Vol.2013-SE-179 No.1
2013/3/11

# フィールド・メソッド関係表によるプログラムの評価

吉田　信一朗[†1]，　紫合　治[†2]

ＵＭＬのクラス図ではクラスのインタフェースとなるフィールドやメソッドの仕様については表現できるが，メソッドの中身の概要については表現できない．クラスの振る舞いを理解する上では，メソッド内の処理概要が必要となる．ここでは，クラスのフィールドとメソッドの関係表によって，クラスの振る舞いの概要を表す表現法を提案する．この表が，複雑すぎるメソッドや理解しにくいフィールド，不適切なクラスなどの判断に利用できることを，実際のプログラムを使って評価した．

# Evaluation of Programs by Fields and Methods Table

SHINICHIRO YOSHIDA[†1]　OSAMU SHIGO[†2]

Class diagram in UML has enough information about fields and methods interface of the class, but nothing about the internal program structure of the methods. The abstract program information about the method is required to understand the class behavior without source code. This paper proposes fields and methods table to represent the abstract of class behavior. The table may be applicable to measure the metrics of class complexity and understandability.

## １．はじめに

以前クラス図を利用したプログラミング支援ツールを開発した．表によるクラスの情報の入力により，クラス図の生成やソースの機械的導出を実行できるツールである．[1]
しかしクラス図では，フィールドやメソッドの引数の情報はあるが，メソッドの中身に関する記述がない．ソースコードでは，クラスの中身をすべて見てプログラム理解をしなくてはならない．その中間にあたる新しい表現法が求められている．
そのため本研究では，設計・デバッグ段階でのメソッドの中身の新しい表現法を考える．
提案手法によるプログラム解析支援ツールの開発を主に行なった．

## ２．関連研究

### 2.1　表を利用したプログラム支援ツール[1]

以前作成したプログラミング支援ツールは，プログラム作成前の設計段階に表でクラス情報を入力することによりプロジェクトでのチーム開発において意識統一を行うことを目的として開発されたツールである．主に３つの機能がある．

一つ目に表入力したデータからのクラス図の生成である．表からクラス図を出力する機能であり，表を入力する際は手入力または次に説明するプログラムからの表生成の機能を使う必要がある．

二つ目にプログラムからの表生成の機能である．完成したプログラムから表を自動生成する機能である．

最後に表入力したデータからプログラムのひな形を生成する機能である．表に手入力したデータから，機械的導出部のひな形を自動で生成できる．

これらの機能により，プログラム作成前後の支援には成功したといえる．しかしこのツールだけでは，メソッド内部の詳細なプログラム理解は不完全である．

### 2.2　プログラムスライシング[2][3]

本研究の類似研究として，プログラムスライシングがある．プログラムスライシングとは，ある文の変数の値に影響するすべての文を抽出し，抽出された文のみに着目し処理方式を理解するプログラム簡略手法である．

本研究では，あるフィールドに着目し，そのフィールドを使うメソッドを抽出してフィールドの理解を深められる．プログラムスライシングはプログラム構造まで読み取って関係を抽出しているのに対し，本研究はメソッド内でフィールド名が出てきた場合に関係性を抽出するという簡易的な手法をとっている点が相違している．

### 2.3　Ｓ－Ｐ表[4]

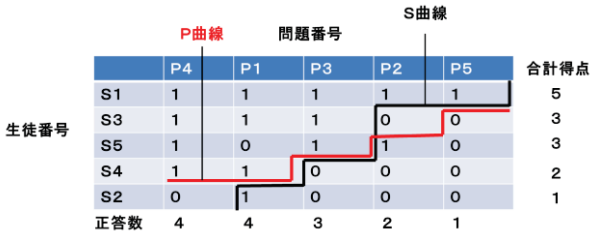
Ｓ－Ｐ表とは，教育情報工学で使われるテスト得点を図表的に表したものである．ひとりひとりの正

†1 東京電機大学大学院　情報環境学研究科
Graduate school of Information Environment , Tokyo Denki University.

†2 東京電機大学　情報環境学部
School of Information Environment , Tokyo Denki University.

答・誤答を照らし合わせて読み取ることのできる表になっている．
表３にＳ－Ｐ表の例を示す．
Ｓ曲線は全生徒の点数区切り線を結んだ得点分布曲線であり，Ｐ曲線は全ての問題の正答数の区切り線を結んだ正答者数分布曲線である．
　このＳ－Ｐ表を利用し，生徒と問題をメソッドとフィールドに置き換えて評価をおこなっていく．

表３．　Ｓ－Ｐ表の例




## ３．提案手法

### 3.1　概要

　フィールド視点からのメソッド毎のフィールドの動きに関する表にする．行部分にメソッド名，列部分にフィールド名を表示し，表の中身には該当するメソッドとフィールドの関係を表す．一つのフィールドが使用されているメソッドが多い順にフィールドをソートし，メソッドも多くのフィールドを使用している順にソートする．実際の例として表１をソートしたものを表３に示す．
　関係を表すためにメソッド内でフィールドが出てきた場合に以下の３種類の表示をする．
　（１）詳細表示：そのフィールド名を含むプログラムの行を表示する．ここでは簡単のためセミコロン（；）や for 文 if 文などのプログラム構造を無視して，改行で区切られた一行を取り出す．
　（２）代入，メソッド呼び出し，参照とその組み合わせにより分類する．
　代入は，フィールド名の後に＝が検出された際に代入と判断される．[]の配列等の時は，配列の要素部分を読み飛ばした後に＝が検出されるかどうか判断する．
　メソッド呼び出しは，フィールド名の後に．が検出された際にメソッド呼び出しと判断する．[]の配列等の時は，配列の要素部分を読み飛ばした後に．が検出されるかどうか判断する．
　代入でもメソッド呼び出しでもない場合にフィールド名が出てきた場合に参照と判断する．
　（３）１つのフィールドに着目した際に，メソッド内でそのフィールドが使用されたかどうかの有無の確認ができるよう Excel 表示を行う．この時に（２）で述べた組み合わせの分類のみを表示することにより，パターン解析が容易になるようにする．

　従来はメソッド視点からプログラムの作成を行っていたが，本研究ではフィールド視点からの設計・デバッグを提案する．従来のメソッド視点の考え方の表を表１に，フィールド視点の考え方の表を表２に示す．
　表１は，従来のエディタ等でプログラムする際にメソッド毎にプログラムをしていく考え方を表したものである．これは一つのメソッドに着目して，そのメソッドで関連するいくつかのフィールドがどのように処理されるかの概要を表す．例えば，ｍ１メソッドではｆ１フィールドに２４を代入し，ｆ２フィールドの set メソッド呼び出しを行っていることがわかる．さらにｆ３フィールドとｆ４フィールドはｍ１メソッドでは使用されていないことがわかる．
これはメソッドの内部処理の概要に対応する．
　一方表２は，従来の考え方とは違い，一つのフィールドに着目して，そのフィールドに関連するいくつかのメソッドでそれがどのように変更・参照されているのかを表したものである．これは 2.2 で述べたプログラムスライシングを抽象化したものと考えることができる．例えば，ｆ２フィールドに着目すると，初期値の宣言部で new され，ｍ１メソッドで set メソッド呼び出しを行い，ｍ３メソッドで get メソッド呼び出しを行っていることがわかる．さらにｍ２メソッドではｆ２フィールドが使用されていないことがわかる．

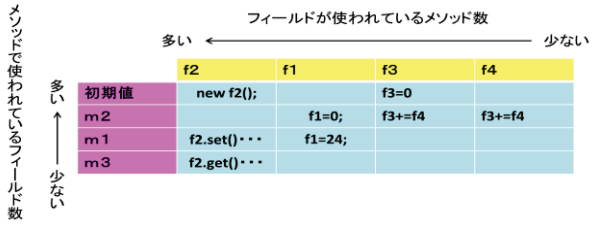
表．１　メソッド視点での考え方

| | f1 | f2 | f3 | f4 |
|---|---|---|---|---|
| 初期値 | | new f2(); | f3=0 | |
| m1 | f1=24; | f2.set()・・・ | | |
| m2 | f1=0; | | f3+=f4 | f3+=f4 |
| m3 | | f2.get()・・・ | | |

表．２　フィールド視点での考え方

| | f1 | f2 | f3 | f4 |
|---|---|---|---|---|
| 初期値 | | new f2(); | f3=0 | |
| m1 | f1=24; | f2.set()・・・ | | |
| m2 | f1=0; | | f3+=f4 | f3+=f4 |
| m3 | | f2.get()・・・ | | |

表．３　表１のソート結果



### 3.2　期待できる結果

表をプログラム作成後に適用した際に期待できる結果は以下のようなものがある．

（1）．バグの発見や不必要なコードの削減ができ、綺麗なプログラム作成に役立つ．
（2）．表のパターン解析を行なうことにより，クラスの複雑さを推定できる．
（3）．表のフィールド・メソッドをソートすることにより，フィールド・メソッドの重要度を推定できる．
（4）．他の開発者が作成したプログラムの理解．

（1）では，表のパターン，通常のプログラムとの相違点を見つけることにより，バグになる可能性のある箇所や不必要なコードの削減が行える．
（2）では，後に述べるＳＰ表等を用いることにより，クラスの複雑さを推定できるのではないかと考えている．
（3）では，フィールド・メソッドをソートすることにより，正確なものではないが大まかな重要度を推定し，プログラム理解をする際に役に立つのではないかと考えられる．
（4）では，3.1 で述べたフィールド視点からの見方により，それぞれのフィールドの役割を理解しやすくなるため，他の開発者が作成したプログラムの理解が深められると考えている．

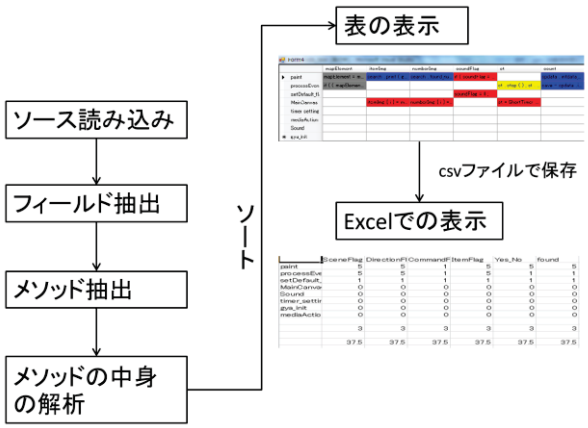
これらの結果が得られるような表作成のためのツール開発を行なった．

## ４．開発

提案手法を用いて，プログラム作成後にソースコードから表を自動生成するツールの開発を行った．開発環境は Visual Studio 2012 で，今回の対象言語は java のみとした．

### 4.1　ツールの構成

ツールの構成を図１に示す．
最初にソース読み込みを行い，ソースを単語単位で分ける．
次に単語ごとに分けられたデータを解析して，フィールド名を抽出する．同様にしてメソッド名の抽出も行なう．
その後，メソッドの中身を解析する．この際に，メソッドの引数やメソッド内のローカル変数の宣言も抽出し，それらと同じ名前のフィールド名が使われた時には this.がついたもののみを抽出する処理も行う．
最後に，メソッド毎にそのメソッドで使われているフィールドの数を求め，フィールド毎にそのフィールドが使われているメソッドの数を求め，それらの数によってメソッドとフィールドをソートする．この時に表の中身は，関係しているコード一行を表示して，使用のされ方によって色分けしている．
表を解析する際に，Excel での表示もできるように csv ファイルでの保存もできる．Excel の表は，主にパターン解析に利用するもののため，ツール上での一行表示の部分は省略し，組み合わせの分類のみを数字で表現した．




図．１　ツールの構成

### 4.2　ツールによって生成される表

本ツールで表示される表の例を図２に，Excel で表示される表の例を図３に示す．
ここで表示される色については６種類あり，Excel での表示は数字で表現されている．その詳細は以下の通りである．

（1） 赤色「１」：変更
（2） 黄色「２」：フィールド名．～
（3） 青色「３」：参照
（4） 水色「４」：（1）＋（2）
（5） 灰色「５」：（1）＋（3）
（6） 桃色「６」：（2）＋（3）
（7） 緑色「７」：（1）＋（2）＋（3）

「」内は Excel での表示方法．

黄色の（3）の場合はメソッド呼び出し等で，変更・参照が判別不可のため，（1）（2）とは別に作った．
また１つのメソッドで複数回同じフィールドが使用される場合には，マウスオーバにより複数行の表示がされる．


図２．　本ツール使用時の表


図３．　本ツール使用時の Excel 表示

## 5. 評価

### 5.1　評価方法・結果

今回の評価において，大学３年次に作成したプログラム２１個とＪＤＫの java ファイル３つを対象とした．

大学３年次に作成したプログラムのツール適用結果の例として，２つのプロジェクトの内の一つ目のプロジェクトのプログラム適用結果とソースコードからのプログラム理解をした際の考察を示す．

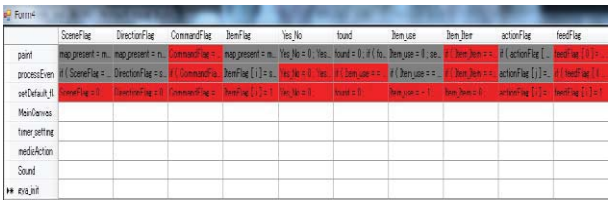



図４．PrisonBreak.java

図４は PrisonBreak.java の適用結果（一部抜粋）である．このプログラムは Main プログラムであり，約 1500 行である．２つのフィールドにおいてメソッド内で全く使われていなかった．ソースコードを解析した結果，２つともものフィールドが削除し忘れたものであった．また１３つのフィールドは１つのメソッドでしか使われておらず，約半数の７つのフィールドはメソッド内のローカル変数としてプログラムしても問題なかった．また２つのメソッドで１つのフィールドも使っていなかったが，これも削除し忘れのものであった．


図５．Base_Display.java

図５は Base_Display.java の適用結果である．このプログラムは約 250 行である．１つのフィールドが１つのメソッドでしか使用されていなかった．このフィールドはメソッド内のローカル変数としてプログラムしても問題なかった．

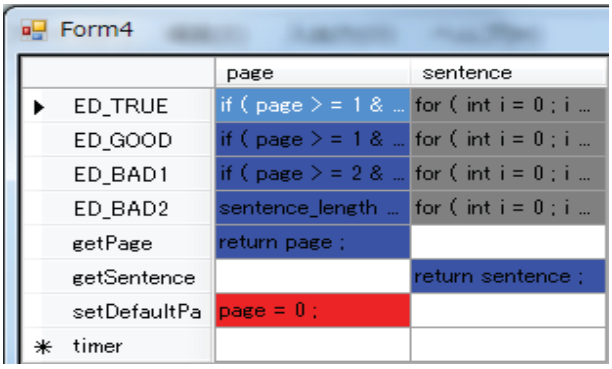



図６．Ending.java

図６は Ending.java の適用結果である．このプログラムは約 850 行である．１つのメソッドでは，フィールドを全く使用していなかった．このメソッドに関しては，引数のみを利用しているメソッドであるためこのクラスにプログラムしている意味はなく，単独のクラスとして作成する方式もとれた．

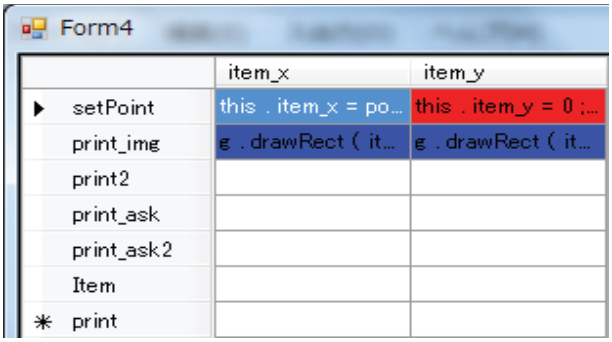



図７．Item.java

図７は Item.java の適用結果である．このプログラムは約 170 行である．４つのメソッドで１つのフィールドも使っていなかった．これは４つ全てのメソッドで引数が多く設定されていて綺麗なプログラムとは言えないものであった．

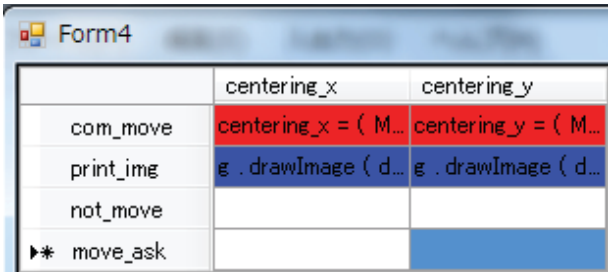



図８．Move.java

図８は Move.java の適用結果である．このプログラムは約 100 行である．２つのメソッドで１つのフィールドも使っていなかった．これは２つともものメソッドで引数が多く設定されていて綺麗なプログラムとは言えないものであった．

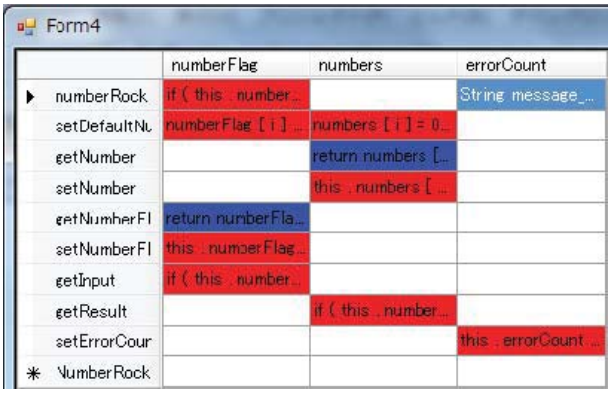



図９． NumberRock.java

図９は NumberRock.java の適用結果である．このプログラムは約 130 行である．表結果から問題点となり得る特徴は見られなかった．

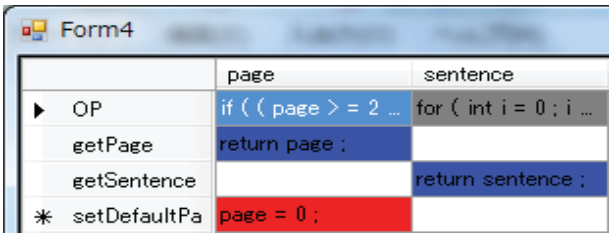



図１０． Openning.java

図１０は Openning.java の適用結果である．このプログラムは約 200 行である．このプログラムでも表結果から問題点となり得る特徴は見られなかった．


図１１． Search.java

図１１は Search.java の適用結果である．このプログラムは約 700 行である．１つのフィールドが１つのメソッドでしか使われていなかった．しかしこのフィールドはローカル変数ではなく，フィールドとしてプログラムすべきものであった．

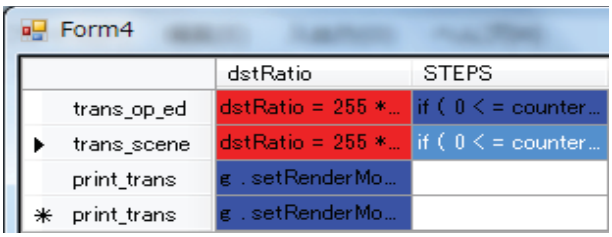



図１２． Transition.java

図１２は Transition.java の適用結果である．このプログラムは約 180 行である．表結果から問題点となり得る特徴は見られなかった．

ここからは二つ目のプロジェクトのプログラム適用結果とソースコードからのプログラム理解をした際の考察を示す．

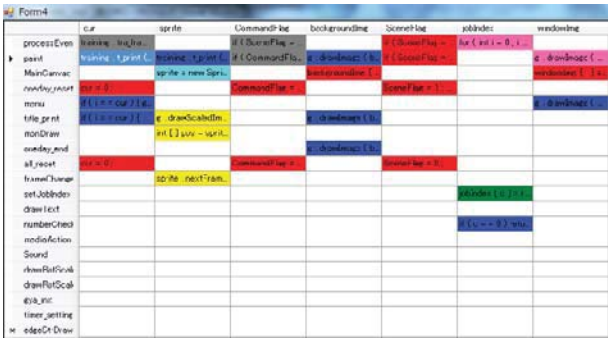



図１３． Symulation.java

図１３は Symulation.java の適用結果である．このプログラムは Main プログラムであり，約 2000 行である．２０個のフィールドが１つのメソッドでも使われていなかった．これは複数の開発者でプログラムの作成を行ったため，不必要なフィールドの消し忘れが原因である．また７つのメソッドでは１つのフィールドも使用していなかった．不必要なメソッドの削除と詳細なクラス分けが必要である．

図１４． Batlle.java

図１４は Batlle.java の適用結果である．このプログラムは約 500 行である．６つのメソッドで１つのフィールドも使用していなかったが，特に問題の無いメソッドであった．


図１５．BatlleCommand.java

図１５は BattleCommand.java の適用結果である．このプログラムは約 100 行である．１つのフィールドで１つのメソッドでしか使われていなかった．ローカル変数としてプログラムしても問題ないものであった．

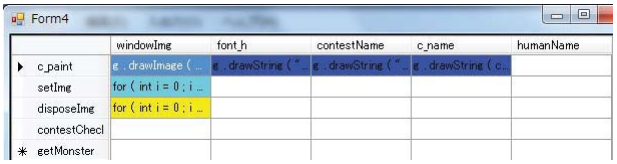



図１６．Contest.java

図１６は Contest.java の適用結果である．このプログラムは約 120 行である．１つのフィールドが１つのメソッドでも使用されていなかった．２つのメソッドでは１つのフィールドも使われていなかった．これは引数が多すぎるものだったので修正するべきである．

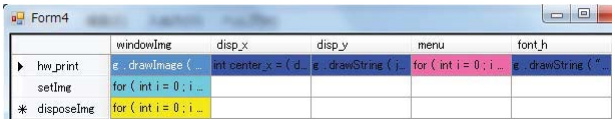



図１７．HelloWork.java

図１７は HelloWork.java の適用結果である．このプログラムは約 120 行である．３つのフィールドが１つのメソッドでしか使われていなかったが，３つ全てがローカル変数としてプログラムしても問題なかった．

図１８．LevelUp.java

図１８は LevelUp.java の適用結果である．このプログラムは約 200 行である．４つのフィールドがメソッド内で使われていないという結果になったが，これは仕様変更をした際に変更前のフィールドが残ってしまったためである．


図１９．Mix.java

図１９は Mix.java の適用結果である．このプログラムは約 250 行である．３つのフィールドが１つのメソッドでも使用されていないが，これは仕様変更による削除漏れであった．


図２０．Openning.java

図２０は Openning.java の適用結果である．このプログラムは約 100 行である．１つのメソッドがフィールドを１つも使用していなかったが，プログラム上問題はなかった．

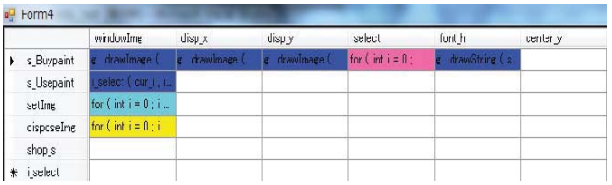



図２１．Shop.java

図２１は Shop.java の適用結果である．このプログラムは約 250 行である．１つのフィールドが使われていない．必要なフィールドだと考え用意したが，実際には使用せず消去し忘れたものであった．


図２２．Sleep.java

図２２は Sleep.java の適用結果である．このプログラムは約 120 行である．このプログラムでは表の

結果からは問題のある点はなかった.

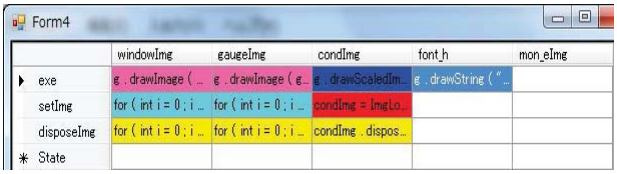



図２３．State.java

図２３は State.java の適用結果である．このプログラムは約 120 行である．１つのメソッドでも使われていないフィールドと１つのメソッドでしか使われていないフィールドが一つずつあった削除を忘れていたものであったり，メソッド内のローカル変数として宣言しても問題ないものであった.

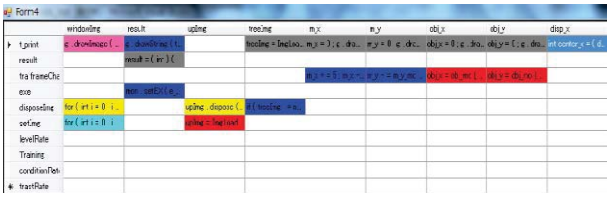


図２４．Trainning.java

図２４は Trainning.java の適用結果である．このプログラムは約 250 行である．１１個のフィールドは１つのメソッドでしか使用されていない．このうち９つのフィールドに関してはメソッド内のローカル変数として宣言できるものであった.

ＪＤＫの java ファイルでは，解析は未完だが，String.java，Short.java，Double.java の３つのプログラムをツール適用してみたところ約半数のフィールド・メソッドは１つのメソッド・フィールドも使用していないことが判明した.

またツールの改良点を発見できた．パッケージへの対応や，メソッドのオーバーロード時の引数の表示等の改善が必要になってくる.

### 5.2　評価のまとめ

大学３年次に作成したプログラムの解析を自身で行ったところ，多くのプログラムで１度も使われていないメソッドやフィールドが見つかった．これらはプログラム作成時にフィールド宣言やメソッド作成を行ったが，実際には使用せずに削除し忘れたものであった．また１つのメソッドでしか使用されていないフィールドもいくつか見つかった．これに関しては，約半数のフィールドがメソッド内でローカル変数として宣言しても問題ないものであり，修正をすることができた.

以上のことから，表結果からプログラム整理をする場合には，１つのフィールド・メソッドでも使用されていないメソッド・フィールドは不必要な可能性がある．１つのメソッドのみで使われているフィールドは，メソッド内のローカル変数としても表現できる可能性がある．ということに注意しながらプログラム整理することにより，綺麗なプログラム作成に役立つといえる.

プロジェクト毎に比較するために，プロジェクト毎の１メソッドあたりの平均フィールド使用率のグラフを図２５に，１フィールドあたりの平均フィールド出現率を図２６に示す.

図にある PrisonBreak プロジェクトは 5.1 の結果の一つ目のプロジェクトで，Symulation プロジェクトは二つ目のプロジェクトである．PrisonBreak プロジェクトが先に作ったもので，Symulation プロジェクトがその半年後に作成したものである．そのため Symulation プロジェクトの方がプログラムを理解した状態で開発したといえる.

図２５，図２６はともにプログラムの知識理解が深まるにつれて使用率・出現率が低くなっている．このことからオブジェクト指向的な考えのもと開発していることがわかる.

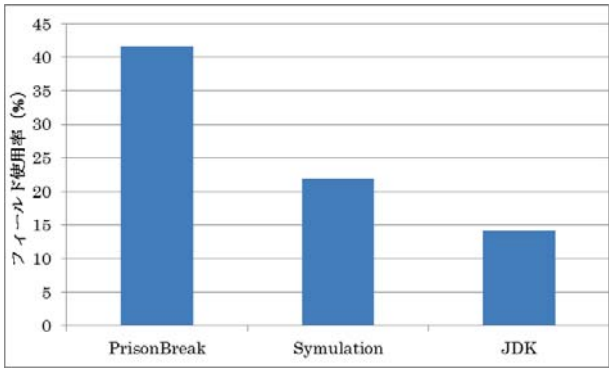



図２５．メソッドの平均フィールド使用率

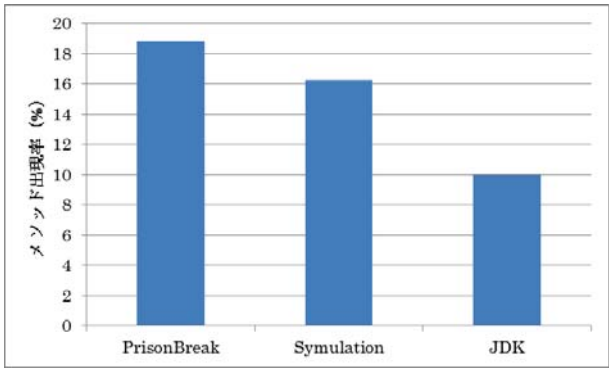



図２６．フィールドの平均メソッド出現率

二つのプロジェクトのフィールド使用率とメソッド数のヒストグラムを図２７，図２８に示す.

図２７は４０％と５０％，１００％に大きく偏った結果になった．０％に多くのメソッド数があるのは消し忘れのメソッドが多くあるからである．

図２８では緩やかに右肩下がりのグラフになった．

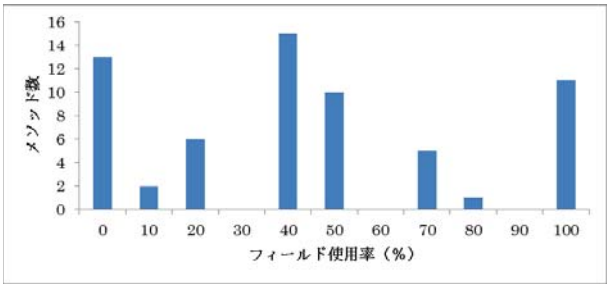



図２７．フィールド使用率のヒストグラム
（PrisonBreak）

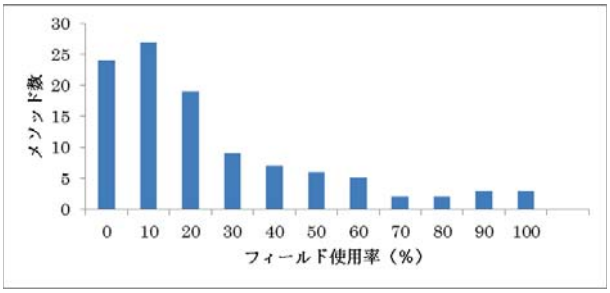



図２８．フィールド使用率のヒストグラム
（Symulation）

二つのプロジェクトのメソッド出現率とフィールド数のヒストグラムを図２８，図２９に示す．

図２９では２０～４０％に多くのフィールドがある．

図３０では３０％以下の出現率のフィールドが多い結果となった．

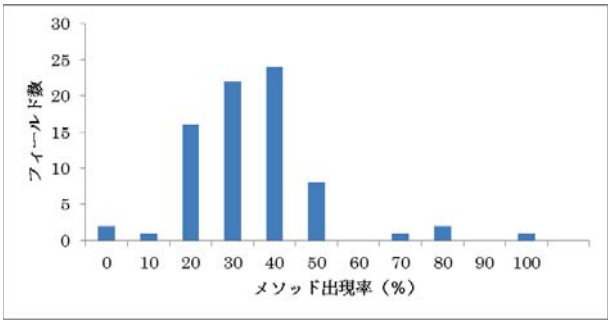



図２９．メソッド出現率のヒストグラム
（PrisonBreak）

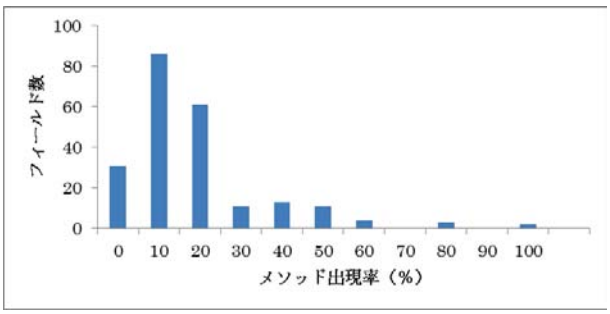



図３０．メソッド出現率のヒストグラム
（Symulation）

## 6. おわりに

フィールド・メソッド関係表によって，クラスの振る舞いの概要を示す方式について述べた．この表からプログラムの理解しやすさの判定の評価を行った．今回の研究で基本的なツールの開発は完成した．またグラフを用いたパターン解析もある一定の成果が得られた．

今後さらなるパターン解析をしていく所存である．Excel で表示する機能が実装済みであるため，それを利用してのデータ解析・パターン解析を行っていく予定である．さらに，2.3 で述べたＳ－Ｐ表等を用いたデータ解析も平行して行っていく予定である．

## 参考文献

[1] 湯浅正徳ほか４名，“表による java プログラミング支援ツールの開発”，情報処理学会学生セッション，2012.

[2] M.Weiser, “Program slicing”, Proc. 5th International Conference on Software Engineering, pp.439-449, 1981.

[3] 高田智則　井上克郎　大畑文明　芦田佳行，“制限された動的情報を用いたプログラムスライシング手法の提案”，電子情報通信学会論文誌　D-1 Vol.J85-D-1 No.2 pp.228-235, 2002.

[4] 佐藤隆博, 教育情報工学入門, コロナ社, 1989.1