ITCollege

無断転載、転用禁止

# ソフトウェアテスト技法

品質の高いソフトウェアとは何か

Ver. 1.0

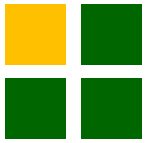


copyright ©ITCollege All Rights Reserved

# アジェンダ

- コンピュータプログラムの欠陥
- バグの現状
- じゃあ仕様ってどう決めるの？
- 品質が良いとは？
- ユーザ要求と品質
- 品質の観点
- ソフトウェア開発の現状
- テスト技術の必要性
- プログラムを作る時に・・・
- テストを行う前に・・・
- ブラックボックステスト
- ホワイトボックステスト
- テスト工程
  - 単体テスト
  - 単体テスト自動化
  - リファクタリング
- 結合テスト
  - 結合テスト自動化
  - 結合テスト実施時
  - 結合テスト結果
  - ドキュメント
- まとめ

ITCollege copyright ©ITCollege All Rights Reserved



# コンピュータプログラムの欠陥

- **バグ**

  コンピュータプログラムの誤りや欠陥をバグといいます。

  バグが発生したプログラムでは、製作者が意図しない結果が現れたり、ときにはコンピュータを暴走させたり停止させたりすることもあります。

  バグが原因で、ユーザに不利益を与えたりしてしまうことがあるため、バグが発生した場合は即座に修正することが求められます。

- **デバッグ**

  バグを取り除くことをデバッグといいます。

  正常な動作をするソフトウェアを作るためには、もっとも重要な要素のひとつと言われています。

  Javaでは例外処理で異常な動作の発見が容易であったり、デバッガと呼ばれるツールが豊富なため、バグを0に近づけるようにしていくことが大切です。


copyright ©ITCollege All Rights Reserved

# バグの現状

- **バグってなんだろう・・・**

バグは「不具合」と呼ばれたり「瑕疵（かし）」と呼ばれたりと色々ありますが、何を基準としてバグと言うのでしょうか。

コンピュータが停止する、暴走する等はわかりやすいバグです。

しかし、以下のような動作はバグでしょうか

1. **個人情報取り扱い同意画面がログイン前に表示される**
2. **ゲームにおいてユニークなアイテムが売却できる**
3. **スマートフォンアプリで、カメラのシャッターが2回押される**

おそらく、「場合による」であったり「そういうシステムならOK」で片付いてしまうような内容ですね。

つまり仕様に左右されるわけです。


copyright ©ITCollege All Rights Reserved

# じゃあ仕様ってどう決めるの？

- **なぜそのシステムを作るのか**

ユーザは、そのシステムを使って何をしたいのかを考えることが重要です。

単純にシステムを作ることが目的ではなく、システムを使って日々の生活が便利になることが目的です。

## ユーザが便利になるような機能＝仕様

よく言われている「ユーザ目線」という言葉がありますが、「ユーザが使ったときの目線」という意味に内包して「ユーザが使った時、日常の業務や生活が効率的になっているか」という部分も考えていきましょう。

- **仕様通りかどうかはテストでチェック**

システムがある程度完成してくると、テストを行います。

作られたシステムが仕様に沿ったものになっているかを確認するわけです。

テストをして、バグがあるかどうかを確認し、見つかれば修正というサイクルを繰り返し、品質を高めていくわけです。


copyright ©ITCollege All Rights Reserved

# 品質が良いとは？

- **品質とはユーザの満足度**

絶対に止まらないシステム、絶対に間違えないシステムであっても、使い勝手が悪いシステムは使われません。

では、その使われないシステムは品質が高いと言えるのでしょうか。

近年では「ユーザビリティ」という言葉が当たり前になってきているように、使い勝手、効率、有効さというものも、品質を測る大切な要素です。

## 品質とは、ユーザの満足度である

- ✔ 手頃な価格
- ✔ 処理速度が速い
- ✔ 使い勝手が良い
- ✔ 他社製品との互換性がある

など


copyright ©ITCollege All Rights Reserved

# ユーザ要求と品質

- **3つの要求**

| 基本要求 | 変動要求 | 潜在要求 |
| --- | --- | --- |
| 製品として成り立つには必要な機能 | あると満足し、なければ不満に感じる要求 | あると満足し、なければ「仕方がない」と思う要求 |

- **3つの品質**

| 当たり前品質 | 一元的品質 | 魅力的品質 |
| --- | --- | --- |
| 最低限のラインとして必要な品質 | ユーザの評価が上がり、十分な満足度を得られる | 魅力的な付加価値を持ち、ユーザの期待値を上回る |

ITCollege copyright ©ITCollege All Rights Reserved

# 品質の観点

- **優先度、QCDのバランスを考える**

  QCDとは「Quality：品質／Cost：コスト／Delivery：納期」を略したもので製造業で使われる言葉です。

  開発しているシステムとして、これらのバランスをいかに取るかが重要です。

- **ISO9126品質特性**

| | |
|---|---|
| **機能性** | 必要な機能が実装されていること |
| **信頼性** | 機能が正しく動作し続けられること |
| **使用性** | 利用者に使いやすく魅力的であること |
| **効率性** | 資源の量に対比して適切な性能であること |
| **保守性** | 維持、変更が容易であること |
| **移植性** | 別環境に移しやすいこと |


copyright ©ITCollege All Rights Reserved

# ソフトウェア開発の現状

プログラム量の増加
高機能化、多機能化

開発期間の短縮
人的資源の不足
検証ノウハウの不足

テストに人回せないよ
実装も終わってないのに

ITCollege copyright ©ITCollege All Rights Reserved

# ソフトウェア開発の現状

プログラム量の増加
高機能化、多機能化

開発期間の短縮
人的資源の不足
検証ノウハウの不足

## ソフトウェアの品質低下

テストに人回せないよ
実装も終わってないのに

ITCollege copyright ©ITCollege All Rights Reserved

# テスト技術の必要性

- **KKD法の横行**

昨今のソフトウェアは、多機能化や高機能化によりプログラミングの量が飛躍的に増加しています。

また、IT技術の普及によりシステム開発会社への要求も高くなってきています。

ソフトウェアを開発する上で、品質を担保することは最も重要な要素ではありますが、工数や技術的に軽視されがちな側面があります。

そのため、テストの方法などが確立されず、人によって品質にばらつきが出てしまうことも多くあります。

KKDとは「勘・経験・度胸」を根拠にするという考えで、バラツキ、想定外事例に対する対処、問題点の根本的解決などといった面での弊害が懸念されています。

そのためには、標準化された手法が必要なのです。


copyright ©ITCollege All Rights Reserved

# テスト技術の必要性

- **標準化された手法**

「標準化された手法」とはなんなのでしょうか？

テストという工程は実施するタイミングが1つというわけではありません。

**・プログラムを作成時に行うテスト**

**・プログラムが完成時に行うテスト**

**・プログラムをユーザ様に提供する状態と同じ環境にして行うテスト**

など何度も工程に合わせて都度確認することで品質を高めていきます。

それぞれの状況や工程において手法が用意されているというわけです。

ではそれら、テスト手法を次のページから見ていきましょう。

ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **静的テスト**

静的テストとは、テスト対象を動作させずに行うテストです。

目的としては、

**・ソースコード可読性の向上**

**・潜在バグの可能性となるコードのチェック**

になります。

主な進め方としては、

**1.コーディング規約に従ってコーディング**

**2.静的解析ツールによるチェック**

**3.ソースコードレビュー**

です。それでは次のページからもう少し詳細を見ていきましょう。

ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **コーディング規約に従ってコーディング**

コーディングは、多くのプログラマが開発する時に好き勝手にコーディングすると統一性がなく保守しづらいソースコードができあがってしまいます。これは後で問題が発生した時にソースを見直す作業が非常に大変になります。

そういったことを防ぐためにソースコードの書き方に対する規則、すなわち**コーディング規約**を作りすべての作業者に守らせるようにします。

```
public class taiyaki{
  public final String Taiyaki=“たいやき”;
  public static void main(String[] args){
    taiyaki taiyaki = new taiyaki();
    taiyaki taiYaki = taiyaki;

    System.out.print(taiyaki.Taiyaki);
  }
}
```



copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **コーディング規約**

コーディング規約を最初から作成するのは大変です。他プロジェクトやWeb上から参照できる規約を流用してカスタマイズすると良いでしょう。

作成する観点としては、

○**命名規則**　　（例：クラス名は先頭を大文字で記載　など）

○**コメントの記述**（例：メソッドは@param,@returnを必須　など）

○**コーディングスタイル**　（例：while文は必ず中括弧を記載　など）

○**性能低下につながるコード**　（例：synchronizedの使い方　など）

が考えられます。

しかし、規約があまりに多すぎるとプログラマが把握しきれない可能性があるため、優先度をつけて高いものを選定しましょう。

ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **静的解析ツールによるチェック**

コーディング規約をすべて理解して厳密にコーディングすることは、経験の長いプログラマでも難しいことです。そこで活躍するのがコーディング規約に従っているかどうかを自動的にチェックするツール類です。

今回は無償で使えるツールを紹介します。

| 項目 | Checkstyle | FindBugs |
|---|---|---|
| 解析対象 | ソースコード(.java) | バイトコード(.class) |
| ルール数 | 約120 | 約300 |
| 用途 | コーディングスタイルのチェック | 潜在バグの検出 |

用途にも記載があるように得意分野が異なるツールとなっています。合わせて利用することでコーディング規約に対応したチェックを網羅的に実施できます。

Pleiadesをダウンロードしている場合、標準装備されているため、ぜひ使ってみましょう。

ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **ソースコードレビューを行う**

ソースコードレビューとは開発者本人や開発チームの担当者などがソースコードを閲読し、可読性や潜在バグの兆候を読み取る作業です。読み取った情報を開発作業者にフィードバックし、ソースコード修正を行います。

フィードバック内容として記録すべき内容は、

**・いつ誰がレビューしたか**

**・どの範囲のどの時点のソースコードが対象か**

**・コーディング規約を逸脱している箇所はどこか**

**・仕様から逸脱した動作箇所はどこか**

**・コードのどの部分にどのような脆弱性の疑いがあるか**

レビューにおいては直すための対策を検討するというよりは、問題をなるべく多く指摘することに注力すると良いとされます。

ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **コードレビューのやり方**

  では実際にコードレビューを行う時にどのような作業があるのでしょう？

  以下に作業工程を記載します。

  1.下読み　　:どこにどのようなソースがあるのか把握

  2.成熟度点検　:ある程度の品質水準に達しているか否か評価

  3.機能点検:機能面の不具合(バグ)の検出

  4.脆弱性点検　:セキュリティ脆弱性の検出

  上記のような段階を踏むことで漏れが少なくなると言われます。

  各作業についてもう少し見ていきましょう。


copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

## 1.下読み

軽くプログラムを読み、以下情報の把握に努めましょう。可能なら設計時に作成したドキュメントも合わせて参照します。

- 構成　：対象はどのような構成に分割され、配置されているか
- 記述対象　：新規作成、修正したソースファイルはどれか
- モジュール間の関係　：各モジュールがどのモジュールを参照しているか
- データ構造　：データはどのような構造をもっているか
- 制御フロー　：どのような制御フローをもっているか
- データとオブジェクトの関係

　：どのデータにアクセスしているか、オブジェクトにどのようなメソッドが集められているか

ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

**2.成熟度点検**

下読みした結果、「コードが雑に書かれたもの」であるか、「十分に考慮を行いつつ書かれた」ものであるかを判断します。著しく水準を下回る場合、ソースコードの改善をプログラマに求めます。

以下に点検項目の例を記載します。

・命名規則が守られていること

・記号名の命名が適切であること

・ひとつのメソッドの行数が多すぎない、かつ機能分割が細かく行われていること

・ループが有限回数で停止すること

・外部から取り込むデータすべてにデータチェックを施していること

などです。


ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

**3.機能点検**

書かれたコードが仕様を果たしているか否かを点検します。仕様をあまりにも満たしていない場合、ソースコードの改善をプログラマに求めます。

**4.脆弱性点検**

セキュリティ対策を行っているかを確認します。SQLインジェクション対策やXSS(クロスサイトスクリプティング)対策※などを施しているか確認します。

※SQLインジェクション対策はDBCプログラミング、クロスサイトスクリプティング対策は、Webプログラミング時に説明します。


ITCollege copyright ©ITCollege All Rights Reserved

# プログラムを作る時に・・・

- **コードレビュー5か条**

では実際にレビューをする際にレビュアー(実施者)はどのような点に注意すべきでしょう？以下に5つまとめました。ぜひ意識してレビューを実施してください。

**①事前に指摘事項はある程度まとめておきましょう。(レビュアー)**

**②コードレビューの観点を決め、チェックリストを用意しておく。**

**③関連メンバに情報共有(記録と数値化)**

**④全否定はしない**

**⑤褒める**

特に④と⑤を徹底しないとレビュー嫌いになってしまいます。率先してレビューをし合える空気づくりをチームで行っていきましょう。


copyright ©ITCollege All Rights Reserved

# テストを行う前に・・・

- **テストケース**

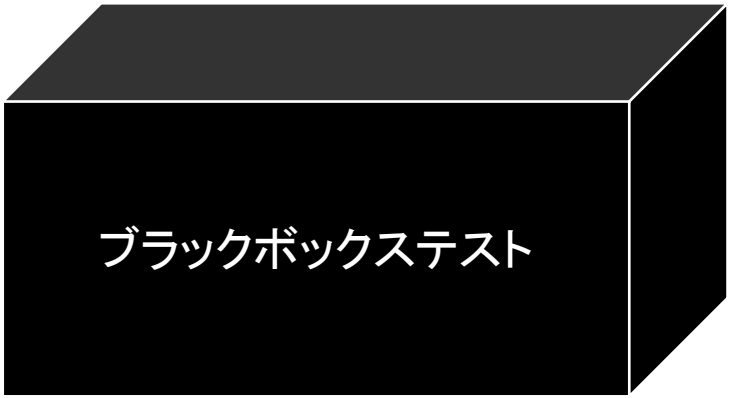
テストを実行するには、

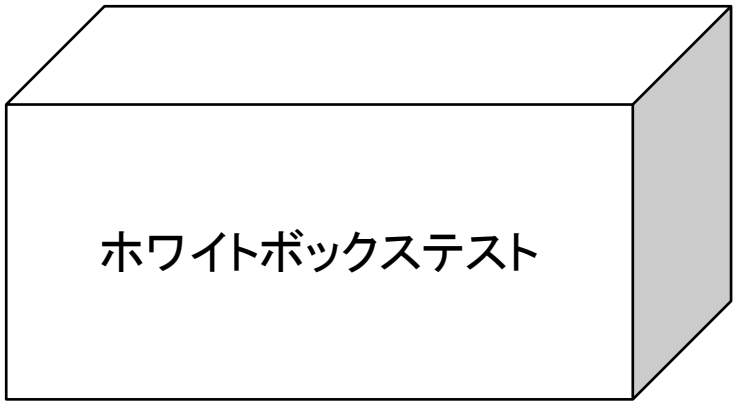
・テスト内容

・実施回数

・確認内容

をまとめた**テストケース**を作る必要があります。このテストケースを作るための技法として大きく2種類あります。それが以下の**ブラックボックステスト**と**ホワイトボックステスト**です。




ITCollege copyright ©ITCollege All Rights Reserved

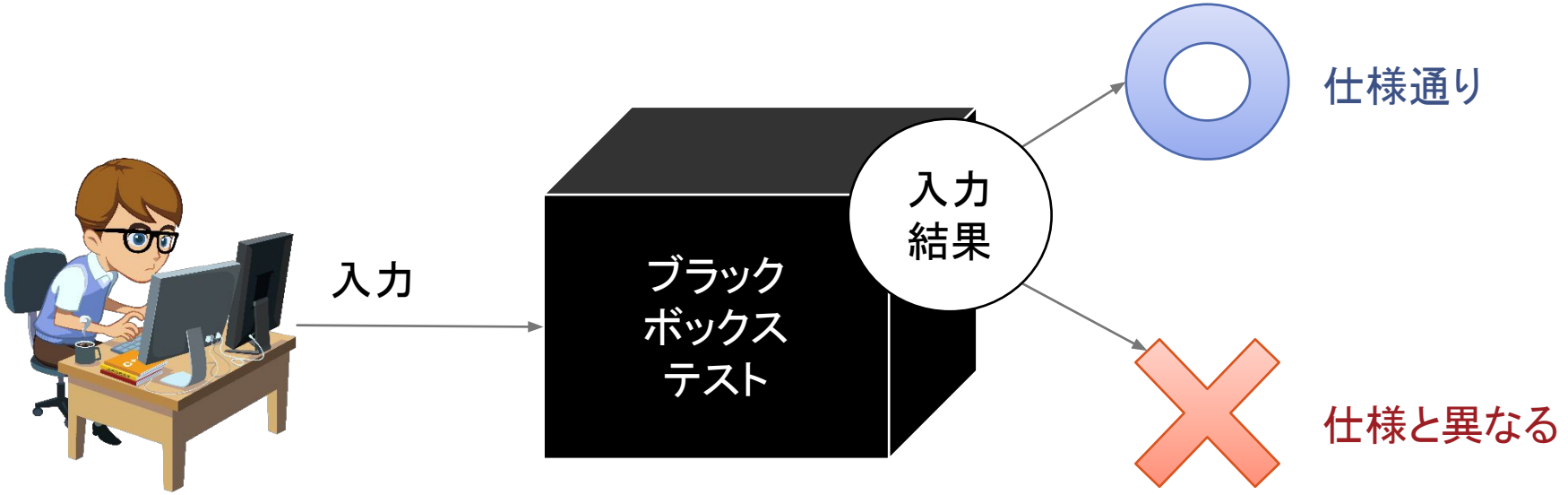
# ブラックボックステスト

- **概要**

テスト対処となる部分を「中の見えない箱(ブラックボックス)」として捉えたテスト方法です。主にテスト対象に入力を与えて、実行された結果が仕様と合っているかどうかを確認します。

重要なことは、このテスト中はテスト対象のプログラムがどのような処理が行われているかは気にしません。

テスト対象の「仕様」に基づいたテストケースの作成技法になります。



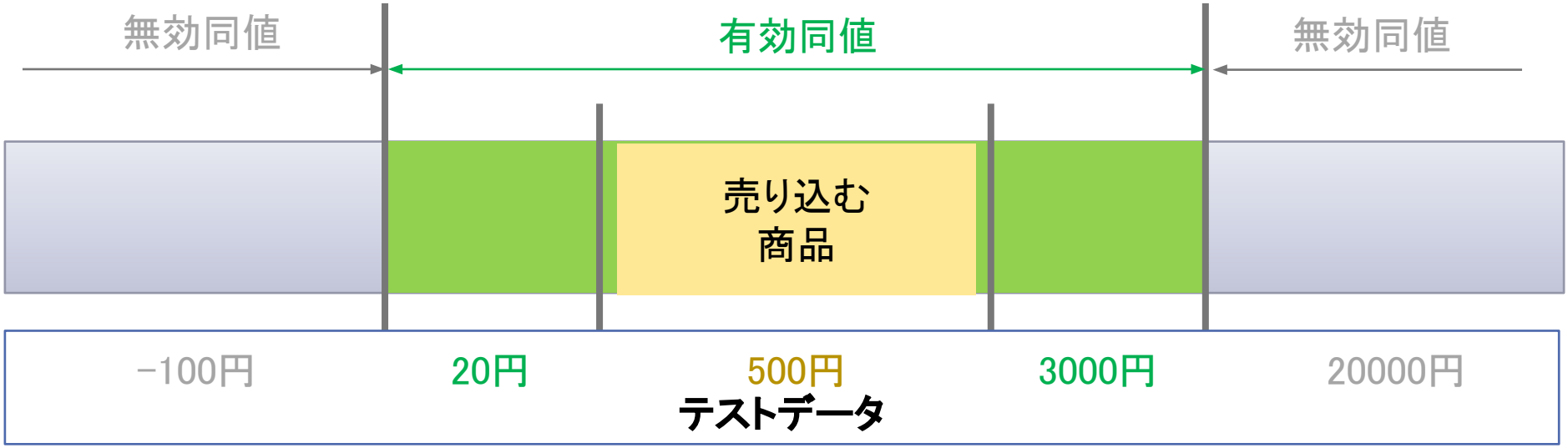

ITCollege copyright ©ITCollege All Rights Reserved

# ブラックボックステスト

- **種類**

ブラックボックステストを行う際に重要な方法として「同値分割法」と「境界値分析法」というものがあります。

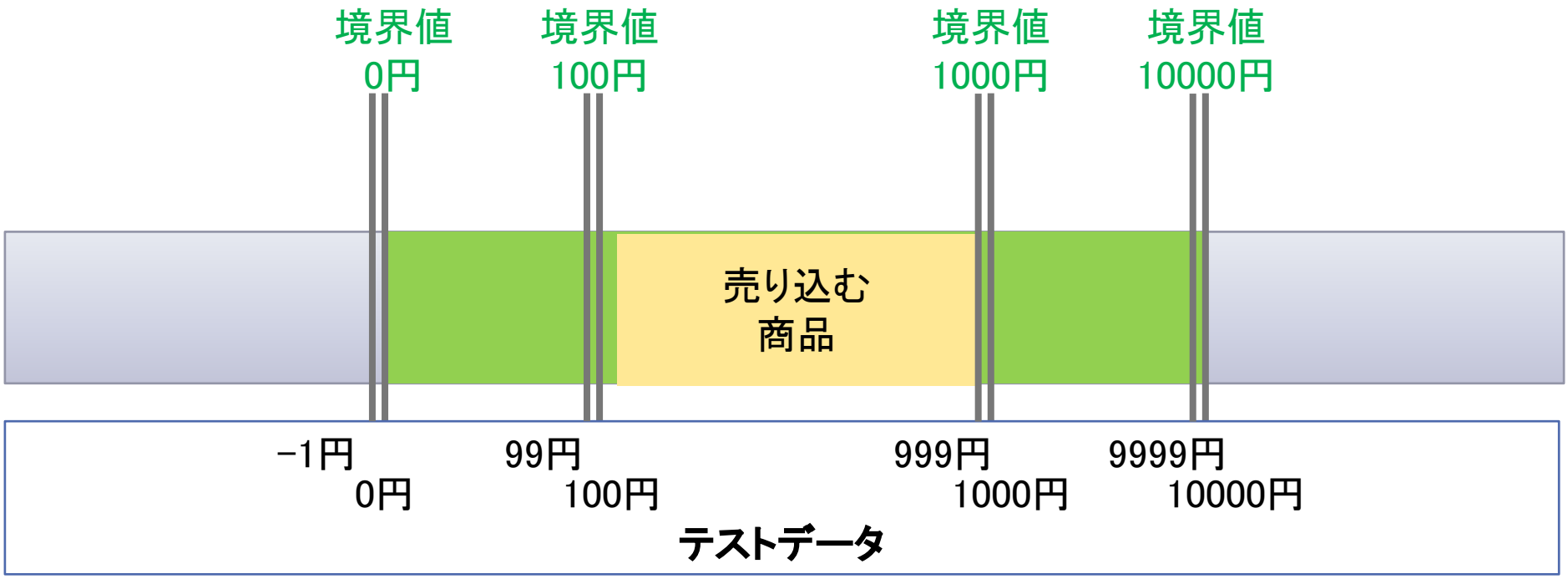
同値分割法とは、「有効同値」と「無効同値」を使って、正常処理とエラー処理をテストする方法です。例えば、0円～10000円までの商品群から100円～1000円の商品を売り込むために判定するためのシステムの場合、-100円(無効同値)、20円(有効同値)、500円(有効同値)、3000円(有効同値)、20000円(無効同値)といった値をテストデータにしてテストを行います。




ITCollege copyright ©ITCollege All Rights Reserved

# ブラックボックステスト

次に境界値分析を見ていきます。この手法は「有効同値」と「無効同値」を組み合わせ、処理と処理の境界となる値(境界値)をテストする方法です。例えば、0円～10000円までの商品群から100円～1000円の商品を売り込むために判定するためのシステムの場合、-1円、0円、99円、100円、999円、1000円、9999円、10000円といった値をテストデータにしてテストを行います。




ITCollege copyright ©ITCollege All Rights Reserved

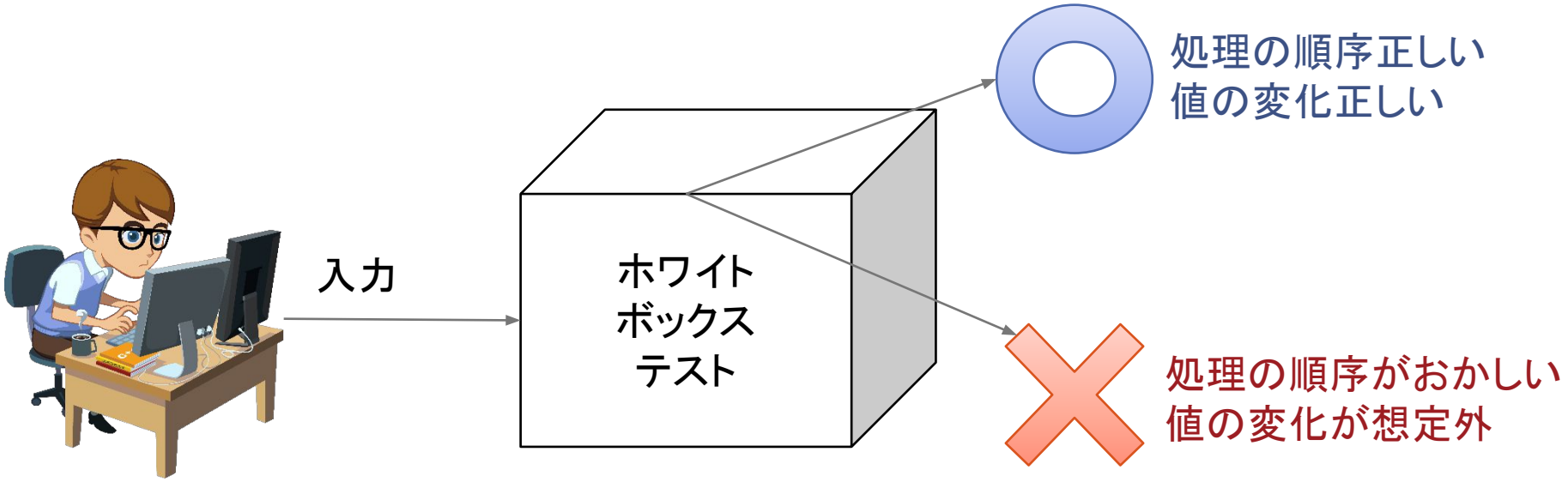
# ホワイトボックステスト

- **概要**

テスト対処となる部分を「中の見える箱(ホワイトボックス)」として捉えたテスト方法です。主にテスト対象に入力を与えて、どのような順序で処理が実行されたり、データの値がどのような変化をするのか確認します。

重要なことは、このテスト中はテスト対象のプログラムがどのような処理が行っているかを気にします。

テスト対象の「構造」に基づいたテストケースの作成技法になります。



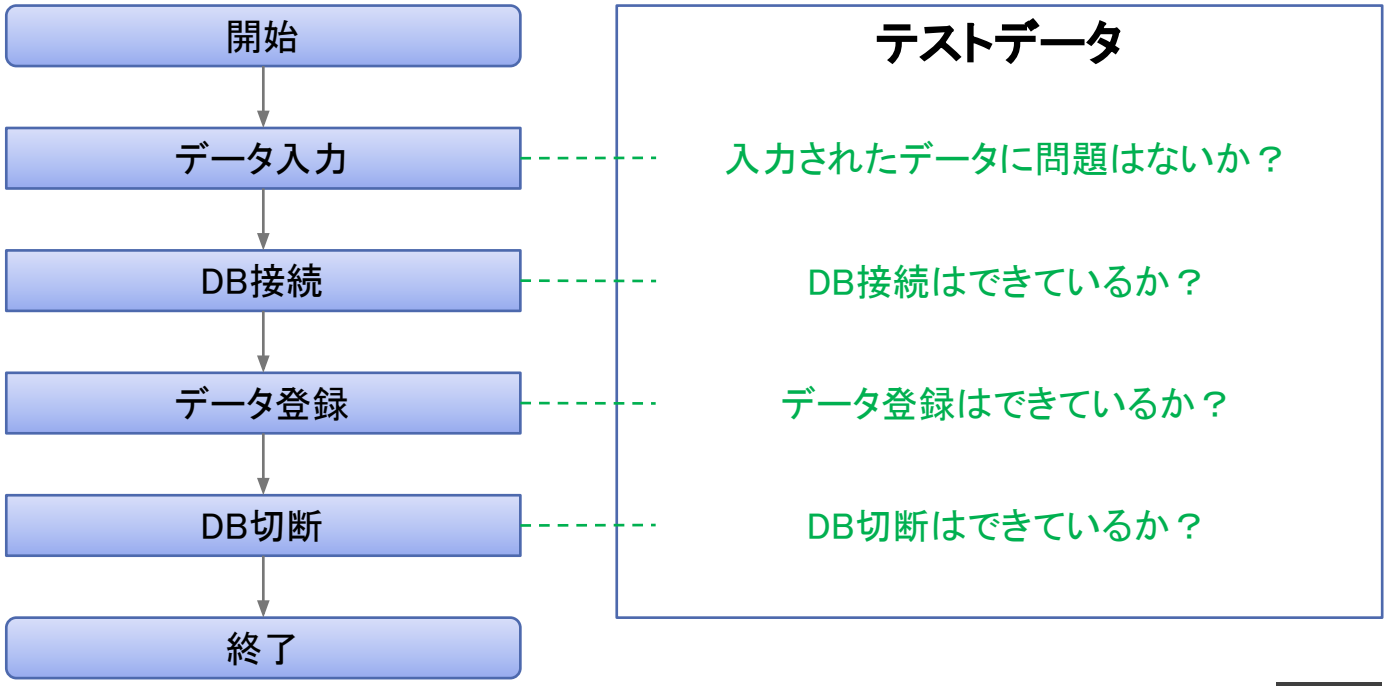

ITCollege copyright ©ITCollege All Rights Reserved

# ホワイトボックステスト

- **種類**

ホワイトボックステストを行う際に重要な方法として「制御フローテスト」と「データフローテスト」というものがあります。

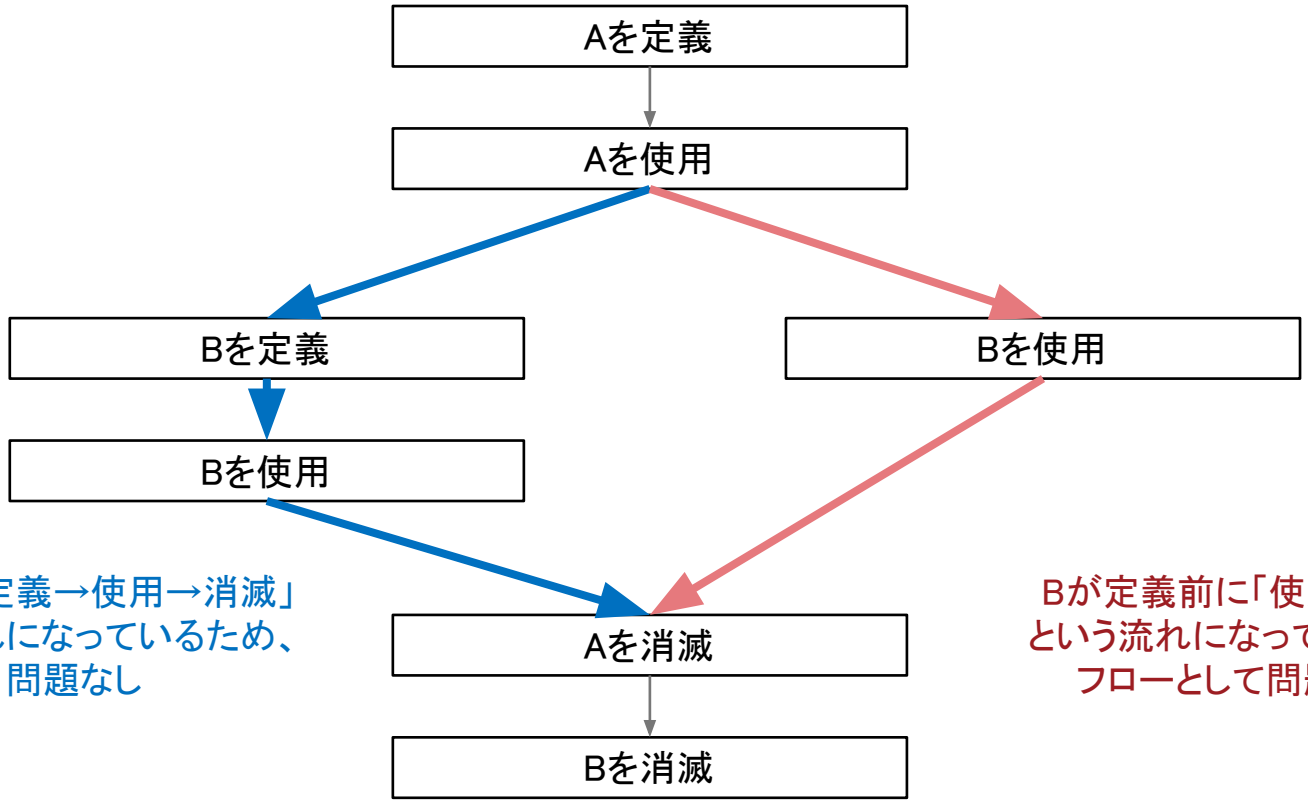
まず制御フローですが、ひとつの処理に対してプログラムがどう動くかを図で表現したものです。しかし、一般的には命令網羅や分岐網羅、条件網羅などで基準を設け、限定的に実施されます。




ITCollege copyright ©ITCollege All Rights Reserved

# ホワイトボックステスト

次にデータフローですが、モジュール内で使用されるデータや変数が持つ「定義→使用→消滅」というライフサイクルの流れをいいます。データや変数がどこで使用され、どこで消滅するのかを検証するのがデータフローテストです。

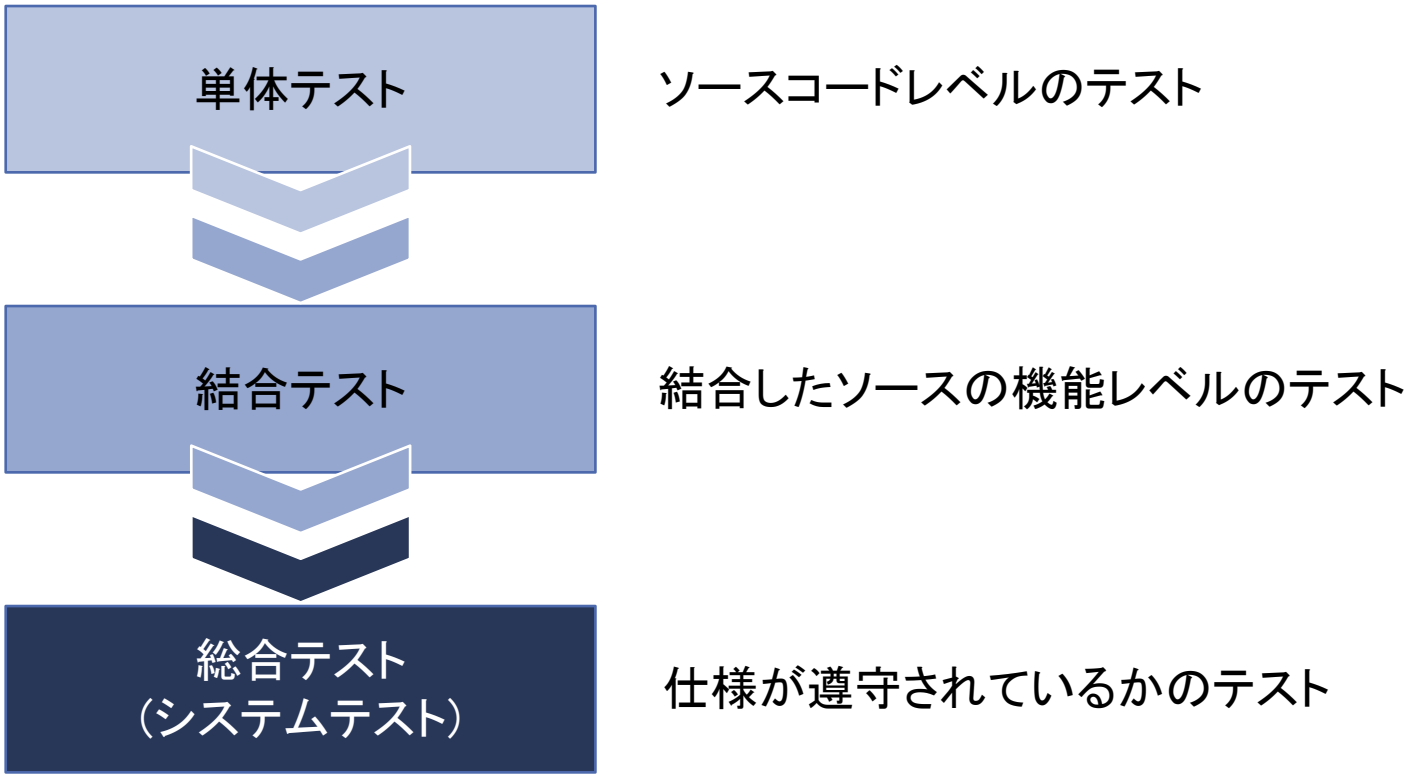



ITCollege copyright ©ITCollege All Rights Reserved

# テスト工程

- **一般的な工程**

いよいよテストケースを決定し、テストを行うわけですが、一般的にはシステムを作成するとテストを工程別に実施します。どのような工程があるか以下にまとめます。




ITCollege copyright ©ITCollege All Rights Reserved

# 単体テスト

- **単体テストのやり方**

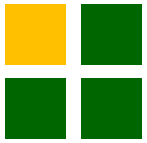
通常単体テストはテスト対象のソースコードを作成した後に実行することになります。


しかし、近年は単体テストの自動化ツールも使われており、ソースコードより先にテストコードを作成する手法もあります。これは「テストファースト」と呼ばれます。

これはテストコードを先に作ることで正しい仕様を明確化し、無駄のないソースコードを実装できるというメリットがあります。一方で、初心者は対象がなく、テストコードを書くことに難儀することになるでしょう。


効率よく品質の高いコードを実装するには、慣れるまでは最初の単体テストサイクルを回して完成させていくことが良いとはされています

ITCollege copyright ©ITCollege All Rights Reserved



# 単体テスト自動化

- JUnit

Java向けの単体テスト自動化ツールでEclipseのプラグインからも利用可能です。アジャイルソフトウェア開発のいくつかの手法で紹介されることが多く、以下のような理由から推奨されることがある。

・一度テストコードを作成するとテストは素早く行える

・テストコードを見ることで仕様が明確化される

・誰でも全くおなじテストを行えるようになる

JUnit4からはアノテーションが使え、さらに使いやすくなりました。

※Java以外の言語向けにはxUnitが存在します。


copyright ©ITCollege All Rights Reserved

# 単体テスト

- **単体テストの実情**

単体テストはソースコードレベルのテストであるがゆえに、テストケースを出し、実際にテストを行うと多くの工数がかかります。結果的に省略して次のテスト工程に進んでしまうという現場もよく見かけます。


しかし、正しく単体テストを実施することで、品質の確保と後工程での手戻りも減らすことができます。そのため、JUnitなどのツールを使いテストを自動化し、効率よくかつ丁寧にテストする方法を見つけることが重要です。

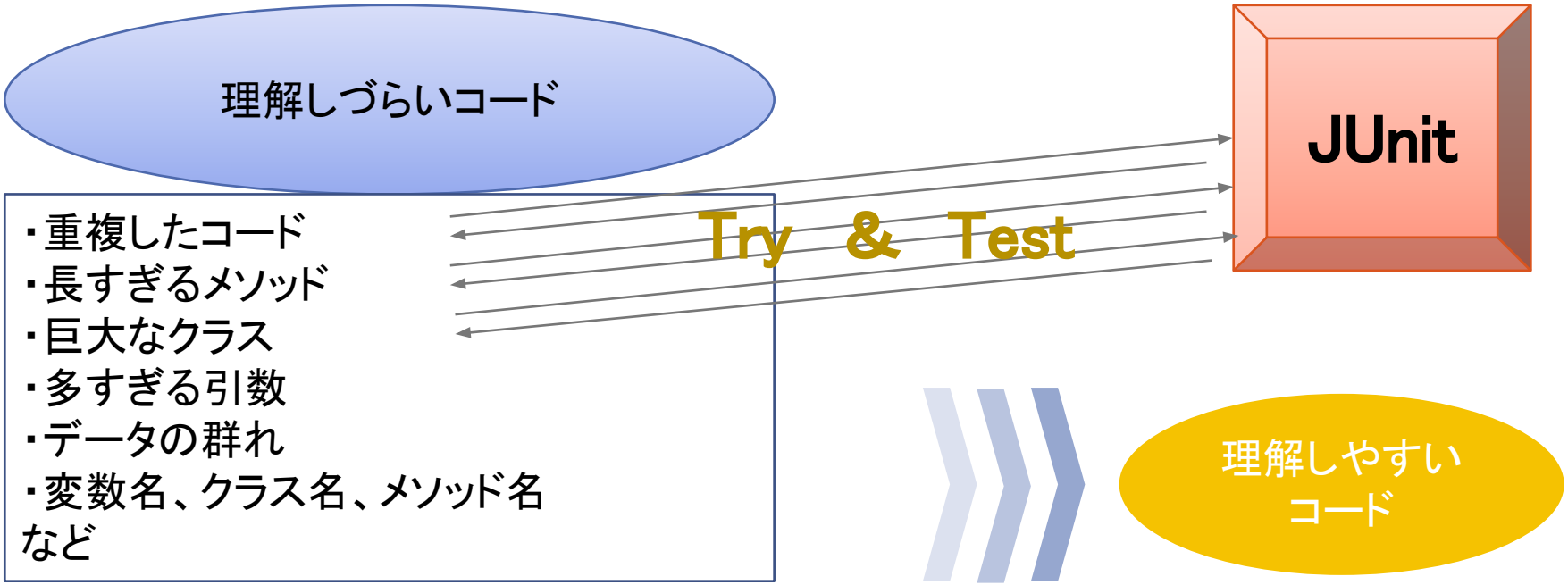
ITCollege copyright ©ITCollege All Rights Reserved

# リファクタリング

- **理解しづらいコード→理解しやすいコード**

リファクタリングはソフトウェアの外部的振る舞いを保ちつつ、内部構造を改善することです。その作業を自信を持って行うために良く使われるのがJUnitなどのテストツールです。

簡単に何度でも同じテストを行えるため、この環境がリファクタリングには適しています。




ITCollege copyright ©ITCollege All Rights Reserved

# 結合テスト

- **機能テスト(例:Webアプリケーション)**

Webアプリケーションでは、画面への入力やボタン押下などのアクションの実行により、画面が遷移して処理結果が表示されます。その裏ではデータベースとやり取りをして、情報を取得したりさまざまな処理が行われます。

機能テストでは、入力やアクション(マウス操作など)に対して表示される結果や実行される処理が正しいかどうかを確認するために行います。

テスト項目として、以下のような情報を確認します。

・画面遷移とシナリオ

・画面表示

・アクション

・データベースの更新

・アクセス権

・セッション管理

など

ITCollege copyright ©ITCollege All Rights Reserved

# 結合テスト

- **対応ブラウザ**

Webアプリケーションのテストで最もやっかいと言えるのが、ブラウザの種類やバージョンによって挙動が変わることです。本来であれば全ブラウザ対応させたいものですが、代表的なブラウザすべてに対応させようとすると、

GoogleChrome　500項目

InternetExplorer　500項目

FireFox　500項目

Safari　500項目

....

とブラウザが増えるたびに項目も倍加していきます。そのため、推奨ブラウザを指定して作成することが一般的です。また、モバイルにも対応となると各キャリアの各端末でも試験する必要があります。

このように利用環境の条件を変えたテストは構成テストと呼ばれます。

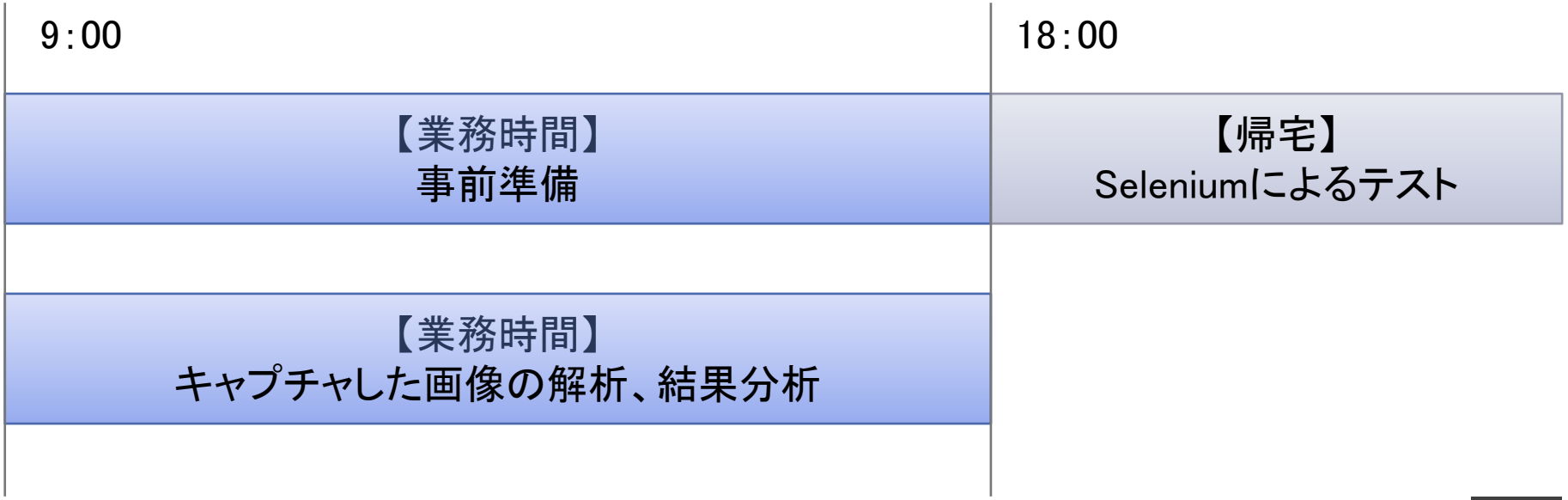

copyright ©ITCollege All Rights Reserved

# 結合テスト自動化

- **キャプチャ/リプレイツール**

  自動化といっても完全自動化ができるわけではなく、入力を与えその結果がどうなったかを出力するところまでを行うツールです。最終的には結果が正しいのか間違いなのかは人間が確認する必要があります。

  代表的なものとしてはSeleniumがあります。オープンソースで提供されているソフトウェアですが、テストのスクリプト記述が容易で新たにプログラミング言語を覚える必要もありません。テストを自動化することで効率の良い時間の使い方も可能です。




ITCollege copyright ©ITCollege All Rights Reserved

# 結合テスト実施時

- **シナリオを使ったテスト**

シナリオとはユーザが行う業務を想定して作成する手順になります。

このシナリオを用いてテストも行います。

シナリオテストを実施すると以下のようなことが分かります。

・業務フローに基づいたプログラム間でのデータ受け渡し結果

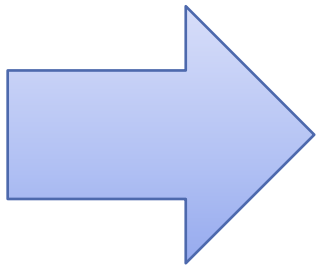
・業務フローに即した組合せでの集計結果

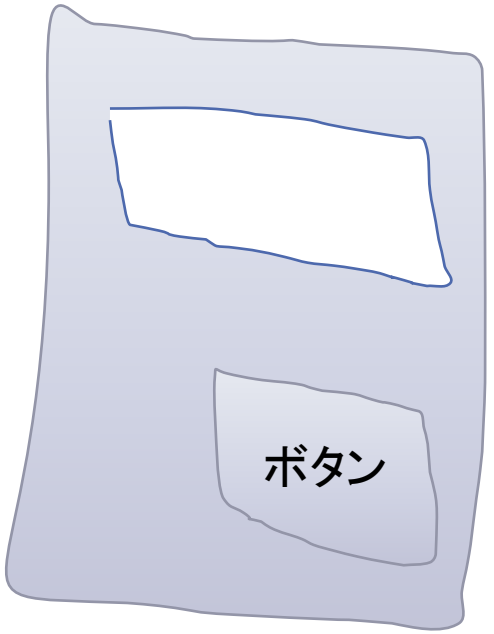
・業務フローを想定した操作に関する**ユーザビリティ**

**時系列シナリオ**
業務フローで想定される処理の流れを意識

**組合せ＆設定変更シナリオ**
様々な設定の組合せを網羅することを意識

など






ITCollege copyright ©ITCollege All Rights Reserved

# 結合テスト実施時

- **シナリオ作成時の注意点**

シナリオ作成時に一番意識したいことは「**業務の種類と流れ**」です。

ECサイトを例に考えてみましょう。

ありがちなことが、「全機能を使う」テストケースを考えようという誤った考えです。「商品登録機能」「商品更新機能」「商品検索機能」「受注機能」をテストする場合、手間を省くために以下のようなテストケースにします。

**1.新規商品を登録**

**2.登録した商品の商品名と価格を変更**

**3.商品検索を行う**

**4.商品を受注**

一見問題ないようですが、全部の機能を1回使っているから全機能OKとしてしまうと非常に危険です。

ITCollege copyright ©ITCollege All Rights Reserved

# 結合テスト実施時

なぜなら、以下観点のテストが済んでいません。

・新規で登録した商品が商品一覧から検索されること

・新規で登録した商品の内容が正しく表示されること

・新規で登録した商品を購入できること

こういった場合に「新規登録時だけできない」などの特定条件下の場合だけ挙動がおかしくなることが発生しがちです。

そのため単に機能を使えばいいというわけではなく、**実際に発生する業務を考慮したテストケース**を考えることが大切です。

そして業務の種類と流れを意識して作成すると、正常系ばかりに意識が向いてしまいがちです。しかし正常系のテストが通るのは当り前でイレギュラーのケースをどれだけ想定できるかが重要です。


copyright ©ITCollege All Rights Reserved

# 結合テスト実施時

イレギュラー系のテストも観点を2つで考えてみましょう。

## ・業務的なイレギュラーテスト

例えばクレジットカードの預金残高オーバーや有効期限切れなど、システムが提供している機能で制限をかけて、そのエラーをシステムの機能で回避できるものをいいます。購入し続けてしまった場合、当然残高オーバーになります。その状況で購入したらエラーにしなくてはいけません。その状況を回避するためにそれまでに行った注文をキャンセルすることで預金残高も解放され、また購入できるといったことが考えられます。

## ・システム的なイレギュラーテスト

「あるはずのファイルがそこにない」などの回避できないエラーが発生した際に、異常終了ではなく適切な例外処理を実施して終了しているかというテストです。しかし、イレギュラーを起こすことは難しく以下のようなことが実装されているかの確認になることが多いです。

・画面やログファイル、テーブルなどのエラーメッセージに発生タイミングや原因を出力しているか

・エラーが発生したタイミングでそれを関係者各位にメールなどで送信しているか

ITCollege copyright ©ITCollege All Rights Reserved

# 結合テスト実施時

前述のとおり、シナリオは業務と密接に絡んでいます。そのため、シナリオ＝業務の単位＝結合テスト仕様書の単位となります。まずは項目を作ろうとせず考えうるシナリオを抽出していきましょう。その次にテストデータを考えましょう。(シナリオもテストデータもExcelで表として作成が一般的)

1.商品登録機能

1-1.新規登録

1-1-1.入力情報Aを入力して登録

1-1-2.入力情報Bを入力して登録

2.商品検索機能

2-1.新規登録した商品

2-1-1.全商品検索

・・・

2-1-2.完全一致検索

2-1-3.部分一致検索

2-2.登録済み商品

・・・・

入力情報などは**テストデータ**として予め作っておきましょう。

ITCollege copyright ©ITCollege All Rights Reserved

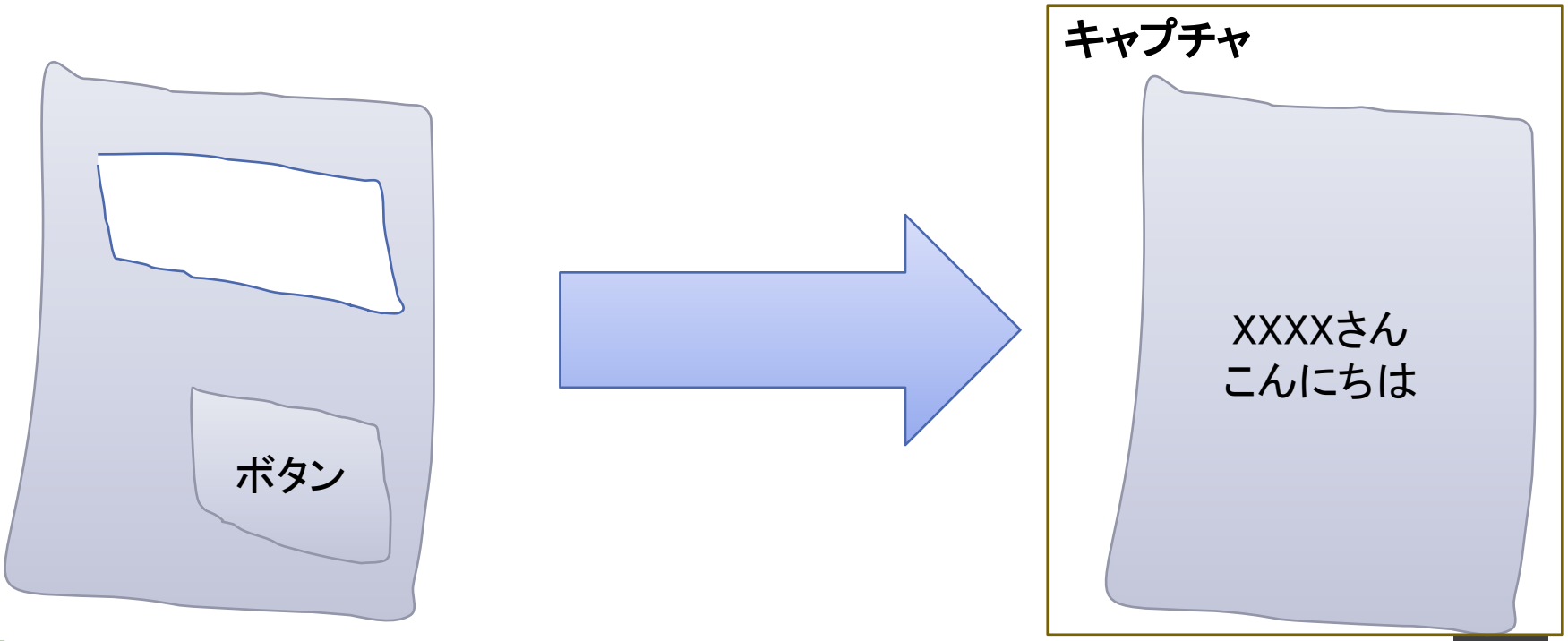
# 結合テスト結果

- **エビデンス**

処理結果の証拠として画像キャプチャして残すことをエビデンスといいます。

このエビデンスを抽出するのに適しているのが、先ほど挙がっていたツール類になります。

さまざまな業界でも用いられる用語です。混乱しないようにしましょう。




ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

- **性能テスト**

「ユーザの期待に応える性能にする」と言ったとしても性能の定義とはなんなのでしょう？例えばボタンを押して次の画面に進むまでに3分かかったとしましょう。皆さんイライラしませんか？　つまり性能とは処理結果を返す力と考えます。その場合、どんな情報を把握すればよいのでしょう？

・スループット　：単位時間あたりの処理結果を返した件数

・レスポンスタイム　：処理要求を出してから結果が返ってくるまでの時間

・リソース使用量　：システムの資源の使用量(CPU使用率が100%だと…)

作成するシステムの性能目標は、例として

「スループット　50件/秒」「レスポンスタイム3秒以内」と立てることができます。

ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

- **性能テストの目的**

前ページの情報を基に以下を確認/把握することが目的となります。

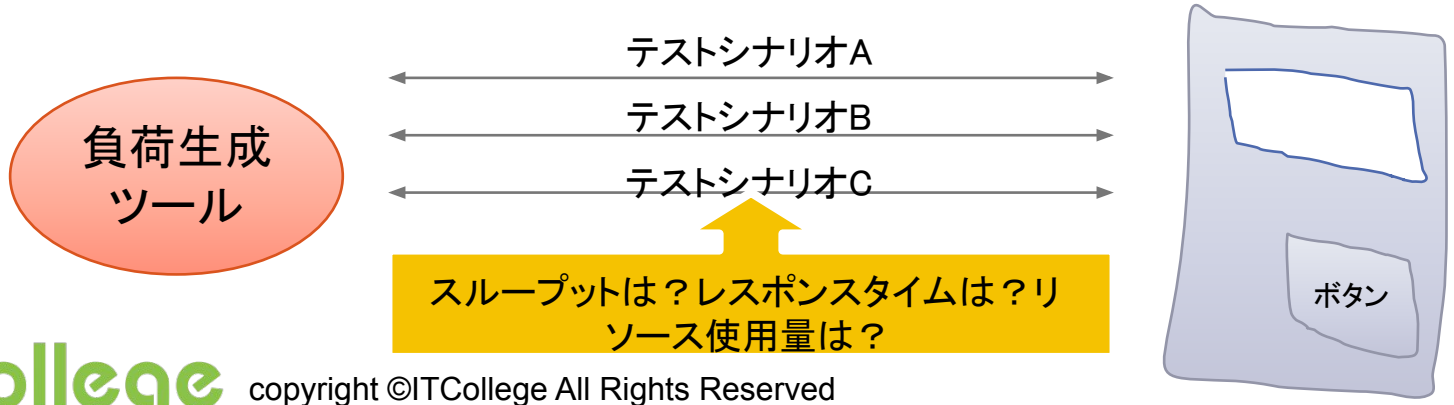
・システムの性能が目標に達しているかを確認

・システムが処理できる最大のアクセス数を把握

・性能限界に対する対応策を把握

そして、これらの情報を得る際に負荷生成ツールを用います。

負荷生成ツールとしては、JMeterなどが挙げられます。これはシステムに負荷をかけながらスループットやレスポンスタイムを計測する機能が備わっています。




ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

- **ユーザビリティ**

ユーザビリティとはそのソフトウェアの使いやすさを示す単語になります。ユーザビリティが悪いとか、使いにくいという人はいますが、エンドユーザが具体的にどう悪いのか説明してくれることは基本的にありません。

原因としては、ボタンの形と配置が悪い、ボタンの色が悪いなどが考えられます。

どちらにせよこれらを客観的に判断する評価軸が必要になります。

今回はその評価軸として割と一般的なニールセンの10原則を紹介します。


ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

- **ニールセンの10原則**

**1.システム状態が視覚的に分かること**

システムの状態をユーザに見せることです。

例)「ダウンロード中です」「…検索中」などシステムが動いていることを認知

**2.システムと実世界のマッチ**

専門用語は使用せず、実社会のなじみの言葉で表現することです。

例)「入力」ボタン→「次へ」ボタン

※テーブルに格納するという意味では入力ですが、あくまで技術者観点の言葉

**3.ユーザ制御と自由度**

画面などでユーザが自由に移動することです。

例)ユーザが操作を間違えた場合「1つ前に戻る」など操作ミスに気が付いたら前の画面に戻れるようにすること

ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

## 4.一貫性と標準化

同じ機能はいつでも同じ名称、デザインにし、画面によって戻るボタンのデザインが異なるなどはNGです。また全体を通して同じデザインとすることです。

(例)同じサイトなのに経理部と総務部の操作画面が異なり、業務ごとに操作方法を覚える必要があるようなものは使いにくい

## 5.エラー防止

事前にエラーを防止する仕組み、表示を行うことです。

(例)「パスワードは8文字以上」という記載がなく、7文字のパスワードを入力して警告が表示されてはいけません

ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

## 6.記憶よりも見た目の分かりやすさ

ユーザの記憶に依存せず、かなり前の画面に戻るとき、今までどの画面を経由してこの画面まで来たか、ユーザの記憶に依存してはいけません。

(例)入力結果が表示される画面において入力内容を前の画面に戻らないと確認できないようなデザインはいけません

## 7.柔軟性と効率性

ユーザのレベルによってインタフェースを変更して操作性を良くすることです。

(例)初心者向けに「初めてガイド」などを作成、熟練者向けに「最近使った機能一覧」などを作成

## 8.美しく最小限のデザイン

デザインはシンプルで不要なデザインや機能、冗長な機能は削除します。

重要なのは画面上の情報内で関連のあるものをグルーピングして配置することです。

(例)同じ画面内に検索と更新があったら分かりづらい

ITCollege copyright ©ITCollege All Rights Reserved

# 総合テスト

## 9.エラー時にユーザが認識、診断、回復が可能

エラーが発生した場合は原因や対処を表示します。

(例)エラーが発生した場合、エラーが発生しましたというダイアログのみ表示して終了すると使いづらい

どんな操作をすれば改善できるのか、問合せ先はどこかなどの情報も記載しておくと連絡が取りやすい

## 10.ヘルプとドキュメント

ヘルプとドキュメントを準備します。

(例)メニューからヘルプ画面を参照できるようにする

※しかし、時間を費やしても使用されないことが多いため、個人的な意見としてはヘルプやマニュアルを作成するよりは画面の作りこみなどに時間を費やした方がよいと思っています。

ITCollege copyright ©ITCollege All Rights Reserved

# ドキュメント

- **テスト項目仕様書**

テストを実施した環境や実施するテストの内容、操作手順、テストの実行結果、テスト実施年月日、テストを実行した担当者などを記載します。各所でフォーマットは異なりますが、概ね下記のような内容が書かれます。

| 項目番号 | 確認内容 | 操作手順 | 確認結果 | 確認日 | 確認者 |
|---|---|---|---|---|---|
| 1 | ～～ | ～～ | OK | yyyy/mm/dd | 山田 |
| | | | | | |
| | | | | | |
| | | | | | |

- **テスト計画書**

テスト工程と目的を記載し、参考資料やテストの範囲、テストの粒度、テスト環境、テストの判定項目(OK、NG、保留)などを記載します。項目仕様書と同様に各所でフォーマットは用意されていることが一般的です。

ITCollege copyright ©ITCollege All Rights Reserved

# まとめ

- **テスト手法**

テストケースの上げ方やテストの自動化など、かなり列挙させて頂きました。その際に使用する便利なツール類は万能ではなく、その時行うテストの内容や工程に合わせてベストなものを選定し、使うことが重要になります。

テストの工程と目的をしっかりと共有してから実施しましょう。

そして標準化されている手法を使い、KKD法から脱却し、よりよい性能のソフトウェアを開発していきましょう。


copyright ©ITCollege All Rights Reserved