Javaソースコード品質点検ツール(EJAQUET)の開発とその効果

技術総括部
ITスペシャリスト

渡部　直和

Naokazu Watanabe
naokazu-watanabe@exa-corp.co.jp

ソースコードの品質をどのように確保するかは、システム開発において重大な問題である。一般的にレビューやインスペクションを実施するなどして品質確保に努めてきた。しかし、これらを実施してもソースコードの問題検出・品質確保は容易ではなく、効果ある実施にはスキル・ノウハウ、そして少なくない工数が必要である。そこでJavaソースコードの問題検出・品質分析を自動化するツールEJAQUET®を開発した。本稿では問題検出・品質分析ツールEJAQUETの開発および、EJAQUETの問題検出・品質分析機能の効果を解説する。

1. はじめに

ソフトウェアには機能的に要件を満たすだけでなく高い品質も求められている。そのためソフトウェア開発では、成果物に対してレビューやインスペクションなどの品質を確保するための方策を実施している。

実装フェーズの成果物であるソースコードにおいても、レビューやインスペクションなど点検を実施し品質の確保を目指している。しかし、これらの点検作業が必ずしも実効性のあるものになっていないことがある。開発の遅延等、スケジュールの問題から点検作業がしわ寄せをうけ、十分に実施できなかったり、あるいは点検を実施しても問題を見逃したりするなどの問題があり、効果のある点検作業を実施することは容易ではない。

そこで、実効性のあるソースコードの点検策として、Java®ソースコードの品質点検・分析ツール“EJAQUET ®”を開発した。EJAQUETはJavaソースコードの問題箇所の検出をスキル不要で自動化し、さらに品質の分析を可能とする。

本稿は、２章で今までの品質改善活動の課題を考察し、次に３章では品質分析・改善ツールの要件とEJAQUETがそれらをどのように達成したかを解説、さらに４章ではEJAQUETの効果を説明する。

2. ソースコードの品質問題と対策

本章では、ソースコードの品質改善のために実施してきた対策の効果と問題点について述べ、品質改善のポイントを考察する。

2.1. Javaソースコードの品質の問題

弊社では、成果物の品質向上のためにQI（Quality Inspection）[1)] を実施している。Javaソースコードを対象にしたQIも実施しており、そこで検出された問題の例を表１に示す。システムリリース時にこれらの問題を除去できていなければ、システム停止や誤動作を引き起したり、保守や維持管理に支障をきたす原因となったりする。また、最終的に除去できたとしても、テスト以降で除去するのでは、スケジュール遅延やコスト増などの、プロジェクトに対する影響ももたらす可能性がある。

ソフトウェア開発において、これらの問題を、可能な限り早い段階で検出・除去し、かつ問題発生防止の効果的な対策を早期に実施すれば、品質の向上、開発プロセスの効率化などの効果を期待できる。

表１　JavaソースコードQIで検出された問題例

問題の種類
(1)例外処理に問題
(2)コードスタイルがコーディング規約に従っていない
(3)不要・冗長・無駄なコード
(4)リテラルのハードコーディング
(5)Javadocに誤り・記述漏れ等
(6)無駄なインスタンス生成
(7)リソース(ストリーム・DB)解放漏れ
(8)アクセス制御が不適切
(9)NullPointerException発生の可能性
(10)明らかなコーディングミス
(11)命名規約違反
(12)実装漏れ
(13)文字列比較の誤り
(14)問題のあるスレッド制御
(15)メソッド引数への代入
(16)switch文中にbreak、defaultがない
(17)コメント不足・コメントの内容に問題
(18)例外発生の可能性(NullPointerException以外)

表１の項目についての補足説明

- 「(1)例外処理に問題」は、例外をキャッチしてもそれを処理しない「例外の握り潰し」や、例外のスロー・キャッチに適切な例外クラスを使っていないなど、適切に例外処理が行われていないもの
- 「(10)明らかなコーディングミス」は、x = x;（自己代入）や、if文の分岐先どちらにも同じコードが書かれているものなど、明らかにコーディングミスと考えられるもの
- 「(12)実装漏れ」は、ブロック中のステートメント全てがコメントアウトされたものや、コメント中に“TODO”や“保留”などの記述があるものや、処理が必要と思われる箇所で実装が空になっているものなど

2.2. 品質改善活動（EJAQUET以前）

品質問題に対して、弊社ではいくつかの全社的な品質改善対策を実施してきた。ここでは、Javaソースコードの品質改善対策（EJAQUET以前）の効果と問題点について述べる。

2.2.1. QI活動

QI（Quality Inspection）とは、日本IBMで考案された成果物を検査するレビュー手法である。短期間で集中的に成果物（仕様書やソースコード）をサンプリング検査する。検査はプロジェクト外の第三者であるQIエンジニアが実施する。ソースコードQIでは、ソースコードの問題検出・メトリクス計測にツールを活用し、その結果を利用し効率的に検査する。

弊社ではQIによる品質改善活動を、2006年から2010年７月までの間に84件のプロジェクトに実施し、ほぼ全てのプロジェクトで、役に立つ・満足と評価され、品質改善に成果を上げてきた。

しかし、QIには次の課題・問題がありプロジェクトによっては十分に効果を上げることができなかった。

- QIが適用できるプロジェクト数に限りがある
  QIエンジニアのリソースの制約から、QI実施数を一定以上に増やせない
- QI結果がプロジェクトで十分に生かされない
  プロジェクトのスケジュールや工数の関係から、QIを適切なタイミングに実施できなかったり、QI結果を十分に反映することができなかったりすることがある

2.2.2. QIツールの活用推奨

QIでは、効率よく短期間で点検できるよう、ツールを有効活用している。これらのツールを、ここではQIツールと呼ぶ。

次に代表的なQIツールを示す。

FindBugs®[2)]
  異常動作、良くない習慣、実行効率、マルチスレッドなどの問題を高い精度で検出
  最も使われているJavaの問題検出ツール

Checkstyle[3)]
  コードスタイルのチェックが中心
  チェック内容をカスタマイズ可能

PMD[4)]
  コード全般(バグ検出からコードスタイルや効率の悪いコード、良くない習慣)のチェック

CAP[5)]
  モジュール間の依存・参照関係、結合度を計測
  モジュールの循環参照の検出

Eclipse Metrics Plug-in[6)]
  クラス行数、メソッド行数、循環的複雑度[7)]、ifネストの深さ、メソッドのパラメータ数などメトリクス計測
  結果の表・グラフの生成

これらのQIツールで欠陥・コーディング規約違反の検出や、メトリクス計測ができ、欠陥の検出や品質傾向分析に大きな効果を上げることができる。

しかし、QIツールの活用には次の問題があった。

第一に、これらのQIツールを使用できない場合があった。これらのQIツールはEclipseのプラグインである。そのため、以下のような場合にこれらのQIツールを使用できない。

- 開発環境にEclipse以外や、Eclipseの旧バージョンが指定されている
- プラグインの導入が許可されていない

第二に、ツールを使いこなすにはノウハウが必要である。FindBugs・Checkstyle・PMDなどの問題検出ツールは、異常動作を引き起こす重大な問題から、Javaプログラミングのお作法的な軽微な問題まで大量に検出する場合があり、大量の検出項目の中から対策の必要なものを選び出さなければならない。これには高度な言語知識・ツールのノウハウが必要である。その例を次に示す。

図１は、あるプロジェクトにFindBugsを実行させた結果の一部抜粋である。このプロジェクトではFindBugsは78種類4521件の問題を検出している。

図１　FindBugsの検出（抜粋）

この検出結果から、例えばプログラムを停止させる可能性のある重大な問題、「NullPointerExceptionを発生させる可能性のある問題」を判別する場合を考える。

FindBugsが検出する問題約370種のうち、NullPointerExceptionを発生させる可能性のある問題は表２の27種である。

表２　FindBugsの検出項目でNullPointerExceptionを発生させる可能性のあるもの

1	nullポインタを参照外しをしています。
2	メソッドの例外経路において、nullポインタの参照外しをしています。
3	メソッドで引数の null チェックが行われていません。
4	Boolean 型を返すメソッドが null を返しています。
5	clone() メソッドが null を返す可能性があります。
6	Dereference of the result of readLine() without nullcheck
7	equals() メソッドが引数の null チェックを行っていません。(NP_DOES_NOT_HANDLE_NULL)
8	equals() メソッドが引数の null チェックを行っていません。(NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT)
9	null ポインタに対するアクセスを行っています。
10	この値は null であり、例外処理経路で必ず利用されています。
11	readLine() の結果をそのまま利用しています。
12	nullと分かっている値のロード。
13	メソッド呼び出しで非 null 引数に null を渡しています。
14	nullポインタの参照外しをしている可能性があります。
15	例外処理において null ポインタを利用している可能性があります。
16	null となっている可能性のあるメソッドの戻り値を利用しています。
17	到達しないコードパス上に null ポインタの可能性があります。
18	メソッド内で無条件に利用される引数へ null を渡しています。(NP_NULL_PARAM_DEREF)
19	メソッド内で無条件に利用される引数へ null を渡しています。(NP_NULL_PARAM_DEREF_ALL_TARGETS_DANGEROUS)
20	無条件に引数を利用する非仮想メソッドに null を渡しています。
21	null であってはならないパラメータが Nullable であるとアノテートされています。
22	NonNullアノテーションのついたフィールドにnullを格納しています。
23	toString メソッドが null を返す可能性があります。
24	書き込まれていないフィールドの読み出し。
25	返り値として null よりもむしろ長さ 0 の配列を返すことを検討すべきです。
26	コンストラクタで初期化前のフィールドを読んでいます。
27	このフィールドは null に設定されるだけです。

「NullPointerExceptionを発生させる可能性のある問題」を修正するためには、FindBugsの検出結果（図１）から表２の項目に一致するものを選び出さなければならない。このプロジェクトで検出された問題の中で、「NullPointerExceptionを発生させる可能性のある問題」は９種類、総計245箇所であった。

このように、ツールの活用には検出項目とその意味についての知識と、数百・数千件の検出項目から該当する項目を選び出す手間が必要である。

QIのツール活用の問題をまとめると以下のようになる。

・　開発環境が限定される

　これらのツールはいずれも統合開発環境Eclipseのプラグインであり、使用には開発環境が限定される。

・　ツールのノウハウが必要

　ツールを最適に設定し、実行後に実行結果を読み解くにはツールのノウハウが必要である。

2.2.3. Javaチェックリスト

Javaチェックリストとは、Javaソースコード欠陥予防を目的として2008年に弊社の標準として制定されたJavaソースコードの品質を確保するためのチェックリストである。Javaチェックリストは、Javaソースコードの品質向上を目的として、実装者のセルフチェック・ピアチェックや実装完了時の最終チェック、あるいはBP発注納品時の検収での使用を想定していた。

しかし、Javaチェックリストは成果を十分に上げることができなかった。Javaチェックリストが効果を上げられなかった原因は次のものであった。

・　ツールのノウハウが必要

　Javaチェックリストではチェック作業者の負荷低減のため、可能な限りツールの使用を採り入れた。そのためQIツールの問題と同様にツールのノウハウが必要となった。

・　チェックコストがかかる

　Javaチェックリストは一部に目視チェックがあり、目視チェックできるスキルを持った要員が必要であり、かつ目視チェックには工数がかかる。

　さらに、ツールを使用できない開発環境の場合は、全項目目視チェックとなり、負荷が大きくなる。

2.3. 品質改善活動の問題のまとめ

品質改善活動の問題をまとめると、次のとおりである。

・　適用できない／されないプロジェクトがある

　QIは、QIエンジニアのリソースの制約により、全てのプロジェクトに適用できない。また、ツールは開発環境に依存し、適用できないことがある。

・　問題予防や改善処置のための最適なタイミングやスケジュールで実施できない

　実施には工数・リソース・時間が必要なため、プロジェクトで必要なタイミングで自由に実施できない。

・　適用にはノウハウが必要

ツールのノウハウや言語知識を必要としている。

・ 適用のコストが必要

プロジェクトにとって実施負荷が大きいため実施が困難である。特に目視チェックはプロジェクトに大きな負荷となる。

3. EJAQUETの開発

この章では、前章で述べた品質改善の問題を解決するために必要なツールの要件と、それをEJAQUETがどのように実現しているかを述べる。

3.1. EJAQUET開発の目標

前章で述べた問題を踏まえて、Javaソースコードの品質点検・分析ツールEJAQUETの開発目標を次のように定めた。

A) バグやコーディング規約違反を検出する

ソースコードを静的解析し、異常動作を引き起こすバグやコーディング規約違反を高い精度で検出する。そして、その検出箇所や検出内容を示す。

B) 各種メトリクスや検出された問題のデータによって品質傾向を分析できる

各種メトリクス、検出したソースコードの問題データの解析と、QIで得られた知見からソースコードの品質傾向を分析する。

C) バリアフリーで実施できる

どのプロジェクトでも、プロジェクトで必要なタイミングで、必要であれば何度でも実施可能とする。そのために、開発環境非依存、スキル不要、目視チェック不要で自動チェックとする。

A)・B)は問題検出・品質分析ツールとして必要な機能である。しかし、これらの機能が十分でも、2.3.節で挙げた品質改善活動の問題を解決しなくては結果的に効果を上げることができない。この問題に対し、C)を満たすことによって解決するものとした。

3.2. EJAQUETの実装

EJAQUETは、コードのチェック・解析をするJava実装部と、ビューを生成するMicrosoft® Excel（以降エクセル）のマクロからなる（図２）。

Java実装部はメトリクス計測、ツール（FindBugs・Checkstyle）のコール、メトリクス・問題検出結果の集計・解析を行う。エクセルマクロはメトリクス・問題検出結果を見やすく整形、グラフ作成を行う。

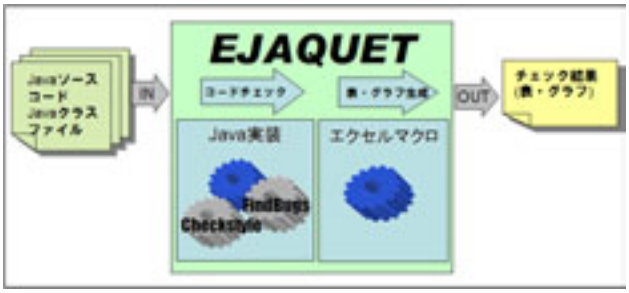

図２　EJAQUETの構造

3.2.1. バグ・コーディング規約違反の検出

EJAQUETは、バグやコーディング規約違反の検出に、QIで実績のあるツールを使用した。独自に開発するより開発を効率化でき、検出性能の優れたものが手に入るからである。ツールは、主にバグ検出するツールと主にコーディング規約違反を検出する２つのツールを使用し、Java実装部からコールされる。

(1)バグ検出ツール

バグ検出には以下の理由よりFindBugsを使うことにした。

・ バグ検出精度に優れている
・ コマンドラインから実行することができ、開発環境に依存せず実行できる
・ 検出結果をXMLで出力できるので、Javaから扱いやすい
・ フリーである
・ メジャーなツールなので、資料や情報が豊富である

(2)コーディング規約違反検出ツール

コーディング規約違反検出には以下の理由よりCheckstyleを使うことにした。

・ FindBugsがバグ検出中心であるの対して、Checkstyleはコーディング規約違反検出が主であり、補完関係にある

- コマンドラインから実行することができ、開発環境に依存せず実行できる
- 検出結果をXMLで出力できるので、Javaで扱いやすい
- 検出内容をコーディング規約に応じてカスタマイズできる
- フリーである

(3)Java実装による問題検出

EJAQUETでは、弊社コーディング規約項目をできる限り多くチェックできることが望ましいが、FindBugs・Checkstyleでチェックできない規約がいくつかある。それらの規約のうち、Javaで実装可能な以下の規約違反を検出している。

- コメントアウトしたコードを残さない
- デバッグ用コードを残さない

3.2.2. メトリクス・検出問題のデータ分析

検出した問題やメトリクスはデータ化し、品質分析に利用する。

3.2.2.1. データ化

品質に関するデータとして次のメトリクスや検出問題のデータを収集し、統合した形で保持する。

(1)メトリクス

JavaソースコードQIで品質の分析に特に有用な以下のメトリクスをEJAQUETでも計測する。

- クラス行数
- ステートメント行数
- コメント率
- 循環的複雑度

QIでのメトリクス計測ではEclipse Metrics Plug-inというツールを使用しているが、このツールはEclipse外から使用できないため、これらの計測はJavaで実装した。ただし、計測が困難で手間のかかる循環的複雑度についてはCheckstyleの検出結果を利用し、開発を効率化している。

(2)検出問題

ツール（FindBugs・Checkstyle）の検出結果より、問題の種類・内容、問題を発見したクラス、行番号、メソッド名・フィールド名など、問題についてのデータを抽出する。

(3)メトリクスと検出問題の統合

メトリクスと検出問題のデータを統合し、図３のクラス図の形で関連を作成し保持する。関連の中心は、検出問題にあたる「問題」クラスである。

ここで、ツールが検出する「問題」についての知識をユーザに要求しないように、検出する問題を種類ごとに「コーディング規約項目」に対応させ、検出された問題は「コーディング規約項目」としてユーザに示すようにしている。

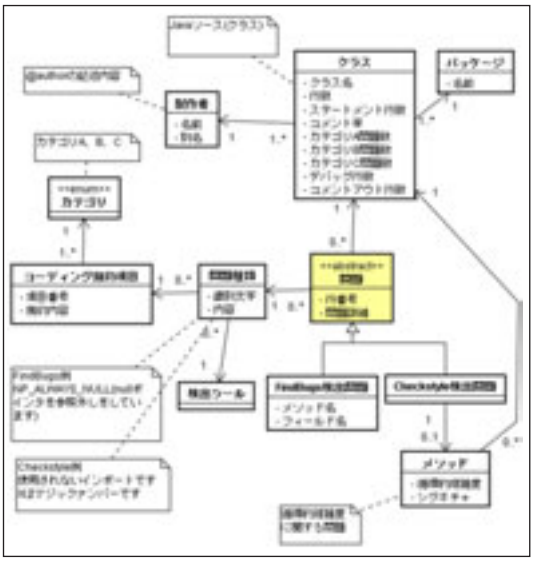

図３　問題データのクラス図

3.2.2.2. 品質分析

品質分析のため、データ化したメトリクスや検出問題から表・グラフを作成する。

ソースコードの品質情報は図３の形でデータ化している。このデータをQIで得た知見を基に、複数の視点から品質の傾向を分析できるようにしている。

QIの実績から、バグが局所的に集中していることや、同じプログラマが同じバグを何度も繰り返したり、コーディング規約違反を繰り返したりしていることが分かっている。前者は「バグは偏在する」という言葉で広く知られている。バグには局所性がある。後者は「（コーディング）スキルは属人的」であることから説明できる。

検出された問題やメトリクス計測の結果から、「バグの局所性」と「コーディングスキルの属人性」の分析ができれば、品質の分析・改善対策に有効である。

(1)局所性分析

EJAQUETは次に示したパッケージ別、クラス別の分析データを作成する。

1. クラス数（パッケージ別）
2. 合計行数・平均行数（パッケージ別）
3. ステートメント行数（クラス別）
4. パッケージごとのカテゴリ別問題検出数・問題種類別検出数
5. クラスごとのカテゴリ別問題検出数・問題種類別検出数
6. パッケージごとの問題カテゴリ別検出密度
7. コメント率
8. 循環的複雑度
9. 検出した問題の密度

パッケージ別、クラス別の分析により、問題の集中箇所・種類が分かり、効果的な対策をとることができる。

例えば、上記の1,2から特定のパッケージが巨大になっていないかどうか、さらにパッケージ構成、ひいては設計方針が適切かどうか分析できる。また、特定のパッケージにバグが多いか少ないか、あるいは複雑になり過ぎていないか、などの評価が可能である。

(2)属人性分析

検出された問題やメトリクス計測からプログラマごとのデータを生成し、プログラマごとに以下のメトリクスやグラフを計測・生成する。

プログラマの判別はJavadocの@author記述を利用する。

1. 作成クラス数
2. 合計行数・平均行数
3. プログラマごとのカテゴリ別問題検出数・問題種類別検出数
4. プログラマごとの問題カテゴリ別検出密度
5. コメント率
6. 循環的複雑度
7. 検出した問題の密度

これらを用いて、プログラマ別に品質傾向やスキルレベルを分析することができ、効果的な対策をとることができる。

例えば、これらデータから、プログラマの習熟度も知ることができる。「プログラマごとの問題種類別の検出数」のデータから、「文字列をequalsメソッドを使わず、‘==’を使って比較する」という規約の違反（表１の「(13)文字列比較の誤り」に該当）が多く検出されるプログラマが存在した場合、このプログラマは初心者であると推測される。この問題は初心者が引き起こすことがあるが、ある程度以上習熟したプログラマはこのミスを犯さない傾向がある。このミスを犯すプログラマはJava言語仕様について習熟が不十分である可能性があるので、Java言語仕様についての指導などの対策が考えられる。

3.3. バリアフリーで実施

どんなに優秀なツールであってもユーザに受け入れられ、使用されなければ意味がない。そこでユーザにとってバリアフリーで実施しやすいものである必要がある。そのためより具体的・詳細に以下の目標を設定した。

1. 自動化・負荷最小
2. 開発環境フリー
3. 簡単・シンプル・分かりやすさ

3.3.1. 自動化・負荷最小

ユーザの作業負荷・工数を最小にするため、ノウハウ不要・目視チェック不要とした。これによりプロジェクトでいつでも必要なタイミングで実施可能となる。

(1)ノウハウ不要

EJAQUETでは、検出内容を対応するJavaコーディング規約で示すことにより、ツールのノウハウがなくても検出内容の理解ができるようにしている。

例えば、2.2.2.節で示したNullPointerExceptionの例では、検出した問題はFindBugsの27種の文言ではなく、いずれも該当するJavaコーディング規約の「nullの可能性のある変数にはnullチェックを実施する」という項目にまとめて表され、ツールの知識は必要ない。

(2)目視不要

弊社Javaコーディング規約は71項目ある。EJAQUET

はそのうち65項目と、コーディング規約以外に22項目、合計87項目の問題を検出している。弊社Javaコーディング規約の90%以上を網羅しており、目視チェックを除いても十分に問題検出・品質分析可能である。ユーザはEJAQUETを実行すれば弊社Javaコーディング規約の90%以上の項目について点検結果が得られる。

3.3.2. 開発環境フリー

EJAQUETではFindBugs・CheckstyleをEclipseのプラグインではなく、コマンドラインから実行する形式で利用して、開発環境には非依存とした。

EJAQUETに必要な実行環境は次のとおりであり、開発環境には依存していない。

- Microsoft Windows®
- Microsoft Exel 2003/2007
- Java5以上

3.3.3. 簡単・シンプル・分かりやすさ

EJAQUETは、以下のようにして構成・使い方をシンプルにし、使いやすいものにするよう考慮している。

(1)ファイル構成

EJAQUETのファイル構成を図６に示す。最上位フォルダには３つのファイルと１つのフォルダがある。

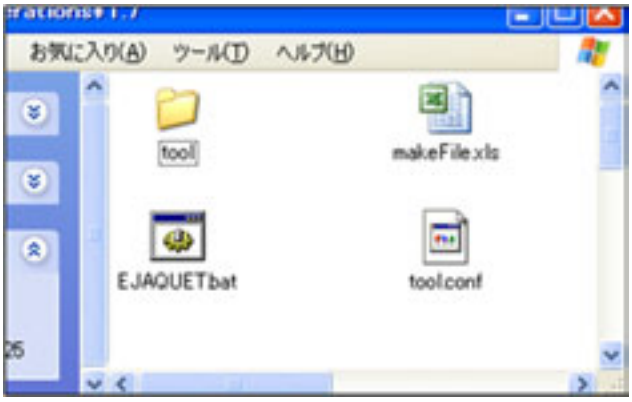

図４　ファイル構成

ユーザが実行に直接操作するファイルのみを最上位フォルダに置いた。これらのファイルのうち、ユーザが直接操作するのは最小で２つ（設定内容によって３つ）である。

ツール本体やユーザが直接操作しない設定ファイル類はtoolフォルダ内に置き、ユーザから隠蔽している。

(2)設定

必須項目は必要最小限の３項目とした。EJAQUETの設定は設定ファイルtool.confに記述する。

- 必須項目　３項目
  ソースファイルのフォルダ、クラスファイルのフォルダ、出力先フォルダ
- オプション項目　12項目
  文字コード、ツールの実行メモリサイズ、テストクラスなど結果から除外するクラスの指定など

(3)実行

最小の操作で実行できるように、アイコンのダブルクリックのみで実行するようにした。EJAQUETは図４のバッチファイル（EJAQUET.bat）のダブルクリックで実行する。ただし、設定内容によりエクセルマクロ（makeFile.xls）の明示的な実行が必要となる場合もある。

(4)実行結果

実行結果を表・グラフで視覚化したエクセルファイルで出力する。ツール・Java実装で検出した問題を集計・分

クラス数	全体行数	製作者数	平均コメント率	ツール検出数合計
694	336518	74	56.0%	13775

カテゴリA	カテゴリB	カテゴリC	巨大クラス	複雑なメソッド	テスト不可能なメソッド	コメント不足クラス
933	12702	6	30	68	6	16

項目	コーディング規約内容	違反件数
A03	nullの可能性のある変数にはnullチェックを実施する	1
A07	TODO項目を残さない	894
A09	定数でないフィールドはpublicにしない	1
A11	文字列の比較にはString#equals()を使用する	26
A15	SerializableクラスのフィールドはSerializableにする	8
A23	メソッドのパラメータへの代入は行わない	2
A94	コーディング規約外(無限ループあるいはループの危険な制御)	1
B01	1メソッドの行数は150以下とする	23
B02	1ファイルの行数は2000以下とする	30
B03	循環的複雑度は20以下とする	68
B04	1行の文字数は100以下とする	3885
B06	Javadocを記述する	946
B09	使われているクラスのみimportする	47
B10	定数リテラルを直接コーディングしない	323
B11	コメントアウトしたコードを残さない	828
B12	デバッグ用コードを残さない	1
B14	発生しない例外をthrowsに記述しない	1
B16	try節をネストしない	4
B17	同じコードは記述しない	270
B19	命名規則を守る	431
B21	不要なコードを記述しない	3731
B31	'++'または'--'とオペランドの間には半角スペースを入れない	1
B32	制御文(if・else・for・do・while・switch)の前後には半角スペースを	1553
B34	中身のない{}ブロックを書かない	3
B35	catch、elseの前の右括弧(})は同一行に書く	75
B37	過度のboolean変数の比較を行わない	32
B38	クラス内の宣言の順序は、フィールド→コンストラクタ→メソッドの	269
B9A	コーディング規約外(コードが整形されている)	12
B9B	コーディング規約外(メソッド・クラスが複雑になっていない)	169
C05	StringのtoStringメソッドは呼ばない	4
C07	外部クラスのstaticなメンバーにのみアクセスする内部クラスはs	2

図５　コーディング規約での検出結果（全体概要）

析し、その結果をエクセルマクロで表・グラフに成形する。

前述のとおり、検出した問題はコーディング規約の内容で示されツールの知識は必要ない。出力例を図５に示す。

図中のカテゴリは弊社Javaコーディング規約のカテゴリA～Cであり、以下の分類となっている。

- カテゴリA ：弊社Javaコーディング規約のカテゴリA(停止や異常動作につながる欠陥)
- カテゴリB ：弊社Javaコーディング規約のカテゴリB(テスト・保守・拡張を難しくする欠陥)
- カテゴリC ：弊社Javaコーディング規約のカテゴリC(パフォーマンス低下やリソースの無駄遣いにつながる欠陥)

3.4. EJAQUETの出力

表３にEJAQUETの出力ファイルのシート構成を示す。出力は24種のシートを持つエクセルファイルである。

局所性データはシート2および7～10、属人性データは11～23、その他のシートは全体データである。

4. 分析例

この章では、EJAQUETを実際のプロジェクトへ適用した分析事例を示す。

4.1. 局所性の分析

4.1.1. クラス

図６は表３に示した「2. クラスごとのメトリクス」シートの抜粋である。実プロジェクトのデータであるため、パッケージ名・クラス名をマスクしている。

２行目のクラスは、カテゴリA（停止・誤動作に関する規約）違反が他のクラスより格段に多い266件検出されており、このクラスを優先的に改善すべきことが分かる。

4.1.2. パッケージ

図７はパッケージ別カテゴリA問題密度のグラフである。カテゴリAは異常動作を引き起こす可能性のある問題である。

表３　出力ファイルのシート構成

シート名	内　容
1.全体概要	EJAQUET実行結果の全体概要 全体のクラス数, 行数, 検出問題数など プロジェクト全般の状態についてのデータ
2.クラスごとのメトリクス	クラスごとのメトリクス・検出した問題を行数降順の全クラスの表. 製作者, カテゴリ別検出問題数, 行数, コメント率, コメントアウト行数, デバッグ行数 クラスごとの品質のチェックに使用する
3.行数_vs_NCSS	行数の大きいクラスから順に行数とNCSS(コメント行・空行を除いたステートメント行数)をプロットしたグラフ 大きすぎるクラス, クラス間のバランスのチェックに使用する
4.コメント率	行数の大きいクラスからコメント率をプロットしたグラフ コメント量が適正か, バランスはないか, コピーコードがないかをチェックする
5.循環的複雑度	循環的複雑度10超のメソッドを降順にリスト 複雑なメソッドの個別チェックに使用する
6.循環的複雑度グラフ	循環的複雑度が10超のメソッド数のグラフ プログラム全体での複雑なメソッドの量のチェックに使用する。
7.パッケージ概要	パッケージごとに集計したクラス数, 行数, カテゴリ別問題検出数, コメント率, 複雑なメソッド数 パッケージごとの品質や, 問題の傾向のチェックに使用する
8.パッケージ詳細	パッケージごとの詳細な問題種類別の検出数 パッケージ別の品質傾向や, 問題がパッケージ固有か全体かのチェックに使用する
9-10.パッケージ別密度(カテゴリA~B)	パッケージ別の検出されたカテゴリ(A-B)の問題の密度(100行あたりの検出数) パッケージ別の問題密度のチェックに使用する。
11.製作者別概要	製作者別の検出した問題の概要 製作者ごとに集計したクラス数, 行数, カテゴリ別問題検出数, コメント率, 複雑なメソッド数の表 製作者ごとの品質や, 問題の傾向のチェックに使用する
12.製作者別詳細	製作者ごとの詳細な問題種類別の検出数 製作者別の品質傾向や, 問題が製作者固有か全体かのチェックに使用する
13.16.19.製作者別検出数(カテゴリA~C)	製作者別のるカテゴリ(A~C)の問題検出数 検出数の多い製作者をチェックするのに使用する
14.17.製作者別検出密度(カテゴリA~B)	製作者別の各カテゴリ(A~B)の検出密度数 問題密度の高い製作者をチェックするのに使用する
15.18.20.製作者別詳細(カテゴリA~C)	製作者別のるカテゴリ(A~C)の問題の詳細分類 製作者別の品質傾向, 問題が製作者固有か全体かのチェックに使用する
21.クラス作成割合	製作者ごとのクラス作成割合のグラフ
22.製作者別クラス平均行数	製作者別の1クラスあたりの行数 大きなクラス(複雑なコード・非構造的)を作っている製作者のチェックに使用する
23.製作者別コメント率	コメントを記述している・記述していない製作者のチェックに使用する
24.全検出問題	検出されたすべての問題の一覧 検出したすべての問題を, コーディング規約単位に表示したもの. 問題の内容, パッケージ, クラス, 検出行, 検出メソッド・フィールド, 製作者などを表示 個別の検出問題の確認などに使用する

枠で括ったパッケージが、飛び抜けて問題の密度の高くなっており、このパッケージを重点的に改善すべきことが分かる。

4.2. 属人性の分析

図８はあるプロジェクトの「12. 製作者別詳細」シートの一部を抜粋したものである。横軸方向が製作者（プログラマ）、縦軸方向が検出した問題の種類であり、検出された問題数を表にしている。

製作者「I12」が「文字列の比較方法の誤り」を24件検出され、他の製作者と比べて飛び抜けて多くなっている。この問題は、文字列の比較をequalsメソッドではなく、

パッケージ	クラス	製作者(別名)	A	B	C	行数	NCSS	CR	C/O	DBG
jp.co.xxx.app.zzz.subbl.sa0000	Class1	T8	8	40	0	9761	4117	47.4	0	0
jp.co.xxx.app.zzz.subbl.sa0609	Class2	R6	266	683	0	6951	2178	63.7	330	0
jp.co.xxx.app.zzz.subbl.sa0000	Class3	W15	21	67	0	5972	2307	52.6	16	0
jp.co.xxx.app.zzz.subbl.cc00034	Class4	J43	0	63	0	5303	1719	58.6	0	0
jp.co.xxx.app.zzz.subbl.sa0000	Class5	T8	3	11	0	4486	1044	73.6	3	0
jp.co.xxx.app.zzz.bl.cc.stcc0073	Class6	X18	9	163	0	4292	2051	42.1	0	0
jp.co.xxx.app.zzz.bl.cc.stcc0069	Class7	A2	2	205	0	3987	1861	39.4	1	0
jp.co.xxx.app.zzz.bl.cc.stcc0035	Class8	D7	0	120	0	3840	1334	55.7	0	0
jp.co.xxx.app.zzz.subbl.sa0163	Class9	L50	0	67	0	3814	1586	50.1	15	0
jp.co.xxx.app.zzz.subbl.sa0154	Class10	Z52	24	131	0	3718	1212	60.8	11	0
jp.co.xxx.app.zzz.subbl.cc00034	Class11	V39	0	74	0	3299	1210	53.8	0	0
jp.co.xxx.app.zzz.subbl.cc00034	Class12	V40	0	44	0	3227	1412	42.5	0	0
jp.co.xxx.app.zzz.subbl.sa0000	Class13	T8	74	415	0	3211	1902	36.0	12	0
jp.co.xxx.app.zzz.subbl.cc00034	Class14	H42	1	9	0	3006	1280	46.6	0	0
jp.co.xxx.app.zzz.bl.cc.stcc0076	Class15	E19	0	181	0	3002	1320	41.3	5	0

図６　クラスごとのメトリクス

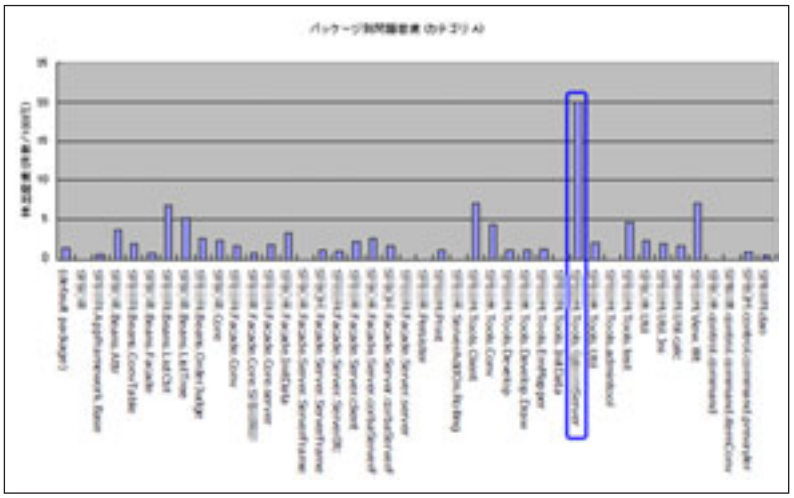

図７　パッケージごとのカテゴリA問題密度

項目	詳細	G4	K5	R6	D7	T8	J9	B10	I11	I12	I13	W
A03	null チェックを行っていない	0	0	0	0	0	0	0	0	0	0	
A03	nullを返すメソッド	0	0	0	0	0	0	1	0	0	0	
A03	アノテーション	0	0	0	0	0	0	0	0	0	0	
A04	ストリームのクローズに失敗する可能性	0	0	0	0	0	0	0	0	0	0	
A05	DBリソースの解放もれの可能性	0	0	0	0	0	0	0	0	0	0	
A06	SQLインジェクションの可能性	0	0	0	0	0	0	0	0	0	0	
A07	TODO項目の残留	12	8	270	0	98	0	0	0	54	16	
A08	System.exitの使用	0	0	0	0	0	0	0	0	0	0	
A09	不適切なアクセス修飾子	0	0	0	0	0	0	0	0	0	0	
A10	異なる型をequals比較	0	0	0	0	0	0	0	0	0	0	
A10	equalsメソッドの引数が必ずnull	0	0	0	0	0	0	0	0	0	0	
A11	文字列比較あるいはオブジェクトの比較方法の誤り	1	0	0	0	0	0	0	0	0	0	
A11	文字列の比較方法の誤り	1	0	0	0	0	0	0	0	24	0	
A12	浮動小数点数の比較方法の誤り	0	0	0	0	0	0	0	0	0	0	
A13	除算の演算を先う可能性	0	0	0	0	0	0	0	0	0	0	
B01	大きすぎるCC・メソッド行数	0	0	6	0	27	0	0	0	3	0	
B02	大きすぎるクラス	2	1	1	1	5	0	0	0	1	0	
B04	行文字数オーバー	76	32	134	126	379	0	0	0	454	17	
B05	Javadocの問題(タグの問題)	2	0	1	0	5	0	0	0	0	2	
B06	Javadocの問題(使用されないタグ)	0	0	0	0	0	0	0	0	0	2	
B06	Javadocの問題(Javadocがない)	0	0	0	0	0	0	0	0	39	0	
B06	Javadocの問題(記述漏れ)	7	7	17	0	46	0	0	0	20	35	
B07	紛らわしい名前	0	0	0	0	0	0	0	0	0	0	
B08	制御ブロックに"{}"不使用	0	0	0	0	0	0	0	0	0	0	
B09	使用されないインポート	1	0	3	0	0	0	0	0	6	0	
B09	"*"を使ったインポート	0	0	0	0	0	0	0	0	0	0	
B10	static finalにすべき変数	0	0	0	0	0	0	0	0	0	0	

図８　製作者ごとの問題種類別検出数

“==”（あるいは“!=”）で比較しているため正しい結果が得られないという問題である。これはJava初心者が犯しやすい初歩的な誤りであり、「I12」はJava初心者である可能性があることが分かる。

また、製作者「T8」の「大き過ぎるCC・メソッド行数」件数は27件と、他の製作者の件数と比べて飛び抜けて多数になっている。「T8」にはメソッド分割・構造化を指示すべきことが分かる。

4.3. その他（コメント率）

図９は３つのプロジェクトの「4. コメント率」シートである。「4. コメント率」はクラスごとのコメント率を、クラス行数の大きい順にグラフ左からプロットしたグラフで、クラスごとのコメント率の分布を示したものである。このコメント率のグラフから、次のようなソースコードの状態を読み取ることができる。

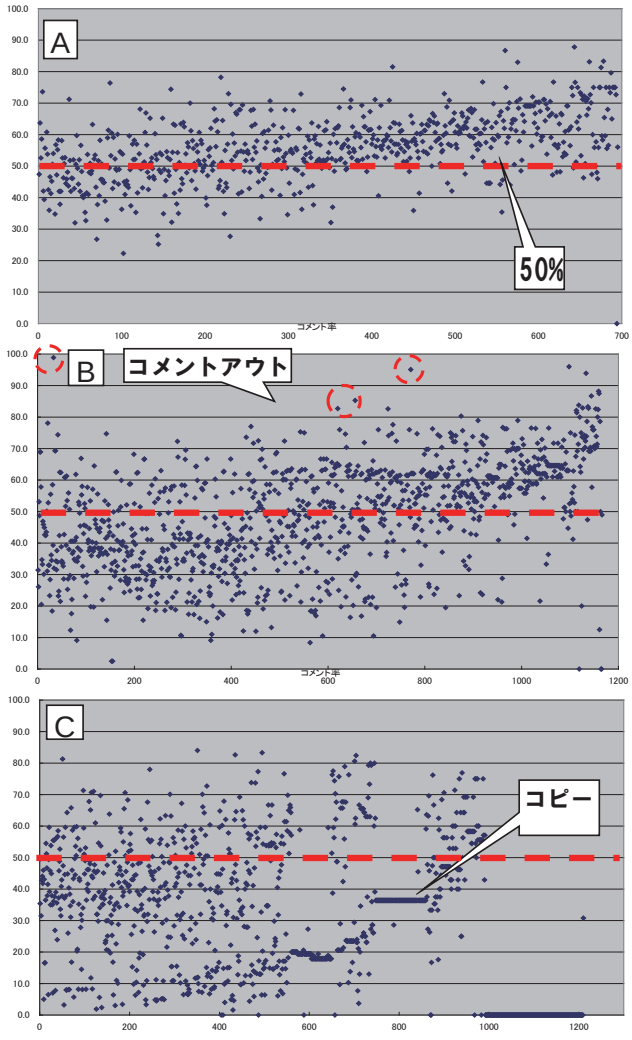

図９　コメント率グラフ（３プロジェクト）

- コメントは十分に記述されているか
  よいコメント記述の場合はコメント率が50%前後に密集し、やや右上がり（図９ A）になる。コメント率が50%よりずっと低いクラスはコメントが不足している。
- コメントアウトが大量に残っていないか
  高過ぎるコメント率はコメントアウトが多数存在している可能性が高い（図９ B）。
- コードが管理されてるか
  グラフが密集せず、ばらついていれば、コメント記述が統制されていない（図９ B・C）。また、統制されていないのはコメントだけでなく、コーディング内容全般についても統制されていない可能性が高い。
- コピーされたクラスがないか
  グラフに、フラットに横並びになっている箇所（図９ C）があれば、その箇所はコピーされたクラスである可能性が高い。

以上のように、コメント率の計測結果より品質についていくつかの問題を検出することができる。

5. おわりに

以上述べてきたように、問題検出・品質分析ツールとしてEJAQUETを開発した。EJAQUETを使えば非常に少ない工数で点検できるので、スケジュールに影響を与えることなく、ソースコード全量を点検できる。また、検出精度が高いので問題箇所を見逃さず、実効性のある点検が期待できる。

EJAQUETは弊社の指定標準として、プロジェクトへの適用が広がっている。そして、使用したプロジェクトには有用性や実行の容易さも評価されている。

現在、さらなる精度の向上と利用の拡大を目指し、次のような改善策を検討している。

(1)機能向上

- 分析機能・性能の向上
  品質やメトリクスの時系列変化の分析

問題検出能力の向上

- 使いやすさの向上

ソースコードの品質のレベル評価

問題箇所・問題種類の判定

(2)利便性・運用性の向上

- 実行環境の拡大
- チーム開発共通サーバにEJAQUETを載せ、ユーザに提供

今後ともJavaソースコードを品質向上させるため、EJAQUETの機能を向上させていきたい。

参考文献

1) 井関知文,「エクサにおける成果物品質管理の取り組み」、exa review Vol.8、2007
2) FindBugs,http://findbugs.sourceforge.net/
3) Checkstyle,http://checkstyle.sourceforge.net/
4) PMD,http://pmd.sourceforge.net/
5) CAP,http://cap.xore.de/
6) Eclipse Metrics Plug in, http://www.eclipsewiki.net/eclipse/index.php?%A5%E1%A5%C8%A5%EA%A5%AF%A5%B9%A5%D7%A5%E9%A5%B0%A5%A4%A5%F3
7) 循環的複雑度：http://ja.wikipedia.org/wiki/%E5%BE%AA%E7%92%B0%E7%9A%84%E8%A4%87%E9%9B%91%E5%BA%A6
