

サバンナ便り

ソフトウェア開発の荒野を生き抜く



サバンナ便り

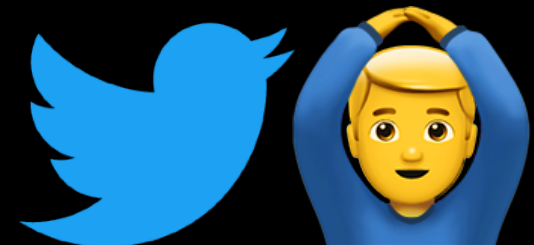
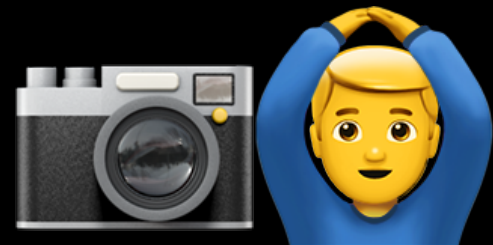
自動テストに関する連載で得られた知見のまとめ

(2023/05版)



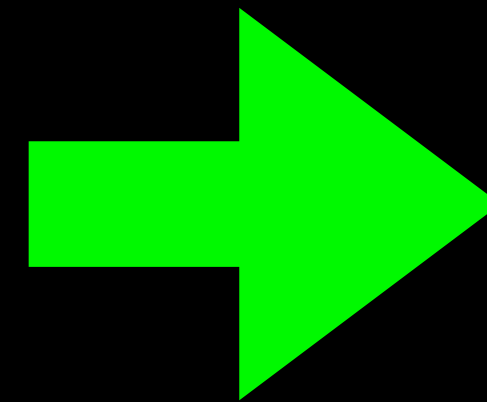
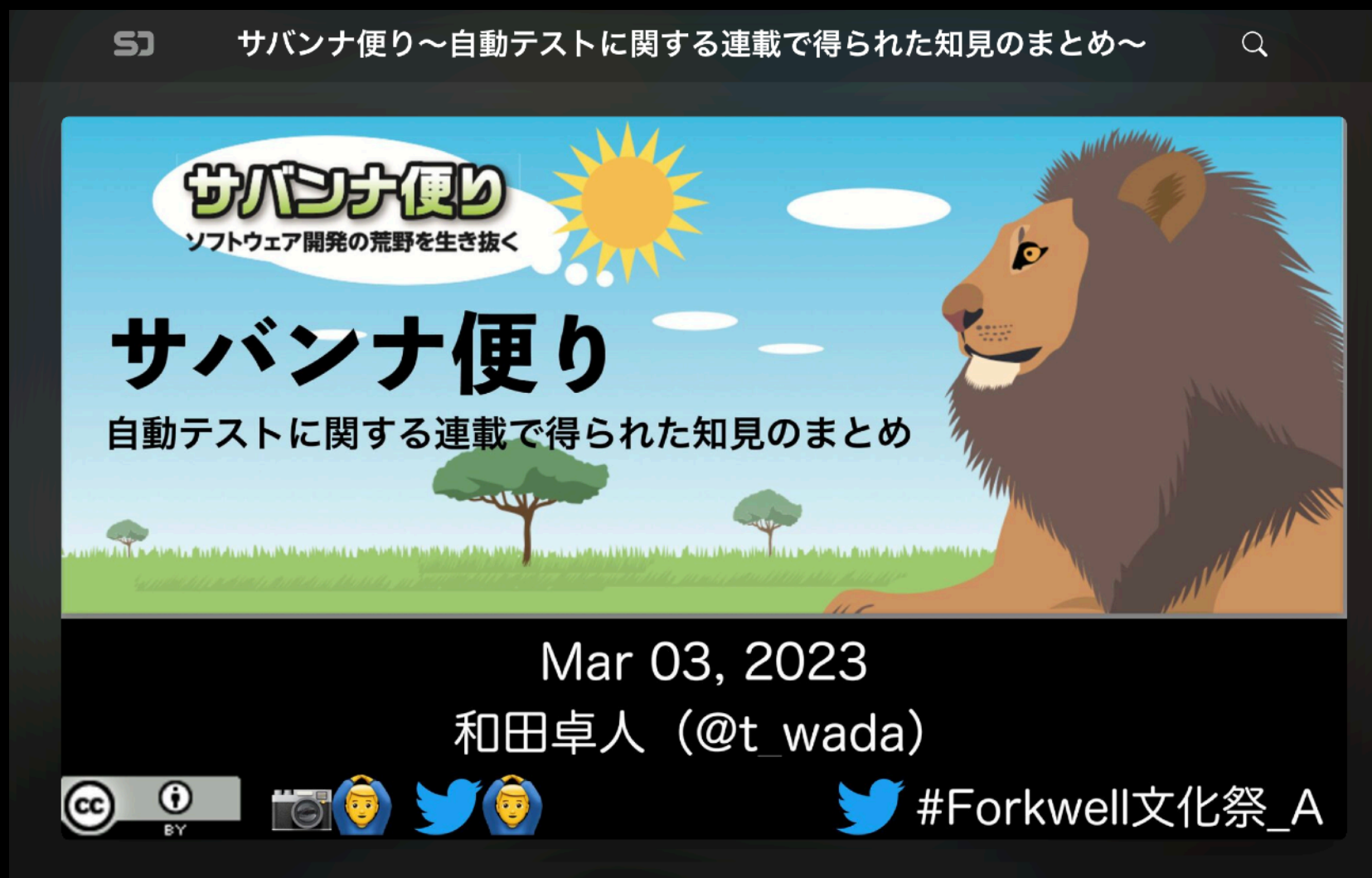
May 17, 2023

和田卓人 (@t_wada)



#QiitaConference

2023/03



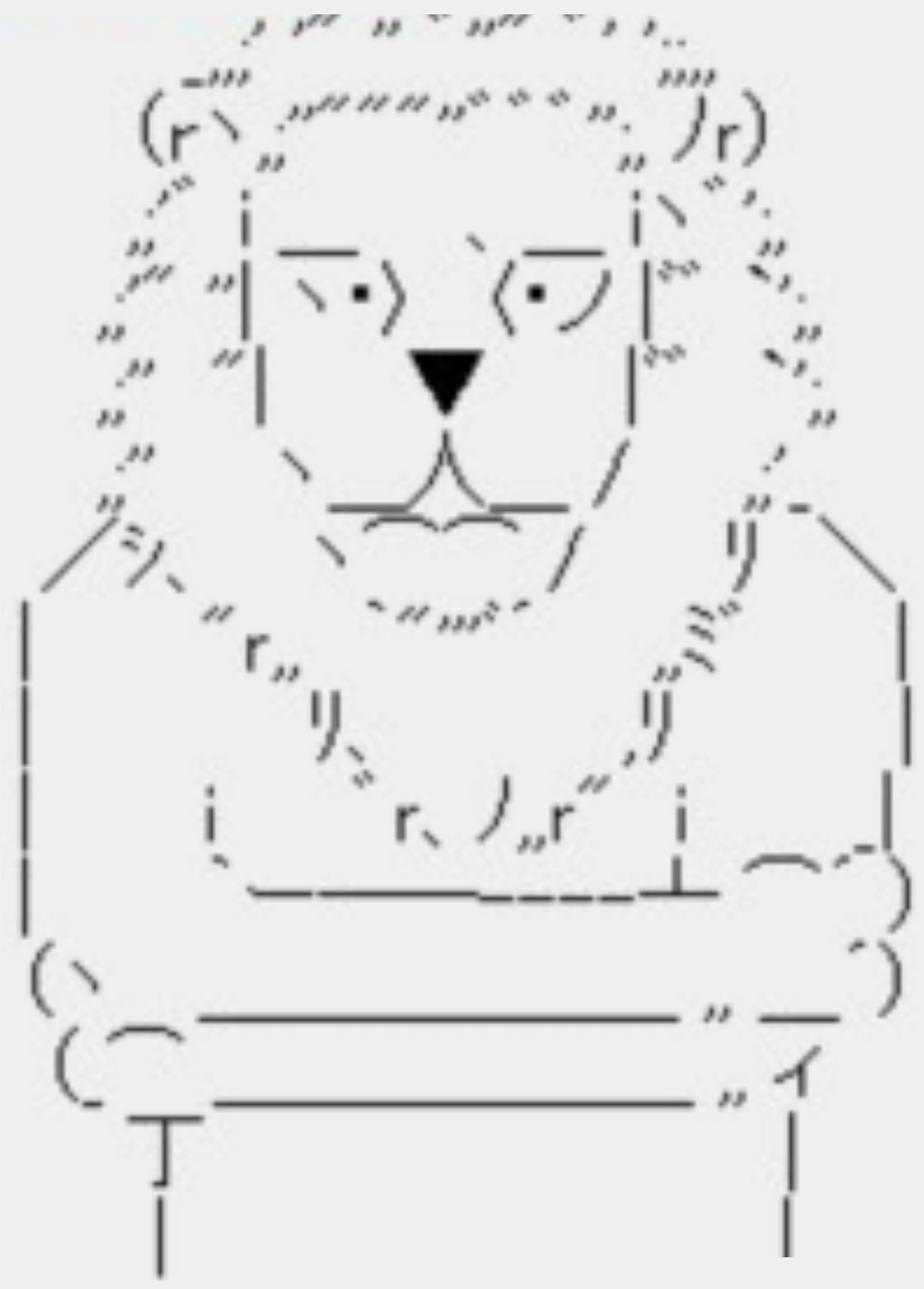
2023/05



コラム連載の第6回まで収録しています

講演の文字起こし等も今後公開予定です

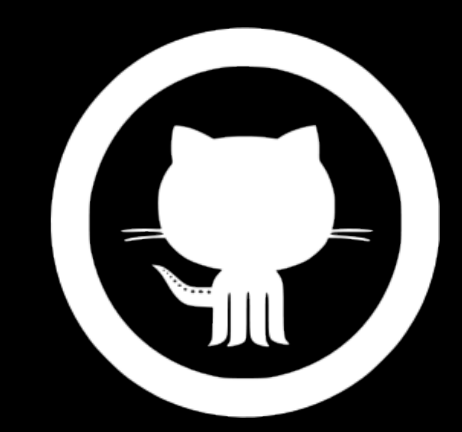
公開されたら speakerdeck の同一 URL で
資料をアップデートします



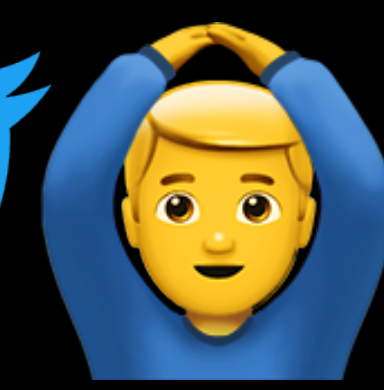
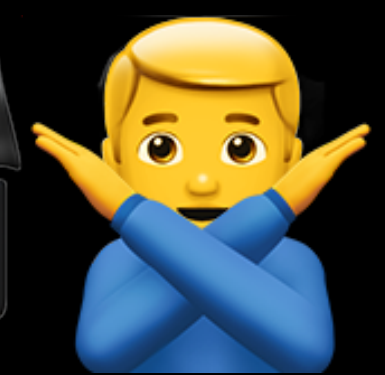
t-wada



t_wada



twada



#QiitaConference

技術書の出版に関わっています

97



Collective Wisdom from the Experts

プログラマが知るべき97のこと

97 Things Every Programmer Should Know

Kevin Henney ■
和田 卓人 監修
夏目 大 訳

O'REILLY®
オライリー・ジャパン

Avoiding the Pitfalls of Database Programming

SQL アンチパターン

Bill Karwin 著
和田 卓人 監訳
和田 省二
見島 修 訳



O'REILLY®
オライリー・ジャパン

Test-Driven Development
By Example

テスト駆動開発

Kent Beck 著
和田卓人 訳



テスト駆動開発 (TDD) を原点から学ぶ

本書は、Kent Beck, *Test-Driven Development by Example* (Addison-Wesley, 2003) の新訳版です。

Ohmsha

事業をエンジニアリングする技術者たち

株式会社 CARTA HOLDINGS 監修
和田卓人 編

フルサイクル開発者がつくる
CARTAの現場



Ohmsha

CARTA

サバミンナ便り……………？

WEB+DB PRESS にコラムを連載しています

Webアプリケーション開発のためのプログラミング技術情報誌
FOR ALL WEB APPLICATION DEVELOPERS **ウェブDBプレス**

WEB+DB PRESS

最新バージョンから読み解く

宣言的UI コンポーネント指向 純粋性

vol. **129**
2022

Reactの深層

変わる常識と変わらない思想

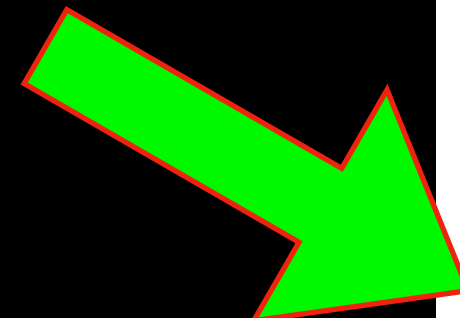
小さく始める

デザインシステム

協調フィルタリングから深層学習まで
レコメンドエンジン **総実装**

Ktor/KotlinでWeb開発

新連載 和田卓人の「サバンナ便り」
SREで開発を加速させる



エンジニアの総実装

Ktor/KotlinでWeb開発

和田卓人の「サバンナ便り」
SREで開発を加速させる

新連載

よろしくお願ひします





Agenda: 連載各回のテーマ



学習用テスト

偽陽性と偽陰性

テストサイズ

テストダブル

テストピラミッド

自動テストのサイズダウン戦略

学びを自動テストとして書く

即時性と再現性

学習例: PHP の DateTime と DateTimeImmutable

DateTime クラス

(PHP 5 >= 5.2.0, PHP 7, PHP 8)

はじめに

このクラスは、[DateTimeImmutable](#) と同じ振る舞いをします。但し、[DateTime::modify\(\)](#) のような、オブジェクトそのものを変更するメソッドが呼び出された時を除きます。

クラス概要

Change language: 

[Submit a Pull Request](#) [Report a Bug](#)

DateTimeImmutable クラス

(PHP 5 >= 5.5.0, PHP 7, PHP 8)

はじめに

日付と時刻を表現します。

クラス概要

DateTime と DateTimeImmutable の違いを学習テストにする

```
/**
 * @test
 * @group learning
 */
public function DateTimeのaddは自身の状態を変更しつつ自身を返す(): void
{
    $halloween = new \DateTime('2021-10-31');
    $oneYear = \DateInterval::createFromDateString('1 year');

    $halloween2022 = $halloween->add($oneYear);

    $this->assertSame($halloween, $halloween2022);
    $this->assertEquals('2022-10-31', $halloween->format('Y-m-d'));
    $this->assertEquals('2022-10-31', $halloween2022->format('Y-m-d'));
}
```

学習テスト（学びが目的のテスト）を
見分けるために
learning タグをつけています

ここが学び

```
/**
 * @test
 * @group learning
 */
public function DateTimeImmutableのaddは自身の状態を変更せず新しい状態を伴う新しいインスタンスを返す(): void
{
    $halloween = new \DateTimeImmutable('2021-10-31');
    $oneYear = \DateInterval::createFromDateString('1 year');

    $halloween2022 = $halloween->add($oneYear);

    $this->assertNotSame($halloween, $halloween2022);
    $this->assertEquals('2021-10-31', $halloween->format('Y-m-d'));
    $this->assertEquals('2022-10-31', $halloween2022->format('Y-m-d'));
}
```

ここが学び

疑問をテストにする

疑問をテストにする

```
/**
 * @test
 * @group learning
 */
public function 同じ時刻を指している場合はタイムゾーンが異なっても等価とみなされる(): void
{
    $utc = new DateTimeImmutable('2021-12-24T15:00:00', new DateTimeZone('UTC'));
    $jst = new DateTimeImmutable('2021-12-25T00:00:00', new DateTimeZone('Asia/Tokyo'));
    $this->assertTrue($utc == $jst);
}
```

テストに聞いてみればいい

**驚きをテストにする
例えば……**

一体

いつから——

コンストラクタは

一回しか呼べないと

錯覚していた？

なん……だと……! ?

```
/**
 * @test
 * @group learning
 */
public function コンストラクタをもう一度呼ぶと破壊的変更ができてしまう(): void
{
    $dt = new \DateTimeImmutable('2021-12-24');
    $this->assertSame('2021-12-24', $dt->format('Y-m-d'));

    $dt->__construct('2022-01-01');
    $this->assertSame('2022-01-01', $dt->format('Y-m-d'));
}
```

明示的に呼べてしまう

余談1: このテストコードをきっかけに php-src に issue として報告され、議論が行われました。

<https://github.com/php/php-src/issues/8351>

余談2: このテストコードをきっかけに静的解析ツール PHPStan と Psalm に機能提案が行われ、PHPStan に @muno_92 さんが作成した pull-request が採用され、リリースされました。

<https://github.com/phpstan/phpstan-src/pull/1208>





Agenda: 連載各回のテーマ

学習用テスト



偽陽性と偽陰性

テストサイズ

テストダブル

テストピラミッド

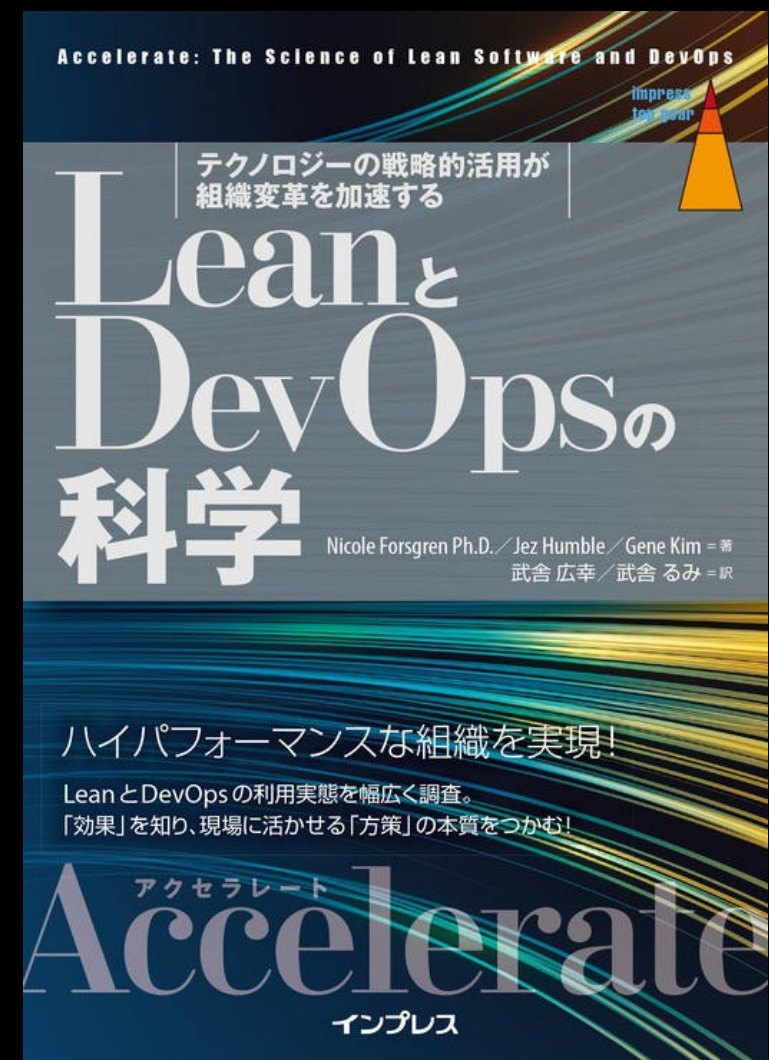
自動テストのサイズダウン戦略

**自動テストの信頼性を
むしろむ現象を理解する**

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)



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）

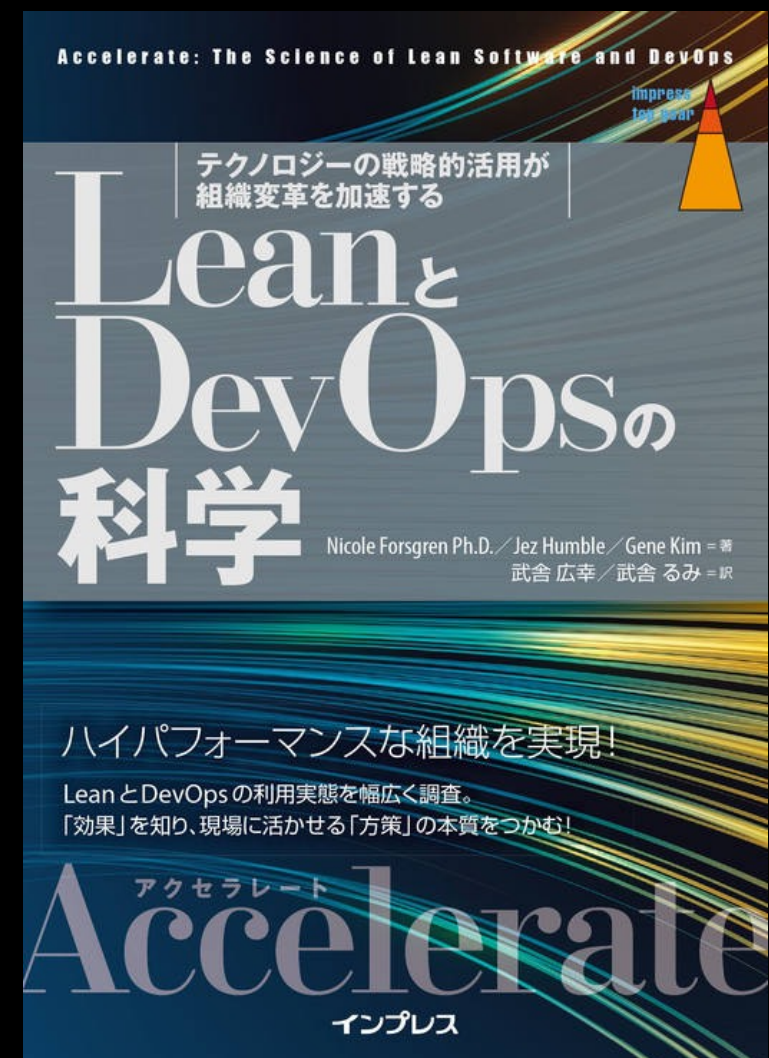


表1 偽陽性と偽陰性

製品コードが テスト結果が	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

- 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
 - テスト対象ロジックのテストコードへの漏れ出し

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;
  }
}
```

1円未満の端数が発生する
バグがある

```
// 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);
});
```

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

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





Agenda: 連載各回のテーマ

学習用テスト

偽陽性と偽陰性



テストサイズ

テストダブル

テストピラミッド

自動テストのサイズダウン戦略

自動テストと CI にフィットする 明確なテスト分類基準

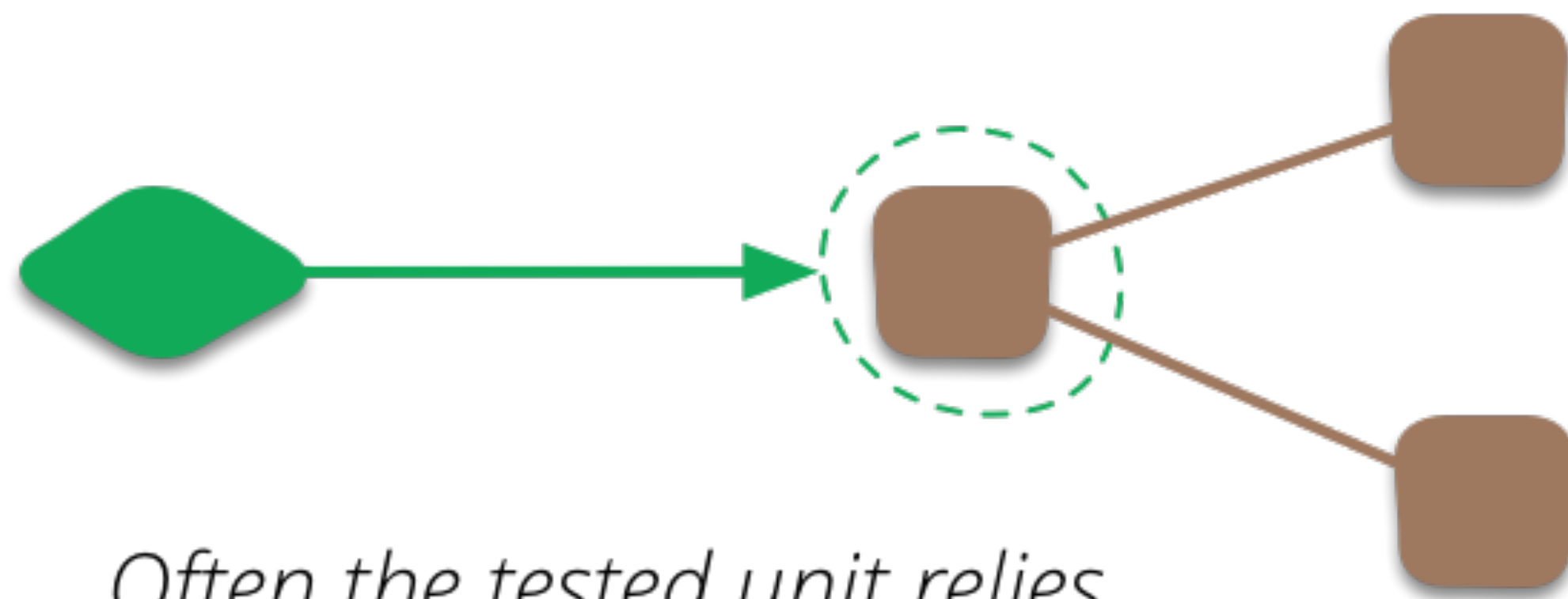
皆さんにお伺いします



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
依存先のクラスに本物を使うのはユニットテスト? Yes / No

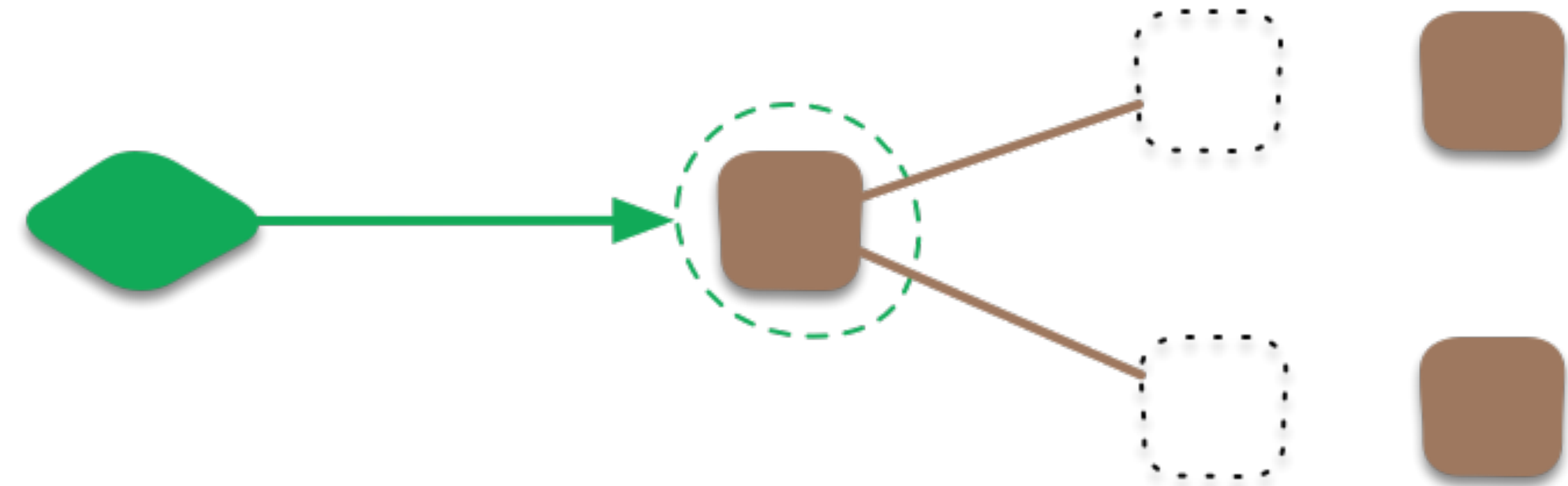
Unit Test の Unit って何？

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

Test Size: より曖昧さの少ない分類

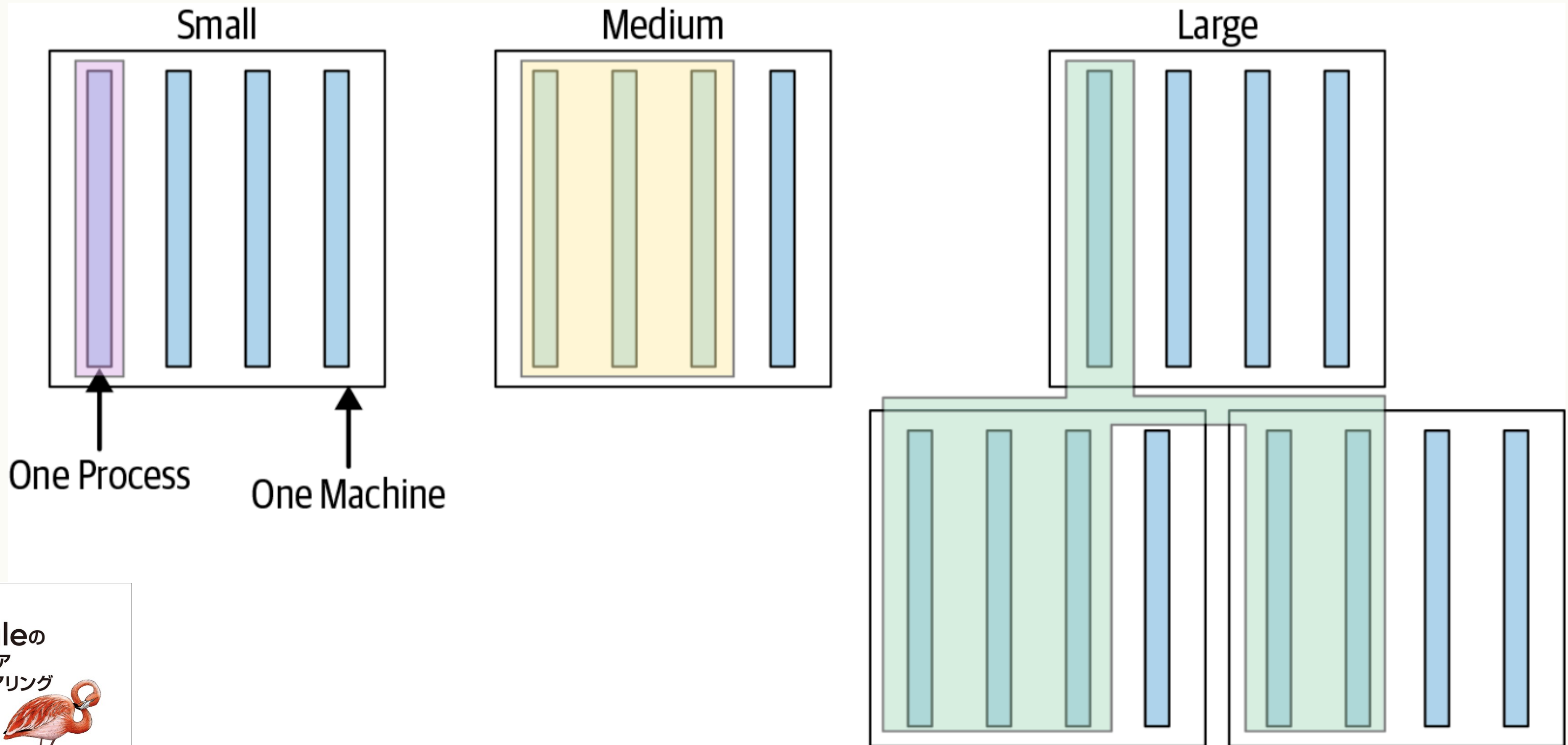


Figure 11-2. Test sizes



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+

Test Scope と Test Size

Test Size

	Small (単一プロセス)	Medium (単一マシン)	Large (制約なし)
Unit (比率 80%)	大いに推奨	避けたいが しかたないときも	最悪だが よく見かける
Integration (比率 15%)	書けるなら コスパ良し	普通	できれば 避けたい
E2E (比率 5%)	原理上不可能に近いが 小さいシステムなら可能?	小さいシステム なら可能	普通かつ CUJに絞りたい

Test Scope

Test Scope と Test Size

		Test Size		
		Small (単一プロセス)	Medium (単一マシン)	Large (制約なし)
Test Scope	Unit (比率 80%)	大いに推奨	コスパ悪し 避けたいが しかたないときも	最悪だが よく見かける
	Integration (比率 15%)	書けるなら コスパ良し	普通	できれば 避けたい
	E2E (比率 5%)	原理上不可能に近いが 小さいシステムなら可能?	コスパ良し 小さいシステム なら可能	普通かつ CUJに絞りたい



Agenda: 連載各回のテーマ

学習用テスト

偽陽性と偽陰性

テストサイズ



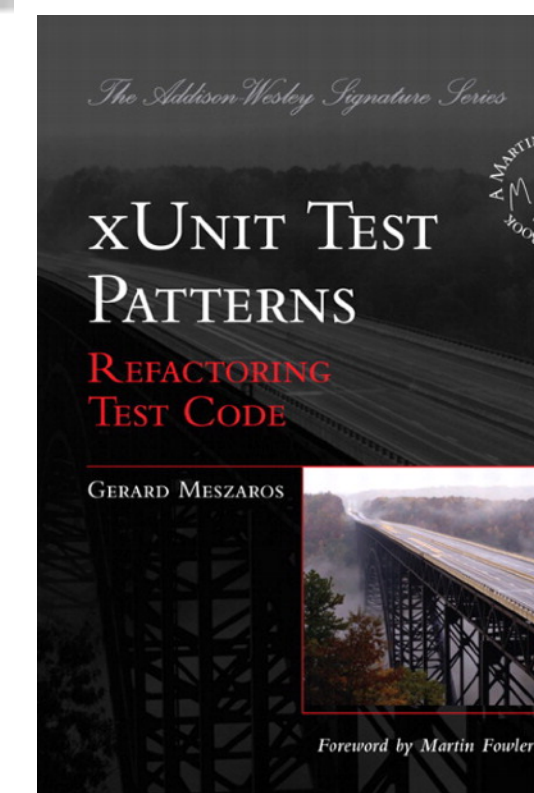
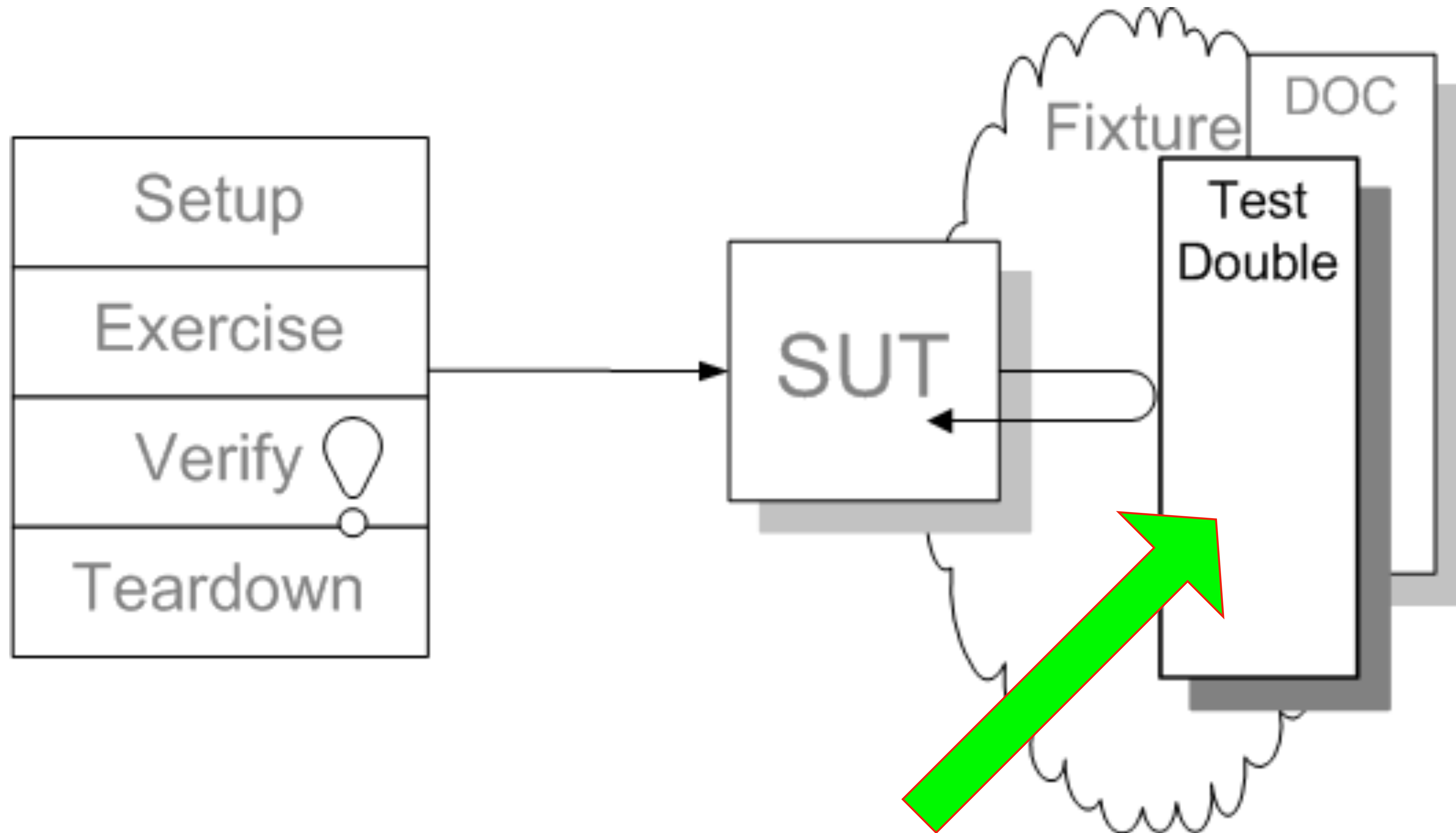
テストダブル

テストピラミッド

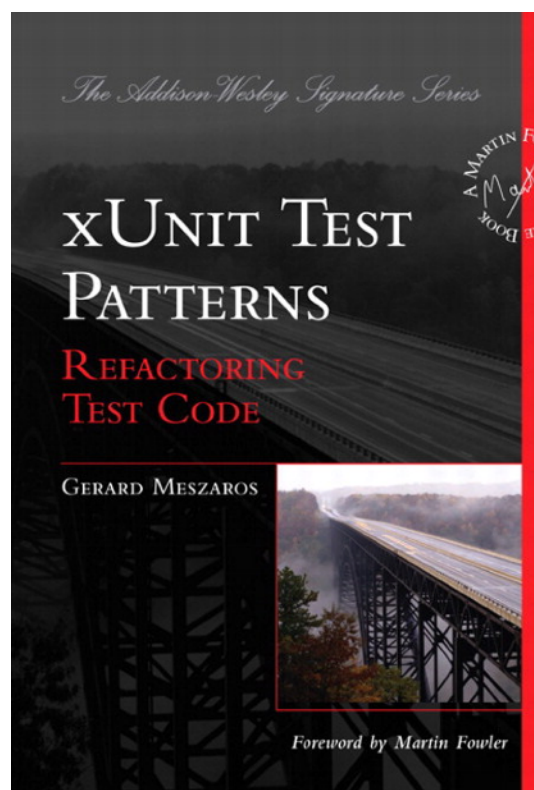
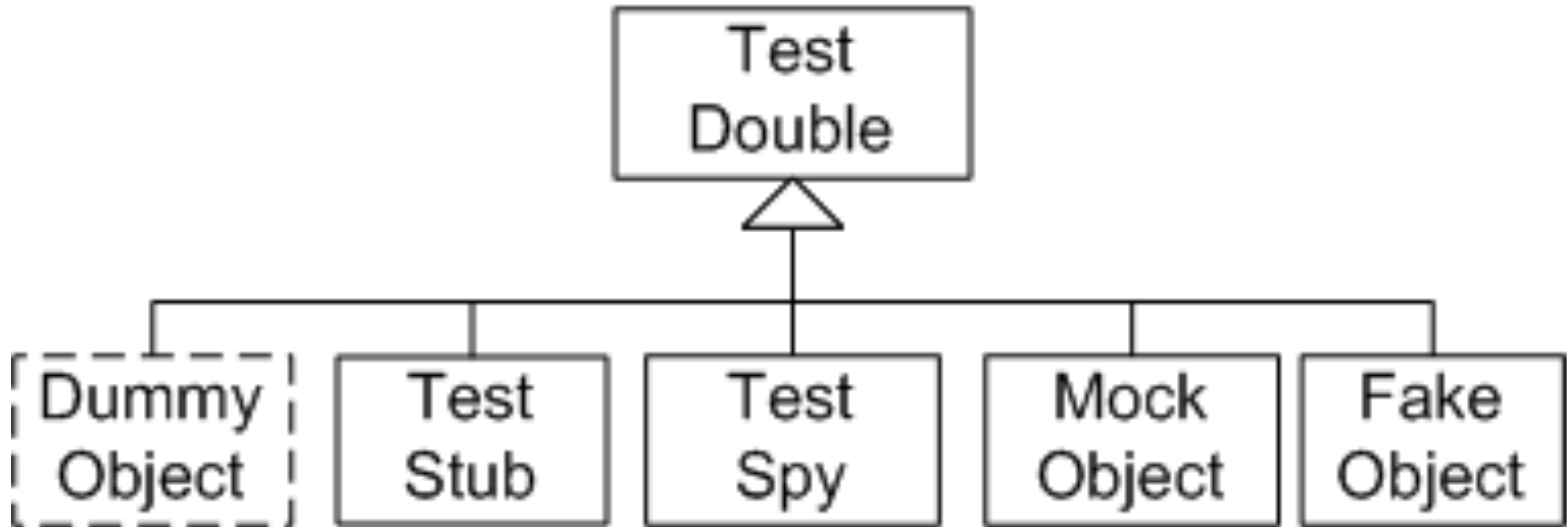
自動テストのサイズダウン戦略

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



テストダブルの分類 (xUTP)



- 利点

- 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

- テストが脆くなり、変更を妨げる（利点の裏返し）
- テストの偽陰性を招く（自作自演テスト）



Agenda: 連載各回のテーマ

学習用テスト

偽陽性と偽陰性

テストサイズ

テストダブル

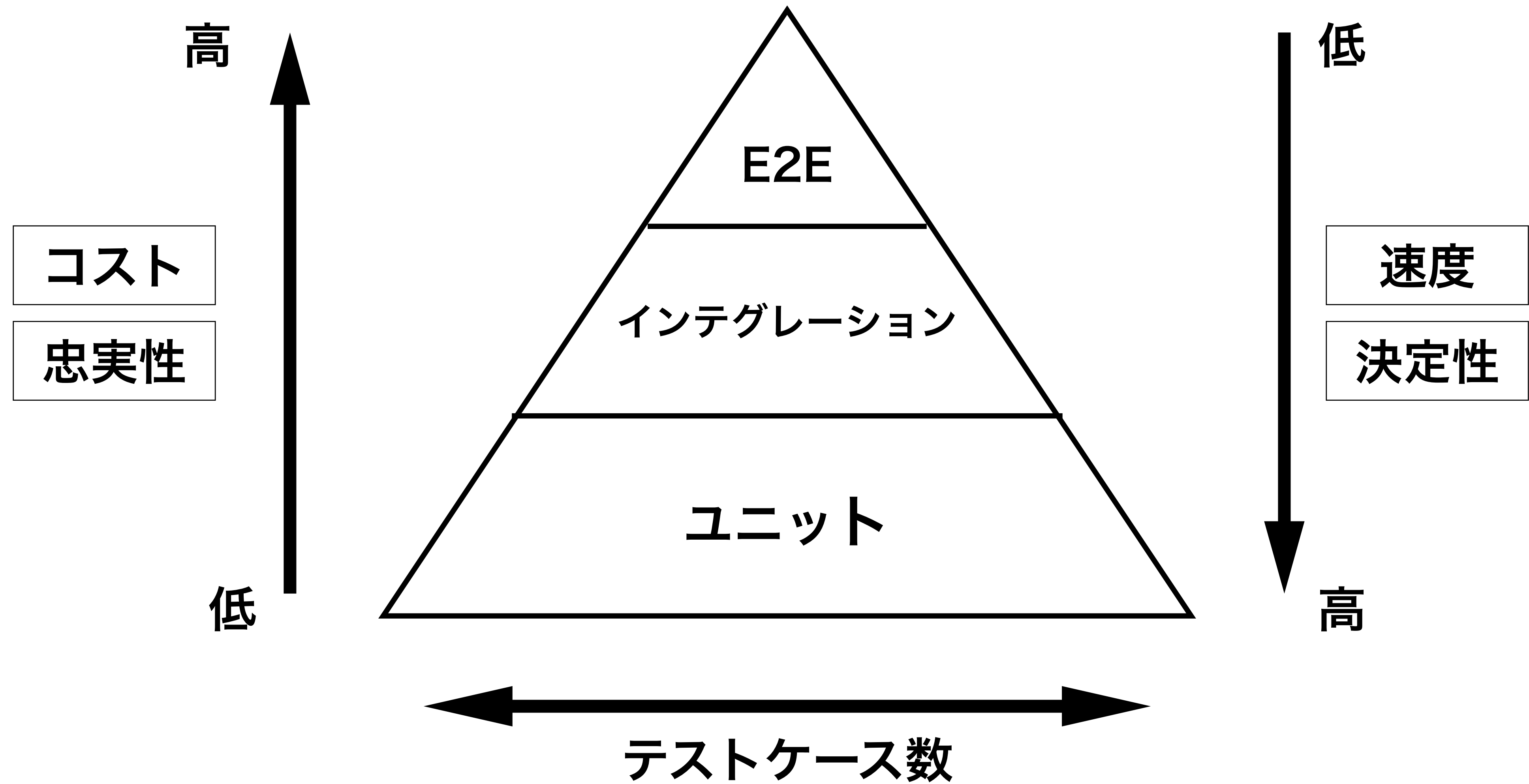


テストピラミッド

自動テストのサイズダウン戦略

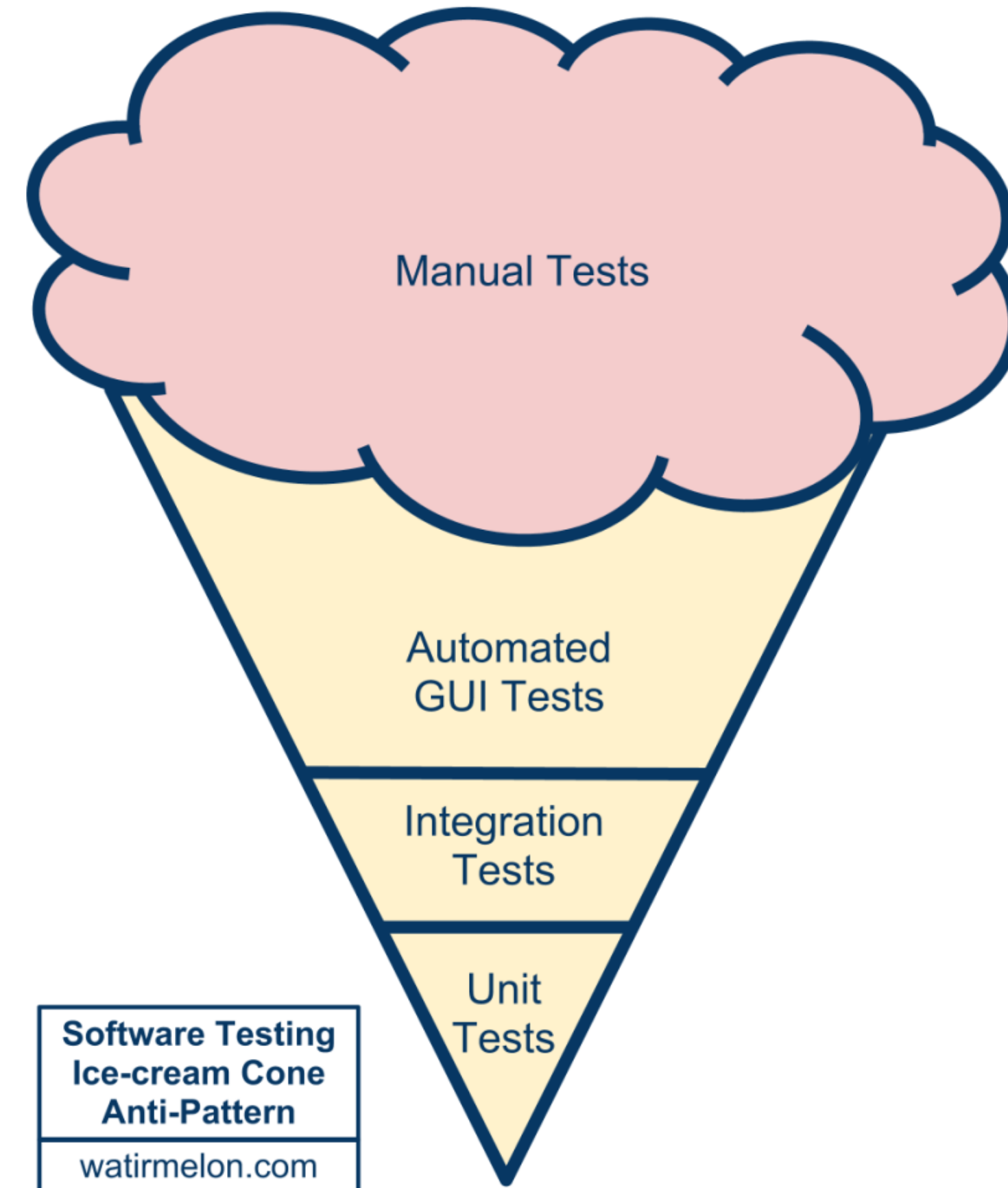
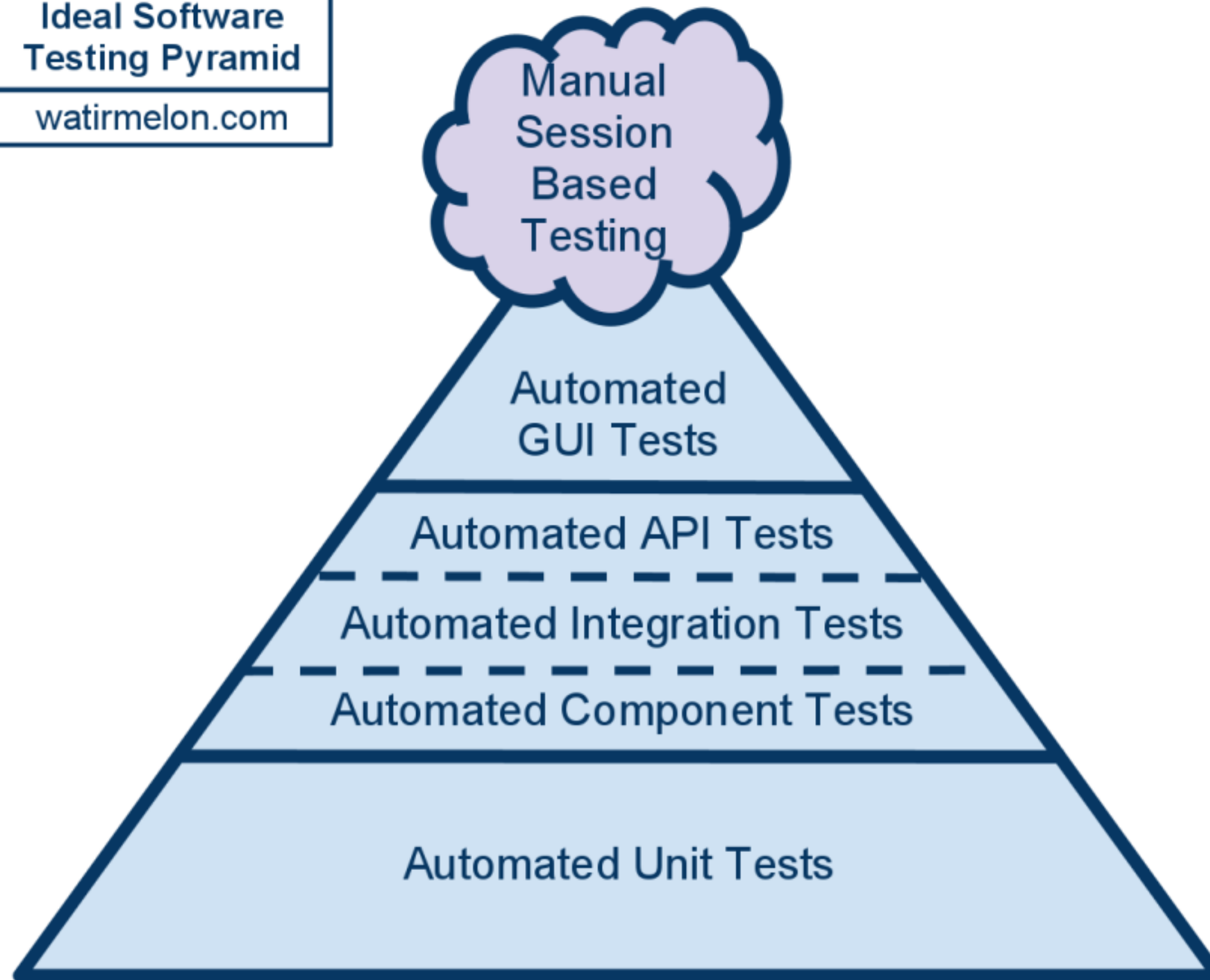
**自動テストの信頼性を
中長期的に保つ
最適なバランス**

テストピラミッド



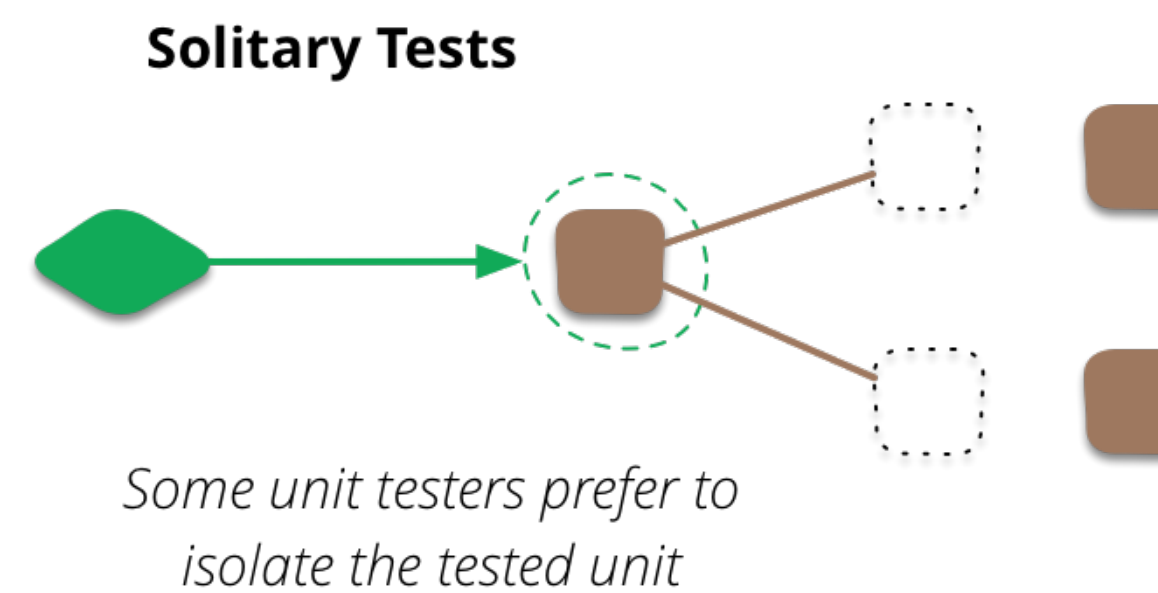
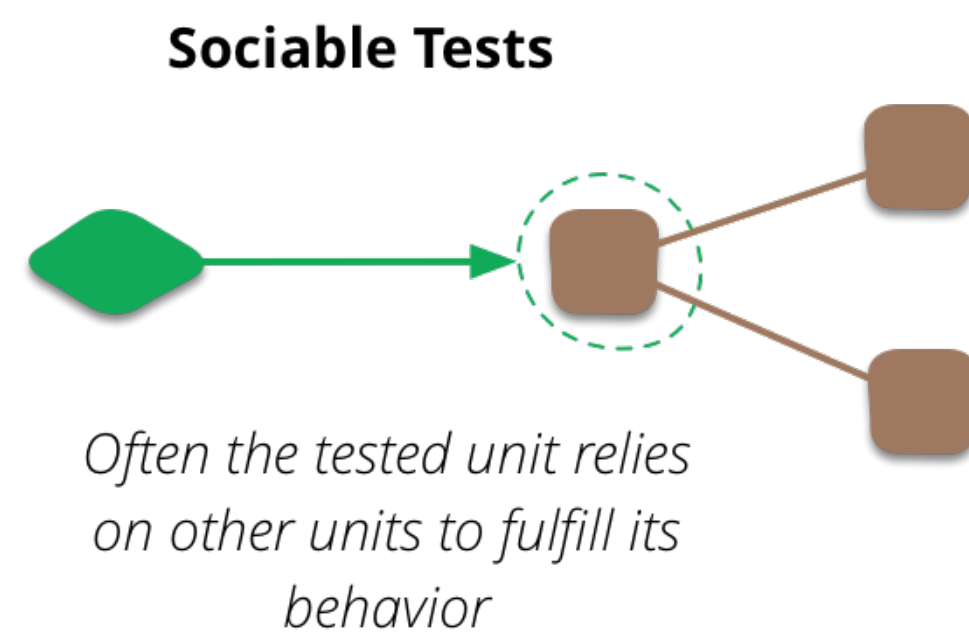
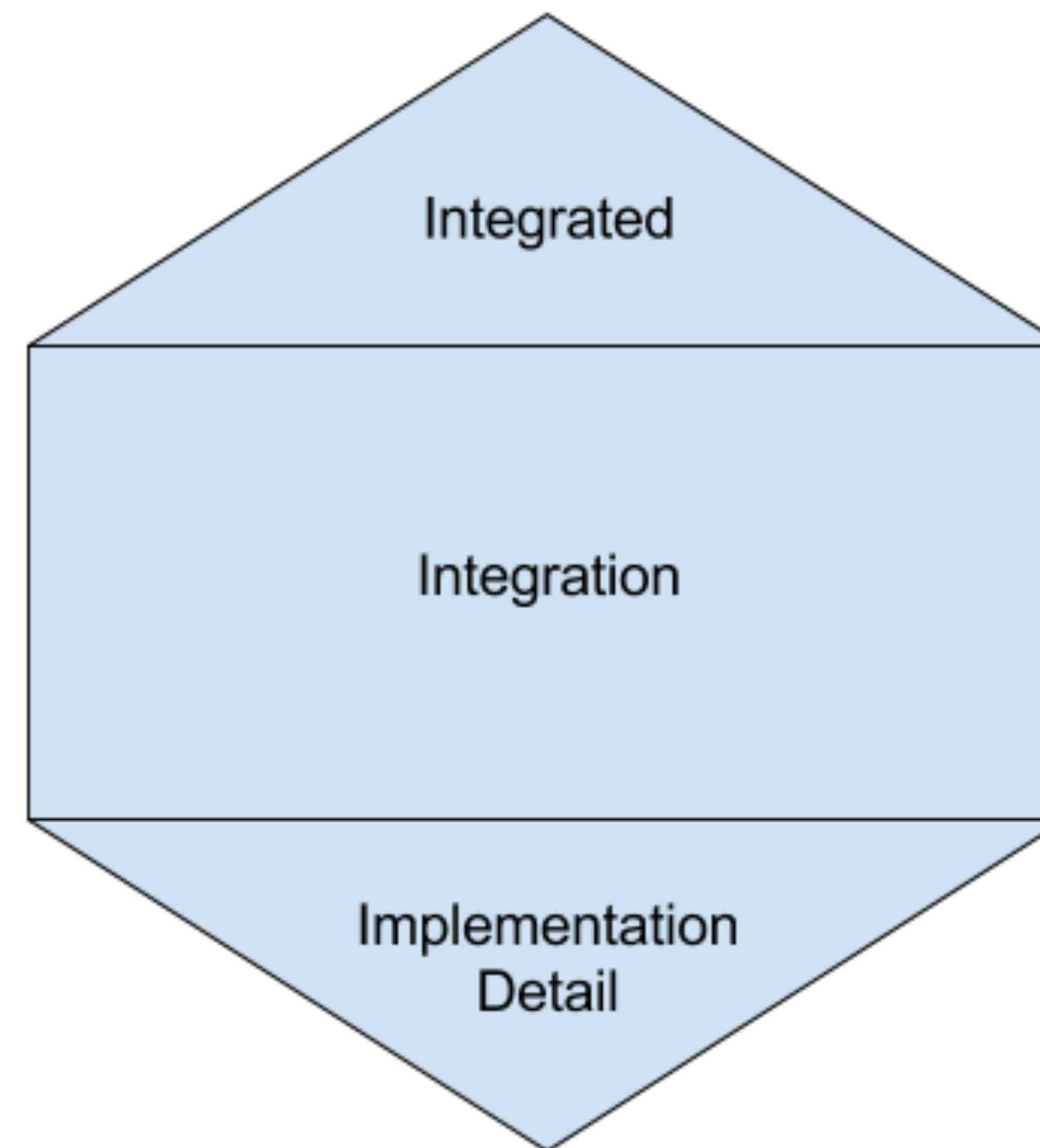
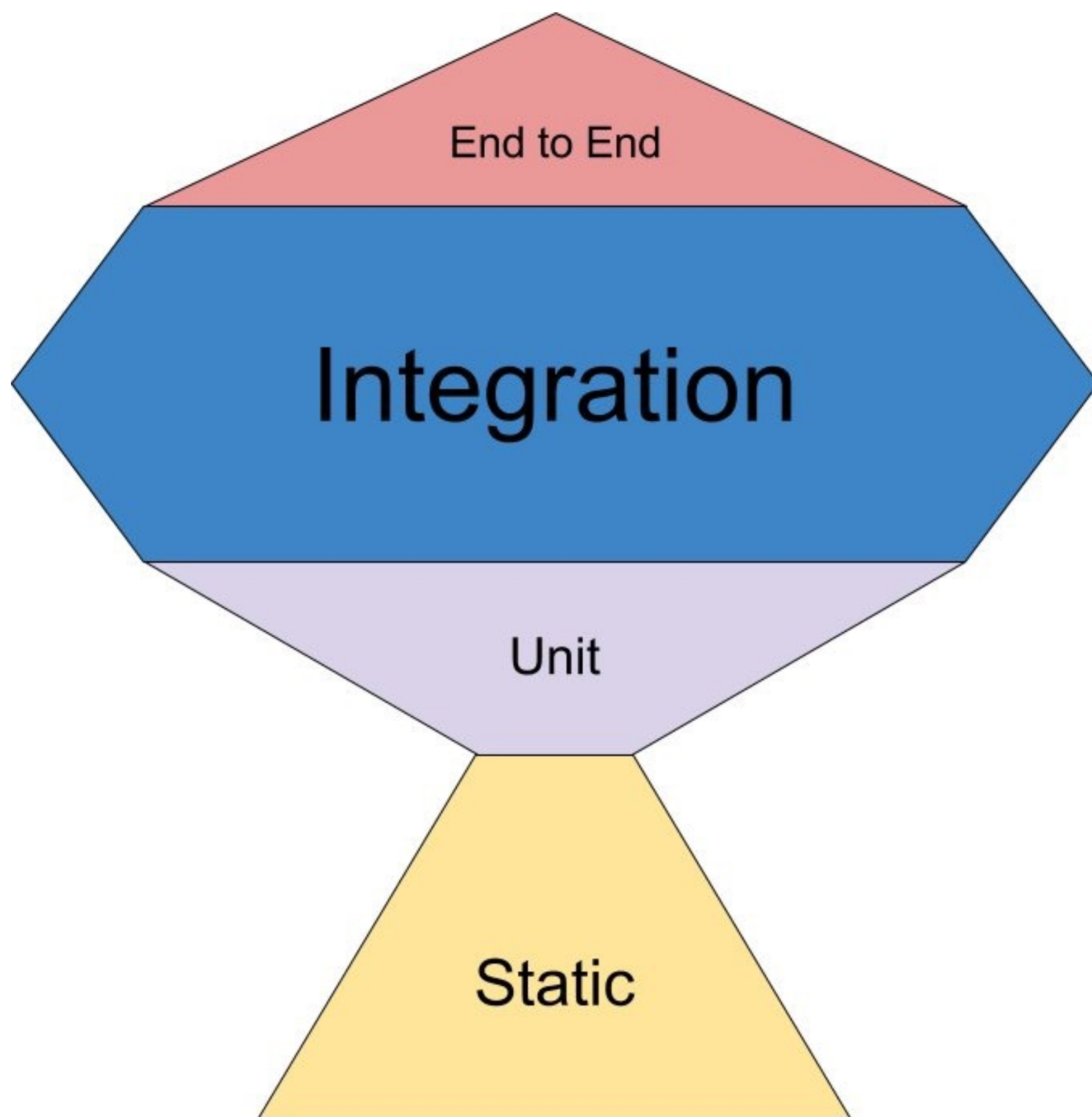
テストピラミッドとアイスクリームコーンアンチパターン

Ideal Software Testing Pyramid
watirmelon.com

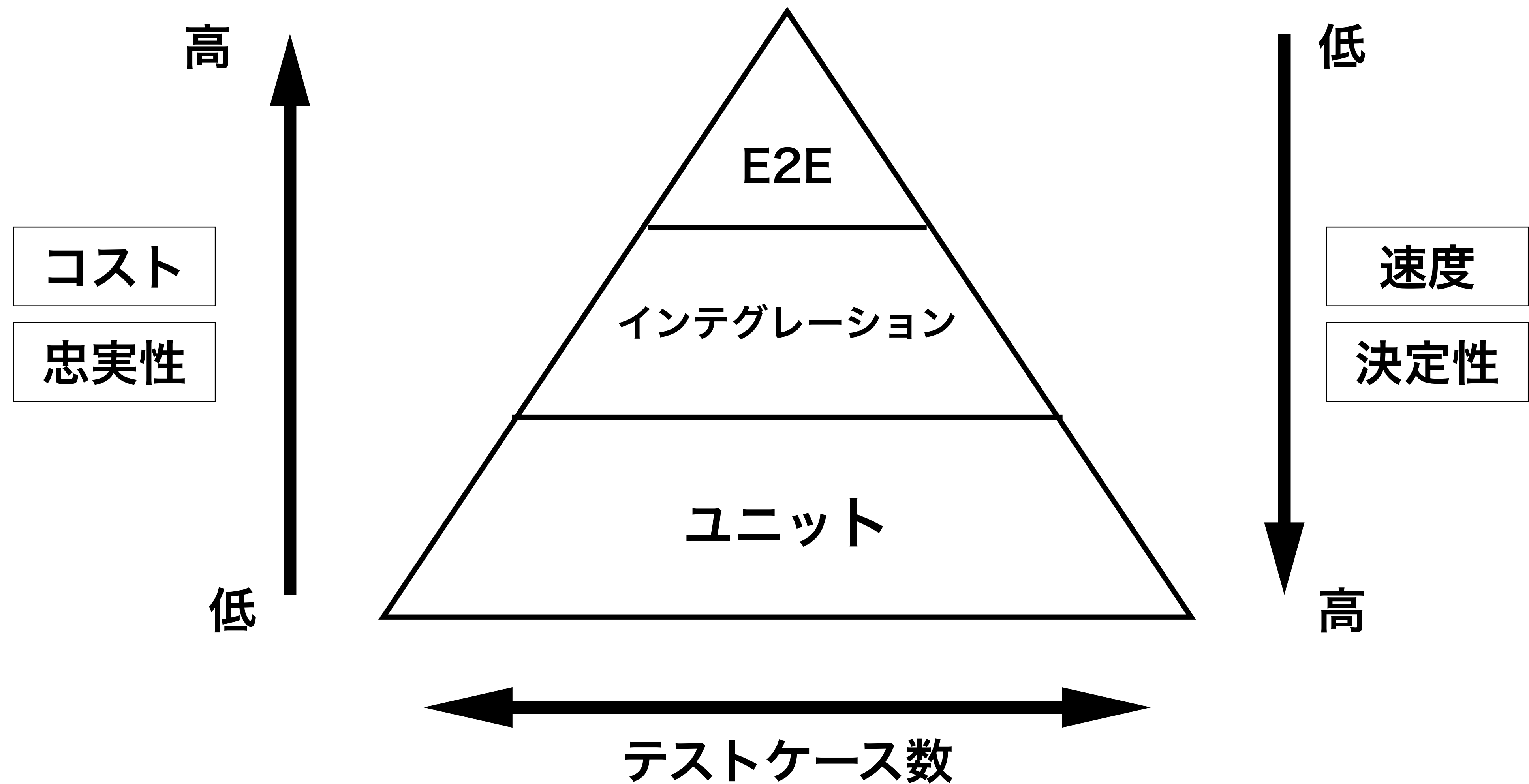


Software Testing Ice-cream Cone Anti-Pattern
watirmelon.com

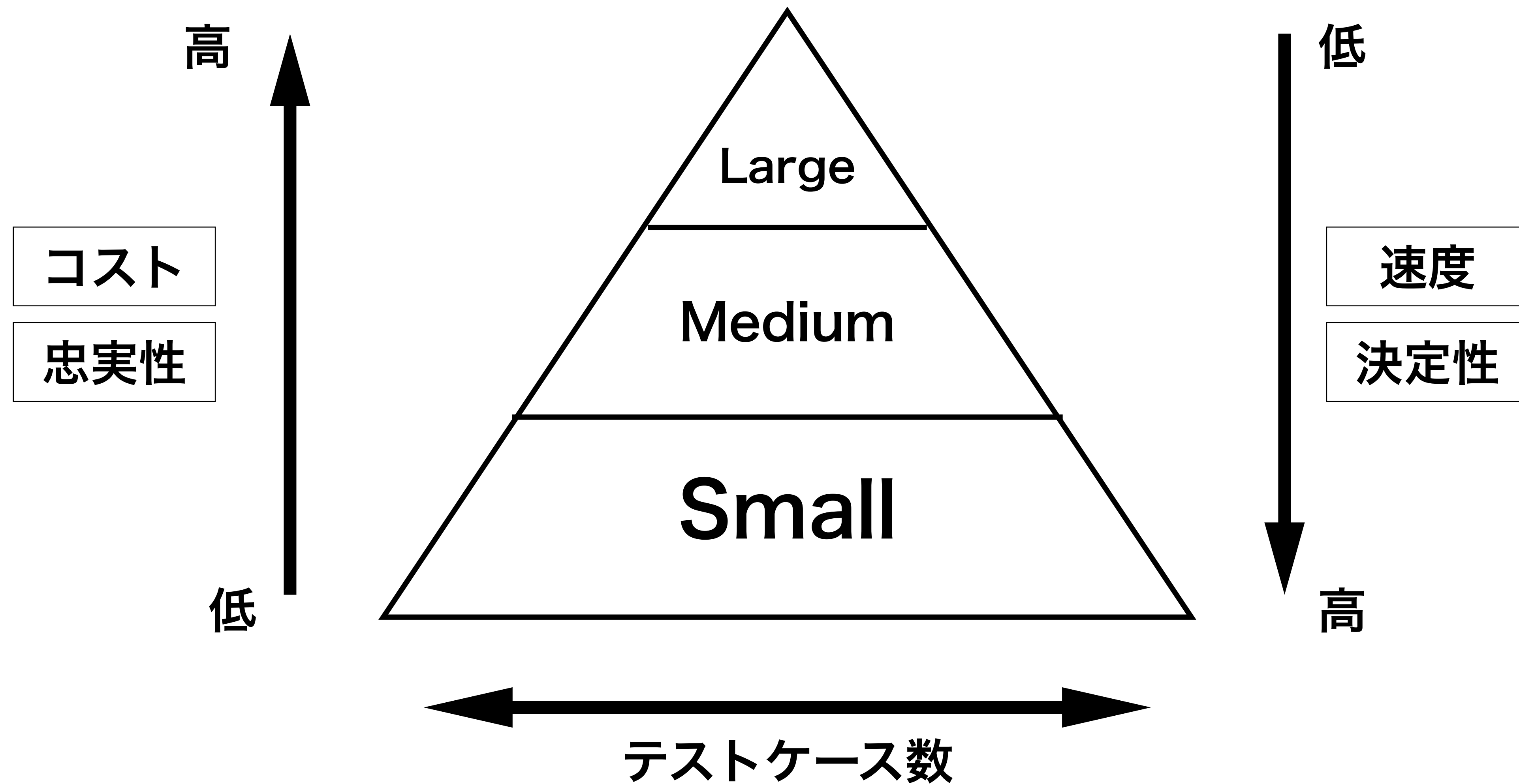
混乱は「ユニット」「インテグレーション」の解釈のブレから生じがち



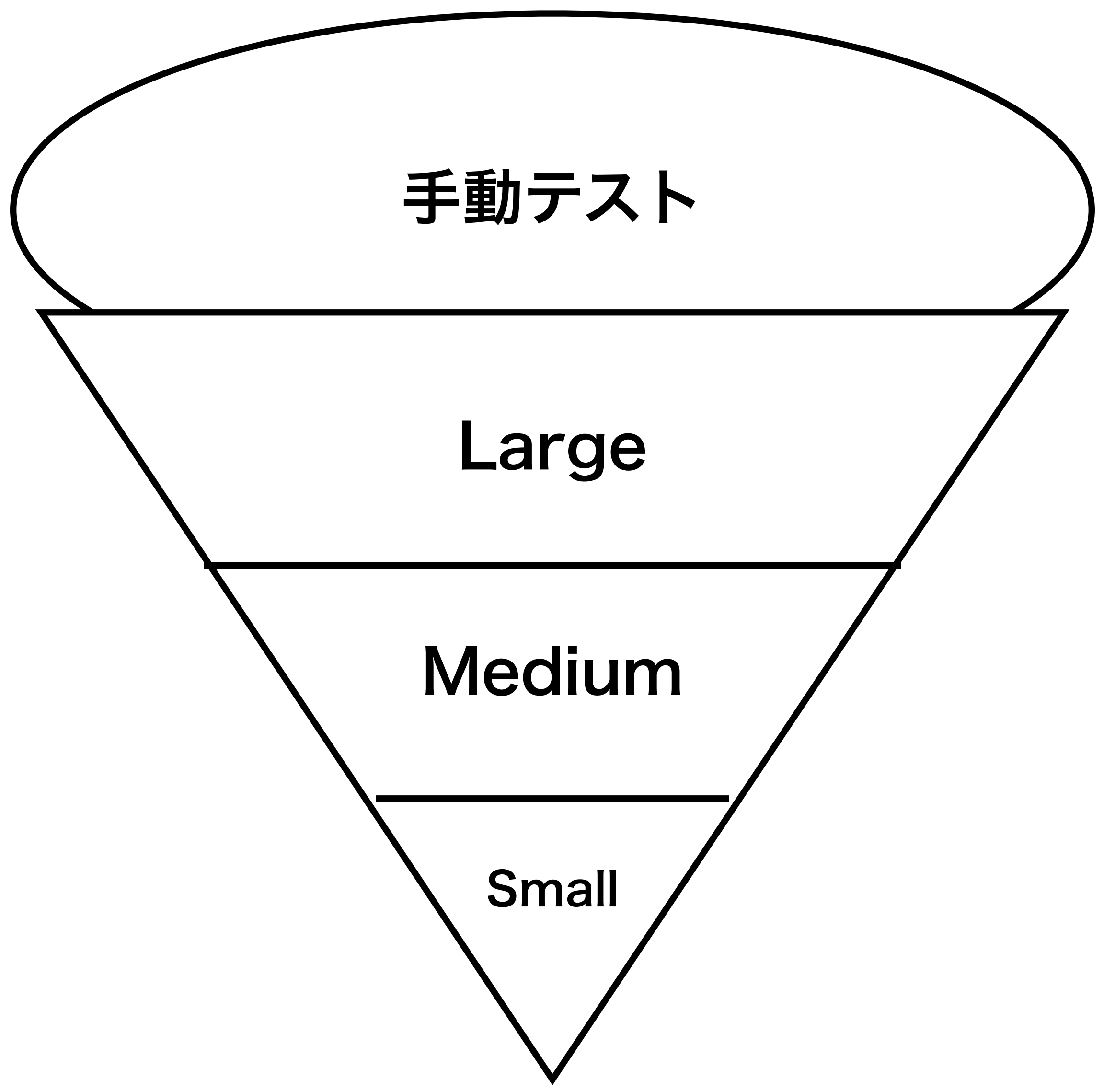
ブレの少ないテストの分類基準は……？



ブレの少ないテストの分類基準 → テストサイズ



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）





Agenda: 連載各回のテーマ

学習用テスト

偽陽性と偽陰性

テストサイズ

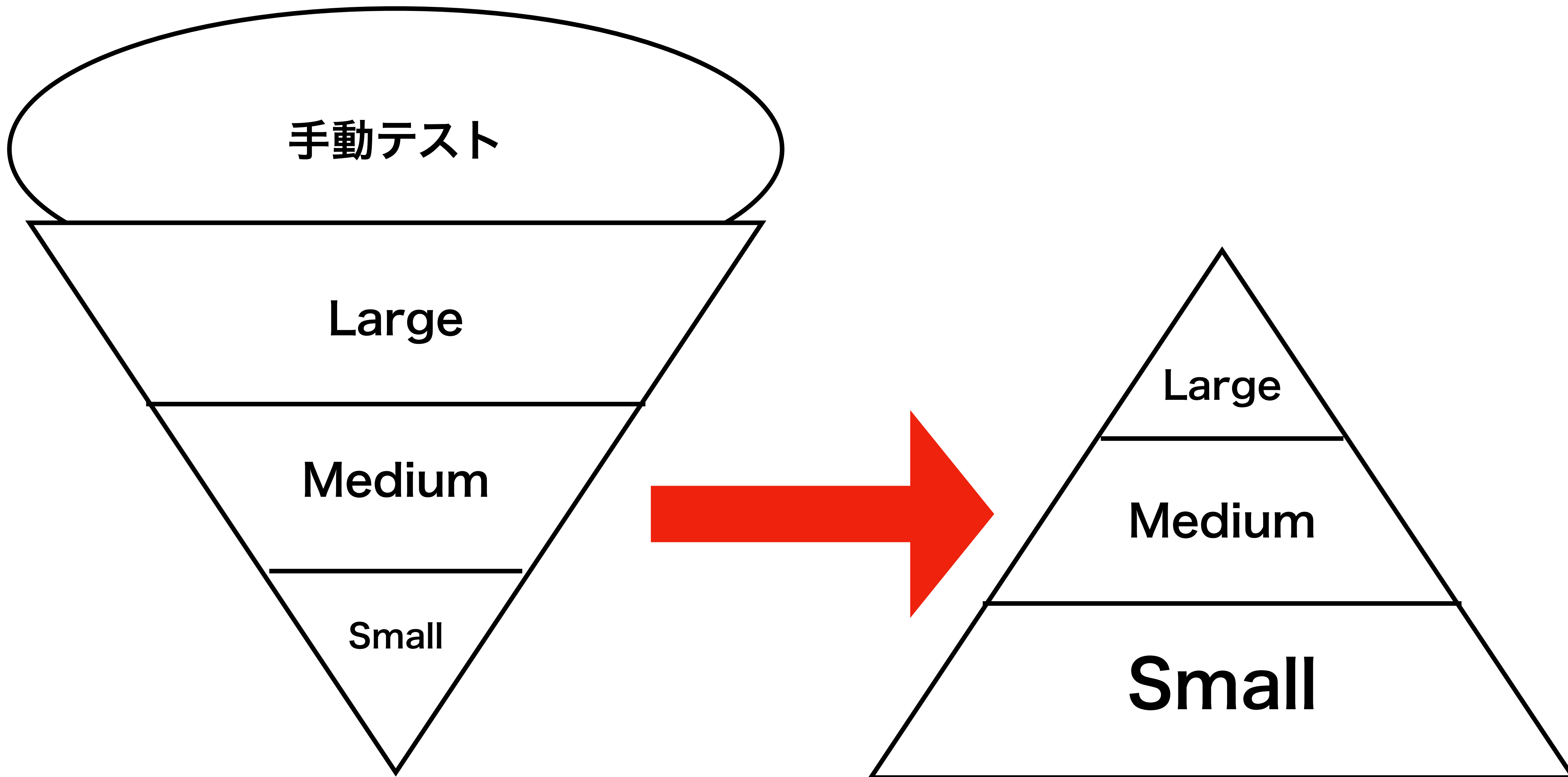
テストダブル

テストピラミッド

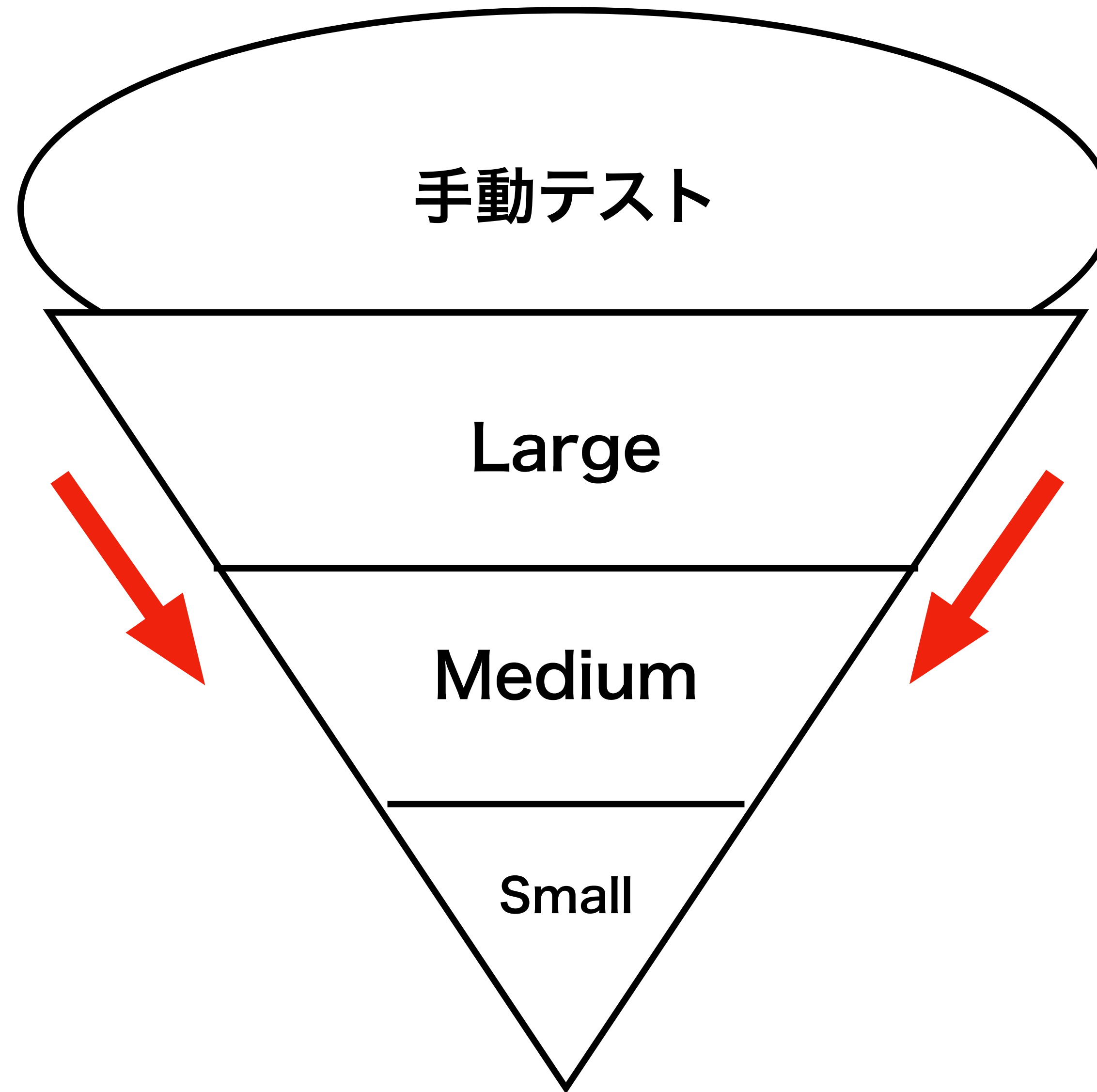


自動テストのサイズダウン戦略

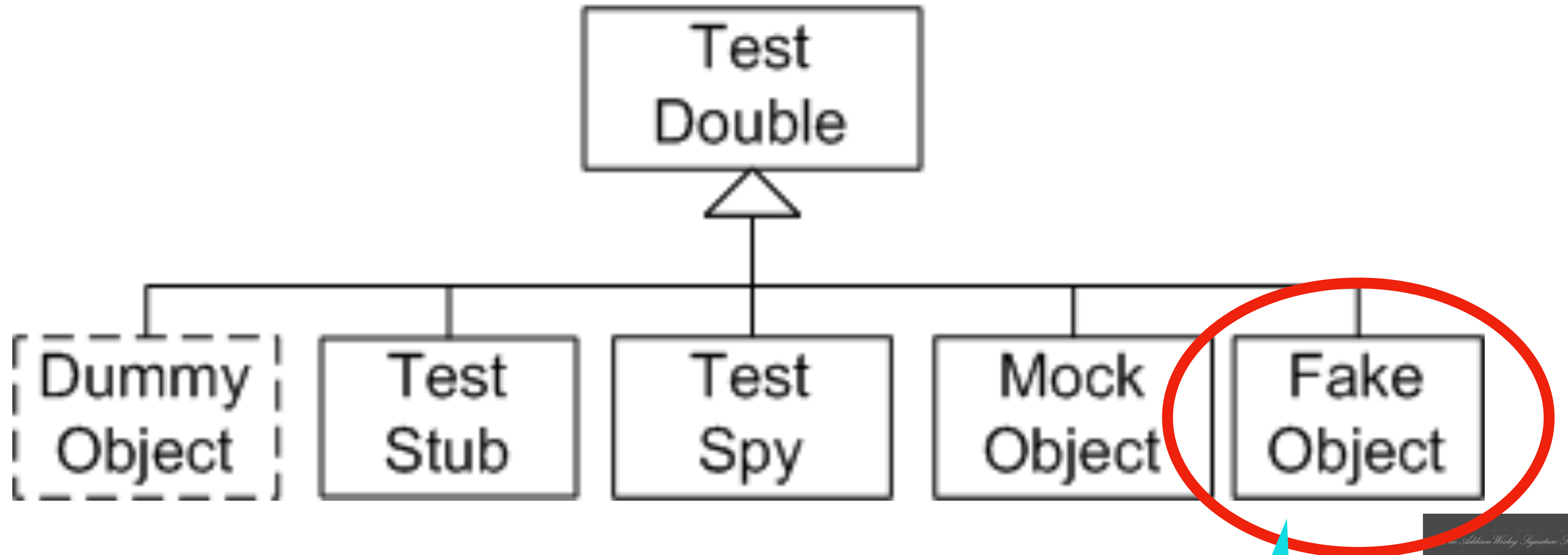
どうやってアイスクリームコーンをピラミッドにするか



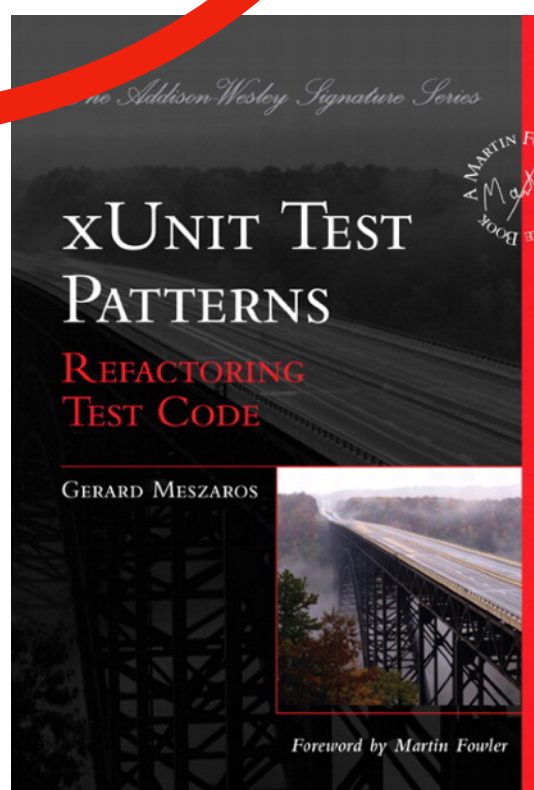
Large から Medium へ

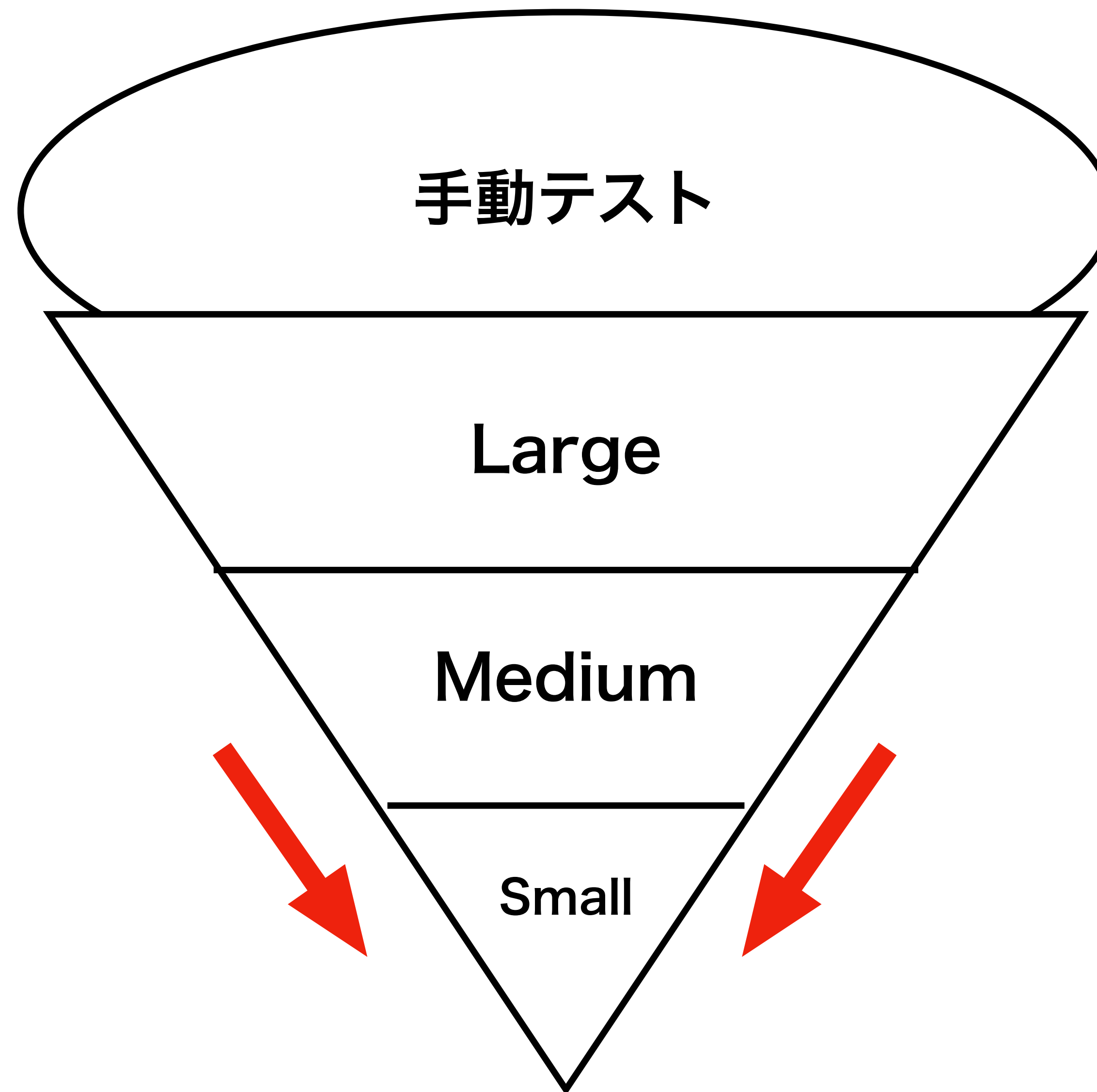


Large から Medium へ: Fake Object



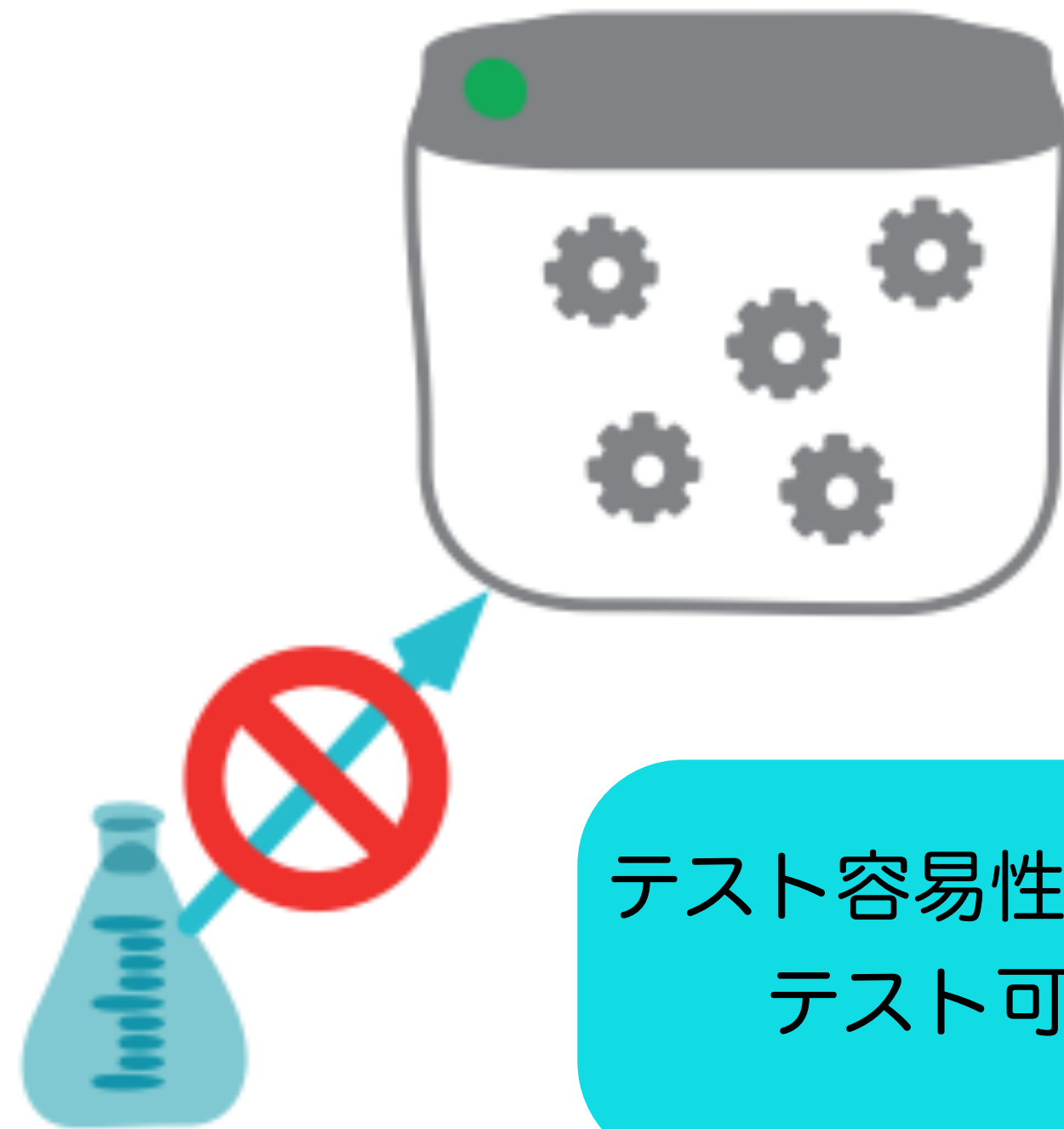
Fake は自動テスト用の代替実装。
DynamoDB に対する DynamoDB local や localstack が代表例。
コンテナで動作させられればテストサイズが Medium に収まる。



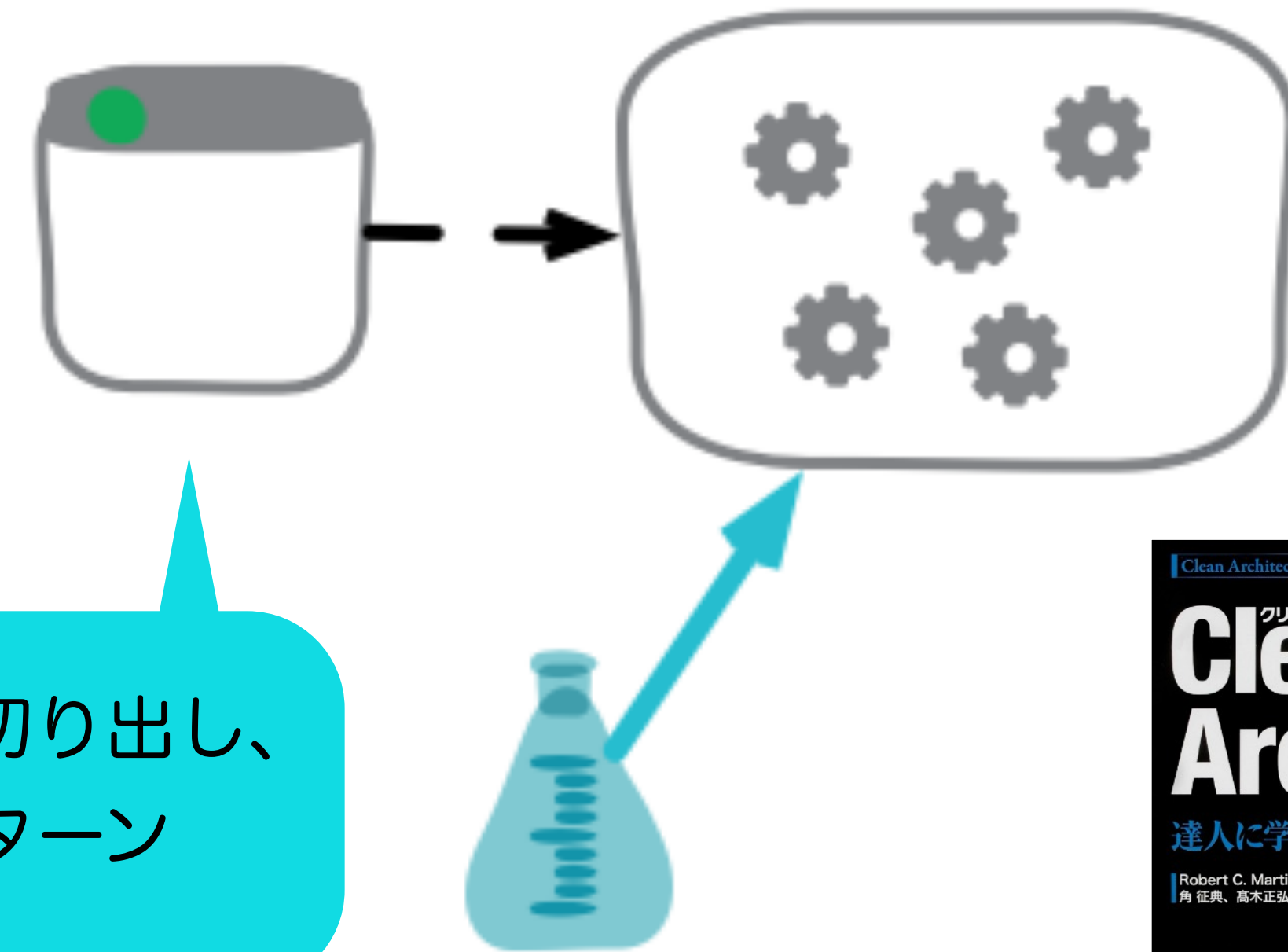


Medium から Small へ: Humble Object

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



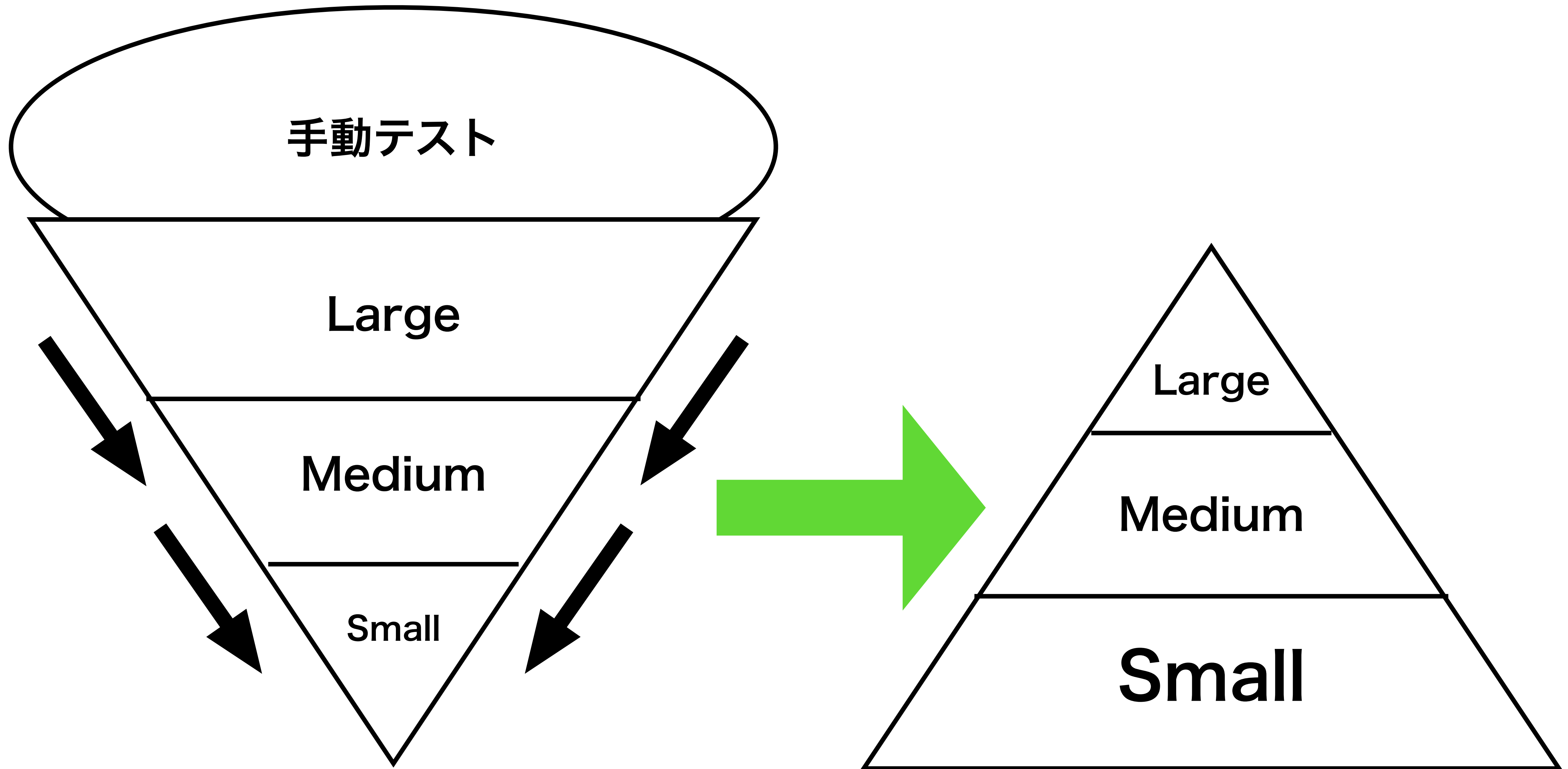
*move the logic into a separate
element that is testable, making
the original object **humble***



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



アイスクリームコーンからピラミッドへ





これまでの知見をまとめると

学習用テスト



偽陽性と偽陰性



テストサイズ



テストダブル

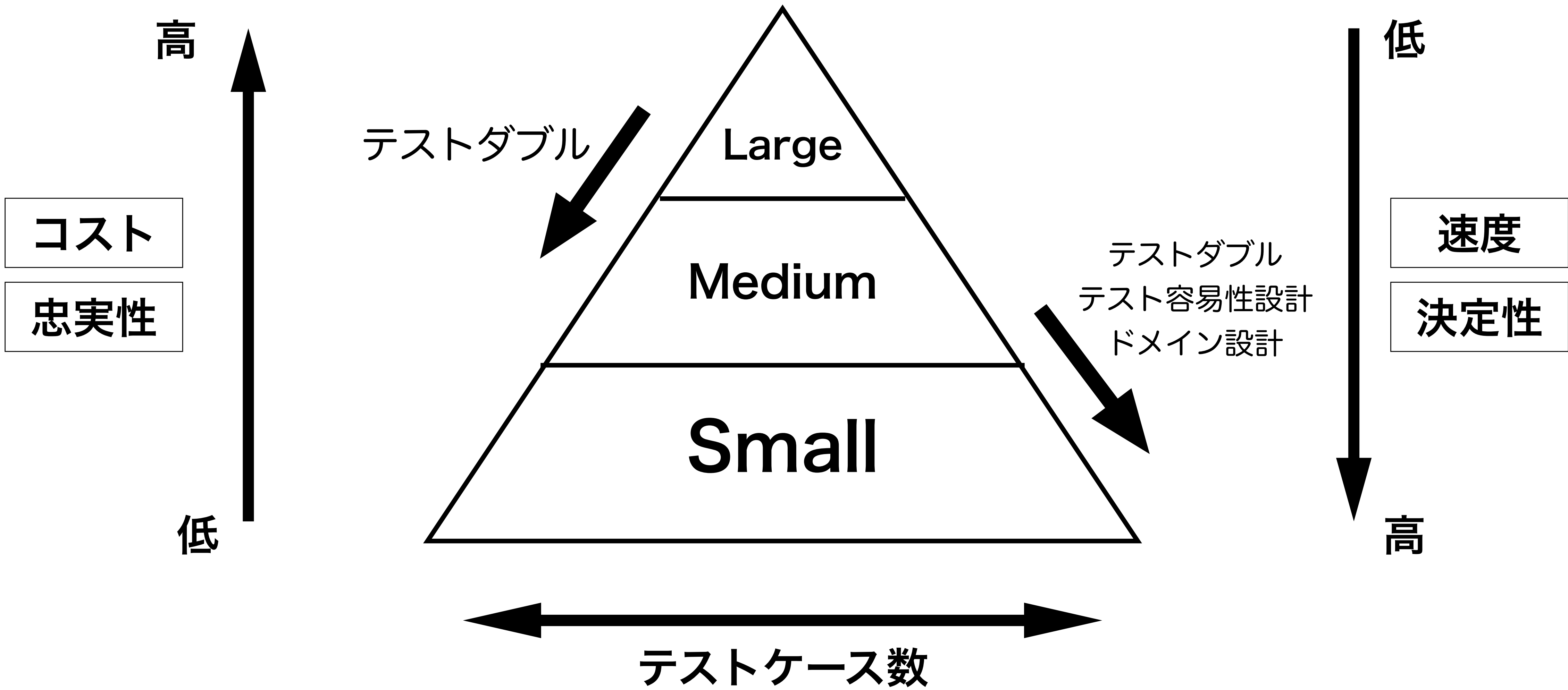


テストピラミッド



自動テストのサイズダウン戦略

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



おわりに

単体テストの 考え方/使い方



Vladimir Khorikov (著)
須田智之 (訳)

Unit Testing

Principles, Practices, and Patterns



質の高いテストを行い、
ソフトウェアに価値を
もたらそう！

- ・プロジェクトの持続可能な成長を実現するための戦略について
- ・単体テストの原則・実践とそのパターン
- ・優れたテストを実践しソフトウェアの品質改善に役立てよう

HANNING

マイナビ

失職の危機2: いかにも私の言いそうなことを言うAI



実行順序に依存するユニットテストについて、t_wadaさんなら何というのでしょうか



t_wadaさんによると、実行順序に依存するユニットテストは避けるべきである可能性があります。これは、実行順序に依存するテストは、それらを実行する順序によって結果が異なるため、結果が不確定になり、信頼性が低くなる可能性があるからです。また、実行順序に依存するテストは、コードの修正や拡張によって、テストが失敗する可能性があり、維持するのが困難になるからです。これはt_wadaさん個人の意見であり、専門家によっては異なる意見を持つこともあるでしょう。

悲しいお知らせ

『WEB+DB PRESS』休刊のお知らせ

ツイート

リスト

B!725

いいね! 947

2023年5月1日

この記事を読むのに必要な時間：およそ 0.5 分

WEB+DB PRESSは、2023年8月発売のVol.136をもって隔月刊誌としては休刊させていただきます。物価上昇による制作費の高騰など諸般の事情により、今回の決定に至った次第です。

突然の休刊案内にてたいへん恐縮ではございますが、何卒ご理解を賜りますよう、お願い申し上げます。

22年以上の長きにわたり、絶大なご支援をいただきましたことを、厚く御礼申し上げます。

弊誌で扱っていた分野のコンテンツは、今後も弊社刊行のSoftware Designやgihyo.jp、書籍などで提供させていただきます。また、必要な場合には「特別号」の編集・刊行なども検討してまいります。

ご清聴ありがとうございました

