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

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？

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？

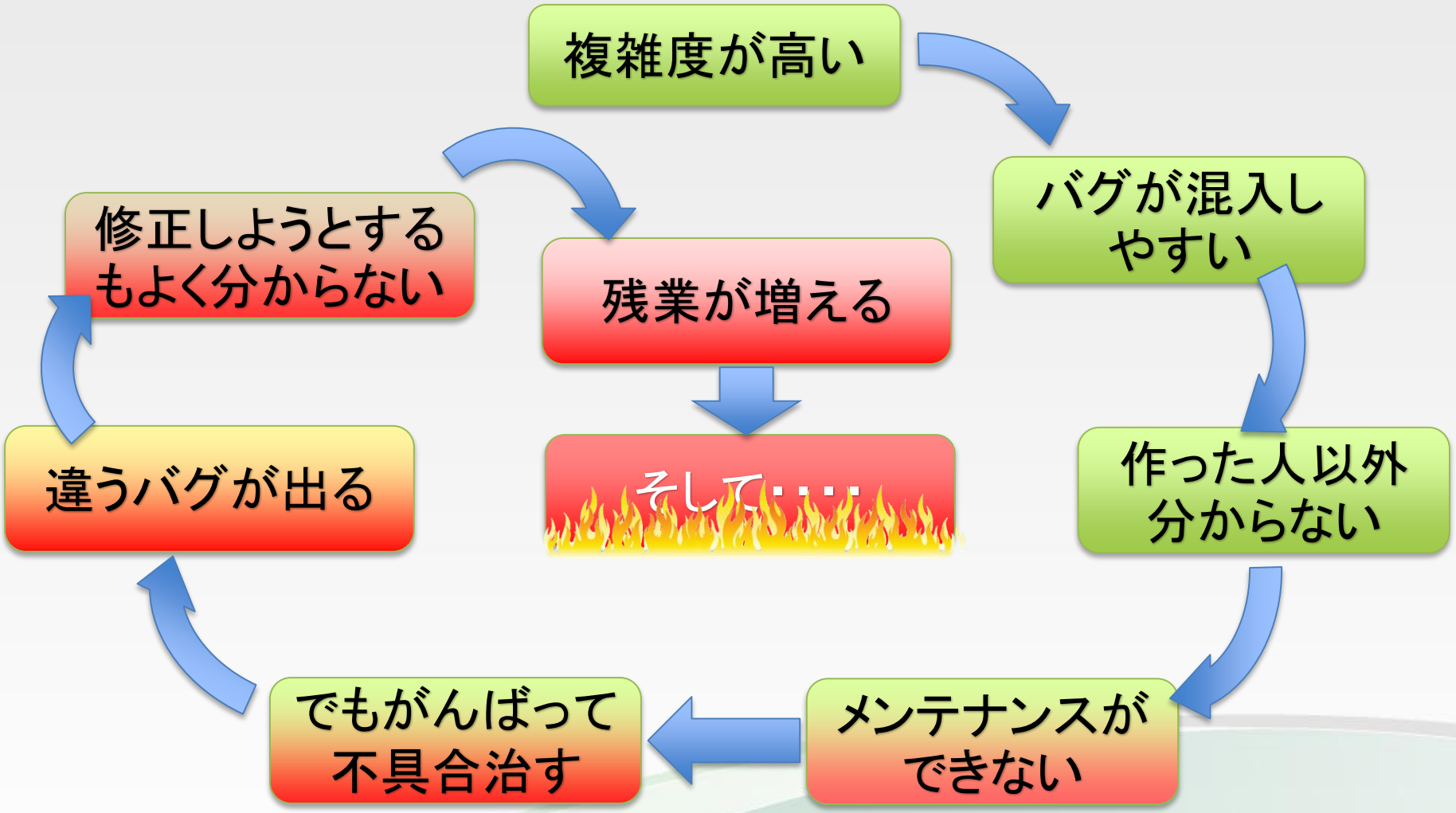
```
private int getdaze(boolean getFlg){
    if( getFlg ){
        if( kind == 0 ){
            if( megaFlg ){
                println (“ピカチュウ”);
            }else{
                println (“ピカチュウ”);
            }
        }else if( kind == 1 ){
            if( megaFlg ){
                println (“ゲッ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


複雑度が高い
バグが混入しやすい
作った人以外分からない
メンテナンスができない
でもがんばって不具合治す
違うバグが出る
修正しようとするもよく分からない
残業が増える
そして・・・

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？

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

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

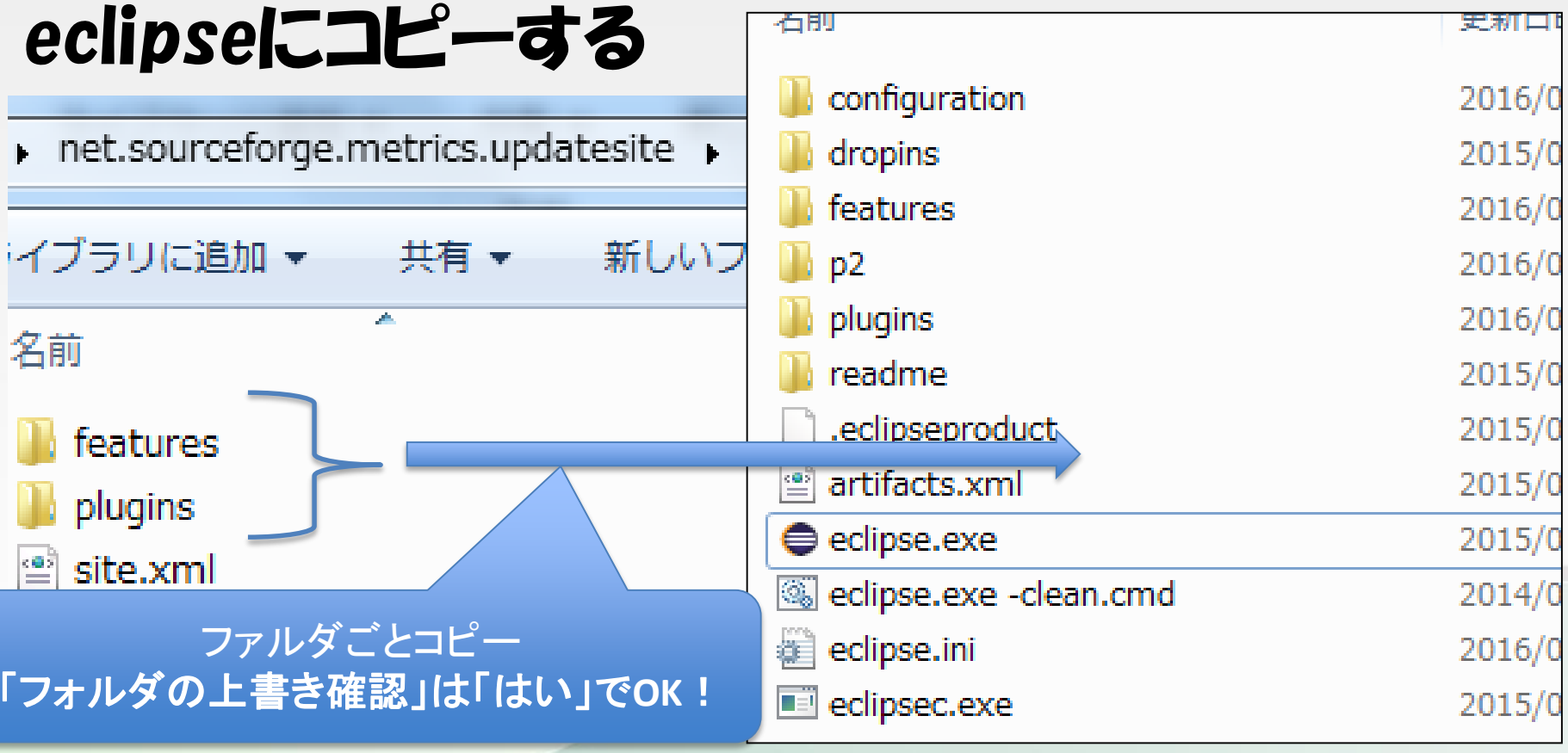
【方法2】

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

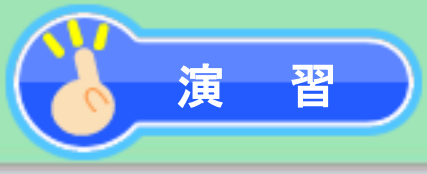


2. ツールの使い方

（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

３．実際にやってみよう



ためしに、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。

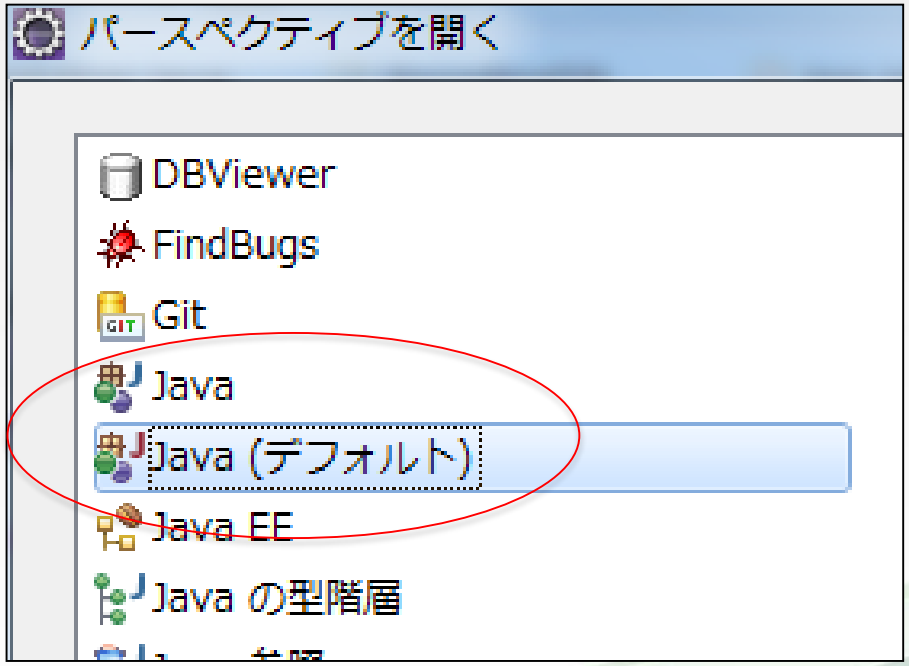
（１）eclipseを起動してPJ開発で使っているソースを開きましょう。（今日手元にない人は、今まで作った何かを使いましょう）

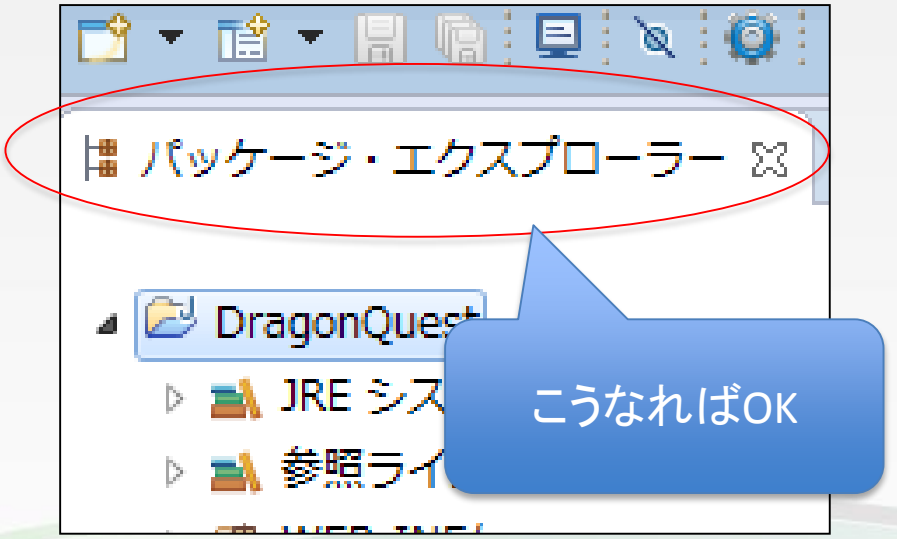
3. 実際にやってみよう

演　習

(2)パッケージエクスプローラーを開きます

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

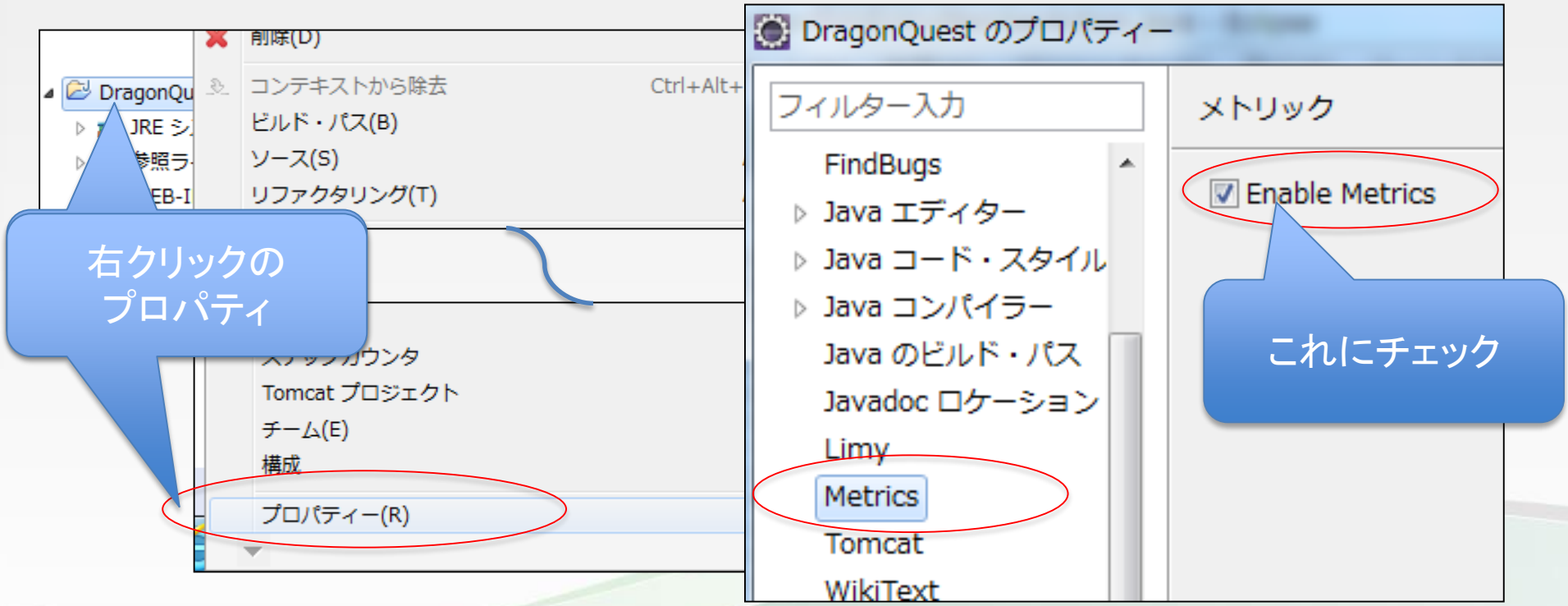




3. 実際にやってみよう

演　習

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



3. 実際にやってみよう

演　習

(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

3. 実際にやってみよう

演　習

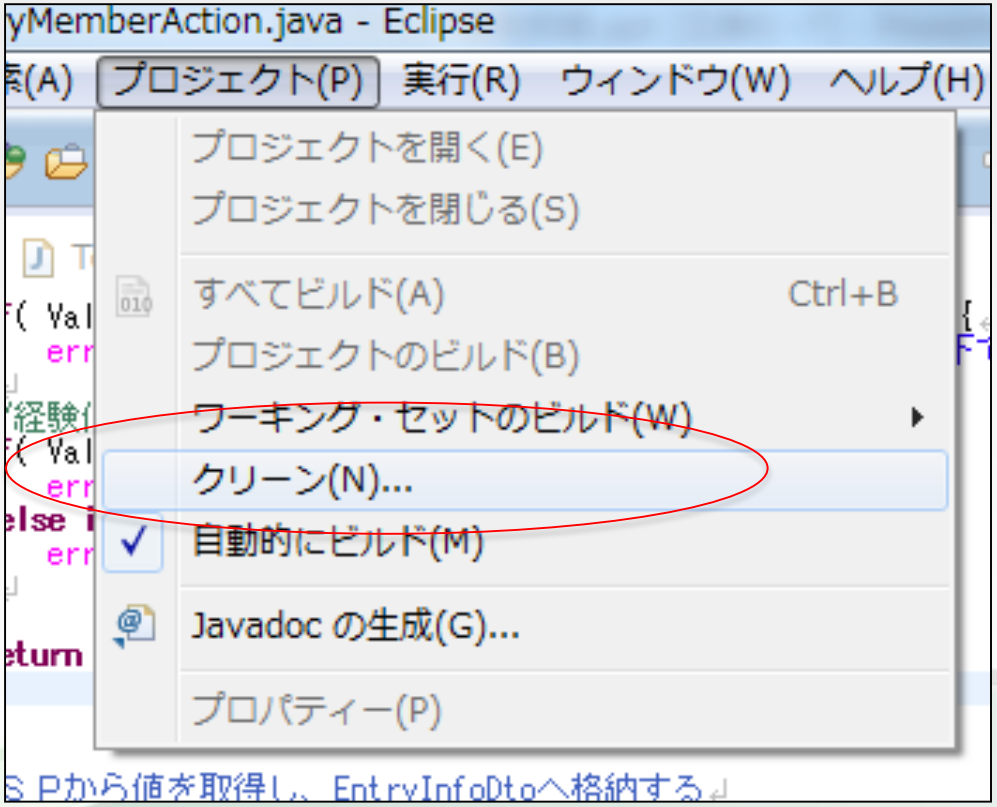
これで準備完了！
いよいよ計測しましょう！！

3. 実際にやってみよう

演　習

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、リビルドを行いましょう！

「プロジェクト」ー「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↓

問題　Javadoc　宣言　メトリック - DragonQuest

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 packageFragn		0	0	0	/DragonQuest/WEB-INF/src/j
Normalized Distance (avg/max per packa		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

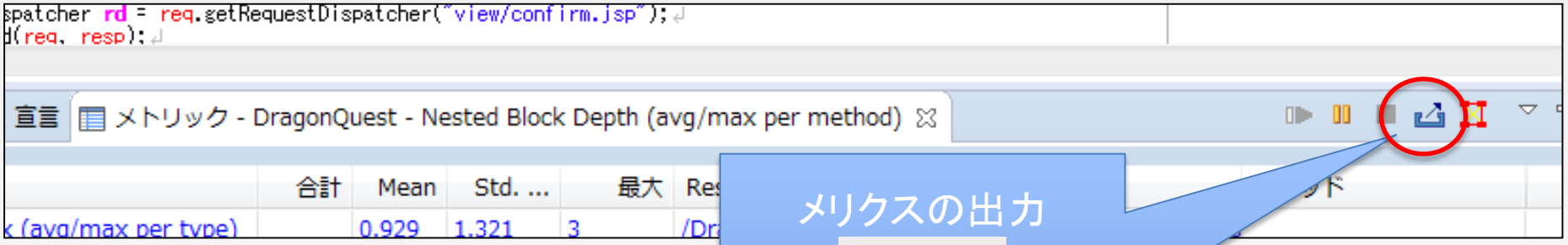
http://www.atmarkit.co.jp/ait/articles/0606/10/news016_2.html

3. 実際にやってみよう

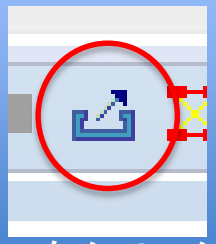
演　習

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



メリクスの出力



↑をクリック

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