B_020

ソースコードにおけるコメント文の密度と保守性との関係に関する考察

A Study on Relationships between Comment Statement Density and Source Code Maintainability

岡崎 博和 † 阿萬 裕久 † 山田 宏之 †
Hirokazu Okazaki Hirohisa Aman Hiroyuki Yamada

1 はじめに

高品質なソフトウェアを開発する有効な手段として，高度な技術者による徹底したコードレビュー[1]が挙げられる．しかしながら，現実には人的資源の不足や開発の遅れといった制約も多く，すべてのコードに対して十分なレビューが実施されるのは困難である．それゆえ，レビューによって発見されなかったフォールトがコード中に残ってしまい，結果的に品質の低いコードが混在してしまう可能性も無視できない．そのような事態を未然に防ぐため，コーディング段階での品質保証も重要視されてきている．具体的には一定のコーディング規則を定め，組織内の技術者に義務付けるといった活動が行われている[2]．その中でもコメント文の記述の徹底は最も基本的なものである．コメント文はソースコードの理解を助ける働き[3],[4]があり，適切にコメント文を記述していくことで開発者によるコードの自己レビューが推進される．それによりコード中に含まれるフォールトが発見されやすくなり，ソースコードの品質向上が期待される．一般に，コメント文が多く書かれているコードの方が品質は高い傾向にあり[5]，またそのような上質なコードであれば小さな変更でバージョンアップに対応できると考えられる．しかし，コメント文の量に統一的で明確な基準が存在するわけではない．そこで本論文ではコメント文のソースコードにおける密度に着目し，実験を通して適切な目安を統計的に見い出す．

2 コメント文の密度と変更率

2.1 コメント文

コメント文はソースコード中に記述される注釈であり，ソースコードの理解を助ける働きがある．コメント文の適切な記述は，ソースコードの自己レビューを促すと考えられ，ソースコードの品質を高める上で重要な役割を担っている．本論文では，現在主流のオブジェクト指向言語である Java 言語を使って議論する．Java 言語の場合，次に示す 3 種類のコメント文を記述可能である[6]：

(1) End Of Line Comment（EOLC）：
"//" から行末までがコメント文として扱われる．主にメソッド内部における注釈文として用いられる．

(2) Documentation Comment（DC）：
"/**" と "*/" で囲まれた内容がコメント文として扱われる．メソッドやクラスの機能説明として用いられる．

(3) Traditional Comment（TC）：
"/*" と "*/" で囲まれた内容がコメント文として扱われる．主にソースコードの一部を無効化するのに使われることが多い．

図 1 に例を示す．便宜上，コード断片の先頭行を 1 行目とすると，EOLC が 13 行目，DC が 1 ～ 4 行目，TC が 7 ～ 9 行目にそれぞれ対応する（表 1）．

```
         .........
 1: /**
 2: * Restores the workspace tree from snapshot files
 3: * in the event of a crash.
 4: */
 5: protected void Snapshots(Monitor monitor) {
 6:     monitor = Policy.monitorFor(monitor);
 7:     /* String message;
 8:     monitor.beginTask(null, Policy.totalWork);
 9:     */
10:     IPath snapLocation = getMetaArea();
11:     java.io.File localFile = snapLocation.toFile();
12:
13:     // Just initialize the snapshot file and return.
14:     if (!localFile.exists()) {
15:         initSnap();
16:         return;
17:     }
         .........
```

図 1 Java ソースコード断片の例

表 1 図 1 におけるコメント文の数

種類	対応行	行数
EOLC	13	1
DC	1 ～ 4	4
TC	7 ～ 9	3

TC はソースコードの一部を無効化するのに使われることが多いため，我々は TC を除く EOLC 及び DC にのみ着目する．本論文では，これらのコメント文がソースコードの品質を高める働きがあると仮定し，その効果を実験によって検証する．

以下，コメント文の量について，クラス間で比較することを考慮し，ソースコードの規模に対する密度でもって評価する：

$$\text{コメント文の密度} = \frac{\text{ソースコード中での EOLC と DC の行数}}{\text{LOC}} \tag{1}$$

ただし，LOC（Lines Of Code）とはソースコード中のコード行数（コメント文，空文を除く）のことであ

† 愛媛大学大学院理工学研究科電子情報工学専攻，
松山市文京町 3, 790–8577 Japan;
E-mail:okazaki@hpc.cs.ehime-u.ac.jp

る．例えば図1のコード断片では LOC = 8 となる．したがって，このコード断片におけるコメント文の密度は 5/8 = 0.625 となる．ここでいうコメント文の密度はソースコード中にコメント文が占める割合ではなく，LOC に対する割合であるので1を超える場合もある．

2.2 変更率

本論文では，ソフトウェアのバージョンアップに伴うソースコードの変更（修正，追加，削除）行数をソフトウェアの変更・発展のための作業により施された変更量として用いる．ただし，コメント文及び空文についてはこの限りではないとする．ここでいう変更には“修正”，“追加”，“削除”の3種類があるが，それぞれ次のように定義し，変更量に計上する[7]：

便宜上，バージョンアップ前のソースコードを S_1，バージョンアップ後のソースコードを S_2 とする．S_1 に対して以下の3種類の操作のいずれかを繰り返し適用すれば S_2 と同一のコードが得られる．

(1) 連続した n（≥ 1）行のコードをそれとは内容の異なる連続した m（≥ 1）行のコードに置き換える．ただし，いずれの行も内容は異なるものとする．

(2) 連続した m（≥ 1）行のコードを挿入する．

(3) 連続した n（≥ 1）行のコードを削除する．

このとき，(1) の操作に該当する部分をソフトウェアの“修正”部分と定義し，変更量の計上には n と m の大きい方を用いる．(2) の操作に該当する部分を“追加”部分と定義し，変更量の計上には m を用いる．(3) の操作に該当する部分を“削除”部分と定義し，変更量の計上には n を用いる．□

本来，ソフトウェアの変更作業にはフォールトの除去のための作業と機能変更・向上のための作業が混在しているが，ソースコードのみからこれらを分類するのは困難であるため本論文では区別しない．

```
変更前
1: public class HelloWorld {
2:   public static void main(String[ ] args) {
3:     for ( int i = 0 ; i < 10 ; i++ ) {
4:       System.out.println("HelloWorld!!");
5:       System.out.println("Hello");
6:     }
7:   }
8: }
変更後
1: public class HelloWorld {
2:   public static void main(String[ ] args) {
3:     for ( int i = 0 ; i < 5 ; i++ ) {
4:       System.out.println("[" + i + "]");
5:       System.out.println("HelloWorld!!");
6:     }
7:     System.out.println("END");
8:   }
9: }
```

図2　変更前及び変更後の Java ソースコードの例

図2の2つの Java ソースコードを用いて変更量の例を示す．ただし本論文では，ソースコードの変更に対するコメント文の影響を解析するため，コメント文そのものの変更については考慮しない．追加行は変更後のソースコードの7行目，削除行は変更前のソースコードの5行目，変更行は変更後のソースコードの3行目と4行目である．したがって図2の場合，総変更行数は4である（表2）．コメント文の密度と同様に，変更量についてもソースコードの規模に対する密度で議論する．ここではこれを変更率と呼び，次式で定義する：

表2　測定結果の内容

種類	対応行	行数
追加行	変更後の7行目	1
削除行	変更前の5行目	1
変更行	変更後の3,4行目	2

$$変更率 = \frac{総変更行数}{\mathrm{LOC}}. \quad (2)$$

ただし，ここでの LOC はバージョンアップ前のものであり，総変更行数は追加，削除，変更の各行数を合計したものである．したがって，図2の変更率は 4/8 = 0.5 となる．実際には変更率が1を超える場合もある．すなわち元のコードと同じかそれ以上の量だけ“修正”，“追加”，“削除”が行われる場合もある．この値が大きいということは，大がかりな変更が行われたことを意味する．例えば，この値が2となったとすると元のコードに対してその2倍の量の“修正”，“追加”，“削除”が行われたことが分かる．上質なソフトウェアであれば，大きな仕様変更が起こらない限り，この変更率は小さいものになると期待される．

3 実験

本研究では，“コメント文が多く書かれている方がソースコードの品質は高くなる傾向があり，小さな変更でバージョンアップに対応できる”という仮定を考え，これを実験によって確認する．

3.1 実験内容

コメント文の密度（(1) 式）とソースコードの変更率（(2) 式）との関係を分析するため，オープンソースソフトウェアに対するデータ収集及び統計的分析を行った．

実験対象には著名なソフトウェアの1つである“Eclipse”[8]を使用した．本実験では Eclipse における3度のバージョンアップ“2.0 → 2.1”及び“2.1 → 3.0”,“3.0 → 3.1”についてデータ収集を行い，分析と検証を行った．

以下，実験の手順（図3）と結果を示す：

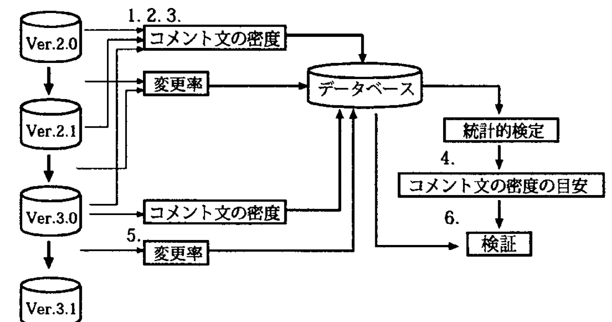

図3　実験の流れ

1. 分析データ（ver.2.0 → 2.1 及び 2.1 → 3.0）の収集
   各クラスにおけるコメント文の密度及びバージョンアップ前後でのソースコードの変更率を算出

した．

2. データの分類：抽象クラスと具象クラス
   抽象クラスでは，一部若しくはすべてのメソッドが実装を持たない．つまり，実装部分に対応するコメント文も含まれないことになる．また，抽象クラスは具象クラスによってオーバーライド（カスタマイズ）されることが前提となっており，そもそも大きな変更は想定されていない．それゆえ，本実験では抽象クラスは除外する．

3. データの分類：クラスの規模
   小規模のコードにおいては，その規模の小ささゆえにコメント文の密度や変更率の個体差が大きいと考えられる．そのようなコードはノイズデータとなるため本実験では除外する．実際には各クラスのLOCの中央値以下のものを分析データから除外した（付録A参照）．今回は具象クラスの中央値が64行であった．結果としてLOC > 64となっている具象クラスのバージョンアップ（3,491件）を分析データとした．

4. コメント文の密度を使った統計的検定
   コメント文の密度についてある閾値τを考え，分析データを2つのグループに分類する（図4）：
   - グループ1：コメント文の密度$\leq \tau$；
   - グループ2：コメント文の密度$> \tau$.

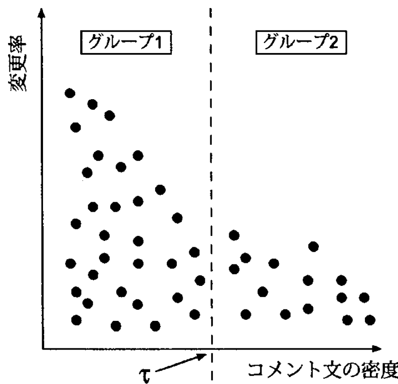

図4　コメント文の密度を使ったグループ分け

ここで，τを$0.05 \leq \tau \leq 1.0$の範囲で0.01刻みで変化させ，それぞれ次の仮説検定を有意水準5%で行った．（本実験では，データの分布の偏りや標本数を考慮して上記の範囲でτを変化させて検定を行った（付録B参照）．）
- 帰無仮説（H_0）：グループ1でのソースコードの変更率とグループ2での変更率は等しい；
- 対立仮説（H_1）：グループ1でのソースコードの変更率はグループ2でのそれよりも高い．

ただし，データ分布には正規性が見られなかったため，t検定は適さず，代わりにノンパラメトリック検定の一種であるWilcoxon検定[9]を用いた．検定の結果，各τ（コメント文の密度の閾値）における仮説検定のp値[*1]は図5のようになった．

帰無仮説を棄却できたτの集合における中央値をコメント文の密度の代表閾値τ_0とした．（本実験では，データの分布の偏りも考慮して，平均値ではなく中央値を代表値とした．）（表3）
実験結果よりこの閾値τ_0は，コメント文の密度の目安として使用できると考えられる．

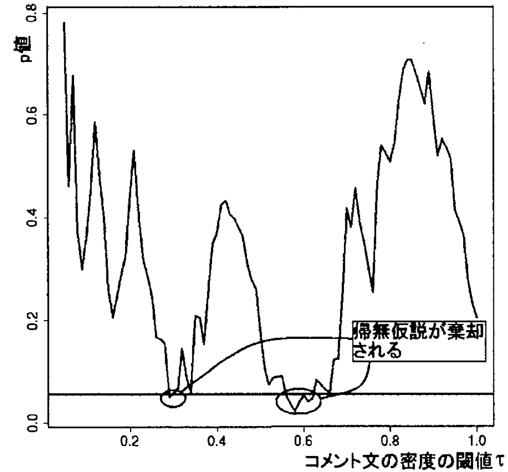

図5　コメント文の密度に対するp値

表3　帰無仮説を棄却できるコメント文の密度

最小値	中央値（τ_0）	最大値
29.0%	58.5%	62.0%

5. 検証データ（ver.3.0 → 3.1）の収集
   検証データとして使用するため，1.〜3.と同様の手順でEclipse ver.3.0に対してデータ収集を行った．結果として3,145件のデータを検証データとした．

6. 検証
   4.で見い出したコメント文の密度の閾値τ_0（= 58.5%）を使って検証データを上述のグループ1と2に分け，変更率の差について検定を行った．その結果，p値は6.10×10^{-6}となり，帰無仮説を棄却できた（表4）．つまり，コメント文の密度が閾値τ_0以下か否かでソースコードを分類することの有意性が確認された．□

表4　検証結果

閾値	p値	検定結果
58.5%	6.10×10^{-6}	○

今回の実験により，LOCが65行以上の具象クラスでコメント文の密度が58.5%以上ならば“コメント文が多く書かれている方がソースコードの品質は高くなる傾向があり，小さな変更でバージョンアップに対応できる”という仮定を満たすことを統計的に確認できた．

この結果はJava言語でソフトウェア開発を行う際の開発基準の1つとして使用でき，ソフトウェア品質向上の一助とすることが可能であると考えられる．ただし，これは単にコメント文を書けば品質が高いという意味で

[*1] 統計的検定の有意性を示すための確率値のこと．p値が有意水準より小さければ，帰無仮説は棄却される．

はない．結果として実験データにそのような傾向が確認されたのは事実であるが，各コメント文がそれぞれどのような効果を持っていたのかという部分までは追跡できていない．この点に関してはコメント文の持つ意味を解析する必要があると考えられる．

4 まとめと今後の課題

本論文では，ソフトウェアのソースコードにおけるコメント文とソースコードの保守性との関係に着目した．コメント文の記述を徹底させることで開発者によるコードの自己レビューが推進され，ソースコードの品質向上が期待される．そこでオープンソースソフトウェアEclipseにおけるバージョンアップ（6,636件）について分析を行った．分析の結果，LOCが65行以上の具象クラスにおいてコメント文の密度が58.5%以上ならば“コメント文が多く書かれている方がソースコードの品質は高くなる傾向があり，小さな変更でバージョンアップに対応できる”という仮定を満たすことを統計的に確認できた．今後の課題として，他のメトリクスとの比較やコメント文を書くことによる効果の大きさを求める等が挙げられる．

参考文献

[1] G. Sabaliauskaite, S. Kusumoto, and K. Inoue： Extended metrics to evaluate cost effectiveness of software inspection, IEICE Trans. Inf. & Syst., vol.E87-D, no.2, pp.475–480, Feb. 2004.

[2] 上田直子，伊藤雅子：組込みソフトウェア開発におけるソースコード品質向上活動, 組込みソフトウェアシンポジウム2004論文集, pp.54–57, 2004.

[3] 小泉寿男，辻秀一，吉田幸二，中島毅：ソフトウェア開発，オーム社，2003.

[4] Shari Lawrence Pfleeger： Software Engineering Theory and Practice Second Edition，Prentice-Hall, Inc., 2001.

[5] Richard W. Selby：Enabling Reuse-Based Software Development of Large-Scale Systems, IEEE Trans. Softw. Eng., vol.31, no.6, pp.495–510, Jun.2005.

[6] http://jp.sun.com/java/.

[7] H. Aman, N. Mochiduki, H. Yamada, and M.T. Noda：A simple predictive method for discriminating costly classes using class size metric, IEICE Trans. Inf. & Syst., vol.E88-D, no.6, pp.1284–1288, June 2005.

[8] http://www.eclipse.org/.

[9] 石村貞夫：分散分析のはなし，東京図書，1992.

付録A LOCの分布

表5 LOCの分布について

最小値	下位25%点	中央値	平均値	上位25%点	最大値
3	29	64	131.8	144	5,230

付録B コメント文の密度の分布

表6 コメント文の密度の分布について

最小値	下位25%点	中央値	平均値	上位25%点	最大値
0	0.0960	0.233	0.344	0.437	5.835