コーディングとテストの並列開発手法実現のための一考察

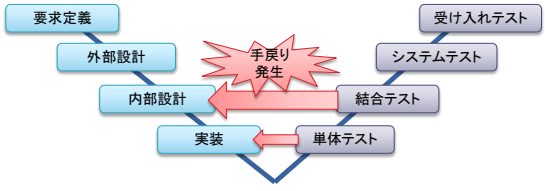
宮崎大学工学部情報システム工学科
大久保暢人、松岡慎吾、喜多義弘、片山徹郎

背景

- ソフトウェアの**大規模化・複雑化**
  - 高機能化・高性能化によるもの
  - 開発にかかる人員や時間が増加する
- **短納期化**の傾向
  - 新規開発のサイクル短縮
  - 開発にかかる時間を短くせざるを得ない
- **ソフトウェアテスト**
  - ソフトウェアの信頼性を上げるために不可欠
  - 手間と時間がかかる

⇒効率の良いソフトウェア開発が求められる

背景-V字モデル



- 単体テストを繰り返すことが重要
  - バグの早期発見につながる
  - 後工程で見つかった欠陥ほど修正コストがかかる

目的

- 大規模化しているソフトウェア開発の生産性を上げたい
- 短納期化している開発の工数に余裕を持たせたい

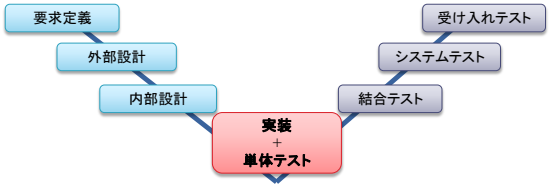
⇩

- テストを自動化する
- できるテストを前倒しで実施する

⇩

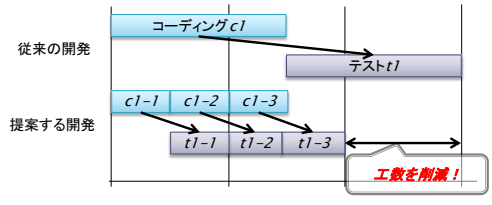
- **コーディングとテストの並列開発手法実現に向けた考察を行う**

コーディングとテストの並列化



- 実装と単体テストを並列して行う
  - コーディング中にバックグラウンドで単体テストを自動で実行
  - 対象は既に実装済みかつテストを行っていないモジュール

コーディングとテストの並列化



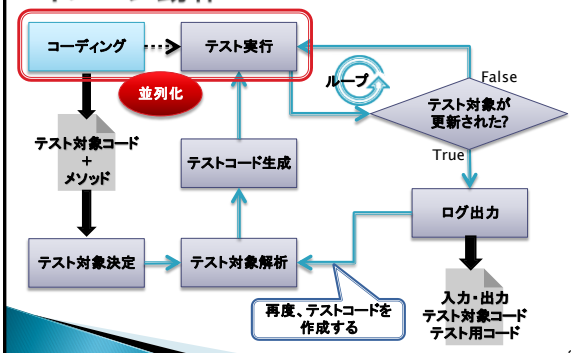
- 実装済みのモジュールから随時テストを実行する
  ⇒工数を削減できる

テスト内容

- 対象言語：Ruby
- 自動でテストコードを実行
- ランダムテスト
  - 入力の型に合わせた乱数値を生成
- テスト対象が更新されたらテストコードも更新
  - テスト対象のファイルを一定時間ごとに監視

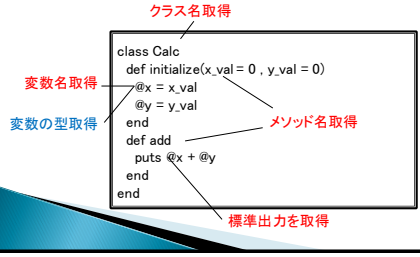
- これらを**コーディングと並列して行う**

イメージ動作



テスト対象モジュールの解析

- 指定したモジュールを**正規表現**で解析
- 変数の型の取得には**動的解析**を行う



正規表現

- 文字列の集合を一つの文字列で表現する方法

- Rubyは強力な正規表現が使える
  - **クラス名**（例“class Calc”）
    ⇒class¥s+[A-Z]¥w+(¥s+<¥s+[A-Z]¥w+)*

  - **条件分岐**（例“if val == 0”）
    ⇒if¥s*¥(*¥s*¥w+¥s*((>|<|>=|<=|!=|==|=~|===)¥s*("|')*¥w*("|')*¥s*)*¥)*

動的解析

- 型に合わせたランダム値を生成したい
- 変数を宣言しないので静的解析では型がわからない
  ⇒**classメソッド**を用いる

```
123.class
⇒ Fixnum(整数)
“hoge”.class
⇒ String(文字列)
[1,2,3].class
⇒ Array(配列)
3.14.class
=>Float(浮動小数)
10000000000000000.class
=>Bignum(多倍長整数)
```


テストコード生成規則

置換前	置換後	説明
class [クラス名]	class MySample	クラス名をテスト用のクラスに置換
module [モジュール名]	my[メソッド名]	モジュール名をテスト用のモジュールに置換
def [メソッド名]	my[メソッド名]	メソッド名をテスト用のメソッドに置換
[変数] = [整数]	[変数] = MyIntRand()	変数に乱数(整数値)を入力
[変数] = [小数]	[変数] = MyFltRand()	変数に乱数(小数値)を入力
[変数] = [文字列]	[変数] = MyStrRand()	変数に乱数(文字列)を入力
[変数] = [配列]	[変数] = MyAryRand()	変数に乱数(配列)を入力
print [標準出力]	MyOutputs << [標準出力]	標準出力を配列に格納する
puts [標準出力]		

⋮

テストコード生成例

- ファイルが更新されるたびに生成する

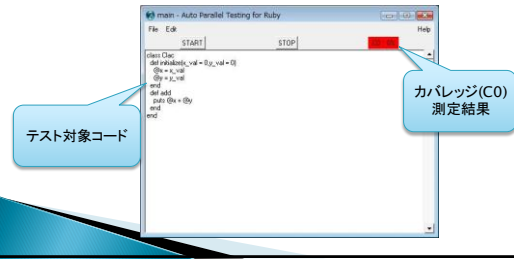
```
class Calc
  def initialize(x_val = 0 , y_val = 0)
    @x = x_val
    @y = y_val
  end
  def add
    puts @x + @y
  end
end
```

```
class MySample
  def initialize(x_val = 0 , y_val = 0)
    @x = MyIntRand()
    @y = MyIntRand()
  end
  def add
    MyOutputs << @x + @y
  end
end
```

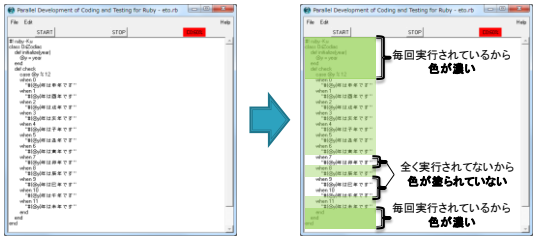
置換前	置換後	説明
class [クラス名]	class MySample	クラス名をテスト用のクラスに置換
[変数] = [整数]	[変数] = MyIntRand()	変数に乱数を入力
puts [出力]	MyOutputs << [出力]	出力を配列に格納する

テストコード実行

- テスト対象が更新されるまでランダムテスト実行
- カバレッジ(C0)100%を満たしたら赤→緑で表示
- 実行された行は網掛けされ、**徐々に濃くなる**



テストの進捗状況の確認



実行前　　実行後

結果・ログ出力

- テストコードが更新されるたびにディレクトリを作成
  - APTR_YYYYMMDDHHMM_[テスト対象ファイル名]
    例) APTR_201110251808_autotest
- 作成されたディレクトリに実行された結果を出力する
  - **テスト項目**(CSVファイル)
    - テスト実行時に生成された乱数
    - 標準出力
  - **テスト用コード**(Rubyファイル)
  - **テスト対象コード**(Rubyファイル)

まとめ

- コーディングとテストの並列開発手法実現に向けた考察を行った
  - 実装済みのモジュールから自動実行
  - 工数の削減につながる
- テスト自動化
  - 入力の型に合わせた乱数値をテストデータとして扱う
  - 生成されたテストデータをログとして出力
- 効率的なソフトウェア開発が可能になる
  - 開発にかかる時間やコストの肥大化を解消できる

今後の課題

- ツールの実装
  - 今回の考察通りの動作が実現できるか
- 実用性の検証
  - 実際にどれくらいの効果があるのか
  - 改善するべきところはないか
- ブランチカバレッジ(C1)への対応
  - ステートメントカバレッジ(C0)ではプログラム構造まで検証できない
  - 分岐条件が正しいか、分岐がすべて網羅されているかを検証する
- ランダムテスト+αの自動実行
  - ランダムテストだけではなかなか網羅しきれないテストを行う
  - 同値分割、境界値分析、一般的にエラーの多い入力など・・・