万人のためのオートメーション: Eclipse プラグインによるコードの改善

便利な 5 つのプラグインを使って Eclipse 内でのコード品質分析を自動化する

Paul Duvall

2007年 1月 11日

コードをビルドする前にコードに潜む問題を発見できるとしたらどうでしょう。大変興味深いことに、JDependや CheckStyle などの Eclipse プラグインをツールとして使うと、問題がソフトウェアに現れる前にそれを見つけ出すことができます。連載「万人のためのオートメーション」では今回、オートメーションのエキスパート Paul Duvall が、開発ライフ・サイクルの初期段階で問題を防ぐための静的分析プラグインをEclipse でインストール、構成、そして使用する例を紹介します。

この連載について

私たちは開発者として、エンド・ユーザーのプロセスを自動化するために作業しますが、自分自身の開発プロセスを自動化するチャンスは見落としがちです。そこで、連載記事「万人のためのオートメーション」では、実用的なソフトウェア開発プロセスの自動化を検討し、自動化を上手に適用するタイミングと方法を教えます。

ソフトウェアを開発するときに私が掲げる主な目標のうちの 1 つは、不具合がコード・ベースに入り込むのを防ぐこと、あるいは不具合の存続期間を制限することです。つまり、不具合はできるだけ早くに見つけるように努めています。当然、より良いコードを作成してソフトウェアを効率的にテストする方法を知れば知るほど、不具合を見つける腕は上達しますが、潜在的な不具合を見つけるための安全網も欲しいものです。

この連載の 8 月の記事では、検査ツールをビルド・プロセスに組み込むと (例えば、Ant や Maven を使用)、潜在する不具合を発見する上で一貫した手法を確立できると結論づけました。この手法は一貫性を実現し、IDEの限界を超えるものですが、少々保守的でもあります。ソフトウェアをローカル側でビルドするか、継続的インテグレーション・ビルドが実行されるまで待たなければならないからです。その一方、Eclipseプラグインを使用すれば、継続的インテグレーションでビルドまたは統合する前にいくつかのコード違反を明らかにすることができます。これによって実現するのが、私がプログレッシブ・プログラミングと呼ぶ手法です。このプログレッシブ・プログラミングは、この上なく早い段階、つまりコーディング中に品質レベルをチェックすることを可能にします。

この記事では、私が考えているコード分析領域の「ビッグ・ファイブ」を取り上げます。

- コーディング標準
- コード重複
- コード・カバレッジ
- 依存関係分析
- 複雑度監視

上記の分析領域を明らかにするのが、以下の巧妙な Eclipse プラグインの数々です。

- CheckStyle: コーディング標準を目的としています。
- PMD の CPD: コード重複を発見できるようにします。
- Coverlipse: コード・カバレッジを測定します。
- JDepend: 依存関係を分析します。
- Eclipse Metrics プラグイン: 複雑度を効率的に見分けます。

Eclipse はビルド・システムではありません

Eclipse プラグインを使用することによって、検査ツールをビルド・プロセルの一部として使用できなくなることはありません。実際、Eclipseプラグインを使用して従うルールは、ビルド・プロセスに適用するルールと同じでなければなりません。

Eclipse プラグインのインストール

Eclipse プラグインのインストール方法はこの上なく単純で、必要なステップはほんのいくつかです。とは言っても、インストールに取り掛かる前にプラグインのダウンロード・サイトのURL を手元に用意しておくと役立ちます。表 1 に、この記事で使用するプラグインをリストします。

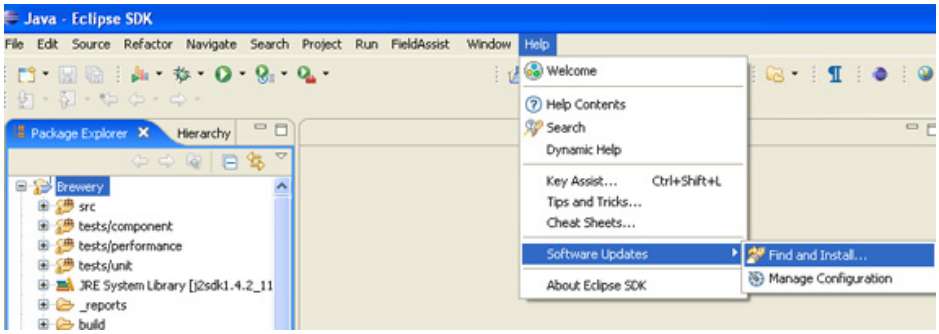
表 1. コード改善用プラグインとインストール URL のリスト

ツール	用途	Eclipse プラグインの URL
CheckStyle	コード標準の分析	http://eclipse-cs.sourceforge.net/update/
Coverlipse	コード・カバレッジのテスト	http://coverlipse.sf.net/update
CPD	コピー・アンド・ペーストの検出	http://pmd.sourceforge.net/eclipse/
JDepend	パッケージ依存関係の分析	http://andrei.gmxhome.de/eclipse/
Metrics	複雑度の監視	http://metrics.sourceforge.net/update

役に立つプラグインがどこにあるかがわかったら、あとは単純なプロセスでインストールできます。Eclipseを起動して、以下のステップに従ってください。

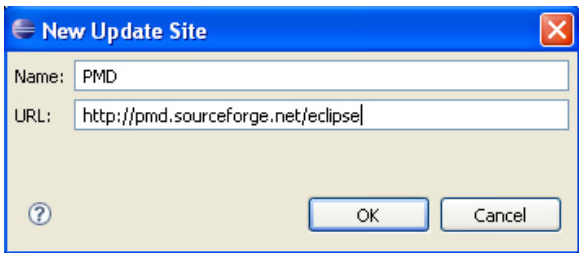
1. Help | Software Updates | Find and Install の順に選択します (図 1 を参照)。

図 1. Eclipse プラグインの検索とインストール方法



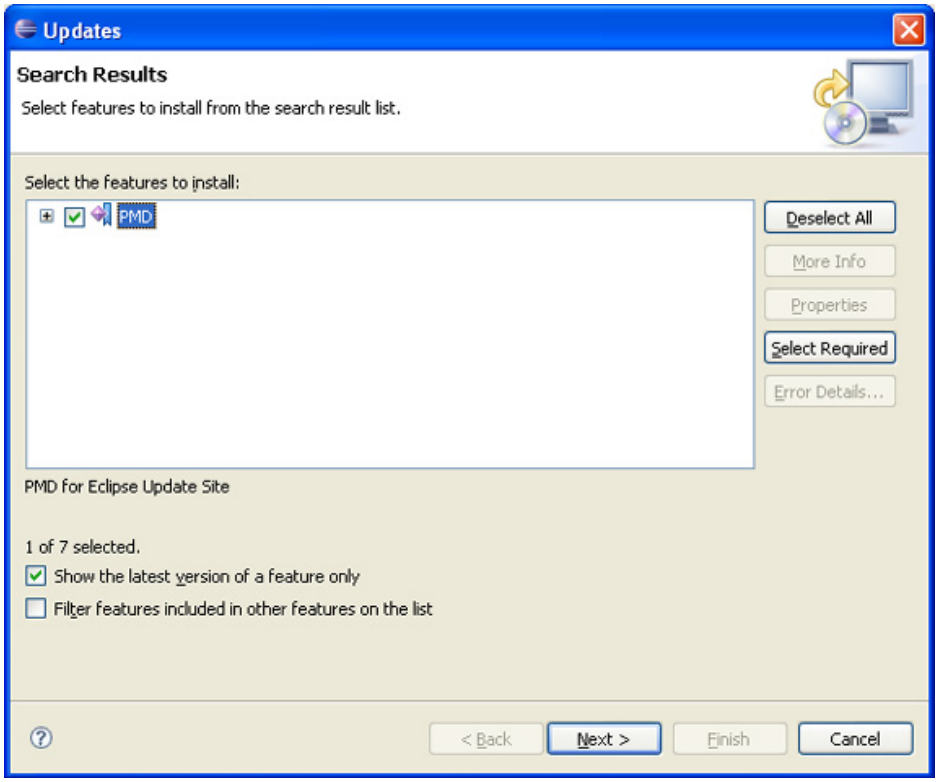
2. Search for new features to install ラジオ・ボタンを選択してから Next をクリックします。
3. New Remote Site をクリックして目的のプラグインの名前と URL を入力し (図 2 を参照)、OK をクリックします。続いて Finish をクリックすると、Eclipse Update Manager が表示されます。

図 2. 新規リモート・サイトの構成方法



4. Eclipse Update Manager には、プラグインのさまざまな特徴を表示するオプションが表示されます。私は通常、図3 に示すように最上位レベルの項目を選択します。選択した後、Finish をクリックすると Eclipse がプラグインのインストールを開始します。インストールが終わったら、Eclipseインスタンスを再起動してください。

図 3. Eclipse プラグインのインストール方法



いずれの Eclipse プラグインをインストールする場合も上記のステップに従います。変更が必要なのは、プラグインの名前と対応するダウンロードの場所だけです。

CheckStyle によるコーディング標準への違反の修正

コード・ベースの保守容易性は、ソフトウェアの全体的なコストに直接影響します。さらに、保守容易性は開発者のフラストレーションを溜める(あるいは溜めない) 一因でもあり、コードを変更しやすければしやすいほど、新しい製品機能を追加するのが簡単になります。CheckStyleのようなツールが支援するのは、保守容易性に影響するコーディング標準に対する違反を見つけることです。これらの違反にはさまざまなものがありますが、そのごく一部として、大きなクラス、長いメソッド、使用されていない変数などが挙げられます。

PMD について

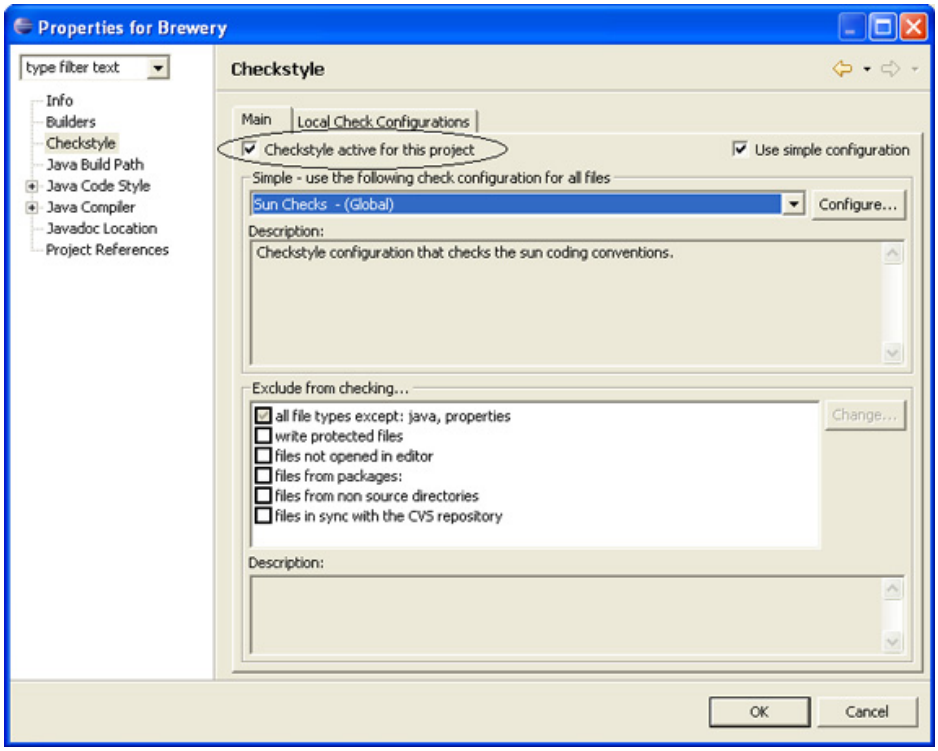
PMD と呼ばれる同じくオープン・ソースのツールも CheckStyle と同様の機能を提供します。私が選ぶのはCheckStyle の方ですが、PMD には熱心な支持者がいるので調べてみてください。多くの人たちが、PMDに信頼を置いていることがわかります。

Eclipse に CheckStyle プラグインを使う利点は、コーディング中にソース・コードのコンテキストで各種の違反を突き止められるので、チェックインの前に開発者によって違反が修正される可能性が高くなることです。CheckStyleプラグインを使用するということは、継続的にコードをレビューするようなものです。

CheckStyle プラグインをインストールして構成 (図 4 を参照) するには、以下のステップに従います。

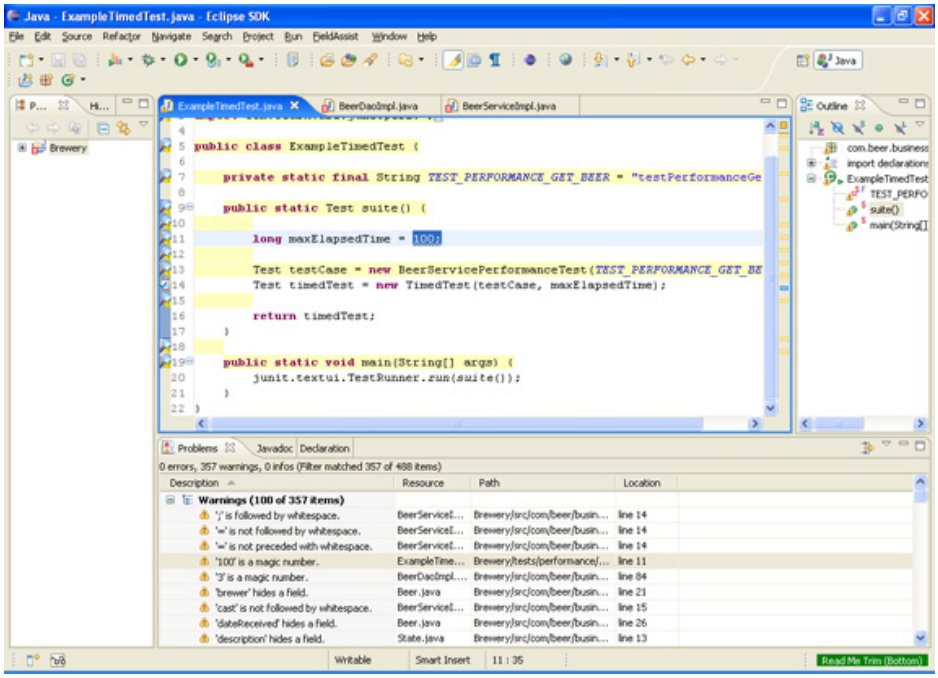
1. Project を選択し、Eclipse メニューから Properties 項目を選択します。
2. CheckStyle active for this project チェック・ボックスを選択して OK をクリックします。

図 4. Eclipse での CheckStyle プラグインの構成方法



Eclipse がワークスペースを再ビルドし、Eclipse コンソール内で検出されたコーディング違反をリストします(図 5 を参照)。

図 5. Eclipse での CheckStyle 違反のリスト



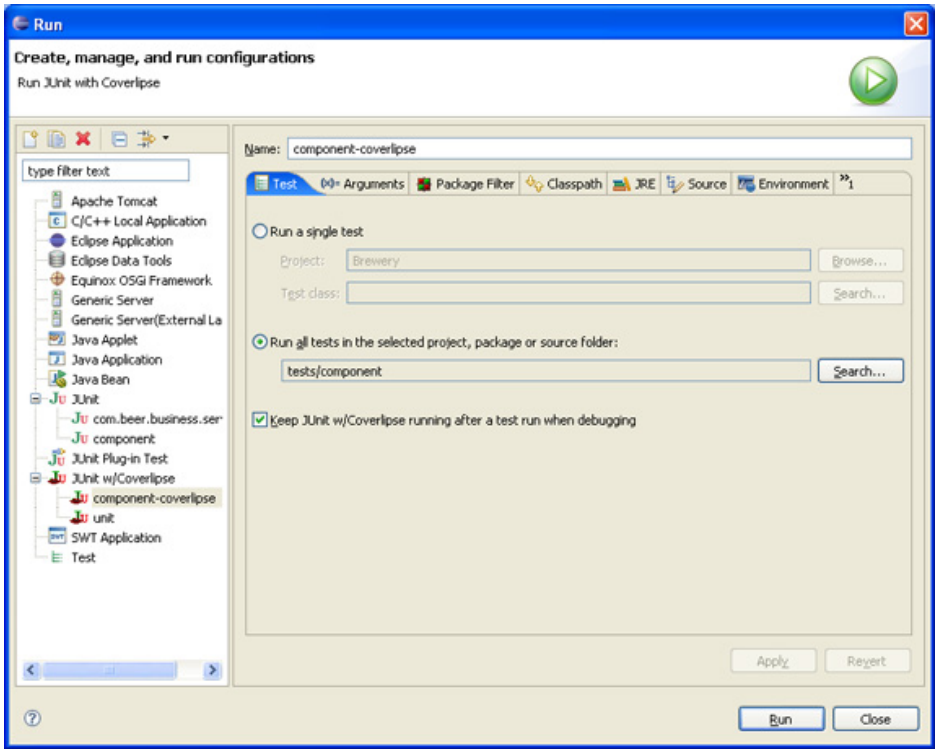
コーディング中に積極的にコードを改善するには、CheckStyle プラグインを使ってEclipse 内にコーディング標準チェックを組み込むのが非常に有効な手段となり、ソース・コードに潜在する不具合を開発サイクルの早い段階で発見するという効果があります。さらに時間の節約、フラストレーションからの解放という追加の利点もありため、プロジェクトのコストまで削減できるかもしれません。それこそ、積極的な姿勢です。

Coverlipse によるカバレッジの確認

Coverlipse は、ソース・コードの中で対応するテストがあるソース・コードのパーセンテージを評価するために使われるEclipse プラグインです。Coverlipse を使えば、コードの作成中にコード・カバレッジを評価できます。これで傾向がわかってきましたか。

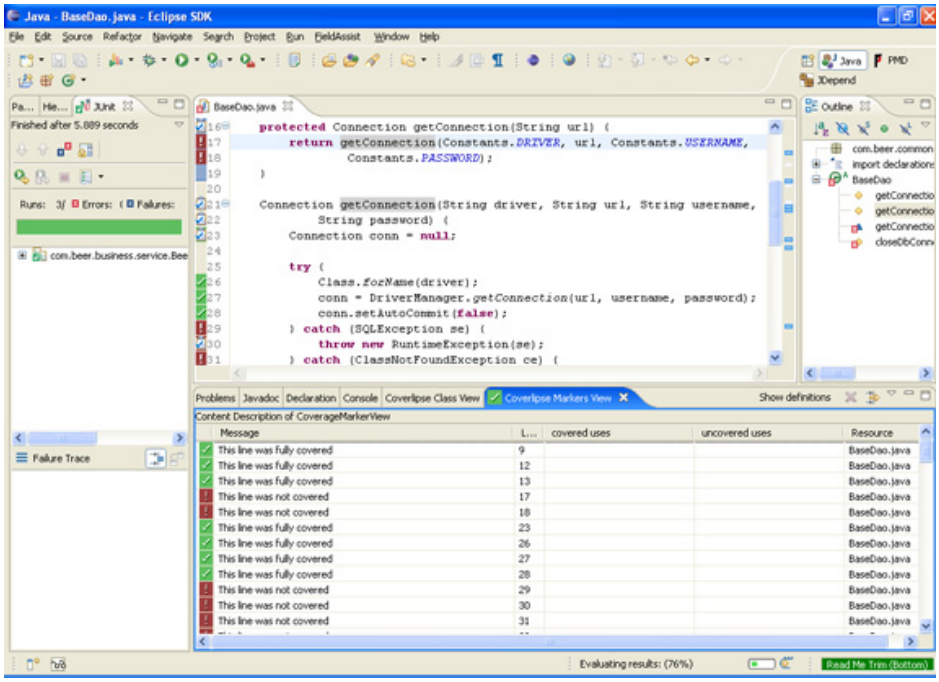
Coverlipse プラグインをインストールして、JUnit と関連付けるには、Eclipseメニュー項目の Run を選択します。Run を選択すると、例えば JUnit、SWT アプリケーション、そしてJava™ アプリケーションなどの一連の実行構成が表示されます。JUnit w/Coverlipse ノードを右クリックして、New を選択してください。図 6 に示すように、ここから JUnit テストのロケーションを指定します。

図 6. コード・カバレッジを取得する Coverlipse の構成方法



Run をクリックすると、Eclipse が Coverlipse を実行してソース・コードにマーカーを組み込みます(図 7 を参照)。これらのマーカーが、JUnit テストと関連付けられたコードの部分を示します。

図 7. Coverlipse が生成する、クラス・マーカーを組み込んだレポート



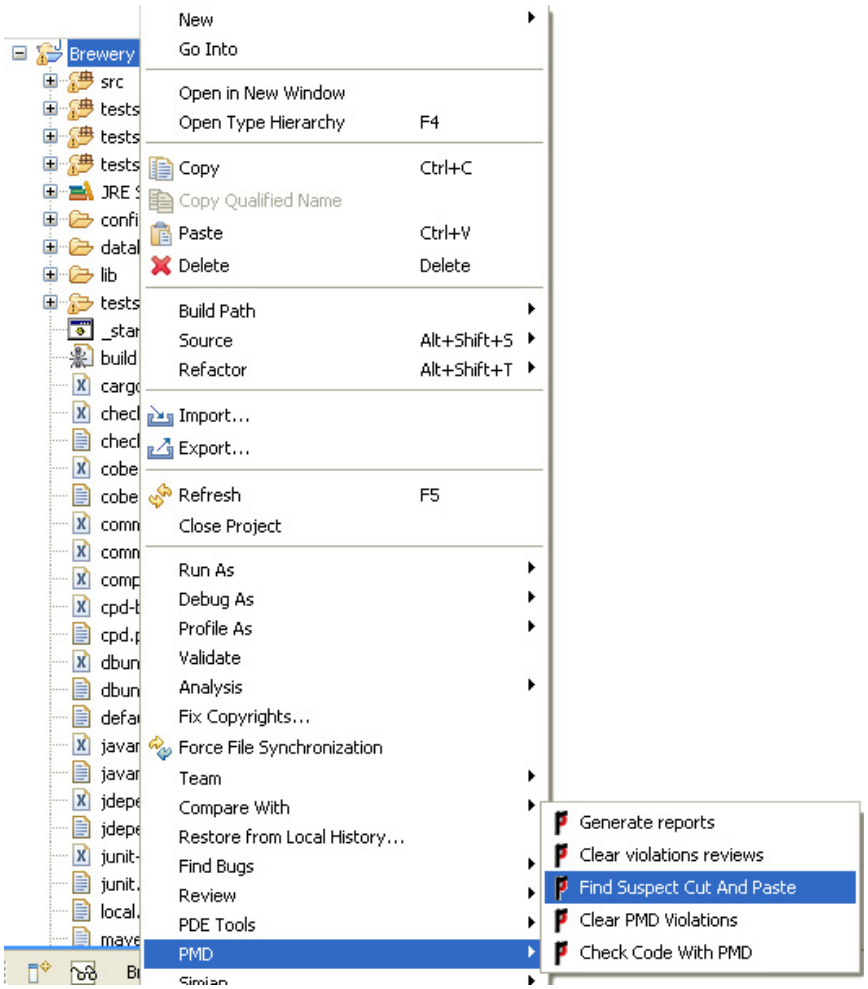
ご覧のとおり、Coverlipse という Eclipse プラグインを使うと遥かに簡単にコード・カバレッジを判断できます。このような実データ機能は、例えばコードをCM システムにチェックインする前のテストの精度を高めます。プログレッシブ・プログラミングには何と素晴らしい機能でしょう。

CPD によるコード重複の検出

Eclipse 対応の PMD プラグインは、重複したコードを検出する CPD (Copy PasteDetector) という機能を提供します。この便利なツールを Eclipse 内で使用するには、CPDユーティリティーに含まれる PDM 用 Eclipse プラグインをインストールする必要があります。

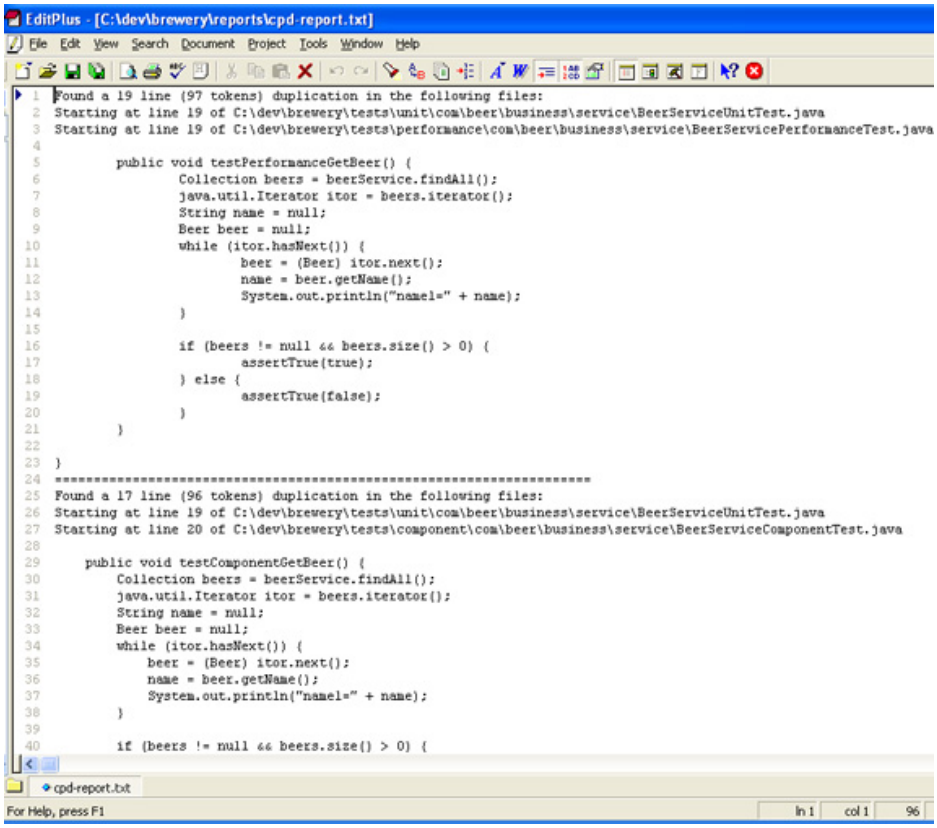
重複したコードを検出するには、Eclipse プロジェクトを右クリックし、PMD | Find Suspect Cut and Paste の順に選択します (図 8 を参照)。

図 8. CPD プラグインによるコピー・アンド・ペースト・チェックの実行方法



CPD を実行すると、Eclipse ルート・ディレクトリーに report フォルダーが作成されます。重複コードをリストしたcpd.txt という名前のファイルは、このフォルダーのなかにあります。図 9 は、cpd.txtファイルの一例です。

図 9. Eclipse プラグインから生成された CPD テキスト・ファイル



重複コードを手作業で見つけるのは困難ですが、CPD のようなプラグインを使用すれば、コーディングの最中に重複したコードを明らかにできます。

JDepend による依存関係の分析

JDepend は無料で入手できるオープン・ソースのツールで、パッケージ依存関係に対するオブジェクト指向のメトリクスを提供します。このツールが示すのは、コード・ベースの弾力性です。言い換えれば、JDependはアーキテクチャーの堅牢性 (逆に言うと、脆弱性) を効率的に測定します。

JDepend は Eclipse プラグインだけでなく、これらのメトリクスを取得するためのAnt タスク、Maven プラグイン、そして Java アプリケーションも提供します。そのそれぞれが同じ情報に対して異なる配信メカニズムを提供しますが、Eclipseプラグインの大きな違いと利点は、Eclipse プラグインはソース・コードにより近い情報を配信するということです(つまり、コーディング中の情報)。

JDepend の Eclipse プラグインを使用するには、図 10 に示すように、ソース・フォルダーを右クリックしてRun JDepend Analysis を選択します。必ず、ソース・コードがあるフォルダーを選択してください。そうしないと、このメニュー・オプションは表示されません。

図 10. JDepend Analysis を使用したコード分析方法

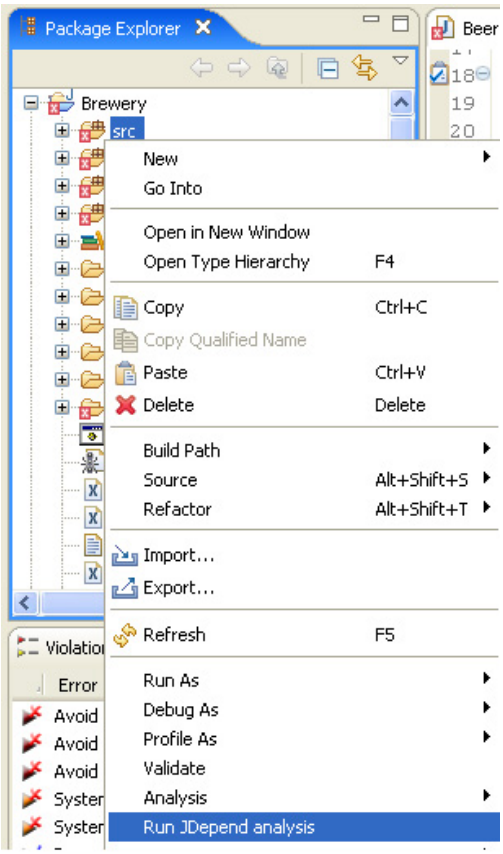
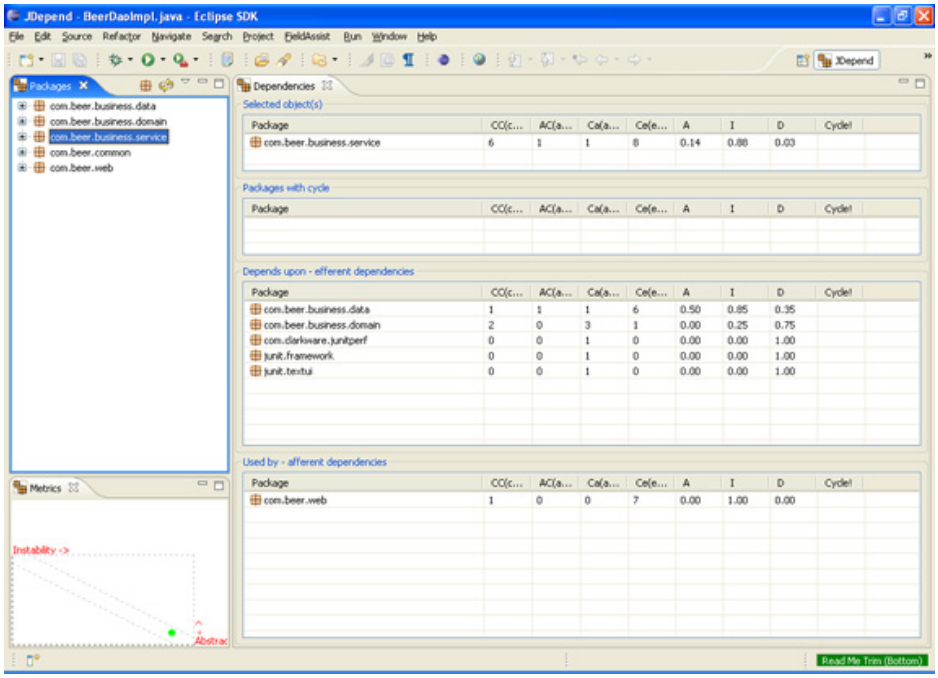


図 11 は、JDepend 分析の実行中に生成されるレポートです。左側にはパッケージが示され、右側にはパッケージごとの依存関係のメトリクスが示されます。

図 11. Eclipse 内プロジェクトのパッケージ依存関係



ご覧のように、JDepend プラグインが提供する豊富な情報により、アーキテクチャーが時間に経つにつれ保守しやすくなっているかどうかが簡単にわかるようになります。そしてこのプラグインの最大の利点は、このデータをコーディング中に見られるということです。

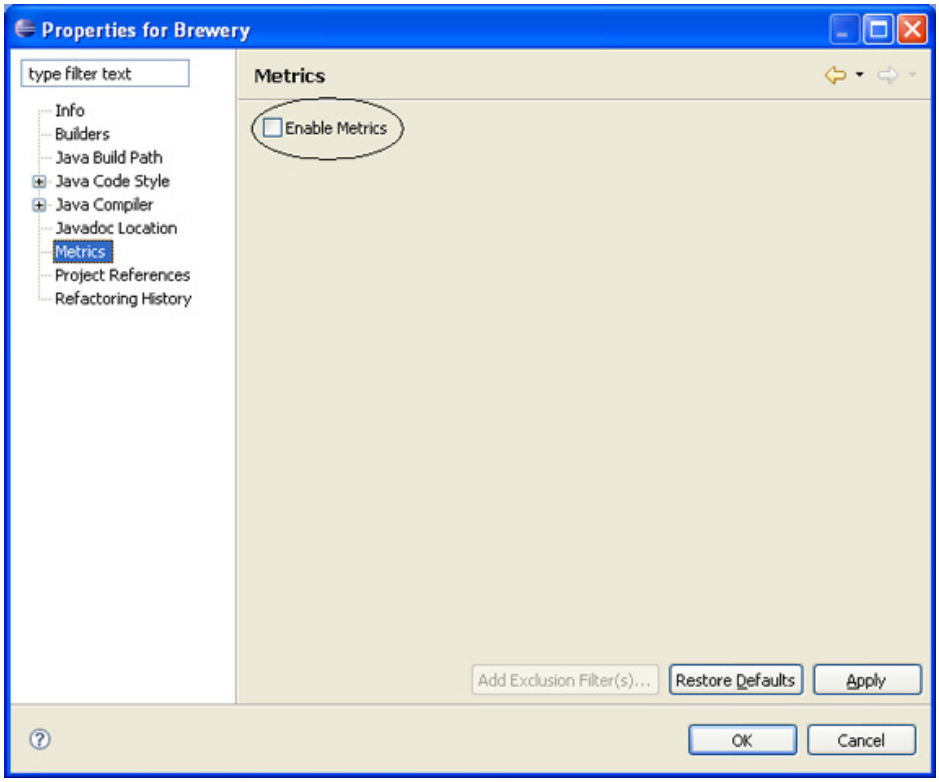
Metrics による複雑度の測定

コード分析領域の「ビッグ・ファイブ」で最後に測定するのは、複雑度です。Eclipseには、Metrics というプラグインが用意されています。このプラグインは、メソッドに含まれる固有パス数を計る循環的複雑度をはじめ、多数の有益なコード・メトリクスを提供します。

Metrics プラグインをインストールして、Eclipse を再起動してください。次に、以下のステップを行います。

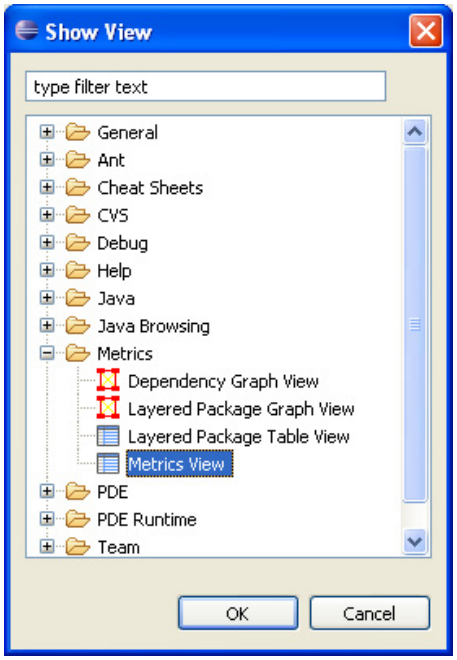
1. **プロジェクト**を右クリックし、Properties メニューを選択します。表示されるウィンドウで、Enable Metrics plugin チェック・ボックスを選択して OK をクリックします (図 12 を参照)。

図 12. プロジェクトを対象に機能する Metrics の構成方法



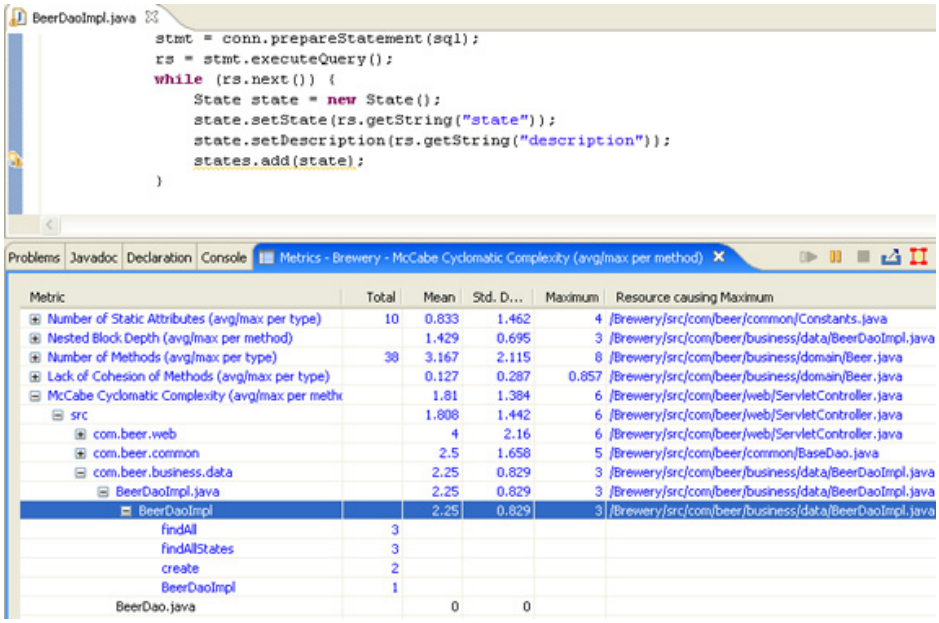
2. Eclipse から Window メニューを選択して Metrics ビューを開き、Show View | Other... の順に選択します。
3. Metrics | Metrics View の順に選択して図 13 に示すウィンドウを表示します。メトリクスを表示するには、Javaパースペクティブを使用してプロジェクトを再ビルドする必要があります。

図 13. Eclipse での Metrics ビューの表示方法



4. OK をクリックして図 14 のようなウィンドウを表示します。 この例では、個別のメソッドの循環的複雑度を表示しています。このプラグインでは非常に便利なことに、Metricsリスト内のメソッドをダブルクリックすると、そのメソッドのソース・コードがEclipse エディターに表示されます。そのため、必要に応じていとも簡単に修正を行うことができます。

図 14. メソッドの循環的複雑度の表示



前にも言ったように、Eclipse Metrics プラグインは、ソフトウェアの開発中にコードを改善するのに役立つ強力なメトリクスを他にもたくさん提供します。Metricsはまさに、プログレッシブなプラグインです。

自分の役に立つプラグインを使用する

この記事を読めば一目瞭然ですが、私がコード品質に重要だと思うのは、「ビッグ・ファイブ」の測定、すなわちコード標準、コード重複、コード・カバレッジ、依存関係分析、そして複雑度監視です。ただし肝心なのは、あなたにとって役に立つのはどれかということです。Eclipseプラグインには他にも PMD や FindBugs などさまざまなツールがあり、いずれも開発サイクルの初期段階でのコードの品質改善に役立つということを忘れないでください。目的のツールや好みの測定が何であれ、コードを改善するための行動を取って、手作業でのコード・レビュー・プロセスを一層効率的にすることが大事なのです。しばらくプラグインを使ってみれば、今までそれなしでどうやってきたのか不思議に思うはずです。

関連トピック

- 「万人のためのオートメーション」(Paul Duvall、developerWorks): 一連の連載記事を読んでください。
- 「Improving Code Quality with PMD and Eclipse」(Levent Gurses 著、Jacoozi、2005年1月): この記事では Eclipse プラグインとしてのPMD を取り上げ、PMD を使ってコード品質を改善してコード・レビュー・プロセスの期間を短縮する方法を説明しています。
- 「Cobertura でテスト対象範囲を調べる」(Elliotte Rusty Harold 著、developerWorks、2005年5月): Elliotte Rusty Haroldが、コード・カバレッジのベスト・プラクティスを用いて Cobertura を利用する方法を紹介しています。
- 「カバレッジ・レポートに騙されないために」(Andrew Glover 著、developerWorks、2006年1月): この記事では、カバレッジ・レポートのさまざまな数値が実際に何を意味するのか、そして何を意味しないのかを詳しく検討しています。
- 「Managing Your Dependencies with JDepend」(Glen Wilcox 著、OnJava、2004年1月): この記事では Glen Wilcox が、ソフトウェア・アーキテクチャーの品質をいくつかの点で詳細に検討できる無料のツール、JDependを紹介しています。
- 「ソフトウェア開発者にとってのコード品質」(Andrew Glover 著、developerWorks、2006年4月): Andrew Glover が、ソフトウェア・アーキテクチャーの長期的存続に影響するコードの品質面を継続的かつ正確に監視する方法を説明しています。
- 「Automation for the people: Continuous Inspection」(Paul Duvall 著、developerWorks、2006年8月): Paul Duvall が、CheckStyle、JavaNCSS、CPDなどの自動インスペクターで開発プロセスを向上させる方法、そしてどんな場合に使用すればいいのかを検討しています。
- 「Detecting Duplicate Code with PMD's CPD」(Tom Copeland 著、OnJava、2003年3月): Tom Copeland が、重複した Java コードを検出してくれるオープン・ソースのユーティリティー、CPD(Copy/Paste Detector) を紹介しています。
- 「Maintain organizational standards with code audits」(testearly.com): コーディング標準によって、さまざまな開発者グループがコード・ベースを同じように理解できるようになります。
- developerWorks Java technology ゾーン: Java プログラミングのあらゆる側面を網羅した記事が、豊富に用意されています。
- PMD 用 Eclipse プラグイン: PMD プラグインを使って、コード内のコピー・アンド・ペースト問題を検出できます。
- JDepend 用 Eclipse プラグイン: このプラグインで、コード・ベース内のパッケージ依存関係の分析が容易になります。
- Coverlipse 用 Eclipse プラグイン: Coverlipse は、コード・カバレッジ情報を提供する Eclipse プラグインです。
- Eclipse Metrics: 循環的複雑度などのメトリクスを提供するこのプラグインは、複雑なコードの検出に非常に役立ちます。
- CheckStyle 用 Eclipse プラグイン: プロジェクトがコーディング標準に準拠しているかどうか調べてください。