

TDD実践講座

CodeZineAcademy

2017.11.9

メイン講師 安井力
サポート 和田卓人



Welcome
to
TDBC!

TDDBCとは

講演
TDDとペアプロ
レビュー大会
ふりかえり

TDDについての講演



ペアプログラミング : Pair Programming



コードレビュー大会



ふりかえり



TDD テスト駆動開発 のこころ



A circular portrait of a man with short dark hair, a beard, and mustache, wearing a light blue button-down shirt under a grey blazer. He is smiling slightly.

安井 力 / やっとむ

twitter:@yattom <https://www.facebook.com/yattom>

プログラマー

Java Python Ruby JavaScript
テスト駆動開発

アジャイルコーチ

ワークショップ 現場導入 技術支援

コンサルタント

モデリング ユーザーストーリー



和田 卓人

id: t-wada

@t_wada

github: twada



質問:TDDをやろうと
思ったのはなぜですか?



t-wada said...

「TDDの
眞の目的は
健康」





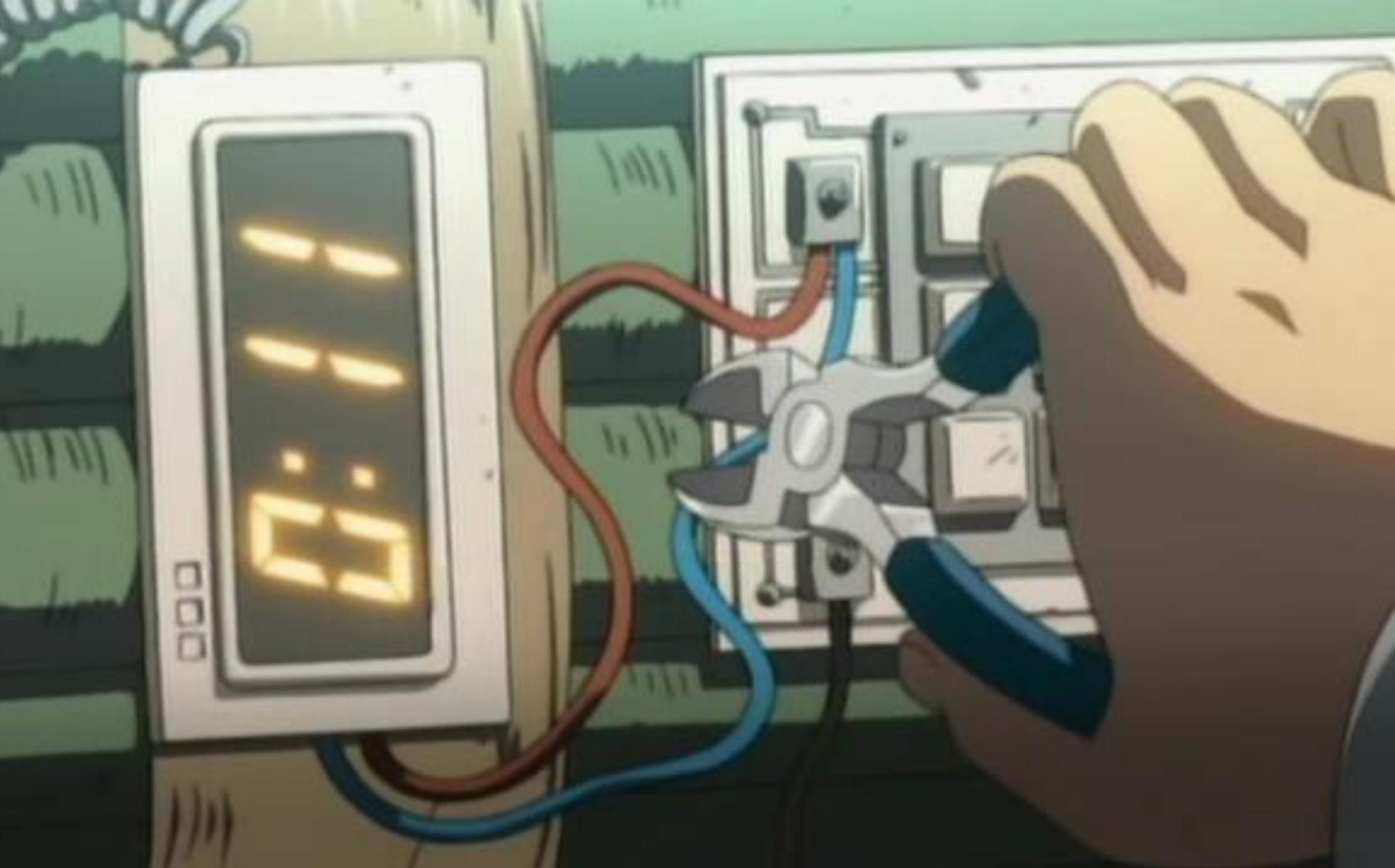
現場の困りごと

荒みきったコード



疲弊しきった現場





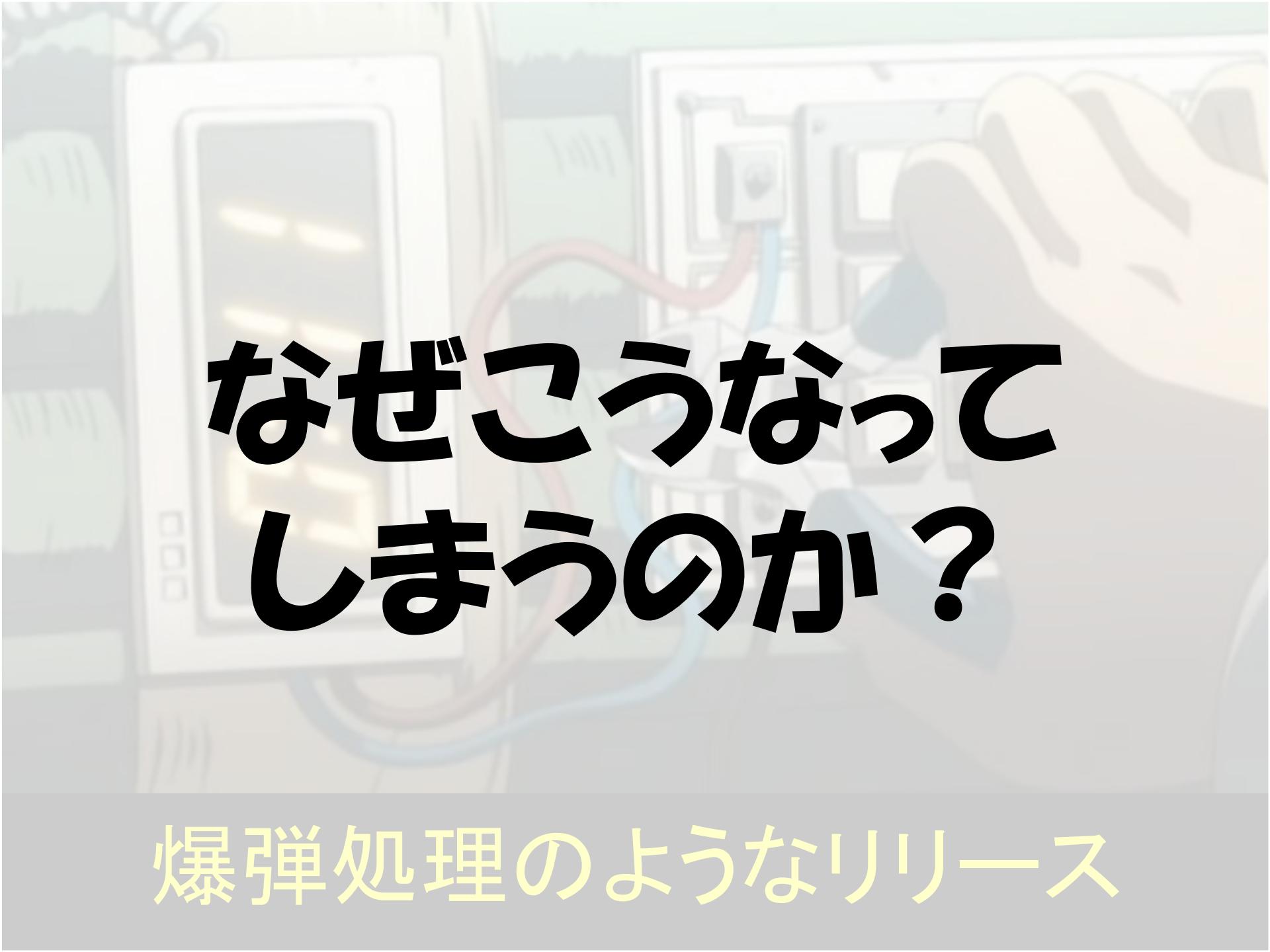
爆弾処理のようなリリース

荒みきったコード

これが当たり前？

疲弊しきった現場

そんなハズはない！



なぜこうなって
しまうのか？

爆弾処理のようなリリース



複雑さに対する恐れと不安

不安を克服する技術の不足



- ・コードが複雑でなにをしているかわからない
- ・コードを変えたら思わぬ影響が出た
- ・使われてなさそうなんだけど自信がない
- ・自信はあったのに、壊れてしまった
- ・何を頼りにすればいいか分からぬ
- ・半日ができるはずがもう二日かかっている
- ・アドバイスを受けたら間違っていた
- ・人にお願いしたいんだけど頼んで平気かな
- ・頼んだら一週間経ってもできてこない
- ・テスト不十分なのにリリースが迫ってきた
- ・リリース間際に修正したけどテストできない
- ・テストせずにリリースしたら(以下略)



<https://www.flickr.com/photos/oococha/3052091706/>



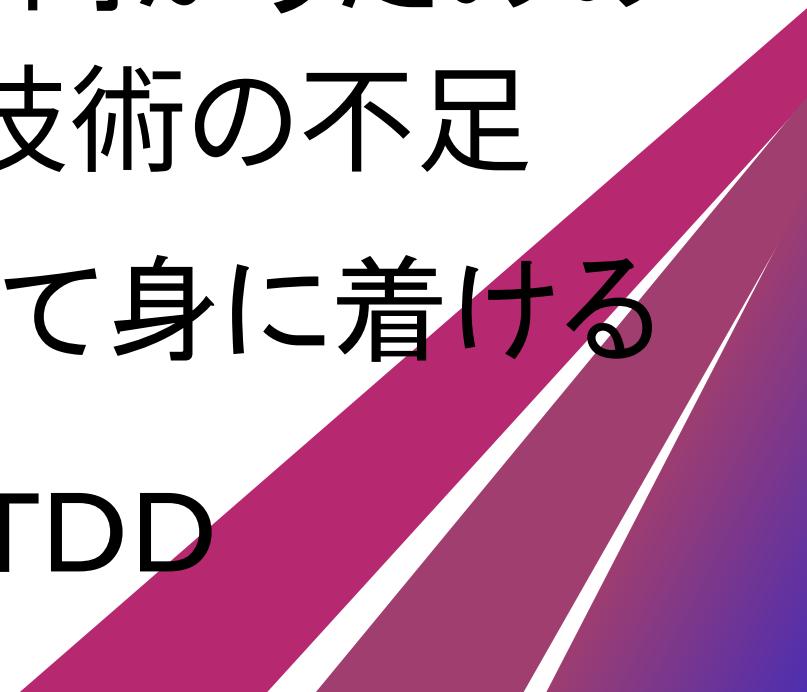
複雑さに対する恐れと不安



不安を克服する技術の不足



複雑さに対する恐れと不安
に立ち向かうための
不安を克服する技術の不足
を補って身に着ける
手法としてのTDD



コンプレックスとコンプリケイト



複雑 – Complex

- 全体的
- 包括的に対応する

ややこしい – Complicated

- 部分的
 - 要素分解して対応できる
- 

TDDとは

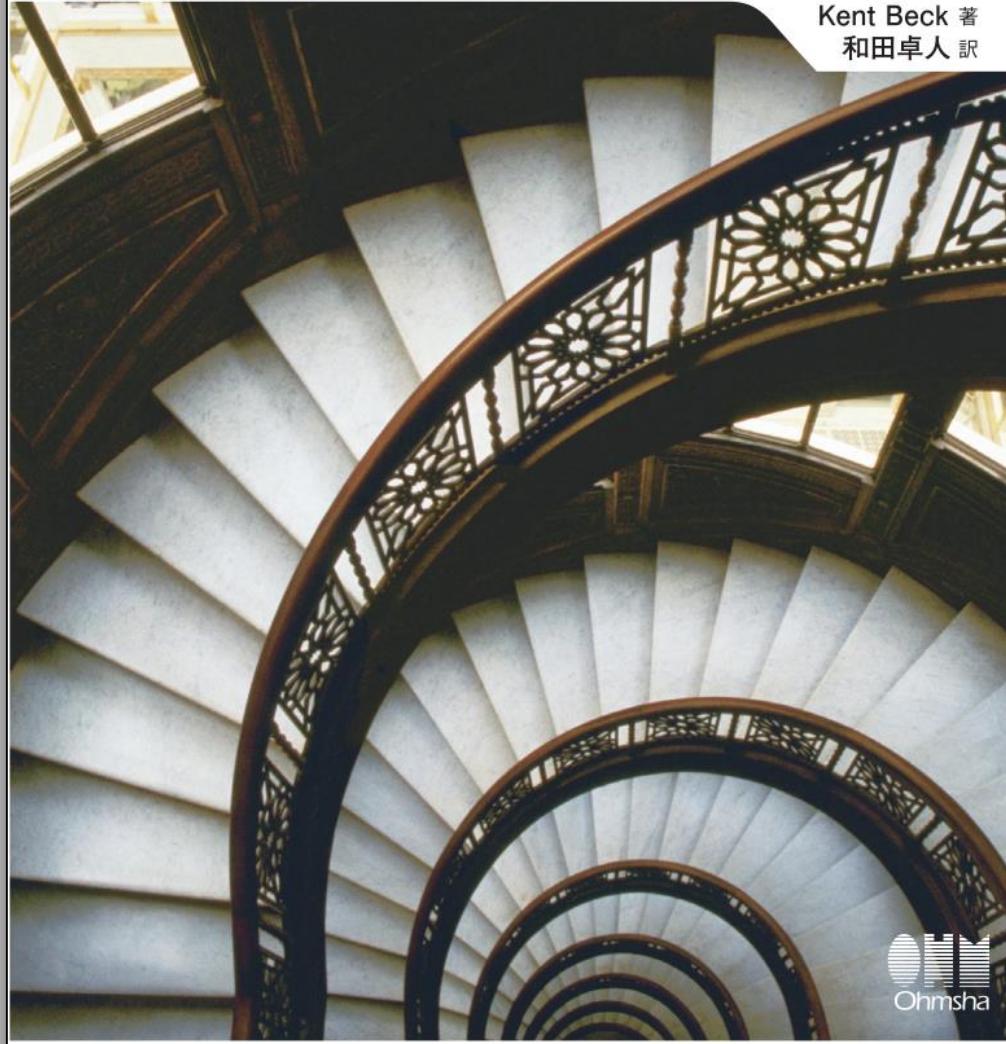




Test-Driven Development
By Example

テスト駆動開発

Kent Beck 著
和田卓人 訳



OHM
Ohmsha

Preface

Clean code that works, in Ron Jeffries' pithy phrase, is the Development (TDD). Clean code that works is a worthw bunch of reasons.

- It is a predictable way to develop. You know wh out having to worry about a long bug trail.

動作する

きれいなコード

「動作するきれいなコード」、ロン・ジェフリーズのこの簡潔な言葉が、テスト駆動開発(TDD)のゴールだ。動作するきれいなコードはあらゆる意味で価値がある。

— Kent Beck



動作するきれいなコードとは？



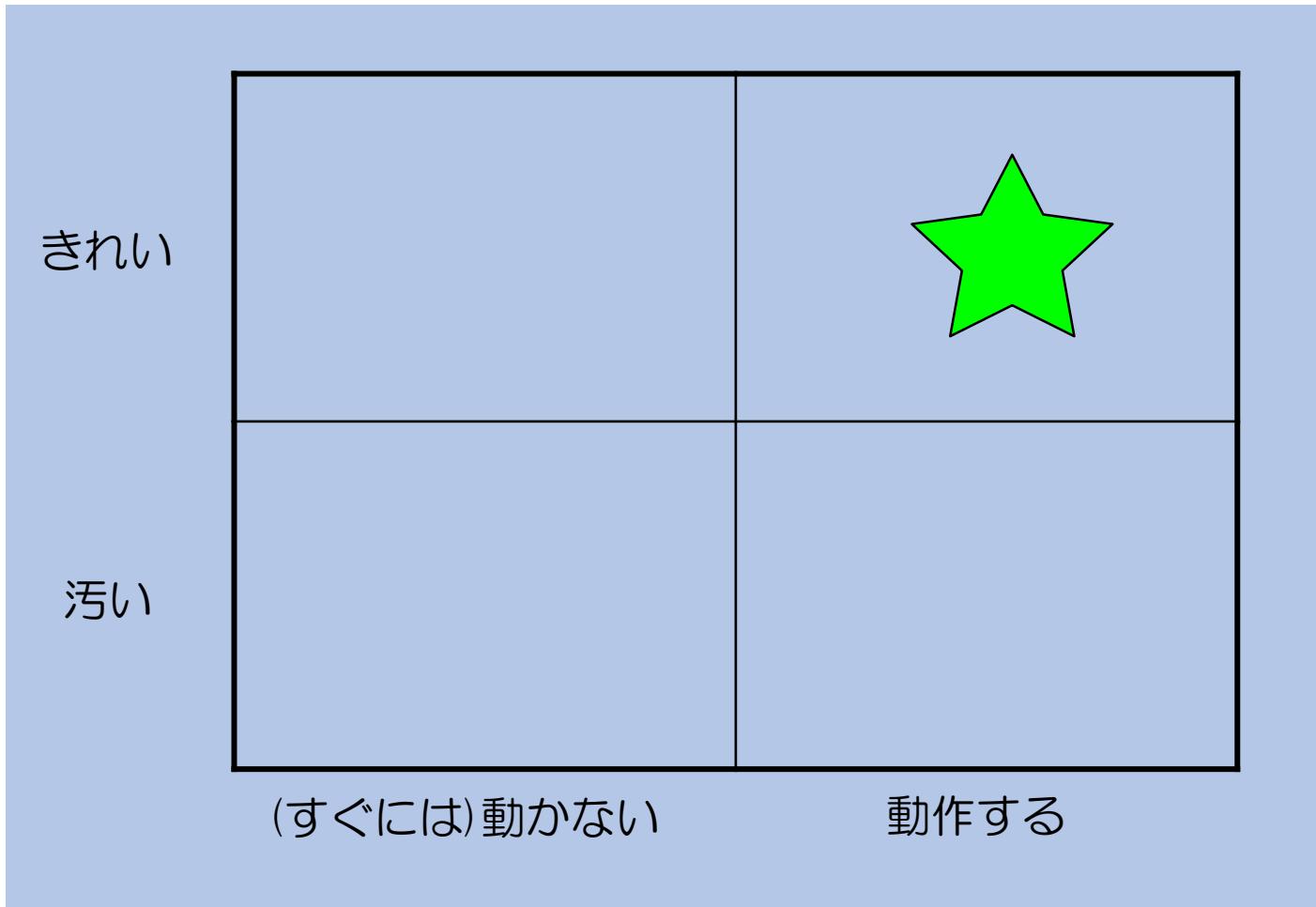
- ・動作している
- ・バグ対応も、追加変更も容易である
- ・作業が予測可能である
- ・チームに信頼が生まれる

複雑ではない
恐れもなく安心

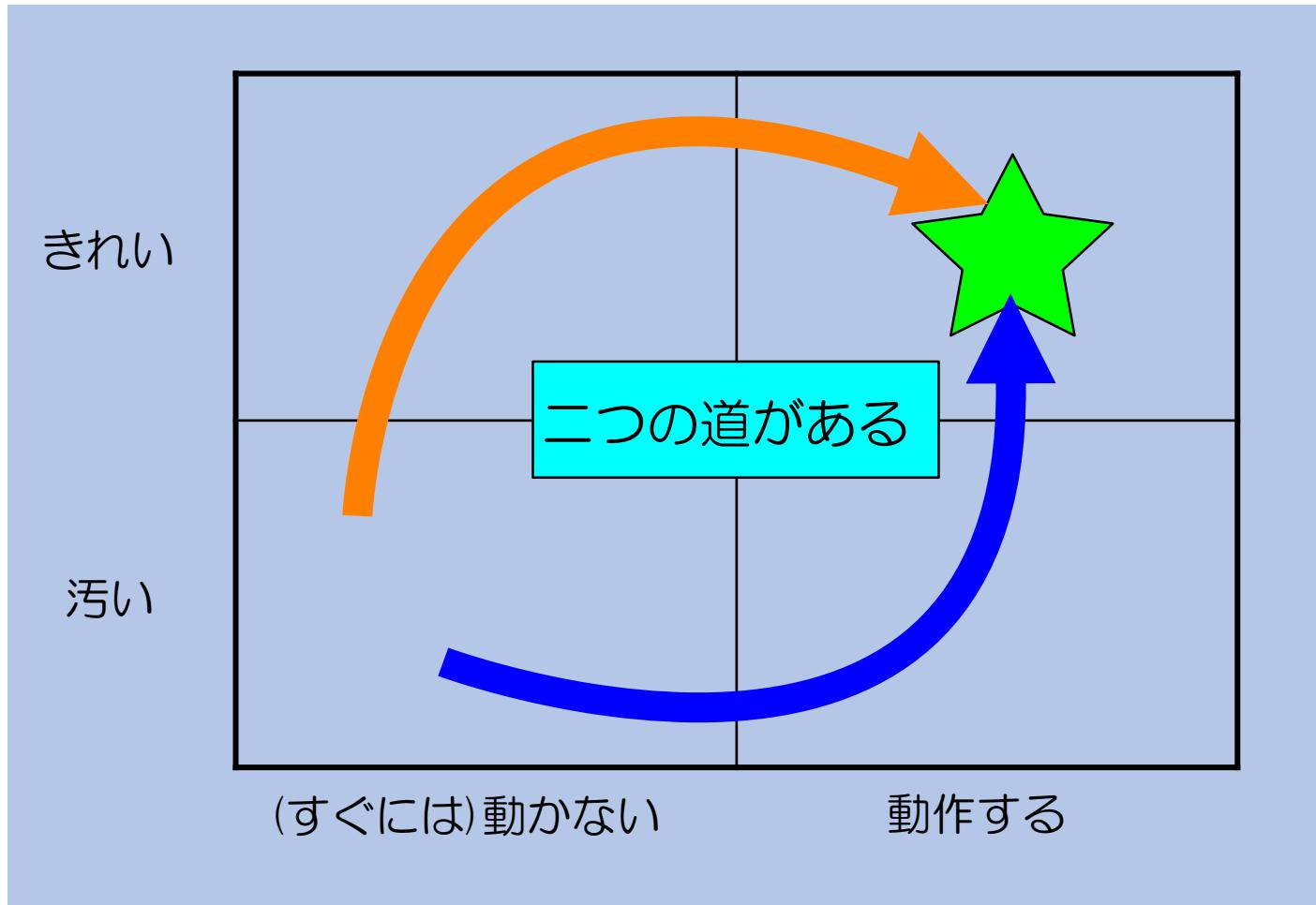




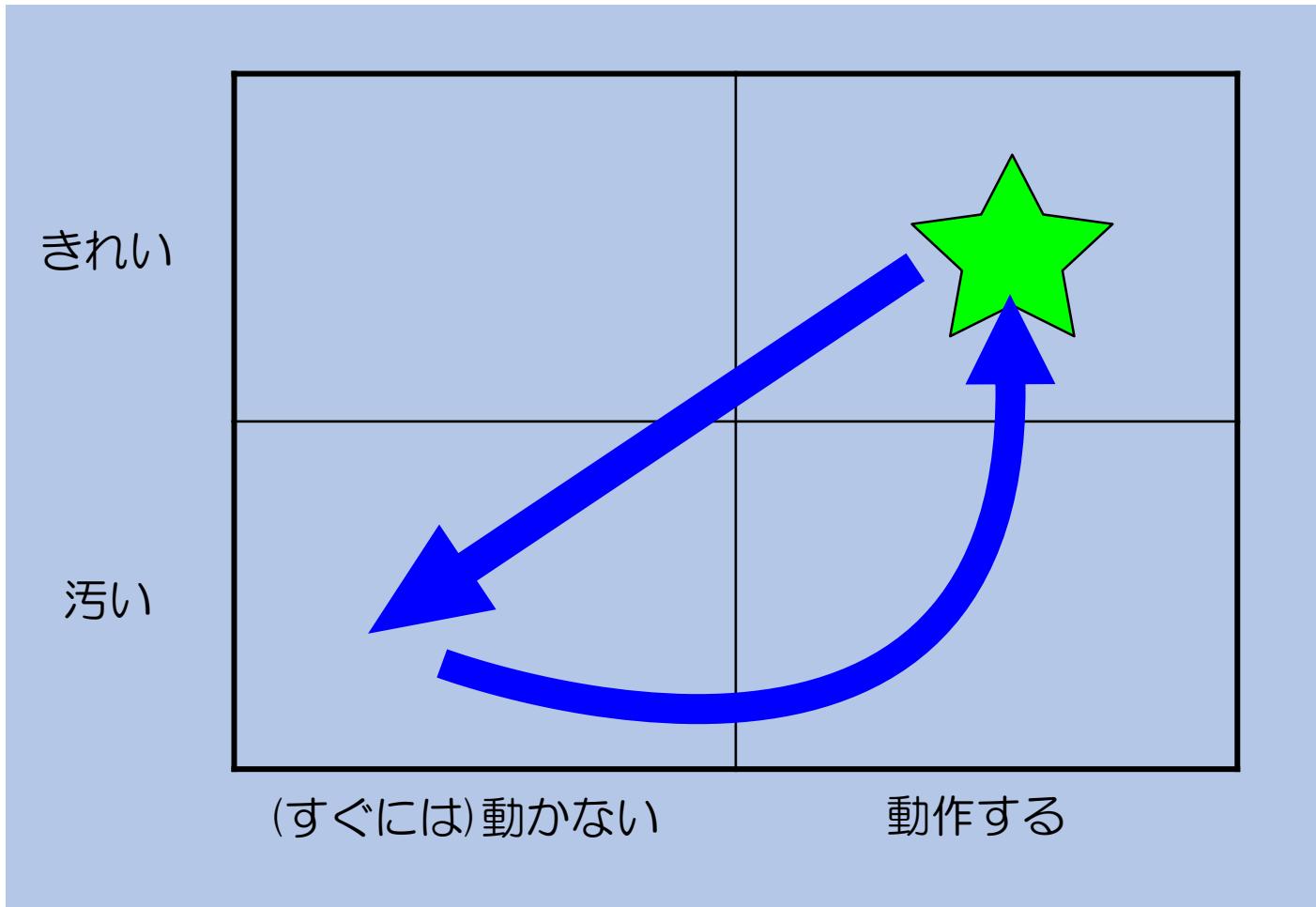
動作する、きれいなコードへ



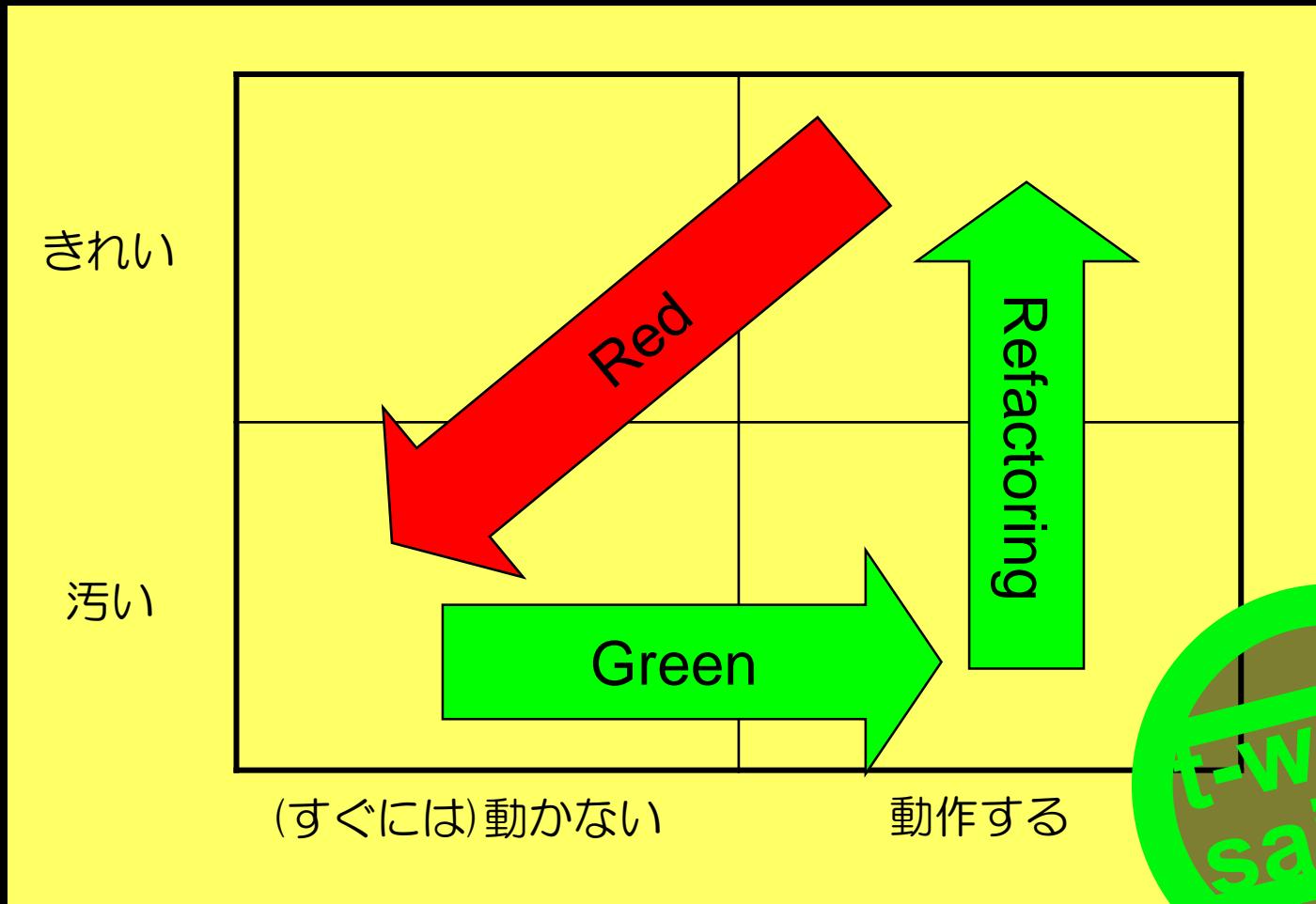
動作する、きれいなコードへ



TDDでサイクルにする

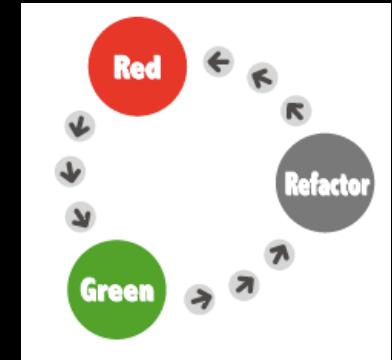


TDDと黄金の回転



TDDのサイクル

1. 次の目標を考える
2. その目標を示すテストを書く
3. そのテストを実行して失敗させる (Red)
4. 目的のコードを書く
5. 2で書いたテストを成功させる (Green)
6. テストが通るまでリファクタリングをおこなう (Refactor)
7. 1～6を繰り返す



テストについて

「テスト」と言えば！

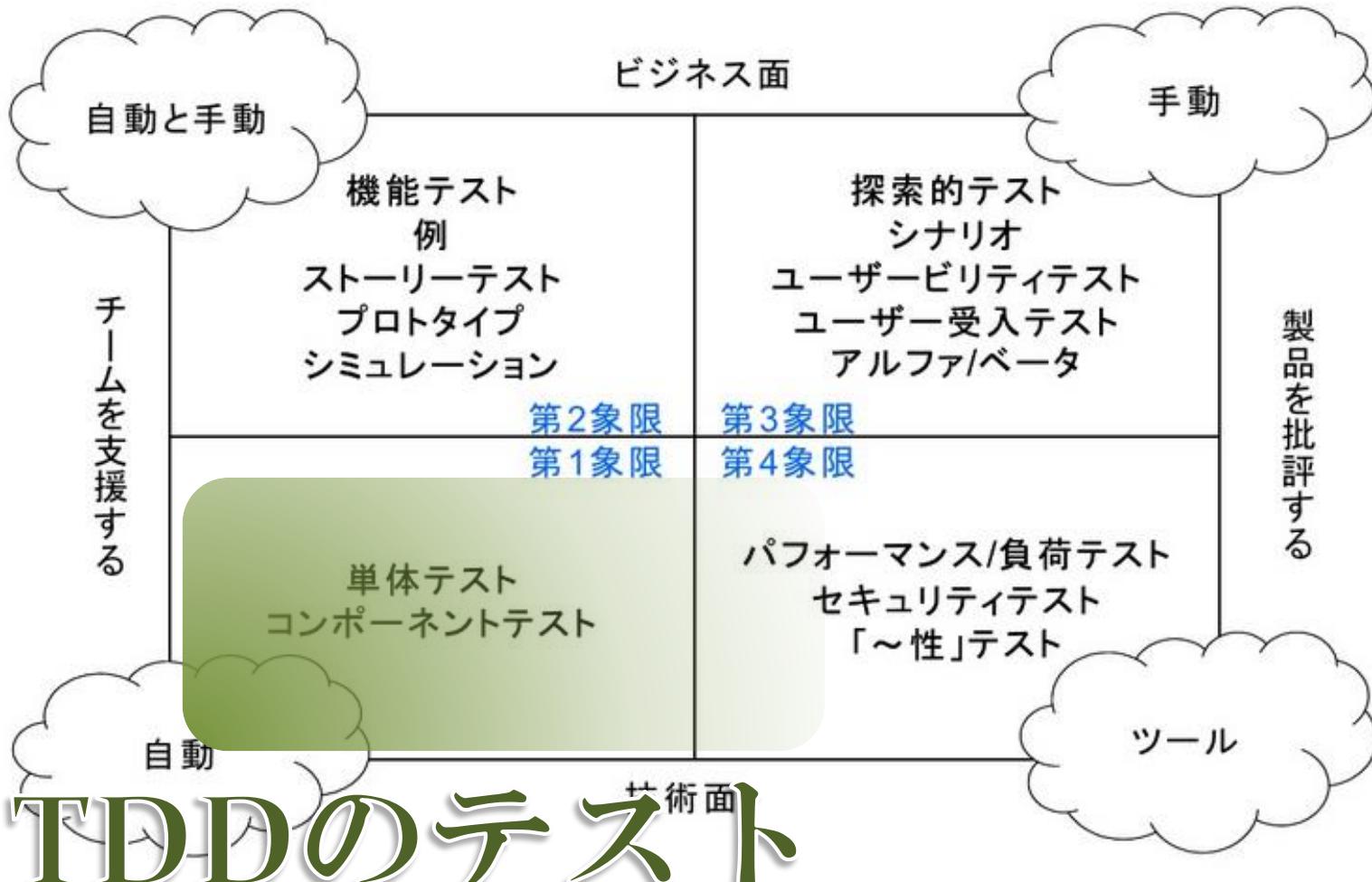
- テストレベル
 - 単体テスト
 - 結合テスト
 - インテグレーションテスト
 - ベータテスト
 - システムテスト
 - 受入テスト
 - 運用テスト
 - ユーザーテスト
- テストタイプ
 - インターフェーステスト
 - UIテスト
 - ユーザビリティテスト
 - パフォーマンステスト
 - 探索的テスト

デベロッパー テスト

- プログラマの
- プログラマによる
- プログラマのための
- プログラムとしてのテストを書きながら
- 開発を行っていく手法

2. アジャイルテストとは

2-2. アジャイルテストの4象限





TDDの「**テスト**」は
「デベロッパーテスト」

=

プログラマの思ったとおりに動くか
プログラマ自身が確認する
プログラマが仕事を進めるための
テスト



良いテストは
どんなものか

- **F**ast
 - **I**ndependent
 - **R**epeatable
 - **S**elf-
- # Validating
- **T**imely



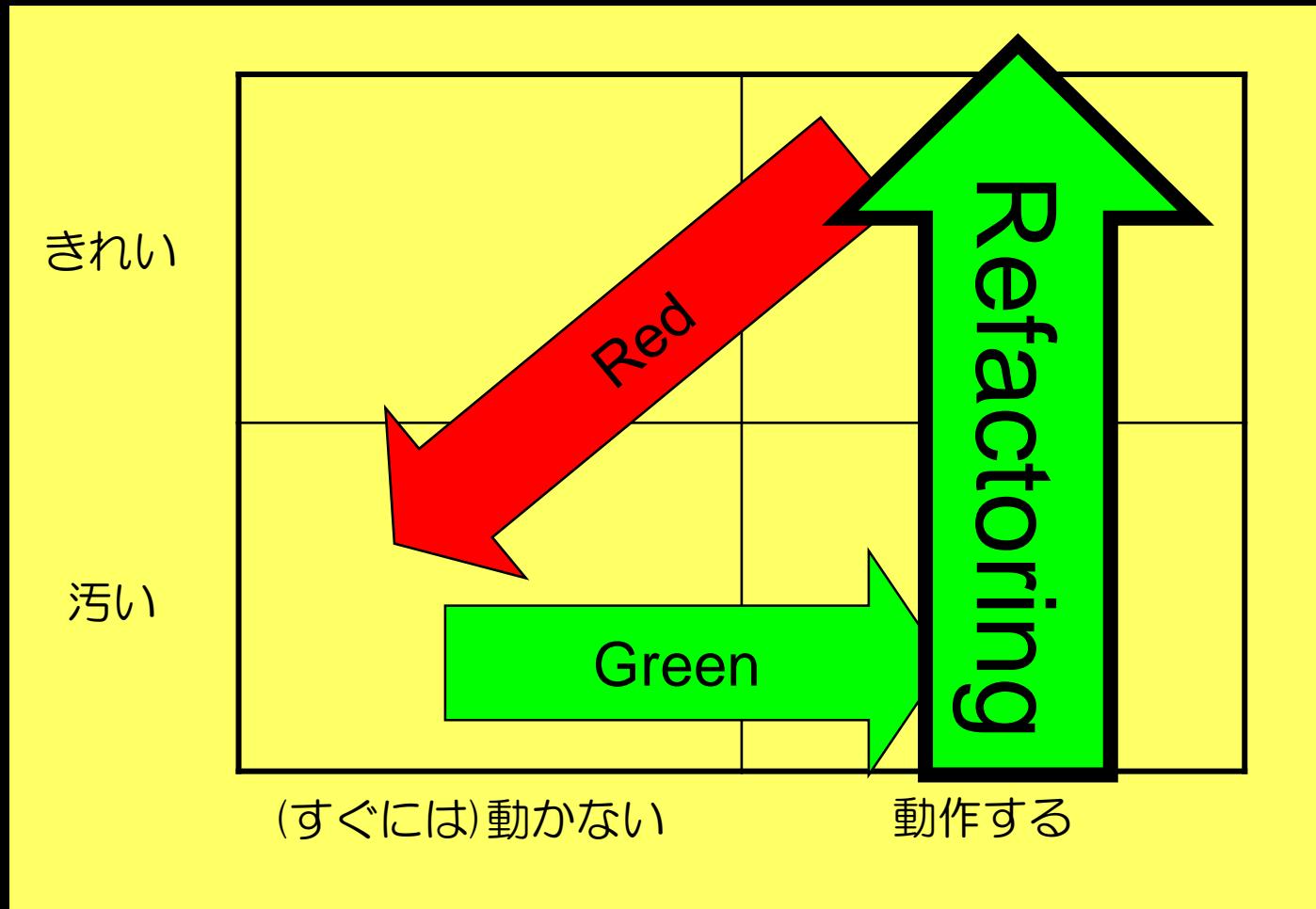
- ・高速である
- ・独立している
- ・再現性がある
- ・自己検証可能
- ・適時性がある



アジャイルソフトウェア導入の技
Clean Code
クリーンコード
Robert C. Martin 著者監修
ASCII

リファクタリングについて

TDDと黄金の回転



きれいを保つ



リファクタリングの対象

- コードをきれいにする
 - メソッド名、変数名
 - インデント
 - コメント
- 設計も見直す
- プロダクトコードもテストコードも見直す
- 常にやり続ける
- 「動くコード」を保証するテストが前提

大

新装版

リファクタリング

既存のコードを安全に改善する

Martin Fowler [著]

児玉公信・友野晶夫・平澤 章・梅澤真史 [共訳]

MARTIN FOWLER

With contributions by Kent Beck, John Brant,
William Opdyke, and Don Roberts

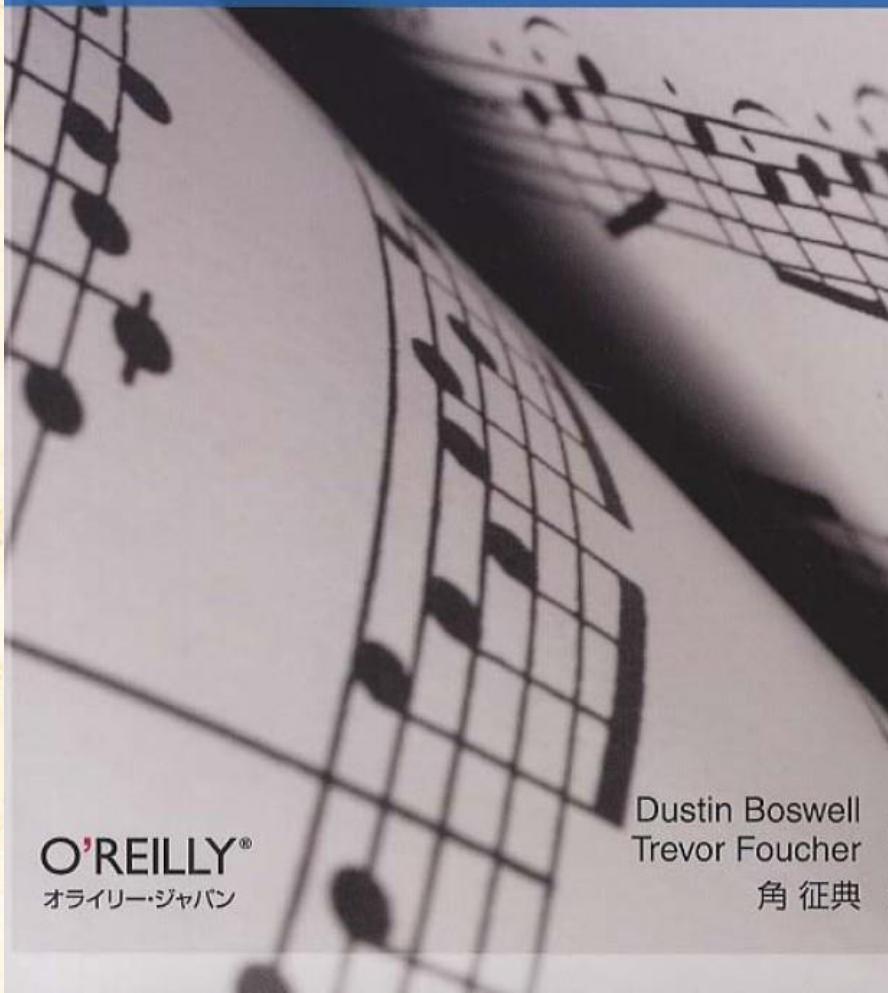
Foreword by Erich Gamma

Refactoring: Improving the Design of Existing Code



リーダブルコード

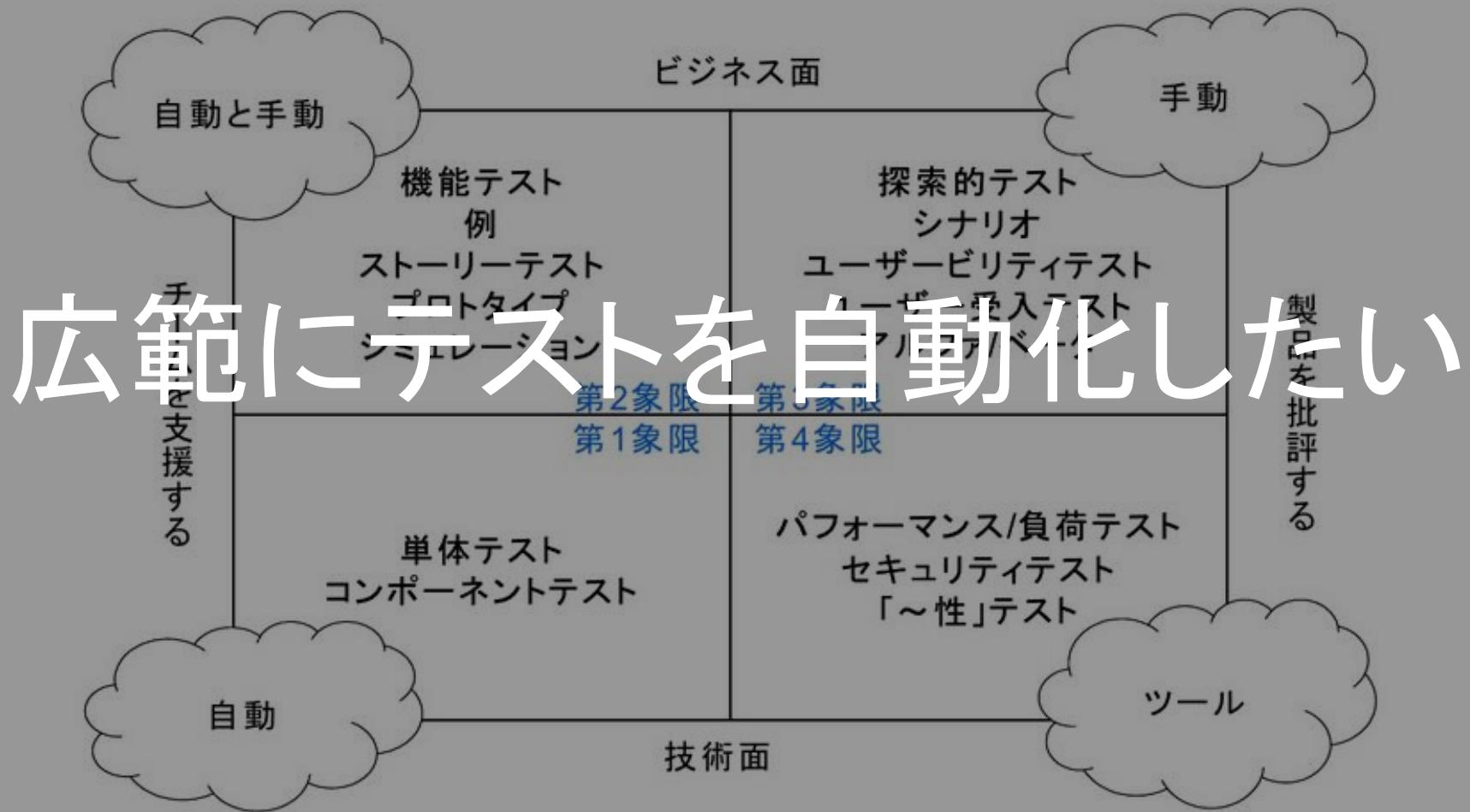
より良いコードを書くための
シンプルで実践的なテクニック



Dustin Boswell
Trevor Foucher
角 征典

2. アジャイルテストとは

2-2. アジャイルテストの4象限





システムテスト自動化 標準ガイド

Mark Fewster, Dorothy Graham 著
テスト自動化研究会 訳

テスト自動化のメリット

- ・回帰テスト
- ・追加・変更が容易、安心
- ・品質管理しやすい、品質向上
- ・機能模擬と多能工
- ・依存性排除
- …他にもたくさん！

TDD



テスト自動化

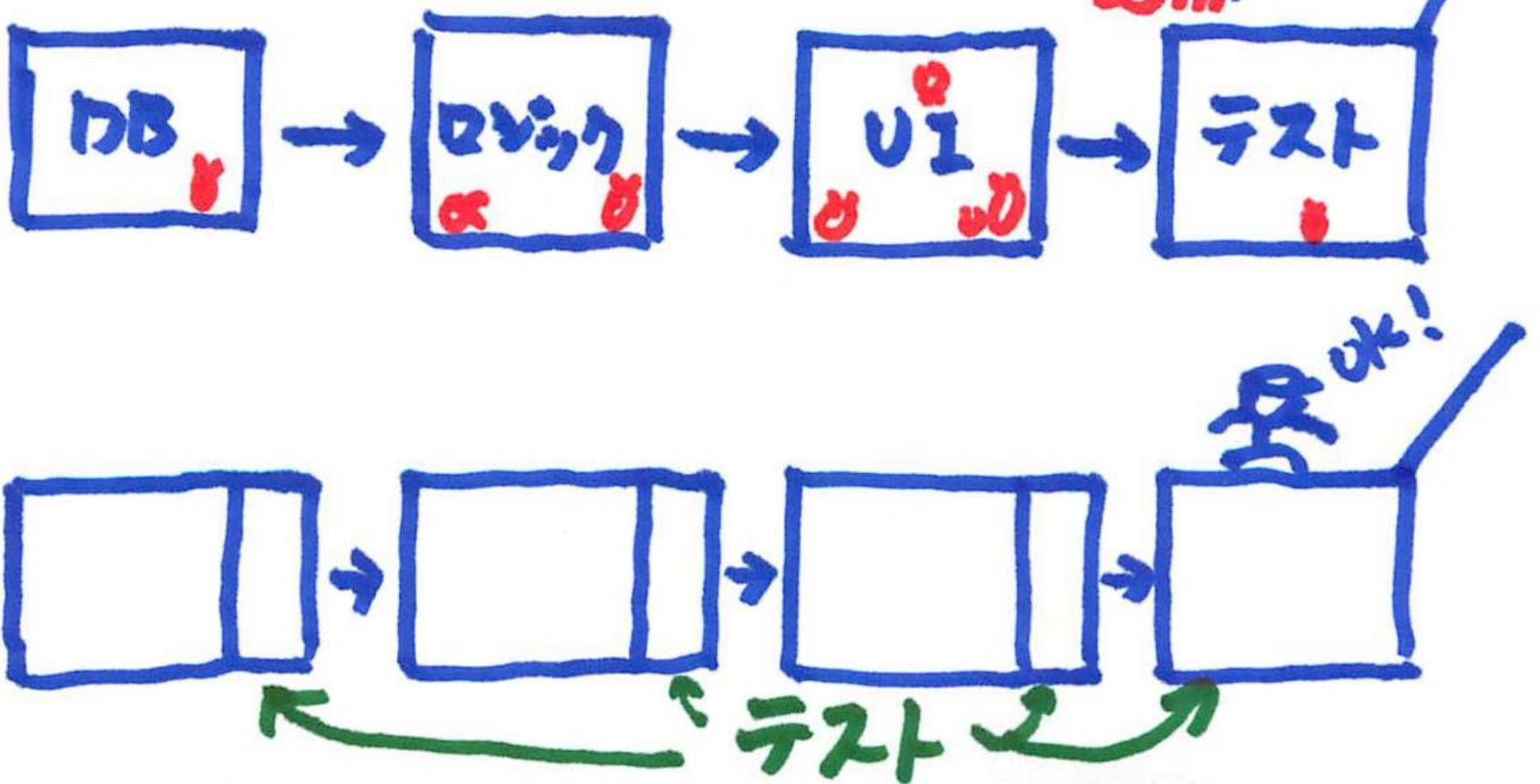
TDD

×

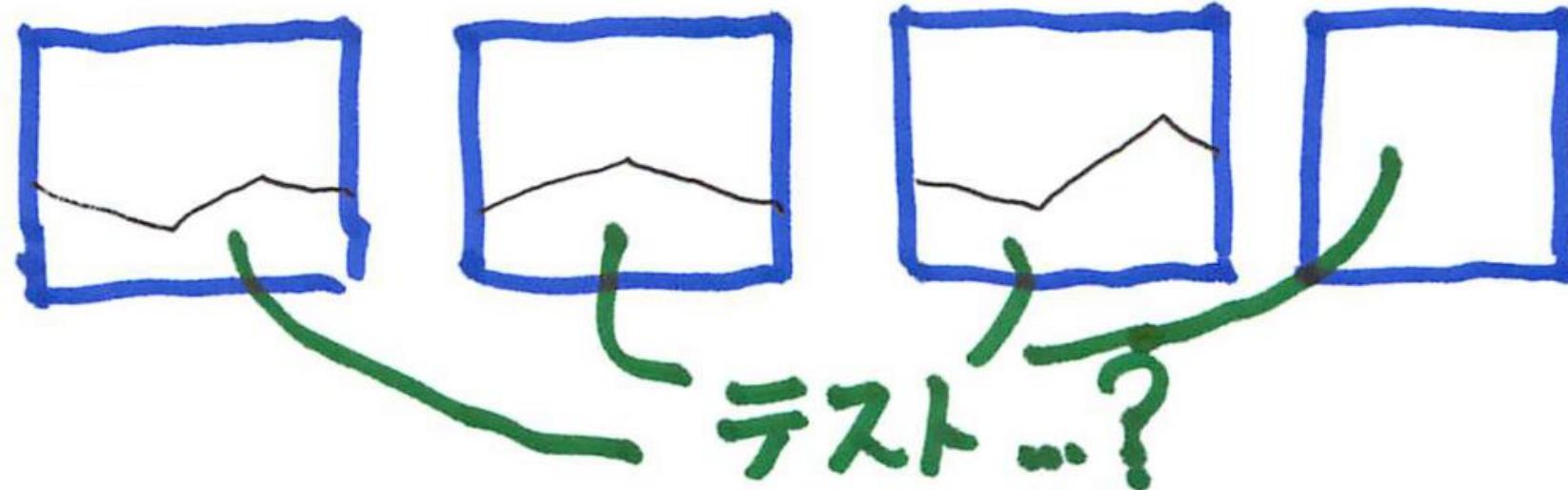
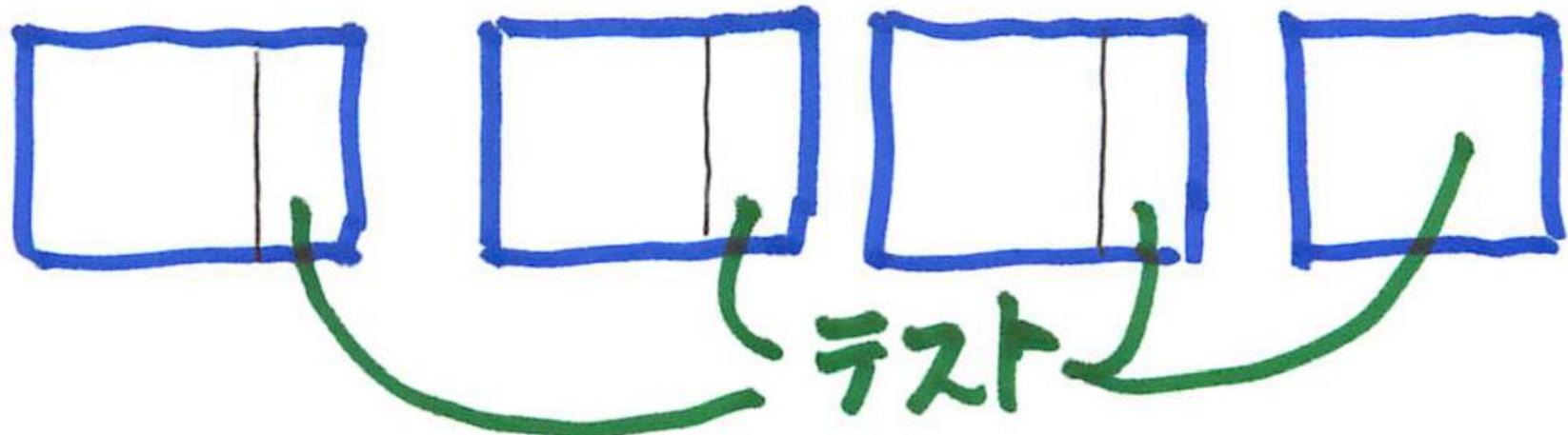
BDD

板3舞い 駆動開発

品質の作り込み



品質の作り込み



テスト駆動開発

+

テスト + 開発
自動化

両方ほしい！

TDDしていれは

自動テストが書ける

試験性

ISO/IEC 9126
ソフトウェアの品質特性

TDDØ
—
—

TDの
テストに出ます！
コードにも出ます！

こころ

A brown and white bulldog is standing on a wooden staircase, looking down. The dog has a white patch on its forehead and a white stripe down its nose. It is leaning its front paws on a wooden railing.

一つずつ
少しずつ

段を
小さく

複数を相手
にしない。

ひとりずつ
対処する。



A fluffy brown hamster is running in a red exercise wheel. A yellow ball is visible in the background.

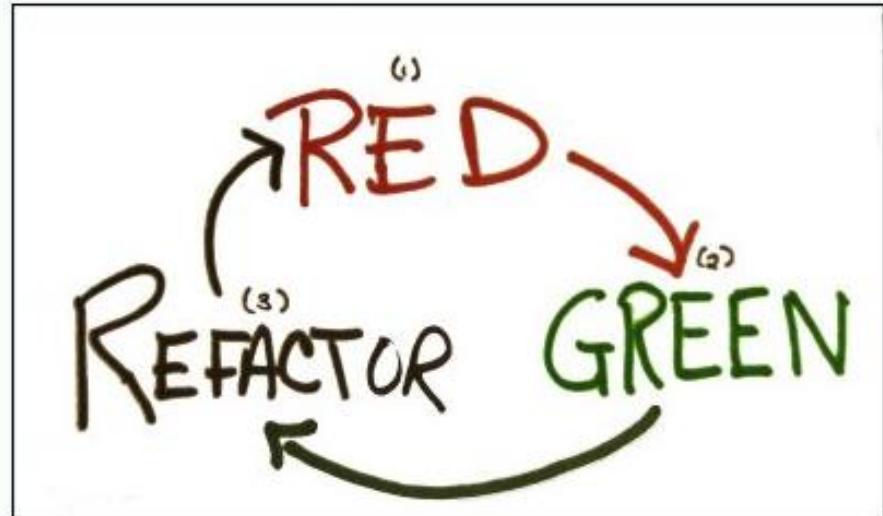
すばやく
まわす

The Clean Code Blog

by Robert C. Martin (Uncle Bob)

The Cycles of TDD

17 December 2014



Second-by-Second *nano-cycle*: The Three Laws of TDD.

Minute-by-Minute: *micro-cycle*: Red-Green-Refactor

Decaminute-by-Decaminute: *milli-cycle*: Specific/Generic

Hour-by-Hour: *Primary Cycle*: Boundaries.

自分が最初の
ユーザ

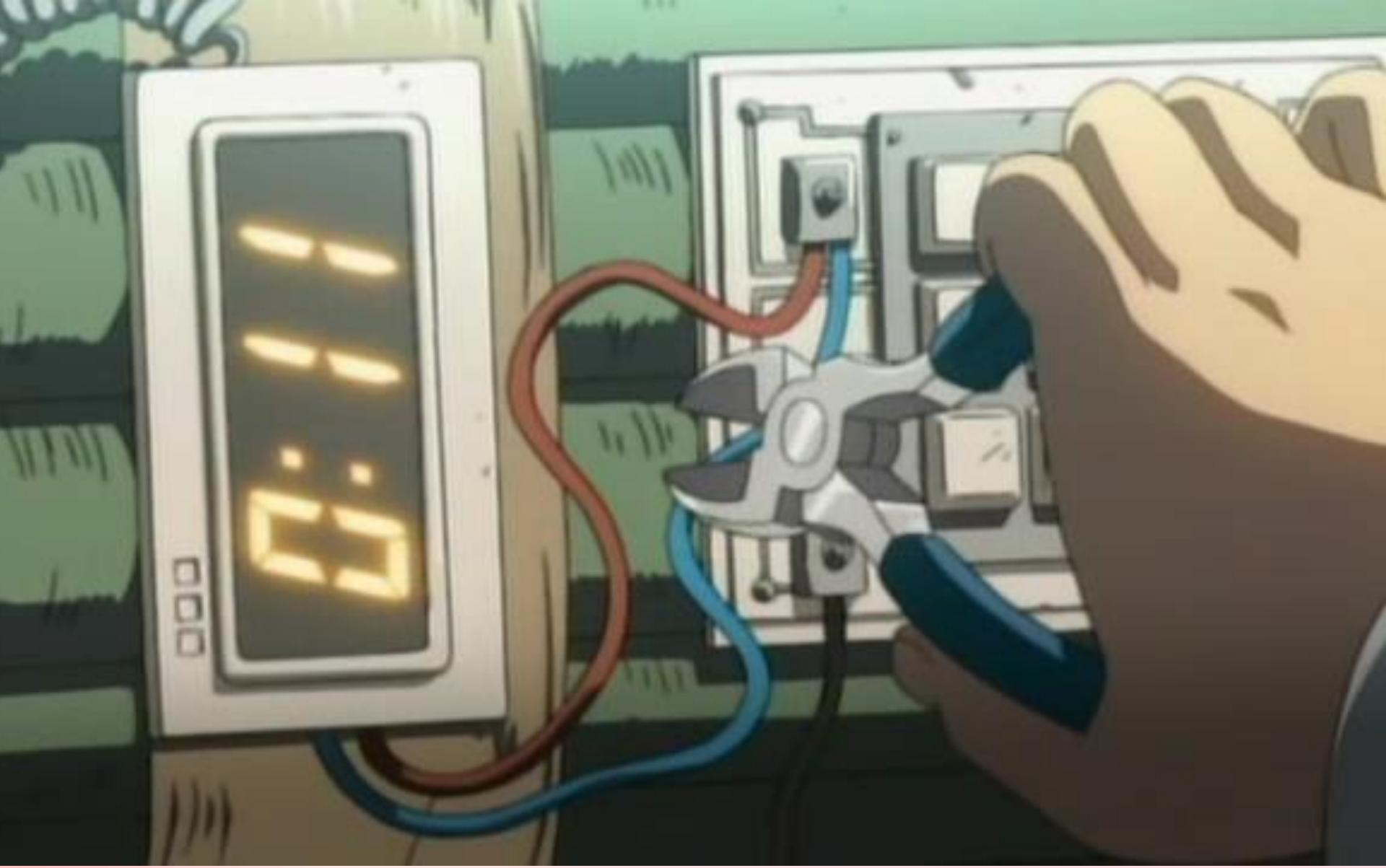


Google™
Apps





不安を
テス



祈るのではダメ

命綱を編む



TDDのこころ

- ・ひとつずつ 少しずつ
- ・複数を相手にしない
- ・すばやく回す
- ・自分が最初のユーザ
- ・不安をテストに
- ・祈るのではダメ
- ・命綱を編む

質問:TDDをやろうと思ったのはなぜですか?

面白そうだから!



- ・脳味噌の想像力を超えた設計に到達
- ・自分の能力を拡大する

事実、ペアプロは楽しい。

楽しみ
ましょっ!



FIZZBUZZ問題

1から100までの数をプリントするプログラムを書け。ただし3の倍数のときは数の代わりに「Fizz」と、5の倍数のときは「Buzz」とプリントし、3と5両方の倍数の場合には「FizzBuzz」とプリントすること。

本日のお題

整数の区間 <http://bit.ly/1hAoTeG>

本日のお題

スーパーの支払金額 <https://is.gd/1sXS1B>

スーパーの支払金額

スーパーで買い物したときの支払金額を計算する
以下の商品リストがあるとする。先頭の数字は商品番号。

1. りんご 100円
2. みかん 40円
3. ぶどう 150円
4. のり弁 350円
5. しゃけ弁 400円
6. タバコ 420円
7. メンソールタバコ 440円
8. ライター 100円
9. お茶 80円
10. コーヒー 100円

スーパーの支払金額



以下の順番で、仕様を追加・実装していく。

お題1 合計金額

商品番号と個数を複数組、引数として受け取り、合計金額を計算する関数を書いてみよう。

ヒント: 複数のものを受け取るために、配列やリストで一括で渡す方法がある。あるいは、1つ渡す関数を何回も呼び出して、最後に合計金額を計算する関数を呼び出すという形式もある。両方のアプローチをTDDで実装し見比べて、どちらが良いか判断してみよう。

いきなり書くのが難しかったら、以下の補題をやってみるとよい。

補題1

商品番号を渡すと、1個あたりの金額を計算する関数を書いてみよう。

補題2

商品番号を複数渡すと、個数1個として金額を合計する関数を書いてみよう。

スーパーの支払金額

お題2 消費税

商品リストの金額は外税なので、合計金額に消費税8%を足して、支払金額を返すようにしよう。

お題3 タバコの消費税

タバコの価格には消費税が含まれているので(内税)、消費税の計算からタバコは除かないといけない。

スーパーの支払金額

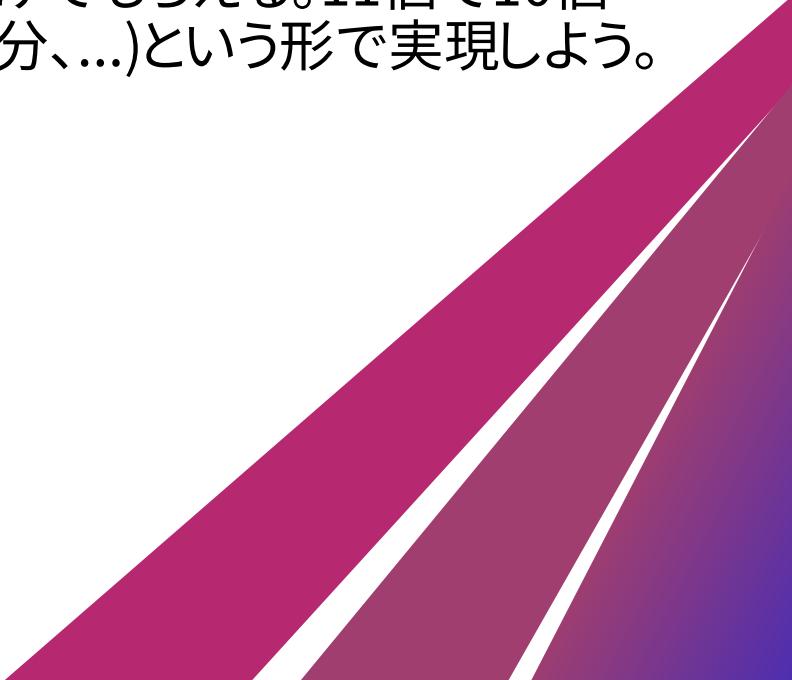


お題4 割引

リンゴは1個100円だが、3つ買うと280円になる。

お題5 おまけ

なんでも、同じものを10個買うと、1個おまけでもらえる。11個で10個ぶんの金額(12個で11個分、20個で19個分、...)という形で実現しよう。



スーパーの支払金額



お題6 おまけのライター

タバコを1カートン(10個)買うと、ライターがおまけでもらえる。引数にライターがあったら無料になるというふうに実現しよう。

お題7 お弁当

弁当類と飲み物(お茶とコーヒー)をいっしょに買うと、20円引きになる。



スーパーの支払金額



お題8 サービスしそぎない

お題4～7のようなサービスは、同じ商品については重複しない。一番安くなるものをひとつだけ適用する。

お題9 タイムセール

お弁当は20時を過ぎると半額になる。

お題10 タイムセールとサービス

お弁当のタイムセールは、他のサービスと重複してよい。



スーパーの支払金額



お題11 コンフィグレーション

以上のようなサービス内容を、プログラムと別の設定ファイルなどで自由に変更できるようにしたい。

事実、ペアプロは楽しい。



おつかれさま
でした!

TDD
テスト駆動開発
応用編



いよいよ実践だ！

実践で困ったら？



テスト駆動開発の効果

| | IBM ドライバ | Microsoft Windows | Microsoft MSN | Microsoft VisualStudio |
|-------------------------|-------------|----------------------|------------------|---------------------------|
| チーム人数 | 9 | 6 | 5-7 | 7 |
| コード量(KLOC) | 41.0 | 6.0 | 26.0 | 155.2 |
| 開発規模(人月) | 119 | 24 | 46 | 20 |
| 欠陥数 (TDD未使用に対 する) | 61% | 38% | 24% | 9% |
| 開発時間の増加 (管理者の見積) | 15~20% | 25~35% | 15% | 20~25% |

Nachiappan Nagappan, E. Michael Maximilien, Thirumalesh Bhat, Laurie Williams
“Realizing quality improvement through test driven development: results and experiences
of four industrial teams” 2008

http://research.microsoft.com/en-us/groups/ese/nagappan_tdd.pdf

TDDの効果の研究をまとめた研究

"Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies" Simo Makinen and Jurgen Munch

- 既存の実証研究を調査し、10の内部・外部品質評価項目で、各研究の結論を整理した
- TDDは欠陥の作り込み(introduced defects)を減らし、メンテナンスしやすいコードを産む
- TDDで実装されたコードは、部分的に、サイズが小さく、複雑度が低い場合がある
- メンテナンスがしやすくなるものの、初期開発では時間がかかる

TDDの効果の研究をまとめた研究

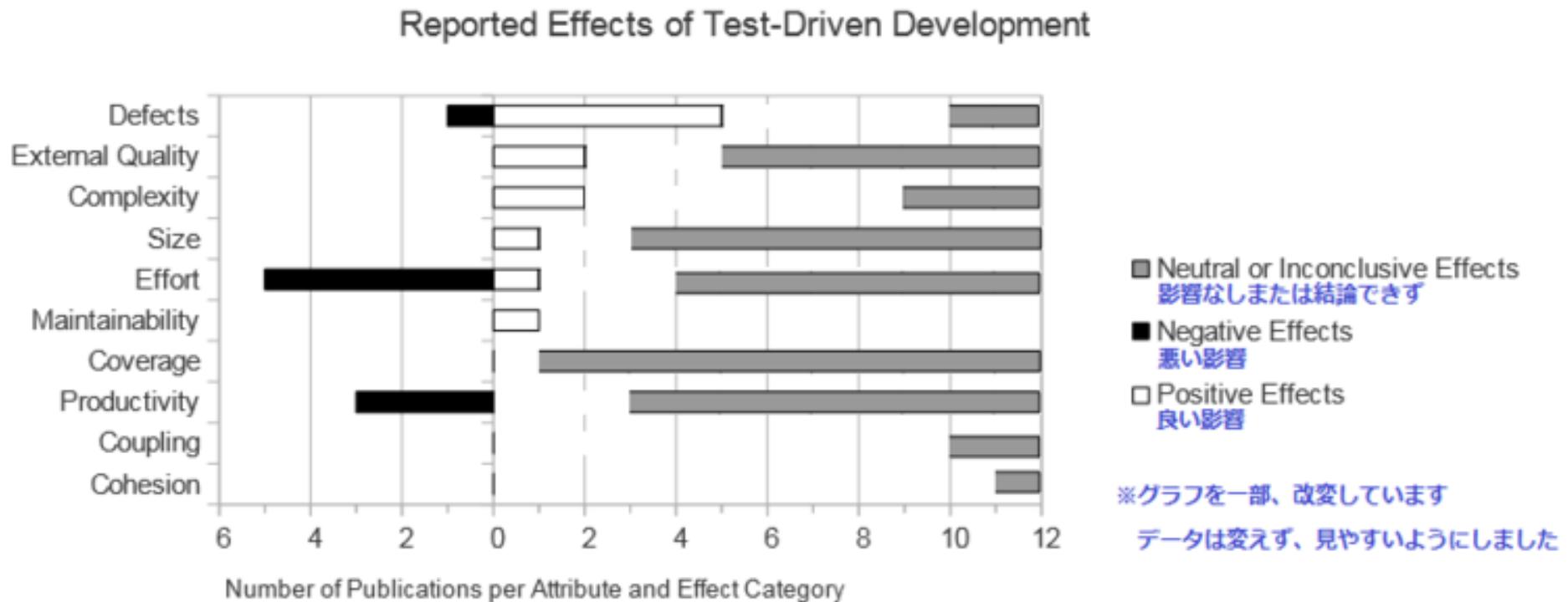


Fig. 1. The occurrence of positive, neutral and negative effects for each quality attribute as reported by the test-driven development publications included in the review

TDDの効果(言い訳編)

- テストを自動化できます！
- 品質が良くなります！
- テストがドキュメントの代わりになります！
- 新しい人もすぐキャッチアップできます！
- メンバーが抜けても大丈夫です！
- 修正してもデグレません！
- (手で)テストしなくてすみます！
- 変更コストが下がります！

※用法、用量を守って安全に使いましょう

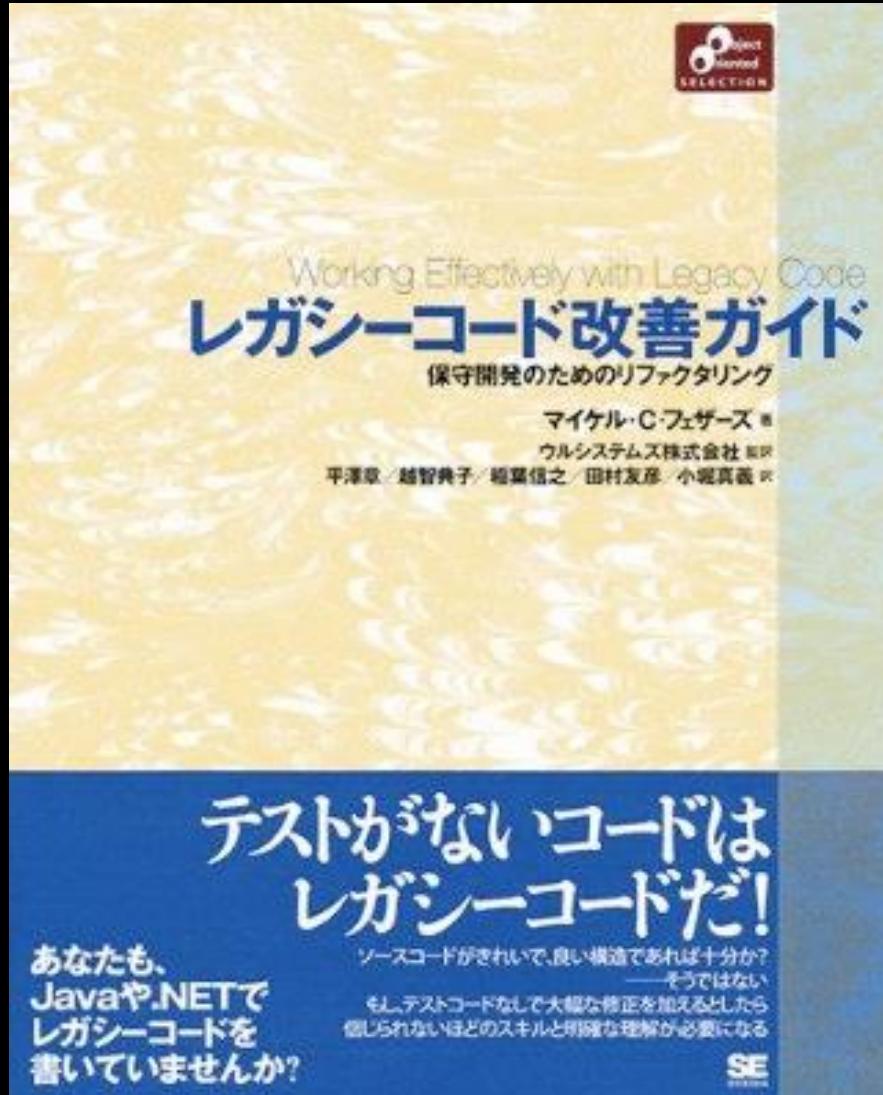
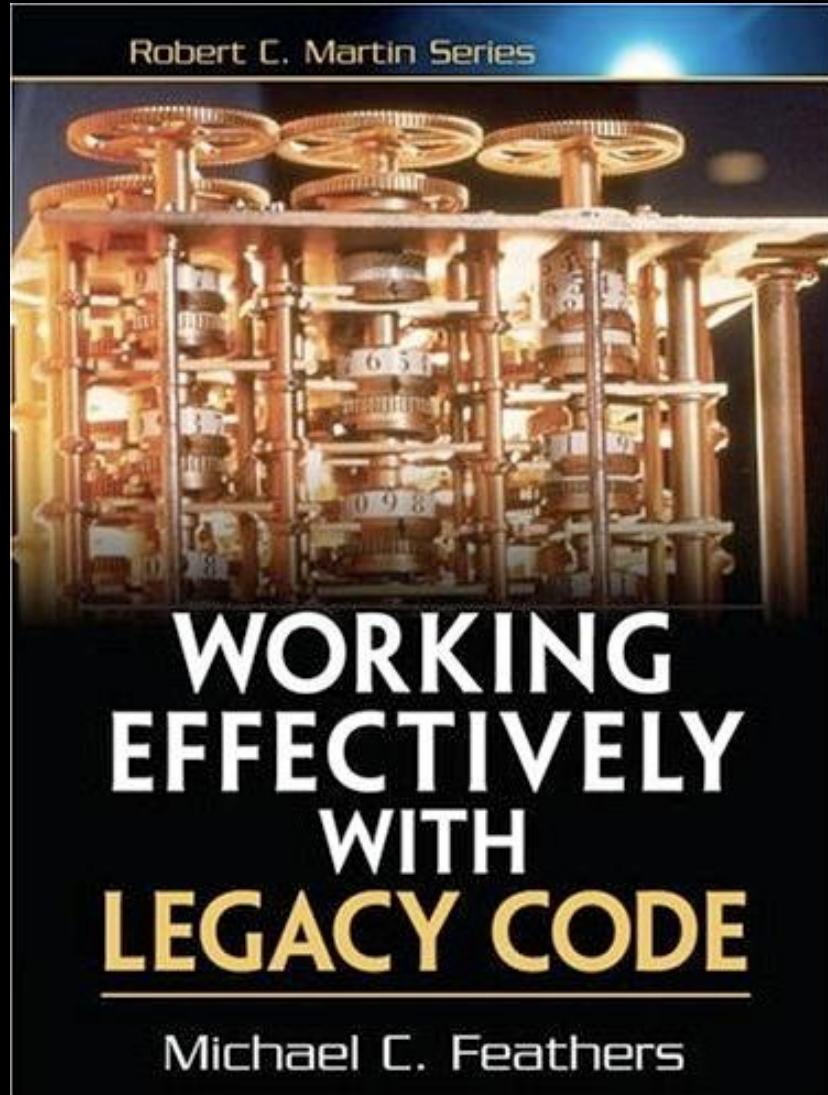
多様な言語とツール



テストの無いコードが
既にたくさんある



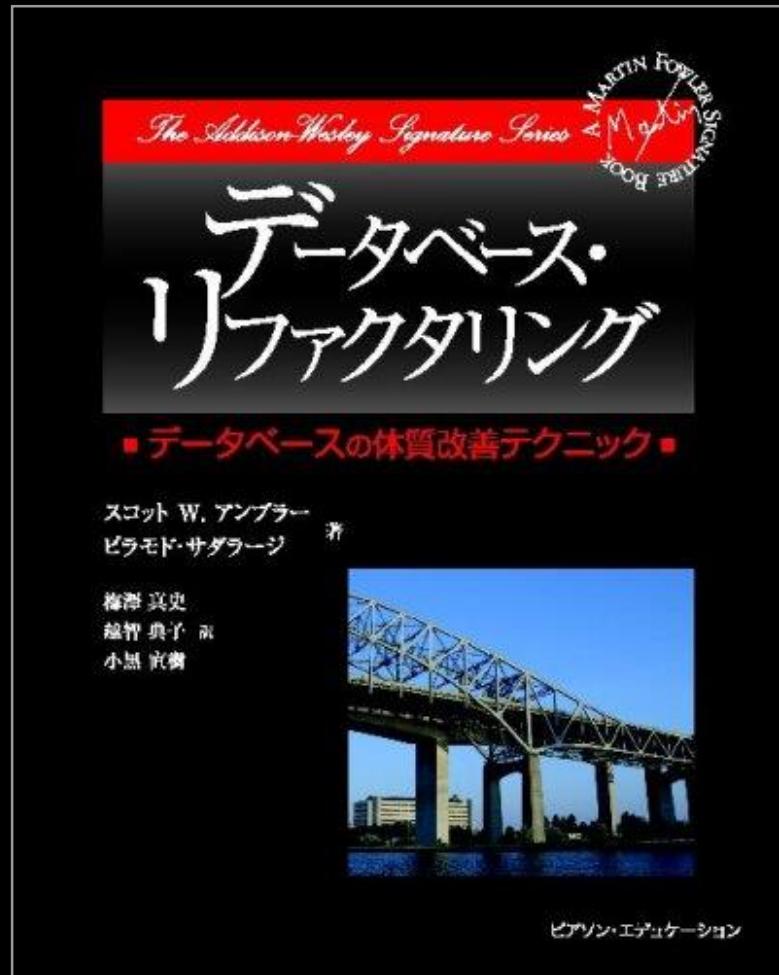
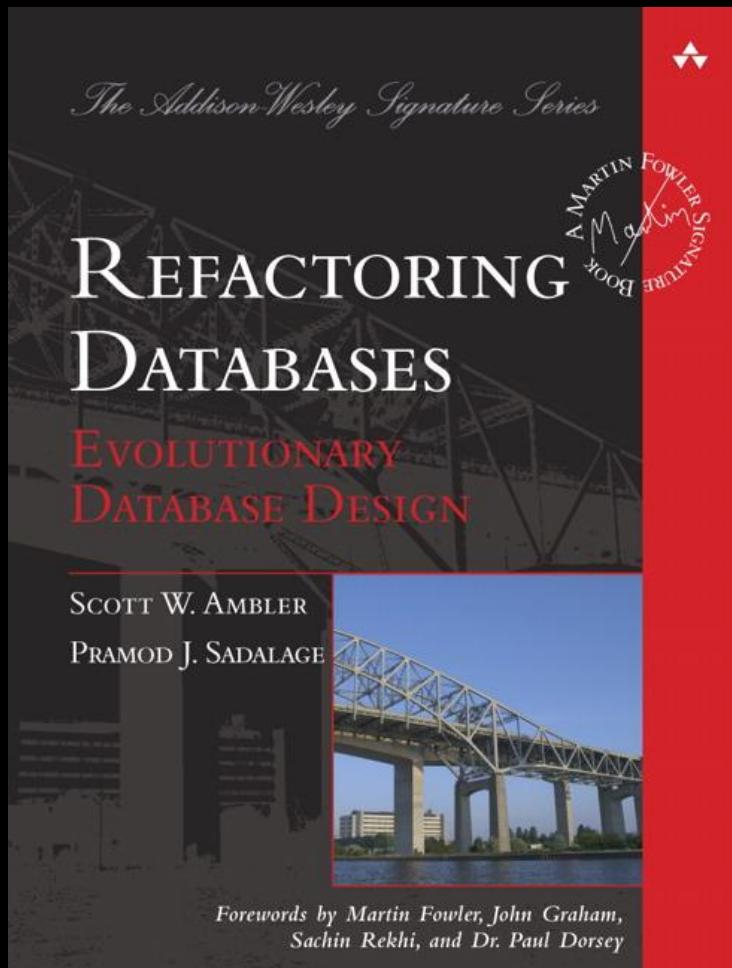
レガシーコード改善ガイド



既にデータの入った
データベースがある



データベース リファクタリング



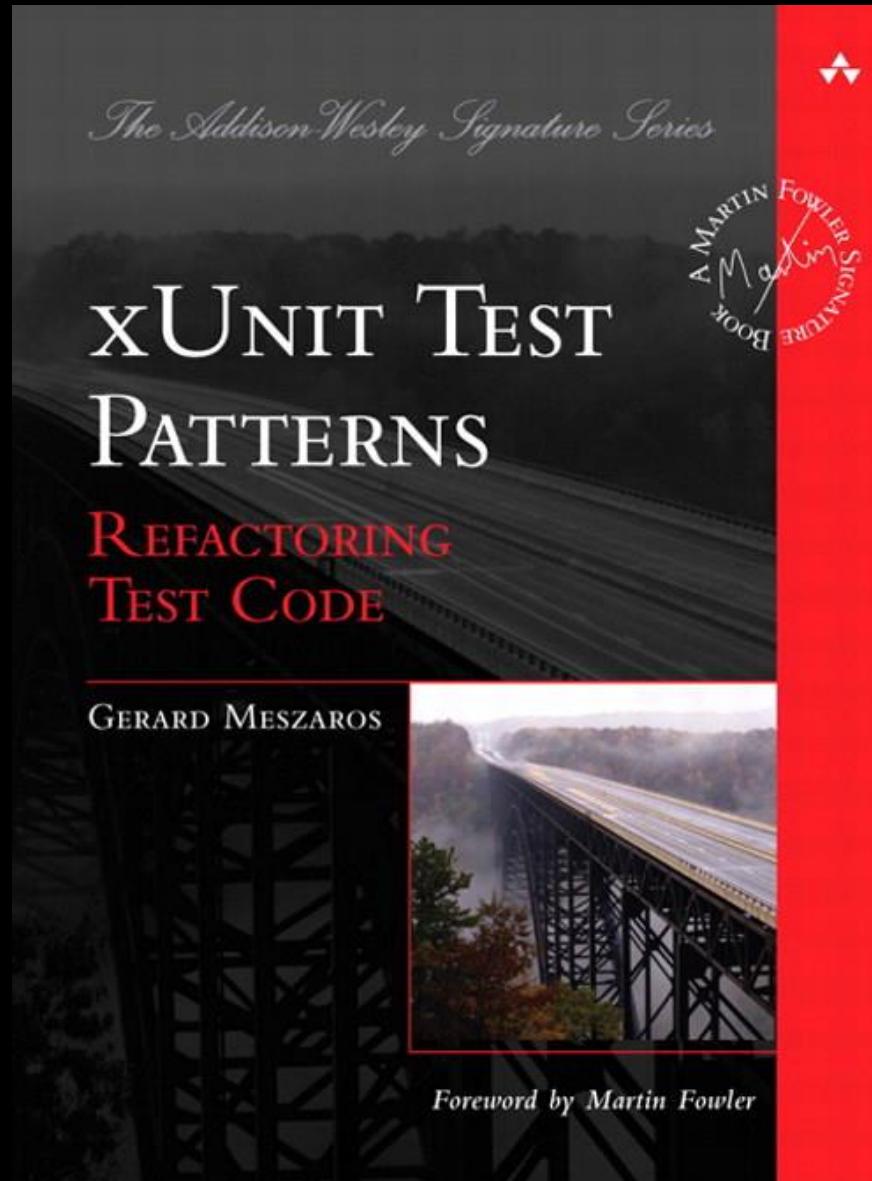
Fragile Tests





Slow Tests

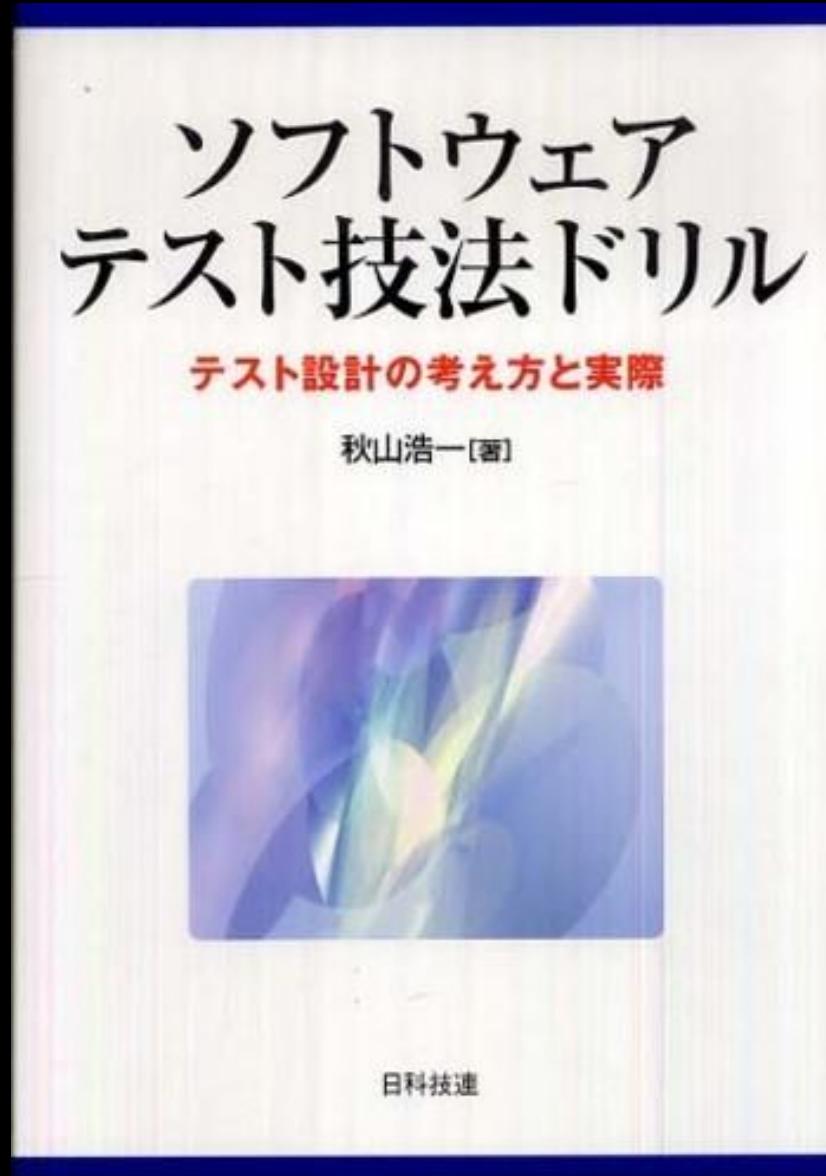
xUnit Test Patterns



どこまでテストすればよいのか



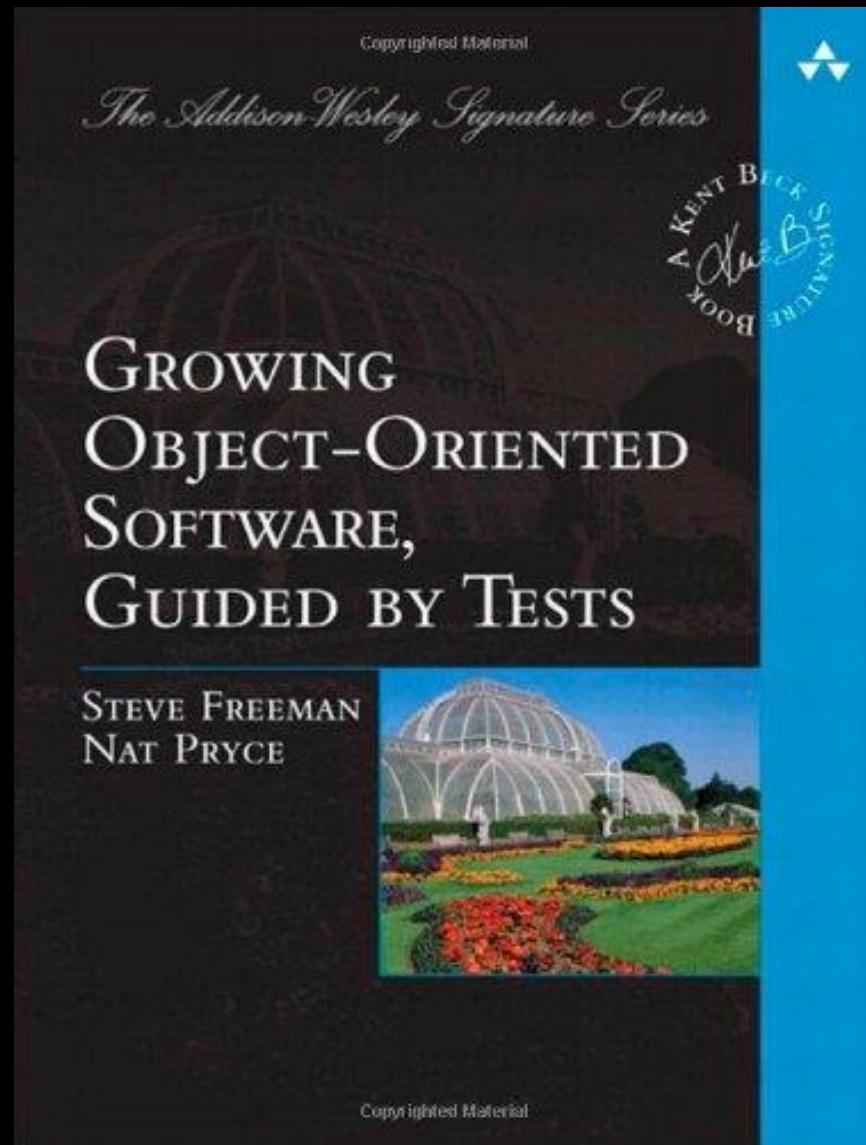
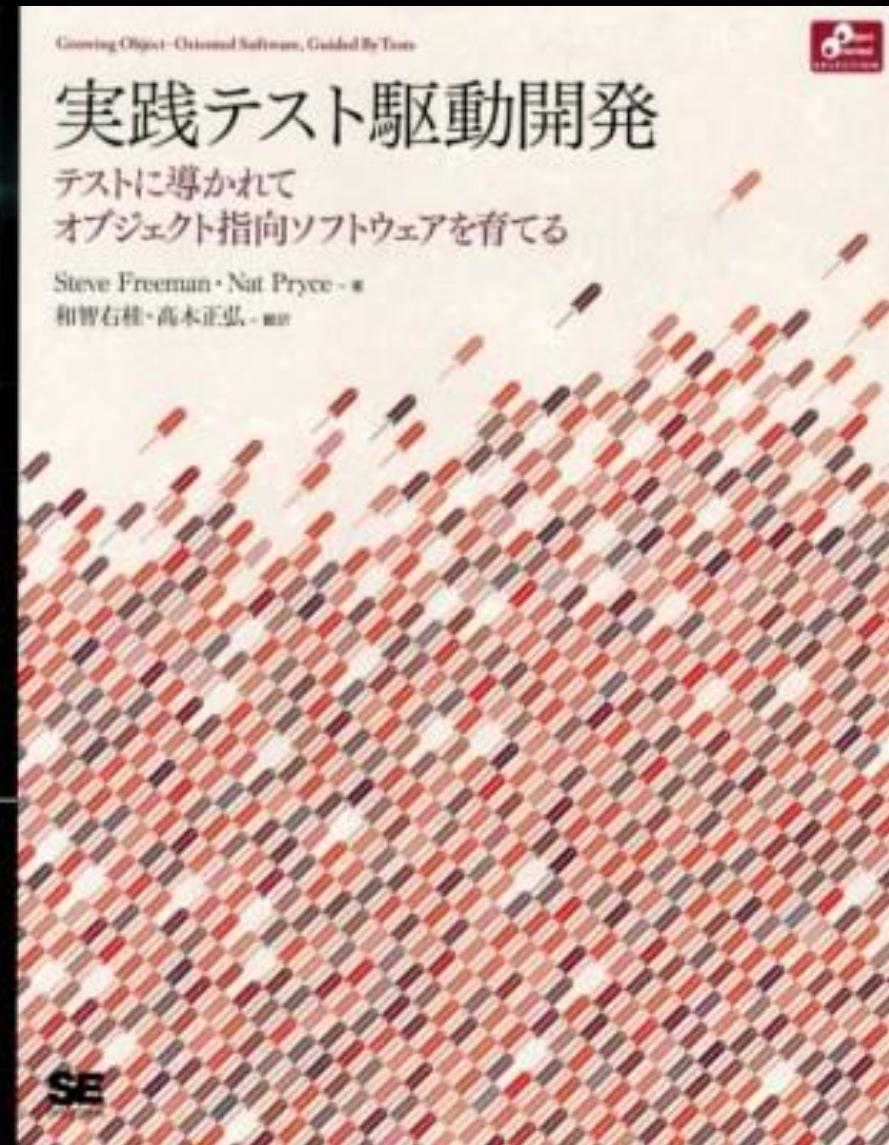
ソフトウェアテスト技法ドリル



現実のシステムはもっと複雑だ



実践テスト駆動開発



組み込みシステムはどうすれば?



テスト駆動開発による 組み込みプログラミング

C言語とオブジェクト指向で学ぶアジャイルな設計

James W. Grenning 著
蜻島 昭之 監訳
菅井 崇司 訳



O'REILLY
オライリー・ジャパン

周囲に広めたい



FEARLESS CHANGE

Patterns for Delivering New Ideas

アジャイルに効く アイデアを組織に 広めるための 48のパターン

Kathy Lynn Monroe, Linda Rising 著

山口恭伸 翻訳

大内正義・塩川利樹・佐藤一貴・中川大輔・大庭賢一

吉井浩・山口恭伸・中澤仁・内田圭一

丸善出版

Organizational Patterns of Agile Software Development



組織パターン

チームの成長により

アジャイルソフトウェア開発の変革を促す

James O. Coplien · Neil B. Harrison - 著

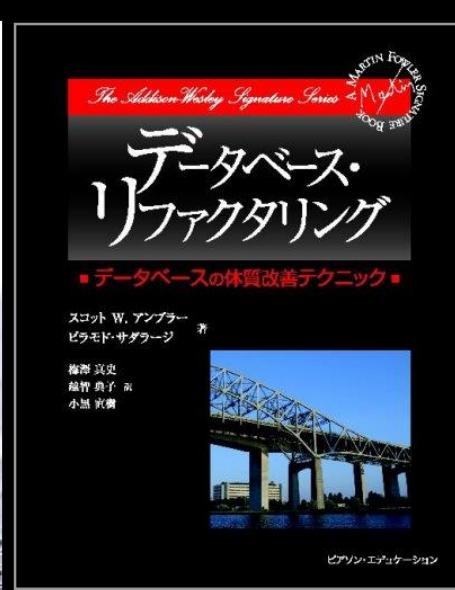
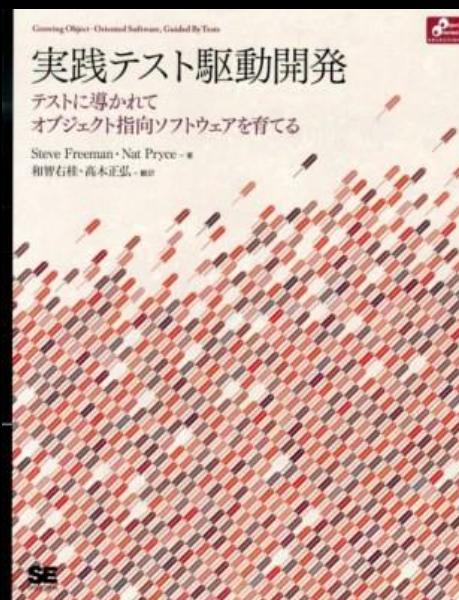
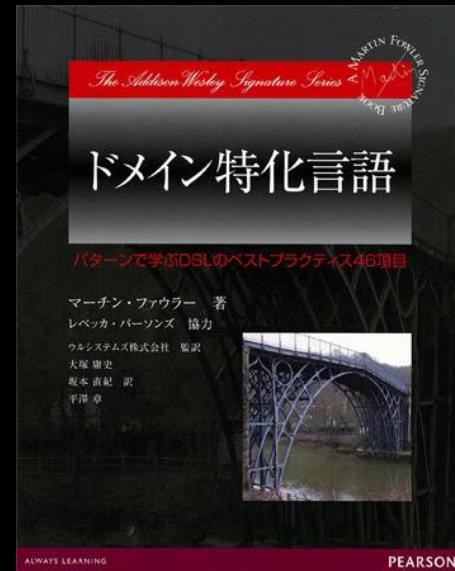
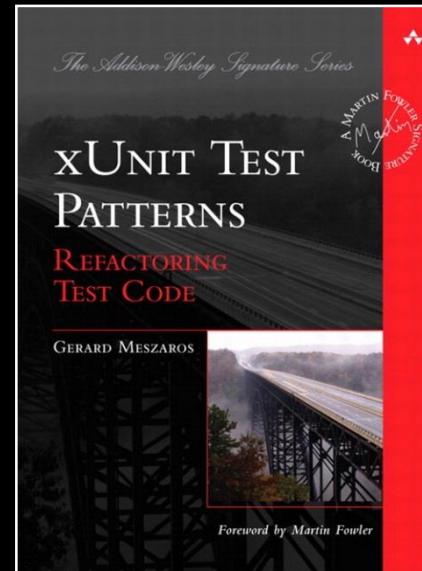
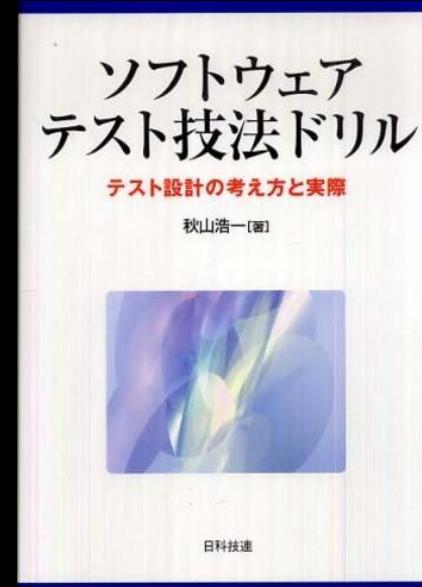
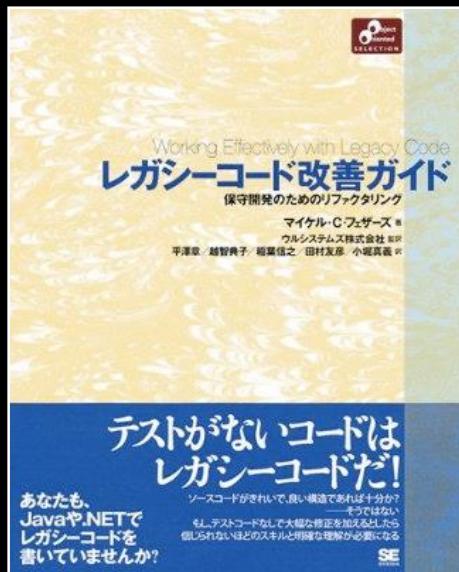
和井 石柱 - 翻訳

「プロセスが答えではないのなら、 どこに答えがあるのだろう?」

- 組織構造とロールの関係に着目し、優れたソフトウェアを効率的に次々と生み出す実例を究明
- クリストファー・アレグザンダーの「パターンの思想を、ソフトウェア開発に継承させ結果させたパターン言語に集約

ソフトウェア開発というダイナミックな変遷を行なう日々にとっての遊しるべとなる一冊

SE
CLASSICS



へ
の
道

実
践
者

T
D
D



TDDは
スキルである



TDDは
練習して上達する



TDDの効果(言い訳編)

- 実践し実績で示す
 - 自分たちが自分の環境・状況で
 - 効果を出せること
- テストがドキュメントになります！
 - 新しい人もすぐキャッチアップできます！
 - メンバーが抜ける大変な時に備えます！
 - 修正してもボグしません！
 - (手で)テストしながら書きます！
 - 変更コストが下がります！

※用法、用量を守って安全に使いましょう

自らを律する



Uncle Bob said ...

「クズコードを書くのは
もうやめだ」

「顧客のため
最高のコードを書く」

「上司のため
最高のテストを書く」

「チームのため
すべてテストを書く」

「上達するため
練習を積む」



Software Craftsmanship: What it's all about.

<http://thecleancoder.blogspot.jp/2011/01/software-craftsmanship-what-is-all.html>

A dark, atmospheric forest scene. A single tree trunk in the foreground is brightly lit from the side, casting a long shadow on the ground. Behind it, a dense line of trees stands in the background, their trunks visible against a dark sky.

TDDは
道を照らす
明かり

質問:TDDを実際の仕事で使いますか?



yattom said...

- ・個人的には好き
- ・今ではできないことが多い
- ・使うときも100%ではない
知らない領域
試行錯誤

t-wada said...

- ・ミスったときや、悩んだとき
- ・テストは多く TDDとは限らない
- ・間合いを調節、制御するのに TDDを利用する
- ・色々なツールのひとつとして

太田健一郎 said...

徹底的にやり込んで熟練すると
使いどころがわかるようになった

Martin Fowler も同じことを
言った

t-wada said...

「TDDの
目的は
真の目的は
健康」



