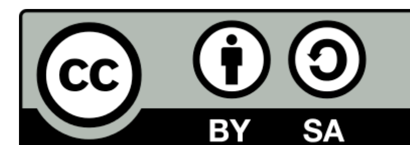
自動テスト vs 手動テスト

本資料は、Bhavin Turakhia氏による
『Automated Testing vs Manual Testing』(http://bit.ly/2uF3JeC)
を日本語化して加筆・修正したものです。
本資料のライセンスは元のスライドと同じくCC-BY-SA 3.0とします。
当該ライセンスを持つスライドについてはページ右下にロゴが掲載されています



手動テストでのコーディングプロセス

- コードを書く
- どこかにコードをアップロードする
- ビルドする
- 手でコードを動かす (多くの場合、順番にフォームに入力したりする)
- ログファイルやデータベースや外部サービスや変数の値や画面出力内容などをチェックする
- もしうまく動作しなかったら、上記を最初から繰り返す

CC BY SA

自動ユニットテストを用いたコーディングプロセス

- １つまたは数個のテストケースを書く
- 自動でビルドしテストが失敗することを確認する
- テストが通るようなコードを書く
- 自動でビルドし再度テストを実行する
- もしテストに失敗したら適切な変更を行う
- もしテストが成功したら次のメソッドも同じようにコーディングする

cc BY SA

自動機能テストを用いたコーディングプロセス

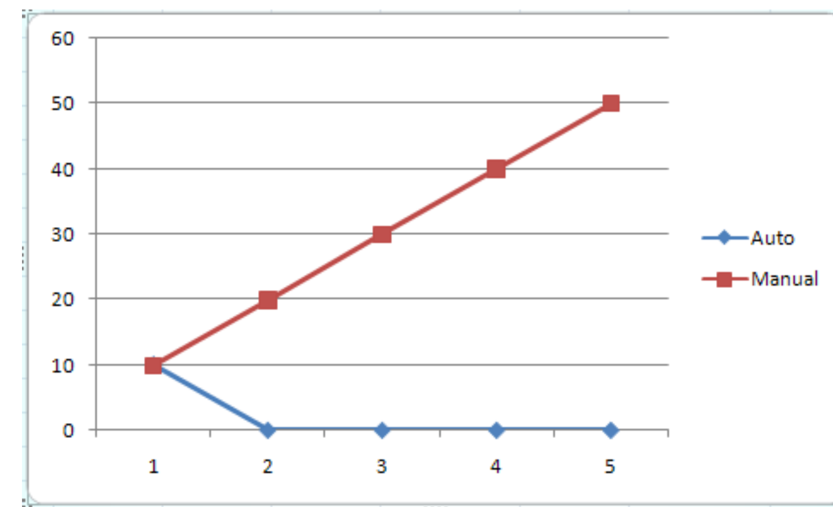
- ユニットテストに全てパスするようなコードを書く
- ツールなどを使って機能テストを書く
- 自動でビルドしてテストを実行する
- もしテストに失敗したら適切な変更を行う
- もしテストに成功したら次に進む

cc BY SA

自動テスト vs 手動テスト (1) 作業量とコスト

- 6つのテストケースがある場合を考える
  - 全てのテストを手動で実行する場合 => 10分
  - 全てのユニットテストを書く場合 => 10分
  - 全てのユニットテストを実行する => < 1分未満
  - テストの繰り返し実行回数 => 5回
  - 手動テストでの総テスト時間 => 50分
  - ユニットテストの総テスト時間 => 10分

Release	手動テスト	自動テスト	手動テスト累計
1	10	10	10
2	10	0	20
3	10	0	30
4	10	0	40
5	10	0	50



cc BY SA

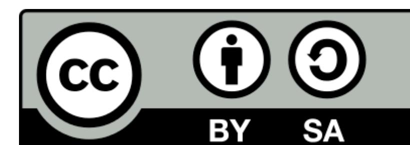
自動テスト vs 手動テスト (1) 作業量とコスト

- インクリメンタルにユニットテストのケースを追加していくのは、インクリメンタルに手動のテストケースを追加していくより費用は安い
- 例）ドメインの登録
  - Case 1: .comドメインを全てのフィールドに正しい値を入れて登録する
  - Case 2: .comドメインの登録の際に無効なDNSサーバ名を入れる

cc BY SA

自動テスト vs 手動テスト (2)クリエイティビティ

- 手動テストは退屈
    - 同じフォームに値を入れ続けたいと思う人なんて誰もいない
    - テストを手動で実行することから学べることはない
    - 手動テストはおろそかにされやすい
    - 手動テストの一覧は誰もメンテナンスしない
- 一方で自動テストはコードであるため…
    - テストを書くことは楽しくてチャレンジング
    - 再利用性とカバレージを注意深く考えなければいけない
    - 分析のスキルも必要
    - 将来に渡って有用である



自動テスト vs 手動テスト (3)再利用性

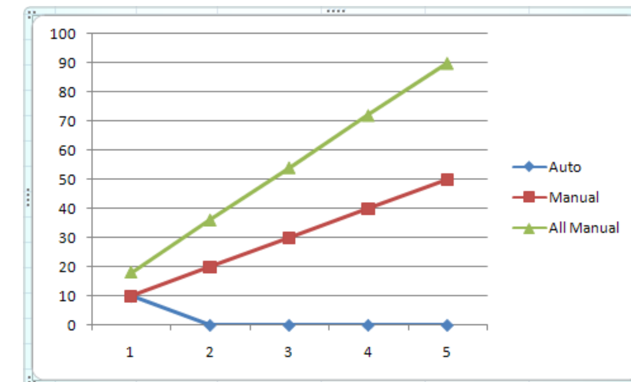
- 手動テストは再利用できない
  - 毎回同じ労力が要求される
- 自動テストは完全に再利用可能
  - 重要：継続的インテグレーションサーバと共通のコードレポジトリと組織構造の見直しが必要
  - 自動テストは一度書かれたらコードベースの一部になる
  - 余計な労力をかけることなく再利用できる

cc BY SA

自動テスト vs 手動テスト (4)可視性

- 手動テストは一部の可視性しか提供せず、全てのステークホルダーが繰り返し実行しなければいけない
  - 開発者によるテストでしか結果を見ることができない
  - 全てのステークホルダーが同じテストを繰り返し実行しなければいけないことがある
    - (例) 開発者、テクニカルリード、GM、管理職
- 自動テストは全体に対して可視性を提供する
  - 開発者、テクニカルリード、管理職はシステムにログインしてテスト結果をみることができる
  - 彼らは特段の労力を必要としない

Release	Manual Testing by Dev	Manual Testing by Team Leads	Manual Testing by Mgmt	Total Manual Testing	Auto Test	Dev Manual Test Cumulative	Total Manual Test Cumulative
1	10	5	3	18	10	10	18
2	10	5	3	18	0	20	36
3	10	5	3	18	0	30	54
4	10	5	3	18	0	40	72
5	10	5	3	18	0	50	90



CC BY SA

自動テスト vs 手動テスト (5)スコープ

- 手動テストは結局は結合テストになってしまう
  - 一般的な手動テストでは、ユニット単体でテストをすることが難しい
  - 多くの環境において、バックエンドのサービスと結合した状態でチェックすることになってしまう
  - 壊れやすいテスト：もし関係する何かが壊れていると手動のテストも失敗してしまう
- 自動テストはスコープを変えることができる
  - ユニット (クラスやメソッド)もテストできるし、モジュールやシステム自体等もテストできる

cc BY SA

自動テスト vs 手動テスト (6)準備作業

- 手動テストは複雑なテスト準備と終了作業が必要になる
  - DBにクエリーを頻繁に流すような作業が含まれることも多い
  - バックエンドのサーバに変更を加えるような作業も含まれるかもしれない
  - 複数の依存性のあるテストケースの場合、テスト準備はどんどん複雑になる
- 自動テストでは複雑なテスト準備や終了手順は必要とされない
  - ユニットテストでは外部の依存性はモックを使って解決する(ことが多い)のでsetupやteardownは必要とされない
  - 機能テストにおいてはテスト準備と終了手順はフレームワークを使うことで自動化される

cc BY SA

自動テスト vs 手動テスト (7)リスク

- 手動テストは何かを見逃してしまうという高いリスクがある
  - 毎回開発者が手動で実行していると重要なテストケースを見逃しがち
  - 新しく加わった開発者にはテスト実行のための手がかりがない
- 自動テストではあらかじめ決めたテストを飛ばしてしまうリスクは一切ない
  - テストが継続的インテグレーションの対象になれば誰も覚えていなくても毎回実行される

cc BY SA

自動テスト vs 手動テスト (8)設計への影響

- 手動テストは設計の助けにならない
    - 手動テストはモジュールが作られた後に実行され、バグへのパッチあてにしかならない
- 自動テストやTDD(テスト駆動開発)は設計の助けになる
    - ユニットテストを最初に書くことで要求事項や設計への影響が明らかに
    - モックオブジェクト等を使ってユニットテストを書くことで強制的にきれいな設計になり、抽象・インターフェイス・ポリモーフィズム等の分離が進む

cc BY SA

自動テスト vs 手動テスト (9)安全性

- 手動テストはセーフティネットを提供しない
  - 手動テストはモジュールが作られた後に実行され、バグへのパッチあてにしかならない
- 自動テストはリファクタリングや機能追加の際のセーフティネットになる
  - このプロジェクトのコードを触ったことがない新しい開発者でも変更に対して自信を持てる

cc BY SA

自動テスト vs 手動テスト (10)教育的観点

- 手動テストはトレーニングには使えない
- 自動テストはドキュメントとしても機能する
  - ユニットテストを読むことでコードベースの目的がはっきりする
  - 自動テストによって要求事項がはっきりする
  - 自動テストによって他のユースケースや期待する結果がわかる
  - 新しい開発者はコードを読むよりもユニットテストを読むことでよりコードが理解できるようになる
  - ユニットテストはコードの期待される振る舞いを定めている

cc BY SA

自動テスト vs 手動テスト (11)コード品質

- 手動テストはごちゃごちゃした汚いコードを生み出す
  - 多くの手動テストにおいて以下のような事象が発生する
  - 変数の確認のためにSystem.outが多用される
  - 役に立たないログがアプリケーションサーバやDBサーバなどに出力される
  - コードやログ出力がガラクタになる
    - IF文で分岐されたフラグベースでのログ出力やイベントベースでのログ出力等
  - アプリケーションが遅くなる

cc BY SA

自動テスト vs 手動テスト (11)コード品質

- 自動テストはごちゃごちゃした汚いコードを減らす
  - ログファイル出力やSystem.outでの出力はテストコードでのアサーションに置き換えられる
  - もし特定のコンソール出力やログ出力が必要な場合でも、それはテストに記述しプロダクションコードには手を加えないで済む
  - アプリケーションやログやコンソールにゴミが出力されず、アプリケーションは高速に保たれる

cc BY SA

まとめ

1. 手動テストは自動テストより多くの労力とコストがかかる
2. 手動テストは退屈
3. 自動テストは再利用可能
4. 手動テストは限られた可視性しか提供せず、全てのステークホルダーによって繰り返し実行されるムダが発生
5. 自動テストではスコープを変え、依存部分にモックを使うことでユニット単体でのテストも可能
6. 自動テストでは複雑な準備手順や終了手順は不要

cc BY SA

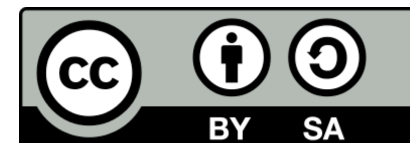
まとめ

7. 自動テストではテストを飛ばすような問題がないことを保証
8. 自動テストは実際に綺麗な設計になるように強制
9. 自動テストはリファクタリングの際のセーフティネットを提供
10. 自動テストはトレーニングに役立つ
11. 自動テストはコードやコンソールやログの中にゴミができない

CC BY SA

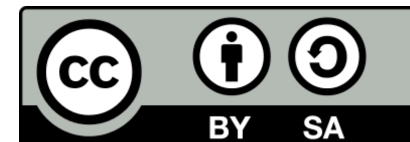
何故自動テストを書かないのか？

- 初期の学習曲線（が高いと考えてしまう）
    - ユニットテストのフレームワークや機能テストのフレームワークの理解
    - 継続的インテグレーションやその効果的な使い方の理解
    - コードカバレッジツールの理解
    - どうやってテストを構成するのかに関する理解
    - モックオブジェクトの作り方
    - 毎回のテストの自動実行のやり方
    - テストをどこにコミットするか
- 本当にもう一回このモジュールの作業をするのか？(という懸念)
- 自分のテストは再利用されるのか？もし再利用されなかったらなんの意味があるのか？(という恐れ)



テストがないという問題への対応

- 最初のリリースまでの時間を使って以下を行う
  - ユニットテストと機能テストを含むコードレポジトリの構造を作る
  - リリースに統合されたCIサーバを構築する
  - xUnitのようなユニットテストフレームワークを導入する
  - Sahi / Watir / Selenium / QTPのような機能テストのフレームワークを導入する
  - Cloverのようなコードカバレッジツールを導入する
  - テストのガイドラインや原則を作る
- 責任を与える
  - 各々の開発者は各ユニットについて複数のユニットテストを記述することをルールにする
  - 特定の開発者は機能テストを書く責任を与える
  - テストを書く開発者は、テストの構成を考えコミットしCIに関連づける責任ももたせる



テストがないという問題への対応

- 諦めてはいけない
  - もしハードルにぶち当たったら, ペアで立ち向かうこと
  - あなたはテストについての責任を全うしなければならないことを肝に銘じること
- コードカバレッジをチェックする
  - コーディング中やコーディング後にコードがテストによってカバーされていることを確認するために、継続的にコードカバレッジツールを使うこと

cc BY SA

何をテストするのか？

- ユニットテスト
  - 理想的にはクラスをまたぐことはない
  - 当然プロセス境界をまたぐことはない
  - 複数のケースをもつユニットテストを記述する
- 機能テスト
  - Watir / Selenium / QTP / WhiteなどをつかったUIのテスト
  - もしくはUIの１階層下のレイヤーをテストする (例えば APIを利用)

cc BY SA

ベストプラクティス

- ユニットテスティングフレームワークを使う
- 自動ビルドプロセス、CIサーバ、コミット時の自動テスト等を行う
- ユニットテストは日中はローカルのマシンで動作させ、その後CIサーバにコミットする(そして夜間に自動で実行する)
- CIサーバが必要な期間中ずっとテストを実行する
- テストはコードと一緒にコミットする
- テストを正しく構成する
- もしテストをコミットしなかったら、テストは再利用できず、労力の削減という利点は失ってしまうことを理解する

cc BY SA