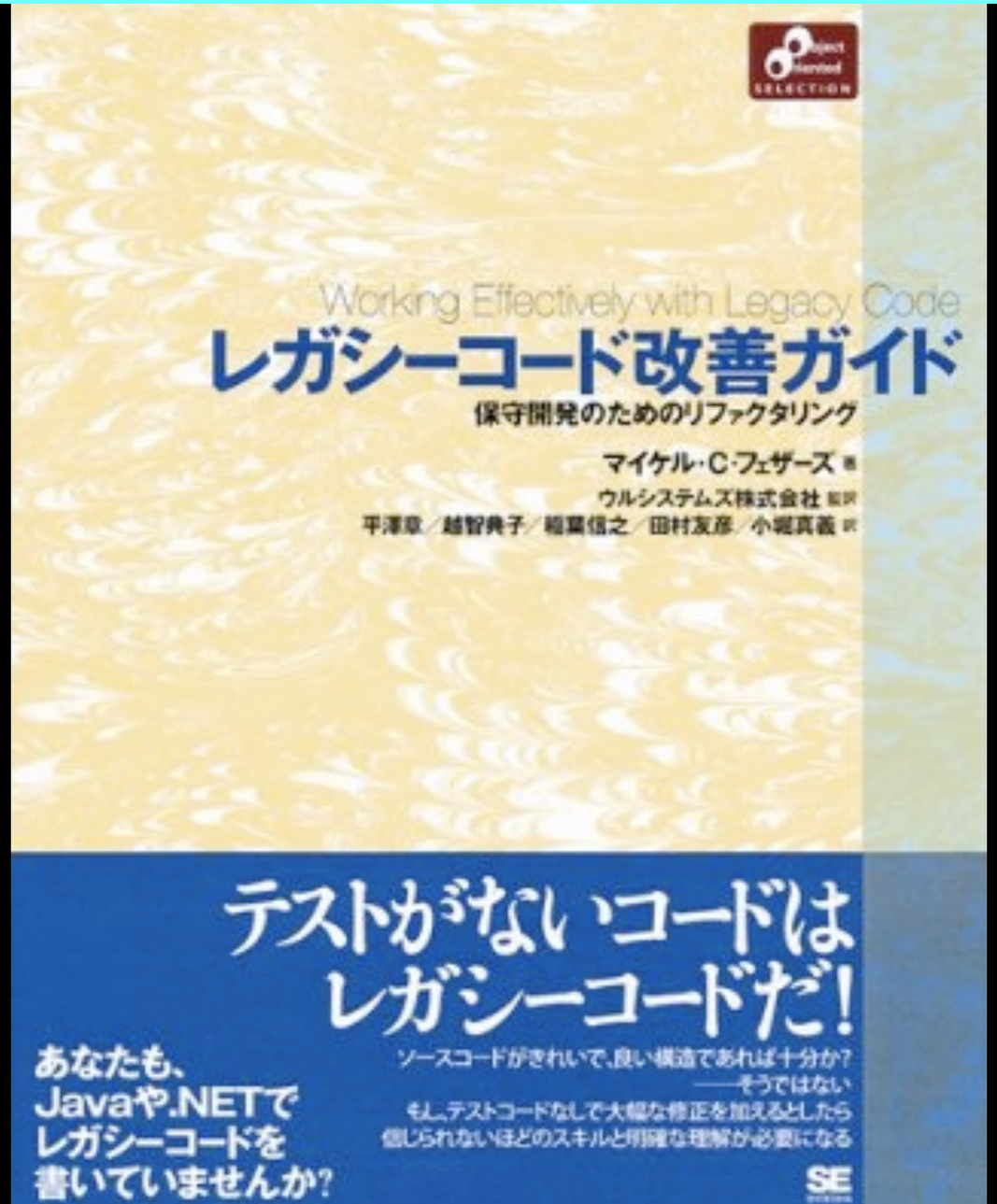
疲弊しきった現場

荒みきったコード

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


爆弾処理のようなリリース

“テストがないコードはレガシーコードだ！”

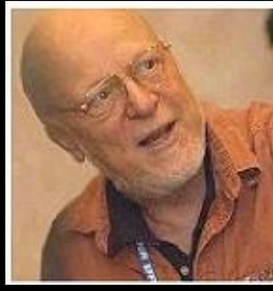


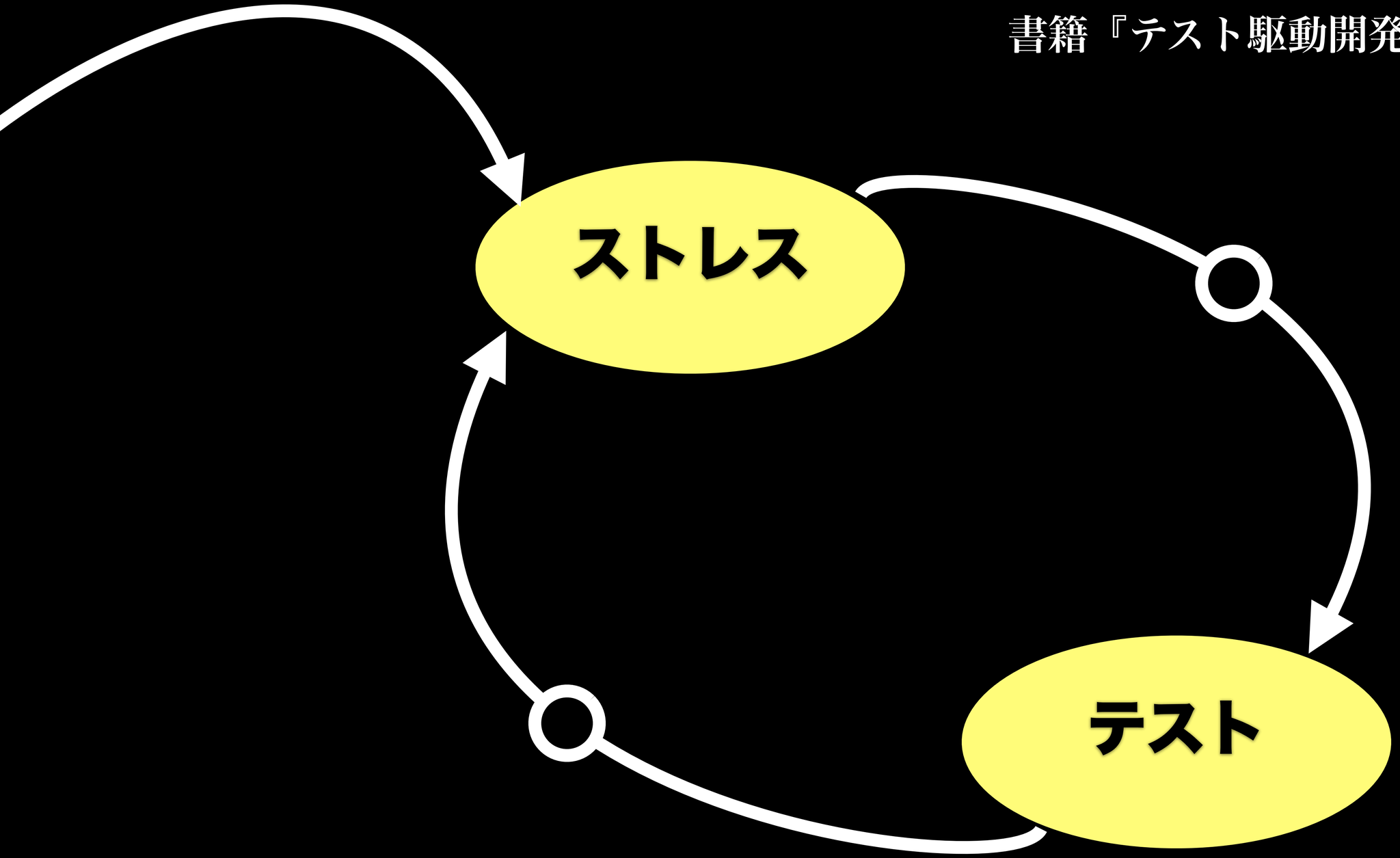
2つの “ならわし”

“テストを書く時間はない”

https://www.flickr.com/photos/dkalo/2909921582

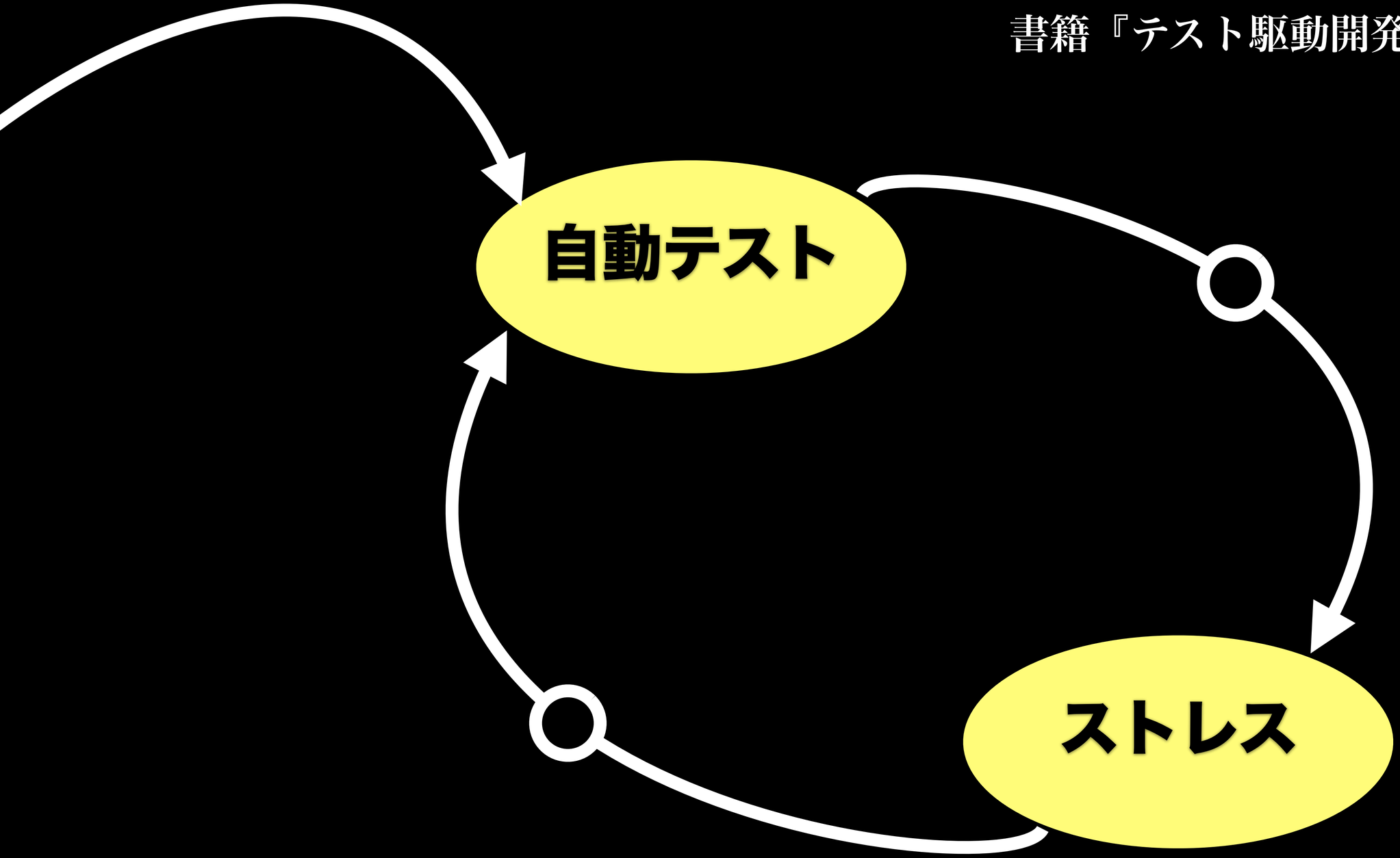
書籍『テスト駆動開発』より





白丸付き矢印: 負の接続 = 根元が増えれば先は減る。根元が減れば先は増える

書籍『テスト駆動開発』より



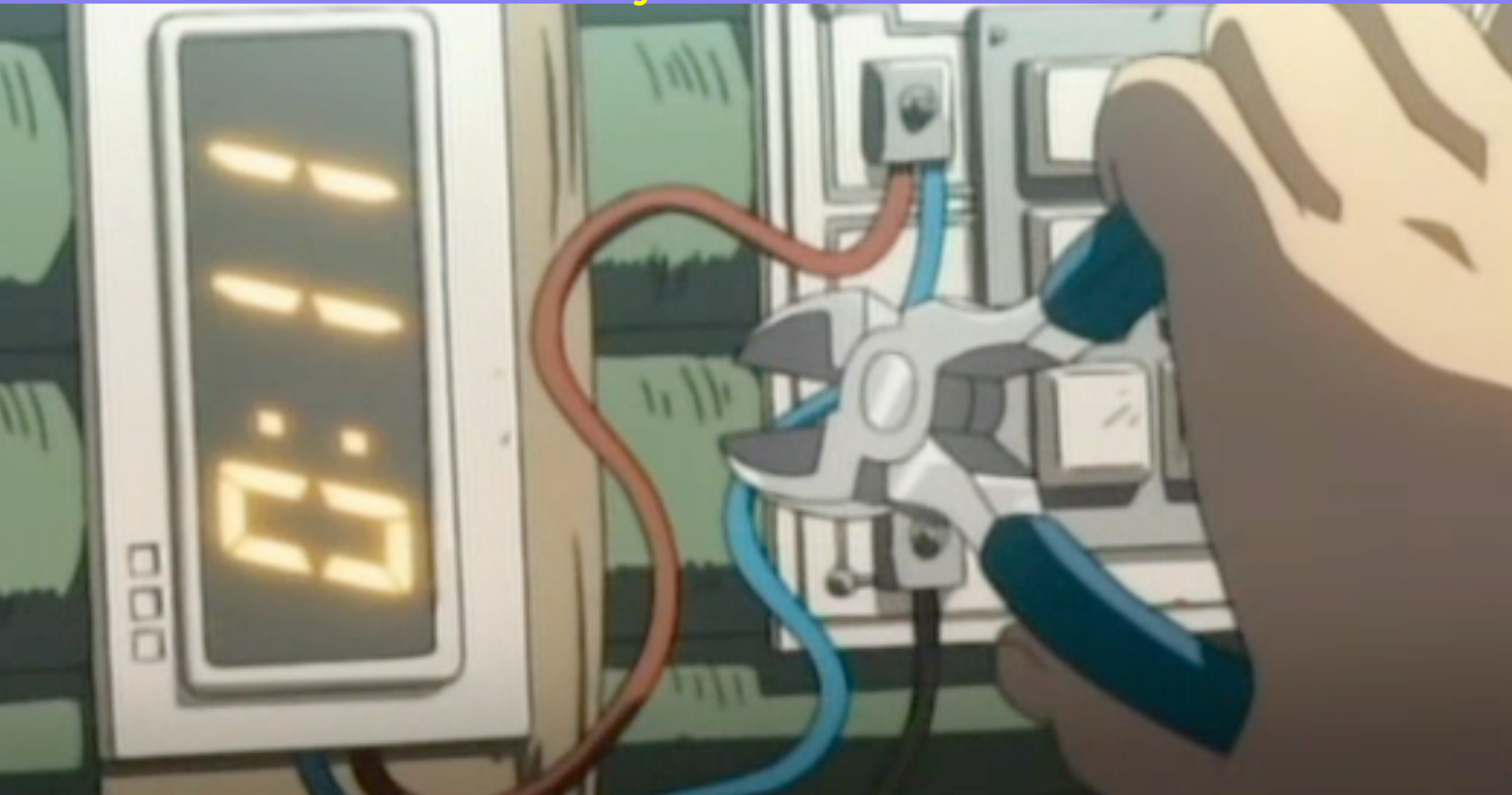
白丸付き矢印: 負の接続 = 根元が増えれば先は減る。根元が減れば先は増える

"テストを書く時間がないのではなく、
テストを書かないから時間がなくなる
のです"

“動くコードに触れるな”

https://www.flickr.com/photos/59145750@N03/5559004171

Edit & Pray には死が待っている



何もしてないから壊れる

Takuto Wada
@t_wada

フォローする

「何もしてないのに壊れた」が問題になる時代から、「何もしてないから壊れる」時代へ。わかる。

> **songmu** @songmu
> 「何もしてないから壊れる」というのが当たり前になりつつあるな

18:37 - 2019年8月5日

84件のリツイート　**225**件のいいね

84　225

https://twitter.com/t_wada/status/1158552706761408515

Cover & Modify

文化を変えていく

- 文化の醸成は年単位の事業になる
- 「テストを書く時間がない」のではなく「テストを書かないから時間がなくなる」
- 「動くコードに触れるな」と戦う。触れなければ競争力が弱まり、事業は緩やかに死んでいく
- コスト削減ではなくアジリティを得るための自動化

銀の弾丸はない

- レガシーコード改善に正解はない
- テスト自動化は銀の弾丸ではない
- 導入方法にも銀の弾丸はない
- 導入を目的にしてはならない
- 状況は現場によって全て異なる

イマココからはじめる

- ToBe ではなく AsIs と NotToBe からはじめる
- 人は自分の速度でしか成長できない
- プロジェクトもプロジェクトの速度でしか成長できない
- それでも座して待たず変化を加速しなければならない

動きにくい組織を
動かすには

日本人を動かす言葉（用法用量を守って使うこと）

- アメリカ人:　飛び込めばあなたは英雄です
- イギリス人:　飛び込めばあなたは紳士です
- ドイツ人:　　飛び込むのがこの船の規則です
- イタリア人:　飛び込むと女性にもてますよ
- フランス人:　飛び込まないでください
- 日本人:　　みんな飛び込んでいますよ

https://www.amazon.co.jp/dp/4121502027

【独自】河野行革相がハンコ使用廃止を要求 「できない場合は今月中に理由を」

23日 17時57分

河野行政改革担当大臣が行政上の手続きではハンコの使用を原則廃止するよう求め、できない場合はその理由を今月中に示すよう、各府省庁に伝えていたことがＪＮＮの取材でわかりました。

２３日に行われた「デジタル庁」創設に関する関係閣僚会議では、河野大臣から行政の手続きに伴い必要とされるハンコについて、速やかに廃止したいとする考えが示されました。さらに河野大臣は各府省庁に対して、行政上の手続きでハンコの使用見直しを速やかに行うこと、ハンコを必要とする手続きおよそ１万１０００件のうち役所が「廃止しない方針」としているものについては、その理由を今月中に示すよう求めていることが関係者への取材で明らかになりました。

説明責任の向きを反転させる

広木 大地/ エンジニアリング組織論への招待
@hiroki_daichi

DXCriteriaでも、「説明責任の向き」を反転させることが大事だと説いています。

「変える理由」を説明するのではなく、「変えない理由」を説明する方に力学を変える。

Translate Tweet

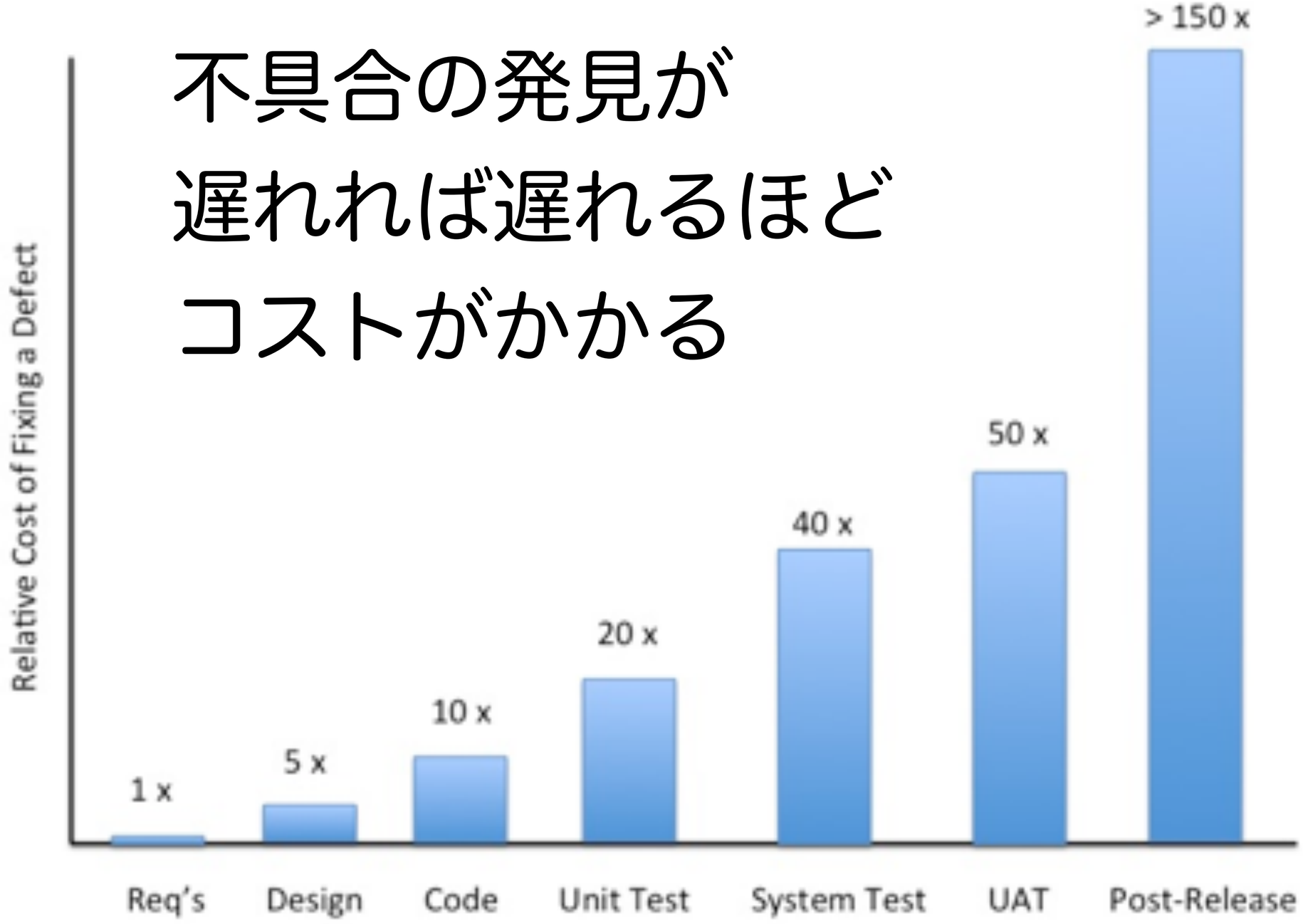
なぜ、変える必要があるのか？	なぜ、変えていかないのか？
DXの実現のためには、今までの常識であったことから「新しい当たり前」を取り入れていく必要があります。 しかし、その外部の基準が存在しないことによって、「なぜそれを取り入れるのか/やってみるのか」の説明をゼロから求められ、結果的に現場は学習性無気力状態になってしまい変革が遅れてしまいます。	本基準で、リストアップされている項目は必ずしもすべてを実現する必要はありません。 しかし、それをやらない場合には、自分たちはなぜやらないのかを説明できる必要があるでしょう。 このように基準を使いながら、「変える理由」から「変えない理由」を確認するような文化形成を育むきっかけになればと考えています。

基準を通じて、「説明責任の向き」を反転させていきたい。

https://twitter.com/hiroki_daichi/status/1308991986687520770

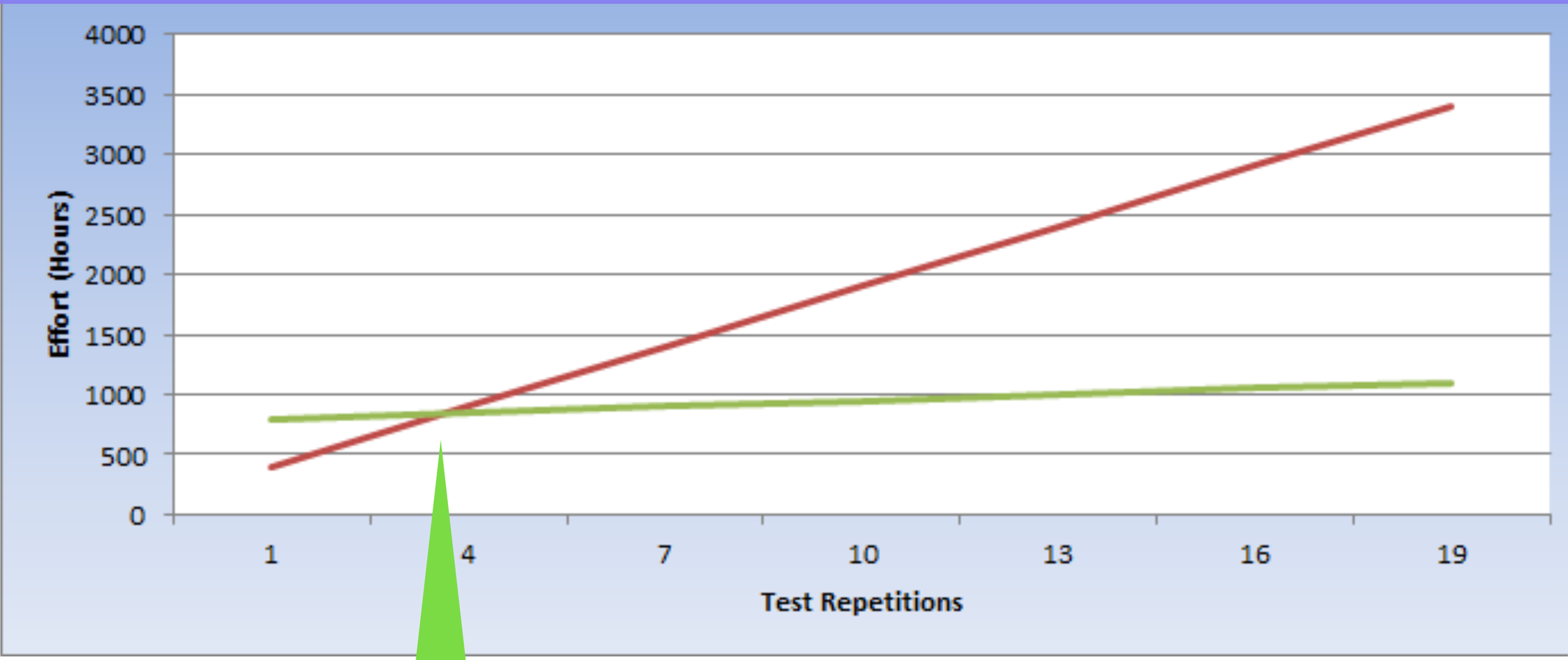
理論武装する

不具合の発見が
遅れれば遅れるほど
コストがかかる

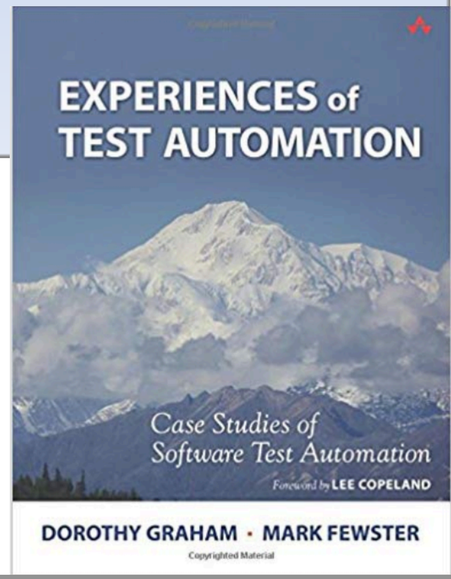


https://www.astqb.org/press-room/ISTQB_Certification_News_2015_1.html

テスト自動化の損益分岐点は「4回」



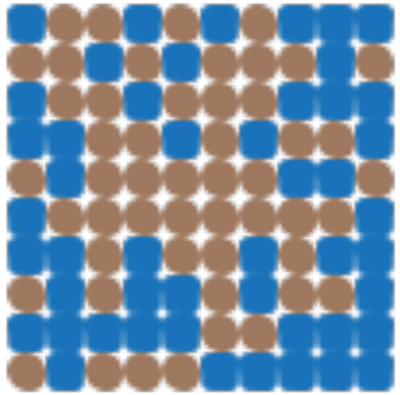
およそ4回で
手動テストと自動化されたテストの
コストが逆転する



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

保守性の低さがもたらすもの

If we compare one system with a lot of cruft...



the cruft means new features take longer to build

this extra time and effort is the cost of the cruft, paid with each new feature

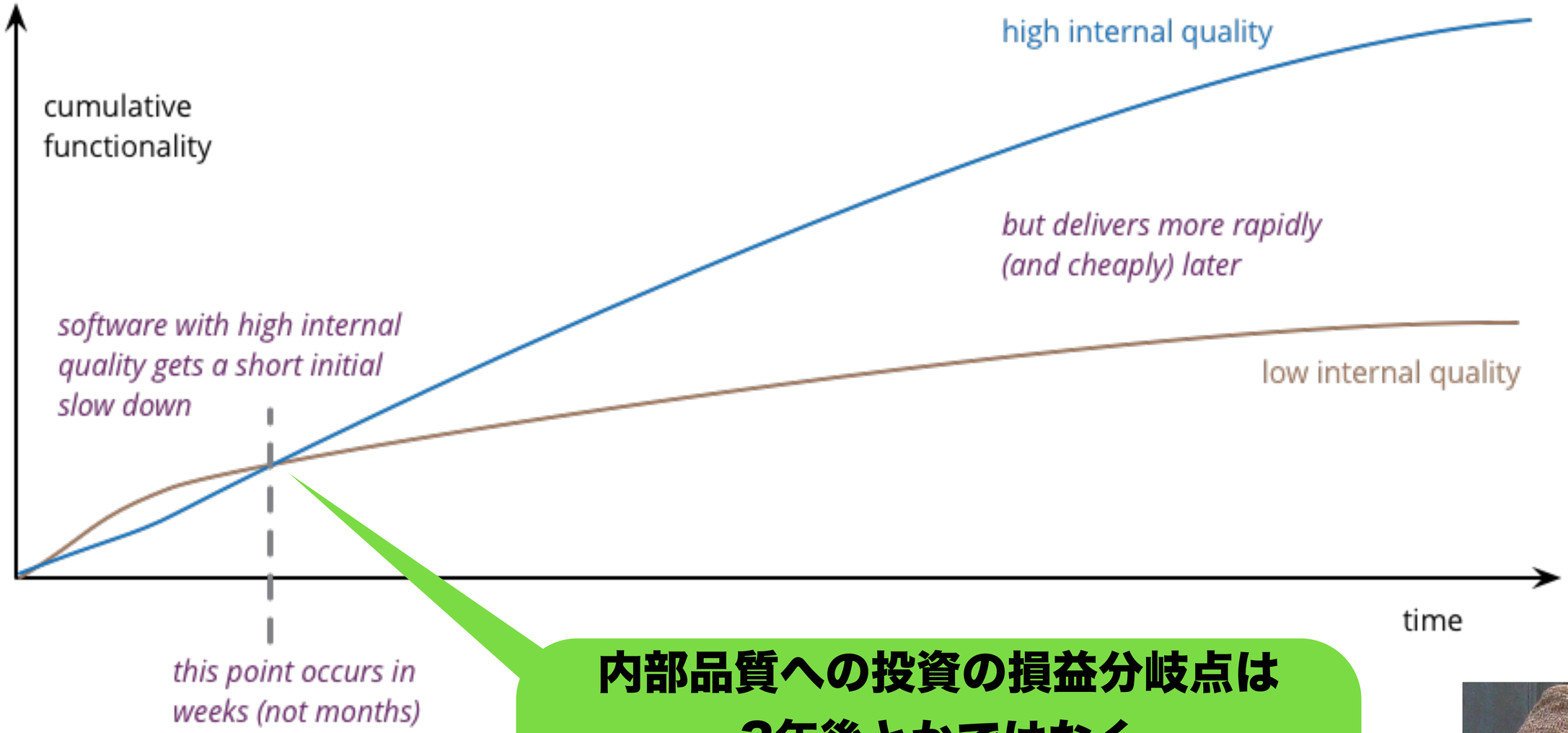
...to an equivalent one without

free of cruft, features can be added more quickly

ひとつひとつの変更に
余計な時間がかかる

https://martinfowler.com/articles/is-quality-worth-cost.html

内部品質への投資の損益分岐点は1ヶ月以内に現れる



内部品質への投資の損益分岐点は
3年後とかではなく
1ヶ月以内に現れる

https://martinfowler.com/articles/is-quality-worth-cost.html

TOWERS QUEST
Computer Software Development

TDD導入効果(MS, IBM)

	IBM Driver	MS Windows	MS MSN	MS Visual Studio
ソースコードサイズ (KLOC)	41	6	26	155.2
テストコードサイズ (KLOC)	28.5	4	23.2	60.3
TDDを採用していない類似プロジェクトでの欠陥密度を１としたときの欠陥密度	0.61	0.38	0.24	0.09
TDD採用により増加したコード実装時間(管理者の見積による)	15～20%	25～35%	15%	20～25%

N. Nagappan, M. E. Maximilien, T. Bhat and L. Williams: Realizing quality improvement through test driven development: results and experiences of four industrial teams, Journal of Empirical Software Engineering, vol. 13, pp. 289-302 (2008)

TOWERS QUEST
Computer Software Development

TDD導入効果(エリクソン他)

- TDDを実施した場合に報告されている知見
  - 機能テストでの不具合検出数が18%削減された
  - コーディング(実装)の時間が16%増えた
  - テストのカバレッジが大きくなった
- 被験者を対象としたアンケート
  - 96%の被験者がデバッグの工数を減らすと感じた
  - 88%の被験者が要求が洗練されると感じた
  - 92%の被験者がコードの品質を上げると感じた
  - 50%の被験者が開発工数を減らすと感じた

Boby George, a and Laurie Williams: A structured experiment of test-driven development, Journal of Information and Software Technology Vol. 46, No. 5, p. 337-342(2004)

テストは品質を上げない

https://www.flickr.com/photos/tompagenet/2271383143

テストは品質を上げない

- 品質が「わかる」ようになる
  - わかることこそ大事
- テストを書くだけでは、良くはならない
  - 体重計に乗るだけでは痩せない
- 品質を上げるのは設計とプログラミング
- 再設計とリファクタリングをテストで支える

“テストでは品質は上がらないですよ。テストはあくまでも品質をあげるきっかけ。品質をあげるのはプログラミングです。これは大昔からそう。”

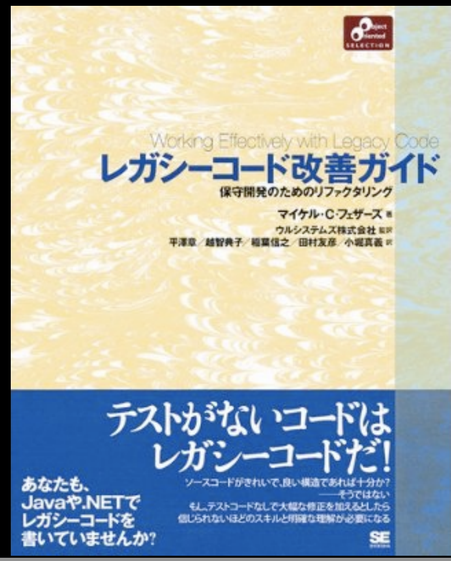
組織にテストを書く文化を根付かせる

戦術編

2つの “道しるべ”

レガシーコード改善ガイド

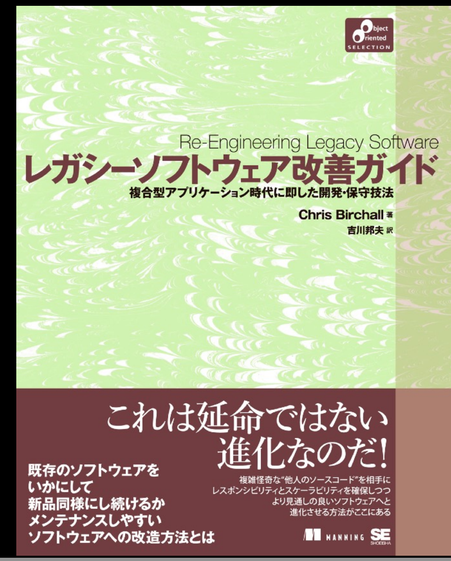
- 「レガシーコードのジレンマ」
  - “コードを変更するためにはテストを整備する必要がある。多くの場合、テストを整備するためには、コードを変更する必要がある”
- レガシーコードに触るための語彙と技法を整理した本
- stackoverflow.com からの被言及数第1位



https://www.amazon.co.jp/dp/4798116831

レガシーソフトウェア改善ガイド

- レガシーコード改善ガイドよりやや抽象度が高い
- ソフトウェアのリエンジニアリングを行う3つの選択肢を示している
  - リファクタリング
  - リアーキテクティング
  - ビッグ・リライト



https://www.amazon.co.jp/dp/4798145149

どこにテストを書いていくか

どこからやるか

- 「何が一番やばいですか？」
- 最も困っているところから
  - お金、個人情報、……
- 新機能開発から
- バグ修正のところから
- 静的解析でピンポイントに

「痛んだ箇所」と「手が届く果実」

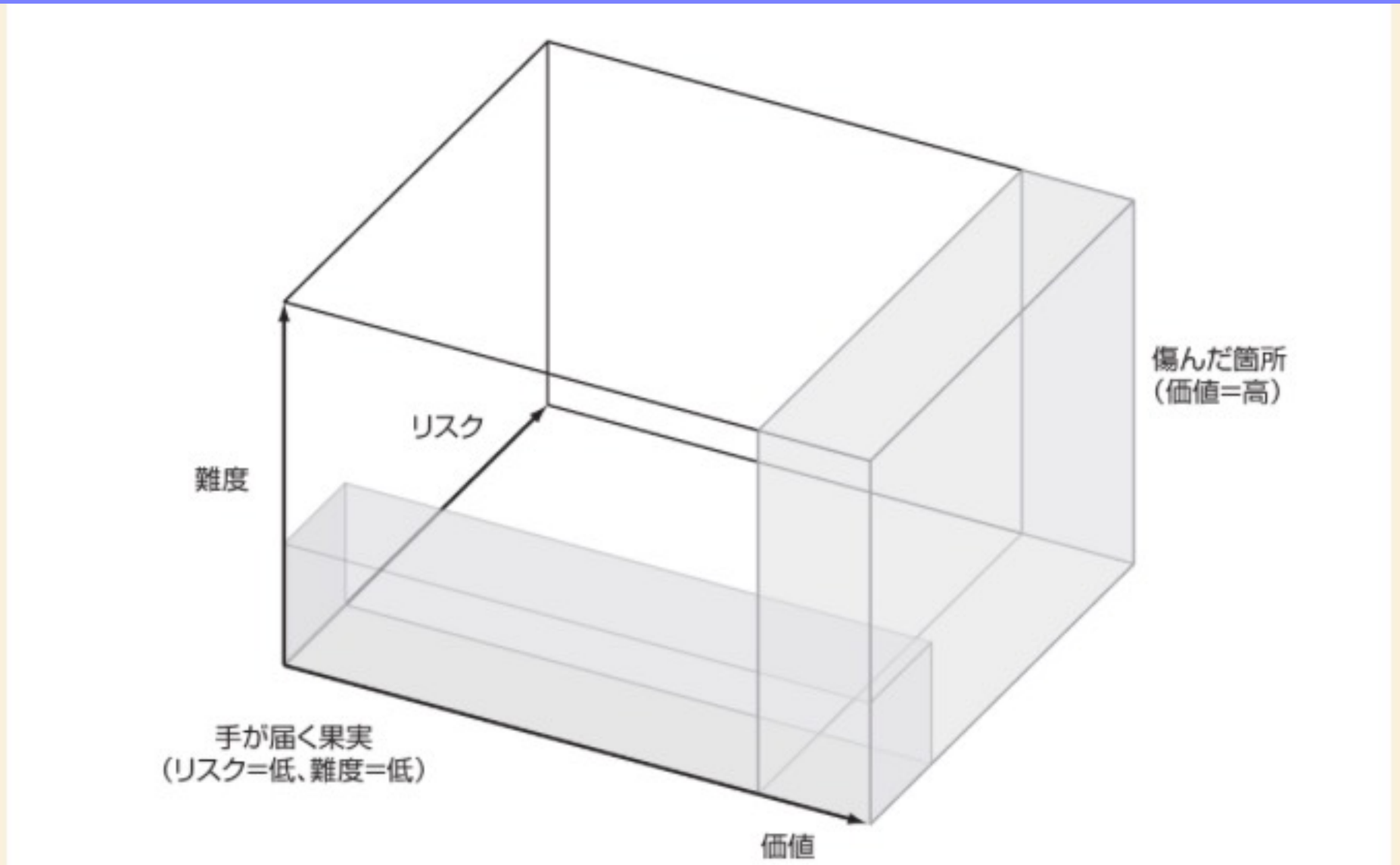
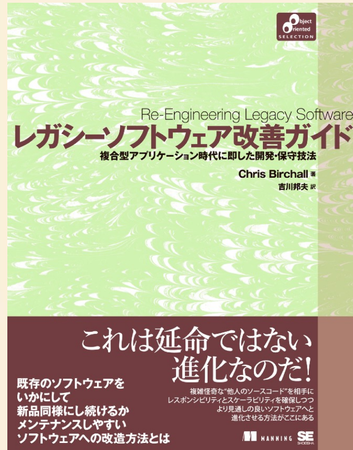


図3-2：「傷んだ箇所」と「手が届く果実」



テストのトリアージ

- リスク
- 手動テストのコスト
- 自動化コスト

https://www.amazon.co.jp/dp/427406932X

テストケースを一覧にまとめる

テストケース
デザイン変更
セキュリティアラート
取引履歴
口座の凍結
新規ユーザ登録
検索結果の並び替え
お金の入金
振り込みの確認

https://www.amazon.co.jp/dp/427406932X

リスクを見積もる（ざっくり高低）

テストケース	リスク
デザイン変更	
セキュリティアラート	
取引履歴	
口座の凍結	
新規ユーザ登録	
検索結果の並び替え	
お金の入金	
振り込みの確認	

https://www.amazon.co.jp/dp/427406932X

手動テストのコストを見積もる（ざっくり高低）

テストケース	リスク	手動テストのコスト
デザイン変更		
セキュリティアラート		
取引履歴		
口座の凍結		
新規ユーザ登録		
検索結果の並び替え		
お金の入金		
振り込みの確認		

https://www.amazon.co.jp/dp/427406932X

自動化コストを見積もる（ざっくり高低）

テストケース	リスク	手動テストのコスト	自動化コスト
デザイン変更			
セキュリティアラート			
取引履歴			
口座の凍結			
新規ユーザ登録			
検索結果の並び替え			
お金の入金			
振り込みの確認			

https://www.amazon.co.jp/dp/427406932X

優先順位を付けて並べ替える

テストケース	リスク	手動テストのコスト	自動化コスト
口座の凍結			
振り込みの確認			
取引履歴			
検索結果の並び替え			
お金の入金			
セキュリティアラート			
新規ユーザ登録			
デザイン変更			

https://www.amazon.co.jp/dp/427406932X

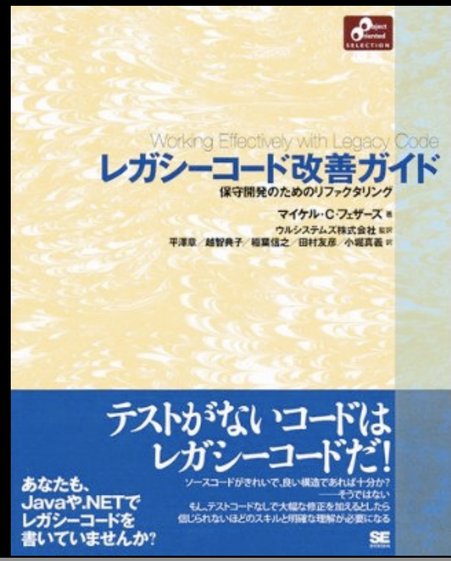
どうテストを書いていくか

自動テストにおけるテスタビリティ

- 実行円滑性（Operability）
  - 自動実行が容易でかつ高速
- 観測容易性（Observability）
  - テスト結果の取得・期待値比較を自動化しやすい
- 制御容易性（Controllability）
  - 事前状態を制御し、テスト対象を自動操作しやすい
- 分解可能性（Decomposability）
  - テスト対象の切り出し、テスト環境への部分置換が容易

レガシーコード改善の技法

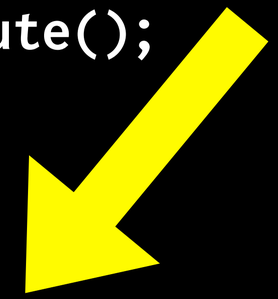
- 接合部の検討と吟味
- 絞り込み点の発見と依存の分離
- 再現性あるテストハーネスの整備
- 超集中編集 (Hyperaware Editing)
- コンパイラ / IDEまかせ
- 仕様化テスト
- 試行リファクタリング



https://www.amazon.co.jp/dp/4798116831

変更前

```
exports.handler = function (event, context, callback) {
  var alexa = Alexa.handler(event, context);
  alexa.registerHandlers(handlers);
  alexa.execute();
};
```



```
var createHandler = function (getNextItemIndex) {
  return function (event, context, callback) {
    var alexa = Alexa.handler(event, context);
    alexa.registerHandlers(createHandlers(getNextItemIndex));
    alexa.execute();
  };
};
exports.createHandler = createHandler;

exports.handler = function () {
  var getNextItemIndex = () => Math.floor(Math.random() * questions.length);
  return createHandler(getNextItemIndex);
}();
```

変更後

例: ランダム性を伴う関数を外から差し込めるようにする

こだわるな

- 最初から全部やろうとしない
- テスト駆動にこだわるな
- テストファーストにこだわるな
- 「ユニット」テストにこだわるな
- テストの実行速度にこだわるな
- テストの網羅性にこだわるな

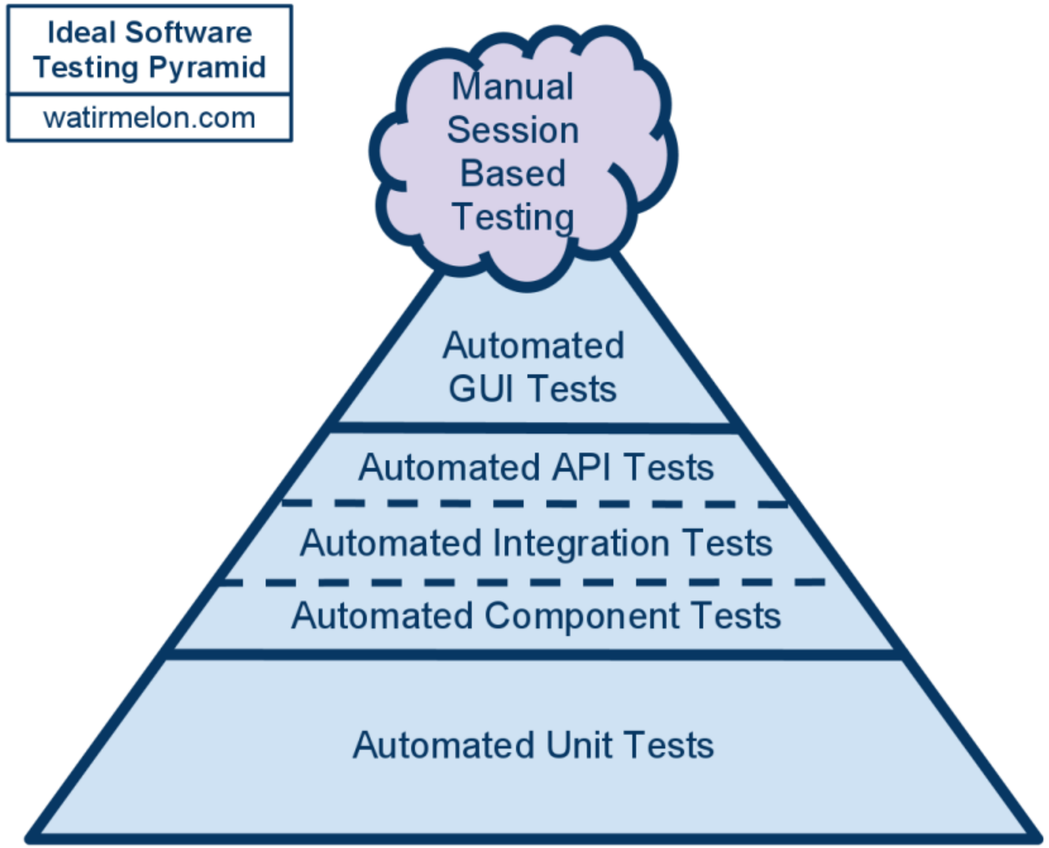
こだわろう

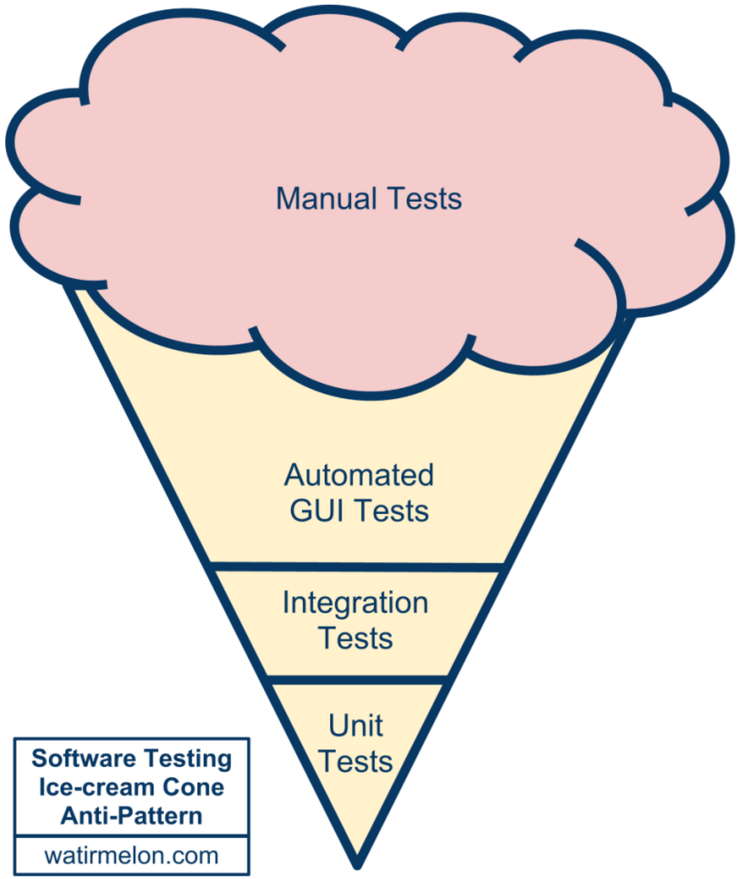
- 良いユニットテストの指標にも優先度がある
- 再現、繰り返し可能 (Repeatable)
- 独立している (Independent)
- 他はそれからでいい

設計の可動域を確保する

- テストがないのは既に設計が悪い兆候
- 設計/実装を変えるのが前提
- 実装のテストを書かないこと
- テストがカバーする範囲に遊びを持たせ、
  カバー範囲内をリファクタリング
- 状況に応じて E2E テストを使いこなす

テスト自動化ピラミッドとアンチパターン





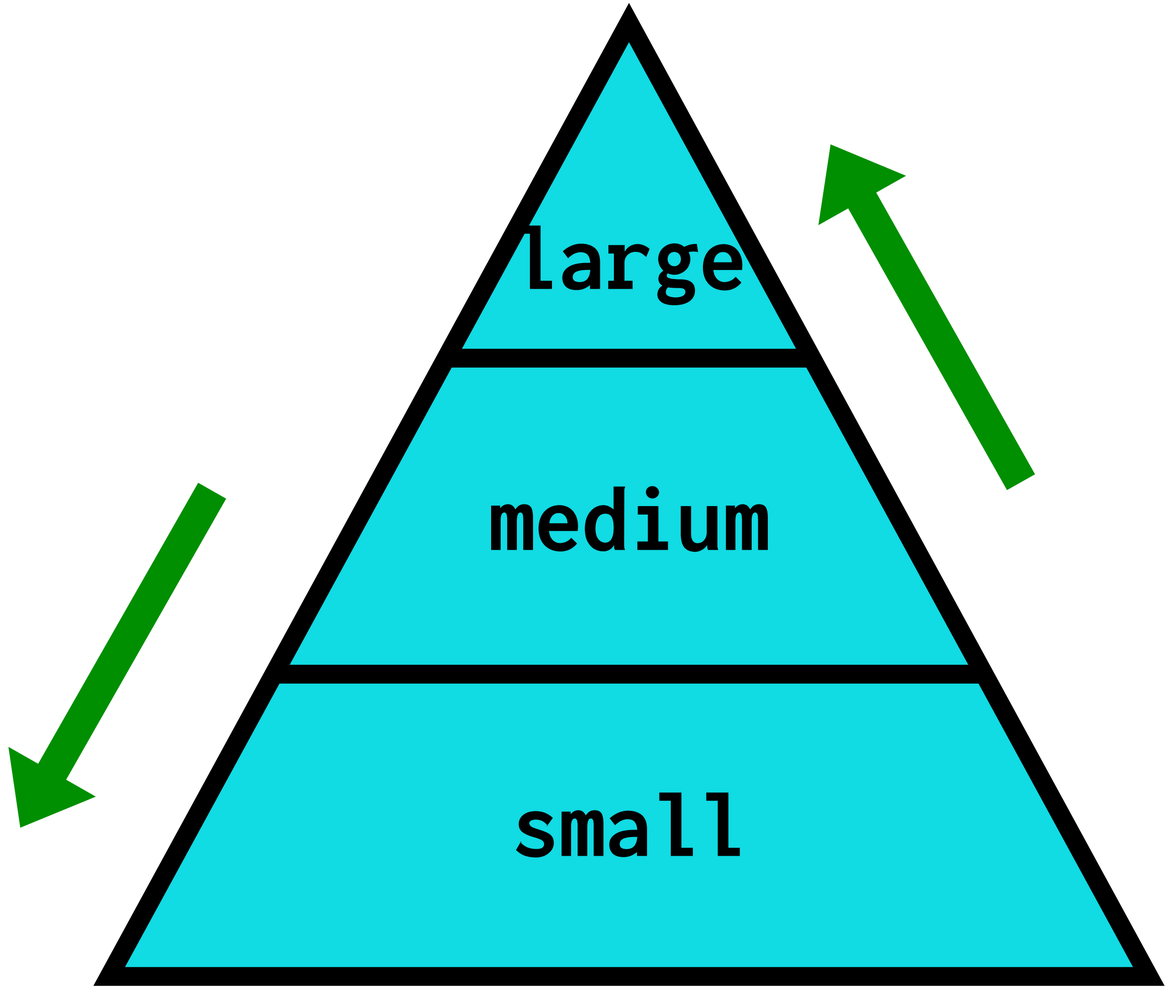
https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

“Test Sizes” at Google

Feature	Small	Medium	Large
Network access	No	localhost only	Yes
Database	No	Yes	Yes
File system access	No	Yes	Yes
Use external systems	No	Discouraged	Yes
Multiple threads	No	Yes	Yes
Sleep statements	No	Yes	Yes
System properties	No	Yes	Yes
Time limit (seconds)	60	300	900+

https://testing.googleblog.com/2010/12/test-sizes.html

自分たちでSMLを定義し、調整する



背中を見せる

- サンプルとデモが大事
- 真似してもらう土台を作る
- 最初はサンプルのコピペでも良い
- テストのある生活を体験してもらうことが大事
- 次にテストのメンテナンスを学ぶ

事例: Cygames 社内ワークショップ

https://tech.cygames.co.jp/archives/3291/

Social Change starts with YOU

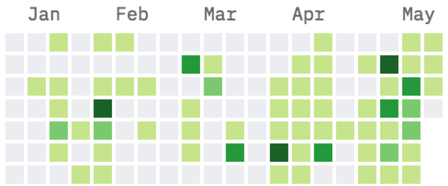
- できるからやるのではない
- やるからできるようになる
- 自分が書けるようにならなければ、誰も書けるようにはならない

Social Change starts with YOU

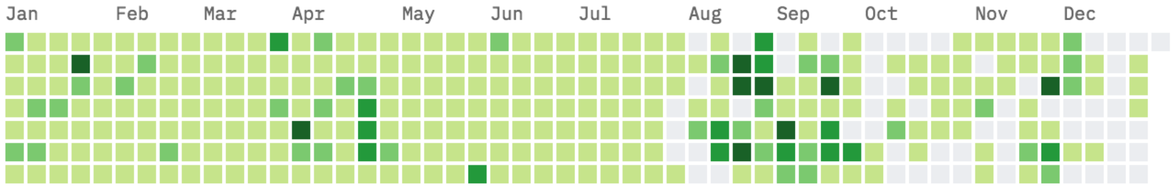
@twada on GitHub

Less More

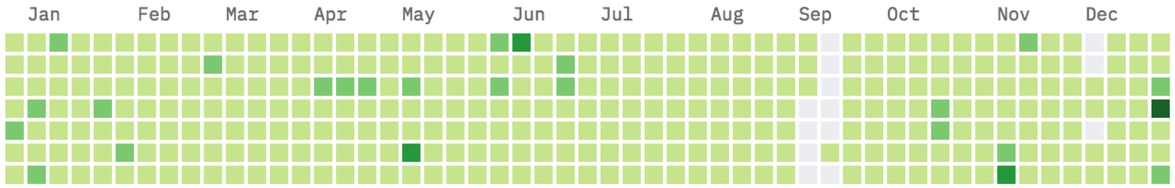
2018: 410 Contributions (so far)



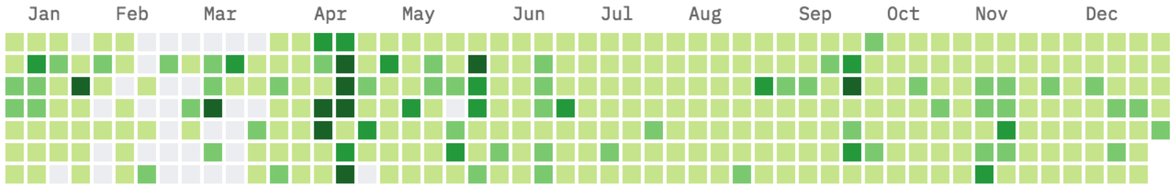
2017: 1,573 Contributions



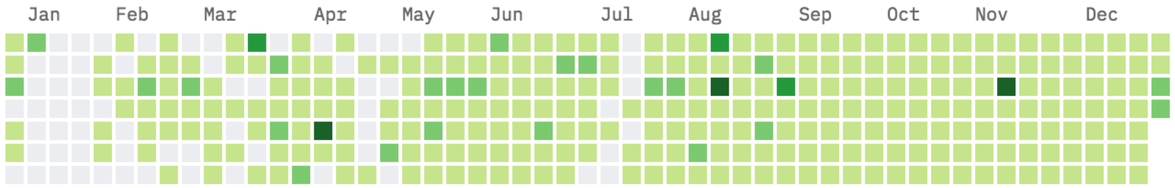
2016: 1,589 Contributions



2015: 2,437 Contributions



2014: 2,582 Contributions



2013: 2,237 Contributions

テストはプロの嗜み

ご清聴ありがとうございました