バグを限りなくゼロにする方法

株式会社セガ
AM R&D2 プログラマ
伊藤周
E-mail:Ito_Makoto@sega.co.jp

自己紹介

- 伊藤　周（いとう　まこと）
- 1973年神奈川県生まれ
- 1998年株式会社セガ入社
- 主にアーケードゲームの開発に携わる
  - 1998年　スターウォーズトリロジー
  - 2000年　EA NASCAR ARCADE
  - 2002年　頭文字D
  - 2006年　アーケードカードゲーム
  - 現在　キッズ向けカードゲーム

注意事項

- 対象受講者
  - 初級以上のプログラマ
  - 大半はプログラマ以外でも理解可能だが
    難マークはプログラマ向け
- 環境
  - C++ （C言語でも可）
  - OSは任意
  - ターゲットは任意
  - LAN環境
  - ビルド用の空いているPCがあればなお良い

（ただし、上記環境に限定されるわけではありません）

題に関して…

「バグをゼロにする方法」はあるのか？

ゼロにする方法は…？

ありません！

★特効薬を期待してしまった方がいたら申し訳ありません

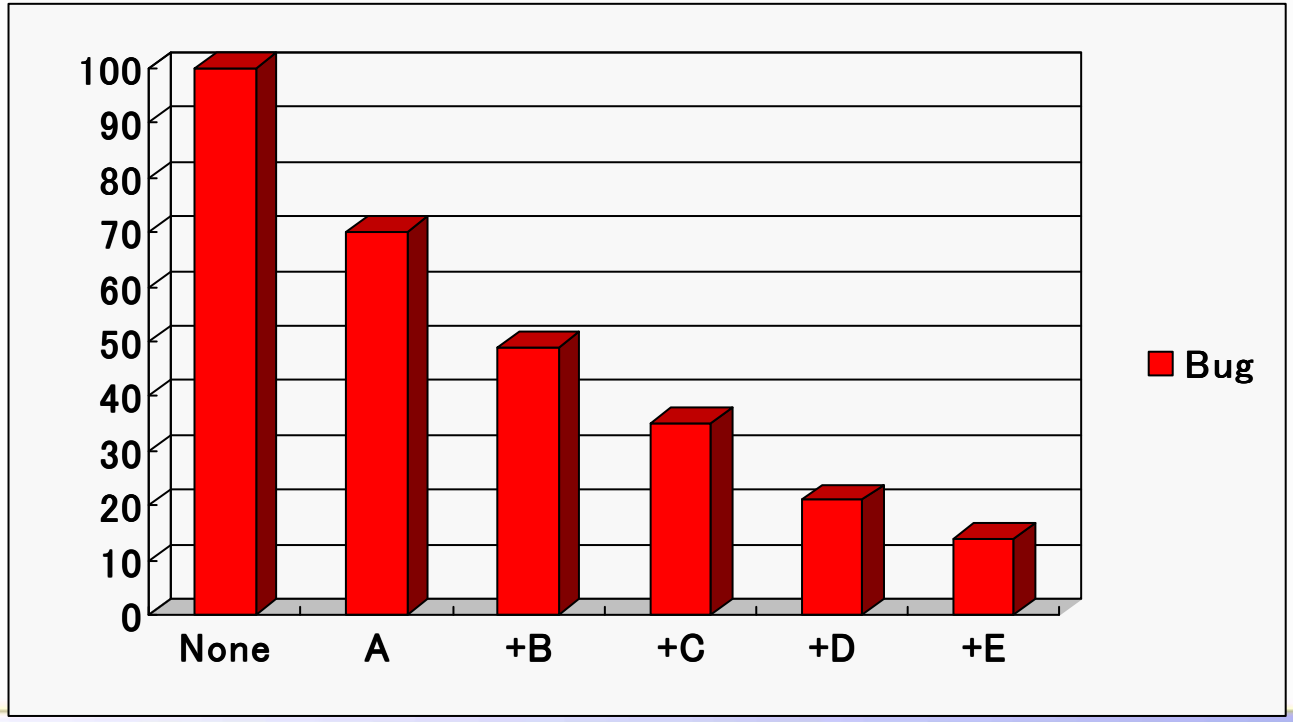
ただ…

バグを少なくする方法がいくつかある

ということは…

それらの方法を組み合わせれば

バグは限りなくゼロにできる！



内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

突然ですが

プログラマ診断テスト

プログラマ診断テスト

「あなたはプログラマに向いている？いない？」診断

バグを企画から指摘されて…

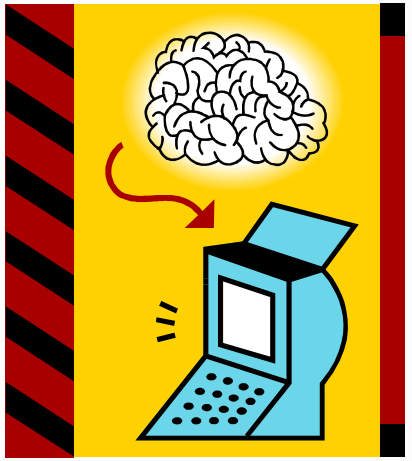
- あなたはどう返しますか？
  1. 「仕様が複雑」と企画のせいにする
  2. 「使っているAPIが…」と専門用語を駆使して煙に巻く
  3. 「あ、了解です」とシレっとスルーする
  4. 「スケジュールがギリギリでチェックの時間が…」と納期のせいにする
  5. 「ああ、すまんすまん」と一応謝るが、毛ほどにも悪いと思わない
  6. 「すみません！」と心の底から反省し、二度としないと誓いを立てる

バグを企画から指摘されて…

- あなたはどう返しますか？
  1. 企画のせい → ○
  2. 専門用語で煙に巻く → ○
  3. 「あ、了解です」 → ◎
  4. 納期のせいにする → ○
  5. 一応謝る → ◎
  6. 心の底から反省 → ×

当セッションを通して言いたいこと

- 人間は間違える
  ただし、創造的なことは人間にしかできない
- コンピューターは間違えない
  ただし、プログラムがなくては何もできない

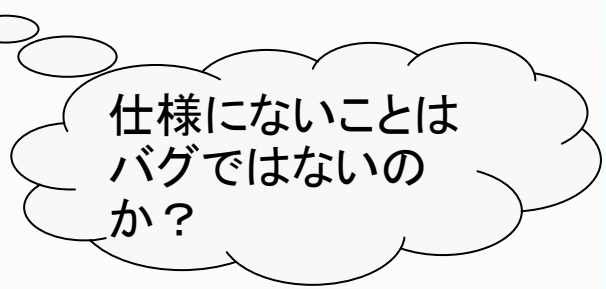


コンピューターに任せられるところは

とことん任せましょう！

バグの定義（このセッション内における）

- 「仕様どおりにソフトウェアが動かないこと」



- 仕様を解釈したプログラマの意図した通りにソフトウェアが動かないこと

テストの定義（このセッション内における）

- ソフトウェアのバグを見つけ出す行為全て
  - 人力テスト…人の手で
  - ソフトウェアテスト…ユニットテスト
  - 自動プレイテスト…コンピューターの手で

- プレイテスト（面白さを追求するテスト）に関しては、当セッションでは触れません

内容

- バグとは
- バグを減らす方法
  - テスト
    - 人力テスト
    - ソフトウェアテスト
    - 自動プレイテスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

テストその1：人力テスト

- 人の手で仕様と見比べて間違っていないかチェックする
- 誰が？
  - 手の空いているチームメンバー
  - QAチーム
  - 外注会社

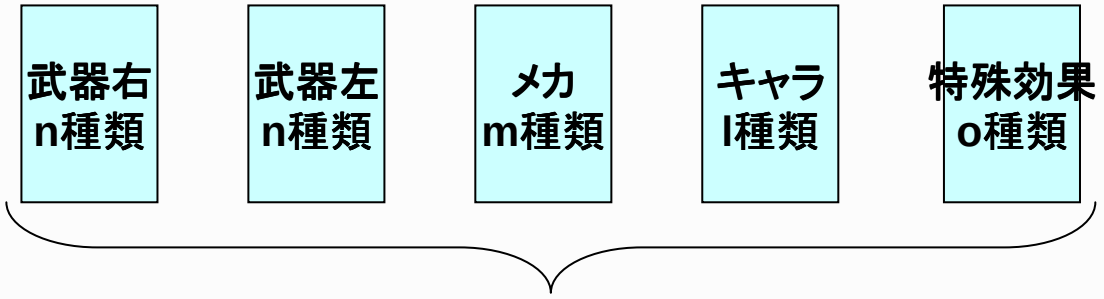
人力テストの利点

- 導入が簡単
- 見積もりやすい
  - 人月コスト
- 人間ならではの複雑なテストができる
- QAチームにノウハウが溜まりやすい
  ＝バグを見つけやすい

人力テストの欠点

- 仕様が複雑になると、テストする項目が指数関数的に増える

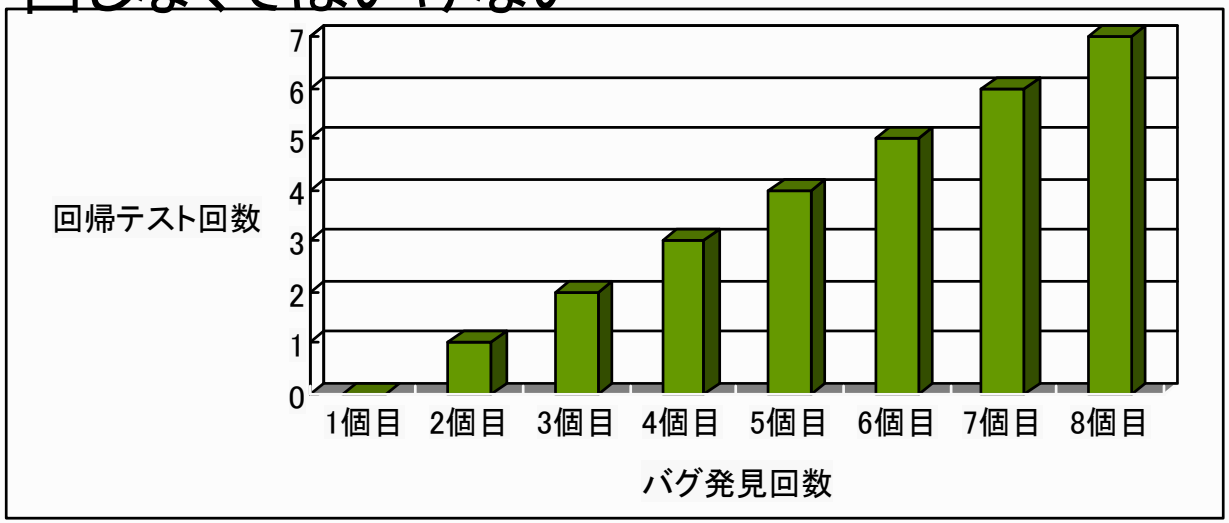
例：カードゲーム
（5枚1デッキ、数十種類のカード）
→デッキの組み合わせで…



n ×n ×m ×l ×o種類の組み合わせ！

人力テストの欠点

- 回帰テスト（以前起こったバグのテスト）が事実上困難
  - n個のバグに対する回帰テストはn-1の階乗回しなくてはいけない



人力テストはダメ？

これはこれで必要不可欠

- 人間ならではの複雑なテストが出来る

ただ…

これに頼ってだけでは破綻してしまう

- コストも
- 個人のリソースも

内容

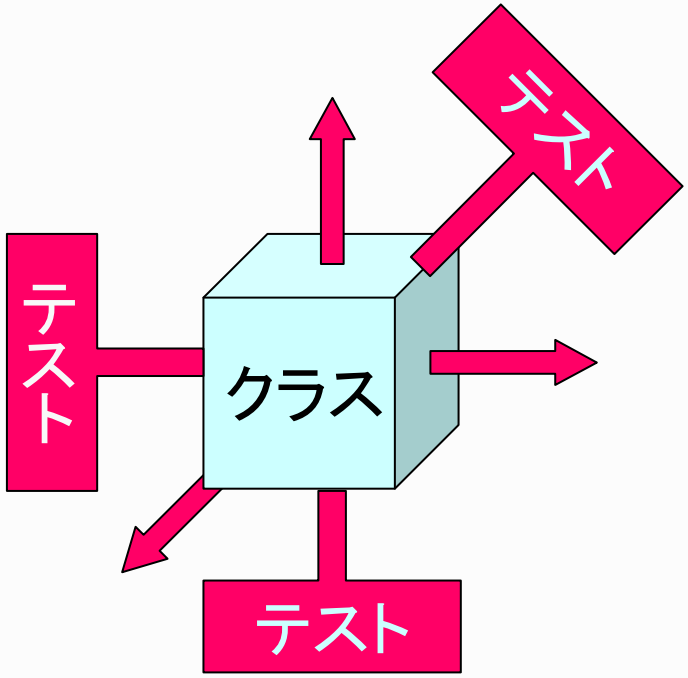
- バグとは
- バグを減らす方法
  - テスト
    - 人力テスト
    - ソフトウェアテスト
      - ユニットテスト
      - 統合テスト
    - 自動プレイテスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

アンケート1

- ユニットテスト知っていますか？
- ユニットテスト使ったことがありますか？

ユニットテストとは

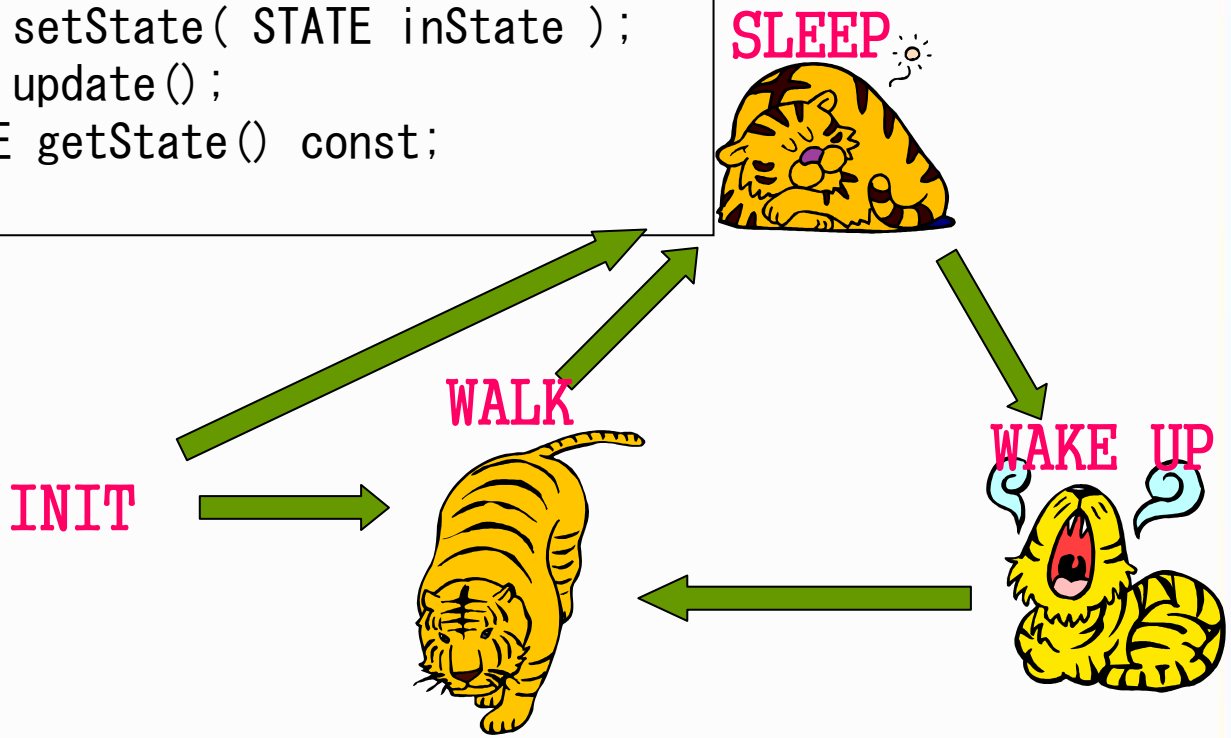
- 別名単体テスト
- １機能（≒１クラス）に対し、1つのテストプログラムを作る
- 「ひとつのユニット（＝クラス）を多方向から調査」するイメージ
- 通常テストフレームワークを使う
  - CppUnit (C++)
  - Google Test(C++)



ユニットテストとは(コーディング例)

- 例:状態遷移クラス

```
class State {
    void setState( STATE inState );
    void update();
    STATE getState() const;
};
```



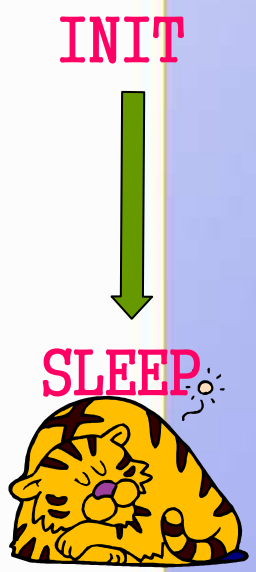
ユニットテストとは(コーディング例)

- 例:状態遷移クラス

```
class State {
    void setState( STATE inState );
    void update();
    STATE getState() const;
};
```

↓テスト実装

```
void StateTest::testSetState()
{
    State state;
    // 事前条件
    CPPUNIT_ASSERT_EQUAL( INIT, state.getState() );
    // 「SLEEP」をセットするとSLEEP状態になる
    state.setState( SLEEP );
    // 事後条件
    CPPUNIT_ASSERT_EQUAL( SLEEP, state.getState() );
}
```



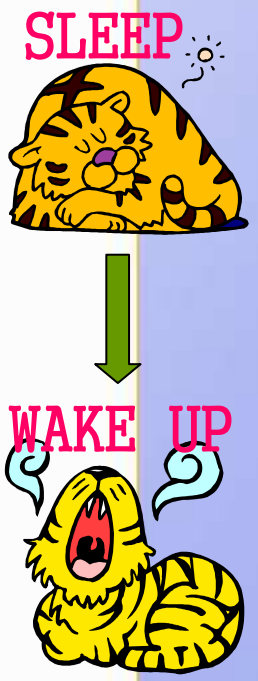
ユニットテストとは(コーディング例)

- 例:状態遷移クラス

```
class State {
    void setState( STATE inState );
    void update();
    STATE getState() const;
};
```

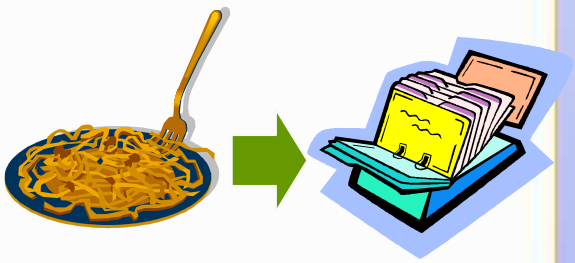
↓テスト実装

```
void StateTest::testUpdate()
{
    State state;
    // 事前条件
    state.setState( SLEEP );
    CPPUNIT_ASSERT_EQUAL( SLEEP, state.getState() );
    // １０回updateするとWAKEUP状態になる
    for( int i=0 ; i<10 ; i++ ) {  state.update(); }
    // 事後条件
    CPPUNIT_ASSERT_EQUAL( WAKEUP, state.getState() );
}
```



ユニットテストの効力

- 何度でも同じテストを行える
  →エンバグ（修正時にバグを混入させてしまう）防げる
- リファクタリングが容易
  →可読性の確保、再利用可
- テストを先に作って、それに合うように実装することも可能



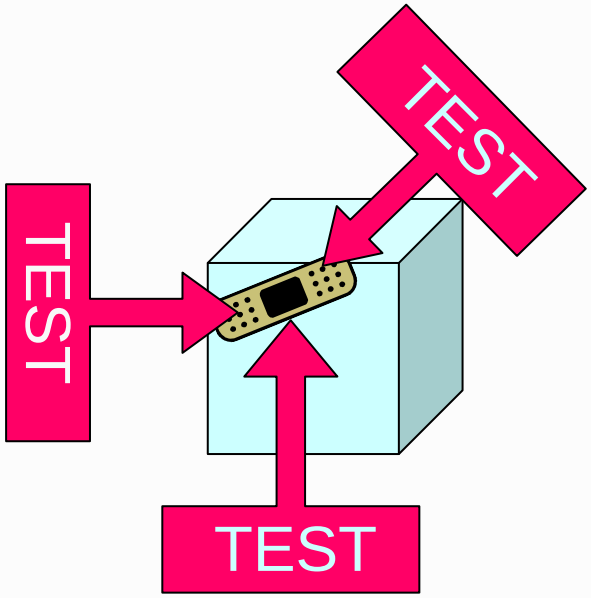
でもそれって…

- 疑問1、作業コスト多すぎない？
  - 1クラスに1テストということは作業量が倍？
- 疑問2、仕様変更の度にテストも変更？
- 疑問3、ゲーム開発では使えないのでは？
  - グラフィックの検証は不可？
- 疑問4、テストそのものにバグがあったらどうするの？
  - テストを作らないほうがよかったのでは…？

そんなことはありません！

疑問1、作業コスト多すぎない？

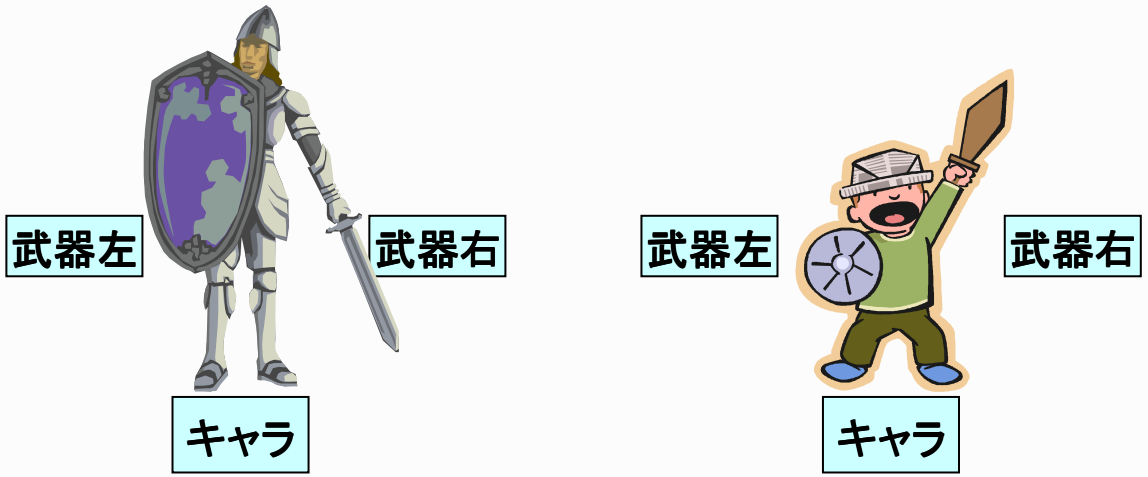
- 解決法1：不具合が実際に起こったところを重点的にテスト
  - 不具合状況の再現
  - 類似した不具合が起こりそうな状況もテスト



疑問1、作業コスト多すぎない？

- 解決法2:複雑な状況を全て網羅

例：武器の組み合わせの整合性をテスト



疑問1、作業コスト多すぎない？

- 解決法2:複雑な状況を全て網羅

  例:全ての武器の組み合わせで整合性をテスト

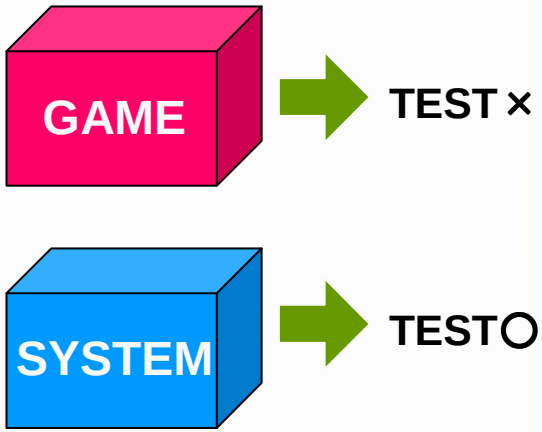
```
void MechaTest::testWeapon()
{
  WeaponChecker cheker( "check.xls" ); // 企画が作ったチェック用Excelを読込
  for( int i=0 ; i<MAX_WEAPON ; i++ ) {// 右手の武器のループ
    for( int j=0 ; j<MAX_WEAPON ; j++ ) {// 左手の武器のループ
      Mecha mecha;
      // 事前条件
      CPPUNIT_ASSERT_EQUAL( false, mecha.canUseWeaponL() );
      CPPUNIT_ASSERT_EQUAL( false, mecha.canUseWeaponR() );
      // メカに左右の武器をセット
      mecha.setWeapon( i, j );
      checker.set( mecha, i, j );
      // 事後条件
      CPPUNIT_ASSERT_EQUAL( checker.canUseWeaponL(), mecha.canUseWeaponL() );
      CPPUNIT_ASSERT_EQUAL( checker.canUseWeaponR(), mecha.canUseWeaponR() );
    }
  }
}
```

疑問2、仕様変更の度にテストも変更？

- 解決法：ゲーム部とシステム部の切り分け
  - ゲーム部＝流動的な箇所
    - 例：ゲームルール、演出、ストーリー、レベルデザイン

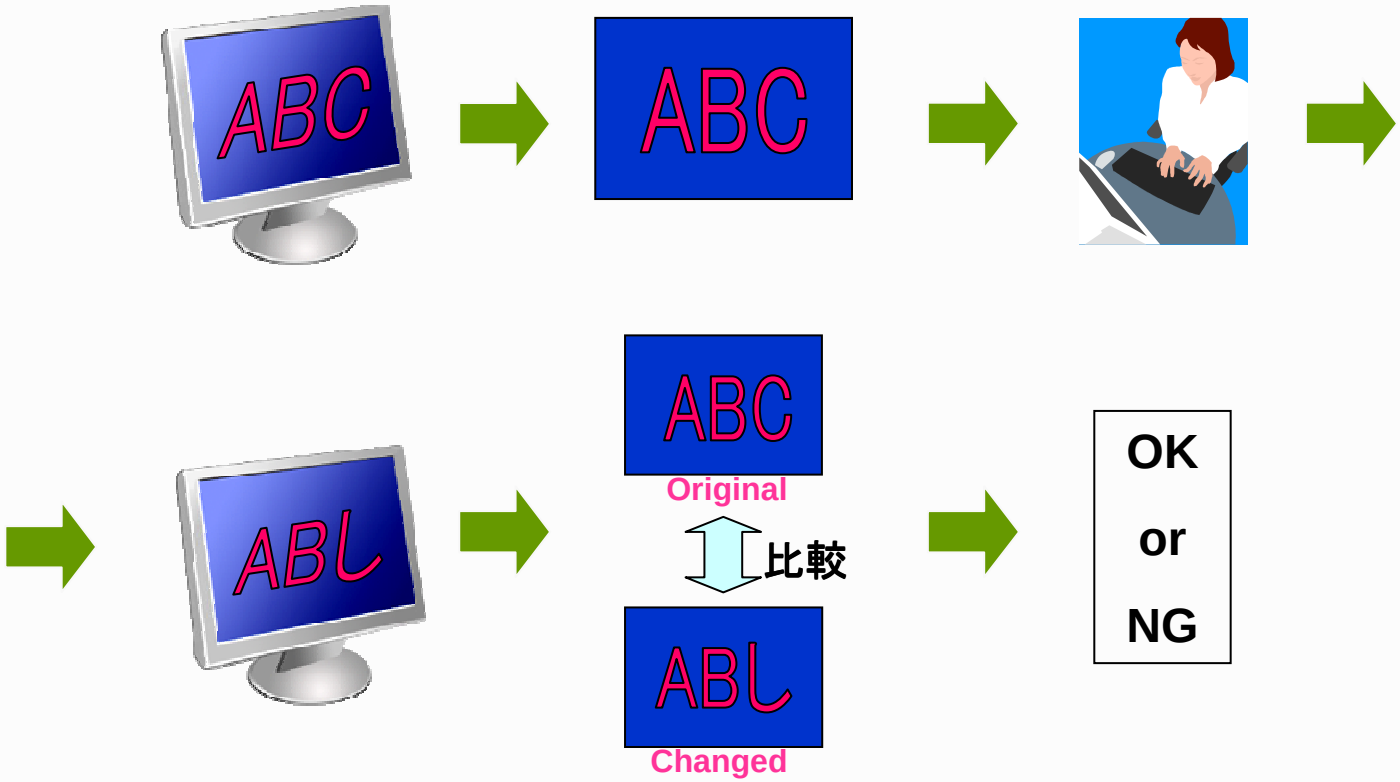
      →テストを作らない
  - システム部＝固定的な箇所
    - 例：算術ライブラリ、コリジョン、リソース管理、描画周り

      →テストを作る



疑問3、グラフィックの検証は不可？

- 解決法：スナップショットテスト



疑問4、テストそのものにバグがあったら？

I. テスト（バグなし）＋プログラム（バグなし）
→OK

II. テスト（バグなし）＋プログラム（バグあり）
→NG!

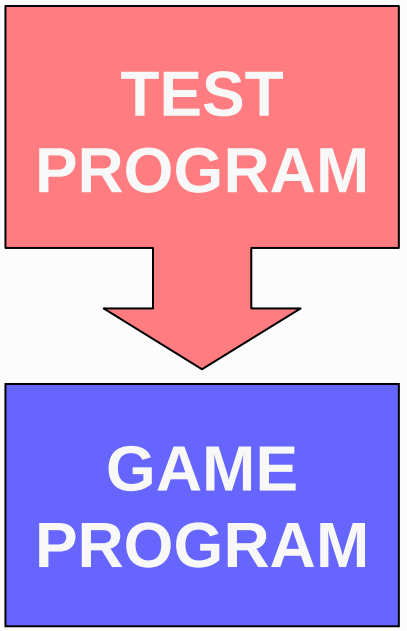
III. テスト（バグあり）＋プログラム（バグなし）
→NG! →あれ？良く見たら… →OK

IV. テスト（バグあり）＋プログラム（バグあり）
→OK ←間違い！ だが…

テストを作らなくてもスルーしていたバグ

疑問4、テストそのものにバグがあったら？

- テストプログラムは
  ゲームプログラムを
  一方的に使うだけ
  - ゲームプログラムを
    荒らすことはない
  - テストにバグがあっ
    てフリーズすること
    があってもゲーム
    に影響はない



その他効率的なテスト生成（VisualStudio限定）

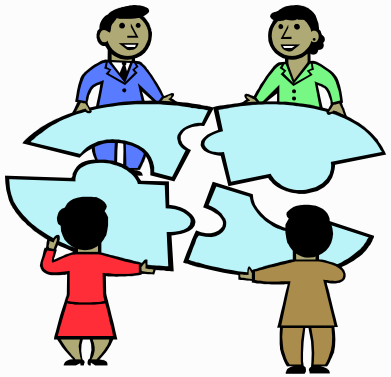
方法1、VisualStudioアドインの利用

内容

- バグとは
- バグを減らす方法
  - テスト
    - 人力テスト
    - ソフトウェアテスト
      - ユニットテスト
      - 統合テスト
    - 自動プレイテスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

統合テストとは

- 別名結合テスト
- 複数のクラスにまたがる状況をテスト
- ユニットテスト同様にテストフレームワークを利用
- 例：モデルファイルをリソース管理クラスで読み込み、キャラクタークラスに結びつける



統合テストはこれが大変！

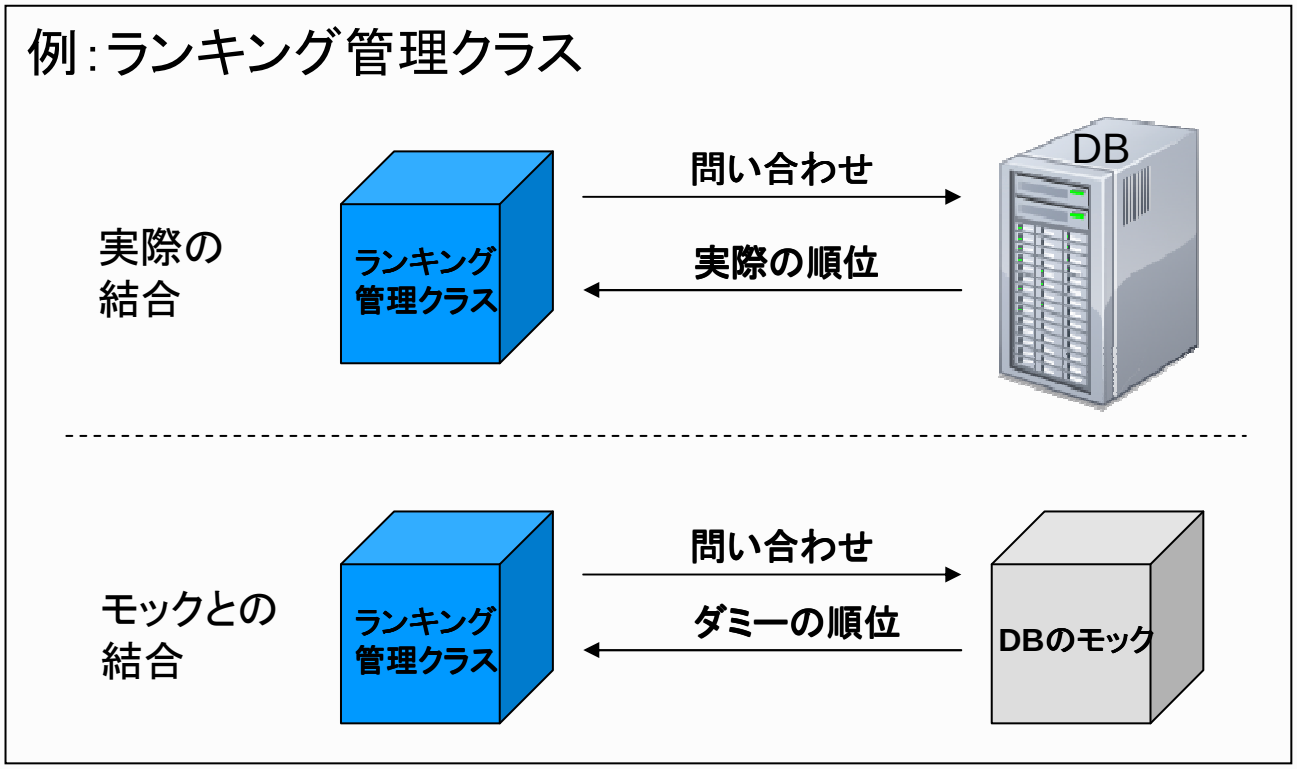
- それぞれのクラスの独立性の確保
  →むしろ望むところ

- DBやファイルアクセス等のゲームアプリ以外とのやり取りの再現

ゲームアプリ以外とのやり取りの再現

- 解決策：モック(Mockpp)の利用

例：ランキング管理クラス

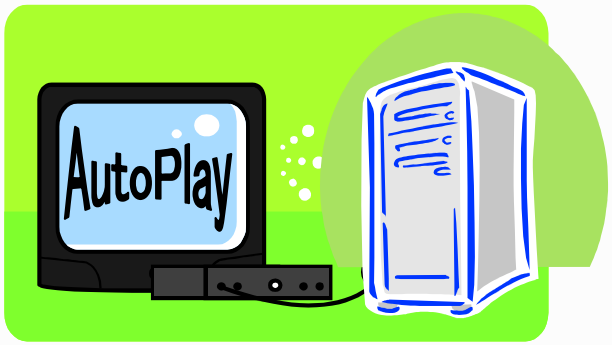


内容

- バグとは
- バグを減らす方法
  - テスト
    - 人力テスト
    - ソフトウェアテスト
    - 自動プレイテスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

自動プレイテストとは

- コンピューターに勝手にプレイさせて、正常に動くかどうかの確認をする
- テストコードを書く必要がない

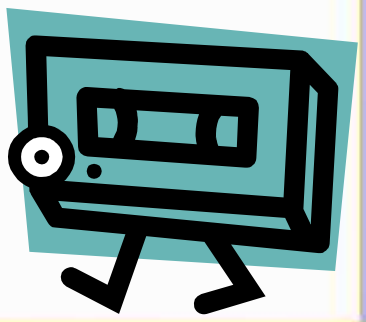


内容

- バグとは
- バグを減らす方法
  - テスト
    - 人力テスト
    - ソフトウェアテスト
    - 自動プレイテスト
      - リプレイテスト
      - AIテスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

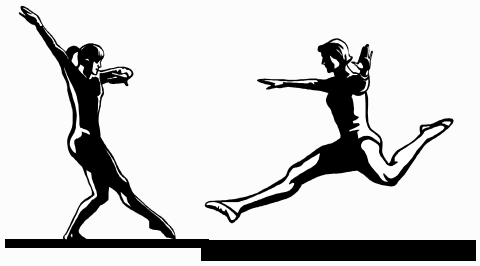
リプレイテストとは

- 適当なゲームプレイを人の手で行い、リプレイデータを保存
- 保存したリプレイデータを使って再生させる



リプレイテストとは

- 開発時期でリプレイする内容を変える（ムダを排除）
  - 開発初期：起動＆終了確認
  - 開発中期：適当なアクションが可能かどうか（剣を振る、ジャンプする）
  - 開発後期：最初から最後まで通したプレイ



リプレイテストとは

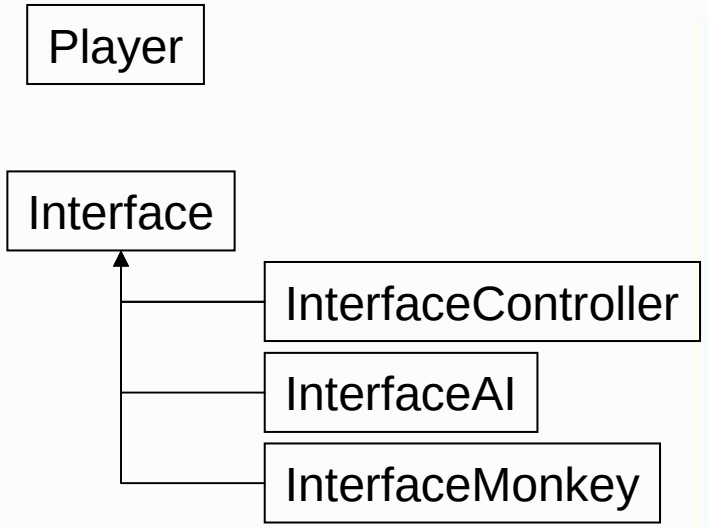
- テスト内容
  - 正常に起動終了すればOK
  - ログで比較
  - 定期的スナップショットで比較

内容

- バグとは
- バグを減らす方法
  - テスト
    - 人力テスト
    - ソフトウェアテスト
    - 自動プレイテスト
      - リプレイテスト
      - AIテスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

AIテストとは

- プレイヤーの代わりとなるAIを作る
- AIに一定時間プレイさせる
- もっと簡単にモンキーテスト（完全にランダムにコントロール）でも可



自動プレイテストの効力

- CIと組み合わせれば、ほとんど人の手を使わず整合性をテストできる
- AIテストはやればやるほど効力が増す
  - 夜中に全員のPCで稼動するとか
  - 土日はフルタイムをAIテストに費やすとか

テストのまとめ

- 人力テストだけでは昨今の複雑なゲーム仕様を網羅できない
- テストの多様性がバグを減らすカギ
- 以下の二つのテストも加えましょう
  - ソフトウェアテスト
    - テストのプログラムを作成
  - 自動プレイテスト
    - 人の手を介さずにテストする

コーヒーブレイク

- 過去、うまいことを言った人(PG)が…
- 面白ければ…
  - ディレクターのおかげ
- 絵が綺麗ならば
  - デザイナーのおかげ
- ゲームが売れたら
  - プロデューサーのおかげ
- バグが出たら
  - プログラマのせい

内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

CIとは

- 継続的インテグレーション/

  Continuous Integration /常時結合

- 「常にゲームプレイ可能な状態」を提供する
  - 処理落ち　→　〇
  - 仕様違い　→　〇
  - フリーズ　→　×
  - アサートで止まる　→　×

  →つまり進行できればオッケーということ

CIを導入する背景

各自の環境で作業

↓

リリース直前に結合

↓

結合した途端バグが多発

↓

メインPGが泣く

CIを導入するとどうなるのか

各自の環境で作業

↓

サーバにあげると同時に結合

↓

バグが出れば差し戻し

常に正常な状態を提供できる

内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
    - Hudsonの紹介
    - やれること
    - 何が素敵なのか
  - 人材育成
- いかに広めるか
- まとめ

Hudsonとは

- Hudsonの特徴
  - 日本人が開発
    - 日本語ドキュメントが豊富
  - インストールが簡単
    - Webサーバーの知識がなくても導入可能
  - 自由度が高い

Not…

アンケート2

- Hudson等のCIツールについて
  - 聞いたことがありますか？
  - 使っていますか？

内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
    - Hudsonの紹介
    - やれること
    - 何が素敵なのか
  - 人材育成
- いかに広めるか
- まとめ

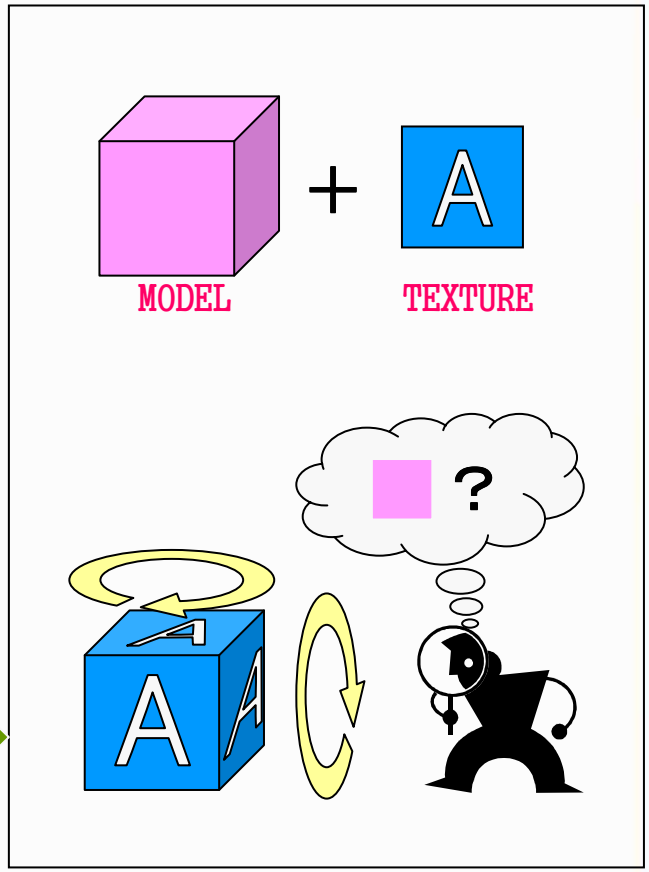
ゲーム開発的Hudson利用法

- やれること（以下のことを毎日）
  - サーバからソースを取得
  - 静的解析ツール
  - ビルド（サンプル等も同時に）
  - テスト（ユニットテスト、自動プレイテスト等）
  - ソースのコミット
  - デザインデータコンバート
  - 最終データ収集
  - NGだったら担当者へメール

:

Hudsonでデザインデータコンバート

- 更新状態を常にチェック
- データが更新されたら即コンバート
- 終了したらデータの整合性テスト
  - モデルデータを表示させる
  - テクスチャの貼り忘れ等の検出

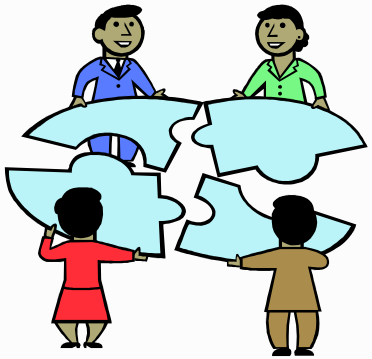


内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
    - Hudsonの紹介
    - やれること
    - 何が素敵なのか
  - 人材育成
- いかに広めるか
- まとめ

CIの何が素敵なのか

- 結合時の障害を最小限にできる



- いつでも誰でもプレイできる
  - 偉い人がいつ来ても対応
  - 常に動く状態、最新の状態を維持

CIのまとめ

- Hudsonを今すぐに導入しましょう
  - 低コストで安心を得られます
- 自動でできることは全て自動で

内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

人材育成の意義

- 社員の勉強なくして、バグ削減なし！
- 勉強
  - 新人研修
  - コードレビュー
  - 勉強会

新人研修

PR

- 本
  - 「ゲームプログラマになる前に覚えておきたい技術」
  - 著者：平山 尚（株式会社セガ）
  - 価格：￥4725
  - 「ページ数が厚くて、内容が薄い」＝すぐ読める
  - 全国の書店でお買い求めください

プレゼント！

If(この本を持っていない)
　If (　業界歴　＜＝　3年　|| 学生)
　　If (CEDECは自腹だ)
　　If (というか欲しい？)
　　If (　サインありだけど？)
　　その中で一番前の人！
　　プレゼント！

コードレビュー

- 本来、コードレビューはコードのチェック機能として行うもの

- もっと単純に「プログラミングについて話し合う場」として、勉強する機会としてみてはどうでしょうか

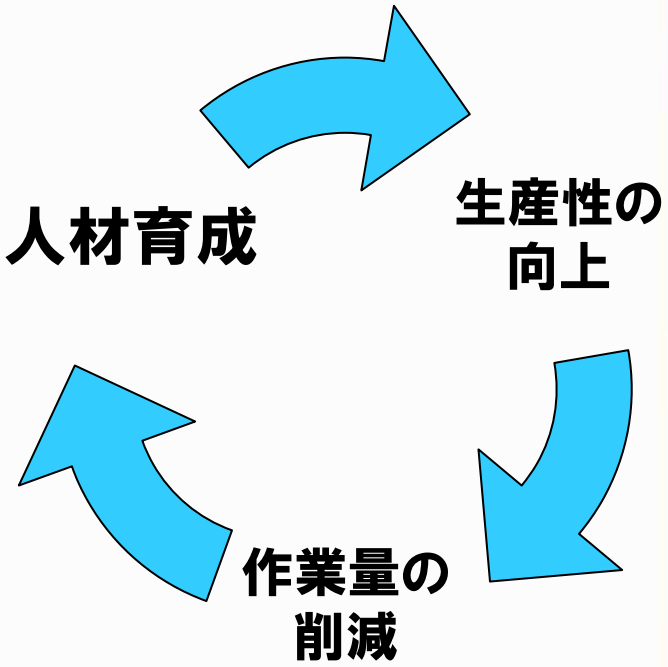
勉強会

- セミナー形式
- ラウンドテーブル形式
- 輪講形式

- 海外のゲーム会社の場合
  - 講演者は自分でWebに内容を登録
  - 受講者は受けたい講演に投票
  - 投票の高い講演を実際に行う

人材育成のまとめ

- ツールだけではバグはゼロになりません！
- 正のスパイラルでバグを作らない人材を育成していきましょう！



内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
  - その他
- いかに広めるか
- まとめ

静的解析ツール

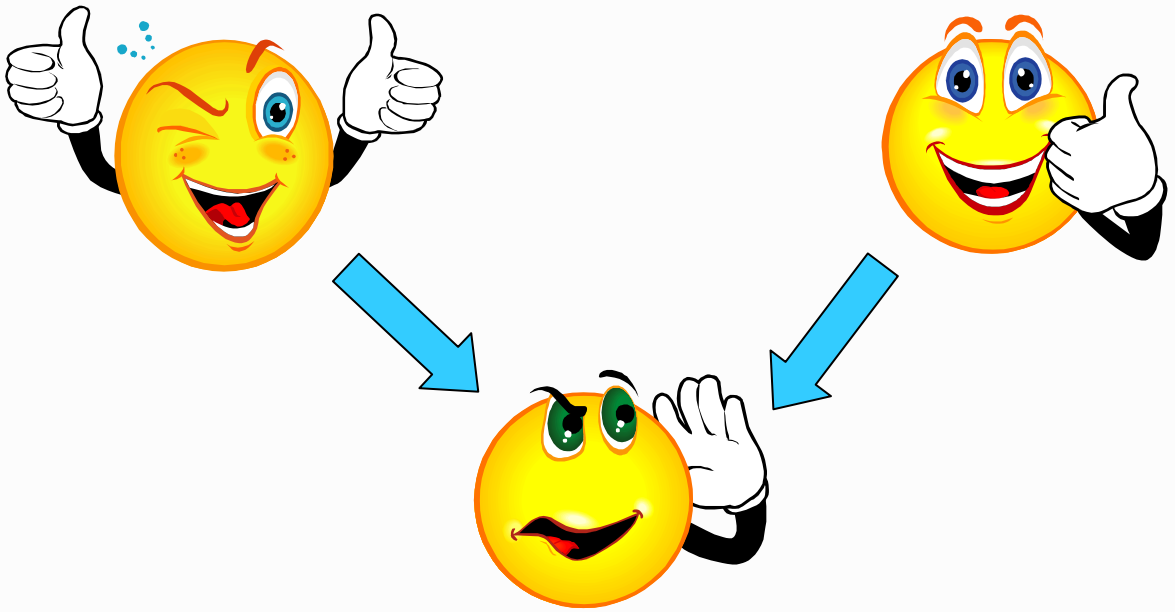
- プログラムを解析して、バグを検出してくれるツール
- お金があるならとってもオススメ！
  →フリーのも…
- ただし運用法がカギ
  - 誤検出とか
  - 修正フローの徹底とか

内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

「広めること」の意義

- 多方面から「良い」と聞くと信頼度が上がる



いかに「テスト」を広めるか

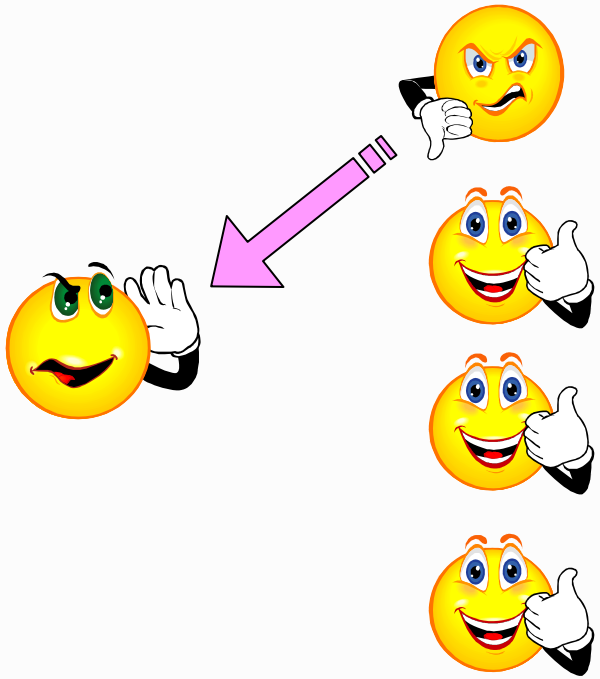
- プログラマ向け
  - いかにしてテストを楽に作るか
  - 「テストを一度作るとこんなに便利！」
  - 一度痛い目に会うと…
- 企画向け
  - 「ギリギリでもある程度修正が効きます」
- マネージャー向け
  - 「余計な費用を払わずにバグが減ります」

いかに「自動化・CI」を広めるか

- プログラマ向け
  - 一日の無駄な作業 × 日数 × 人数
    → 膨大な時間を他に割り当てましょう
- デザイナー向け
  - クリエイティブじゃないところはとことん自動化可能
    - 規則的な音がしたら要注意
- 企画
  - 常にゲームが確認できます
- プロデューサー
  - いつ偉い人が来ても大丈夫です

いかに「良いもの」広めるか

- マネージャーに対して
  - 良いものは良いと伝えましょう
  - なぜなら悪い情報ばかり上に上っていくから



内容

- バグとは
- バグを減らす方法
  - テスト
  - CI
  - 人材育成
- いかに広めるか
- まとめ

まとめ

- さまざまな方法を組み合わせることでバグをゼロに近づけましょう
  - ソフトウェアテスト
  - CI
  - 人材育成
  - Etc…
- そして、最後にユーザーにバグのないゲームを提供しましょう