
変更容易性と理解容易性を支える

# 自動テスト

Feb 10, 2024 @ YAPC::Hiroshima 2024

Takuto WADA

id:t-wada 𝕏 @t_wada @twada @twada



📷🤦 𝕏 🤦

𝕏 #yapc_i #yapcjapan

# 結論から

# 自動テストの目的

信頼性の高い実行結果に
短い時間で到達する状態を保つことで、
開発者に根拠ある自信を与え、
ソフトウェアの成長を持続可能にすること

# Agenda

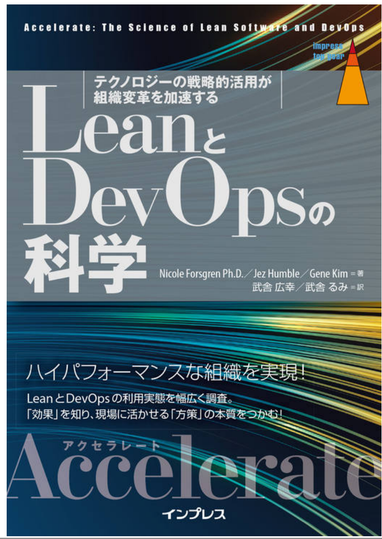
- 信頼性の高い
- 実行結果に
- 短い時間で到達する
- 状態を保つ

# 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）

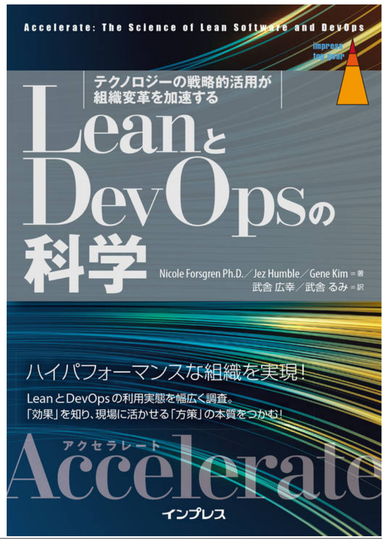



https://www.amazon.co.jp/dp/4295004901/

# 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/

# 成功と失敗

| 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 | | |
| 失敗 | | |

# 成功と失敗

| 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 | デプロイ、マージ | |
| 失敗 | | |

# 成功と失敗

| 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 | | |
| 失敗 | | 問題箇所の特定と修復 |

# 成功と失敗、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 | | |
| 失敗 | 偽陽性 | |

# 成功と失敗、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 | | 偽陰性 |
| 失敗 | | |

# 成功と失敗、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 | デプロイ、マージ | 偽陰性 |
| 失敗 | 偽陽性 | 問題箇所の特定と修復 |

# Agenda

信頼性の高い

🦁実行結果に

短い時間で到達する

状態を保つ

# 情報の役割は意思決定と行動を促すこと

自動テストの実行結果は「情報」であり、情報の役割とは意思決定と行動を促すこと

テストの実行結果が促す行動とは、デプロイ、マージ、コードの修正など

# 成功時のアクション、失敗時のアクション

| プロダクトコードが / テスト結果が | 正しい | 誤っている |
| --- | --- | --- |
| 成功 | デプロイ、マージ | |
| 失敗 | | 問題箇所の特定と修復 |

# テストの出力と狙い

Demo

- テスト結果の出力と狙い
  - シグナルとして
  - 欠陥の絞り込みとして
  - ドキュメントとして
  - データとして
- 成功時の情報量をコントロールする技術
  - テスト名と構造
  - reporter
- 失敗時の情報量をコントロールする技術
  - テスト名と構造
  - テストサイズ（後述）
  - assertion, expectation, matcher

# Agenda

信頼性の高い

実行結果に

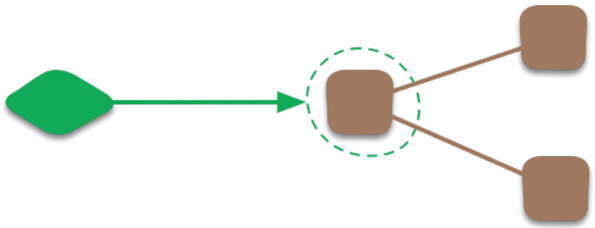
🦁 短い時間で到達する

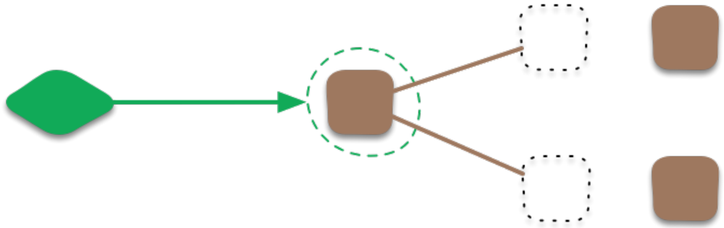
状態を保つ

# テストサイズ:
# 自動テストと CI にフィットする
# 明確なテスト分類基準

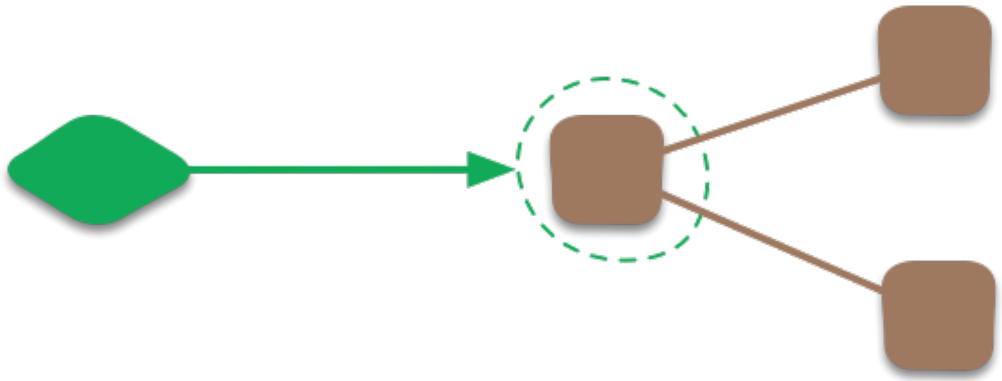
# 皆さんにお伺いします

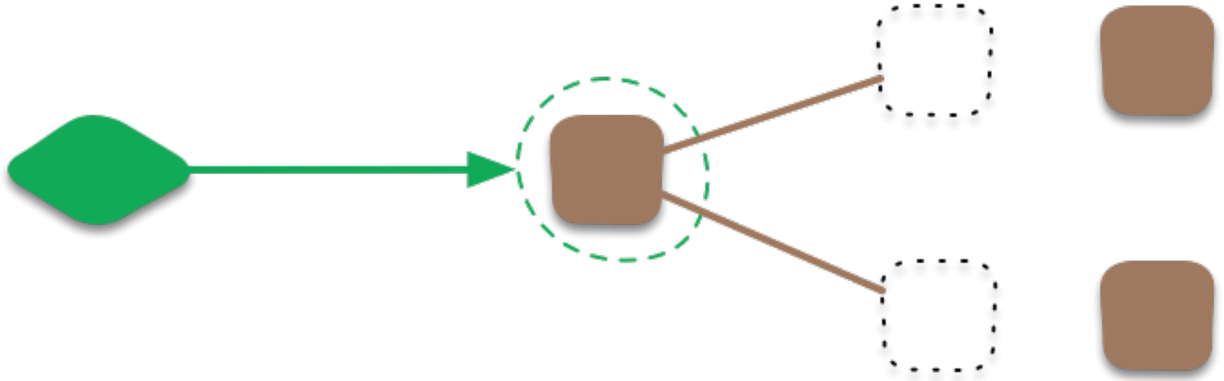
データベースにアクセスするのはユニットテスト? Yes / No
ネットワークにアクセスするのはユニットテスト? Yes / No
ファイルにアクセスするのはユニットテスト? Yes / No
現在時刻にアクセスするのはユニットテスト? Yes / No
依存先のモジュールに本物を使うのはユニットテスト? Yes / No





# Unit Test の Unit って何？

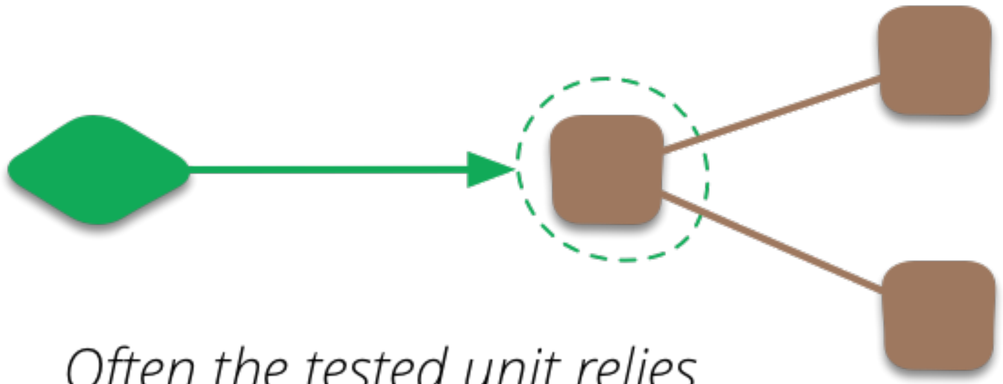




https://martinfowler.com/bliki/UnitTest.html

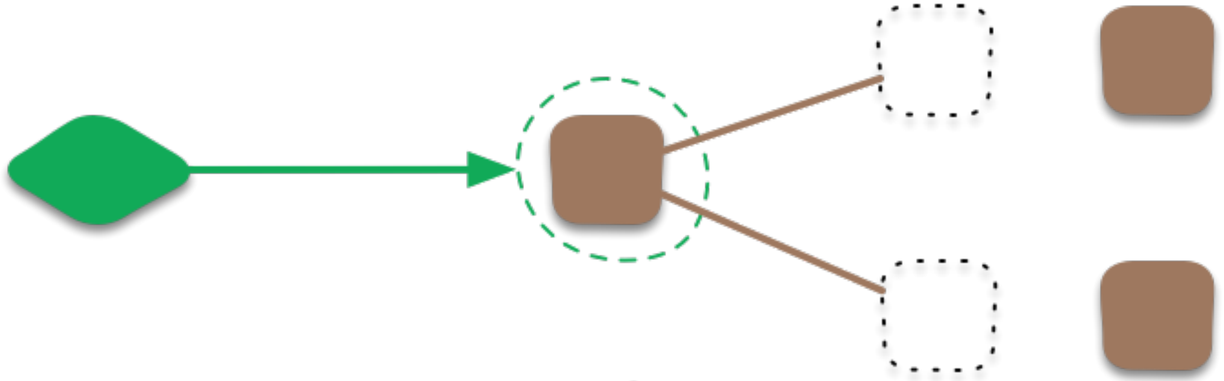
# Unit Test の Unit って何？

**Sociable Tests**



*Often the tested unit relies on other units to fulfill its behavior*

**Solitary Tests**



*Some unit testers prefer to isolate the tested unit*

https://martinfowler.com/bliki/UnitTest.html

# Test Size: より曖昧さの少ない分類

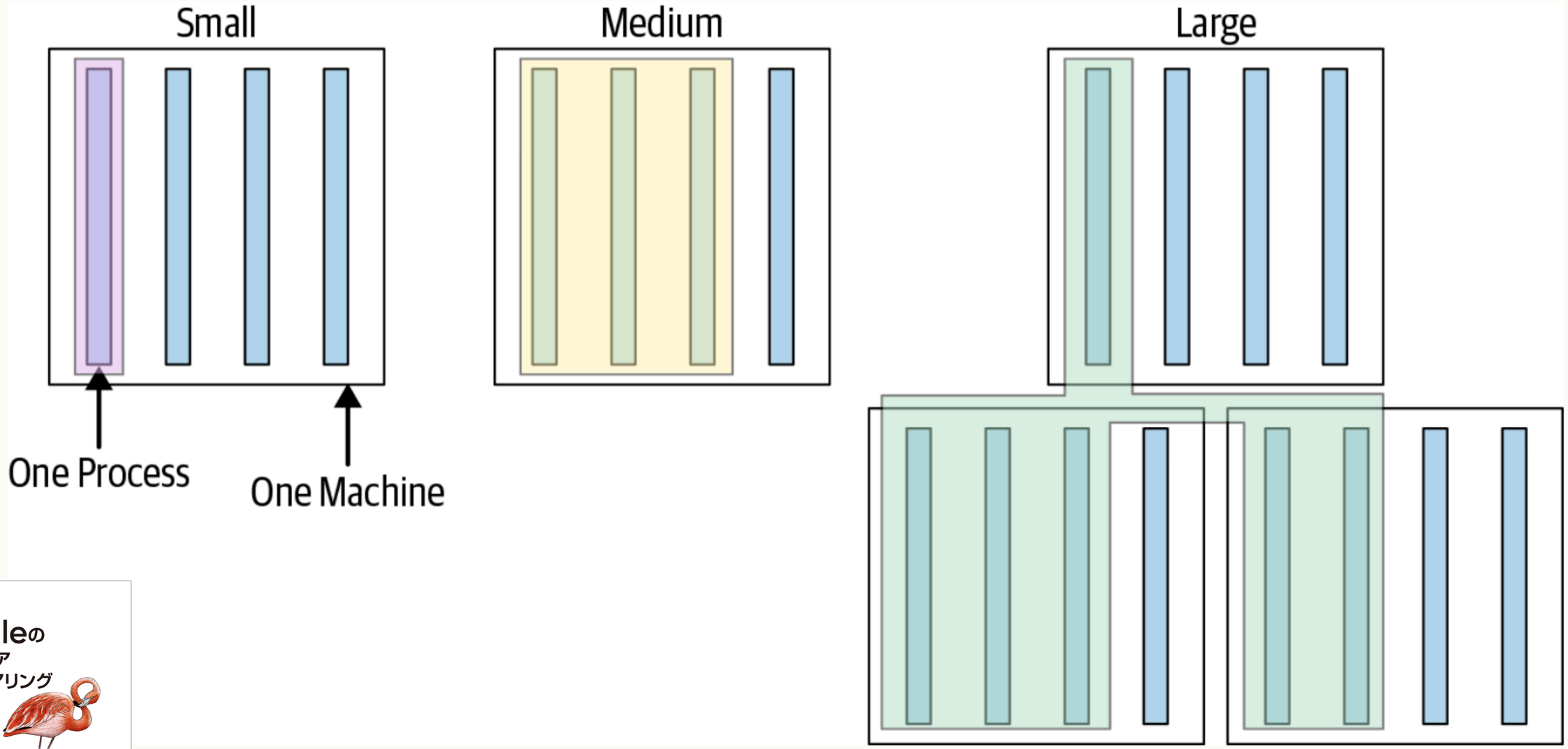




Figure 11-2. Test sizes

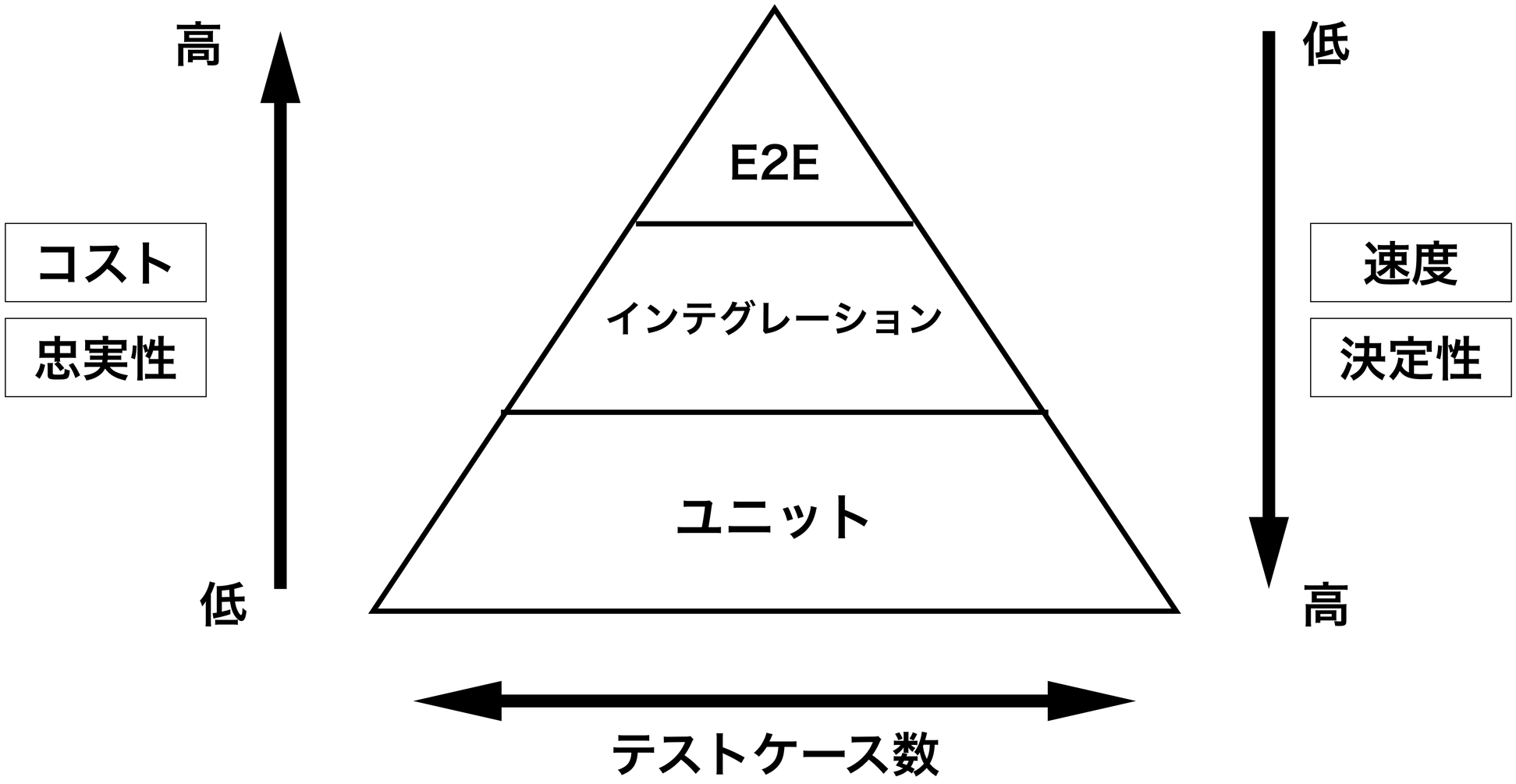
# 例: Google の Android 開発チームにおける Test Size

| 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+ |

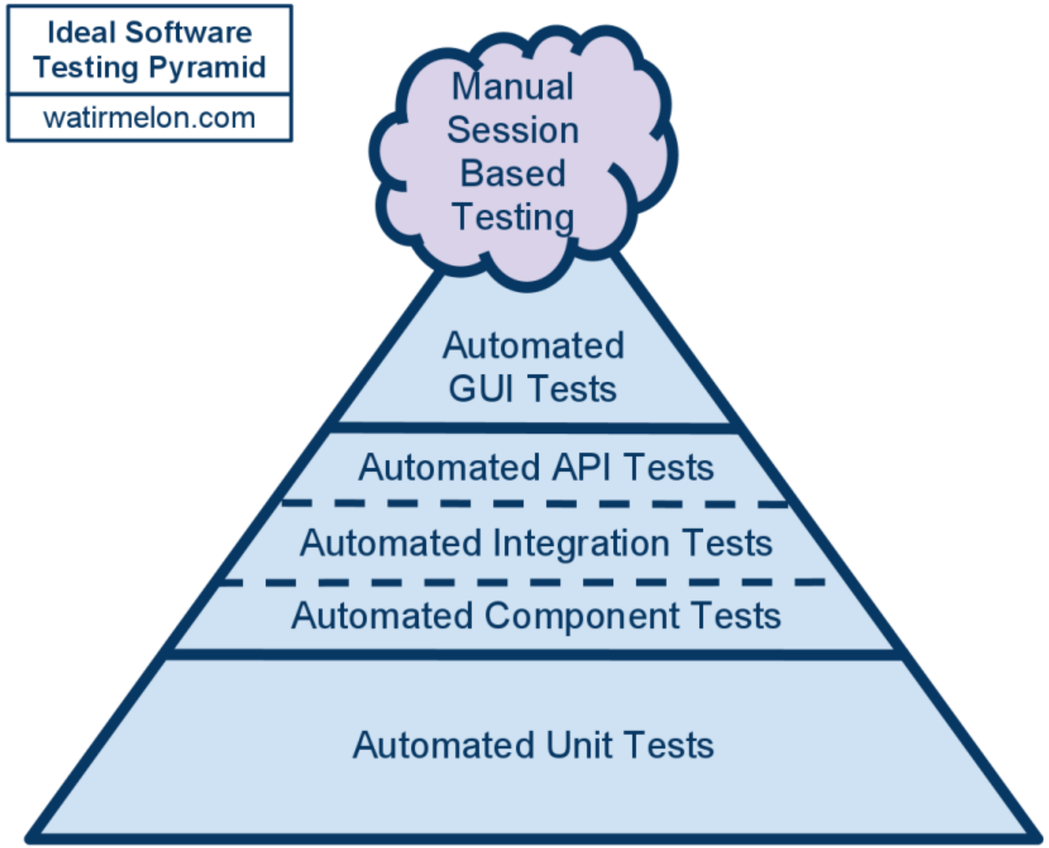
https://testing.googleblog.com/2010/12/test-sizes.html

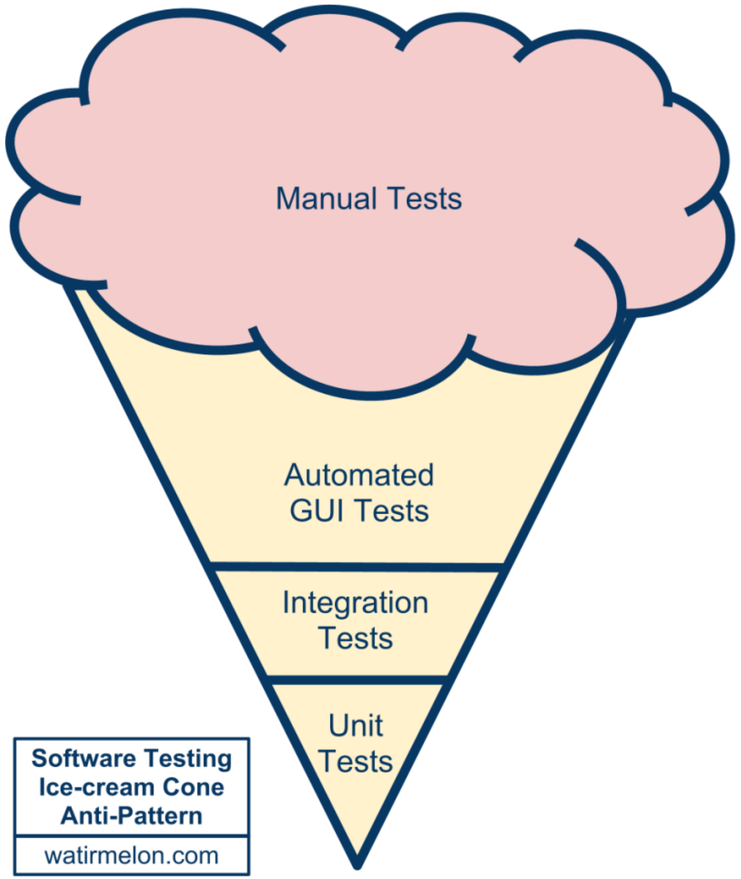
# テストピラミッド:
# 自動テストの信頼性を
# 中長期的に保つ
# 最適なバランス

# テストピラミッド



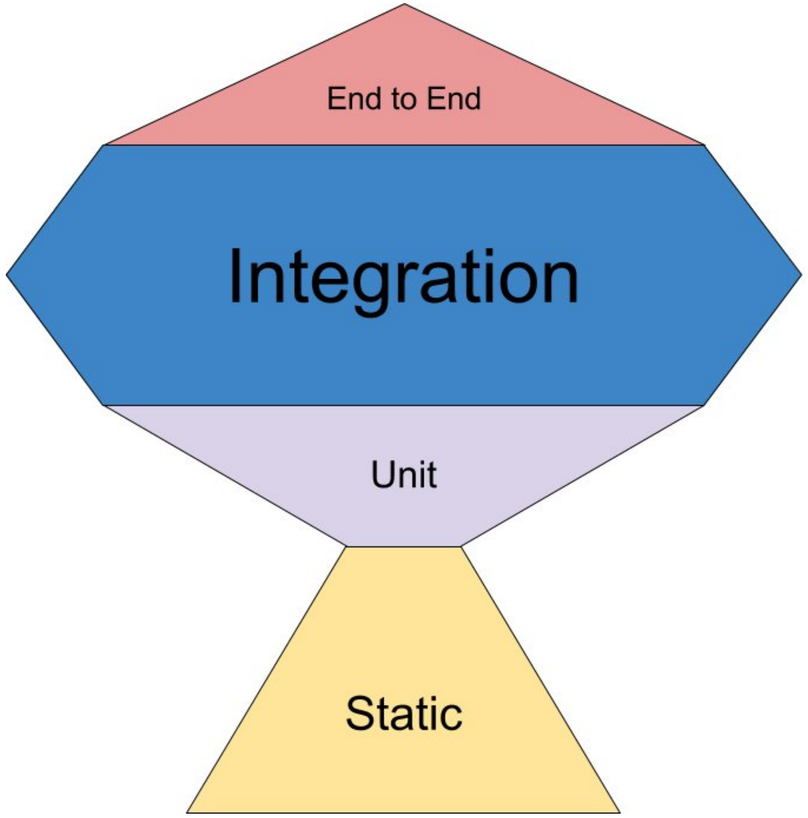
# テストピラミッドとアイスクリームコーンアンチパターン

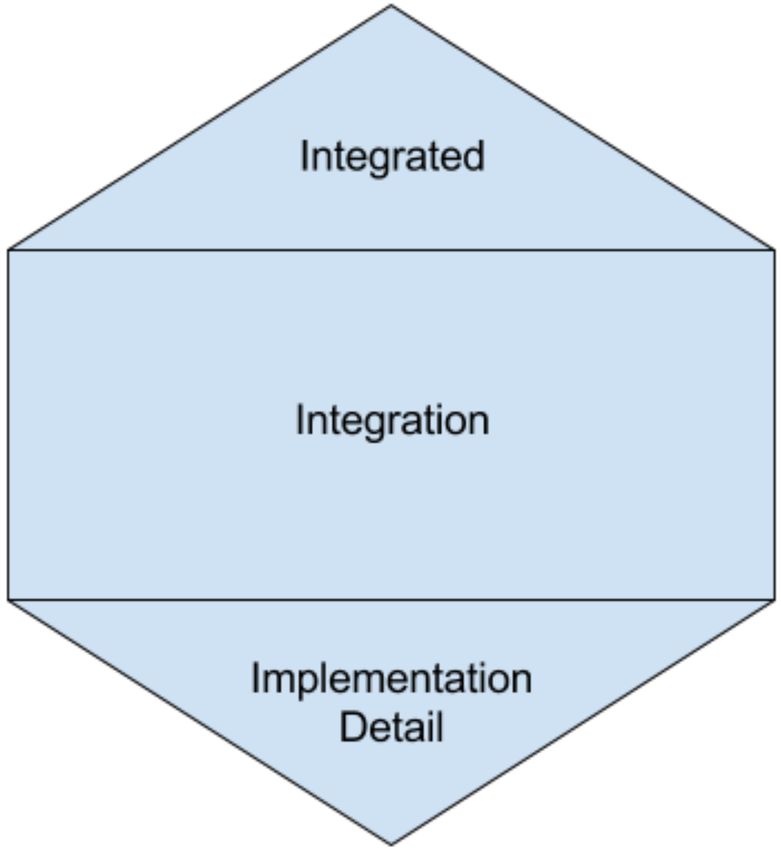


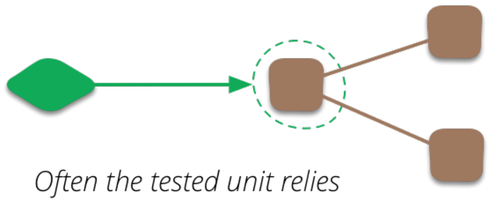

https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

# 混乱は「ユニット」「インテグレーション」の解釈のブレから生じがち



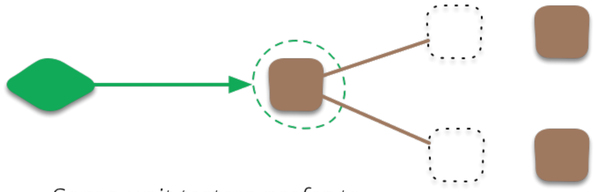




**Sociable Tests**



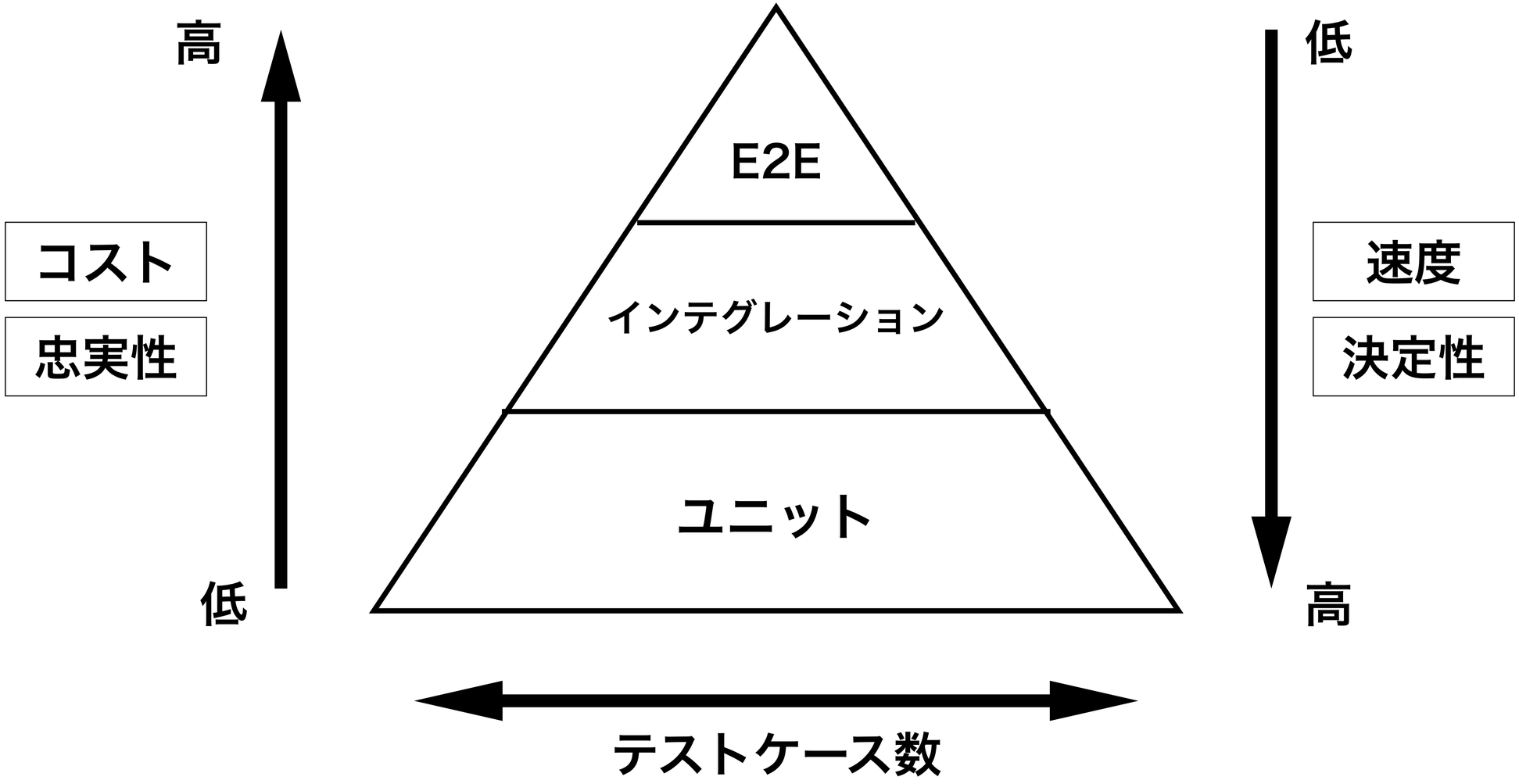
*Often the tested unit relies on other units to fulfill its behavior*

**Solitary Tests**

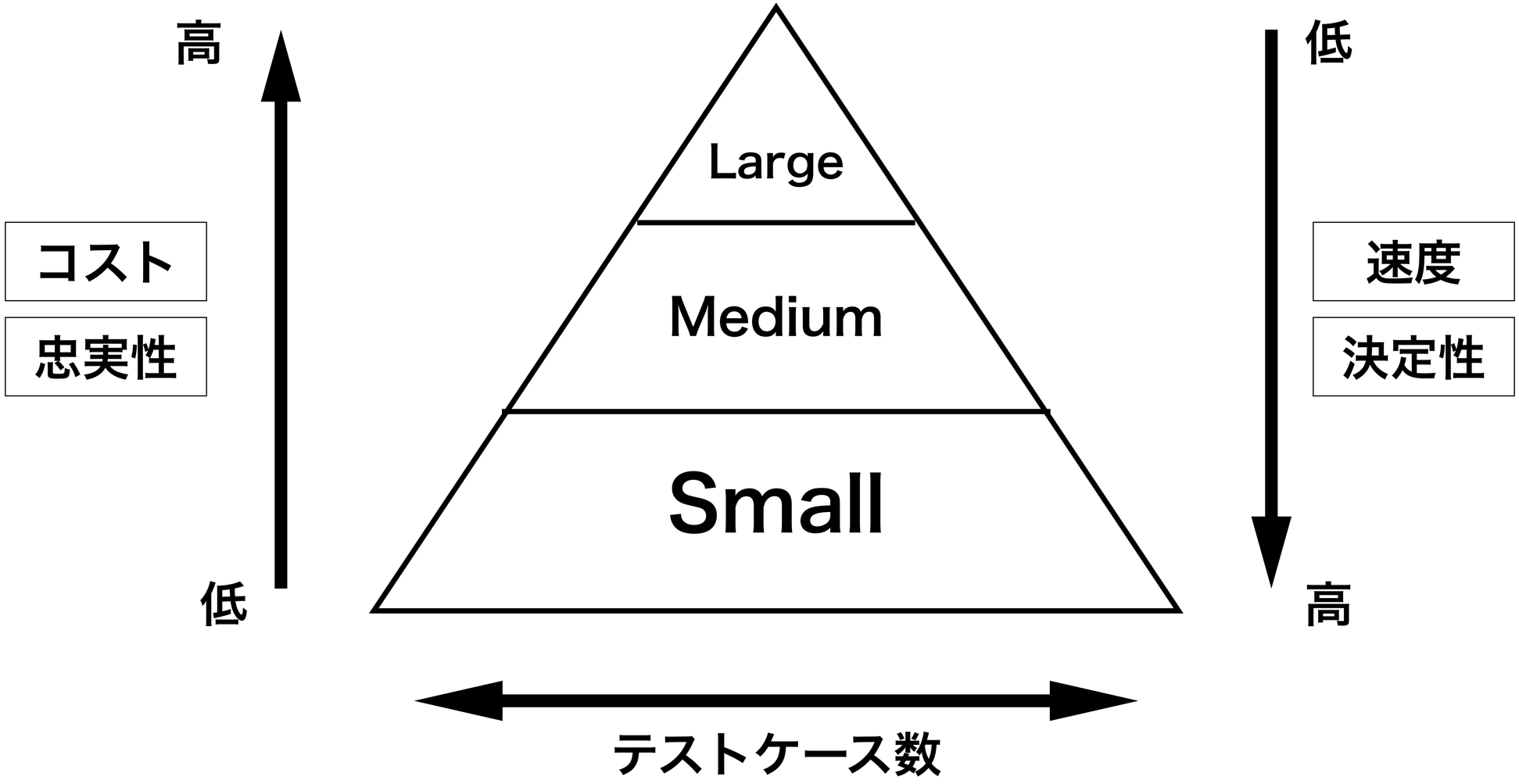


*Some unit testers prefer to isolate the tested unit*

# ブレの少ないテストの分類基準は……？

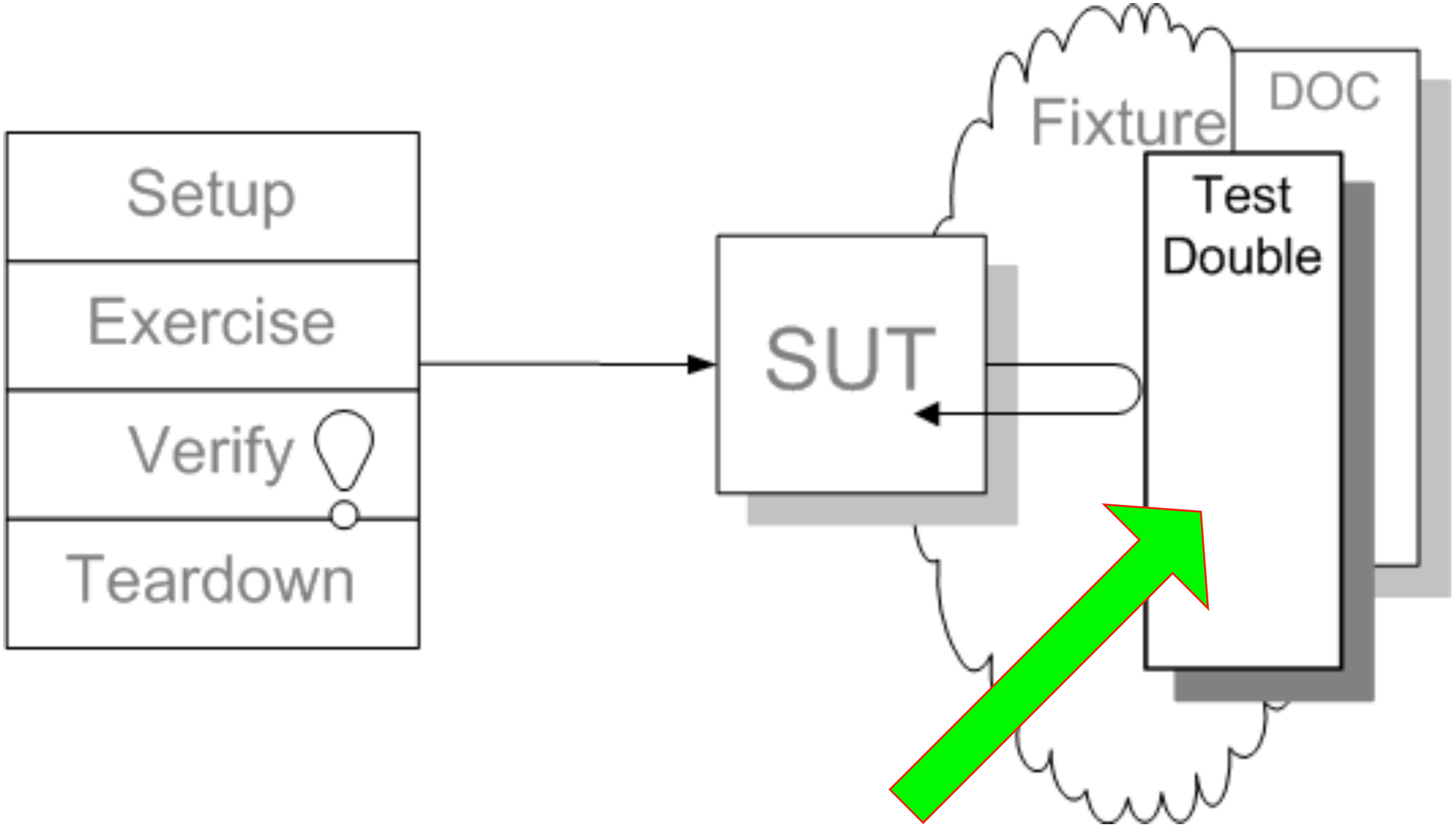


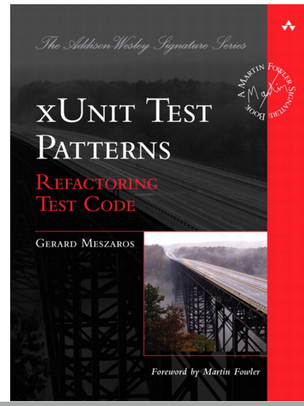
# ブレの少ないテストの分類基準 → テストサイズ



# テストダブル:
# 忠実性と決定性の
# トレードオフを理解する

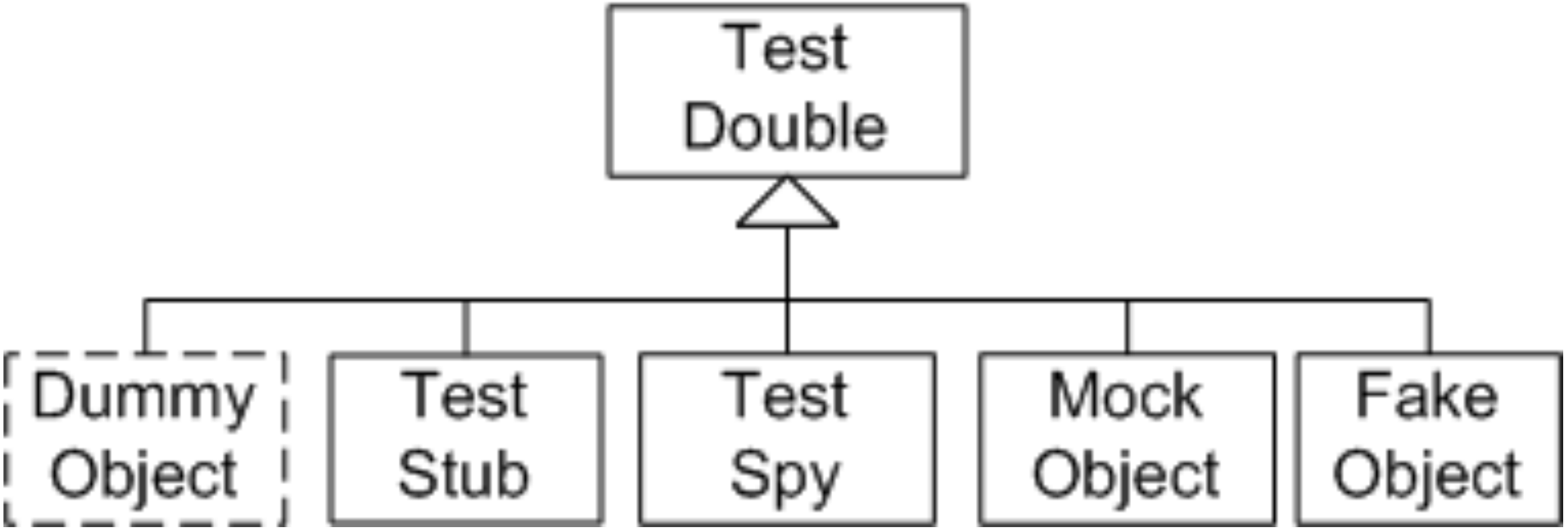
# テストダブル: 自動テスト用の偽物

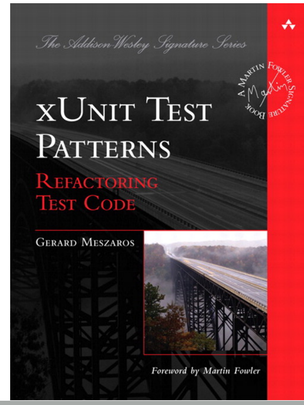



http://xunitpatterns.com/Test%20Double.html

# テストダブルの分類（xUTP）



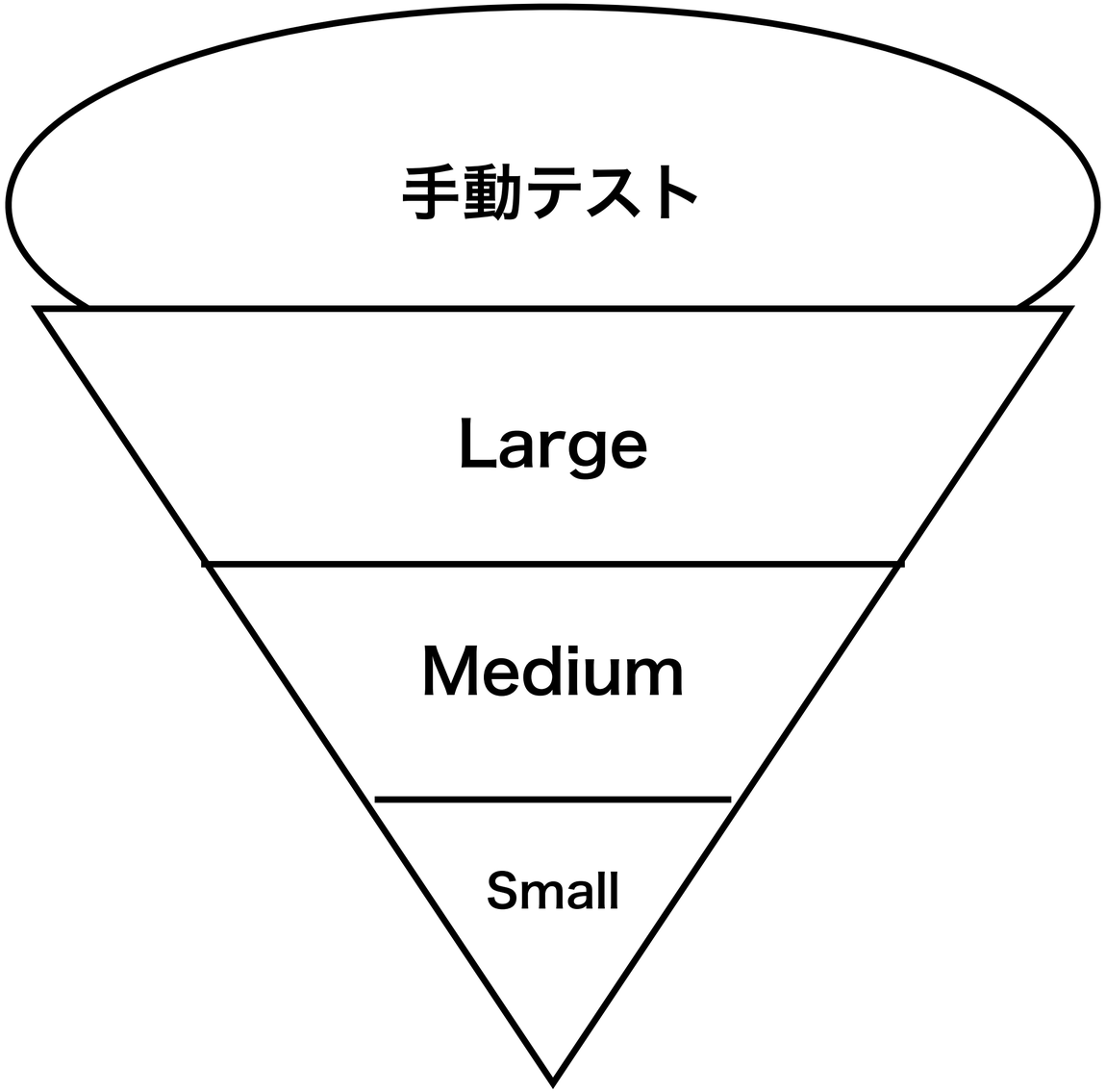

http://xunitpatterns.com/Test%20Double.html

# テストダブルの利点と注意点

- 利点
    - そもそもテストしにくいものをテスト可能にする
    - テストの速度と決定性を向上させる
- 注意点
    - テスト対象の実装との構造的結合が高まり、テストが脆くなり、偽陽性を招く
    - 自作自演テストのリスクがあり、偽陰性を招く

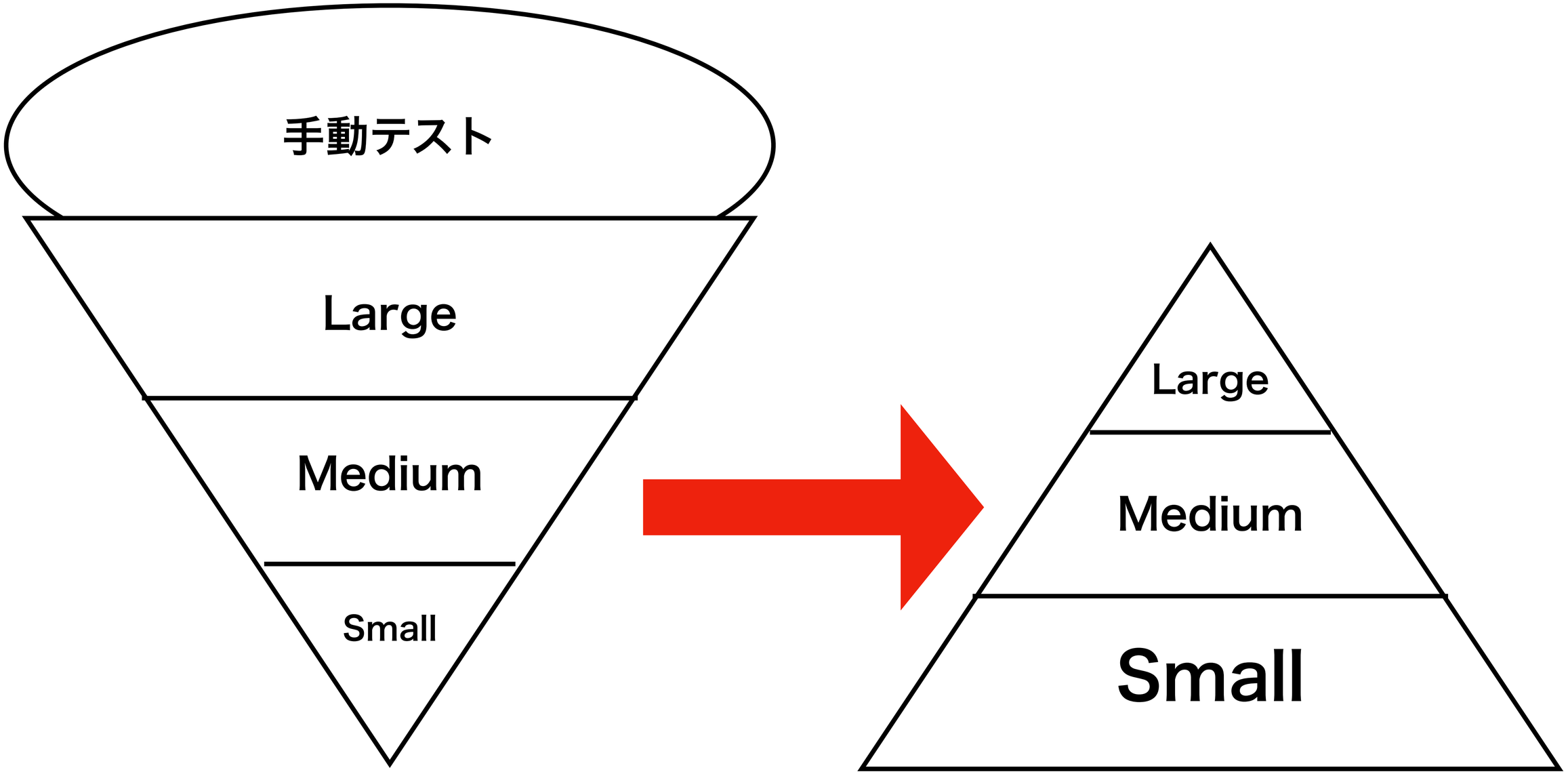
# サイズダウン戦術と
# テストピラミッド

# でも、多くの現場ではアイスクリームコーンから始まる（それは悪いことではない）



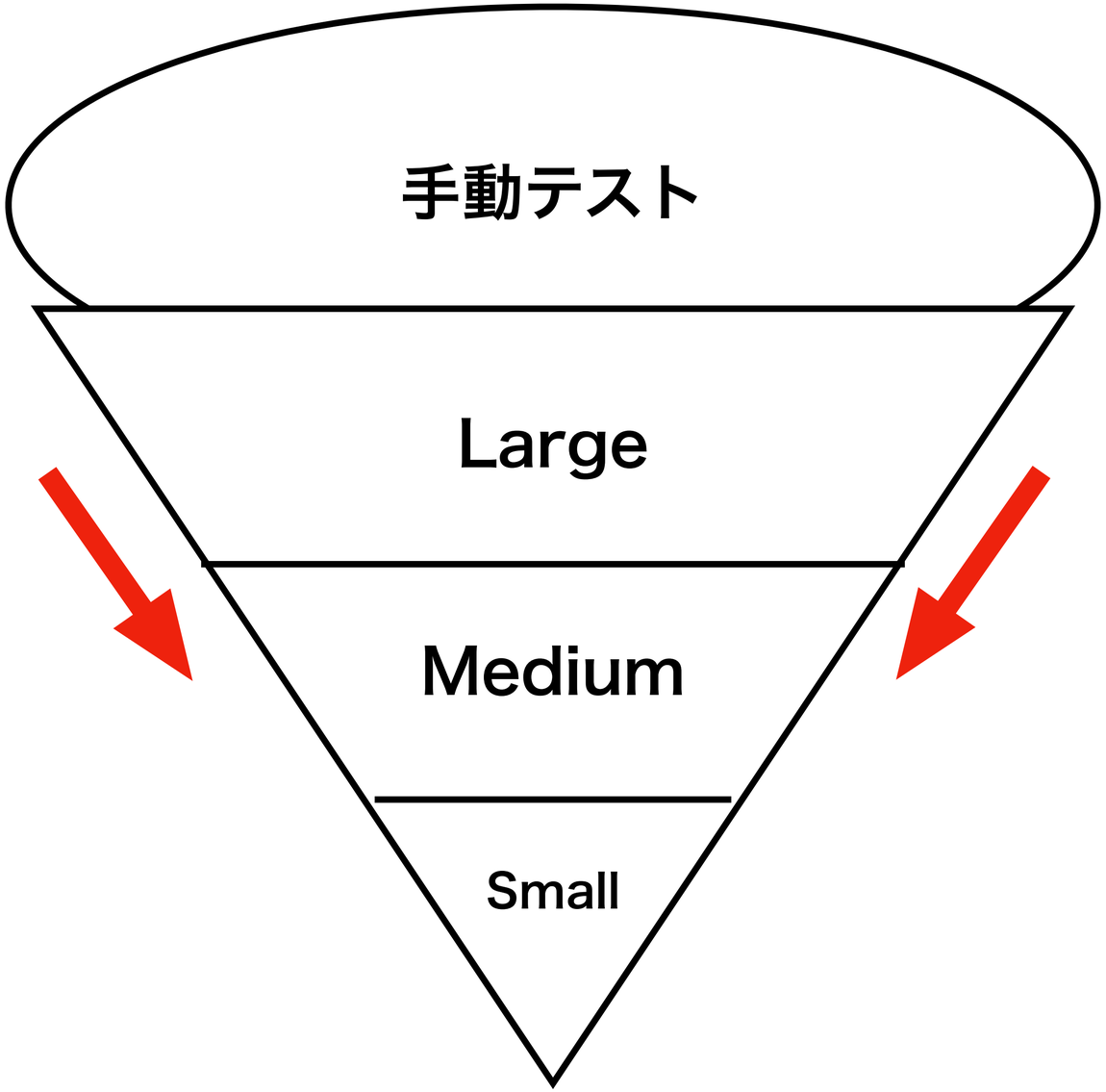

https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

# どうやってアイスクリームコーンをピラミッドにするか



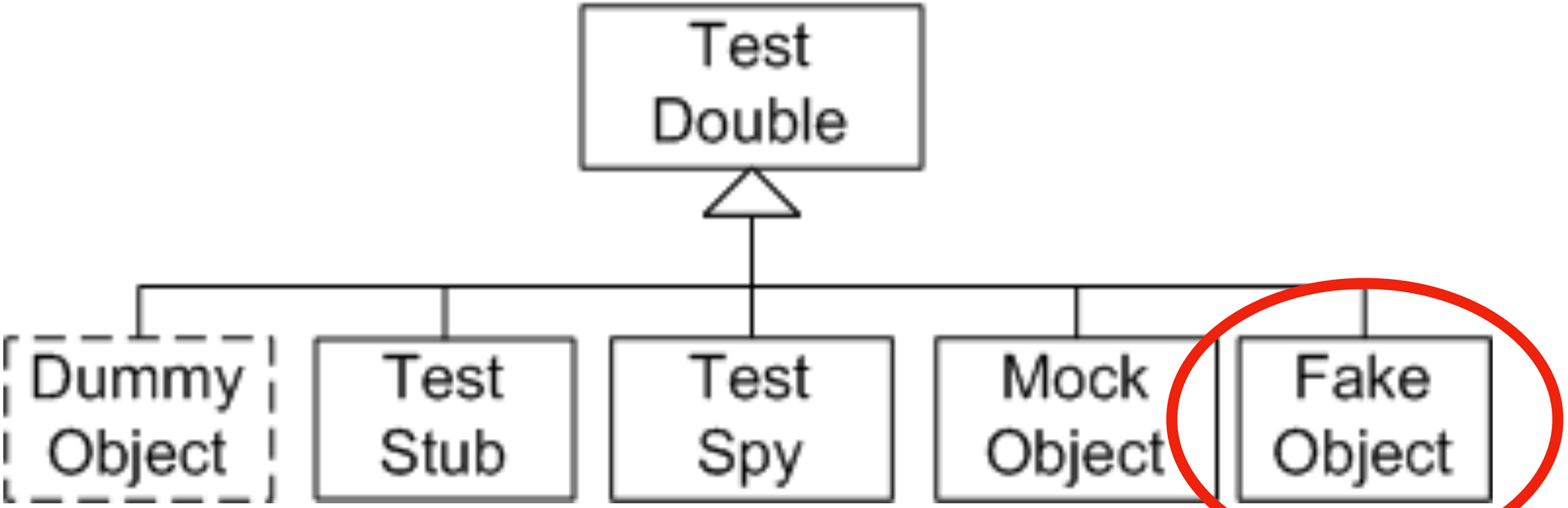

https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

# Large から Medium へ

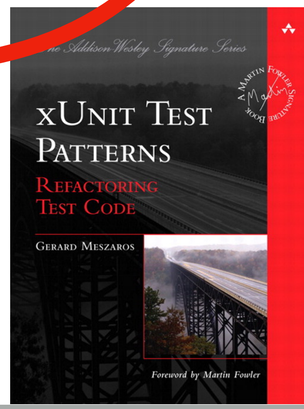



https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

# Large から Medium へ: Fake Object

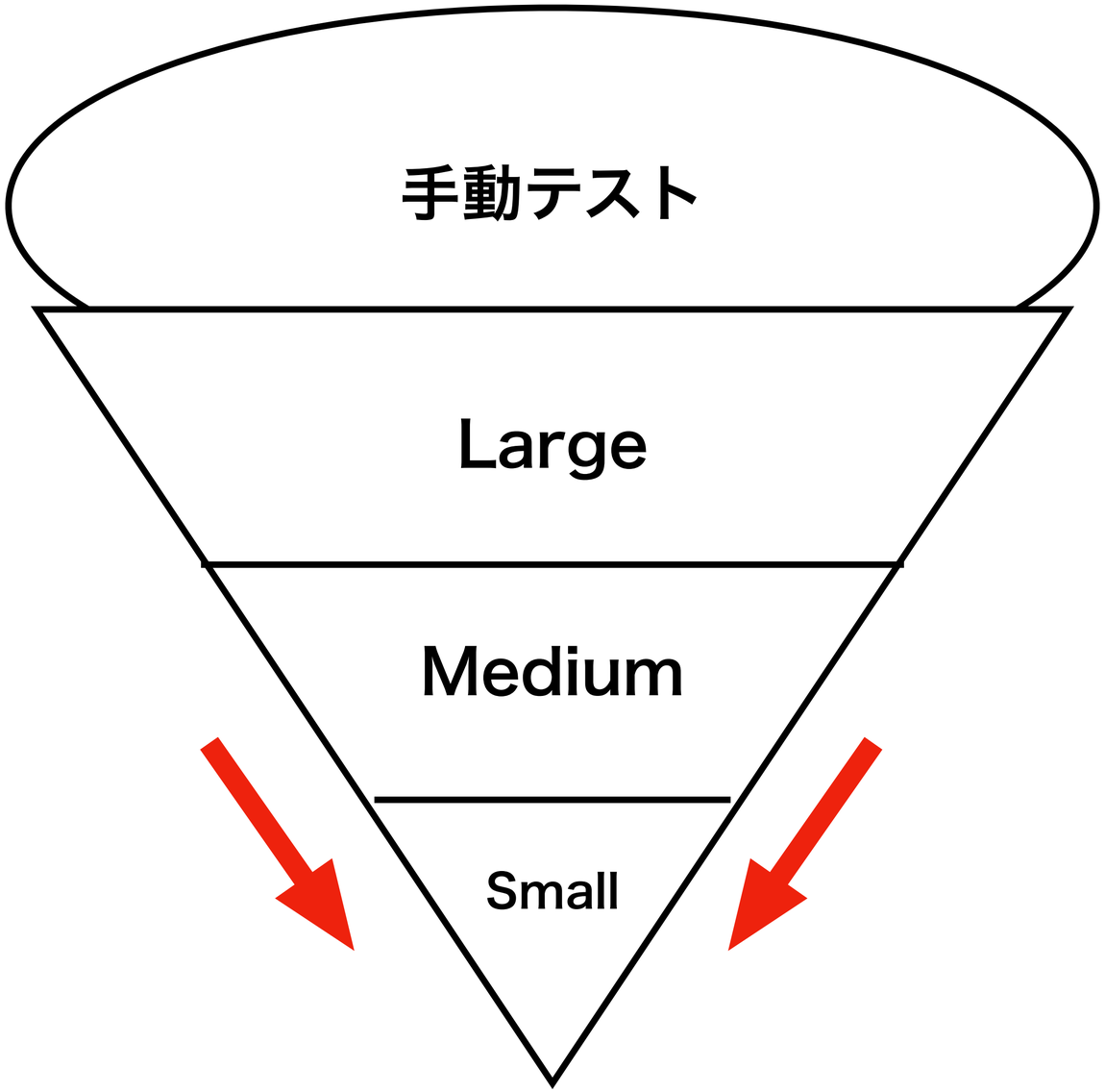



Fake は自動テスト用の代替実装。
DynamoDB に対する DynamoDB local や locakstack が代表例。
コンテナで動作させられればテストサイズが Medium に収まる。




http://xunitpatterns.com/Test%20Double.html

# Medium から Small へ

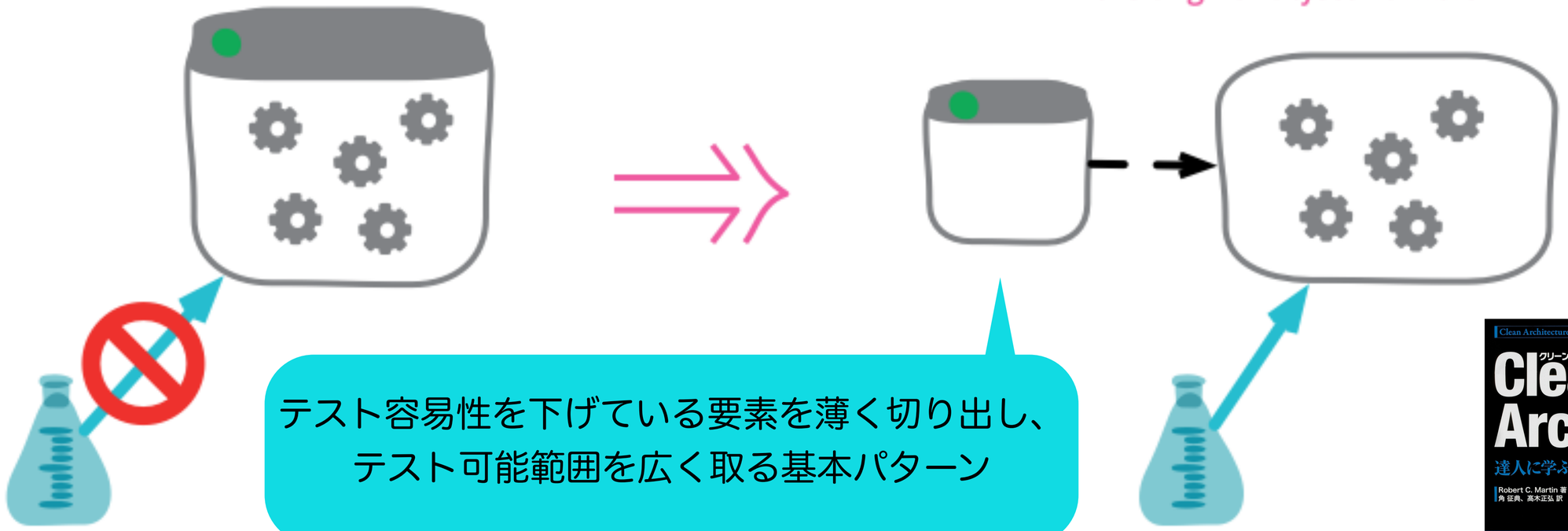



https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/

# Medium から Small へ: Humble Object

*faced with a software element that's difficult to test*

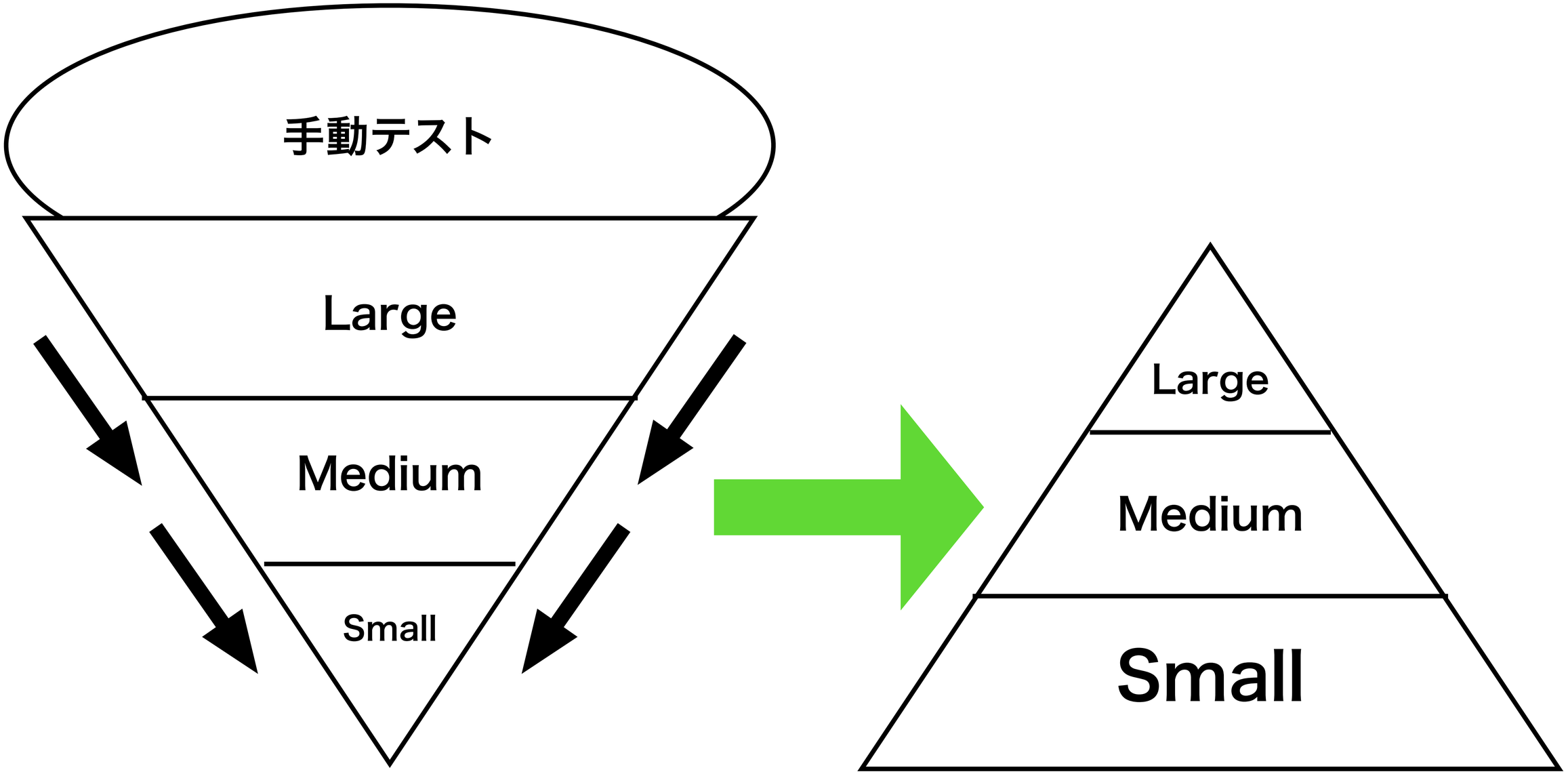
*move the logic into a separate element that is testable, making the original object **humble***



テスト容易性を下げている要素を薄く切り出し、
テスト可能範囲を広く取る基本パターン

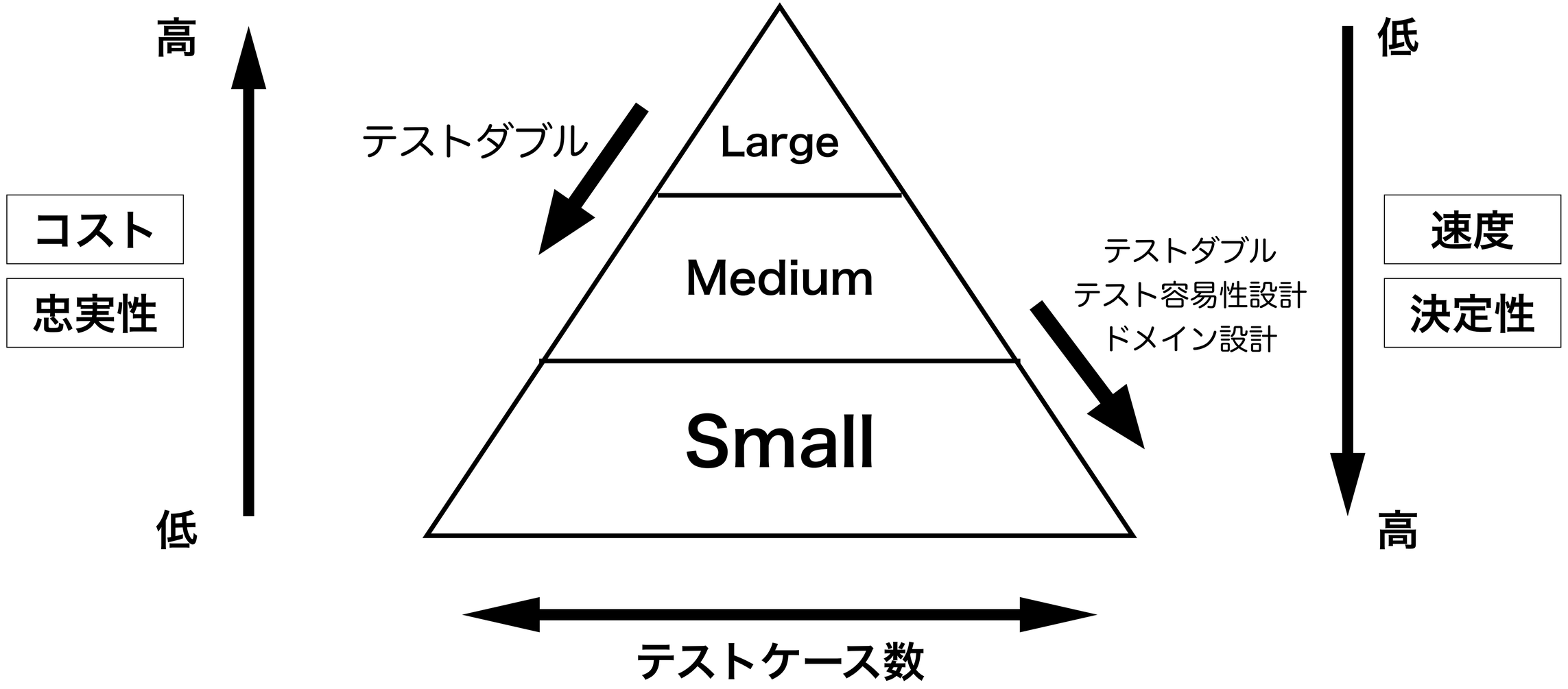

https://www.amazon.co.jp/dp/4048930656

# アイスクリームコーンからピラミッドへ




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

# テストダブルでサイズダウンして、各サイズをピラミッド型に配置し、テスト全体の信頼性を維持する



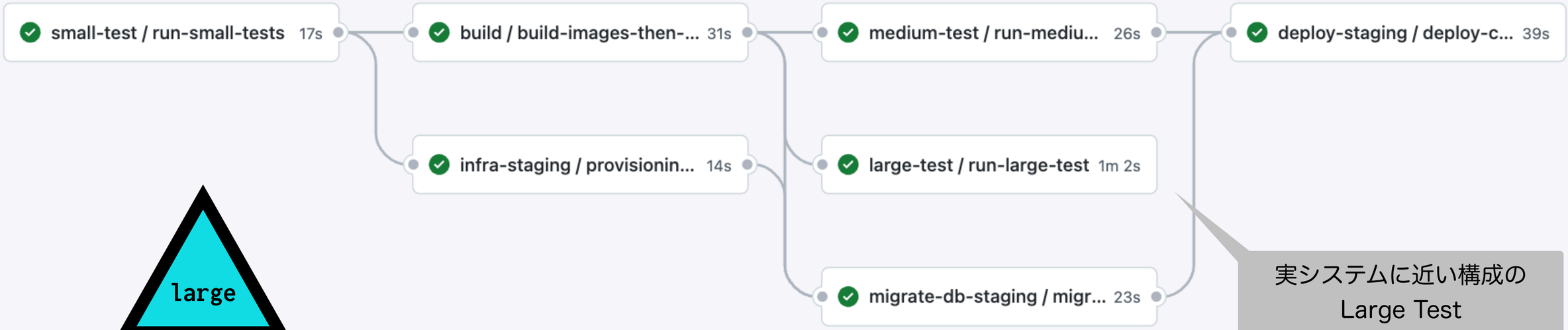
# テストピラミッドとテストサイズとビルドパイプライン

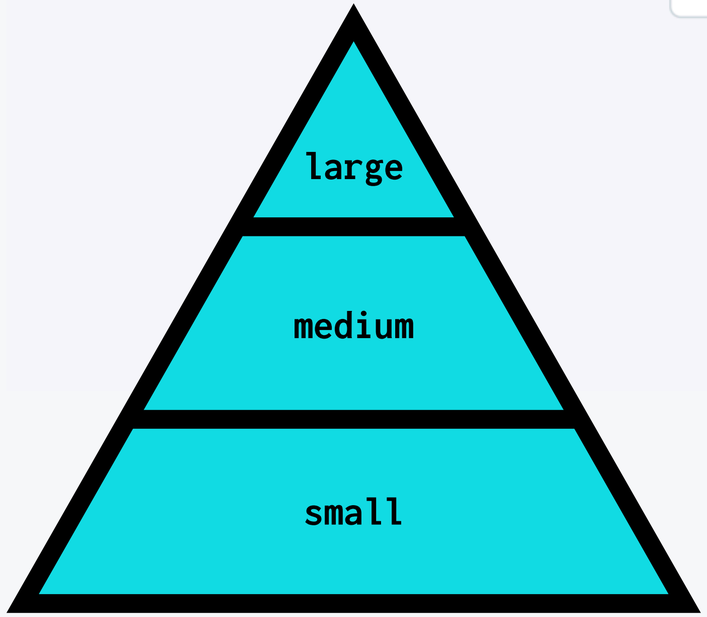
**ci.yml**

on: pull_request

メモリ内で動く Small Test
できるだけここの比率を高めたい

ビルド後のイメージや DB 等を組み合わせた
Medium Test




実システムに近い構成の
Large Test
（なので遅くて不安定）

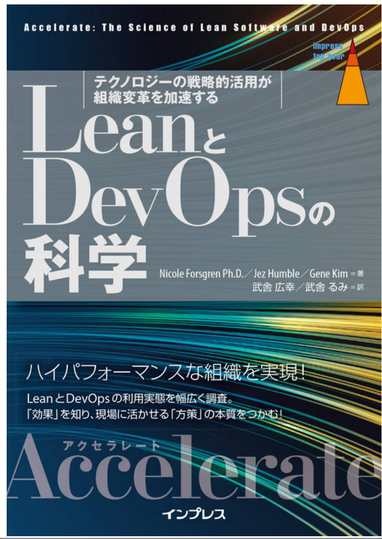
# Agenda

- 信頼性の高い
  実行結果に
  短い時間で到達する
- 状態を保つ

# 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

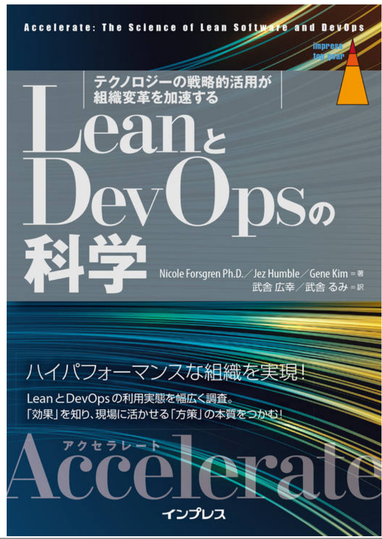
1. 誤検知（偽陽性: false positive）や見逃し（偽陰性: false negative）が多く、信頼性に欠けるテストスイートがあまりにも多すぎる
2. 信頼度の高いテストスイートを作り上げる継続的な努力と投資は価値がある




『LeanとDevOpsの科学』 p.65（※訳を一部変更）

# 信頼性の高い自動テストを備えること

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




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

# 偽陽性と偽陰性のパターン

- 偽陽性
    - 脆いテスト (brittle test, fragile test)
    - 信頼不能テスト (flaky test)
- 偽陰性
    - 空振り
    - カバレッジ不足、テスト不足
    - 自作自演

# 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/

# 偽陽性と偽陰性のパターン

- 偽陽性
  - 脆いテスト (brittle test, fragile test)
  - 信頼不能テスト (flaky test)
- 偽陰性
  - 空振り
  - カバレッジ不足、テスト不足
  - 自作自演

# Test Scope と Test Size

問題箇所特定の容易さ

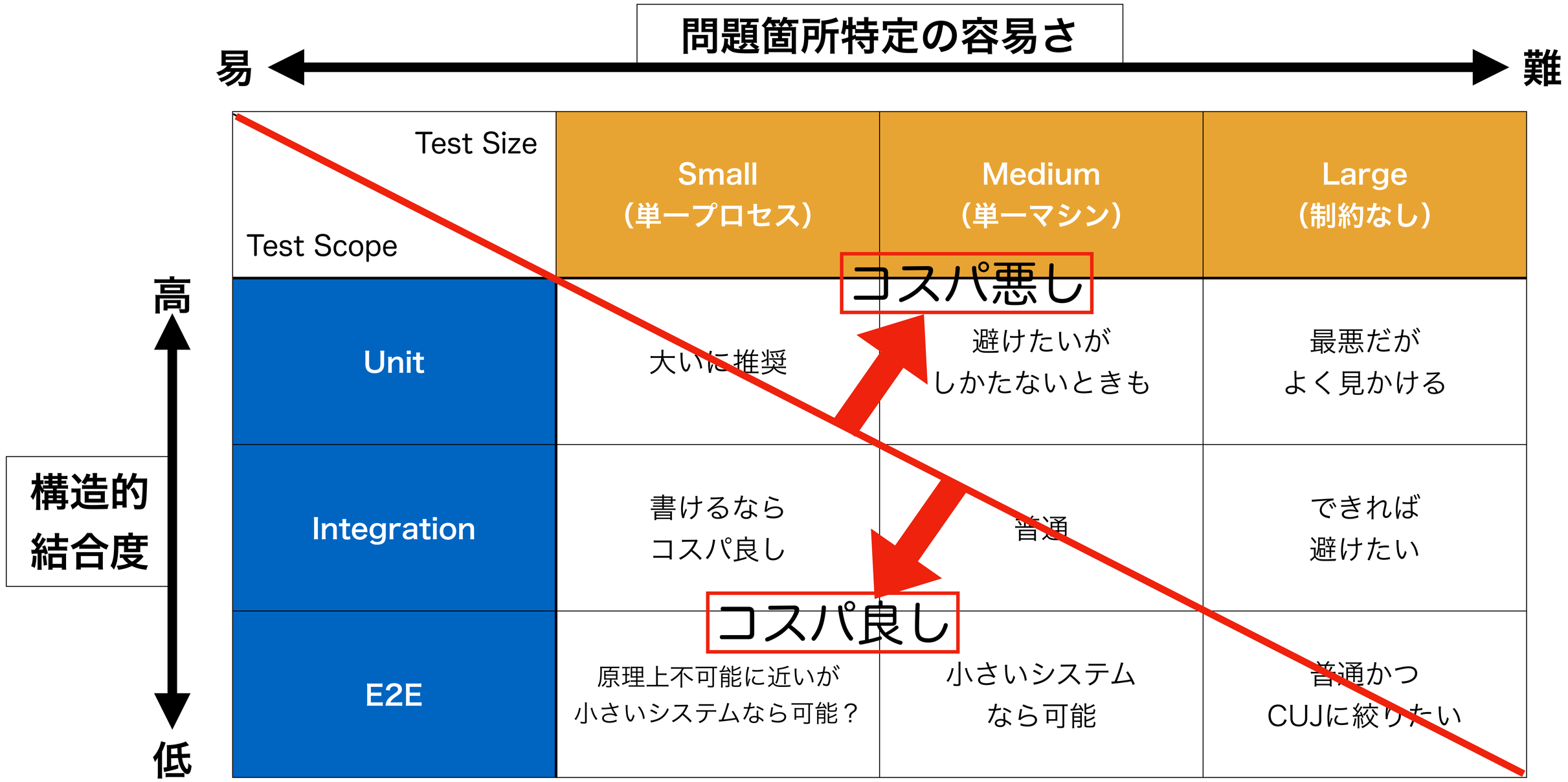
易 ◀ ▶ 難

構造的結合度

高 ▼ 低

| Test Scope \ Test Size | Small（単一プロセス） | Medium（単一マシン） | Large（制約なし） |
|---|---|---|---|
| Unit | 大いに推奨 | 避けたいが<br>しかたないときも | 最悪だが<br>よく見かける |
| Integration | 書けるなら<br>コスパ良し | 普通 | できれば<br>避けたい |
| E2E | 原理上不可能に近いが<br>小さいシステムなら可能？ | 小さいシステム<br>なら可能 | 普通かつ<br>CUJに絞りたい |

# Test Scope と Test Size



# 偽陽性と偽陰性のパターン

- 偽陽性
    - 脆いテスト (brittle test, fragile test)
    - 信頼不能テスト (flaky test)
- 偽陰性
    - 空振り
    - カバレッジ不足、テスト不足
    - 自作自演

# 自作自演の例: テスト対象ロジックのテストコードへの漏れ出し

```
// プロダクトコード
class Item {
  // コンストラクタ割愛
  tax_amount() {
    const rate = (this.tax_rate / 100);
    return (this.price / (1 + rate)) * rate;
  }
}

// テストコード
it('税込価格から税額を返す', () => {
  const item = new Item('技評茶', 130, 8);
  const expected = (130 / (1 + (8 / 100))) * (8 / 100);
  assert.equal(item.tax_amount(), expected);
});
```

1円未満の端数が発生する
バグがある

テストコードの方も同じロジックで
期待値を計算しているので
テストが成功してしまう

# Agenda

信頼性の高い

実行結果に

短い時間で到達する

🦁 状態を保つ

# 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。

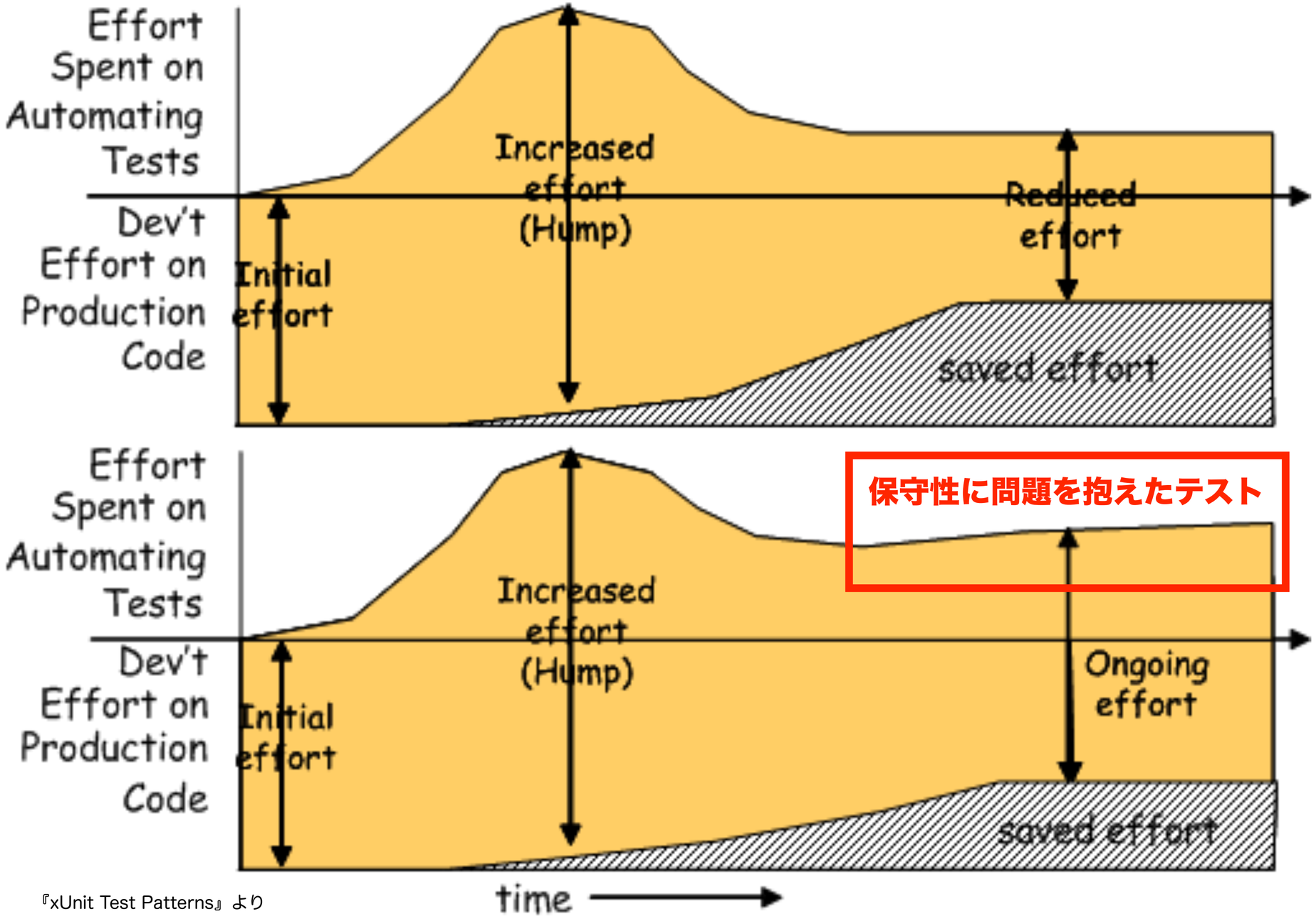
『Googleのソフトウェアエンジニアリング』 p.249


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

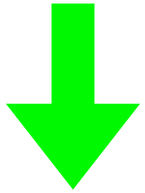



『xUnit Test Patterns』より

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

# 自動テスト文化とは

- 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文化

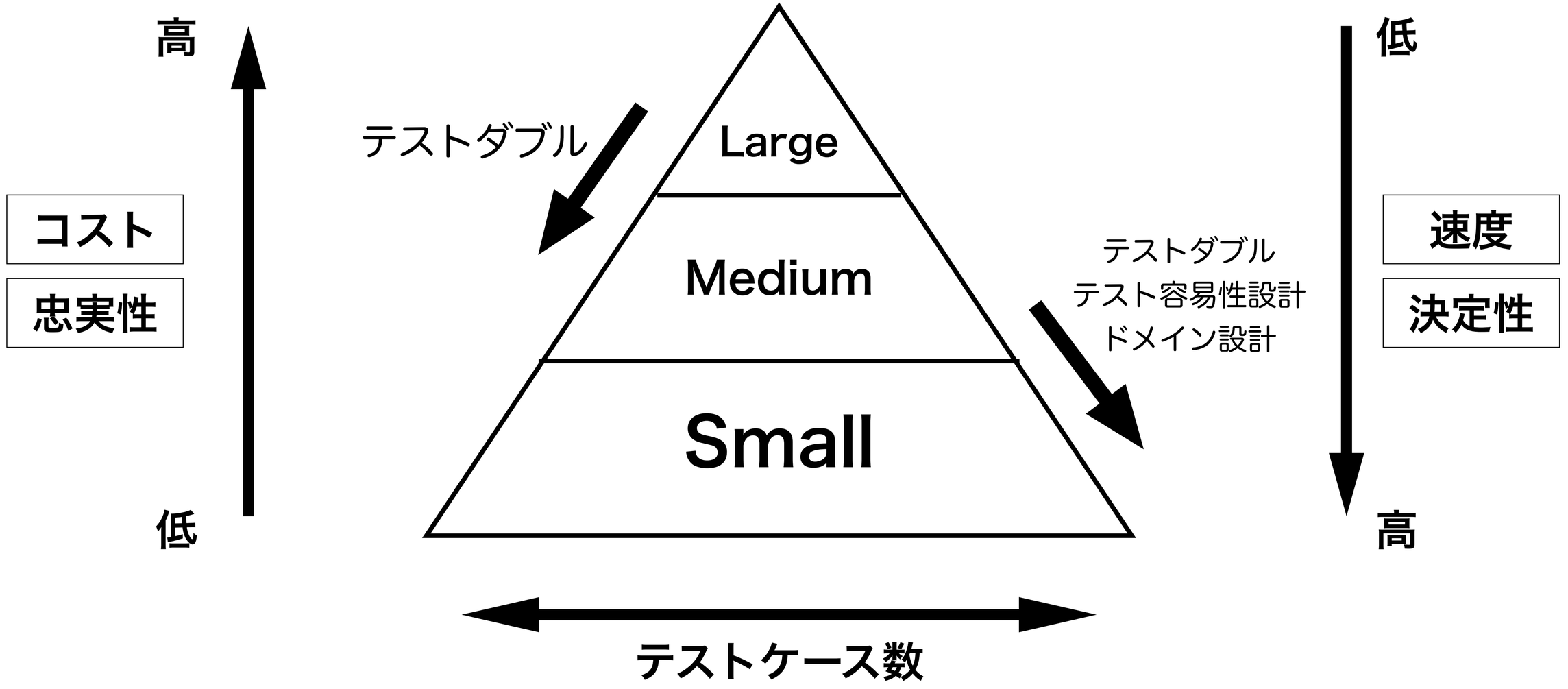
テストにコストがかかることの解決方法は、テストをやめることではありません。うまくなることです。

-- 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

# まとめ: テストダブルでサイズダウンして、各サイズをピラミッド型に配置し、テスト全体の信頼性を維持する



ご清聴ありがとうございました