

ソフトウェア開発技術

メトリクスとは？

前回までの内容

前回いろんなことを学びました！

- ・なぜテストが必要なのか？
- ・FindbugsやCheckStyleなどの静的ツールの使い方
- ・テストケースの考え方(境界値分析法など)

そして、総仕上げとして前回までの3コマを使って
実際にテスト仕様書の作成・テストの実施を行いました

今日のゴール・・・

自分のソースコードの品質を確認しよう！

目次

- 1. ソースコードの品質？**
- 2. ツールの使い方**
- 3. 実際にやってみよう**

目次

- 1. ソースコードの品質？**
2. ツールの使い方
3. 実際にやってみよう

1. ソースコードの品質？

突然ですが、次のソースコードどっちが見やすい？

```
private int func(Player[] p){
    for( int i = 0; i < 9; i++ ){
        if( p[i] != null ){
            p[i].setAtBat(0);
            p[i].setHitNum(0);
            p[i].setStealBase(0);
            p[i].setHomrun(0);
            p[i].setRuns(0);
            p[i].setBB(0);
            p[i].setHBP(0);
            p[i].setSO(0);
        }
    }
}
```

```
private int func(Player[] p){
    for( int i = 0; i < 9; i++ ){
        init( p[i] );
    }
}

private void init(Player player){
    if( player == null ){
        return;
    }
    player.setAtBat(0);
    player.setHitNum(0);
    player.setStealBase(0);
    player.setHomrun(0);
    player.setRuns(0);
    player.setBB(0);
    player.setHBP(0);
    player.setSO(0);
}
```

1. ソースコードの品質？

これはどう？

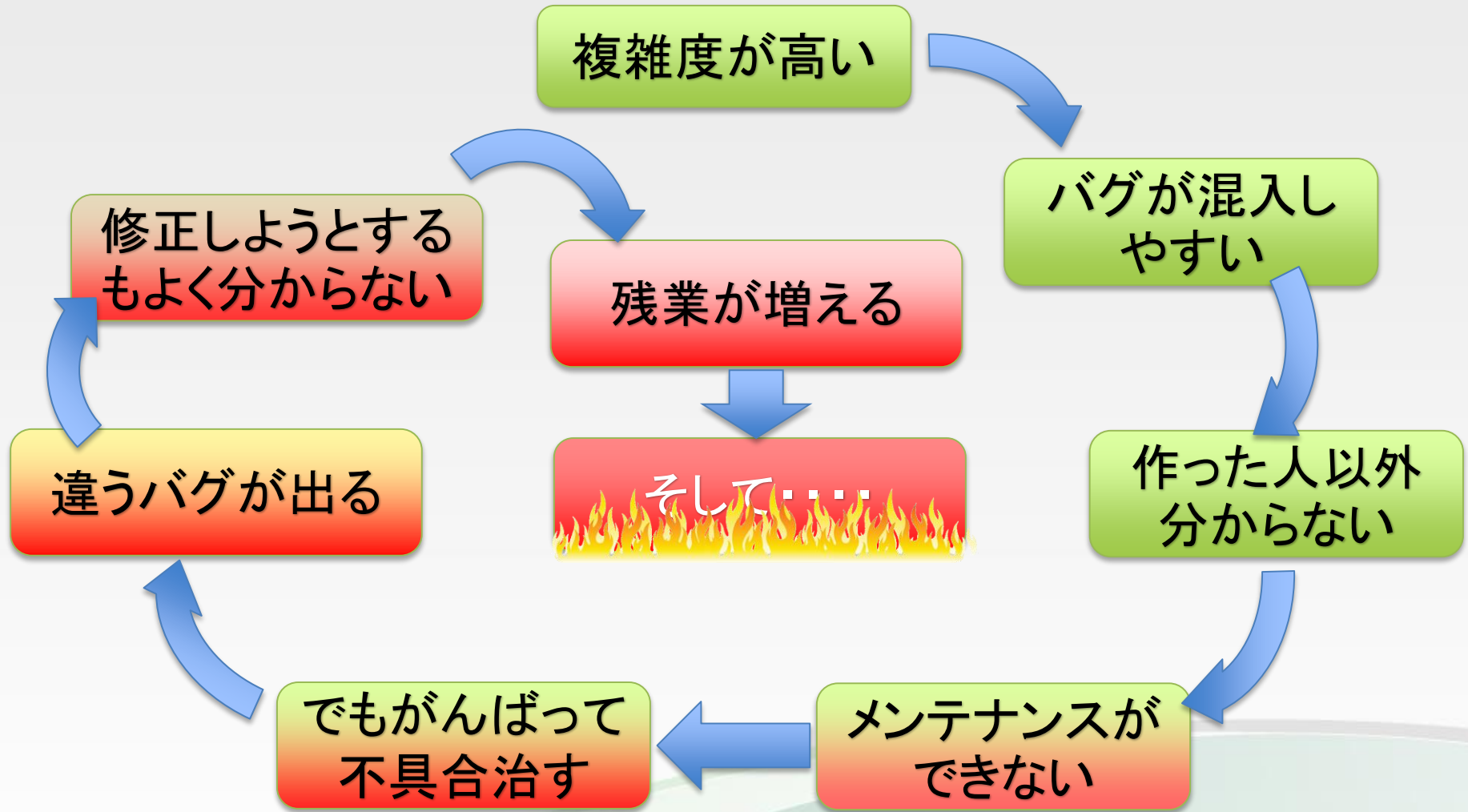
```
private int getdaze(boolean getFlg){
    if( getFlg ){
        if( kind == 0 ){
            if( megaFlg ){
                println (“ピカチュウ”);
            }else{
                println (“ピカチュウ”);
            }
        }else if( kind == 1 ){
            if( megaFlg ){
                println (“ゲッコウガ”);
            }else{
                println (“ゲッコウガ”);
            }
        }
    }else{
        println (“ゲットならず”);
    }
}
```

```
private int getdaze(boolean getFlg){
    if( getFlg ){
        chek( kind, megaFlg );
    }else{
        println (“ゲットならず”);
    }
}
private void chek( int kind,boolean megaFlg){
    if( kind == 0 ){
        if( megaFlg ){
            println (“ピカチュウ”);
        }else{
            println (“ピカチュウ”);
        }
    }else if( kind == 1 ){
        if( megaFlg ){
            println (“ゲッコウガ”);
        }else{
            println (“ゲッコウガ”);
        }
    }
}
```

1. ソースコードの品質？

**複雑度は、一般的にif(分岐)の数やネストの深さ
などが多い(深い)ほど、高くなります。**

1. ソースコードの品質？



1. ソースコードの品質？

IT業界の格言

プログラマー半年経ったら自分も他人

自分が書いたプログラムも半年後にみると他人が書いたコードに見える(それだけ忘れている)



1. ソースコードの品質？

ポイント

複雑度が低いプログラムを作るのは必須。

その品質を高さを測るには、eclipseのプラグインを使って計測することがあります。

計測ツールは色々ありますが、今回は

metrics

というeclipseのプラグインを使います。

目次

1. ソースコードの品質？
- 2. ツールの使い方**
3. 実際にやってみよう

2. ツールの使い方

では、ツールをインストールしましょう。

(1) metricsプラグインをゲットだぜ(自分のPCにコピー)

【方法1】

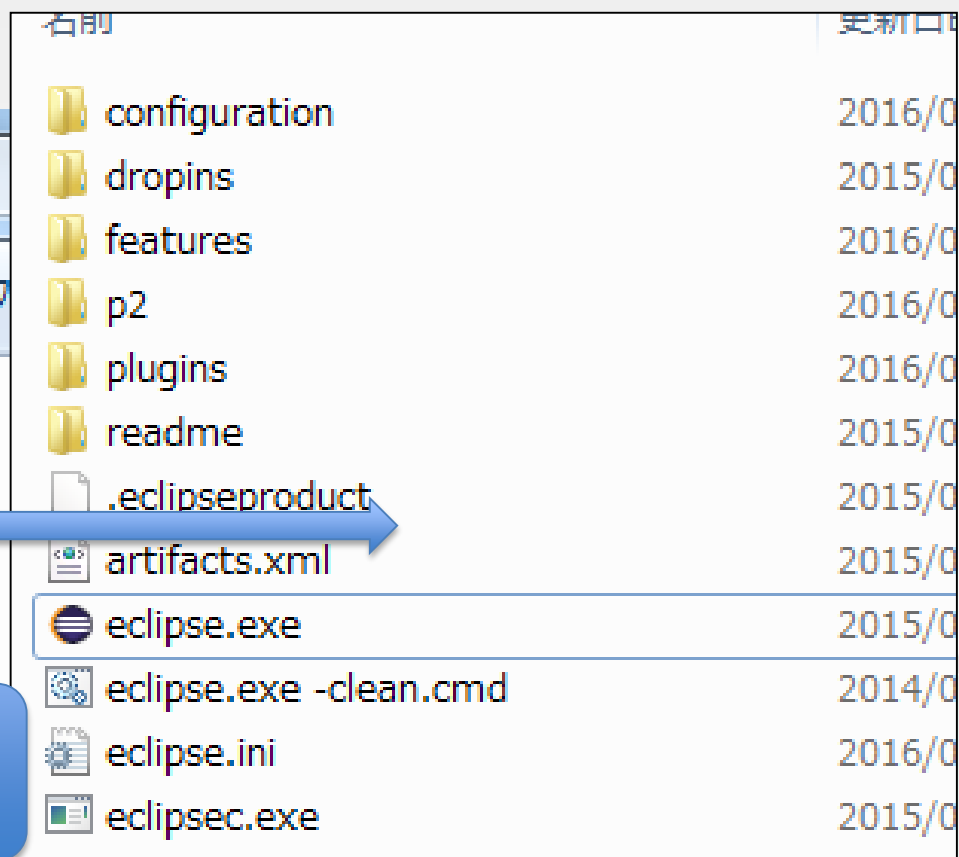
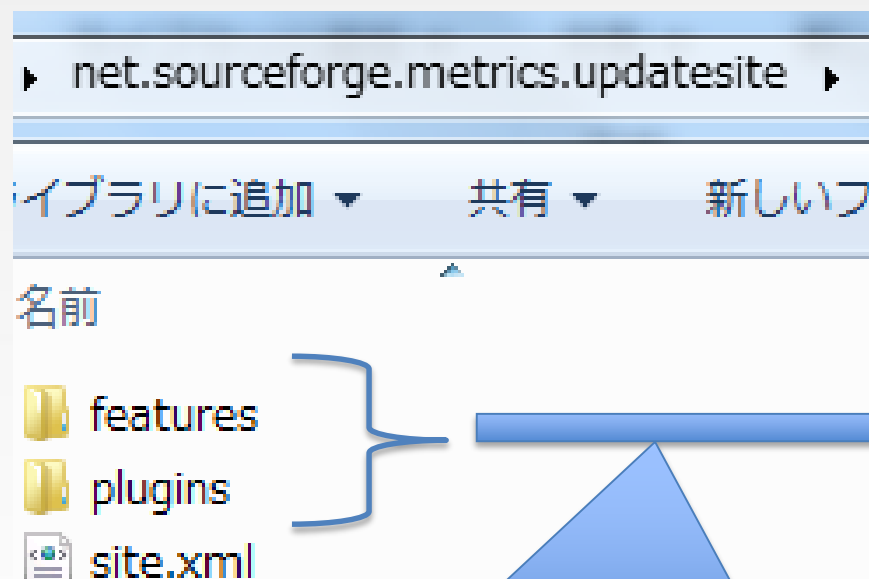
¥¥634sv¥share¥nishino¥ソフトウェア開発技術¥99_sample からupdatesite_1.3.6.zipをゲット

【方法2】

<https://github.com/nishino-naoyuki/tools/>
からupdatesite_1.3.6.zipをゲット

2. ツールの使い方

(2) updatesite_1.3.6.zip を解凍する
解凍した puginフォルダ feature フォルダを
eclipseにコピーする



フォルダごとコピー
「フォルダの上書き確認」は「はい」でOK！

2. ツールの使い方

(3) インストールは以上で終了！

ね、簡単でしょ？！(by ボブ・ロス)



目次

1. ソースコードの品質？
2. ツールの使い方
- 3. 実際にやってみよう**

3. 実際にやってみよう



ために、プロジェクト開発で作った自分のソースコードの複雑度を確認してみましょう！

まずは、計測の為の準備をします。

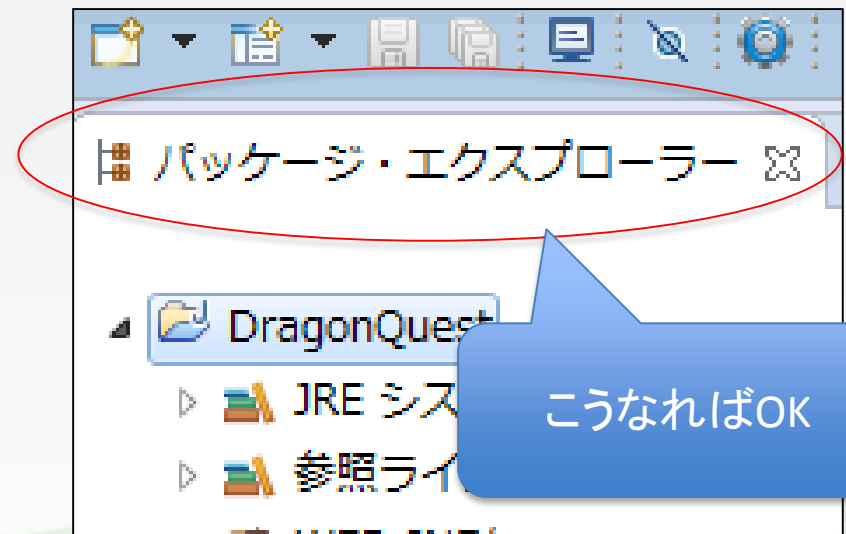
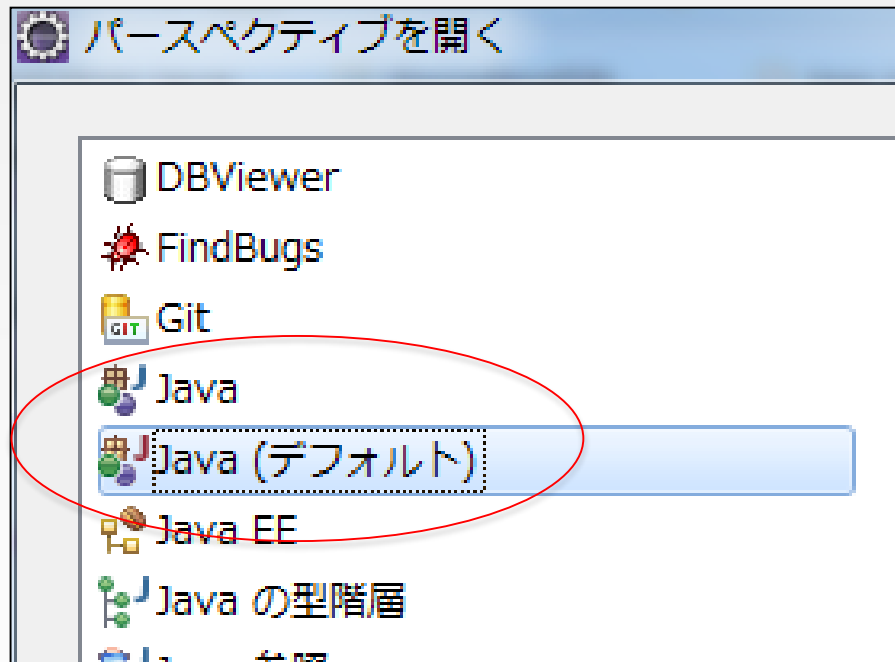
(1)eclipseを起動してPJ開発で使っているソースを開きましょう。(今日手元にはない人は、今まで作った何かを使いましょう)

3. 実際にやってみよう



(2) パッケージエクスプローラーを開きます

「ウインドウ」－「パースペクティブ」－「パースペクティブを開く」－「その他」で **java**を選ぶ



3. 実際にやってみよう



(3) プロジェクトを選択して、右クリック「プロパティ」でプロパティ画面を表示させ、ツリーから「metrics」を見つけ「Enable metrics」にチェックを入れる

The screenshot shows the 'DragonQuest のプロパティ' (Properties of DragonQuest) dialog box. On the left, a tree view lists various metrics, with 'Metrics' selected and circled in red. On the right, the 'メトリック' (Metrics) section shows the 'Enable Metrics' checkbox checked, also circled in red. A blue callout bubble points to the 'プロパティ(R)' (Properties) option in the right-click context menu, with the text '右クリックのプロパティ' (Right-click Properties). Another blue callout bubble points to the 'Enable Metrics' checkbox, with the text 'これにチェック' (Check this).

3. 実際にやってみよう



(4)「ウインドウ」-「ビューの表示」-「その他」 で出た画面で「Metrics View」を選択する

The screenshot shows the Eclipse IDE interface. On the left, the 'View' menu is open, displaying a list of views under the 'Metrics' folder. The 'Metrics View' is highlighted with a red circle. A red arrow points from this circle to the 'Metrics View' tab in the IDE's tab bar. A blue callout bubble points to the 'Metrics View' tab with the text: '画面下部に「Metrics View」のタブが出ればOK！' (If the 'Metrics View' tab appears at the bottom of the screen, it's OK!). Another blue callout bubble points to the 'Metrics View' in the menu with the text: 'これを選択' (Select this).

ビューの表示

フィルター入力

- Maven ワークスペース・ビルド
- メトリック
 - ☒ Dependency Graph View
 - ☒ Layered Package Graph View
 - ☒ Layered Package Table View
 - ☒ Metrics View
- Mylyn
- QuickREx
- SVN

```
EntryInfoDto entryInfoDto = new EntryInfoDto();  
//数値変換  
int sex = Integer.parseInt(req.getParameter("sex")
```

Metrics View

画面下部に「Metrics View」のタブが出ればOK！

これを選択

3. 実際にやってみよう



**これで準備完了！
いよいよ計測しましょう！！**

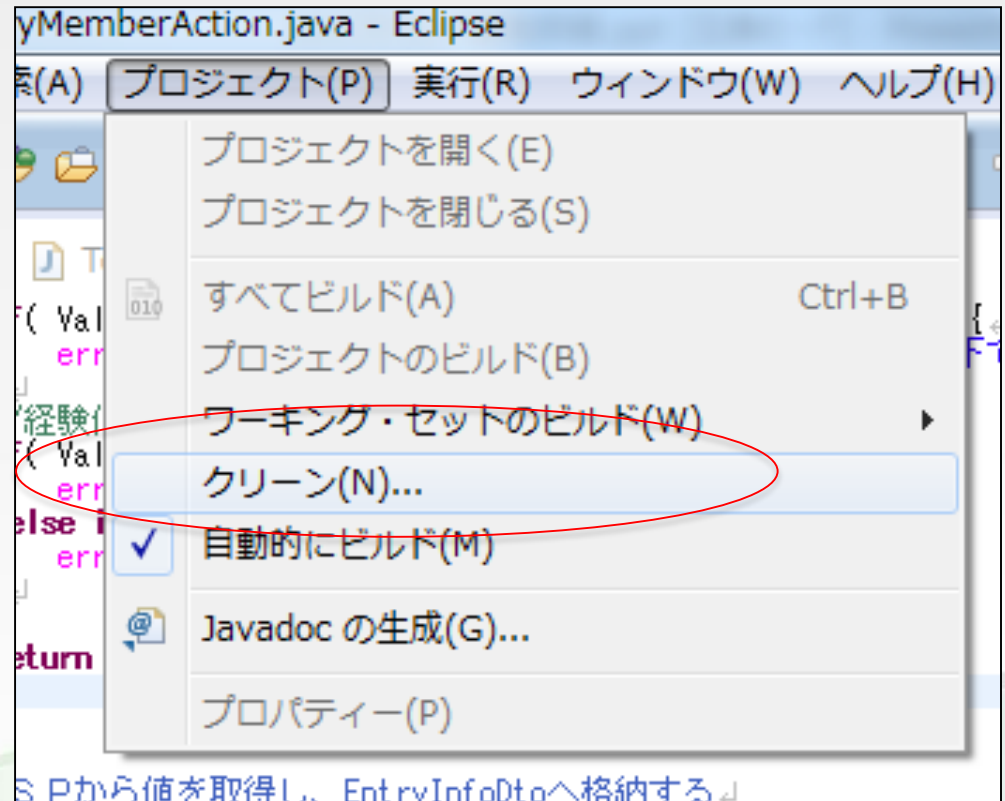


3. 実際にやってみよう



計測はソースをビルドすることで自動で計測が始まります。 ですので、**リビルド**を行きましょう！

「プロジェクト」 –
「クリーン」を選択
することで**リビルド**
が走ります！



3. 実際にやってみよう



計測が終わると、Metrics Viewの表示が変わります↓

メトリック	合計	Mean	Std. ...	最大	Resource causing Maximum
▶ パラメーター数 (avg/max per method)		0.971	0.822	3	/DragonQuest/WEB-INF/src/j
▶ Number of Static Attributes (avg/max per	26	3.714	7.905	23	/DragonQuest/WEB-INF/src/j
▶ 遠心性結合 (avg/max per packageFragmer		0.2	0.4	1	/DragonQuest/WEB-INF/src/j
▶ Specialization Index (avg/max per type)		0.929	1.321	3	/DragonQuest/WEB-INF/src/j
▶ Number of Classes (avg/max per package	7	1.4	0.8	3	/DragonQuest/WEB-INF/src/j
▶ Number of Attributes (avg/max per type)	9	1.286	3.149	9	/DragonQuest/WEB-INF/src/j
▶ Abstractness (avg/max per packageFragm		0	0	0	/DragonQuest/WEB-INF/src/j
▶ Normalized Distance (avg/max per packag		0.4	0.49	1	/DragonQuest/WEB-INF/src/j
▶ Number of Static Methods (avg/max per t	5	0.714	1.75	5	/DragonQuest/WEB-INF/src/j

3. 実際にやってみよう



項目の説明(抜粋)

項目名	説明
Total Lines of Code	クラスのコード行数 (コメント行・空行は含まない)
Method Lines of Code	メソッドのコード行数 (コメント行・空行は含まない)
Nested Block Depth	メソッド中の最大のネスト数
McCabe Cyclomatic Complexity	メソッドの複雑度 10 以下であればよい構造 30 を越える場合, 構造に疑問 50 を越える場合, テストが不可能 75 を越える場合, いかなる変更も誤 修正を生む原因を作る
Number of Parameters	メソッドのパラメータ数

他の項目は以下を参照

http://www.atmarkit.co.jp/ait/articles/0606/10/news016_2.html

3. 実際にやってみよう



最後に、検査した結果を出力して、メールで送って下さい。

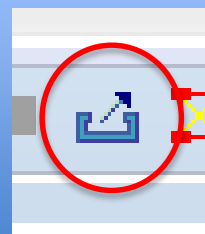
検査結果の作り方

```
dispatcher rd = req.getRequestDispatcher("view/confirm.jsp");  
d(req, resp);
```


宣言	メトリック - DragonQuest - Nested Block Depth (avg/max per method)
	合計 Mean Std. ... 最大 Res
k (avg/max per type)	0.929 1.321 3 /Dr

出力したXMLをメールで西野(Nishino@asojuku.ac.jp)まで提出

メトリクスの出力



↑をクリック