NAIST-IS-MT1651123

修士論文

デザインパターン導入に起因するバグの混入調査

吉田 貴信

2018 年 3 月 15 日

奈良先端科学技術大学院大学

情報科学研究科

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士(工学) 授与の要件として提出した修士論文である。

吉田貴信

審査委員：

飯田 元　教授　(主指導教員)

松本 健一 教授　(副指導教員)

市川 昊平 准教授 (副指導教員)

崔　恩瀞 助教　(副指導教員)

デザインパターン導入に起因するバグの混入調査*

吉田　貴信

内容梗概

デザインパターンとは、ソフトウェアの設計上の問題対する解決策である。デザインパターンを利用する事で先人の優れた設計を利用でき、拡張性にすぐれたソフトウェア設計などが可能になると考えられる。

従来研究においては、デザインパターン適用がソフトウェアの品質に及ぼす影響が定量的に調査されている。たとえば、これまでに、デザインパターンを適用したソースコードに対するバグの混入についていくつかの調査が実施されている。これらの研究ではデザインパターンを適用したソースコードにおけるバグの混入率などが調査されてきた。

しかし、あるデザインパターンを適用した際に、ソフトウェアの保守性に具体的にどのような影響が出るのかについては明確になっていない。バグの混入率だけではなく、どのようなバグが混入しやすいかまでが明確になれば、開発者やテスト実施者などにとってデザインパターン適用の検討やテスト時に有益な情報となることが期待できる。

本研究では、ソースコード中でデザインパターンを適用した部分にどのようなバグが混入しているかを明らかにするため、オープンソースソフトウェアを対象としてデザインパターンを適用した部分にあたるソースコードに対するバグ修正履歴を調査し、その傾向を分析した。

キーワード

デザインパターン，Factory Methodパターン，バグ修正の分類

* 奈良先端科学技術大学院大学 情報科学研究科 修士論文，

NAIST-ISMT1651123，2018 年2 月1 日.

Investigation of code histories to find bugs caused by introduction of design patterns*

Takanobu Yoshida

Abstract

Design pattern is a solution to software design improvement. By using design patterns, it is possible to reuse the preexisting sophisticated software designs with expendability or other superiorities.

Recent studies quantitatively investigate the effect of applying design patterns on the quality of software. In these studies, the occurrence rate of bugs in the source code to which the design pattern was applied, have been investigated.

However, it is not clear what kind of effect will be exerted on software maintainability when applying a certain design pattern. If it is clarified what kinds of bugs are easy to mix in, it will be useful information for designing and applying the design pattern to developers, test operators, etc.

In this research, in order to clarify what kind of bugs occur in the part where the design pattern is applied in the source code, we investigated open source software repository to examine trend of the bug correction history of the part of code where the design pattern was applied to open source software

Keywords:

design pattern, open source software, factory method pattern, bug classification

---

* Master's Thesis, Graduate School of Information Science,
Nara Institute of Science and Technology, NAIST-IS-MT123456, February 20, 1995.

目次

図目次

表目次

1. はじめに

デザインパターンとは、過去の開発者によって作られたソフトウェアの設計問題に対する解決策である [1]。デザインパターンを利用することで、ソフトウェア開発者は先人の優れた設計を利用でき、拡張性にすぐれたソフトウェア設計などが可能となるといわれている。ソフトウェア開発現場などでは、過去のソフトウェア設計の理解やソフトウェア開発の効率化などの点でデザインパターンを利用している [2] [3]。しかし、これらのデザインパターンを利用した際の保守性やバグの混入率の関係について明らかにされておらず、調査が必要とも言われている [4] [5]。

デザインパターンとソフトウェアの品質に関わる研究として、Mahmoudらの研究がある [5]。彼らはJavaで書かれたオープンソースソフトウェア（以降、OSS）を対象に、デザインパターンを適用したクラスと適用していないクラス、デザインパターン同士等でのバグの混入率を調査した。調査の結果、「構造に関するデザインパターン」と呼ばれるデザインパターンにおいて、バグ混入率が低い事が分かった。また、「生成に関するデザインパターン」の 1 つであるBuilderパターンはバグ混入率が高いことが判明した。

バグ混入率が高いデザインパターンが分かることで、デザインパターン適用時の注意やテストやレビューなどで注力する箇所を絞る事が可能となると考えられる。一方、バグの混入率が判明するだけではどのようなバグ混入に注意するべきかといった、具体的な方法までは分からない。

デザインパターンを適用した際に、どのようなバグが混入しやすいかまで分かれば、デザインパターン適用時の注意やテストやレビュー時の注力する箇所がより明確になると考えられる。そこで、本研究ではデザインパ

ターン適用時に混入しやすいバグ修正の種類に着目した。従来研究と比べ、本研究はデザインパターン適用時のバグ修正の種類に着目した点で新規性があると考える。

本研究では、まず、OSSから取得できるコミット情報とデザインパターンリポジトリから取得できるOSSへの適用デザインパターンの正解集合を基に、Javaで書かれたOSSのクラスに適用しているデザインパターンを調査した。つぎに、デザインパターンを適用しているクラスのバグ修正履歴からバグ修正の種類を調査した。調査の結果、バグ修正として他クラスのメソッド呼出しの追加等を観察した。

以降、2章では本研究の背景となるデザインパターンやバグ混入率の調査の研究等を紹介する。3章では調査の目的と方法を述べ、4章で調査の結果と考察を述べる。そして、5章で妥当性の脅威を述べ、6章で本研究のまとめを述べる。

2. 関連研究および関連技術

本研究の背景であるデザインパターンと、デザインパターンとバグ混入に関する既存研究、そしてバグ修正の種類について説明する。はじめに、2.1節では本研究で扱うGoFのデザインパターンについて説明し、次に2.2節でデザインパターンとバグに関する既存研究、最後に2.3節でバグ修正に関する既存研究を紹介する。

2.1 デザインパターン

デザインパターンとは過去の開発者によって作られたソフトウェアの設計問題に対する解決策である。最も有名なデザインパターンの一つとして、GoF（Gang of Four）のデザインパターンがある [1]。GoFのデザインパターンには、Gammaらが提案した23種類のパターンがまとめられている。これらのパターンは「生成に関するパターン」「構造に関するパターン」「振る舞いに関するパターン」と大きく３つに分類され、デザインパターンを適用する背景、適用した際の効果、適用サンプルなどが記述されている。このGoFのデザインパターンはソフトウェアの設計上に現れる背景にあわせてデザインパターンを適用することができると考えられる。

デザインパターンは他にも存在しているが [6]、本研究ではGoFのデザインパターンを利用する。理由は、後述するデザインパターンリポジトリがGoFのデザインパターンを対象としているためである。デザインパターンリポジトリは特定のOSSに適用しているデザインパターンの正解集合として利用する。

ここで、GoFのデザインパターンの例としてFactory Methodパターンについて述べる。Factory Methodパターンは「生成に関するデザインパターン」の１つである。FactoryMethodパターンをソフトウェアへ適用するこ

とで、将来どのオブジェクトが必要になるかを気にすることなく、適宜オブジェクトを生成する事が可能となる。ここで述べているオブジェクトとは、オブジェクト指向におけるクラスの実体のことである。

図 1にFactory Methodパターンのクラス構造を示す。Factory Methodパターンは、Product、ConcreteProduct、Creator、ConcreteCreatorの4つの役割を担うクラス（以降、ロール）で構成されている。CreatorはProductを生成するためのインタフェースを定義する。ConcreteCreatorはCreatorの定義したインタフェースに対して、何のConcreteProductを生成するかを定義する。この構造により、将来どのConcreteProductが必要かわからない場合でも、対応するConcreteCreatorを作成することでオブジェクトを容易に生成することが可能となる。

また、本研究では、これらのロールの塊をデザインパターンインスタンス（以下、インスタンス）と呼ぶ。インスタンスの例を図 2に示す。図 2では、1つのソフトウェアに2つのインスタンスが使用されている。

インスタンス①はClassA、ClassB、ClassC、ClassD、ClassEの5つのクラスで構成される。ClassAはProduct、ClassBはConcreteProduct、ClassCはCreator、ClassDとClassEはConcreteProductのロールを担う。

また、インスタンス②はClassF、ClassG、ClassH、ClassI、ClassJ、ClassKの6つのクラスで構成される。ClassFはProduct、ClassGとClassHはConcreteProduct、ClassIはCreator、ClassJとClassKはConcreteProductのロールを担う。

Mainクラスは、FacideAとFacideBの何らかのメソッドを呼び出す。Mainクラスから呼び出されたFacideAとFacide Bのメソッドはそれぞれ FactoryMethodパターンのインスタンス①と②のConcreteCreatorを担う

クラスのfactoryMethodメソッドを呼び出す。各ConcreteCreatorは対応するConcreteProductを生成する、という動作を実施する。

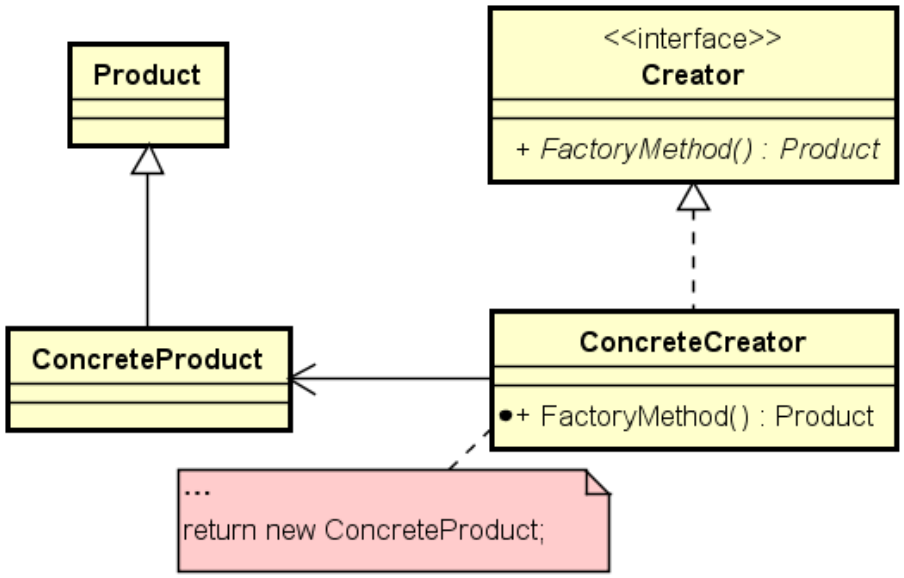


図 1　GoF のデザインパターンの例：Factory Method パターン

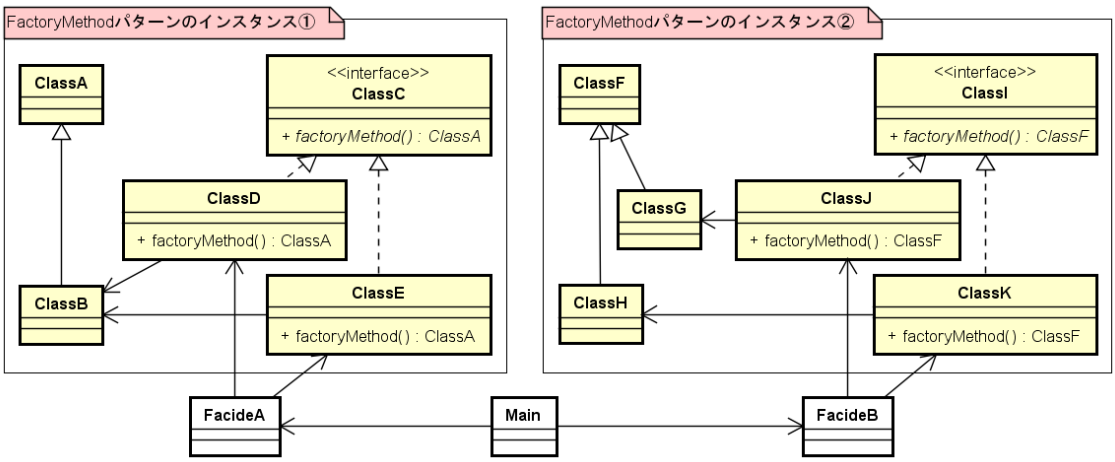


図 2　デザインパターンインスタンスが 2 つ存在する例

2.2 デザインパターンとバグに関する既存研究

デザインパターンとバグの関連を調査した研究として、Mahmodらの研究があげられる [5]。彼らはJavaで記述した５つのOSSを対象に、デザインパターンを使用したクラスにおけるバグの混入率を調査した。彼らの調査したバグの混入率とは、デザインパターンを使用しているクラスの数に対するバグ混入回数である。調査結果として、「生成に関するパターン」の１つであるBuilderパターンでのバグ混入率が高いことを定量的に示した。また、「構造に関するデザインパターン」はバグ混入率が低いことを定量的に示した。

これらの結果より、Builderパターンを利用する際はバグを混入しないようにレビューを厳しくする事などのバグ混入対策が可能になると考えられる。しかし、具体的にどのような点に気を付ければバグ混入を防げるかまでは分からない。

Ampatzoglouらは、デザインパターンを使用したソースコードにおけるバグ混入とバグ修正率を調査した [7]。彼らの研究では、96個のJavaで書かれたゲーム分野のOSSを対象として、デザインパターン適用クラスにおける欠陥率とデバッグ効率を分析している。彼らの述べる欠陥率とは、バグトラッキングシステムにおいて、bug Openと記述している数を指し、また、デバック効率とはbug Fixedと記述している数をbug Openの数で割った数を指す。調査の結果、8種類のデザインパターン（AbstractFactory、Singleton、Composite、Observer、State、Strategy、Prototype、Proxy）は欠陥率が低く、2種類（Adapter、TemplateMethod）のデザインパターンは欠陥率が高い事が分かった。

また、デバッグ効率については3種類（Adapter、Decorator、Observer）のデザインパターンがデバッグ効率を低下させ、1種類のデザインパターン（Singleton）はデバッグ効率を向上させることが分かった。この結果よ

り、8種類のデザインパターン利用時には、レビューやテストにかける負荷を大きくすることでバグ混入を防ぐことが可能になると考えられる。しかし、Mahmoudらの研究と同様に、どのようなバグが入っていたか、どのようなバグ修正が実施されていたかまでは明らかにしていない。

この他、M. Vokacの産業用ソフトウェアにおけるデザインパターンとバグ修正に関する研究 [8]においても、定量的な調査結果を記述しているが、「どのような種類のバグが混入しているか」「どのような種類のバグ修正を実施しているか」といったバグやバグ修正の種類まで述べていない。

以上より、従来のデザインパターンとバグに関する研究では、バグの混入率などは明らかにしているものの、どのようなバグが混入しているか、どのようなバグ修正を実施しているかという内容まで考慮されていない。

もし、バグの混入率だけではなく、どのようなバグが混入しやすいかまで明らかにできれば、開発者にとってはデザインパターン適用時にバグを発生させないように、どのような注意が必要かを示す事が可能になると考える。また、テストやレビューにおいて、ソフトウェアにデザインパターンが適用されている事が分かれば、ソースコードのどの部分に注意してテストやレビューすればよいか、という指針を立てる事が可能になると考える。

この他、企業がOSSを利用する際の判断基準にデザインパターンの適用の有無を利用する事も考えられる。そこで、本研究では、デザインパターンを適用したソースコードの出現しやすいバグを明らかにするために、OSSを対象にデザインパターンに混入するバグの種類、バグ修正の種類を調査する。

2.3 バグ修正パターン

本研究では、デザインパターンを使用した箇所におけるバグ修正内容を分析するために、Kaiらの提案するバグ修正パターン [9]を用いる。表 1はKaiらが提案したバグ修正パターンの一覧である。Kaiらは、Java言語で書かれた5つのOSSを対象にバグ修正の内容を分析し、バグ修正の種類を定義した。さらに、バグ修正の分類のために調査した7つのOSSを対象に、バグ修正パターンの傾向を調査した結果、バグ修正パターンのカテゴリとして、7つのOSSで「メソッド呼出し（MC）に」6つのOSSにおいても「メソッド呼出しでの、引数の値の変更」と「IF文の条件式の変更」の２つのバグ修正がよく実施されている事が判明した。本研究においても、デザインパターンを使用しているクラスでのバグ修正傾向を分析し、その結果をKaiらの調査したバグ修正の傾向と比較する。

なお、Yangyangらの研究でも同様にバグ修正パターンを定義している[10]。彼らはソースコードの修正箇所や修正内容等を自動検出するツールを開発し、そのツールを11のOSSに用いてバグ修正パターンを調査した。調査の結果、メソッド呼出しの変更というバグ修正はよく現れるバグ修正であること等を示した。

本研究ではJava言語を対象にしているKaiらのバグ修正パターンを用いる。理由としては、調査対象のOSSがJava言語で記述されている事や、呼び出しメソッドの変更というバグ修正パターンにおいても3種類のバグ修正パターンを定義しており、細かい分析を実施が可能であると判断したため、本研究ではKaiらのバグ修正パターンを用いる。

表 1　Kaiらの定義するバグ修正パターン [9]

カテゴリ	パターン名	ショートネーム
アサインメント	代入操作の右側を変更する	AS-CE
クラスフィールド	クラス変数の追加	CF-ADD
	クラス変数定義の変更	CF-CHG
	クラス変数の削除	CF-RMV
IF 文	else 文の追加	IF-ABR
	事前条件チェックの追加	IF-APC
	ジャンプ文（return や break、continue）を伴う事前条件チェックの追加	IF-APCJ
	事後条件チェックの追加	IF-APTC
	if 文内の条件式の変更	IF-CC
	else 文の削除	IF-RBR
	if 文の削除	IF-RMV
Loop 文	ループ条件の変更	LP-CC
	ループ条件で使用している変数の変更	LP-CE
メソッド呼出し	異なる実パラメータを伴うメソッド呼出し	MC-DAP

	クラスインスタンスの呼び出すメソッド変更	MC-DM
	異なるパラメータ数か異なるパラメータ型を伴うメソッド呼出し	MC-DNP
メソッド定義	メソッド定義の変更	MD-CHG
	メソッド定義の追加	MD-ADD
	メソッド定義の削除	MD-RMV
シーケンス処理	代入処理の追加	SQ-AFO
	メソッド呼出し処理の追加	SQ-AMO
	ブロックで囲まれた 2、3 行でのメソッド呼出しの追加または削除	SQ-AROB
	代入処理の削除	SQ-RFO
	メソッド呼出し処理の削除	SQ-RMO
Switch 文	スウィッチ分岐の追加または削除	SW-ARSB
トライ文	トライ文でのキャッチブロックの追加または削除	TY-ARCB
	トライ/キャッチ文の追加または削除	TY-ARTC

3. 調査目的及び調査方法

本章では、デザインパターンを適用したソースコードにおけるバグの調査について説明する。3.1節で調査の目的を述べ、3.2節で調査の方法を述べる。

3.1 調査目的

本研究の目的は、「デザインパターンを適用したソースコードにおいてどのような種類のバグがどれだけ混入しているかを明らかにすること」である。そのため、以下のリサーチクエスチョン（以下、RQ）を設ける。

RQ：FactoryMethodパターンの各ロールにおいて、どのような種類のバグ修正がどれだけ実施されているか。

なお、RQにおけるバグ修正の種類は2.3節で紹介したKaiらのバグ修正パターン [9]を用いる。バグ修正パターンの実施回数の数え方は3.2.2項にて記述する。また、調査ではGoFのデザインパターンの1つであるFactoryMethodパターンを使用しているクラスのバグ修正情報を取得し、分析する。FactoryMethodパターンを選択した理由は3.2.1項で記述する。

3.2 調査方法

3.1節で記述したRQを明らかにするために、3.2節では調査方法を説明する。3.2.1項で調査対象について説明し、3.2.2項で調査手順について説明する。

3.2.1 調査対象

本項では、調査するデザインパターンとOSSについて記述する。本研究では、GoFのデザインパターンの１つであるFactoryMethodパターンを調査する。

調査開始時点において、Mahmoudらの研究 [5]により、Builderパターンのバグ混入率が高いことが判明していたため、P-Martリポジトリを参照し、Builderパターンが適用されているOSS（PMD）において、Builderパターンを適用したクラスのロールを調査した。しかし、PMDにおいてBuilderパターンを使用しているクラスのロールが判別できなかった。そこで、Builderパターンと同様「生成に関するデザインパターン」であるパターンに調査対象を変更することとした。P-martリポジトリにおいて使用されている「生成に関するデザインパターン」は、AbstructFactory、Builder、FactoryMethod、Prototype、Singletonの５つであった。

一方、事前調査に着手していたOSS（PMD）では、P-martリポジトリにおいて使用されている「生成に関するデザインパターン」５つののうち、FactoryMethodパターンのみが使用されていたため、本研究ではまずFactoryMethodパターンについて調査を行なうことした。

P-martリポジトリにおいて、FactoryMethodパターンを使用しているOSS（「PMD」と「JHotDraw」）を調査対象とする。これらを対象とする理由は、P-martリポジトリにおいて、FactoryMethodパターンを使用しているOSSはPMDとJHotDrawの２つのみであったことである。PMDはJavaのソースコードを解析するツールであり、バグの可能性や利用されていないローカル変数、重複コードなどを識別する。JHotDrawは2次元グラフィックのフレームワークである。

表 2にPMDとJHotDrawの概要、使用言語、使用デザインパターン、FactoryMethodパターンのインスタンスの数をまとめる。なお、インスタンスの数は3.2.2項の調査手順①で調査した結果のものである。

表 2 調査するプロジェクトの諸元

OSS名	OSSの概要	使用言語	使用しているデザインパターン	FactoryMethodパターンのインスタンス数
PMD	Javaのソースコード解析ツール	Java	FactoryMethod, Builder, Adapter, Iterator, Composite, Observer, Template, Visitor	2つ
JHotDraw	図形描画エディター	Java	FactoryMethod, Prototype, Singleton, Adapter, Composite, Decorator, Command, Observer, State, Strategy, Template	2つ

3.2.2 調査手順

調査手順について記述する。調査では、バージョン管理システムとデザインパターンリポジトリから得られるデータを用いる。バージョン管理システムとデザインパターンリポジトリから取得できるデータのうち、調査で扱うデータを表 3、表 4にまとめる。

表 3に示すように、バージョン管理システムから得たデータからは、「コミットID」、「コミット日時」、「コミットコメント」、「コミットで変

更したファイル」、「コミットで変更した各ファイルの変更内容」を参照利用する。

また、表 4に示すように、デザインパターンリポジトリから得たデータからは、「OSS名」と「バージョン」、「OSSに適用しているデザインパターン名」、「OSSに適用しているデザインパターンを適用しているファイル名」を利用する。たとえば、「PMDのバージョン1.8において、FactoryMethodパターンを適用しているファイル名はファイルAとファイルB」のような形で、「特定のOSSの特定のリリースバージョンにおいて、デザインパターンを適用しているファイル」の正解集合として利用する。

調査手順の概略を図 3に示す。調査は以下の３つの手順に分けられる。

表 3　バージョン管理システムにおける調査で扱うデータ

扱うデータ	例
コミットID	b5546dd388fc7514eb884dfc8a0e08ea8cec2d8a
コミット日時	2004-08-04 03:18:41
コミットコメント	Added some Javadoc, thanks to Brant Langer …
変更したファイル	pmd/src/net/sourceforge/pmd/TargetJDK1_3.java pmd/src/net/sourceforge/pmd/TargetJDK1_4.java …
各変更したファイルの変更内容	@@ -1,6 +1,6 @@ -/** +/* …

表 4　デザインパターンリポジトリにおける調査で扱うデータ

扱うデータ	例
プロジェクト名	PMD v1.8
プロジェクトに適用しているデザインパターン名	Factory Method
プロジェクトに適用しているデザインパターンに関するファイル名	net.sourceforge.pmd.TargetJDKVersion net.sourceforge.pmd.TargetJDK1_4 …

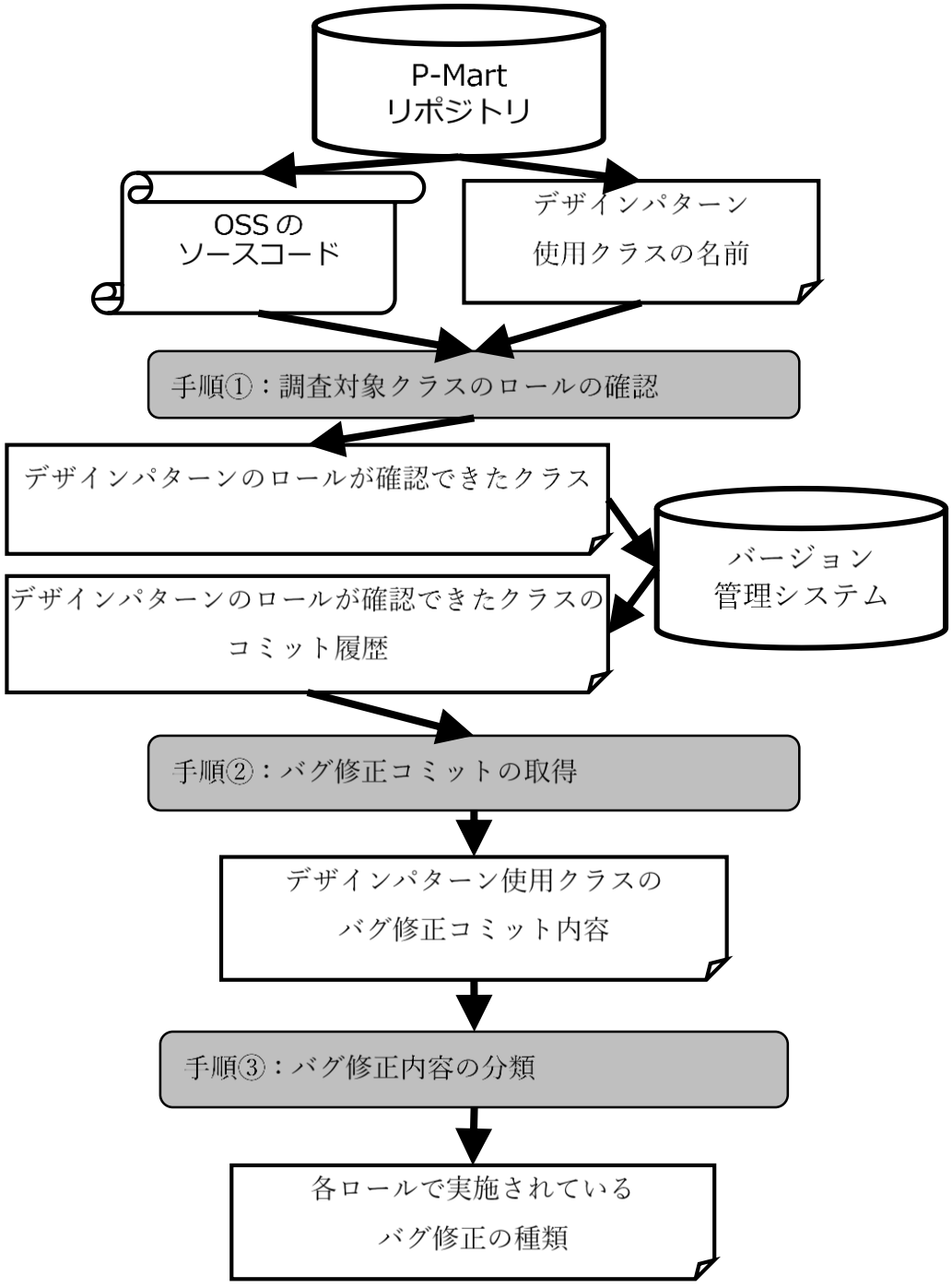

P-Mart
リポジトリ
OSS の
ソースコード
デザインパターン
使用クラスの名前
手順①：調査対象クラスのロールの確認
デザインパターンのロールが確認できたクラス
バージョン
管理システム
デザインパターンのロールが確認できたクラスの
コミット履歴
手順②：バグ修正コミットの取得
デザインパターン使用クラスの
バグ修正コミット内容
手順③：バグ修正内容の分類
各ロールで実施されている
バグ修正の種類

図 3　調査手順の概要

手順①：調査対象クラスが担当する FactoryMethod パターンの役割確認

手順①の目的は、FactoryMethodパターンを使用しているファイルがFactoryMethodパターンの何のロールを担当しているかを確認する事である。入力として、P-martリポジトリからOSSのソースコードとデザインパターン使用クラス名を用いる。出力として、FactoryMethodパターンのロールが確認できたクラスを取得する。

上記の目的を達成するために、はじめに表 4に示したP-martリポジトリから得られるデータを用いて、調査するデザインパターンと調査するOSSプロジェクトを決定する。次に、調査対象としたOSSのソースコードをP-martリポジトリ配布サイト* よりダウンロードする。次に、ダウンロードしたソースコードをデコンパイルツール（本研究ではJava Decompiler†を使用した）などで開き、P-martリポジトリから得られる情報から調査対象デザインパターンに関するクラス構造を確認する。次に、UMLツール（本研究ではAstah‡を使用した）などを用いてクラス構造を可視化する。最後に、可視化したクラス構造に対して、デザインパターンが使用されているかどうかの根拠を明示し、それらのクラスファイルはデザインパターンが使用されているものとして、調査対象とする。なお、図 4は、筆者が目視で確認したPMDのバージョン1.8におけるFactoryMethodパターンのインスタンスの１つである。

---

* http://www.ptidej.net/tools/designpatterns/

† http://jd.benow.ca/

‡ http://astah.change-vision.com/ja/

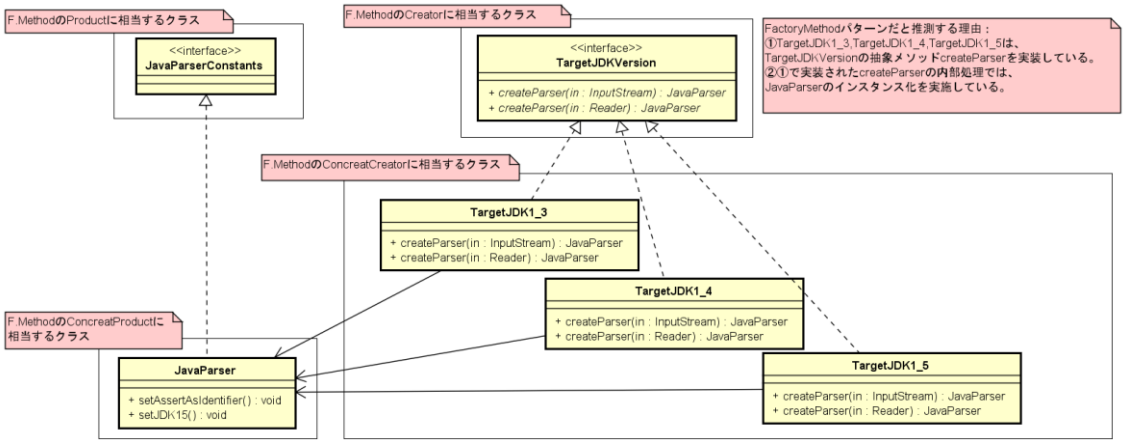


図 4 目視にて確認した PMD における FactoryMethod パターンのインスタンスの 1 つ

手順②：バグ修正コミットの取得

手順②の目的は、デザインパターン使用クラスのバグ修正コミットの取得である。入力として、手順①で取得した「FactoryMethodパターンのロールが確認できたクラス」のコミット履歴を用いる。出力として、「FactoryMethodパターンのロールが確認できたクラスのバグ修正内容」を取得する。

具体的には、手順①でFactoryMethodパターンのうち、Creator、Product、ConcreteCreator、ConcreteProductのいずれかのロールを担当している事が確認できたファイルのコミット履歴を分析することで、バグ修正コミットを取得する。

バグ修正コミットであるかどうかは、コミットコメントに”bug”または”fix”のキーワードが使用されているかを基準に判断する。ここで、JungやKimらの調査によると [11] [12]、コミットコメントからバグ修正コミットと判断した場合でも、実際はリファクタリングや機能追加、コメント文の変更などが含まれている事がある。したがって、取得したバグ修正コミ

ットに対しては、修正内容を目視で確認し、修正前後で機能が変化していない場合や修正箇所が1000行を超える場合などは、バグ修正でない可能性があるため本研究では調査対象から除外する。

さらに、FactoryMethodパターン使用クラスにおいてバグ修正が確認できたコミットでは、FactoryMethodパターンが保たれているかどうかを確認する。具体的には、バグ修正コミットの日付を参考に、FactoryMethodパターンを使用しているクラスの消去や継承関係の変化を確認して、FactoryMethodパターンが保たれているかどうかを確認する。

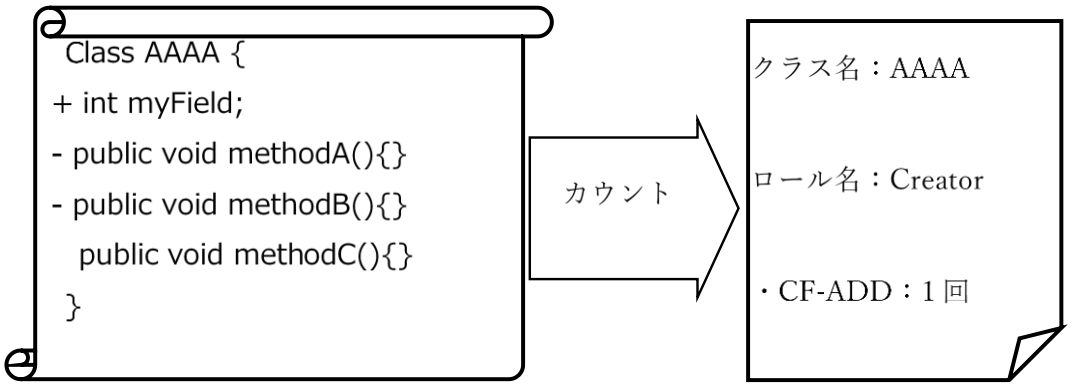
これらの作業により、手順②では「FactoryMethodパターン使用クラスのバグ修正コミットの修正内容」を取得する。

手順③：バグ修正内容の分析

手順③の目的は、手順②で取得したFactoryMethodパターン使用クラスのバグ修正内容を分類し、その数を数える事である。

バグ修正内容の分類には、2.3節で記述したKaiらのバグ修正パターン[9]を用いて分類する。バグ修正パターンのカウント方法を図 5に示す。図5の(a)のように、FactoryMethodパターン使用クラスの1回のバグ修正コミットで、メソッドの削除（MD-RMV）を2か所で実施し、クラス変数の追加（CF-ADD）を1か所で実施している場合は、MD-RMVを2回、CF-ADDを1回実施したとして数える。

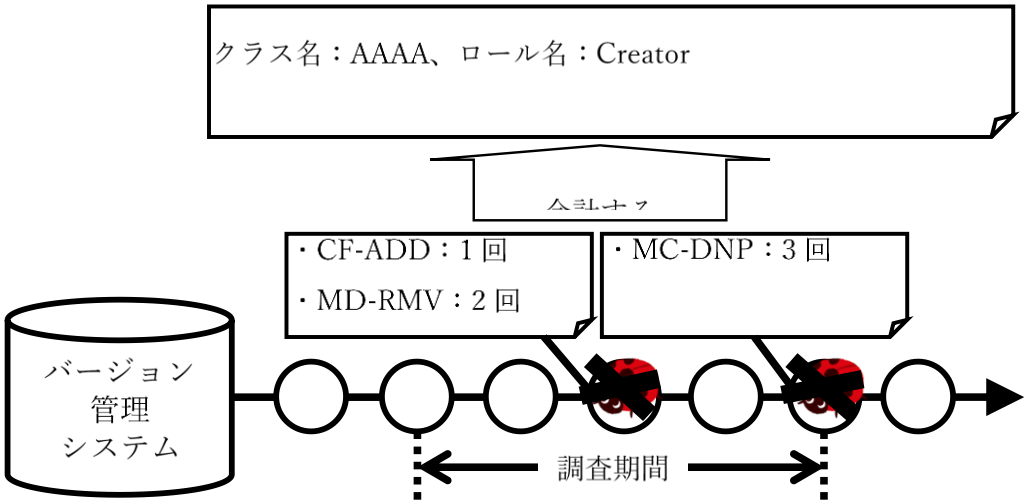
さらに、図 5の(b)のようにFactoryMethodパターン使用クラスの各バグ修正コミットのバグ修正を合計したものが、そのFactoryMethodパターン使用クラスのバグ修正内容とその数とする。



FactoryMethod パターンの
Creator を担うソースコードの
バグ修正内容

FactoryMethod パターンの
Creator を担うソースコードの
バグ修正内容のカウント結果

(a) 1 つのコミットの 1 つのクラスにおけるバグ修正内容の
カウント方法



(b) 1 つのクラスにおけるバグ修正内容のカウント方法

図 5　バグ修正パターンのカウント方法

4. 調査の結果

3章で記述した調査手順を基に調査した結果をRQと対応づけて記述する。なお、調査結果の詳細は、付録にまとめる。4.1節でRQに対応する調査の結果を示し、4.2節で調査結果に対する考察を述べる。

4.1 調査結果

RQ：各デザインパターンの各ロールにおいて、どのような種類のバグ修正がどれだけ実施されているか。

調査手順①の結果を付録Aに記載する。付録Aのとおり、PMDとJHotDraw共にFactoryMethodパターンのインスタンスは2つ存在した。また、各インスタンスに含まれるクラスの数は、PMDのインスタンス①、②が、それぞれ、6クラス、7クラスである事に対し、JHotDrawでのインスタンス①、②は、それぞれ、13クラス、34クラスであり、JHotDrawの方がFactoryMethodパターンに関与しているクラスが多い事が分かる。

調査手順②の結果を付録B－1、B－2、B－3、B－4に記載する。付録B－1、B－2、B－3、B－4には、FactoryMethodパターンに関するクラス名、そのクラスを調査した期間（以下、調査期間）、調査期間におけるコミット総数、調査期間における1カ月当たりの平均コミット数、そして調査期間においてコミットコメントに“bug”や“fix”が含まれているコミット総数を記述している。調査期間の始まりは「クラスが作成された日」または「FactoryMethodパターンの適用が確認できた日」を調査期間の始まりとした。調査期間の終わりは「クラスが削除された日」か「FactoryMethodパターンの適用が確認できなくなった日」か「最終コミット日」を調査期間の終わりとした。

付録B－1にあるように、TargetJDKVersionとJavaParser以外のクラスは約97カ月の間に10回コミットされている。TargetJDKVersionは49カ月の間に8回コミットされている。JavaParserは97カ月の間に115回コミットされている。つまり、PMDのインスタンス①ではJavaParserが最もコミット回数が多い。

付録B－2にあるように、BasicScopeFactory以外のクラスは約27カ月の間に約20コミットされている。BasicScopeFactoryは18カ月の間に10コミットされている。

付録B－3にあるように、調査期間においてコミットコメントに”bug”や”fix”が含まれるコミットの存在しているクラスはFigure、PolyLineFigure、RoundRectangleFigure、EllipseFigure、PolygonFigure、AbstractFigureの6クラスである。

付録B－4においても、調査期間においてコミットコメントに”bug”や”fix”が含まれるコミットの存在しているクラスはHandle、Figure、ChangeConnectionStartHandle、ChangeConnectionEndHandle、PolyLineFigure、RoundRectangleFigure、EllipseFigure、BorderDecorator、GroupFigure、TextFigure、NodeFigure、LineConnection、NumberTextFigure、StandardDrawingの14クラスである。また、TextFigureとStandardDrawingはコミットコメントに”bug”や”fix”が含まれるコミット数が3回であることに対し、その他のクラスは1回のコミット数である。

調査手順③の結果の一部を表 5、表 6 に記載する。表 5 は PMD でのバグ修正パターンの調査結果を示し、表 6 は JHotDraw でのバグ修正パターンの調査結果を示している。表 5、表 6 は、バグ修正パターンのカテゴリ、バグ修正の操作内容、FactoryMethod パターンのロールおける

バグ修正パターンの実施回数、そして各バグ修正パターンの実施回数の合計を記述している。

また、調査手順③結果の詳細を付録Ｃ－１、C－２、C－３、C－４、C―５、C―６、C－７、D－１、D－２、D－３、D－４、D―５、E－１、E－２、E－３、F－１、F－２、F－３、F－４、F―５、F―６、F－７に記載する。

付録Ｃ－１、D－１、E－１、E－１は、バグ修正のカテゴリ等について表 5、表 6と同じ項目を記述している。

付録Ｃ－２、D－２、E－２、E－２は、FactoryMethodパターンインスタンスごとのバグ修正日、バグ修正を実施しているクラス名とバグ修正パターンを記述している。

付録Ｃ－２、D－２、E－２、E－２以降の、付録 C－３や付録 E－５等は、バグ修正したコミット日時、コミット ID、そのコミットで修正されているファイルパス、そのコミットで FactoryMethod パターンを適用しているファイルの修正内容を記述している。

表 5　PMD におけるバグ修正パターンの調査結果

カテゴリ	操作内容	FactoryMethodパターンの各ロールにおけるバグ修正の実施回数			バグ修正パターン実施回数の合計
		Creator	Concrete Creator	Concrete Product	
CF	ADD			2	2(7%)
IF	APCJ			1	1(4%)
MC	DNP		6	2	8(30%)
MD	CHG			2	2(7%)
	ADD			1	1(4%)
	RMV			2	2(7%)
SQ	AFO			3	3(11%)
	AMO	1		6	7(26%)
	RMO			1	1(4%)

表 6　JHotDraw におけるバグ修正パターンの調査結果

カテゴリ	操作内容	FactoryMethodパターンの各ロールにおけるバグ修正の実施回数		バグ修正パターン実施回数の合計
		Concrete Creator	Concrete Product	
MC	DAP	4		4(33%)
	DM	1		1(8%)
	DNP	1		1(8%)
MD	ADD	2	3	5(42%)
SQ	AMO	1		1(8%)

4.2 考察

4.2.1　バグ修正パターンの傾向について

表 5 と表 6 と Kai らバグ修正パターン調査結果のうち、バグ修正実施率の高いバグ修正パターン上位 3 つを表 7 としてまとめる。表 7 よると、PMD において実施回数が多いバグ修正パターン上位 3 パターンは MC-DNP（8 回）、SQ-AMO（7 回）、SQ-AFO（3 回）となる。また、JHotDraw に実施回数が多いバグ修正パターン上位 3 パターンは MD-ADD（5 回）、MC-DAP（4 回）、MC-DM、MC-DNP、SQ-AMO（それぞれ 1 回）となる。一方、Kai らの調査結果では、7 つの OSS においてバグ修正が多いものは、MC-DAP（14.9-25.5%）、IF-CC（5.6-18.6%）、AS-CE（6.0-14.2%）として挙げられている。

この結果において、Kaiらの調査結果と比較すると、本研究の調査におけるPMDとJHotDrawではIF-CCとAS-CEに関するバグ修正パターンが見られなかった。IF-CCとは、if文の条件式を変更するバグ修正である。例えば、 if(var == 1) を if(var <= 1) に変更するようなバグ修正である。また、AS-CEとは、代入式の代入する値を変更するバグ修正である。例えば、 var = "woo" を var="spring" に変更するようなバグ修正である。

IF-CCとAS-CEが見られなかった共通の要因としては、調査データが少ない事とコミットコメントに”bug”または”fix”が含まれるコミットからバグ修正コミットではないと判断して調査対象から除外したものにIF-CCやAS-CEのバグ修正が含まれていた可能性等の影響が考えられる。例えば、PMDにおいては、コミットコメントに”bug”または”fix”が含まれるコミットからバグ修正コミットではないと判断して調査対象から除外したコミットにおいて、一回のコミットで変更箇所が1000行を超える場合等は、リファクタリングや機能追加も実施している可能性がある事から、調査対象から除外した。ただし、調査対象から除外したコミットにおいて、AS-CEを実施しているコミットも確認できた。

したがって、修正箇所が1000行を超えるような修正に対して、バグ修正だと確証が得られる場合は、Kaiらが実施したようなバグ混入パターンの傾向がみられる可能性がある。ここで、IF-CCとAS-CEをKaiらが調査した「一般のOSSにおけるバグ修正パターンの傾向」であると仮定すると、FactoryMethodパターンでは一般のOSSのバグ修正パターンと異なる傾向である可能性が考えられる。そこで、今後は「FactoryMethodパターンは『一般のOSSにおけるバグ修正パターンの傾向』と傾向が異なる」と仮定

を立て、他の OSS 等を対象に調査する事で新たな知見が得られる可能性が考えられる。

表 7　PMD、JHotDraw、Kai らのバグ修正パターン調査結果のうち、実施頻度が高いバグ修正パターン上位 3 パターン

PMD の FactoryMethod パターン使用クラスにおけるバグ修正パターン調査結果	JHotDraw の FactoryMethod パターン使用クラスにおけるバグ修正パターン調査結果	Kai らの 7 つの OSS におけるバグ修正パターン調査結果
MC-DNP（8 回）	MD-ADD（5 回）	MC-DAP（14.9-25.5% ）
SQ-AMO（7 回）	MC-DAP（4 回）	IF-CC（5.6-18.6% ）
SQ-AFO（3 回）	MC-DM、MC-DNP、SQ-AMO（1 回）	AS-CE（6.0-14.2% ）

4.2.2　バグ修正パターンのカテゴリの傾向について

表 5 と表 6 より、カテゴリ別でバグ修正パターンの傾向を上位 3 カテゴリをあげると、PMD は SQ（11 回のバグ修正実施）、MC（8 回のバグ修正実施）、MD（5 回のバグ修正実施）がバグ修正カテゴリとして多い。JHotDraw は MC（6 回のバグ修正実施）、MD（5 回のバグ修正実施）、SQ（1 回のバグ修正実施）がバグ修正カテゴリとして多い。一方、Kai らの 7 つの OSS では、MC（21.9-33.1%）、IF（19.7-33.9%）の２つのバグ修正カテゴリが多いと記述されていた。

このことから、バグ修正カテゴリ MC は本研究の調査結果である PMD と JHotDraw、そして Kai らの調査結果と同様によく見られるバグ修正カテゴリであると考えられる。

このような結果になった理由としては、一番大きな要因はサンプル数の少なさが影響していると考えられる。例えば、PMD における MC の割合は 30%であるが、バグ修正の数としては 8 個であり、1 回のコミットで MC のバグ修正が 1 箇所以上実施されている事もある。したがって、サンプル数が少ないために MC の値が割合として多くなった可能性が考えられる。

一方、Kai らの調査結果ではよくみられるバグ修正カテゴリとして IF のカテゴリが多かったことを示しているが、本研究で調査した結果では、IF に関するバグ修正が PMD で 1 つしか見つけられなかった。

この他、SQ の数については、PMD が 11 回、JHotDraw が 1 回と異なっている。この違いについては、PMD の場合はメソッド呼出しの追加をバグ修正として実施している回数が多くなったため、現れた違いなのだと考えられえる。ここで、PMD において SQ が増えたのは、FactoryMethod パターンの Product を担うクラスであり、1 クラスで 8000 行程ある。この行数の長さのため、コードのロジックを追いきれず、メソッドの追加を忘れてしまった可能性が考えられる。一方、JHotDraw には PMD の SQ を増加させたクラスのような 8000 行程のサイズの大きいクラスは存在しない。したがって、行数の長さが PMD と JHotDraw の SQ の数に影響を与えた可能性が考えられる。

今後は「FactoryMethod パターン適用クラスでは IF に関するバグ修正が少ない」という仮説を立て、条件文の使用率やソフトウェアのリリー

スバージョン間で比較すれば、新たな知見が得られる可能性が考えられる。

4.2.3 他クラスの MD-CHG による MC-DNP への影響

表 7より、本研究の調査における最も多いバグ修正パターンとしてMC-DNPが挙げられる。MC-DNPとは、メソッド呼出しにおいてメソッドへ与える実引数の数や型を変更するバグ修正である。例えば、funcA("woo") をfuncA("woo", "spring") に変更するようなバグ修正である。

このようなバグ修正を実施する理由としては、呼び出していたメソッドを間違えていたか、呼び出していたメソッドのシグネチャの変更が影響したかの２つが考えられる。

付録C－3より、今回の調査で発見したPMDのMC-DNPは、他のクラスのMD-CHGが影響している事が分かった。具体的には、図 6に示すように、JavaParserTokenManager.javaのメソッドのシグネチャ変更（MD-CHG）の影響で、そのメソッドを呼び出していたJavaParser.javaにおいて呼び出しメソッドの実引数の修正（MC-DNP）を実施し、さらにJavaParser.javaのメソッドシグネチャ変更（MD-CHG）の影響で、そのメソッドを呼び出していたTargetJDK1_3.javaにおいて呼び出しメソッドの実引数の修正（MC-DNP）を実施している事を確認した。したがって、今回の調査においてはMC-DNPはバグ修正というよりも、他クラスのMD-CHGの影響を受けたリファクタリングであると考えられる。

加えて、注目する事は1か所のバグ修正が複数個所の変更に影響を与えている事である。図 6に示したように、１つのバグ修正が別の箇所の修正に影響を与える事や、複数のクラスで同じメソッドが呼び出されている場合、１つのバグ修正が複数個所の変更に影響する事が考えられる。この「１つのバグ修正が複数箇所へ影響する事」はPMDにおけるConcreteProductの1か所のMD-CHGが３つのConcreteCreatorへ影響し、結果的にMC-DNPを6か所で実施する事が確認できた。したがって、1か所のMD-CHGの実施が複数個所のMC-CHGにつながる可能性が考えられるため、開発時にはメソッドのAPI設計を実施する事でMD-CHGを減少させることで、保守作業の負担を減少させることができると考えられる。

また、調査したクラス数やバグ数が少ないため断定的な判断はできないが、このConcreteProductとConcreteCreator間でのMD-CHGによるMC-DNPが発生する関係はFactoryMethodパターンに起因するバグ修正である可能性がある。理由としては、FactoryMethodパターンはConcreteCreatorがConcreteProductのコンストラクタを呼び出し、ConcreteProductのインスタンス生成を実施するためである。仮に、ConcreteProductのコンストラクタでMD-CHGを実施するとConcreteCreatorでMC-DNPを実施する必要が出てくる。したがって、このConcreteProductとConcreteCreator間でのMD-CHGとMC-DNPの関係はFactoryMethodパターンに起因するバグ修正であると考えられる。そこで、FactoryMethodパターンにおいて「ConcreteProductのMD-CHGがConcreteCreatorのMC-DNPに影響を与えている」という仮定を立てて調査する事で、新たな知見が得られる可能性が考えられる。

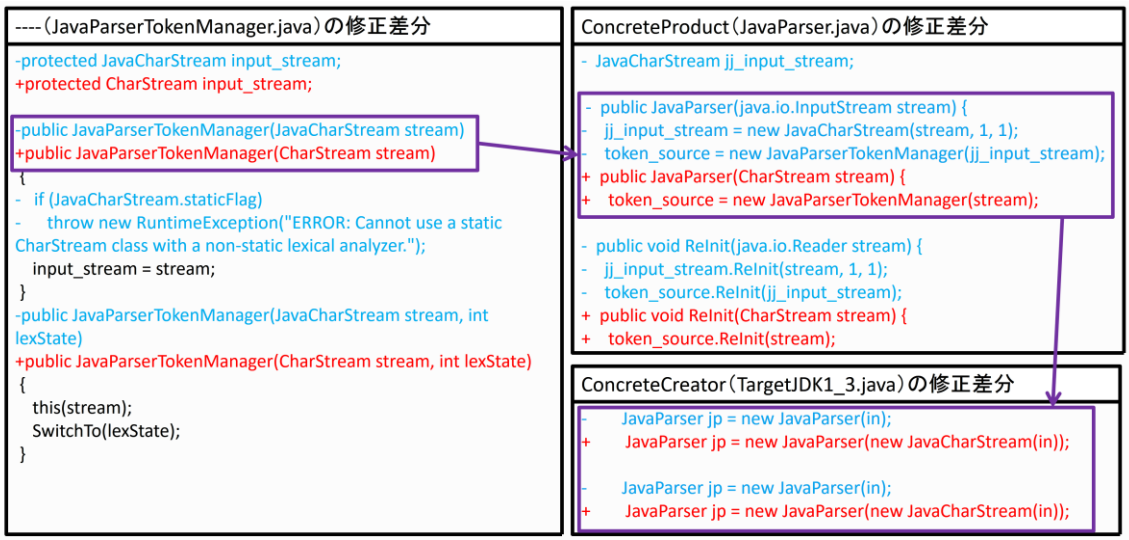


図 6　あるクラスの MD-CHG が別のクラスの MC-DNP に与える影響の例

4.2.4　メソッド呼出し忘れ（SQ-AMO）

表 7より、PMDで7回、JHotDrawで1回、SQ-AMOを確認できた。SQ-AMOはメソッド呼出し処理を追加するバグ修正である。このSQ-AMOを実施する理由として、メソッド呼び出し忘れが発生した時や、呼び出し忘れではないが新たに作成したメソッドを追加する時等にSQ-AMOを実施すると考えられる。このSQ-AMO実施の原因において、本研究で調査した全てのSQ-AMOは、メソッド呼出し忘れが原因で実施していると考えられる。

理由は、SQ-AMOを実施しているコミットにおいて、メソッド作成と作成したメソッドの追加（SQ-AMO）が実施されていないためである。例えば、1度のコミットにてメソッド作成と作成したメソッドの追加を実施している場合は、メソッドの追加忘れではなく、機能追加の可能性等が考えられる。しかし、本研究の調査結果におけるSQ-AMOでは、同じコミットで修正されているファイルの中にSQ-AMOとして追加したメソッドのクラスは確認できないものが8個の全てのSQ-AMOにて確認できた。

したがって、プロジェクト特有のバグ修正である可能性も考えられるが、FactoryMethodパターンでは「メソッドの追加忘れ」というバグのためにSQ-AMOが実施されやすい可能性が考えられる。そのため、「FactoryMethodパターン使用クラスでメソッド追加忘れが原因となったSQ-AMOが多い」という仮定を立てて調査する事で、新たな知見が得られる可能性が考えられる。

4.2.5 メソッドの引数チェック漏れ（IF-APCJ）

表 5と表 6より、PMDにて1か所のみ確認できたバグ修正パターンとしてIF-APCJが挙げられる。IF-APCJは何らかのアルゴリズム実施前に、ジャンプ文（returnやcontinue、break）を伴う条件判定IF文を加えるバグ修正である。IF-APCJを実施する理由として、条件判定処理漏れが考えられる。

付録D－3より、IF-APCJの内容が確認できるが、if文の条件文においてメソッド呼出しを行っている事が確認できる。呼び出されたメソッドは、IF-APCJと同じコミットで新しく定義されているメソッドである事から、メソッド呼出し忘れではないと考えられる。また、新しく作成したメソッドを利用してreturn文を伴うIF文を実施しており、IF-APCJを実施しているメソッドの引数をIF文の条件式に利用している事から、今回確認できたIF-APCJはメソッドの引数チェック漏れというバグであると考えられる。

4.2.6 値間違い（MC-DAP）

表 5、表 6 より、PMD で 1 か所、JHotDraw で 5 か所 MC-DAP の実施を確認した。MC-DAP はメソッド呼出しにおいて、メソッドへ渡すパラメータを変更するバグ修正である。MC-DAP を実施する理由として、渡す値を間違えていた事や呼び出しているメソッドの内部処理変更の影響が考えられる。

今回の調査結果では、MC-DAP を実施しているコミットと同じコミットで他クラスが MD-CHG を実施していない事を確認できた。したがって、今回の調査結果における MC-DAP はメソッドに渡す値を間違えていたために実施したバグ修正だと考えられる。

この MC-DAP について、Basil らのソフトウェアエラーの調査（Fortran でプログラミングされた衛星計画システム（約 90,000 行）が対象）では、インタフェースのバグ（正しいパラメータを与えない等）はよく見られるバグの 1 つであると述べている [13]。ここで、プログラミング言語や対象システム、開発環境等は異なるが、Basil らのソフトウェアエラーの調査結果を一般的なバグであると仮定すると、今回の MC-DAP は一般的なバグ修正の 1 つである可能性が考えられる。

4.2.7 呼出しメソッド間違い（MC-DM）

表 5、表 6 より、JHotDraw にて 1 か所のみ確認できたバグ修正パターンとして MC-DM が挙げられる。MC-DM は呼び出すメソッド自体を変更するバグ修正である。この MC-DM を実施する理由としては、呼び出していたメソッドを間違えているか他クラスの MD-CHG の影響などが考えられる。今回確認した MC-DM では、MC-DM を実施しているコミットと同じコミットにて、呼び出しているメソッドを実装しているクラスでの変更が無い事を確認した。

したがって、今回の MC-DM には他クラスの MD-CHG の影響はないと考えられるため、呼び出していたメソッドを間違えていたバグであると考えられる。

5. まとめと今後の課題

本研究では、デザインパターン適用ソースコードにおいて混入するバグの種類を明らかにするために、バグ修正の種類を調査した。従来研究がデザインパターン適用ソースコードにおけるバグ混入率を調査している事に対し、本研究はデザインパターン適用時のバグ修正の種類に着目した点で新規性があると考える。

本研究では、デザインパターンを適用したソースコードにおけるバグ修正の種類を調査するために、FactoryMethodパターンを使用しているOSSプロジェクト２つ（「PMD」と「JHotDraw」）を対象にバグ修正内容を調査した。調査したFactoryMethodパターンのインスタンスはPMDで２つ、JHotDrawで2つの計４つのFactoryMethodパターンのインスタンスを調査した。

調査の結果、得られたサンプル数が少ないため断定的な判断はできないものの、FactoryMethodパターンのバグ修正傾向は従来研究で示された結果と異なる可能性やFactoryMethodパターンに起因すると考えられるバグ修正（MC-DNP）等を見つけた。

今回の調査結果を基に、今後はFactory Methodパターンを適用していないソースコード等を対象に、同様のバグ修正が実施されているかの調査を今後の課題と考える。

具体的には、FactoryMethodパターンを対象にした調査サンプル数の増加やFactoryMethodパターン以外のデザインパターンを調査対象とした調査、デザインパターン未適用のソースコードにおける調査などである。

このために、ソースコードに対するデザインパターン適用未適用を判断する必要がある。その際に、NikolaosらやBahareh等が提案しているデ

ザインパターン検出ツールや手法 [14] [15]を利用する事でデザインパターンの適用や 1 つのクラスにデザインパターンが複数適用されていないかどうか等の判断効率が向上すると考えられる。

謝辞

本研究を進めるにあたり、多くの方々に御指導、御協力、御支援を頂きました。ここに謝意を添えて御名前を記させていただきます。

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学研究室　飯田 元 教授には、本研究において熱心な御指導を賜りました。研究方針だけではなく、研究に対する姿勢、論文執筆、発表方法についても多くの御助言を頂きました。心より厚くお礼申し上げます。

奈良先端科学技術大学院大学ソフトウェア工学研究室　松本 健一 教授には、本研究において貴重な御指導を賜りました。学内の発表において、多数のご質問、御指導を頂きました。先生から頂いたご指導は、研究を進めるうえで大変重要なものとなりました。心より厚くお礼申し上げます。

奈良先端科学技術大学院大学ソフトウェア設計学研究室　市川 昊平 准教授には、本研究において貴重な御指導を賜りました。また、研究活動や学生生活に関しましても多くの御助言を頂きました。心より厚くお礼申し上げます。

奈良先端科学技術大学院大学ソフトウェア設計学研究室　崔 恩瀞 助教には、本研究において熱心な御指導を賜りました。研究方針だけではなく、研究に対する姿勢、論文執筆、発表方法についても多くの御助言を頂きました。心より厚くお礼申し上げます。

I would like to have my warmest thanks to Professor. Putchong and Professor. Chantana for all the helpful advices and comments during my internship. I am also greatful thanks Kasetsart University students who participate in my experiment.

小川 暁子 様、森本 恭代 様、仲田 綾奈 様には、研究活動に必要な事務処理など、多岐にわたりご助力いただきました。心より厚くお礼申し上げます。

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学研究室の皆様には、日頃より多くの御協力、御助言を頂きました。また、研究だけではなく、日頃の生活において支えていただきました。心より厚くお礼を申し上げます。

最後に、大学院への進学にあたって、多くの支援を頂いた両親、祖母、姉に心より深く感謝申し上げます。

参考文献

1. **Erich GammaHelm, Ralph Johnson, John VlissidesRichard.** Design patterns: elements of reusable object-oriented software. : Addison-Wesley Longman Publishing, 1995.

2. **鷲崎 弘宜一憲，大杉 直樹，権藤 克彦，服部 哲，久保 淳人，小林 隆志，大月 美佳，丸山 勝久，榊原 彰坂本.** デザインパターンへのソフトウェア工学的取り組み.: コンピュータソフトウェア，岩波書店，Vol.29, No.1, pp.1_130-1_146, 2012.

3. **Jing DongSun , Yajing ZhaoYongtao.** Design Pattern Detection by Template Matching. : Proceedings of the 2008 ACM symposium on Applied computing, March 16-20, 2008.

4. **MayvanRasoolzadegan, A., Yazdi, Z.GB.B.,.** The state of the art on design patterns: A systematic mapping of the literature. : Journal of Systems and Software. Volume 125, March 2017, Pages 93-118, 2017.

5. **Mahmoud O. ElishA. MohammedMawal.** Quantitative analysis of fault density in design patterns: An empirical study. : Journal Information and Software Technology Volume 66 (2015) pp. 58-72., 2015.

6. **PLoPD Editors.** プログラムデザインのためのパターン言語: Pattern Language of Program Design 選集.: ソフトバンクパブリッシング，2001.

7. **A. AmpatzoglouKritikos, E.-M. Arvanitou, A. Gortzis, F. Chatziasimidis, I. StamelosA.** An empirical investigation on the impact of design pattern application on computer game defects.: in: Proceedings of the

15th International Academic MindTrek Conference: Envisioning Future Media Environments, Tampere, Finland, 2011.

8. **VokacM.** Defect frequency and design patterns: an empirical study of industrial code. 2004.

9. **Kai PanKim , E. James Whitehead, Jr.Sunghun.** Toward an understanding of bug fix patterns.: in Journal Empirical Software Engineering Volume 14 Issue 3, June, 2009.

10. **Yangyang ZhaoLeung , Yibiao Yang , Yuming Zhou , Baowen XuHareton.** Towards an understanding of change types in bug fixing code. : in journal Information and Software Technology, Volume 86, June 2017, Pages 37-53, 2017.

11. **Yungbum JungOh, Kwangkeun YiHakjoo.** Identifying static analysis techniques for finding non-fix hunks in fix revisions. : in Proceeding: DSMM '09 Proceedings of the ACM first international workshop on Data-intensive software management and mining, Pages 13-18, 2009.

12. **Sunghun KimZimmermann, Kai Pan, E. James Jr. WhiteheadThomas.** Automatic identification of bug-introducingchanges. : in Proceeding: ASE '06 Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, Pages 81-90, September 18 - 22, 2006.

13. **Victor R. BasiliT. PerriconeBarry.** Software errors and complexity: an empirical investigation. : Communications of the ACM, v.27 n.1, p.42-52, Jan., 1984.

14. **Nikolaos TsantalisChatzigeorgiou , George Stephanides , Spyros T. HalkidisAlexander.** Design Pattern Detection Using Similarity Scoring. 出版地不明 : IEEE Transactions on Software Engineering, v.32 n.11, p.896-909, November, 2006.

15. **Bahareh Bafandeh MayvanRasoolzadeganAbbas.** Design pattern detection based on the graph theory. 出版地不明 : in Journal: Knowledge-Based Systems archive Volume 120 Issue C, Pages 211-225, March, 2017.

付録Ａ．調査手順①の結果

OSS名	インスタンス	ロール	クラス
PMD	①	Creator	TargetJDKVersion
		Product	JavaParserConstants
		Concrete Creator	TargetJDK1_3、TargetJDK1_4、TargetJDK1_5
		Concrete Product	JavaParser
	②	Creator	ScopeFactory
		Product	Scope
		Concrete Creator	BasicScopeFactory
		Concrete Product	GlobalScope、ClassScope、MethodScope、LocalScope
JHotDraw	①	Creator	Figure
		Product	Connector
		Concrete Creator	PolyLineFigure、RoundRectangleFigure、EllipseFigure、PolygonFigure AbstractFigure
		Concrete Product	ChopBoxConnector、ShortestDistanceConnector、ChopPolygonConnector、ChopEllipseConnector、PolyLineConnector
	②	Creator	Figure
		Product	Handle

		Concrete Creator	PolyLineFigure、PolygonFigure、AnimationDecorator、BorderDecorator、BouncingDrawing、GroupFigure、LineFigure、PertFigure、TextFigure、NodeFigure、LineConnection、ElbowConnection、TriangleFigure、NumberTextFigure、PertDependency、StandardDrawing
		Concrete Product	PolygonScaleHandle、TriangleRotationHandle、ElbowHandle、PolyLineHandle、PolygonHandle、WestHandle、EastHandle、SouthHandle、NorthHandle、ChangeConnectionStartHandle、ChangeConnectionEndHandle、FontSizeHandle、ConnectionHandle、LocatorHandle、NullHandle、GroupHandle

付録Ｂ－１．調査手順②の結果（PMD のインスタンス①）

クラス名	調査期間	調査期間におけるコミット総数(回)	調査期間における1カ月当たりの平均コミット数(コミット数/月)	調査期間において、コミットコメントに"bug"または"fix"が含まれているコミット総数(回)
TargetJDKVersion	2003/9/4 - 2007/10/25	8	0.16	0
JavaParserConstants	2003/9/4 - 2011/10/14	12	0.12	4
TargetJDK1_3	2003/9/12- 2011/10/14	10	0.10	2
TargetJDK1_4	2003/9/4 - 2011/10/14	9	0.09	1
TargetJDK1_5	2004/3/26- 2011/10/14	7	0.08	1
JavaParser	2003/9/11- 2011/10/14	115	1.19	32

付録Ｂ－２．調査手順②の結果（PMD のインスタンス②）

クラス名	調査期間	調査期間におけるコミット総数(回)	調査期間における1カ月当たりの平均コミット数(コミット数/月)	調査期間において、コミットコメントに"bug"または"fix"が含まれているコミット総数(回)
ScopeFactory	2002/10/4-2005/1/11	19	0.70	2
Scope	2002/10/2-2005/1/11	23	0.85	2
BasicScopeFactory	2003/6/28-2005/1/11	10	0.56	0
GlobalScope	2002/10/4-2005/1/11	22	0.81	3
ClassScope	2002/10/4-2005/1/11	21	0.78	3
MethodScope	2002/10/10-2005/1/11	15	0.56	3
LocalScope	2002/10/2-2004/12/21	19	0.73	4

付録Ｂ－３．調査手順②の結果（JHotDraw のインスタンス①）

クラス名	調査期間	調査期間におけるコミット総数(回)	調査期間における1カ月当たりの平均コミット数(コミット数/月)	調査期間においてコミットコメントに"bug"または"fix"が含まれているコミット総数(回)
Connector	2000/10/12-2002/8/5	4	0.19	0
Figure	2000/10/12-2003/1/30	8	0.30	1
ChopBoxConnector	2000/10/12-2001/10/25	4	0.33	0
ShortestDistanceConnector	2000/10/12-2002/8/5	6	0.29	0
ChopPolygonConnector	2000/10/12-2001/10/25	3	0.25	0
ChopEllipseConnector	2000/10/12-2001/10/25	3	0.25	0
PolyLineConnector	2000/10/12-2001/8/5	4	0.19	0
PolyLineFigure	2000/10/12-2002/9/14	8	0.23	1
RoundRectangleFigure	2000/10/12-2003/5/25	9	0.29	1

EllipseFigure	2000/10/12-2003/5/25	6	0.19	1
PolygonFigure	2000/10/12-2002/8/5	4	0.19	0
AbstractFigure	2000/10/12-2003/2/4	11	0.41	1

付録Ｂ－４．調査手順②の結果（JHotDraw のインスタンス②）

クラス名	調査期間	調査期間におけるコミット総数(回)	調査期間における1カ月当たりの平均コミット数(コミット数/月)	調査期間において、コミットコメントに"bug"または"fix"が含まれているコミット総数(回)
Handle	2000/10/12-2003/5/9	6	0.20	1
Figure	2000/10/12-2003/1/30	8	0.30	1
PolygonScaleHandle	2000/10/12-2002/8/5	4	0.19	0
TriangleRotationHandle	2000/10/12-2002/8/5	4	0.19	0
ElbowHandle	2000/10/12-2002/5/1	5	0.28	0
PolyLineHandle	2000/10/12-2003/11/10	6	0.17	0
PolygonHandle	2000/10/12-2002/8/5	4	0.19	0
ChangeConnectionStartHandle	2000/10/12-2003/11/10	8	0.22	1

ChangeConnectionEndHandle	2000/10/12-2003/11/10	9	0.25	1
FontSizeHandle	2000/10/12-2002/8/5	5	0.24	0
ConnectionHandle	2000/10/12-2003/5/9	6	0.20	0
NullHandle	2000/10/12-2001/10/25	3	0.25	0
GroupHandle	2000/10/12-2001/10/25	3	0.25	0
PolyLineFigure	2000/10/12-2003/9/14	8	0.23	1
RoundRectangleFigure	2000/10/12-2003/5/25	9	0.29	1
EllipseFigure	2000/10/12-2003/5/25	6	0.19	1
PolygonFigure	2000/10/12-2002/8/5	4	0.19	0
AnimationDecorator	2000/10/12-2003/9/23	6	0.17	0
BorderDecorator	2000/10/12-2003/9/23	6	0.26	1
BouncingDrawing	2000/10/12-2002/9/7	6	0.27	0
GroupFigure	2000/10/12-2003/2/23	9	0.32	1
LineFigure	2000/10/12-2002/8/5	4	0.19	0

PertFigure	2000/10/12-2003/11/10	6	0.17	0
TextFigure	2000/10/12-2003/10/6	29	0.83	3
NodeFigure	2000/10/12-2003/1/30	5	0.19	1
LineConnection	2000/10/12-2003/5/20	8	0.26	1
ElbowConnection	2000/10/12-2003/2/4	5	0.19	0
TriangleFigure	2000/10/12-2003/2/4	5	0.19	0
NumberTextFigure	2000/10/12-2003/1/30	6	0.22	1
PertDependency	2000/10/12-2003/9/14	7	0.20	0
StandardDrawing	2000/10/12-2003/1/30	15	0.56	3

付録 C－１．インスタンス別でのバグ修正パターンの実施回数（PMD のインスタンス①）

カテゴリ	操作内容	FactoryMethodパターンの各ロールにおけるバグ修正の実施回数		バグ修正パターン実施回数の合計
		Concrete Creator	Concrete Product	
CF	ADD		1	1(5%)
MC	DNP	6	2	8(36%)
MD	CHG		2	2(9%)
	RMV		2	2(9%)
SQ	AFO		2	2(9%)
	AMO		6	6(27%)
	RMV		1	1(5%)

付録 C－２．インスタンス別でのバグ修正実施コミットの概要（PMD のインスタンス①）

バグ修正日	修正しているFactoryMethodパターン使用クラスと修正内容
2004-04-14 000049	TargetJDK1_3.java [public JavaParser createParser(InputStream in)] 【MC-DNP】 [public JavaParser createParser(Reader in)] 【MC-DNP】 TargetJDK1_4.java [public JavaParser createParser(InputStream in)] 【MC-DNP】 [public JavaParser createParser(Reader in)] 【MC-DNP】 TargetJDK1_5.java [public JavaParser createParser(InputStream in)] 【MC-DNP】 [public JavaParser createParser(Reader in)] 【MC-DNP】 JavaParser.java 【SQ-AFO】 [JavaParser(java.io.InputStream stream)->JavaParser(CharStream stream)] 【MC-DNP】【MD-CHG】 [JavaParser(java.io.Reader stream)]

	【MD-RMV】 [ReInit(java.io.InputStream stream)] 【MD-RMV】 [ReInit(java.io.Reader stream)] 【SQ-RMO】【MC-DNP】【MD-CHG】
2005-04-23 095307	JavaParser [final public void AnnotationTypeDeclaration(int modifiers) throws ParseException] 【SQ-AMO】【SQ-AFO】
2005-04-28 070240	JavaParser [final public void ShiftExpression() throws ParseException] 【SQ-AMO】【SQ-AMO】【SQ-AMO】【SQ-AMO】
2005-05-05 070406	JavaParser [final public void AnnotationTypeDeclaration(int modifiers) throws ParseException] 【SQ-AMO】
2006-03-09 235634	JavaParser [final public void Type() throws ParseException] 【CF-ADD】

付録 C－３．インスタンス別でのバグ修正コミットの内容①（PMD のインスタンス①）

コミット日時	2004-04-14 00:00:49
コミット ID	ce980c596221820c753377bbe8c2733b39bf3d0f
コミットコメント	Fixed bug 923410 - PMD now uses the default platform character set encoding; optionally, you can pass in a character encoding to use.

このコミットで修正されているファイル	pmd/etc/build.xml pmd/etc/changelog.txt pmd/etc/grammar/Java1.4-c.jjt pmd/regress/test/net/sourceforge/pmd/CommandLineOptionsTest.java pmd/regress/test/net/sourceforge/pmd/ast/EncodingTest.java pmd/regress/test/net/sourceforge/pmd/symboltable/AcceptanceTest.java pmd/src/net/sourceforge/pmd/CommandLineOptions.java pmd/src/net/sourceforge/pmd/PMD.java pmd/src/net/sourceforge/pmd/TargetJDK1_3.java pmd/src/net/sourceforge/pmd/TargetJDK1_4.java pmd/src/net/sourceforge/pmd/TargetJDK1_5.java pmd/src/net/sourceforge/pmd/ant/PMDTask.java pmd/src/net/sourceforge/pmd/ast/CharStream.java pmd/src/net/sourceforge/pmd/ast/JavaCharStream.java pmd/src/net/sourceforge/pmd/ast/JavaParser.java pmd/src/net/sourceforge/pmd/ast/JavaParserTokenManager.java pmd/src/net/sourceforge/pmd/ast/SimpleCharStream.java pmd/xdocs/ant-task.xml pmd/xdocs/credits.xml pmd/xdocs/running.xml
このコミットにおいて、FactoryMethodパターンを使用しているファイルの修正内容	ファイルパス：pmd/src/net/sourceforge/pmd/TargetJDK1_3.java デザインパターンでの役割：Factory Method パターンの ConcreateCreator 変更内容： 【MC-DNP】 【MC-DNP】

	変更内容の元（diff）：
	@@ -3,6 +3,7 @@
	 */
	 package net.sourceforge.pmd;
	
	+import net.sourceforge.pmd.ast.JavaCharStream;
	 import net.sourceforge.pmd.ast.JavaParser;
	
	 import java.io.InputStream;
	@@ -11,13 +12,13 @@
	 public class TargetJDK1_3 implements TargetJDKVersion {
	
	 public JavaParser createParser(InputStream in) {
	- JavaParser jp = new JavaParser(in);
	+ JavaParser jp = new JavaParser(new JavaCharStream(in));
	 jp.setAssertAsIdentifier();
	 return jp;
	 }
	
	 public JavaParser createParser(Reader in) {
	- JavaParser jp = new JavaParser(in);
	+ JavaParser jp = new JavaParser(new JavaCharStream(in));
	 jp.setAssertAsIdentifier();
	 return jp;
	 }

	ファイルパス：pmd/src/net/sourceforge/pmd/TargetJDK1_4.java
	デザインパターンでの役割：Factory Method パターンの ConcreateCreator
	変更内容：
	【MC-DNP】
	【MC-DNP】
	変更内容の元（diff）：
	@@ -13,11 +13,11 @@
	public class TargetJDK1_4 implements TargetJDKVersion {
	public JavaParser createParser(InputStream in) {
	- return new JavaParser(in);
	+ return new JavaParser(new JavaCharStream(in));
	}
	public JavaParser createParser(Reader in) {
	- return new JavaParser(in);
	+ return new JavaParser(new JavaCharStream(in));
	}
	public JavaParserTokenManager createJavaParserTokenManager(Reader in) {

	ファイルパス：pmd/src/net/sourceforge/pmd/TargetJDK1_5.java

	デザインパターンでの役割：Factory Method パターンの ConcreateCreator
	変更内容：
	【MC-DNP】
	【MC-DNP】
	変更内容の元（diff）：
	@@ -1,5 +1,6 @@
	package net.sourceforge.pmd;
	
	+import net.sourceforge.pmd.ast.JavaCharStream;
	import net.sourceforge.pmd.ast.JavaParser;
	
	import java.io.InputStream;
	@@ -8,13 +9,13 @@
	public class TargetJDK1_5 implements TargetJDKVersion {
	
	public JavaParser createParser(InputStream in) {
	- JavaParser jp = new JavaParser(in);
	+ JavaParser jp = new JavaParser(new JavaCharStream(in));
	jp.setJDK15();
	return jp;
	}
	
	public JavaParser createParser(Reader in) {
	- JavaParser jp = new JavaParser(in);
	+ JavaParser jp = new JavaParser(new JavaCharStream(in));

	jp.setJDK15();
	return jp;
	}
	
	--- 	
	ファイルパス：pmd/src/net/sourceforge/pmd/ast/JavaParser.java
	デザインパターンでの役割：Factory Method パターンの ConcreateProduct
	変更内容：
	【SQ-AFO】
	・コンストラクタ：
	[JavaParser(java.io.InputStream stream)]
	【MD-CHG】
	【MC-DNP】
	[JavaParser(java.io.Reader stream)]
	【MD-RMV】
	・メソッド：
	[メソッド ReInit(java.io.InputStream stream)]自体を削除
	【MD-RMV】
	[メソッド ReInit(java.io.Reader stream)]
	【MD-CHG】
	【SQ-RMO】
	【MC-DNP】
	変更内容の元（diff）：
	@@ -7654,7 +7654,6 @@
	}

```
    public JavaParserTokenManager token_source;
-   JavaCharStream jj_input_stream;
    public Token token, jj_nt;
    private int jj_ntk;
    private Token jj_scanpos, jj_lastpos;
@@ -7689,9 +7688,8 @@
    private boolean jj_rescan = false;
    private int jj_gc = 0;

-   public JavaParser(java.io.InputStream stream) {
-     jj_input_stream = new JavaCharStream(stream, 1, 1);
-     token_source = new JavaParserTokenManager(jj_input_stream);
+   public JavaParser(CharStream stream) {
+     token_source = new JavaParserTokenManager(stream);
      token = new Token();
      jj_ntk = -1;
      jj_gen = 0;
@@ -7699,30 +7697,8 @@
      for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
    }

-   public void ReInit(java.io.InputStream stream) {
-     jj_input_stream.ReInit(stream, 1, 1);
-     token_source.ReInit(jj_input_stream);
-     token = new Token();
```

```
-     jj_ntk = -1;
-     jjtree.reset();
-     jj_gen = 0;
-     for (int i = 0; i < 117; i++) jj_la1[i] = -1;
-     for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
-   }
-
-   public JavaParser(java.io.Reader stream) {
-     jj_input_stream = new JavaCharStream(stream, 1, 1);
-     token_source = new JavaParserTokenManager(jj_input_stream);
-     token = new Token();
-     jj_ntk = -1;
-     jj_gen = 0;
-     for (int i = 0; i < 117; i++) jj_la1[i] = -1;
-     for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
-   }
-
-   public void ReInit(java.io.Reader stream) {
-     jj_input_stream.ReInit(stream, 1, 1);
-     token_source.ReInit(jj_input_stream);
+    public void ReInit(CharStream stream) {
+      token_source.ReInit(stream);
       token = new Token();
       jj_ntk = -1;
       jjtree.reset();
```

付録 C－４．インスタンス別でのバグ修正コミットの内容②（PMD のインスタンス①）

コミット日時	2005-04-23 09:53:07
コミット ID	093c3b5dead95f80f43b533ce30706fa6fbe516d
コミットコメント	Fixed bug 1188372 - AtLeastOneConstructor no longer fires on interfaces. Also, twiddling grammar to try to figure out annotations bug
このコミットで修正されているファイル	pmd/etc/grammar/Java.jjt pmd/regress/test/net/sourceforge/pmd/rules/AtLeastOneConstructorRuleTest.java pmd/regress/test/net/sourceforge/pmd/rules/UnusedImportsRuleTest.java pmd/rulesets/controversial.xml pmd/src/net/sourceforge/pmd/ast/ASTAnnotationTypeDeclaration.java pmd/xdocs/credits.xml
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス：pmd/src/net/sourceforge/pmd/ast/JavaParser.java デザインパターンでの役割：FactoryMethod パターンの ConcreatProduct 変更内容： ・メソッド： [final public void AnnotationTypeDeclaration(int modifiers) throws ParseException] 【SQ-AMO】 【SQ-AFO】 変更内容の元（diff）： @@ -5397,13 +5397,14 @@ /* Annotation Types. */ final public void AnnotationTypeDeclaration(int modifiers) throws ParseException { /*@bgen(jjtree) AnnotationTypeDeclaration */ - ASTAnnotationTypeDeclaration jjtn000 = new ASTAnnotationTypeDeclaration(this, JJTANNOTATIONTYPEDECLARATION); - boolean jjtc000 = true;

```
-   jjtree.openNodeScope(jjtn000);
+ASTAnnotationTypeDeclaration jjtn000 = new
ASTAnnotationTypeDeclaration(this,
JJTANNOTATIONTYPEDECLARATION);
+boolean jjtc000 = true;
+jjtree.openNodeScope(jjtn000);Token t;
      try {
        jj_consume_token(AT);
        jj_consume_token(INTERFACE);
-       jj_consume_token(IDENTIFIER);
+        t = jj_consume_token(IDENTIFIER);
+                                          jjtn000.setImage(t.image);
        AnnotationTypeBody();
      } catch (Throwable jjte000) {
      if (jjtc000) {
@@ -6692,6 +6693,12 @@
      return false;
   }


+   final private boolean jj_3_48() {
+     if (jj_scan_token(IDENTIFIER)) return true;
+     if (jj_scan_token(IDENTIFIER)) return true;
+     return false;
+   }
+
   final private boolean jj_3R_152() {
     if (jj_scan_token(LBRACE)) return true;
     Token xsp;
@@ -6709,15 +6716,16 @@
     return false;
   }
```

```
-   final private boolean jj_3_48() {
-      if (jj_scan_token(IDENTIFIER)) return true;
-      if (jj_scan_token(IDENTIFIER)) return true;
-      return false;
-   }
-
    final private boolean jj_3R_261() {
       if (jj_scan_token(COMMA)) return true;
       if (jj_3R_260()) return true;
+      return false;
+   }
+
+   final private boolean jj_3_47() {
+      if (jj_3R_63()) return true;
+      if (jj_scan_token(IDENTIFIER)) return true;
+      if (jj_scan_token(LPAREN)) return true;
       return false;
    }

@@ -6734,13 +6742,6 @@
    final private boolean jj_3R_64() {
       if (jj_scan_token(LBRACKET)) return true;
       if (jj_scan_token(RBRACKET)) return true;
-      return false;
-   }
-
-   final private boolean jj_3_47() {
-      if (jj_3R_63()) return true;
-      if (jj_scan_token(IDENTIFIER)) return true;
-      if (jj_scan_token(LPAREN)) return true;
```

	return false; } ---

付録 C－５．インスタンス別でのバグ修正コミットの内容③（PMD のインスタンス①）

コミット日時	2005-04-28 07:02:40
コミット ID	a104045fd79f85e90a2a8c5c640a3e0f7538faa2
コミットコメント	Fixed bug 1190461 - UnusedLocal no longer misses usages which are on the RHS of a right bit shift operator.
このコミットで修正されているファイル	pmd/etc/grammar/Java.jjt pmd/regress/test/net/sourceforge/pmd/rules/UnusedLocalVariableTest.java pmd/src/net/sourceforge/pmd/ast/ASTInitializer.java
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス：pmd/src/net/sourceforge/pmd/ast/JavaParser.java デザインパターンでの役割： 変更内容： [final public void ShiftExpression() throws ParseException] 【SQ-AMO】 【SQ-AMO】 【SQ-AMO】 【SQ-AMO】 ・メソッド内部処理 ・追加：RSIGNEDSHIFT();実行後に、他クラスのメソッド呼出しの追加。（jjtn000.setImage("<<");jjtn000.setUnDiscardable();） ・追加：RUNSIGNEDSHIFT();実行後に、他クラスのメソッド呼出しの追加。（jjtn000.setImage("<<");jjtn000.setUnDiscardable();） 変更内容の元（diff）： @@ -2777,8 +2777,10 @@ jj_la1[69] = jj_gen;

	if (jj_2_20(1)) {
	RSIGNEDSHIFT();
	+
	jjtn000.setImage("<<");jjtn000.setUnDiscardable();
	} else if (jj_2_21(1)) {
	RUNSIGNEDSHIFT();
	+
	jjtn000.setImage("<<");jjtn000.setUnDiscardable();
	} else {
	jj_consume_token(-1);
	throw new ParseException();

付録 C－６．インスタンス別でのバグ修正コミットの内容④（PMD のインスタンス①）

コミット日時	2005-05-05 07:04:06
コミット ID	a25567f5b2efbd3d83953f9893631f45373f5340
コミットコメント	Fixed bug 1187325 - UnusedImports no longer reports a false positive on imports which are used inside an Annotation.
このコミットで修正されているファイル	pmd/bin/build.xml pmd/etc/changelog.txt pmd/etc/grammar/Java.jjt pmd/regress/test/net/sourceforge/pmd/ast/DiscardableNodeCleanerTest.java pmd/regress/test/net/sourceforge/pmd/rules/UnusedImportsRuleTest.java pmd/regress/test/net/sourceforge/pmd/rules/UnusedPrivateMethodRuleTest.java pmd/regress/test/net/sourceforge/pmd/rules/javabeans/MissingSerialVersionUIDTest.java pmd/src/net/sourceforge/pmd/ast/ASTAnnotationTypeDeclaration.java pmd/src/net/sourceforge/pmd/ast/ASTModifiers.java pmd/src/net/sourceforge/pmd/ast/DiscardableNodeCleaner.java pmd/src/net/sourceforge/pmd/rules/UnusedPrivateMethodRule.java

	pmd/xdocs/credits.xml

このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス：pmd/src/net/sourceforge/pmd/ast/JavaParser.java
	デザインパターンでの役割：
	変更内容：
	・メソッド：
	[final public void AnnotationTypeDeclaration(int modifiers) throws ParseException]
	【SQ-AMO】
	・クラス内部処理：
	・追加：他クラスのメソッド呼出しの追加（jjtn000.setModifiers(modifiers);）
	
	変更内容の元（diff）：
	@@ -5402,6 +5402,7 @@
	ASTAnnotationTypeDeclaration jjtn000 = new ASTAnnotationTypeDeclaration(this, JJTANNOTATIONTYPEDECLARATION);
	boolean jjtc000 = true;
	jjtree.openNodeScope(jjtn000);Token t;
	+jjtn000.setModifiers(modifiers);
	try {
	jj_consume_token(AT)
	jj_consume_token(INTERFACE);

付録 C－７．インスタンス別でのバグ修正コミットの内容⑤（PMD のインスタンス①）

コミット日時	2006-03-09 23:56:34
コミット ID	04ab35e4ab798ef548e76adbea901c7ea75b12f2
コミットコメント	Fixed bug 1445765 - PMD no longer uses huge amounts of memory. However, you need to use RuleViolation.getBeginLine(); RuleViolation.getNode() is no more.
このコミットで修正されて	pmd/etc/grammar/Java.jjt

いるファイル	pmd/regress/test/net/sourceforge/pmd/AbstractRuleTest.java pmd/regress/test/net/sourceforge/pmd/RuleViolationTest.java pmd/regress/test/net/sourceforge/pmd/jsp/ast/XPathJspRuleTest.java pmd/regress/test/net/sourceforge/pmd/rules/XPathRuleTest.java pmd/src/net/sourceforge/pmd/Report.java pmd/src/net/sourceforge/pmd/RuleViolation.java pmd/src/net/sourceforge/pmd/dfa/report/ReportHTMLPrintVisitor.java pmd/src/net/sourceforge/pmd/dfa/report/ViolationNode.java pmd/src/net/sourceforge/pmd/renderers/CSVRenderer.java pmd/src/net/sourceforge/pmd/renderers/EmacsRenderer.java pmd/src/net/sourceforge/pmd/renderers/HTMLRenderer.java pmd/src/net/sourceforge/pmd/renderers/IDEAJRenderer.java pmd/src/net/sourceforge/pmd/renderers/PapariTextRenderer.java pmd/src/net/sourceforge/pmd/renderers/TextPadRenderer.java pmd/src/net/sourceforge/pmd/renderers/TextRenderer.java pmd/src/net/sourceforge/pmd/renderers/VBHTMLRenderer.java pmd/src/net/sourceforge/pmd/renderers/XMLRenderer.java
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス：pmd/src/net/sourceforge/pmd/ast/JavaParser.java デザインパターンでの役割： 変更内容： ・クラス変数： 【CF-ADD】 ・追加：Token 型のクラス変数の追加（Token t;） 変更内容の元（diff）： @@ -1776,9 +1776,9 @@ */ final public void Type() throws ParseException { /*@bgen(jjtree) Type */ - ASTType jjtn000 = new ASTType(this, JJTTYPE); - boolean jjtc000 = true;

	- jjtree.openNodeScope(jjtn000);
	+ ASTType jjtn000 = new ASTType(this, JJTTYPE);
	+ boolean jjtc000 = true;
	+ jjtree.openNodeScope(jjtn000);Token t;
	try {
	if (jj_2_13(2)) {
	ReferenceType();

付録 D－1. インスタンス別でのバグ修正パターンの実施回数（PMD のインスタンス②）

カテゴリ	操作内容	FactoryMethodパターンの各ロールにおけるバグ修正の実施回数		バグ修正パターン実施回数の合計
		Creator	Concrete Product	
CF	ADD		1	1(20%)
IF	APCJ		1	1(20%)
MD	ADD		1	1(20%)
SQ	AFO		1	1(20%)
	AMO	1		1(20%)

付録 D－2. インスタンス別でのバグ修正実施コミットの概要（PMD のインスタンス②）

バグ修正日	修正しているFactoryMethodパターン使用クラスと修正内容
2002-10-04 044844	LocalScope [public void addDeclaration(NameDeclaration nameDecl)]【IF-APCJ】
2003-01-30 023110	ScopeFactory [private void initScopeTriggers()]【SQ-AMO】
2003-06-27 061154	ClassScope 【CF-ADD】 [public ClassScope()]【SQ-AFO】

	[public String toString()]【MD-ADD】 ScopeFactory（インポート文の呼び出しのみ）

付録 D－３．インスタンス別でのバグ修正コミットの内容①（PMD のインスタンス②）

コミット日時	2002-10-04 04:48:44
コミット ID	658fc4a1fb27408fb690399d0c2d957080d28915
コミットコメント	cleaned up ideaj renderer, fixed symbol table bug
このコミットで修正されているファイル	pmd/etc/changelog.txt pmd/etc/go.bat pmd/regress/test/net/sourceforge/pmd/rules/UnusedLocalVariableTest.java pmd/src/net/sourceforge/pmd/PMD.java pmd/src/net/sourceforge/pmd/renderers/IDEAJRenderer.java pmd/src/net/sourceforge/pmd/symboltable/ContextManager.java pmd/src/net/sourceforge/pmd/symboltable/LocalScope.java pmd/src/net/sourceforge/pmd/symboltable/LookupController.java pmd/src/net/sourceforge/pmd/symboltable/NameDeclaration.java pmd/src/net/sourceforge/pmd/symboltable/SymbolFacade.java pmd/test-data/Unused15.java
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス： pmd/src/net/sourceforge/pmd/symboltable/LocalScope.java デザインパターンでの役割：Factory Method パターンの ConcreateProduct 変更内容： ・メソッド： [public void addDeclaration(NameDeclaration nameDecl)] 【IF-APCJ】

	「考察：この if 文の追加において、クラス NameDeclaration のメソッド getNode()が利用されている。
	同じコミットにおいて、クラス NameDeclaration はメソッド getNode()を新しく定義している。
	したがって、これはメソッド呼出し忘れのバグ修正ではないと考えられる。
	むしろ、メソッド addDeclaration の引数が ASTFormalParameter の場合、
	『何もせずに処理を終了する処理の記述忘れ』のためのバグ修正だと考えられる。」
	変更内容の元（diff）：
	@@ -14,6 +14,10 @@
	private Map names = new HashMap();
	
	public void addDeclaration(NameDeclaration nameDecl) {
	+ // this declaration needs to go somewhere... should this be delegated to a higher scope?
	+ if (nameDecl.getNode().jjtGetParent() instanceof ASTFormalParameter) {
	+ return;
	+ }
	if (names.containsKey(nameDecl)) {
	throw new RuntimeException(nameDecl + " is already in the symbol table");
	}

付録 D－４．インスタンス別でのバグ修正コミットの内容②（PMD のインスタンス②）

コミット日時	2003-01-30 02:31:10
コミット ID	be11288e2e0855a52298fdedb8363e66684b2252
コミットコメント	Fixed bug in symbol table; it wasn't creating a scope level when it hit a switch statement
このコミットで修正されて	pmd/etc/build.xml pmd/etc/changelog.txt

いるファイル	pmd/src/net/sourceforge/pmd/symboltable/ScopeCreator.java pmd/src/net/sourceforge/pmd/symboltable/ScopeFactory.java

このコミットにおいて、FactoryMethodパターンを使用しているファイルの修正内容	ファイルパス：
	pmd/src/net/sourceforge/pmd/symboltable/ScopeFactory.java
	デザインパターンでの役割：Factory Methodパターンの Creator
	変更内容：
	・メソッド：
	[private void initScopeTriggers()]
	・メソッド内部処理：
	【SQ-AMO】
	「考察：このバグ修正はメソッドの呼び出し忘れだと考えられる。
	このコミットにおいて、クラス ASTSwitchStatement に変更はなく。また、localTriggers であるクラス HashSet にも変更はない。したがって、『メソッドの呼び出し忘れ』を修正するためのバグ修正だと考えられる。」
	変更内容の元（diff）：
	@@ -16,6 +16,7 @@
	import net.sourceforge.pmd.ast.ASTUnmodifiedInterfaceDeclaration;
	import net.sourceforge.pmd.ast.Node;
	import net.sourceforge.pmd.ast.SimpleNode;
	+import net.sourceforge.pmd.ast.ASTSwitchStatement;
	
	import java.util.HashSet;
	import java.util.Set;
	@@ -48,6 +49,7 @@
	localTriggers.add(ASTBlock.class);
	localTriggers.add(ASTTryStatement.class);
	localTriggers.add(ASTForStatement.class);
	+ localTriggers.add(ASTSwitchStatement.class);
	localTriggers.add(ASTIfStatement.class);
	methodTriggers.add(ASTConstructorDeclaration.class);
	methodTriggers.add(ASTMethodDeclaration.class);

付録 D－５．インスタンス別でのバグ修正コミットの内容③（PMD のインスタンス②）

コミット日時	2003-06-27 06:11:54
コミット ID	c63615a765fc51e453cfffc10f0592fbb4098350
コミットコメント	Fixed symbol table bug
このコミットで修正されているファイル	pmd/regress/test/net/sourceforge/pmd/rules/StringToStringRuleTest.java pmd/regress/test/net/sourceforge/pmd/symboltable/AcceptanceTest.java pmd/regress/test/net/sourceforge/pmd/symboltable/ScopeCreatorTest.java pmd/regress/test/net/sourceforge/pmd/symboltable/ScopeFactoryTest.java pmd/src/net/sourceforge/pmd/symboltable/ClassScope.java pmd/src/net/sourceforge/pmd/symboltable/ScopeCreator.java pmd/src/net/sourceforge/pmd/symboltable/ScopeFactory.java pmd/src/net/sourceforge/pmd/symboltable/Search.java pmd/xdocs/credits.xml
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス： pmd/src/net/sourceforge/pmd/symboltable/ClassScope.java デザインパターンでの役割：Factory Method パターンの 変更内容： 【CF-ADD】 ・メソッド： [public ClassScope()] 【MD-ADD】 [public String toString()] 【SQ-AFO】 変更内容の元（diff）： @@ -7,10 +7,18 @@

```
 public class ClassScope extends AbstractScope {


+      private static int anonymousCounter = 1;
+
       private String className;


       public ClassScope(String className) {
            this.className = className;
+           anonymousCounter = 0;
+      }
+
+      public ClassScope() {
+           this.className = "Anonymous$" +
String.valueOf(anonymousCounter);
+           anonymousCounter++;
       }


       public Scope getEnclosingClassScope() {
@@ -41,7 +49,6 @@
                 return (NameDeclaration)
variableNames.keySet().iterator().next();
           }


-
           List images = new ArrayList();
           images.add(occurrence.getImage());
           if (occurrence.getImage().startsWith(className)) {
@@ -53,12 +60,11 @@
       }
```

	public String toString() {
	- return "ClassScope:" + super.glomNames();
	+ return "ClassScope:" + className + ":" + super.glomNames();
	}
	
	private String clipClassName(String in) {
	int firstDot = in.indexOf('.');
	return in.substring(firstDot + 1);
	}
	-
	}
	
	--- 	
	ファイルパス：pmd/src/net/sourceforge/pmd/symboltable/ScopeFactory.java
	デザインパターンでの役割：Factory Method パターンの
	変更内容：
	【】
	（インポート文の呼び出しのみなので、バグ修正だとみなさない）
	変更内容の元（diff）：
	@@ -12,6 +12,8 @@
	import net.sourceforge.pmd.ast.ASTUnmodifiedInterfaceDeclaration;
	import net.sourceforge.pmd.ast.Node;
	import net.sourceforge.pmd.ast.SimpleNode;
	+import net.sourceforge.pmd.ast.ASTClassBodyDeclaration;
	+import net.sourceforge.pmd.ast.ASTClassBody;
	
	import java.util.HashSet;
	import java.util.Set;

付録 E－1. インスタンス別でのバグ修正パターンの実施回数（JHotDraw のインスタンス①）

カテゴリ	操作内容	FactoryMethodパターンの各ロールにおけるバグ修正の実施回数	バグ修正パターン実施回数の合計
		Concrete Creator	
MC	DAP	2	2(100%)

付録 E－2. インスタンス別でのバグ修正実施コミットの概要（JHotDraw のインスタンス①）

バグ修正日	修正しているFactoryMethodパターン使用クラスと修正内容
2002-11-27 103028	EllipseFigure.java [public void drawBackground(Graphics g)]【MC-DAP】 RoundRectangleFigure.java [public void drawBackground(Graphics g)]【MC-DAP】

付録 E－3. インスタンス別でのバグ修正コミットの内容①（JHotDraw のインスタンス①）

コミット日時	2002-11-27 10:30:28
コミット ID	38ac091fbe53cdaf06099f3b4e48a7b0e33aceac
コミットコメント	bug-fix 634574: Ellipse, Ronded Rect off by 1
このコミットで修正されているファイル	JHotDraw/src/CH/ifa/draw/figures/EllipseFigure.java JHotDraw/src/CH/ifa/draw/figures/RoundRectangleFigure.java jhotdraw6/src/org/jhotdraw/figures/EllipseFigure.java jhotdraw6/src/org/jhotdraw/figures/RoundRectangleFigure.java
このコミットにおいて、FactoryM	ファイルパス：JHotDraw/src/CH/ifa/draw/figures/EllipseFigure.java デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct

ethod パターンを使用しているファイルの修正内容	変更内容：
	・メソッド：
	[public void drawBackground(Graphics g)]
	【MC-DAP】「考察：描画における、大きさのずれを修正するためのバグ修正だと考えられる。」
	・クラス内部処理：変更：他クラスの呼び出しメソッドへ与える値を変更
	(g.fillOval(r.x, r.y, r.width, r.height); -> g.fillOval(r.x, r.y, r.width-1, r.height-1);)
	変更内容の元（diff）：
	@@ -66,7 +66,7 @@
	
	public void drawBackground(Graphics g) {
	Rectangle r = displayBox();
	- g.fillOval(r.x, r.y, r.width, r.height);
	+ g.fillOval(r.x, r.y, r.width-1, r.height-1);
	}
	
	public void drawFrame(Graphics g) {
	
	--- 	
	ファイルパス：
	JHotDraw/src/CH/ifa/draw/figures/RoundRectangleFigure.java
	デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct
	変更内容：
	・メソッド：
	[public void drawBackground(Graphics g)]
	【MC-NAP】　・クラス内部処理：変更：他クラスの呼び出しメソッドへ与える値を変更
	(g.fillRoundRect(r.x, r.y, r.width, r.height, fArcWidth, fArcHeight);
	-> g.fillRoundRect(r.x, r.y, r.width-1, r.height-1, fArcWidth, fArcHeight);)

	変更内容の元（diff）：
	@@ -94,7 +94,7 @@
	
	　　public void drawBackground(Graphics g) {
	　　　　Rectangle r = displayBox();
	-　　　　g.fillRoundRect(r.x, r.y, r.width, r.height, fArcWidth, fArcHeight);
	+　　　　g.fillRoundRect(r.x, r.y, r.width-1, r.height-1, fArcWidth, fArcHeight);
	　　}
	
	　　public void drawFrame(Graphics g) {

付録 F－1. インスタンス別でのバグ修正パターンの実施回数（JHotDraw のインスタンス②）

カテゴリ	操作内容	FactoryMethodパターンの各ロールにおけるバグ修正の実施回数		実施回数の合計
		Concrete Creator	Concrete Product	
MC	DAP	2		2(20%)
	DM	1		1(10%)
	DNP	1		1(10%)
MD	ADD	2	3	5(50%)
SQ	AMO	1		1(10%)

付録 F－2. インスタンス別でのバグ修正実施コミットの概要（JHotDraw のインスタンス②）

バグ修正日	修正しているFactoryMethodパターン使用クラスと修正内容
2002-09-23 164726	BorderDecorator [public void drawBackground(Graphics g)]【MC-DNP】 TextFigure [public void figureRemoved(FigureChangeEvent e)]【MC-DM】 StandardDrawing [public synchronized Figure orphan(Figure figure)]【MC-DAP】 [public synchronized Figure add(Figure figure)]【MC-DAP】

2002-11-27 103028	保留（EllipseFigure.java【MC-DAP】） 保留（RoundRectangleFigure.java【MC-DAP】）
2003-02-05 013301	GroupFigure [public void setAttribute(FigureAttributeConstant fac, Object object)]【MD-ADD】
2003-05-20 235947	LineConnection [public void release()]【SQ-AMO】 [public void removeFromContainer(FigureChangeListener c)]【MD-ADD】
2003-05-25 192004	ChangeConnectionEndHandle [protected boolean canConnectTo(Figure figure)]【MD-ADD】 ChangeConnectionStartHandle [private Connector findConnectionTarget(int x, int y, Drawing drawing)]【MD-ADD】

付録Ｆ－３．インスタンス別でのバグ修正コミットの内容①（JHotDraw のインスタンス②）

コミット日時	2002-09-23 16:47:26
コミット ID	ea2cb8be59db9721209682394b69158a5e6559bf
コミットコメント	Fixed problem with TextArea figures + cascading FigureChangeEvent
このコミットで修	JHotDraw/src/CH/ifa/draw/contrib/GraphicalCompositeFigure.java

正されているファイル	JHotDraw/src/CH/ifa/draw/contrib/SimpleLayouter.java JHotDraw/src/CH/ifa/draw/contrib/TextAreaFigure.java JHotDraw/src/CH/ifa/draw/contrib/TextAreaTool.java JHotDraw/src/CH/ifa/draw/figures/BorderDecorator.java JHotDraw/src/CH/ifa/draw/figures/TextFigure.java JHotDraw/src/CH/ifa/draw/framework/FigureChangeEvent.java JHotDraw/src/CH/ifa/draw/samples/javadraw/JavaDrawApp.java JHotDraw/src/CH/ifa/draw/standard/CompositeFigure.java JHotDraw/src/CH/ifa/draw/standard/StandardDrawing.java jhotdraw6/src/org/jhotdraw/contrib/GraphicalCompositeFigure.java jhotdraw6/src/org/jhotdraw/contrib/SimpleLayouter.java jhotdraw6/src/org/jhotdraw/contrib/TextAreaFigure.java jhotdraw6/src/org/jhotdraw/contrib/TextAreaTool.java jhotdraw6/src/org/jhotdraw/figures/BorderDecorator.java jhotdraw6/src/org/jhotdraw/figures/TextFigure.java jhotdraw6/src/org/jhotdraw/framework/FigureChangeEvent.java jhotdraw6/src/org/jhotdraw/samples/javadraw/JavaDrawApp.java jhotdraw6/src/org/jhotdraw/standard/CompositeFigure.java jhotdraw6/src/org/jhotdraw/standard/StandardDrawing.java
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス： JHotDraw/src/CH/ifa/draw/figures/BorderDecorator.java デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct 変更内容： ・メソッド： [public void figureInvalidated(FigureChangeEvent e)] 【MC-DNP】 「考察：同じコミットにて、クラス FigureChangeEvent で新しくコンストラクタが作られている。 このクラス BorderDecorator では、同じコミットで作られたコンストラクタに変更している。 つまり、メソッドの呼び出し間違いではないという事は確実に言える。

	ただ、このバグ修正の他の考察が思い浮かばない。」
	
	・クラス内部処理：変更：他クラスのインスタンス呼び出しで与える引数の変更
	(super.figureInvalidated(new FigureChangeEvent(e.getFigure(), rect));->super.figureInvalidated(new FigureChangeEvent(this, rect, e));)
	変更内容の元（diff）：
	@@ -91,7 +91,7 @@
	public void figureInvalidated(FigureChangeEvent e) {
	Rectangle rect = e.getInvalidatedRectangle();
	rect.grow(getBorderOffset().x, getBorderOffset().y);
	- super.figureInvalidated(new FigureChangeEvent(e.getFigure(), rect));
	+ super.figureInvalidated(new FigureChangeEvent(this, rect, e));
	}
	
	public Insets connectionInsets() {
	
	--------------- 	
	ファイルパス：JHotDraw/src/CH/ifa/draw/figures/TextFigure.java
	デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct
	変更内容：
	・メソッド：
	[public void figureRemoved(FigureChangeEvent e)]
	【MC-DM】
	「考察：修正部分で利用している listener()はクラス AbstractFigure のメソッドで、
	実行するとインスタンス FigureChangeListener を返却する。
	クラス FigureChangeListener はリスナーであり、このコミットでの修正はない。
	また、このコミットよりも前にメソッド figureRemoved は作られている。

	したがって、この MC-DM は『メソッド呼出し間違いを修正するためのバグ修正』だと考える。」
	
	・クラス内部処理：削除：listener().figureRequestRemove(new FigureChangeEvent(this));
	
	・クラス内部処理：追加：Rectangle rect = invalidateRectangle(displayBox());
	
	・クラス内部処理：追加：listener().figureRemoved(new FigureChangeEvent(this, rect, e));
	
	変更内容の元（diff）：
	
	@@ -328,7 +328,8 @@
	
	public void figureRemoved(FigureChangeEvent e) {
	
	if (listener() != null) {
	
	- listener().figureRequestRemove(new FigureChangeEvent(this));
	
	+ Rectangle rect = invalidateRectangle(displayBox());
	
	+ listener().figureRemoved(new FigureChangeEvent(this, rect, e));
	
	}
	
	}
	
	--- 	
	
	ファイルパス：JHotDraw/src/CH/ifa/draw/standard/StandardDrawing.java
	
	デザインパターンでの役割：FactoryMethod パターンでの
	
	変更内容：
	
	・メソッド：
	
	[public synchronized Figure orphan(Figure figure)]
	
	【MC-DAP】
	
	「考察：同じコミット実施している MC-DNP と異なり、コンストラクタへ与える値を変更しているだけである。
	
	また、同じコミットで変更しているクラス FigureChangeEvent では、
	
	引数 2 つのコンストラクタに修正を加えていない。

	したがって、このバグ修正は『コンストラクタへ渡す値を間違えていたため』、修正したと考える。」
	
	・メソッド変数：追加：（Rectangle r = invalidateRectangle(displayBox());）
	
	・メソッド内部処理：変更：他クラスのコンストラクタへ与える引数の変更。
	
	（new FigureChangeEvent(figure, null); -> new FigureChangeEvent(figure, rect);）
	
	[public synchronized Figure add(Figure figure)]
	
	【MC-DAP】
	
	・メソッド変数：追加：（Rectangle r = invalidateRectangle(displayBox());）
	
	・メソッド内部処理：変更：他クラスのコンストラクタへ与える引数の変更。
	
	（new FigureChangeEvent(figure, null); -> new FigureChangeEvent(figure, rect);）
	
	変更内容の元（diff）：
	
	@@ -94,7 +94,8 @@
	
	Figure orphanedFigure = super.orphan(figure);
	
	// ensure that we remove the top level figure in a drawing
	
	if (orphanedFigure.listener() != null) {
	
	-
	orphanedFigure.listener().figureRequestRemove(new FigureChangeEvent(orphanedFigure, null));
	
	+ Rectangle rect = invalidateRectangle(displayBox());
	
	+
	orphanedFigure.listener().figureRequestRemove(new FigureChangeEvent(orphanedFigure, rect));
	
	}
	
	return orphanedFigure;
	
	}
	
	@@ -102,7 +103,8 @@
	
	public synchronized Figure add(Figure figure) {
	
	Figure addedFigure = super.add(figure);

	if (addedFigure.listener() != null) {
	-
	addedFigure.listener().figureRequestUpdate(new FigureChangeEvent(figure, null));
	+ Rectangle rect = invalidateRectangle(displayBox());
	+
	addedFigure.listener().figureRequestUpdate(new FigureChangeEvent(figure, rect));
	return addedFigure;
	}
	return addedFigure;
	
	---------- 	

付録 F－４．インスタンス別でのバグ修正コミットの内容②（JHotDraw のインスタンス②）

コミット日時	2002-11-27 10:30:28
コミット ID	38ac091fbe53cdaf06099f3b4e48a7b0e33aceac
コミットコメント	bug-fix 634574: Ellipse, Ronded Rect off by 1
このコミットで修正されているファイル	JHotDraw/src/CH/ifa/draw/figures/EllipseFigure.java JHotDraw/src/CH/ifa/draw/figures/RoundRectangleFigure.java jhotdraw6/src/org/jhotdraw/figures/EllipseFigure.java jhotdraw6/src/org/jhotdraw/figures/RoundRectangleFigure.java
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス：JHotDraw/src/CH/ifa/draw/figures/EllipseFigure.java デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct 変更内容： ・メソッド： [public void drawBackground(Graphics g)] ・メソッド内部処理：

	【MC-DAP】「考察：描画における、大きさのずれを修正するためのバグ修正だと考えられる。」
	・変更：他クラスの呼び出しメソッドに与える引数の変更。
	変更内容の元（diff）：
	@@ -66,7 +66,7 @@
	
	public void drawBackground(Graphics g) {
	Rectangle r = displayBox();
	- g.fillOval(r.x, r.y, r.width, r.height);
	+ g.fillOval(r.x, r.y, r.width-1, r.height-1);
	}
	
	public void drawFrame(Graphics g) {
	
	--- 	
	ファイルパス：
	JHotDraw/src/CH/ifa/draw/figures/RoundRectangleFigure.java
	デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct
	変更内容：
	・メソッド：
	[public void drawBackground(Graphics g)]
	・メソッド内部処理：
	【MC-DAP】「考察：描画における、大きさのずれを修正するためのバグ修正だと考えられる。」
	・変更：他クラスの呼び出しメソッドに与える引数の変更。
	変更内容の元（diff）：
	@@ -94,7 +94,7 @@
	
	public void drawBackground(Graphics g) {
	Rectangle r = displayBox();
	- g.fillRoundRect(r.x, r.y, r.width, r.height, fArcWidth, fArcHeight);

	+ g.fillRoundRect(r.x, r.y, r.width-1, r.height-1, fArcWidth, fArcHeight);
	}
	
	public void drawFrame(Graphics g) {

付録 F－５．インスタンス別でのバグ修正コミットの内容③（JHotDraw のインスタンス②）

コミット日時	2003-02-05 01:33:01
コミット ID	1222bc0ad06a91363ebee1c96e51d90228c81995
コミットコメント	fix - [675576] GroupFigure needs to forward new AttributeConstant calls
このコミットで修正されているファイル	JHotDraw/src/CH/ifa/draw/figures/GroupFigure.java jhotdraw6/src/org/jhotdraw/figures/GroupFigure.java

このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス：JHotDraw/src/CH/ifa/draw/figures/GroupFigure.java
	デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct
	変更内容：
	・メソッド：
	[public void setAttribute(FigureAttributeConstant fac, Object object)]
	【MD-ADD】
	・追加：新しいメソッドの追加（setAttribute(String name, Object value)のオーバーライド）。
	（参考までに、
	public void setAttribute(String name, Object value) {
	super.setAttribute(name, value);
	FigureEnumeration fe = figures();
	while (fe.hasNextFigure()) {
	fe.nextFigure().setAttribute(name, value);
	}

```
        }
)
変更内容の元（diff）：
@@ -75,6 +75,7 @@


    /**
        * Sets the attribute of all the contained figures.
+       * @deprecated see setAttribute(FigureAttributeConstant,Object)
        */
        public void setAttribute(String name, Object value) {
                super.setAttribute(name, value);
@@ -83,4 +84,14 @@
                        fe.nextFigure().setAttribute(name, value);
                }
        }
+       /**
+        * Sets the attribute of the GroupFigure as well as all contained Figures.
+        */
+       public void setAttribute(FigureAttributeConstant fac, Object object){
+               super.setAttribute(fac, object);
+               FigureEnumeration fe = figures();
+               while (fe.hasNextFigure()) {
+                       fe.nextFigure().setAttribute(fac, object);
+               }
+       }
 }
```

付録 F－６．インスタンス別でのバグ修正コミットの内容④（JHotDraw のインスタンス②）

コミット日時	2003-05-20 23:59:47
コミット ID	a89a06af206ff6f03a3d9a8495c43537b305de95
コミットコメント	Bug fix for undo/redo
このコミットで修正されているファイル	JHotDraw/src/CH/ifa/draw/figures/LineConnection.java jhotdraw6/src/org/jhotdraw/figures/LineConnection.java
このコミットにおいて、FactoryMethod パターンを使用しているファイルの修正内容	ファイルパス： JHotDraw/src/CH/ifa/draw/figures/LineConnection.java デザインパターンでの役割：FactoryMethod パターンでの Concreat 変更内容： ・メソッド： [public void release()] 【SQ-AMO】 「考察：同時にコミットされているコミットが他にないため、『メソッド呼出し忘れ』のバグ修正だと考えられる。」 ・メソッド内部処理： [条件文：if(getStartConnector() != null）] ・内部処理：追加：依存している図の削除に関するメソッド呼出し（startFigure().removeDependendFigure(this);） [条件文：if(getEndConnector() != null）] ・内部処理：追加：依存している図の削除に関するメソッド呼出し（endFigure().removeDependendFigure(this);） [public void removeFromContainer(FigureChangeListener c)] 【MD-ADD】 ・追加：新しいメソッドの追加。（同時期に修正したメソッド release を内部で呼び出している） 変更内容の元（diff）： @@ -317,9 +317,11 @@ handleDisconnect(startFigure(), endFigure());

	if (getStartConnector() != null) {
	
	startFfigure().removeFigureChangeListener(this);
	+ startFigure().removeDependendFigure(this);
	}
	if (getEndConnector() != null) {
	endFigure().removeFigureChangeListener(this);
	+ endFigure().removeDependendFigure(this);
	}
	}
	
	@@ -360,4 +362,13 @@
	public void visit(FigureVisitor visitor) {
	visitor.visitFigure(this);
	}
	+
	+ /**
	+ * @see CH.ifa.draw.framework.Figure#removeFromContainer(CH.ifa.draw.framework.FigureChangeListener)
	+ */
	+ public void removeFromContainer(FigureChangeListener c) {
	+ super.removeFromContainer(c);
	+ release();
	+ }
	+
	}

付録 F－７．インスタンス別でのバグ修正コミットの内容⑤（JHotDraw のインスタンス②）

コミット日時	2003-05-25 19:20:04

コミットID	f898e6d840754850a1876b8087ef9e161a552bc6
コミットコメント	Fix for findConnectionTarget() method of ChangeConnectionHandle. Old version didn't take direction of connection into account, which could result in incorrect drawings (if syntax rules and directional connections were a concern). New version solves this by abstract method which is overridden properly in ChangeConnectionStartHandle and ChangeConnectionEndHandle.
このコミットで修正されているファイル	JHotDraw/src/CH/ifa/draw/standard/ChangeConnectionEndHandle.java JHotDraw/src/CH/ifa/draw/standard/ChangeConnectionHandle.java JHotDraw/src/CH/ifa/draw/standard/ChangeConnectionStartHandle.java jhotdraw6/src/org/jhotdraw/standard/ChangeConnectionEndHandle.java jhotdraw6/src/org/jhotdraw/standard/ChangeConnectionHandle.java jhotdraw6/src/org/jhotdraw/standard/ChangeConnectionStartHandle.java
このコミットにおいて、FactoryMethodパターンを使用しているファイルの修正内容	ファイルパス： JHotDraw/src/CH/ifa/draw/standard/ChangeConnectionEndHandle.java デザインパターンでの役割：FactoryMethodパターンでの Concreat 変更内容： ・メソッド： [protected boolean canConnectTo(Figure figure)] 【MD-ADD】 「新しく作成したメソッド canconnectTo()で使用されているメソッド getConnection()はクラス ChangeConnectionHandle のメソッド。 getConnection()を実行すると、インスタンス ConnectionFigure を返却する。 インターフェース ConnectionFigure における、メソッド canConnect(Figure s, Figure e)は Boolean を返却する抽象メソッドである。 修正前は、スーパークラスが canConnectTo()と同じ動作を定義していた。 しかし、そのような形にすると、継承する他のどのクラスも canConnectTo()と同じ動作をしてしまうため、

	ChangeConnectionStartHandle と ChangeConnectionEndHandle のみが canConnectTo()を実行するために、
	今回の修正が必要だった。
	つまり、『スーパークラスで具体的に定義すべきではなかった実装だったという事』と、
	この『クラス ChangeConnectionStartHandle と ChangeConnectionEndHandle で実装すべきアルゴリズムだったという事』
	（もっと言えば、各サブクラスで適宜実装が必要なアルゴリズムだった）という理由より、
	修正を行っているバグ修正だと考えられる。」
	
	・追加：新しいメソッドの追加。<－同時期に追加されたスーパークラス ChangeConnectionHandle の抽象メソッドを実装している。
	変更内容の元（diff）：
	@@ -83,4 +83,8 @@
	return tempEndConnector;
	}
	}
	+
	+ protected boolean canConnectTo(Figure figure) {
	+ return getConnection().canConnect(source().owner(), figure);
	+}
	}
	
	--- 	
	ファイルパス：JHotDraw/src/CH/ifa/draw/standard/ChangeConnectionHandle.java<－P-mart リポジトリによれば、このクラスは Strategy パターンのインスタンスらしい。その影響か。（P-mart によれば、このクラスは FactoryMethod パターンではない）
	デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct
	変更内容：
	・メソッド：

[private Connector findConnectionTarget(int x, int y, Drawing drawing)]

・メソッド内部処理：条件文：条件式：変更：呼び出しメソッドの変更

(getConnection().canConnect(source().owner(), target)) -> canConnectTo(target))

[protected abstract boolean canConnectTo(Figure figure);]

・追加：抽象メソッドの追加

変更内容の元（diff）：

```
@@ -145,15 +145,36 @@
        private Connector findConnectionTarget(int x, int y, Drawing drawing) {
                Figure target = findConnectableFigure(x, y, drawing);

-               if ((target != null) && target.canConnect()
+               if ((target != null)
+                       && target.canConnect()
                         && target != fOriginalTarget
                         && !target.includes(owner())
-                        && getConnection().canConnect(source().owner(), target)) {
+               /*
+                * JP, 25-May-03: Fix for ignored direction when checking
+                * connectability. Old version didn't take direction of
+                * connection into account, which could result in incorrect
+                * drawing if syntax rules were a concern.
+                *
+                * See also new canConnectTo() method below.
+                *
+                * Was:
+                *
```

```
+                 * && getConnection().canConnect(source().owner(),
target)) {
+                 */
+                         && canConnectTo(target)) {
                                 return findConnector(x, y, target);
                         }
                         return null;
                 }


+       /**
+        * Called to check whether this end of the connection can
connect to the
+        * given target figure. Needs to be overriden by the start and end
changers
+        * to take the connection's direction into account during the
check. JHD 5.4
+        * beta and before did not do this.
+        */
+       protected abstract boolean canConnectTo(Figure figure);
+
          protected Connector findConnector(int x, int y, Figure f) {
                 return f.connectorAt(x, y);
          }


---------------
```

ファイルパス：JHotDraw/src/CH/ifa/draw/standard/ChangeConnectionStartHandle.java

デザインパターンでの役割：FactoryMethod パターンでの ConcreatProduct

変更内容：

・メソッド：

[protected boolean canConnectTo(Figure figure)]

【MD-ADD】

	・追加：新しいメソッドの追加。＜－同時期に追加されたスーパークラス ChangeConnectionHandle の抽象メソッドを実装している。
	return getConnection().canConnect(source().owner(), figure);
	return getConnection().canConnect(figure, source().owner());
	
	変更内容の元（diff）：
	@@ -83,4 +83,8 @@
	return tempStartConnector;
	}
	}
	+
	+ protected boolean canConnectTo(Figure figure) {
	+ return getConnection().canConnect(figure, source().owner());
	+}
	}