

オブジェクト指向設計の構造的欠陥検出モデルの検討

佐藤 美穂[†], チョインゾン ムンフナサン[†], 上田 賀一[†]

[†]茨城大学

ソフトウェアライフサイクルの初期に欠陥を検出し、改善することは重要である。本研究では多くの文献によって得られたオブジェクト設計欠陥（設計上の構造的な問題点）を検出するモデルを検討する。設計欠陥リストを作成し、それら欠陥を検出するためメトリクスとその閾値を定義した。また、欠陥検出ツールを開発し、UML 図での欠陥検出を行った。その結果、検出を試みた欠陥 47 種中 41 種の欠陥を検出することができた。

A Study of Structural Defects Detection Model for Object-Oriented Design

Miho SATO[†], Munkhnasan CHOINZON[†] and Yoshikazu UEDA[†]

[†]Ibaraki University

It is important that defects are detected and modified in an earlier stages of software life cycle. This study examines a model to detect object-oriented(OO) design defects (structural problems in the OO design) provided by many literatures. We create a list of design defects and define metrics and their thresholds for detecting these defects. We also developed defect detection tool and detected design defects from UML diagrams. As the result, 41 design defects were detected from 47 design defects defined.

1 はじめに

高品質なソフトウェアを開発することはソフトウェア開発の重大な目標のひとつであり、プログラムの問題点を検出し、修正することによって品質を高めることができる。設計工程で作られた欠陥をソフトウェア納入後に修正した場合、設計工程で修正した場合と比較するとコストが 100 倍以上かかる [1] と言われており、設計工程で設計欠陥（設計上の構造的な問題点）を検出し、修正することは有用であると考えられる。

今日、オブジェクト指向 (Object-Oriented: OO) 設計はソフトウェア開発環境において重要な役割を果たしている。OO 方法論は設計工程でオブジェクトやクラス、その属性・操作、それらの関係等を特定するため、その工程で欠陥を検出することができる。

これまでに多くの OO メトリクスが提案されているが、これらの多くは測定結果の解釈方法や分析方法が明確に定義されておらず、それが重大な状況を示すか否かを判断するための評価基準や閾値も明確でない。したがって、意図されたメトリクスを適用することやその有用性を示すことが非

常に困難である [2, 3]。

一方で、専門家や熟練設計者によって多くの OO 設計ルールや経験則が定義されており、これらに対する違反は設計品質に影響を及ぼす。しかし、多くの経験則の定義は自然言語形式で記述されているため、手作業でルールおよび経験則の違反を検出することは、特に大規模なシステムでは困難であり、時間がかかる。メトリクスを用いることによって、いくつかの設計経験則の違反を自動的に検出することができる。

本研究は OO 技術を用いて記述された設計書からのメトリクス測定結果とその閾値によって OO 設計欠陥を検出する方法を検討する。また、実際に UML 設計からの欠陥検出を試みた。我々は本研究が OO 設計品質を特徴付ける内部属性の評価に貢献すると考えている。

2 研究の背景

様々な文献 [4-8] でクラスやオブジェクト、クラス間の関係、クラス階層を定義する際に起こりうる OO 設計特有の問題が多く明らかにされているが、自動識別はサポートされていない。

これまでに様々なメトリクスが提案されている [9–17] が、これらの欠陥の検出に有効なメトリクスは数少ない。本研究は既存の OO 設計欠陥およびメトリクスにのみ焦点をあてる。

Chidamber と Kemerer [11] が提案した OO メトリクススイートは、Basili ら [18] によってその大部分が欠陥傾向にあるクラスを予測するための品質指標として有効であることが実証的に確認された。また、Briand ら [19] はシステムクラス中の OO 設計測定と欠陥検出確率の関係を定義するために実証的検証を行い、メソッド呼び出しの頻度と継承階層の深さが欠陥傾向にあるクラスの重要な品質要因であることを示した。

Erni ら [20] は同一コンポーネント全てに関連するシンプルなメトリクスセットであるマルチメトリクスを用いて問題のある箇所を検出する体系的なアプローチを提案した。この研究ではメトリクス値を解釈するため、外れ値や重大な値の特定に有効な閾値と動向分析を用いているが、メトリクス値から結論を導くにはソフトウェア開発者による検査が必要である。

Colin Kirsopp ら [21] は様々なメトリクスが測定可能な分析ツールの測定結果と設計経験則により、15 のクラスから構成される小規模なシステムの品質評価を行った。その結果、いくつかのメトリクスは測定結果を具体的な問題として解釈することが困難であった。また著者らによって事前に定義されたいくつかの設計問題に関してはそれらを検出する適切なメトリクスを収集することができなかったが、これは適切なメトリクスの選択と収集が、重要であることを示す。

近年、UML ベースの OO 設計品質測定ツールが開発された [22]。11 種の UML 図に対応し、設計要素の構造的特性をカバーする様々な OO 設計の測定が可能である。効果的に設計検出するために、SDMetrics ツールで得られたソフトウェア設計の測定結果を解釈するには、データ分析技術が必要である。SDMetrics はベンチマークや予測モデルのような、多くのデータ分析技術を提供する。また、Java ソースコードを対象とした JStyle [23] のような商用ツールでも設計欠陥を検出できる。しかしこれらのツールは多くの例外も検出してしまうという問題がある。

このように現在の設計上の問題を検出する方法は、まず測定対象に適すると思われるメトリクスを測定した後に、その測定対象と測定値を調査する。つまり、特定の設計上の問題ごとに事前に定義されたメトリクスがなく、測定後の調査によって測定対象にどんな設計上の問題が存在しているかが決定される。そのためメトリクスが誤用され

やすく、測定結果に誤解が生じる場合がある。

3 アプローチ

この章ではメトリクスを用いた OO 設計欠陥検出モデル [24] に関して以下の順序で記述する。

- 関連研究を調査して作成した設計欠陥リスト
- 設計欠陥が影響を及ぼす設計特性への分類
- 各設計欠陥を検出するメトリクスの特定
- 各メトリクスに関する閾値の特定

3.1 OO 設計欠陥の特定

関連研究からクラスやオブジェクト、クラスの属性やメソッド、クラス間の関係およびクラス階層を定義する際に起こりうる OO 設計欠陥を調査し、多くの設計ガイドライン、ルール、欠陥を発見した [4–8, 25–29]。設計ルールの違反は、ソフトウェアに潜在的な設計欠陥を与える可能性がある。しかし、経験則は単なるガイドラインであり、必ず従わなければならないものではなく [29]、違反がいつも設計欠陥に該当するとは限らない。測定対象によっては、経験則のいくつかの違反はある程度許容できるかもしれない。

我々は 45 個の OO 設計欠陥をリスト化した。参考文献とともに表 1 に示す。収集したがリスト化しなかったものは、他と冗長であるか、検出に詳細な情報が必要であり設計から検出することが困難であるかのどちらかである。

3.2 設計欠陥の OO 設計特性への分類

設計欠陥リストを、違反した際に影響が及ぶ設計特性に分類する。本研究での設計特性とは、内部属性の観点からみた OO 設計品質の特徴である。本研究では OO 設計の内部品質の特徴として、表 2 に示す 9 個の設計特性を選択した。様々なプロジェクトや設計者により、異なる設計特性が選択される可能性がある [30]。今回は OO 設計の内部品質を示すために一般的で重要な特徴であると考えられる設計特性を選択した。設計欠陥を設計特性に分類したものを表 2 に示す。欠陥を検出するために用いたメトリクスも共に示し、メトリクスの概要については表 3 と表 4 に示す。

3.3 欠陥検出メトリクスの特定と定義

設計欠陥リストの各欠陥をメトリクスによって検出する。既に多くの OO 設計メトリクスが提案されている [9, 11–15]。このうち表 3 に示す 20 個のメトリクスを欠陥検出に用いる。いくつかの設

表 1: 設計欠陥リスト

#	欠陥名	概要
D1	大規模なクラス	クラスのインスタンス変数またはメソッドが多すぎる [7]
D2	大規模なメソッド	クラスのメッセージ送信数またはパラメータが多すぎる [28]
D3	小規模なクラス	クラスの属性またはメソッドが少ない/無い [7, 25]
D4	公開インタフェースの過多	クラスの public インタフェースが多すぎる [29]
D5	データの貧弱なカプセル化	クラスが public 属性を持つ [29]
D6	貧弱なインタフェース	クラスが public メソッドを持たない [25]
D7	隠すべきメソッド	自クラスでしか用いないメソッドを公開している [29]
D8	非公開メソッドの過多	クラスの private/protected メソッドが多すぎる [25]
D9	不十分な抽象化クラス	抽象クラスが非常に少ない [28]
D10	複雑な抽象化	祖先クラスが多い [37]
D11	余分なサブクラス	スーパークラスが1つのサブクラスしか持たない [27]
D12	関連しないデータと振舞い	クラスの属性とメソッド間に関連がない [29]
D13	多面的な抽象化	スーパークラスがいくつかの抽象側面を持つ [29]
D14	データクラス	クラスがデータだけをもつ [25]
D15	引数のないメソッド	クラスの大抵のメソッドがパラメータを持たず、処理を行うために public/クラス変数を用いる [5]
D16	クラスに変更されたメソッド	メソッドがクラスにされている [29]
D17	関連しない抽象化	クラスが多くの public アクセサを持つ [29]
D18	複数のタスク	複数のタスクを行うメソッドがある [26]
D19	未利用	クラスの責務に関係ないメソッドが存在する [26]
D20	高い外部属性アクセス	外部属性へアクセスし過ぎる [28]
D21	高い協調性	関連するクラスの数が多い [29]
D22	相互依存関連	異なるクラスのメソッド間が相互依存する [25]
D23	誤ったクラスにあるメソッド	メソッドの大抵のメッセージが同一オブジェクトに対するものである [25]
D24	無関係なクラス	システム外にあるべきクラスが存在する [29]
D25	メッセージ送信の過多	コラボレータへ送信するメッセージが多い [29]
D26	複雑なインタフェース	メソッドのパラメータ数が多い [28]
D27	高いメソッド複雑性	メソッド実行時に呼び出すメソッドが多い [8]
D28	高いクラス複雑性	クラスのメソッドの大抵が高い複雑性をもつ [15]
D29	非一貫性	メソッドが同一のパラメータを2度渡す [31]
D30	冗長な引数	同一引数がクラスの大抵のメソッドへ渡される [25]
D31	不十分な共通点	2つ以上のクラスが共通データだけを共有する [29]
D32	貧弱なサブクラス	継承されるメソッドの割合が低い [15]
D33	避けられた抽象化	サブクラスによって追加されるメソッドが多い [15]
D34	仕事しないスーパークラス	スーパークラスの public メソッドが少ない/無い [8]
D35	高いクラス階層	クラス階層が深過ぎる [28]
D36	冗長な属性宣言	同一スーパークラスから継承されるサブクラスが同じデータを持つ [6]
D37	スーパークラスの誤ったメンバ	1つのサブクラスとしか関連しないメンバがスーパークラスにある [6]
D38	継承の不適当な使用	サブクラスがスーパークラスの一部のインタフェースだけを使用し、継承したデータを用いない [6]
D39	サブクラスに変更されたオブジェクト	クラスのオブジェクトがそのクラスのサブクラスにされる [29]
D40	サブクラスに変更された属性	不変データを返すメソッドのみのサブクラスがある [6]
D41	データ隠蔽の弱化	スーパークラスの大抵のデータが private でない [29]
D42	高い関連性	クラスが多くのオブジェクトを含む [29]
D43	役立たない抑制関連	クラスが他クラスのオブジェクトを含むとき、集約元クラスから集約先オブジェクトへの送信するメッセージがない [29]
D44	冗長な依存	同一集約元クラスを持つ集約先オブジェクトが集約先オブジェクト間で使用関係を持つ [29]
D45	委譲の不適当な使用	頻繁に多くの単純な委譲を用いる [6]

表 2: 設計特性に関する設計欠陥とメトリクス

設計特性	#	対応するメトリクス
サイズ	D1-1	NOA
	D1-2	NOM
	D2-1	NMSM
	D2-2	NOP
	D3-1	NOA*1
	D3-2	NOM*1
カプセル性	D4	NPubM
	D5	NPA
	D6	NPubM*2
	D7	NACUM
抽象性	D8	NPM
	D9	RAC
	D10	NOAC
	D11	NOS
凝集度	D12	LCOM
	D13	LCOM, NOA, NOM
	(D13+)	ASRMU, NOA, NOM
	D14	NOM*2
	D15-1	RMNP, NPA
	D15-2	RMNP, NSV
	D16	NOM*1
	D17	NPAM
	D18	NMSM, NOP
	D19-1	NMMI
	D19-2	NTAC
結合度	D20	RTAO
	D21	NOR
	D22	NTWD
	D23	RMSO
	D24	NMR
	D25	MPC
メッセージング	D26	NOP
複雑度	D27	NMSM
	D28	RMHC
	D29	NSP
	D30	RMHSP
継承	D31	RIM*3
	D32	RIM
	D33	RAM
	D34	NPMBC
	D35	DIT
	D36	NSAIS
	D37	NITBM
	D38	NIA
	D39	NOIDC, NOA, NOM ☆
	D40	NOM, NPAM, NGIS ★
	D41	NPABC
コンポジション	D42	DAC
	D43	NMCCC
	D44	NMOCC
	D45	RMDC

☆の閾値…NOIDC=1 かつ NOA=0 かつ NOM=0 : bad

★の閾値…NOM=0 かつ NPAM-NGIS=0 : bad

*1の閾値…1 : bad, 0 : very bad, *2の閾値…0 : bad,

*3の閾値…1 : bad, その他の閾値は表 3, 表 4 参照

(D13+) は D13 の代わりに用いるメトリクス

計欠陥は、その検出に適したメトリクスを見つけることができなかったため、欠陥検出用に 21 個のメトリクスを新しく定義する。これらのメトリクスの定義の概要を表 4 に示す。設計欠陥を検出するためにいくつかのメトリクスを組み合わせる用いる場合がある。

3.4 メトリクスの閾値の特定

メトリクス値を得たときに、その値がどのような状況を示すかを判断する必要があり、その際に閾値は有用である。メトリクス値がある閾値を越えるならば、その設計要素は重大な状況であると考えることができ、再設計が必要である可能性がある。閾値は文献 [15] において、メトリクス値の適正な範囲を設定するために用いられる経験則値として定義される。これらの閾値は実際の問題の有無にかかわらず、例外を特定するために用いられる。

表 3 と表 4 のメトリクス全てに対して閾値を定義する。あるメトリクス値が好ましくない範囲にあるなら、欠陥が存在する可能性があることを意味し、設計を再考するべきである。表 2 の対応関係から、それがどんな設計欠陥であるかが容易に分かる。本研究では各閾値を定義するため、メトリクスを表 5 に示す 3 つのグループに分類した。表 3 と表 4 の右欄に、閾値グループの番号を示す。

表 5: 閾値グループ概要

Gr.	閾値グループ概要
1	専門家による閾値に基づいて定義された閾値
2	設計欠陥の記述から得ることができる閾値
3	本研究で提案する閾値

メトリクスの閾値に関する一般的な基準は無く、同一のメトリクスについて様々な研究者が様々な閾値を提案してきた。また Benlarbi ら [32] によって、閾値以下の値を持つ設計要素であってもまだ高い欠陥傾向を示し得ることが指摘されている。閾値以下であるがまだ“重大な状況”である可能性が高い設計要素の検出は、複数の閾値を定義することにより増やすことができる。そこで閾値 Gr.1 と Gr.3 において 3 段階の閾値を定義する。3 つの閾値で分けられる 3 つの範囲の意味を直観的に分かりやすくするため、言語的な値「little bad」「bad」「very bad」を用いる。

Gr.1 の多くは Lorenz と Kidd [15] により提案された閾値に基づいて閾値を設定する。彼らは Smalltalk や C++ プロジェクトでの経験に基づいた OO 測定に関する多くの閾値を提案した。このグループの

表 3: 既存のメトリクス

メトリクス	概要	閾値	Gr.
NOA	クラスの属性数	7-9 :LB, 9< :B [15]	1
NOM	クラスのメソッド数	20-30 :LB, 31-50 :B, 50< :VB [15, 28]	1
NPM	クラスの private メソッド数	14-22 :LB, 22-34 :B, 34< :VB	3
NMSM [15]	メソッドのメッセージ送信数	10-11 :LB, 12-14 :B, 14< :VB [15]	1
NOP [15]	メソッドの引数の数	5-6 :LB, 7-8 :B, 8< :VB [15]	1
NPubM	クラスの public メソッド数	6-8 :LB, 9-10 :B, 10< :VB [15]	1
NPA	クラスの public 属性数	2-3 :B, 3< :VB [15]	1
RAC	クラス総数における抽象クラス数の割合 継承したメソッドの割合	10-12%未満 :LB, 8-10%未満 :B, 8%未満 :VB [15]	1
NOS [15]	スーパークラスが持つサブクラスの数	1 :B	2
NOAC [15]	クラスの祖先クラスの数	3-4 :LB, 5-6 :B, 6< :VB	3
LCOM [14]	メソッドの凝集の欠如	0.6-0.8 :LB, 0.4-0.6 :B, 0.4 :VB [14]	1
NPAM	クラスの Public アクセサメソッドの数	4-6 :B, 6< :VB	3
DAC [13]	データ抽象結合度 (あるクラスで用いられる参照型の数)	3-4 :LB, 5-6 :B, 6< :VB [15]	1
CBO [11] (NOR)	オブジェクトクラス間の結合度 (結合する他クラスの数)	6-7 :LB, 8-9 :B, 9< :VB [33]	1
NMR [22]	クラスが他のクラスから受け取るメッセージの数	0 :B	2
MPC [13]	メッセージパッシング結合 (クラスのメッセージ送信数)	36-63 :LB, 64-96 :B, 96< :VB	3
RIM [15]	サブクラスのメソッド総数における 継承したメソッドの割合	0.4 未満 :VB, 0.4-0.6 未満 :B, 0.6-0.8 未満 :LB [15]	1
RAM [15]	サブクラスのメソッド総数における 追加されたメソッドの割合	0.2-0.4 :B, 0.4< :VB [15]	1
DIT [11]	継承木の深さ	4-6 :B, 6< :VB [15]	1
NIA [22]	継承した属性数	0 :B	2

※ little bad:LB, bad:B, very bad:VB とする。

閾値定義例をいくつか記述する。文献 [4, 15, 20] において NOM メトリクスの閾値を 20, 50 および 60 としている。上記の考えに従い、NOM メトリクスの閾値を次のように設定する：20～30 の範囲は「little bad」、31～50 は「bad」、51 以上なら「very bad」。Lorenz らは NMSM の閾値を 9 と定義した [15]。また、Rosenberg は CBO の閾値を 5 と定義した [33]。この 2 つの閾値に基づき、NMSM の閾値を次のように設定する：9～11 の範囲は「little bad」、12～14 は「bad」、15 以上なら「very bad」。RAC メトリクスに関して、Lorenz ら [15] は、総クラスの 10～15 % が抽象であるべきであると提案した。したがって、RAC の閾値を次のように設定する：10～12 % は「little bad」、8～10 % は「bad」、8 % 未満は「very bad」。

Gr.2 は設計欠陥や設計規則の記述から明確である閾値である。ここでは例として 2 つの欠陥の閾値についてのみ取り上げる。欠陥 D38 はサブクラスがスーパークラスのデータを継承しないことを意味する。よって、NIA メトリクス「継承した属性の数」の閾値を 0 とする。欠陥 D43 は集約元クラスが集約先オブジェクトにメッセージを送信しないなら、集約関係である必要性がない [34]。よって、NMCCC の閾値は 0 とする。

Gr.3 は、Gr.1 にも Gr.2 にも属さないため、本

研究によって提案する閾値である。

NPABC と NPMBC メトリクスの閾値は専門家による設計ルールに基づいて設定する。Riel [29] はスーパークラスの全てのデータは private であるべきだと提案した。これに基づき、NPABC メトリクス (D41) の閾値は「bad」を 1～3、「very bad」を 4 以上とする。Webster [8] はスーパークラスがほとんど何もしないなら、それは明らかな汎化もしくはコードの再利用を意味しないと述べた。したがって、NPMBC メトリクス (D34) の閾値は「little bad」を 2、「bad」を 1、「very bad」を 0 とする。

欠陥 D15, D23, D28, D30 の説明に「大抵」のような記述がある。これらの欠陥を検出する NMNP, RMSSO, RMHC, および RMHSP メトリクスに関する閾値は、この記述に基づき設定する。例えば、NMNP メトリクスの閾値をメソッドの総数の 30～50 % の範囲は「little bad」、50～70 % は「bad」、70～100 % は「very bad」とする。また、欠陥 D13, D20 もその欠陥記述とメトリクス定義に基づいて同様に設定する。

残りの閾値は閾値 Gr.1 のメトリクスの閾値に基づいて設定される。例えば、NPM メトリクス (D8) の閾値は NOM, NPubM, および NPAM メトリクスの閾値に ($NPM \cong NOM - NPubM - NPAM$)、NOAC メトリクス (D10) の閾値は DIT

表 4: 新しいメトリクスの定義

Metric	概要	閾値	Gr.
NOIDC	サブクラスのインスタンス数	1 :B	2
NACUM	public メソッドを用いる他のクラスの数	0 :B	2
NMMI	メソッドが呼び出される回数	0 :B	2
NMNP	クラスのメソッド総数における 引数を持たないメソッド数の割合	30-50% :LB, 50-70% :B, 70% :VB	3
RTAO	クラスのメッセージ送信数における アクセスメッセージの割合	50-60%未満 :LB, 60-70%未満 :B, 70% $\leq$:VB	3
NTWD	異なるクラスにあるメソッド間の相互依存関係の数	1 :B, 1 $\leq$:VB	2
RMSO	メソッド送信総数における, 同一オブジェクトへの送信メッセージ数の割合	60-70%未満 :LB, 70-80%未満 :B, 80% $\leq$:VB	3
NSP	メソッドが持つ同一引数の数	0 $\leq$:B	2
RMHC	クラスのメソッド総数における, 高い複雑度を持つ メソッド (本研究では NMSM で判定) 数の割合	50-60%未満 :LB, 60-70%未満 :B, 70% $\leq$:VB	3
RMHSP	クラスのメソッド総数における, 同一引数を持つ メソッドの数の割合 (型の一致で判断)	50-60%未満 :LB, 60-70%未満 :B, 70% $\leq$:VB	3
NPABC	スーパークラスの public 属性の数	1-3 :B, 3 $\leq$:VB	3
NPMBC	スーパークラスの public/protected メソッドの数	2 :LB, 1 :B, 0 :VB	3
NSAIS	同一の親クラスを持つサブクラスの全てで宣言された 同一属性 (型名, 名前) の一致で判断) の数	0 $\leq$:B	2
NITBM	スーパークラスのメンバを継承した回数	1 :B	2
NGIS	サブクラスで定義された getter メソッドの数	(2 参照)	
NMCCC	集約元クラスから集約先オブジェクトへのメッセージの数	0 :B	2
NMOCC	同一クラスに集約されるオブジェクト間のメッセージ数	0 $\leq$:B	2
RMDC	委譲元クラス内のメソッド総数における, 委譲元クラスに用いられる委譲先クラスのメソッドの 割合 (委譲はメッセージ送信数とその引数から判断)	90-100% :B	2
ASRMU	スーパークラスのメンバ総数における各サブクラスで 用いた継承したメンバの割合の, 子クラスの平均	40-50%未満 :LB, 30-40%未満 :B, 30% $\leq$:VB	3
NSV	クラスのクラス変数の数	2-3 :B, 3 $\leq$:VB	3
NTAC	メソッドが自クラスの属性にアクセスする回数	0 :B	2

※ little bad:LB, bad:B, very bad:VB とする。

メトリクスの閾値に, NPAM メトリクス (D17) の閾値は NPA メトリクスの閾値に ($NPAM = NPA * 2$), NSV メトリクス (D15) の閾値は NPA メトリクスの閾値に基づいて設定する。MPC メトリクス (D25) の閾値は NPubM, NMSM, および NPAM メトリクスの閾値に基づいて設定する ($MPC \cong NPubM * NMSM + NPAM$)。MPC メトリクスでは「little bad」閾値を設定するため, 適切な範囲の最大値 (NPubM=4, NMSM=9) を用いた。「bad」は NPubM と NMSM の「little bad」閾値, 「very bad」は NPubM と NMSM の「bad」閾値を用いて設定する。

4 実験

実際に OO 設計から欠陥を検出し, 欠陥がどの程度検出できるかを検証する。以下に, 測定対象となる OO 設計のサンプルの概要と計測ツールの概要を記す。

4.1 サンプル

以下のグループに分けられた UML 図をサンプルとして用いる。

- Gr.A(10 種, クラス総数 96 個)
情報工学科の学生が初めて記述した UML 図
- Gr.B(7 種, クラス総数 35 個)
一般的なデザインパターンが解説されている文献 [35] に記述されている UML 図
- Gr.C(6 種, クラス総数 106 個)
Web や書籍などで公開されている UML 図

4.2 欠陥検出ツール

UML のクラス図とシーケンス図からメトリクス値を測定し, 欠陥を検出するツールを使用した。このツールの入力には描画ツールである Rational XDE によって XMI 形式で出力されたファイルである。クラス図とシーケンス図から今回測定する際に必要なほとんどの情報を得ることができるが, この 2 つからは属性へのアクセスがわからないため, D12, D13, D19-2 を検出する際に用いる LCOM, NTAC

表 6: 欠陥種類総数 (47 種中)

	little bad	bad	very bad
初心者サンプル	38	33	17
デザインパターン	20	17	5
公開サンプル	32	28	14
全サンプル	41	38	19

メトリクスを測定することができない。これらを検出するためにはアクティビティ図などのダイアグラムを追加する必要がある。今回は D13 のみ代替案によって検出できる可能性があるため、D13+を定義した (表 2)。

4.3 検出結果

総計 237 のクラスから表 3 および表 4 のメトリクスを収集し、little bad 以上、bad 以上、very bad の場合に検出される欠陥 (D12, D13, D19-2 を除く 47 欠陥) を調査した。表 6 に各グループの欠陥検出種類数を示す。また、各欠陥が主にどのグループで検出されたかを表 7 に示す。

5 検出結果考察

今回 3 つの閾値を用いて検出したが、欠陥 D2-1, D5, D9, D15-1, D27 はどの閾値を用いても検出サンプル数や検出数がほぼ変わらなかった。欠陥 D9 は今回用いているサンプルが、抽象化クラスを多く用いているグループ (特に Gr.B) と全く用いていないグループ (特に Gr.A) にほぼ二分されていたためである。欠陥 D2-1, D27 は測定の内容に対して閾値の幅が狭いためである。この欠陥は閾値に幅を持たせる必要があるかもしれない。欠陥 D5 も閾値の幅が狭いためであるが、この欠陥は幅を持たせるために閾値を変更するよりも、厳しく検出するために幅を狭くしたままである方がよいと考えられる。

little bad 以上を欠陥とした時、サンプル全 23 種中欠陥 D6, D9, D16 は 15 種以上で、欠陥 D3-1,

D3-2, D7, D19-1, D24 は 20 種以上で検出された。欠陥 D3-1, D3-2, D6, D16 が検出されたのは、Gr.B 全てと Gr.C の一部が教育用で小規模なものであったことが影響しているため、欠陥 D9 については Gr.A と Gr.C のほとんどが抽象クラスを用いていないためであると考えられる。また、欠陥 D19-1, D24 については数多く検出されており、大抵のサンプルはシーケンス図の記述が省略されていることがわかる。欠陥 D7 についてはそれに加え、初心者が全てのメソッドを public にしがちであることが影響していると考えられる。

表 7 に示されている全グループで検出された欠陥の多くは、システムが小規模であるか、シーケンス図の記述に不備があったために検出されている。全グループで検出されなかった欠陥は主に、今回用いたサンプルの規模では検出しにくいもの (欠陥 D10, D35) や、規模に関わらず検出しにくいもの (欠陥 D29, D41 等) である。欠陥 D39 については NOA, NOM の閾値が高過ぎる可能性があるので変更すべきかもしれない。Gr.A で主に検出された欠陥は、特に初心者が起こしやすい欠陥である。1 つのクラスに多くのメソッドを詰め込むなどして責務がうまく分けられていないため、他のクラスとの関連が強かったりメッセージ送信が複雑になるなどの欠陥が検出された。Gr.B 以外で主に検出された欠陥は、何回か設計を記述している人でも起こしやすい欠陥である。抽象クラスを用いていない、カプセル化がうまくできていないなどの欠陥が検出された。

6 まとめ

本研究では様々な文献から OO 設計欠陥を調査し、その欠陥をメトリクスと閾値を用いて検出するモデルを提案した。また、欠陥とメトリクスを対応させたリストを用いることで、検出結果から修正したい欠陥の問題箇所が容易に検出できる。実際にサンプルを用いて測定した結果、検出を試みた欠陥 47 種中 41 種を検出することができた。

表 7: 各 Gr で検出された欠陥

検出 Gr	欠陥
全 Gr 共通に検出	D3-1, D3-2, D6, D7, D9, D11, D14, D16, D19-1, D20, D21, D22, D24, D32, D33, D34, D43, D44
主に Gr.B 以外で検出	D1-1, D2-1, D2-2, D4, D6, D14, D15-2, D17, D26, D27, D28, D30, D31
主に Gr.A 以外で検出	D36, D38, D45
主に Gr.A で検出	D1-1, D1-2, D2-1, D2-2, D5, D8, D15-1, D17, D18, D20, D22, D23, D25, D26, D27, D40, D42
Gr.A だけで検出	D1-2, D5, D8, D15-1, D18, D23, D25, D40, D42
未検出	D10, (D12), (D13), (D19-2), D29, D35, D37, D39, D41

3.2節において、欠陥を設計特性に振り分けたが、我々が今回用いた欠陥検出アプローチやメトリクス、閾値は、階層を定義し、それらのリンクを明確にすることで、設計特性を評価するための十分な品質基準になることができると考えられる。すでに他の文献で今回用いたメトリクスを用いて品質評価を行う方法を検討している [38]。現在提案されている品質モデルには、早期ソフトウェアプロダクト品質モデルの一つで品質特性の3つのタイプに基づいてソフトウェアプロダクト特性を定める McCall の品質モデル [34] や、ソフトウェアプロダクト品質の測定のためのフレームワークである国際規格 ISO9126 [36]、Dromey モデル [31] を拡張した OO 設計における階層品質モデル [37] などが挙げられる。

メトリクスの閾値は一意に決められないため、メトリクス値が重大な状況を示すかどうか、ユーザ自身が判断する必要がある。そのため、メトリクス値がユーザの許容できる範囲かどうかを直観的に判断しやすいように、本研究では複数の閾値を設定した。将来的には複数の閾値を設定するのではなく、ファジー推論などを用いて設計欠陥の表示を行う予定である。

謝辞 本研究を進めるにあたり、測定ツールを作成していただいた高野純知さん、田村真吾さんに深甚なる謝意を表します。

参考文献

- [1] B. Boehm and V. Basili: Software Defect Reduction Top 10 list, IEEE-CS, 34(1), 135-137, 2001.
- [2] L. Briand, et al: A Unified Framework for Cohesion Measurement in Object-Oriented Systems, Empirical Software Engineering - An International Journal, 1998.
- [3] L. Briand, et al: A Unified Framework for Coupling Measurement in Object-Oriented Systems, IEEE-TSE, 25(1), 91-121, 1999.
- [4] M. Akroyd: AntiPatterns Session Notes, Object World West, 1996.
- [5] W.J. Brown, et al: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, 1998.
- [6] M. Fowler, et al: Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.
- [7] B. Meyer: Object-Oriented Software Construction - Second Edition, Prentice Hall, 1997.
- [8] B. Webster: Pitfalls Of Object Oriented Development, M&T Books, 1995.
- [9] J.M. Bieman and B.K. Kang: Cohesion and Reuse in an Object-Oriented System, in Proc. ACM SSR'94, 259-262, 1995.
- [10] L. Briand, et al: An Investigation into Coupling Measures for C++, Proc. of ICSE 97, 1997.
- [11] S.R. Chidamber, et al: A Metrics Suite for Object Oriented Design, IEEE-TSE, 21(3), 263-265, 1994.
- [12] Y.S. Lee, et al: Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow, in Proc. ICSQ, 1995.
- [13] W. Li and S. Henry: Object-Oriented Metrics that Predict Maintainability, J. Systems and Software, 23(2), 111-122, 1993.
- [14] M. Hitz and B. Montazeri: Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective, IEEE -TSE, 22(4), 267-270, 1996.
- [15] M. Lorenz, et al: Object-Oriented Software Metrics: A Practical Guide, Prentice Hall, 1994.
- [16] D.P. Tegarden, et al: A Software Complexity Model of Object-Oriented Systems, Decision Support Systems, 13(3-4), 241-262, 1995.
- [17] H. Washizaki: Software Component Metrics and its Empirical Evaluation, Proc. of ISESE, Vol. II, 2002.
- [18] V.R. Basili, et al: A Validation of Object-Oriented Design Metrics as Quality Indicators, IEEE-TSE, 22(10), 751-761, 1996.
- [19] L. Briand, et al, Exploring the Relationships Between Design Measures and Software Quality in Object Oriented Systems, J. Systems and Software, 51(3), 245-273, 2000.
- [20] K. Erni, et al, Applying Design-Metrics to Object-Oriented Frameworks, 3rd IEEE METRICS, 1996.
- [21] C. Kirsopp, et al: An Empirical Study into the Use of Measurement to Support OO Design Evaluation, 6th IEEE METRICS, 1999.
- [22] The Software Design Metrics tool for the UML [online], Available from: www.sdmmetrics.com/ [Accessed 28/09/2007].
- [23] JStyle [online], Available from: www.codework.com/JStyle/product.html, [Accessed 28/09/2007].
- [24] Ch. Munkhnasan, Y. Ueda: Detecting Defects in Object-Oriented Designs Using Design Metrics, JCKBSE, 2006.
- [25] CodeSmell [online], Available from: c2.com/cgi/wiki?CodeSmell [Accessed 28/09/2007].
- [26] J. Eder, et al: Coupling and Cohesion in Object-Oriented systems, Technical Report, University of Klagenfurt, 1994.
- [27] E. Gamma, et al: Design Patterns, Elements of reusable Object-Oriented Software, Addison-Wesley, 1994.
- [28] R.E. Johnson and B. Foote: Designing reusable classes, JOOP, 1(2), 22-35, 1988.
- [29] A.J. Riel: Object Oriented Design Heuristics, Addison-Wesley, 1996.
- [30] S.A. Whitmire, Object-Oriented Design Measurement, John Wiley & Sons, 1997.
- [31] R.G. Dromey: A Model for Software Product Quality, IEEE-TSE, 21(2), 146-162, 1995.
- [32] S. Benlarbi, et al: Thresholds for Object-Oriented Measures, 11th ISSRE, 2000.
- [33] L. Rosenberg, et al, Object-Oriented Metrics for Reliability, 6th IEEE METRICS, 1999.
- [34] J.A. McCall, et al: Factors in Software Quality, vols. 1-3, AD/A-049-014/015/055 NTIS, 1977.
- [35] 結城 浩, 増補改訂版 Java 言語で学ぶデザインパターン入門, ソフトバンククリエイティブ, 2004.
- [36] Software Product Evaluation - Quality Characteristics and Guidelines for their Use, ISO/IEC Standard ISO-9126, 1991.
- [37] J. Bansiya and G. Davis: A Hierarchical Model for Object-Oriented Design Quality Assessment, IEEE-TSE, 28(1), 4-17, 2002.
- [38] 佐藤美穂, 田村真吾, 上田賀一: UML 設計を対象とした品質評価モデルの検討, 情報処理学会論文誌, vol. 49, No. 7, 2008.