

テスト駆動開発入門 ネクストステップ

井芹洋輝

TDD Boot Camp 東京 for C++
2011/10/8 @国立情報学研究所

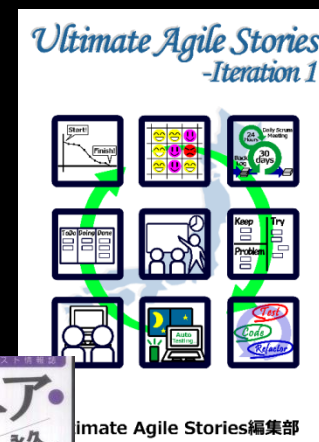
謝辞

- 主催の今給黎さん
- 和田さん、会場提供、スタッフの方々
- 参加者の皆さま

深くお礼申しあげます

自己紹介

- 井芹 洋輝(@goyoki/id:goyoki)
- 組み込みエンジニア
- WACATE実行委員/TDD研究会
- 講演/執筆：
 - XP祭り関西「ユニットテストの保守性を作りこむ」
 - Androidテスト祭り「テストの活用による開発効率化」
 - 並カン「FPGA/HDLを活用したソフトウェア並列処理の構築」等



概要

本講義はTDDの基本サイクルを学んだ方が対象です。

本講義ではTDDを開発で実践するための知識、TDDについて自立して学習を進めるための情報を学び、一人前のTDD使いへの道筋を明らかにします。

概要

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更
に
対処する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

実践のネクストステップ

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更
に
対処する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

実践のネクストステップ

TDDを継続していくと、TDDの基本サイクルにテストの再利用と変更のアクティビティが加わります。それらはしばしばTDDの効率を左右するため注意や工夫が必要です。

テストを整える

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更を
対処する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

テストを整える

TDDの継続においては定期的にテスト設計を見直し、**テストに穴がないか、これまでの作業が適切だったか**チェックする必要があります。

テストを整える

不適切なテストは

実装ミスの見逃し

デグレードの見逃し

テスト再利用の阻害

を誘発するリスクを持っています。

また乱雑なテストはテストコードを不必要に増大させ、テストの再利用コストを悪化させます。

テストを整える

テストの網羅度をチェック

- 仕様ベースの網羅

- テストが仕様を網羅しているか
- 仕様ベースのテスト設計等

4で割り切れる	N	Y	Y	Y
100で割り切れる	N	N	Y	Y
400で割り切れる	N	N	N	Y
うるうどし	N	Y	N	Y

- 構造ベースの網羅

- テストがコードを網羅しているか
- コードカバレッジ等

```
//うるう年か判定する
bool isLeapYear(unsigned int year)
{
    if (year % 400 == 0)
    {
        return true;
    }
    if ((year % 4 == 0) && (year % 100 != 0))
    {
        return true;
    }
    return false;
}
```

仕様ベースの網羅

ex)同値分割法によるチェック

- 出力やふるまいで同じように扱えるグループに、入出力をグルーピングする
- グループを同値クラスと呼ぶ

仕様ベースの網羅

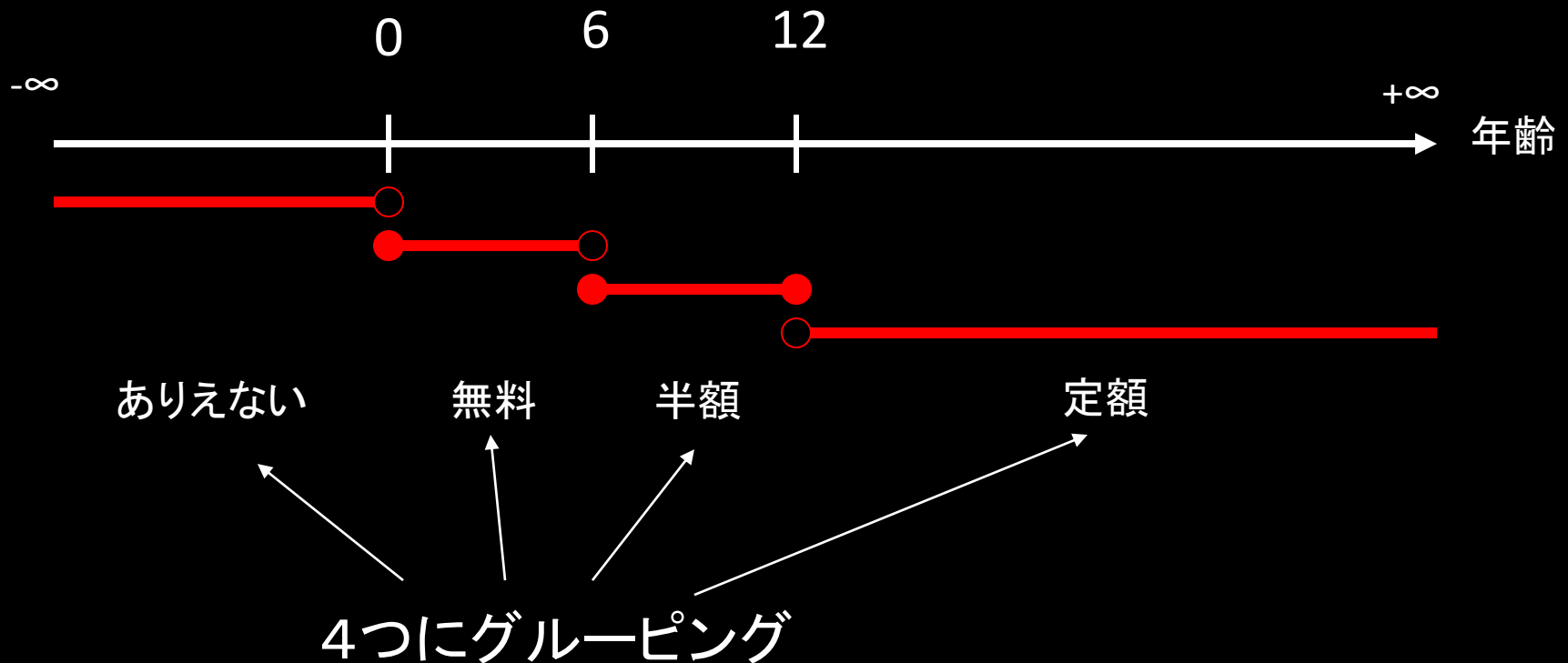
ex)同値分割法によるチェック

- 年齢が入力。割引率が出力
- 「6歳未満は無料。6歳以上12歳以下は半額。13歳以上は定額」

仕様ベースの網羅

ex)同値分割法によるチェック

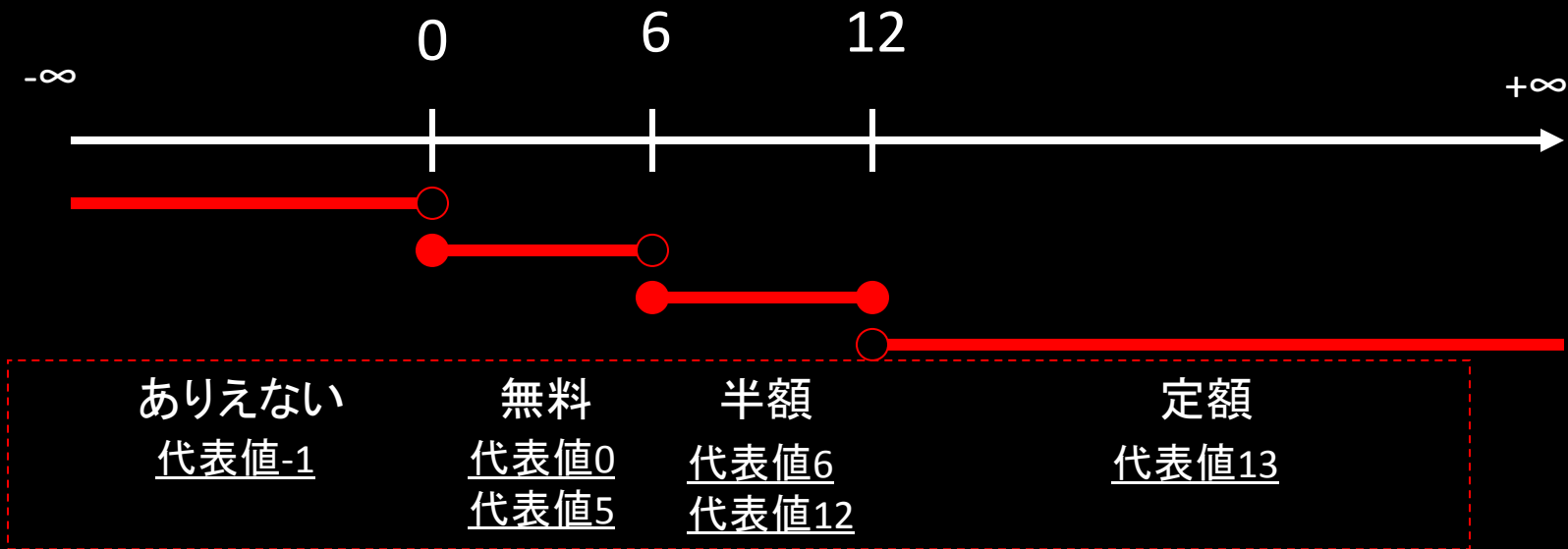
- 「6歳未満は無料。6歳以上12歳以下は半額。13歳以上は定額」
 - 出力をグルーピング



仕様ベースの網羅

ex)同値分割法によるチェック

- 「6歳未満は無料。6歳以上12歳以下は半額。13歳以上は定額」
 - 同値クラスごとに代表値を求め、テストの入力値に展開



代表値をテストの入力に指定

仕様ベースの網羅

ex)同値分割法によるチェック

```
TEST(HogeTest, Invalid) {  
    EXPECT_EQ(..., checkFee(-1))  
}  
  
TEST(HogeTest, Free) {  
    EXPECT_EQ(..., checkFee(0))  
    EXPECT_EQ(..., checkFee(5))  
}  
  
TEST(HogeTest, Half) {  
    EXPECT_EQ(..., checkFee(6))  
    EXPECT_EQ(..., checkFee(12))  
}  
  
TEST(HogeTest, Full) {  
    EXPECT_EQ(..., checkFee(13))  
}
```

テストコードが同値クラスや代表値を網羅しているかチェック
穴があれば埋める

あるいは
最初から意識してテストを書く

構造ベースの網羅

ex) ブランチカバレッジによるチェック

```
//うるう年か判定する
bool isLeapYear(unsigned int year)
{
    if (year % 400 == 0)
    {
        return true;
    }
    if (year % 4 == 0)
    {
        if (year % 100 != 0)
        {
            return true;
        }
    }
    return false;
}
```

構造ベースの網羅

ex) ブランチカバレッジによるチェック

```
//うるう年か判定する
bool isLeapYear(unsigned int year)
{
    if (year % 400 == 0)
    {
        return true;
    }
    if (year % 4 == 0)
    {
        if (year % 100 != 0)
        {
            return true;
        }
    }
    return false;
}
```

各々の条件分けのtrue/false

両方をテストが網羅しているか

チェック

不足があれば埋める

テストを整える

仕様ベース、構造ベースのテストのチェックを使い分けて適切なテストを構築しましょう。
規律あるテスト設計はコンパクトかつ網羅的なテストを生み出し、作業ミスの防止やテストの再利用向上をサポートします。

テストを整える

現場のプロダクトコードでは、複雑な組み合わせ/状態/ユニットテスト不能なコードなどテスト設計が困難な要素が溢れています。またTDDではテンポとスピードが要求されるため、様々なテスト設計手法を使い分け、呼吸するようにテスト設計の視点を活用できるように精進が必要です。

変更^①に備える

TDD実践のネクストステップ

テストを
整える

変更に
備える

変更^②に
対処する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

変更に対応する

TDDではリファクタリングやCIのテスト等を目的にしばしば**テストを再利用**します。また開発の進展や仕様変更に応じて、しばしば**テストの変更**も発生します。

変更に対応する

テストは変更・再利用を支える砦となりえますが、同時にテストは変更・再利用の障害ともなりえます。備えを怠れば、テストコードも保守困難なレガシーコードと化します。

変更に対応する

そのため、TDDの生産性確保の点で**テストに保守性は重要です**。柔軟な開発を支えるためにも、**プロダクト/テストを区別せずコードを洗練させる必要があります**。

変更に対応する

- 読みやすくする
- 危ないコードを分離する
- 重複をなくす
- 影響範囲を限定する/副作用をなくす

変更にも備える

- 読みやすくする
- 危ないコードを分離する
- 重複をなくす
- 影響範囲を限定する/副作用をなくす

読みやすくする

- 何をテストしているのかわかりやすい
- 変更箇所の特特定が楽
- 変更ミスを防げる
- テストのバグを見つけやすい
- テストが失敗したときに、バグがどこにあるか特定しやすい

読みやすくする

```
TEST(HogeTest, Test1)
```

```
{
```

```
    ...
```

```
}
```

```
TEST(HogeTest, Test2)
```

```
{
```

```
    ...
```

```
}
```

読みやすくする

```
TEST(HogeTest, Test1)
{
    ...
}
```

```
TEST(HogeTest, Test2)
{
    ...
}
```

なんのテストかわからない

読みやすくする

```
TEST(HogeTest, commandInputInvalidError)
{
    ...
}
```

```
TEST(HogeTest, commandInputBOFError)
{
    ...
}
```

適切な名前を与える

読みやすくする

```
TEST(HogeTest, Fuga)
{
    MotorStatus motorStatus(133, 232);
    InspectionFuga inspector;
    inspector.set(createMaintenanceInfo(motorStatus));

    EXPECT_EQ(START, inspector.getState());
    EXPECT_EQ(true, inspector.isEmpty());

    inspector.initialize();

    EXPECT_EQ(INFO, inspector.getState());
    EXPECT_EQ(false, inspector.isEmpty());
    ....
}
```

読みやすくする

```
TEST(HogeTest, Fuga)
```

```
{
```

```
    MotorStatus motorStatus(130, 232);
```

```
    InspectionFuga inspector;
```

```
    inspector.set(createMaintenanceInfo(motorStatus));
```

```
    EXPECT_EQ(START, inspector.getState());
```

```
    EXPECT_EQ(true, inspector.isEmpty());
```

```
    inspector.initialize();
```

```
    EXPECT_EQ(INFO, inspector.getState());
```

```
    EXPECT_EQ(false, inspector.isEmpty());
```

```
    ....
```

```
}
```

何をテストしているかが散漫
テストのバグをみつけにくい

読みやすくする

```
TEST(HogeTest, FugaConstructor) {  
    InspectionFuga inspector = createInspectionFugaDummy();  
  
    EXPECT_EQ(START, inspector.getState());  
    EXPECT_EQ(true, inspector.isEmpty());  
}
```

```
TEST(HogeTest, FugaInitialize) {  
    InspectionFuga inspector = createInspectionFugaDummy();  
    inspector.initialize();  
  
    EXPECT_EQ(INFO, inspector.getState());  
    EXPECT_EQ(false, inspector.isEmpty());  
}
```

分離し適切な名前を与える

変更にも備える

- 読みやすくする
- 危ないコードを分離する
- 重複をなくす
- 影響範囲を限定する/副作用をなくす

危ないコードを分離する

```
TEST(FooTest, Bar)
{
    MotorStatus motorStatus(0, 0);
    MaintenanceData mtData;
    MaintenanceType mtType(createRegionID(EU));
    setInitialData(mtData, mtType);
    ...

    InspectionFuga inspector;
    inspector.set(MaintenanceInfo(mtData, mtType), motorStatus);
    EXPECT_EQ(inspector...)
}
```

危ないコードを分離する

プロダクトコードに過依存

```
TEST(FooTest, Bar)
```

```
{
```

```
    MotorStatus motorStatus(0, 0);  
    MaintenanceData mtData;  
    MaintenanceType mtType(createRegionID(EU));  
    setInitialData(mtData, mtType);  
    ...
```

```
    InspectionFuga inspector;
```

```
    inspector.set(MaintenanceInfo(mtData, mtType), motorStatus);
```

```
    ...
```

```
}
```

その他:

変更リスクの高いコード

堅牢性の劣るコード

危ないコードを分離する [テスト側でラッピング]

```
TEST(FooTest, Bar)
```

```
{
```

```
    InspectionFuga inspector = CreateInspectionFuga(0, EU);
```

```
    ...
```

```
    inspector.set(MaintenanceInfo(mtData, mtType), motorStatus);
```

```
    ...
```

```
}
```

危ないコードを分離する

[プロダクト側のインターフェースを改善]

```
TEST(FooTest, Bar)
```

```
{
```

```
    InspectionFuga inspector(0, 0, EU);
```

```
    ...
```

```
    inspector.set(MaintenanceInfo(mtData, mtType), motorStatus);
```

```
    ...
```

```
}
```

変更に対応する

- 読みやすくする
- 危ないコードを分離する
- 重複をなくす
- 影響範囲を限定する/副作用をなくす

重複をなくす

[Test Utility Method]

```
TEST_F(BuyerTest, addSameStatus)
{
    Buyer buyer;
    Customer customer1("Taro", "Yamada", 15, 2, "HOGE|FUGA");
    customer1.addCategory(STATE_ACTIVE);
    Customer customer2("Taro", "Yamada", 15, 2, "HOGE|FUGA|HOGEHOGE");
    customer2.addCategory(STATE_ACTIVE);
    ...

    buyer.add(customer1);
    buyer.add(customer2);
    ...
    EXPECT_EQ(0, buyer.getSection());
}
```

重複をなくす

[Test Utility Method]

```
TEST_F(BuyerTest, addSameStatus)
{
    Buyer buyer;
    Customer customer1 = createCustomer("HOGE|FUGA");
    Customer customer2 = createCustomer("HOGE|FUGA|HOGEHOGE");
    ...

    buyer.add(customer1);
    buyer.add(customer2);
    ...
    EXPECT_EQ(0, buyer.getSection());
}
```

```
Customer createCustomer(string status) {
    Customer customer("Taro", "Yamada", 15, 2, status);
    customer.addCategory(STATE_ACTIVE);
    return customer;
}
```

Parameterized Creation Method

重複をなくす

[Parameterized Test]

```
TEST_P(HogeTest, InvalidValueMinus)
```

```
{  
    Hoge hoge(-1);  
    EXPECT_EQ(0, hoge.size());  
}
```

```
TEST_P(HogeTest, InvalidValueZero)
```

```
{  
    Hoge hoge(0);  
    EXPECT_EQ(0, hoge.size());  
}
```

```
TEST_P(HogeTest, InvalidValueTooBig)
```

```
{  
    Hoge hoge(124566);  
    EXPECT_EQ(0, hoge.size());  
}
```

```
...
```

重複をなくす

[Parameterized Test]

```
class HogeTest : public testing::TestWithParam<int>
{
};
```

```
INSTANTIATE_TEST_CASE_P(InvalidValueInstance, HogeTest,
    testing::Values(-1, 0, 124566));
```

```
TEST_P(HogeTest, hogehoge)
{
    Hoge hoge(GetParam());
    EXPECT_EQ(0, hoge.size());
}
```

Parameterized Test

変更にも備える

- 読みやすくする
- 危ないコードを分離する
- 重複をなくす
- 影響範囲を限定する/副作用をなくす

影響範囲を限定する/副作用をなくす

```
Foo foo;
```

```
TEST_F(HogeTest, Fuga)
```

```
{
```

```
    ...
```

```
}
```

```
TEST_F(HogeTest, Piyo)
```

```
{
```

```
    ...
```

```
}
```

影響範囲を限定する/副作用をなくす

```
TEST_F(HogeTest, Fuga)
```

```
{
```

```
    Foo foo;
```

```
    ...
```

```
}
```

```
TEST_F(HogeTest, Piyo)
```

```
{
```

```
    Foo foo;
```

```
    ...
```

```
}
```

ローカル変数にする
テストクラスのメンバにする

影響範囲を限定する/副作用をなくす

```
Void SetUp()  
{  
    外部コンポーネントの初期状態を記録する  
}  
  
TEST_F(Buyer, test_add_sameStatus)  
{  
    外部コンポーネントを使ってテスト  
    ....  
}  
  
Void TearDown()  
{  
    外部コンポーネントを初期状態にロールバックする  
}
```

構造的にも時間軸的にも独立させる

他のテストコードを変更しても結果が変わらない

順序を変えても、どのようなタイミングでも結果が変わらない

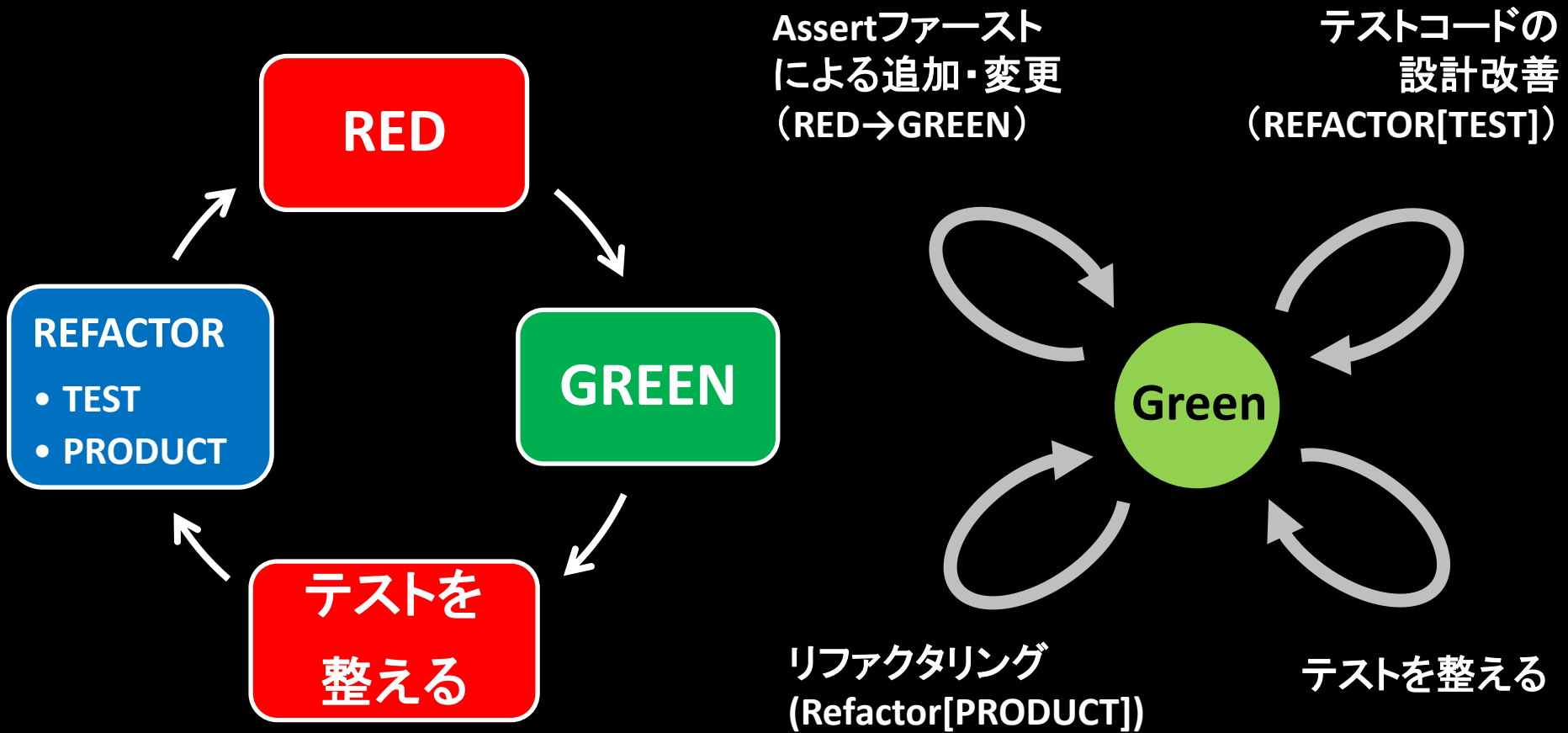
テストを整える&変更に備える 実施タイミング

ユニットテストの実装はプログラミングそのものです。プロダクトコードもテストコードも区別せず設計・記述を洗練させていきましょう。
プロダクトコードのリファクタリングと同じ扱いでテストコードも設計改善していきましょう。

テストを整える & 変更に備える 実施タイミング



テストを整える & 変更に備える 実施タイミング



変更に対処する

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更
に
対応
する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

変更に対処する

プログラミングでは、リファクタリング、仕様変更などによりしばしばプロダクトコードの変更が発生します。

TDDではプロダクトコードに依存するテストが早期から作られるため、**テストを如何に効率よく変更に対応させるかは生産性確保の鍵**となりえます

変更に対処する TDDの4つのモード

1. THINK

(1.5. 変更を受け入れられるように設計改善)

2. RED

3. GREEN

4. REFACTOR

...

変更に対処する

```
Class TestTarget {  
    void TestTarget(int hoge) {  
        ....  
    }  
};
```

```
TEST(...)  
{  
    TestTarget target(0);  
    ...  
}  
TEST(...)  
{  
    TestTarget target(1);  
    ...  
}  
....
```

変更に対処する

```
Class TestTarget {  
    void TestTarget(int hoge) {  
        ....  
    }  
}
```

```
TEST(...)  
{  
    TestTarget target(0);  
    ...  
}  
TEST(...)  
{  
    TestTarget target(1);  
    ...  
}  
....
```

TestTarget(int hoge) から TestTarget(int hoge, int fuga) に変更。
Int fuga に応じて複雑な処理を...

変更に対処する

```
Class TestTarget {  
    void TestTarget(int hoge) {  
        ....  
    }  
}
```

```
TEST(...)  
{  
    TestTarget target(0);  
    ...  
}  
TEST(...)  
{  
    TestTarget target(1);  
    ...  
}  
....
```

Think !

無理のない小さなステップで
効率よく変更できるように

TestTarget(int hoge) から TestTarget(int hoge, int fuga) に変更。
Int fuga に応じて複雑な処理を...

変更に対処する[1]

Parallel Change

```
Class TestTarget {  
    void TestTarget(int hoge) {  
        ...  
    }  
    void TestTarget(int hoge, int fuga) {  
        ...  
    }  
}
```

新旧共存でTDDを進める
逐次テストを置き換え、古いコードを削除

```
TEST(...)  
{  
    TestTarget target(0);  
    ...  
}  
TEST(...)  
{  
    TestTarget target(1);  
    ...  
}  
....  
(新しいコードに対するテスト)
```

TestTarget(int hoge) からTestTarget(int hoge, int fuga)に変更。
Int fugaに応じて複雑な処理を...

変更に対処する[1]

Parallel Change

```
Class TestTarget {  
    void TestTarget(int hoge) {  
        ...  
    }  
    void TestTarget(int hoge, int fuga) {  
        ...  
    }  
}
```

新旧共存でTDDを進める
逐次テストを置き換え、古いコードを削除

```
TEST(...)  
{  
    TestTarget target(1);  
    ...  
}  
....  
(新しいコードに対するテスト)
```

TestTarget(int hoge) からTestTarget(int hoge, int fuga)に変更。
Int fugaに応じて複雑な処理を...

変更に対処する[2]

インターフェース設計の先行

```
Class TestTarget {  
    void TestTarget(int hoge, int fuga) {  
        ...  
    }  
}
```

仮実装によりインターフェースを先行して変更
次にTDDで仮実装を本実装に置き換える

```
TEST(...)  
{  
    TestTarget target(0, 0);  
    ...  
}  
TEST(...)  
{  
    TestTarget target(1, 0);  
    ...  
}  
....
```

TestTarget(int hoge) からTestTarget(int hoge, int fuga)に変更。
Int fugaに応じて複雑な処理を...

変更に対処する

プロダクトコード変更時は、テストは変更のサポートにも、変更の障害にもなりえます。テストの保護を活かしつつ、無理のない小さなステップで、効率よく変更できる道を考えていきましょう。

番外

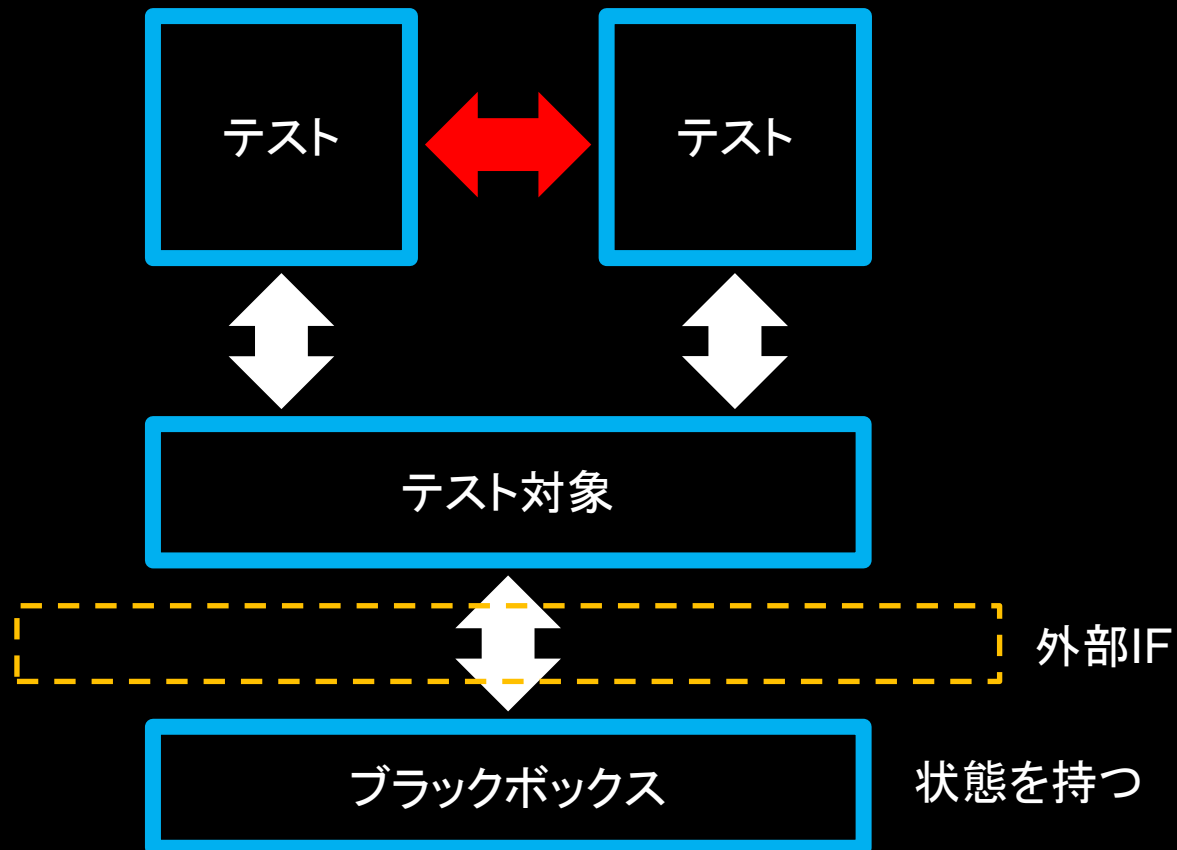
テスト困難なコードに対応する

TDDの活用で変更対応と同じく課題となるのが、**テスト困難なコードへの対応**です。

- 実行が困難
- 操作不能な状態を持つコンポーネント
- 時間やタイミングに依存
- 出力の再現性が低い
- 過剰なカプセル化

テスト困難なコードに対応する

ブラックボックスを介してテストが結合



目指すべきテスト

- 構造的、時間的に独立している
- 自己完結している
- 結果の再現が容易にできる

テスト困難なコードに対応する

テスト困難なコードに対しては以下の対応を行います

- 影響範囲を抑え、かつテスト可能なコードで置換可能にする
- バグドアやテスト容易性に優れたインターフェースを確保する

テスト困難なコードに対応する

```
void CameraCtrl::hoge()
{
    _motorCtrl.run();
    ...
}

void CameraCtrl::fuga()
{
    _motorCtrl.stop();
    ... 外部コンポーネントを制御
}
```

テスト困難なコードに対応する

```
void CameraCtrl::CameraCtrl()
{
    _motorCtrl.open(...);
    ...
}
```



```
void CameraCtrl::CameraCtrl(MemoryCtrl memoryCtrl)
{
    _motorCtrl = motorCtrl;
    _motorCtrl.open(...);
    ...
}
```

学習のネクストステップ

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更
に
対処する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

TDDを学ぶ

TDDを外部から学ぶ手段として文献、情報発信源、コミュニティがあります。

TDDの原則はシンプルですが、様々な関連分野、応用分野とリンクしているため、勉強の余地が大いに存在します。

基礎を身につける

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更を
対処する

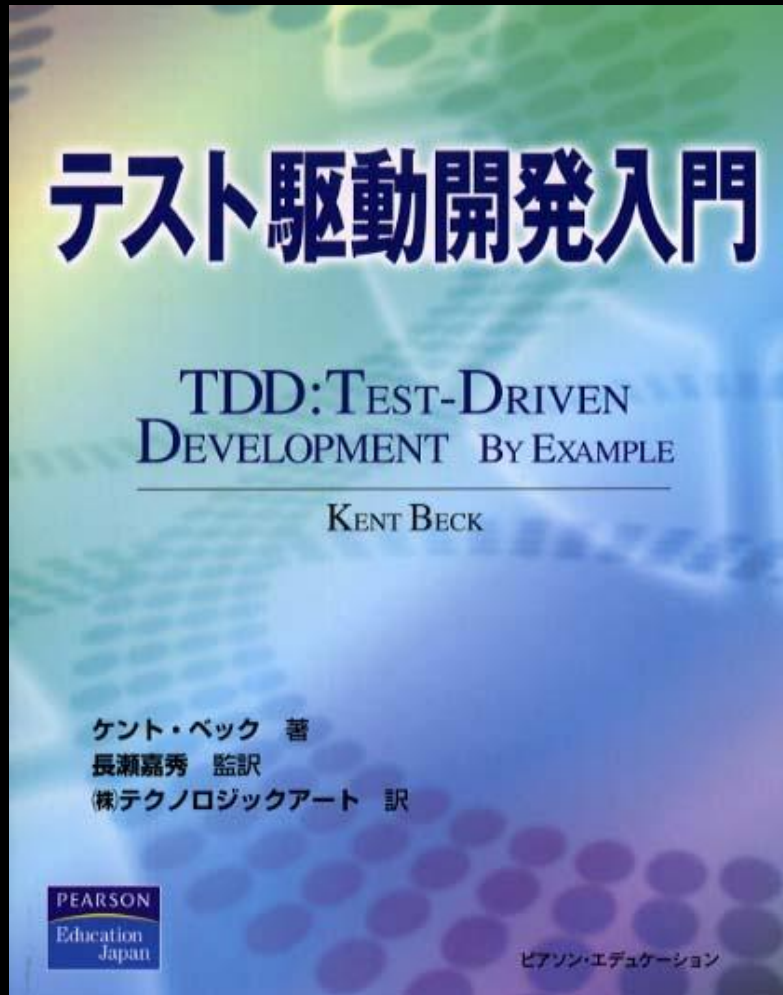
TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

基礎を身につける



『[動画で解説] 和田卓人の“テスト駆動開発”講座』
<http://gihyo.jp/dev/serial/01/tdd/>

全20回すべて動画付き解説
ニコニコ動画でも見れます



WEB+DB過去記事の特設サイトと動画も



WEB+DB PRESS Vol.37 特集 特設ページ 実演! リファクタリング



@t_wada
Id:t-wada

TDDBC横浜 (募集中)

<http://kokucheese.com/event/index/18180/>



Id:setoazusa
@setoazusa

基礎を身につける

TDDはプログラミングの技能です。実際に手を動かして鍛えましょう。

良質なTDDの題材を使って写経しましょう

より活用する

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更
に
対処する

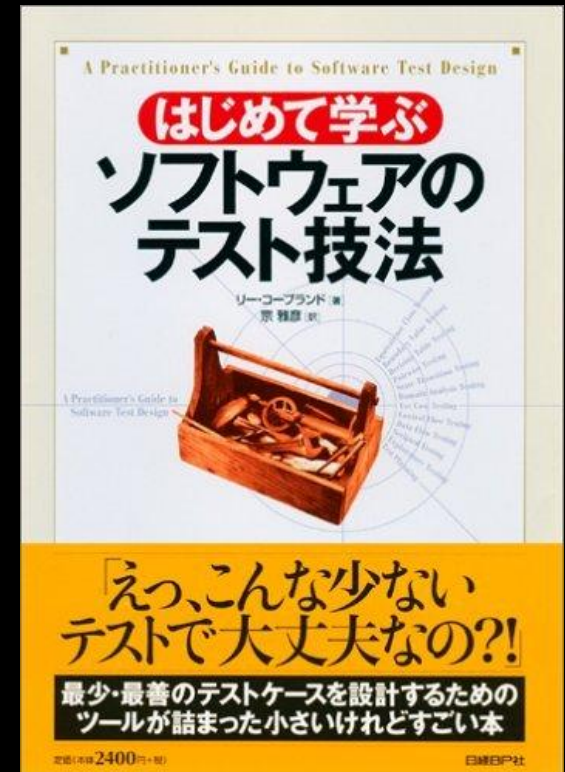
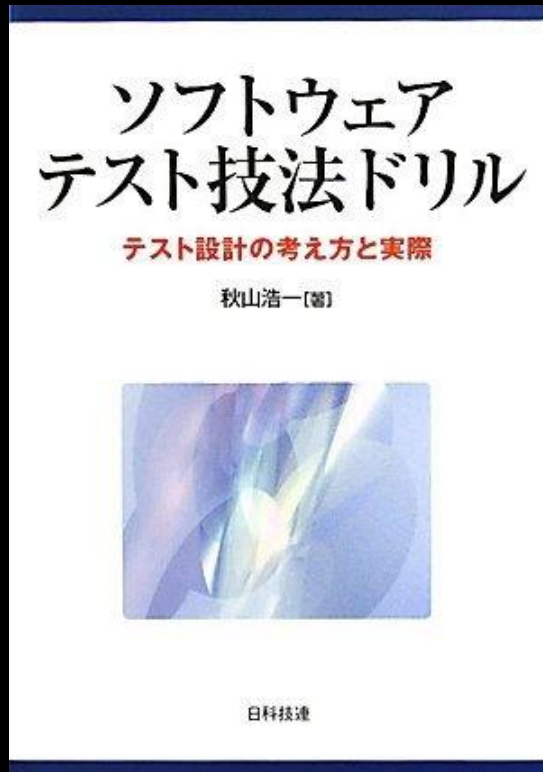
TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

より活用する テスト設計



Testing Engineer's Forum

<http://www.swtest.jp/wiki/index.php?swtest.jp/wiki/forum>

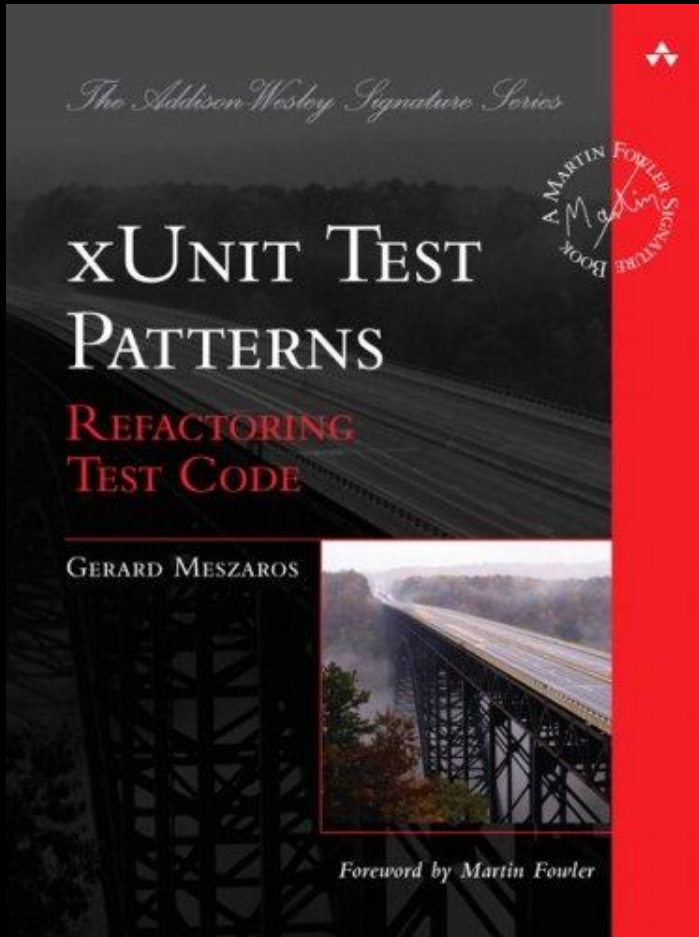
ソフトウェアテスト技法ドリル勉強会(第7回募集中)

<http://atnd.org/events/20302>



@softest

より活用する テストコードの実装

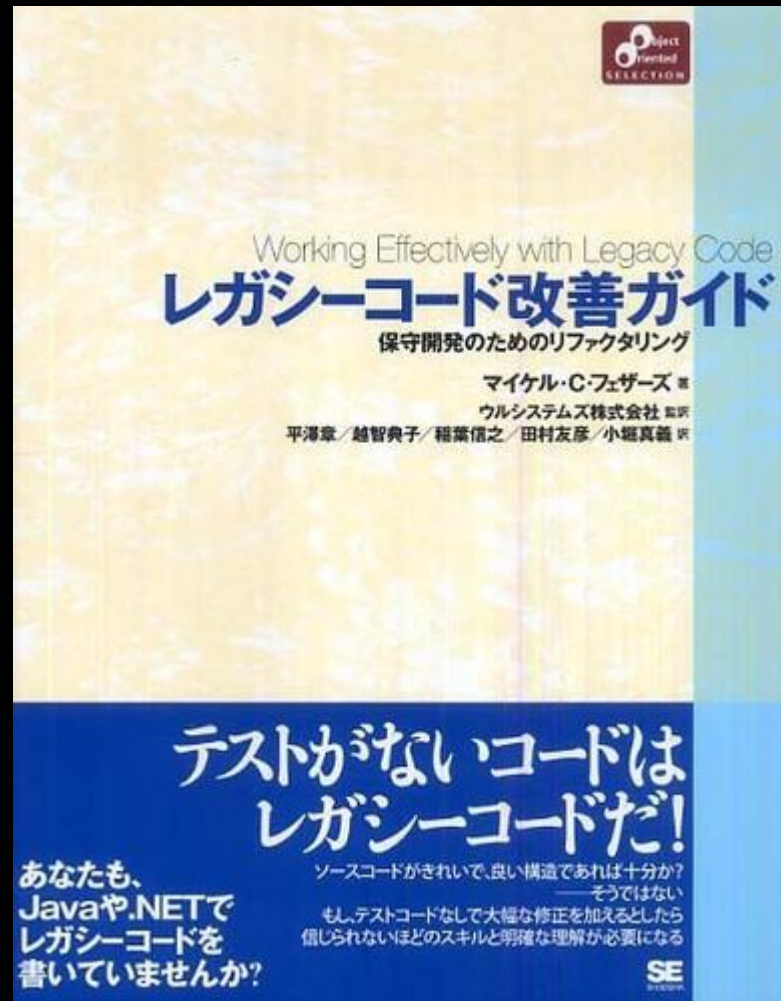


xUnit Test Patterns読書会Wiki
<http://www.fieldnotes.jp/xutp/>



Id: setoazusa
@setoazusa

より活用する
変更への対処 &
テスト困難な対象への対応



応用分野を学ぶ

TDD実践のネクストステップ

テストを
整える

変更
に
備える

変更
に
対処する

TDD学習のネクストステップ

基礎を
身につける

より
活用する

応用分野
を学ぶ

応用分野を学ぶ テストの活用×TDD

品質と信頼性という付加価値を実践するテスト情報誌

ソフトウェア・ テストPRESS 永久 保存版!

総集編

全10号分を
大収録!

プレミアム
CD-ROM

- iPad・Androidタブレットでも快適!
- PCからは全文一括検索対応!
- 1冊1ファイルの記事PDF

表紙
企画 バックナンバー目次と座談会を振り返る
**ソフトウェア・テストPRESS
10号の軌跡**

特集1 JSTQB Advanced Level対応!
テストマネジメント実践入門

特集2 プログラミングにテストで貢献!
テストエンジニア
のための **テスト駆動開発入門**

● 本日復活!!ソフトウェアテスト技法道場・総集編



WACATE(もうすぐWACATE2011冬 募集開始!)
<http://wacate.jp/>

Testing Engineer's Forum
<http://www.swtest.jp/wiki/index.php?swtest.jp/wiki/forum>

応用分野を学ぶ CI×TDD



日本Jenkinsユーザの会
<http://build-shokunin.org/>

応用分野を学ぶ DVCS×TDD

 Id:bleis-tift
@bleis

SCM Boot camp(第2回募集中)

http://d.hatena.ne.jp/kyon_mm/archive?word=%5Bscmbc%5D

<http://groups.google.com/group/scmbootcamp?hl=ja>



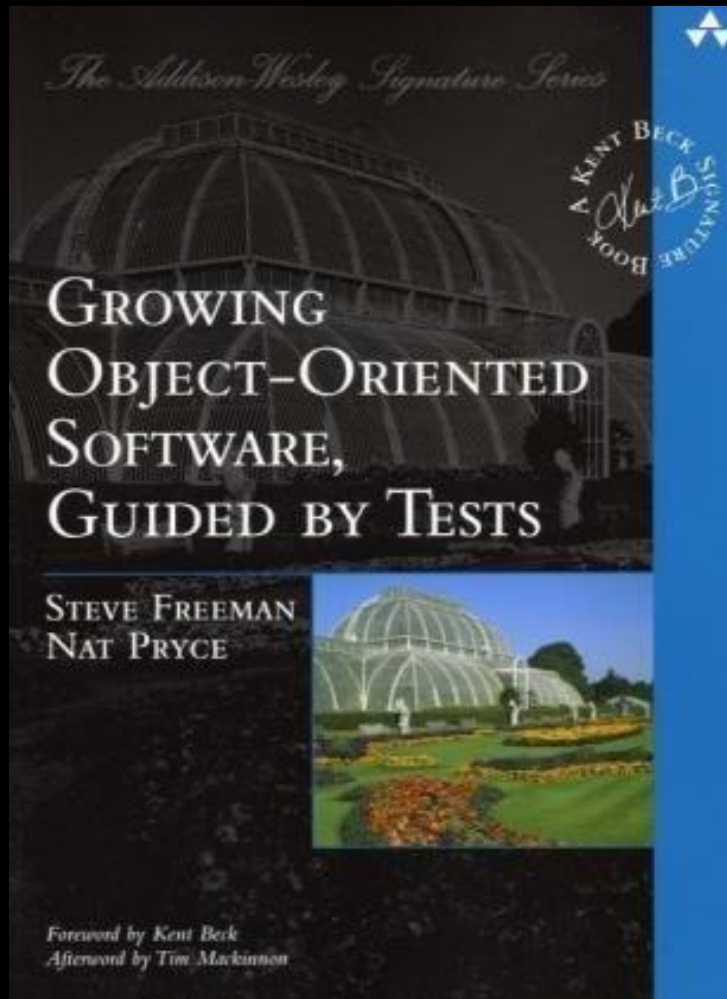
Id:kyon_mm
@kyon_mm



Id:pocketberserker
@pocketberserker

応用分野を学ぶ

BDD/Outside-In TDD



Growing Object-Oriented Software,
Guided by Tests(goos)読書会
(第4回募集中)

<http://devtesting.jp/goos/>



Id: setoazusa
@setoazusa

その他

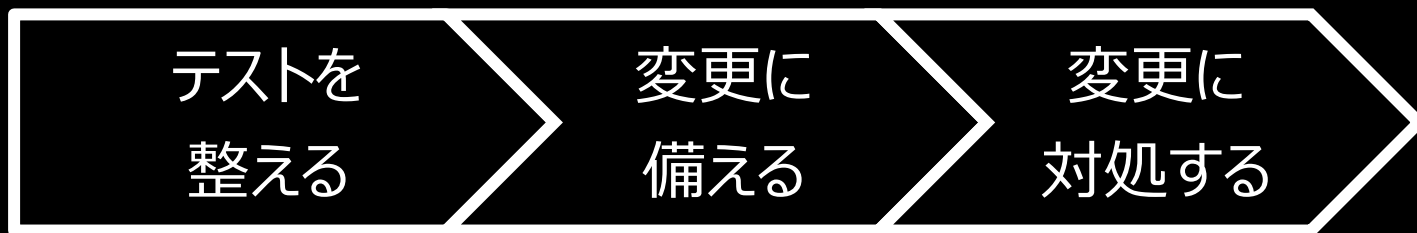
TDDBC運営コミュニティ

TDDBC

<http://devtesting.jp/tddbc/>

ご清聴ありがとうございました

TDD実践のネクストステップ



TDD学習のネクストステップ

