解釈を変更可能なプログラムソースコード保守性評価ツールの実現と適用

Implementation and application of a program source code conservativeness evaluation tool which can change an interpretation

早稲田大学大学院,基幹理工学研究科
Waseda University Graduate School of Fundamental Science and Engineering
株式会社小松製作所,開発本部,ICT 開発センタ
Komatsu Ltd. Development Division, ICT Development Center

○高田 正樹[1)] 鷲崎 弘宜[1)] 遠藤 匠[1)] 佐藤 雅宏[2)] 杉村 俊輔[2)] 関 洋平[2)]
○Masaki Takada[1)] Hironori Washizaki[1)] Takumi Endou[1)] Masahiro Sato[2)]
Shunsuke Sugimura[2)] Yohei Seki[2)]

Abstract As a technique for measuring the quality characteristic of software, there is a method of evaluating quality which conducted static code analysis using the metrics drawn in the form connected to the goal which should be made clear on quality based on the Goal-Question-Metric (GQM) method. However, if the characteristic that a developer minds for every domain or project may differ from each other and the quality evaluation of two or more projects (especially domains differ) is carried out by the same GQM model, the evaluation which is different from a developer's viewpoint or the characteristic of a domain may be obtained.

Then, after including a developer's intention in a GQM model, it is desirable to carry out quality evaluation, but in the quality evaluation tool for many source codes, the GQM model given a definition is easily unchangeable. Moreover, the influence on the evaluation result in the real project at the time of changing the evaluation result based on the measurement result of metrics according to a domain, the characteristic for every project, or a developer's consideration is not clear. Then, in evaluating the quality of a source code from the measurement result of the metrics in a GQM model, we implement the program source code conservativeness evaluation tool which made variable the interpretation of the measurement result according to the characteristic of a domain, or a developer's viewpoint. Furthermore, we checked that some evaluation results changed by comparing the case where the standard quality evaluation for Adqua and the case where the conservativeness evaluation tool is applied, after setting up interpretive procedure through the hearing of a development pursuer.

1. はじめに

ソフトウェアの品質特性を測定するための手法としては、ソースコードを静的解析し、Goal-Question-Metric (GQM) 法[1]に基づいて品質上の明らかとすべきゴールに結びつけられた形で導出したメトリクスを用いて品質を評価する方法がある。しかしながら、ドメインやプロジェクト毎に開発者が留意する側面や特性が異なることがあり、複数の（特にドメインの異なる）プロジェクトを同一の GQM モデルで品質評価すると、開発者の観点やドメインの特性から離れた評価を得てしまう可能性がある。そこで開発者の意図を GQM モデルに組み込んでから品質評価す

1) 早稲田大学大学院基幹理工学研究科
Waseda University Graduate School of Fundamental Science and Engineering
東京都新宿区大久保 3-4-1
3-4-1, Okubo, Shinjuku, Tokyo, Japan
2) 株式会社小松製作所 開発本部・ICT 開発センタ
Komatsu Ltd. Development Division, ICT Development Center
神奈川県平塚市四之宮 3-25-1
3-25-1 Shinomiya, Hiratsuka-shi, Kanagawa, Japan

ることが望ましいが、ソースコードを対象とした多くの品質評価ツールにおいては、定義済みの GQM モデルを容易に変えられない。また、ドメインやプロジェクト毎の特性や開発者が留意する側面に応じて、メトリクスの測定結果に基づく評価結果を変えた場合、その実プロジェクトにおける評価結果への影響は明らかとなっていない。

そこで我々は、GQM モデルにおけるメトリクスの測定結果からソースコードの品質を評価するにあたり、ドメインの特性や開発者の観点に応じて測定結果の解釈（インタプリテーション）を可変としたプログラムソースコード保守性評価ツールを実現する。本ツールでは、既存のソースコード品質評価ツール Adqua の後処理系として実現する。以降において、2 節で背景と研究課題を説明し、3 節でツールでの適用方法、4 節でツールを建機メーカの 2 つの組み込みソフトウェアに適用した結果および得られた知見を説明し、5 節で関連研究について言及したのち、6 節で結論と展望を述べる。

2. GQM 法と研究課題

ソフトウェアの品質測定として「GQM 法」が使われる。GQM 法とは、ゴール(Goal)、 クエスチョン(Question)、メトリクス(Metrics)のそれぞれの頭文字を合わせたものであり、無目的にメトリクスやデータを集めるのでなく測定の目的(ゴール)は何であるかを決めて、その目的の達成のために評価すべき質問(クエスチョン)を定義し、その質問に対する答えを提供する情報を測定するメトリクスを定めるものである。例えば、ゴールとして「保守性のうち、解析性について調べたい」と設定するとする。そのゴールを達成するために「可読性の良いコードになっているか」や「抽象化が適切に行われているか」など様々なクエスチョンが考えられる。それらのクエスチョンのうち「抽象化が適切に行われているか」という質問に対して答えを提供する情報として、「メソッドの数」や「実行コード行数」などのメトリクスが必要になることがわかる。このように、測定の目的を達成するためのメトリクスを決めることがソフトウェアの品質評価にとって重要な点である。

従来のソースコード品質評価ツールにおける GQM モデルは、例えば Adqua の場合、Adqua の開発元が組込み開発の様々なドメイン・組織から収集した 270 以上のプロジェクトデータから定めている。Adqua はプロジェクトのソースコードを静的解析し、メトリクス毎に測定値を求め、それを元に組み込みソフトウェアの品質を品質特性の観点（信頼性、効率性、保守性、移植性、再利用性）で得点化している。他社プロジェクトに対する相対的な品質レベルを把握するために、このような網羅的な品質評価を実現している。

しかしながら、その GQM モデルは開発者の意図に基づいていないので、特定の組織やドメインの特定のプロジェクトにおいて適当とは限らない。例としては、ソースコード中に現れる 0 と 1 以外のリテラルをマジックナンバというが、マジックナンバは何の数値であるかがソースコード中に明記されないがために、コードの可読性が下がり保守性の低下を招くという問題がある。Adqua でもマジックナンバの使用は保守性の低下につながると解釈しているが、開発者によってはソースコードのマジックナンバにはコメントで補足されている場合がある。コメントで補足されたマジックナンバはコードの可読性を下げることはないので、評価の対象から除外してほしいという開発者の意図が出てくる。

それゆえ、メトリクスの測定結果からソースコードの品質を評価するにあたり、ドメインの特性や開発者の意図に応じて測定結果の解釈（インタプリテーション）を可変にすることが必要である。

そこで本研究では以下の 2 つの Research Question を掲げる。

RQ1. ツールによって容易に GQM に基づいた品質評価の方法を変更できるか

RQ2. 品質評価結果から提示された改善案は現実的で妥当なものか

3. 解釈を変更可能な品質評価ツール

3.1 解釈の変更

解釈を伴う GQM モデルの概要を表 1 に示す。我々の保守性評価ツールにおいては、表 1 におけ

る解釈の部分を可変としており、開発者が利用にあたってドメインやプロジェクトの特性を加味して閾値の変更や評価式そのものを変更可能とすることを実現した。例えば表 1 においてゴール G1 の指針 Q2 に関する評価にあたり、Q2 の指針においてはメトリクス M3 の値が XX 以上であるとしているところを、閾値 XX の値を変更したり、あるいは、評価式そのものを新たなもの（例えば M4=0）に変更したりすることができる。

表 1: 解釈を伴う GQM モデルの例

Goal 品質	Question 指針	Metric メトリクス	Interpretation 解釈	Result 評価結果
G1: ・・・	Q1: ・・・	M1: ・・・ M2: ・・・	M1> 0 or (M1 =0 and M2 > 0)	NG
	Q2: ・・・	M3: ・・・	M3 >= XX	NG

3.2　ツールでの適用

(1) 解釈の変更方法

解釈の容易な変更の実現のため、我々は Adqua の後処理系としての保守性評価ツールを、診断ツールと Config ツール（図 1）の 2 つのツールから構成させることとした。Adqua はソースコードを静的解析した後、点数をつけることで評価するが、本ツールでは Adqua での中間成果物であるメトリクスとその測定値だけを使用する。

開発者は Config ツールを用いて Question 単位で評価式（閾値を含む）を定義する。診断ツールは、Adqua の測定結果に対して、定義した解釈式に基づいて解釈し、その結果を最終的に図 2,3 に示すレポートファイルが自動生成される。開発者や品質保証従事者は、これらの出力されるレポートを参照することで、ソースコードにおいてドメインやプロジェクトの特性を加味したうえで保守上改善すべき観点（指針）や箇所（ファイル）を容易に特定できる。

図 1: Config ツールの利用例（Question 2 に対する利用例）

(2) 解釈（インタプリテーション）の構成

我々はメトリクスの測定値の解釈を、可変な「ソースコードメトリクス」「閾値」「評価式の組み合わせ」の組み合わせで構成することとし、Questionに対応付けることで最終的にGoalの達成判断を可能とした。

解釈を設定するにあたって、まず始めに我々は評価対象のプロジェクトにおいて、Adquaの評価

結果で点数の低かったメトリクスに着目した。それらのメトリクスにたいして、どのような理由があって評価が低くなったのかを調べるために、対象プロジェクトのアーキテクトにヒアリングを行った。ヒアリングの流れとしてはメールベースで大まかな設計意図を把握し、それを参考にしてAdquaの結果の評価の低い部分をもう一度見直し、開発者の意図となりうる部分を開発者と共に模索した。その結果、ソースコードの保守性評価について開発従事者のヒアリングを通じて10個のQuestionを設定し、それらを24個のメトリクスと閾値、評価式の組み合わせにより評価可能とした。その一部として、2個のQuestionとそれに対応するメトリクスと評価式の関係を表2に示す。

表2　Questionとそれに対応するメトリクスと評価式

ID	Question	Metrics
2	1 ファイルの責務の量は適切か	MFl004 ステートメント数
		MFl019 関数の数
		MFl477 実行コード行数
	(MFl004 ≦ 100 || MFl019 ≦ 10 || MFl477 ≦ 150) && (MFl004 < 300 && MFl019 < 25 && MFl477 < 300)	
5	関数の処理が複雑すぎないか	MFl004K ファイル内の制御フローグラフの最大ネスト数の平均値
		MFl005K ファイル内の制御フローグラフの最大ネスト数の最大値
		MFl006K ファイル内のサイクロマティック複雑度の平均値
		MFl007K ファイル内のサイクロマティック複雑度の最大値
	(MFl004K < 2 || MFl005K ≦ 4 || MFl006K ≦ 3 || MFl007K < 10)&&(MFl004K < 2.8 && MFl005K ≦ 8 && MFl006K ≦ 5.5 && MFl007K < 28)	

3.3　解釈の例

解釈の例として、Question2 の「1 ファイルの責務の量は適切か」を挙げる。この Question を評価するためのメトリクスは 3 つであり、「MFl004 ステートメント数」「MFl019 関数の数」「MFl477 実行コード行数」となっている。Adqua ではこの 3 つを含む関連したメトリクスをファイル毎に得点化し、平均値を求める。その平均値が一定以上であれば、問題はないという判断ができる。しかし、開発者にとって優先的に直すべきファイルは、3 つのメトリクス全てが基準値を満たさない、もしくは 3 つのメトリクスのどれかが極端に基準値を超えたものであるということを、ヒアリングを通して知ることができた。このような開発者の意図をインタプリテーションとして組み入れるため、式 1 として定義した。

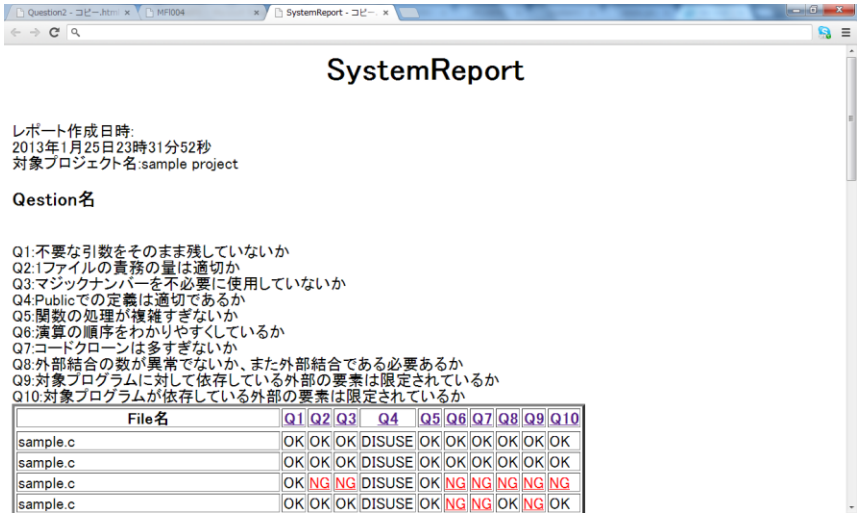
(MFl004 < X1 || MFl019 < X2 || MFl477 < X3) && (MFl004 < Y1 && MFl019 < Y2 && MFl477 < Y3)　(式 1)

ここでの X1,X2,X3 は、MFl004,MFl019,MFl477 のそれぞれの基準値であり、Y1,Y2,Y3 はそれぞれの極端な異常値である。この式を満たすようなファイルであれば、判定は「OK」として出力される。逆にこの式を満たしていない時、判定は「NG」として出力される。

今回ツールを適用するにあたり、X1～3 のような基準値と Y1～3 のような極端な異常値については、対象プロジェクトのメトリクス測定値を元に定義した。メトリクス測定値それぞれのヒストグラムを見比べ、上位 7 割を基準値、下位 1 割を極端な異常値として設定した。

3.4　診断ツールの出力

診断ツールの出力は html ファイルと csv ファイルの 2 種類である。html ファイルはシステム全体のレポートと Question 毎のレポートとなる。図 2 にシステム全体のレポート、図 3 に Question 単位のレポートの例を示す。システム全体のレポートでは、全てのファイルと Question の関係を「OK」or「NG」で見ることができる。また使われていない Question については「DISUSE」と表示される。一方 Question 毎のレポートは、その Question で NG となったファイルの一覧を見ることができる。Question で使われた各メトリクスについて、「true」or「false」で条件を満たしているかを見ることができる。「false」には赤色と青色の 2 種類があり、赤色が NG と評価される原因となるもので、青色は NG と評価される直接の原因とならないものを表す。



SystemReport

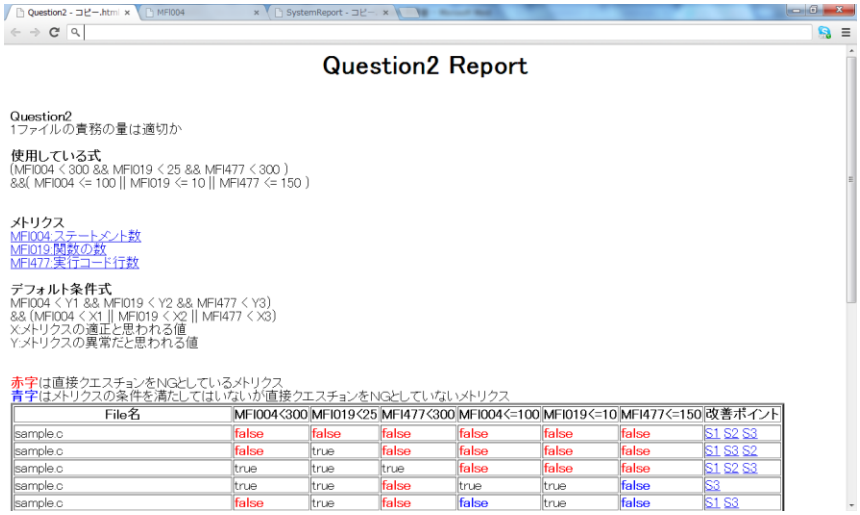
レポート作成日時:
2013年1月25日23時31分52秒
対象プロジェクト名:sample project

Qestion名

Q1:不要な引数をそのまま残していないか
Q2:1ファイルの責務の量は適切か
Q3:マジックナンバーを不必要に使用していないか
Q4:Publicでの定義は適切であるか
Q5:関数の処理が複雑すぎないか
Q6:演算の順序をわかりやすくしているか
Q7:コードクローンは多すぎないか
Q8:外部結合の数が異常でないか、また外部結合である必要あるか
Q9:対象プログラムに対して依存している外部の要素は限定されているか
Q10:対象プログラムが依存している外部の要素は限定されているか

File名	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
sample.c	OK	OK	OK	DISUSE	OK	OK	OK	OK	OK	OK
sample.c	OK	OK	OK	DISUSE	OK	OK	OK	OK	OK	OK
sample.c	OK	NG	NG	DISUSE	OK	NG	NG	NG	NG	NG
sample.c	OK	OK	OK	DISUSE	OK	NG	NG	OK	NG	OK

図 2 生成されたレポート（システム全体、ただし図中ではファイル名を一律に sample.c としてある）



Question2 Report

Question2
1ファイルの責務の量は適切か

使用している式
(MFI004 < 300 && MFI019 < 25 && MFI477 < 300)
&&(MFI004 <= 100 || MFI019 <= 10 || MFI477 <= 150)

メトリクス
MFI004:ステートメント数
MFI019:関数の数
MFI477:実行コード行数

デフォルト条件式
MFI004 < Y1 && MFI019 < Y2 && MFI477 < Y3)
&& (MFI004 < X1 || MFI019 < X2 || MFI477 < X3)
X:メトリクスの適正と思われる値
Y:メトリクスの異常だと思われる値

赤字は直接クエスチョンをNGとしているメトリクス
青字はメトリクスの条件を満たしてはいないが直接クエスチョンをNGとしていないメトリクス

File名	MFI004<300	MFI019<25	MFI477<300	MFI004<=100	MFI019<=10	MFI477<=150	改善ポイント
sample.c	false	false	false	false	false	false	S1 S2 S3
sample.c	false	true	false	false	false	false	S1 S3 S2
sample.c	true	true	true	false	false	false	S1 S2 S3
sample.c	true	true	false	true	true	false	S3
sample.c	false	true	false	false	true	false	S1 S3

図 3　生成されたレポート（Question 単位）

4.　ツールを適用した結果とその知見

我々は、実現した保守性評価ツールの検証を目的として、ある建機メーカにおける実際の 2 つの組込みソフトウェア開発プロジェクトを対象として適用実験を行った。具体的には、同メーカにおける開発従事者のヒアリングを通じて解釈を設定したうえで、保守性評価ツールを適用した場合と、元の Adqua 標準の品質評価を行った場合との比較により、ソースコードのいくつかの側面について評価結果がいくらか変更することを確認した。対象プロジェクト p1, p2 の特徴を表 3 に示す。

表 3: 評価対象プロジェクトの特徴

プロジェクト	p1	p2
開発言語	C のみ	C, C++
ソースコード行数	約 5 万	C:約 2.5 万, C++:約 15 万
総ファイル数	100 以下	C:100 以下, C++:500 以下

4.1 ツールでの診断結果と Adqua での評価

我々は、開発従事者のヒアリングを通じて解釈方法を設定したうえで同保守性評価ツールを適用した場合と、元の Adqua 標準の品質評価を行った場合との比較により、ソースコードのいくつかの側面について評価結果がいくらか変更することを確認した。比較の例として、図 1,3 に示した Question2 に対する各ファイルの評価結果の推移を図 4 に示し、Question5 に対する各ファイルの評価結果の推移を図 5 に示す。図 4,5 において O (N) →O (N) とは、「O」が「OK」の略で「N」が「NG」の略となっており、Adqua 標準の解釈により Question が満足される（されない）と判断され、新たな保守性評価ツールを用いた解釈により Question が満足される（されない）と判断されたことを表す。従って、「O→N」に該当するファイルとは、Adqua 標準では問題ない（OK）と判断されていたものの、実際にはドメインやプロジェクトの特性を加味した解釈により問題あり（NG）と判断されるものである。逆に「N→O」に該当するファイルとは、Adqua 標準では問題あり（NG）と判断されていたものの、実際にはドメインやプロジェクトの特性を加味した解釈により問題なし（OK）と判断されるものである。また、今回 Adqua 標準の評価方法としては、診断ツールのインタプリテーションの式に使用するメトリクスの点数を調べ、その平均値が 75 点以上なら「OK」、75 点未満なら「NG」と判断した。

図 4 から見られるように、Question2 において元の Adqua 評価結果が NG であったファイルが、いくつか OK へと推移したことがわかる。これは開発者の意図によって基準が緩くなったためである。プロジェクト p1 では NG と評価されたファイルの半分が本ツールによって OK へと推移した。プロジェクト p2 においても、NG から OK へと評価が変化したファイルあることがわかった。

また、図 5 においても Adqua 評価結果から本ツールによって評価が NG から OK となったファイルが多数確認できた。p1 については約 6 割、p2 については約 5 割ものファイルが OK へと変化していたが、これは元々NG となっていたファイルが多く、閾値を Adqua 準拠のものから対象プロジェクト準拠のものに変えたのが大きな要因である。

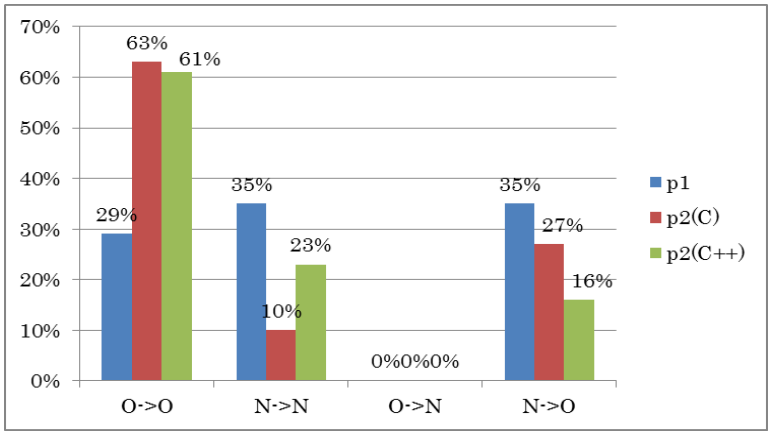


図 4 保守性評価ツール適用前後における評価結果 OK/NG ファイル数の推移の割合（Question 2 のみの抜粋）

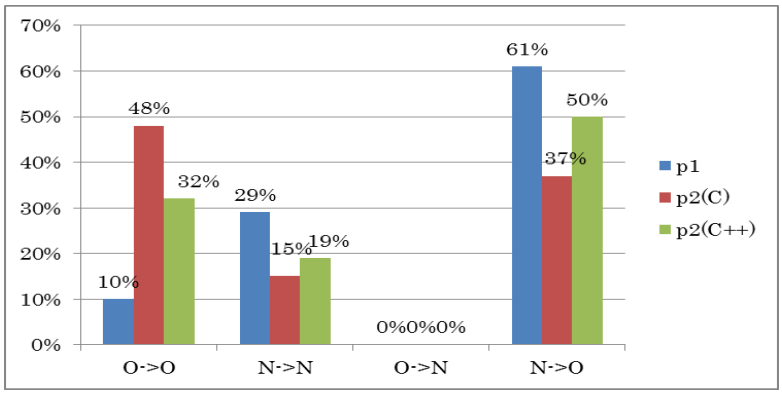


図 5 保守性評価ツール適用前後における評価結果 OK/NG ファイル数の推移の割合（Question 5 のみの抜粋）

4.2 妥当性評価

我々は、これらの結果を用いてプロジェクトの開発者へ再度ヒアリングを行った。それによりp1とp2の2つのプロジェクトの開発者から下記の意見を得た。

p1については、p1の機能に直結するファイルとNG→OKと評価が変化したファイルに着目したが、ほとんどのファイルが閾値の影響を大きく受けていた。したがって、ファイルの規模に関係しない別のメトリクスとの組み合わせにより評価式を定義することで、より良い評価結果が得られるのではないかという意見が得られた。また、評価対象もファイル単位ではなく関数単位レベルで行えると、より適切な粒度で評価できるとの意見も得られた。本手法の良い点としては、開発チーム内で改善要求があるファイルが数値化されNGと診断されることで、開発サイドとして改善すべきであると視覚的に分かる事と、本手法によって得られた評価が次期開発の際のコード品質向上に役に立つ情報になり得る事であった。しかし、派生開発における差分の無い親コードについては評価しても修正に多くの時間と大きなリスクが関わるので、改善することができないという問題も存在した。

p2については、プロジェクトの規模が大きかったため、p2に含まれる十数個のドメインのうちの一つに着目してヒアリングを行った。着目したドメインはプロジェクトの中核となるドメインであり、また評価ツールによる結果変化も顕著だった。

まず、Q2でNG→OKと変化したファイルに関して、次の意見を得た。IF（インタフェース）と呼ばれている、他ドメインから呼び出し可能なAPIのような役割を担うファイルの多くが、NGからOKへと変わっていた。これらは「MF1019 関数の数」の得点は低かったものの、「MF1004 ステートメント数」「MF1477 実行コード行数」の得点は低くはなかった。IFは他ドメインへ値の提供をするという特性上、関数の数は提供する値の数に比例して大きくなってしまう。ただし値の数が増えるからといって、クラスやファイルを分割してしまうと、ファイルの役割が曖昧となりコードの可読性は損なわれてしまう。Adquaの評価ではNGとなってしまっていたが、極端な異常値と閾値を分割した考えを持つことにより、このようなIFの特性が認知される形となり、意図通りの結果となったといえる。ヒアリングドメインにはNGだったIFが4個あったが、そのうちの3個がOKとなった。残りの1個に関しては、関数の数が極端な異常値を超えていたためNGとなったが、ステートメント数と実行コード行数の両方も閾値を超えていたことから、関数の数以外にも問題を抱えていることが分かり、極端な異常値と閾値の関係性がうまく保たれているといえる。

次に、Q2のメトリクス項目6つがすべてFalseとなったファイルに関して、次の意見を得た。ヒアリングドメインの処理の中核を担っているファイルが共通して全項目Falseとなっている。これらのファイルに関しては処理が多いことからコードの規模が大きくなってしまうのは避けられなく、処理の分離についても上位クラスを作成するなどして出来る限り対策もしている。このようなファイルに対して、処理が多いから単にNGとするのではなく、また別のアプローチから判定できればより良い。

最後に新たな Question として、コードが長くても複雑度が低ければ OK、コードが短くても複雑度が高ければ NG などができれば、より良いかもしれないとの意見を得た。Question の追加や評価式の変更はツールにより簡単に行えるので、閾値の決定ができ次第追加し、検証していきたい。

5. 関連研究

門田ら[2]の研究では、ソフトウェアツール EPM(発見的プロジェクト監視)からのデータを生かし、そして文脈と各プロジェクトおよび組織の成功基準の仮説（Hypothesis）に基づき、解釈（Interpretation）モデルのテーラリングを可能とするため、多様なプロジェクトと組織に対する GQM プロジェクト監視モデルをカスタム化する。本研究では開発者の意図を品質評価に取り入れるために、GQM モデルに解釈モデルを追加した。

楠本ら[3]の研究では、ソフトウェアの開発において定量的評価を導入した開発とそうでない開発でその成功率に差があるというデータを示したうえでその実現が重要であると述べている。そのうえで挙げられる問題を解消するために、定量的評価の導入のチュートリアルを担う形で言及している。GQM パラダイムという手法では測定するメトリクスとその目的を明確化することで何のために計測されたデータなのか、何のために収集されたデータなのかということを確認することを容易にしている。本研究においても、何のために、という Question を設定することでそれを測定するためのメトリクスを変更、拡張という形で作成した。

6. 結論と今後の展望

診断ツールの適用の結果により、RQ1 のツールによって容易に GQM に基づいた品質評価の方法を変更できるか、というクエスチョンに対しては解釈の変更のためのツールを作成し、図 4、5 にあるようにツール適用前後でその OK/NG の結果が変わったことから品質評価の方法を変更できたことが分かる。このことから RQ1 に対しては十分な答えを得られたといえるだろう。

一方、RQ2 の品質評価結果から提示された改善案は現実的で妥当なものかというクエスチョンに対しては、レポート形式で Question を NG としている原因のメトリクスを含む品質評価結果を得ることはできたものの、具体的な改善案の提示とは言い難い。また、ファイル単位でなく関数単位レベルの評価ができれば、より具体的に問題箇所を提示できるはずである。しかしながら、本手法で評価された OK/NG については、Adqua での結果よりも多くのファイルを救済することができ、妥当性も損なわれていなかった。

今後の展望としては、今回取り扱った p1,p2 以外のプロジェクトに対して検証を繰り返していくことで解釈の変更のデータを集め、プログラムソースコードの保守性評価の枠組みに対して有用性を高めることでより効率的な品質評価の貢献につながると考えている。また、開発者の意図から新しい Question を定義し、それについても繰り返し検証していくことで有用性を高めていきたい。Question を増やすことで様々な意図を含んだ評価が可能となると考えている。

今回のヒアリングの中に、閾値を変更し重要な NG 項目を浮かび上がらせるためにも、閾値と評価結果のリアルタイム応答性がもっと高いほうが良いという意見があった。閾値をスライダにしてスライダ値によって評価結果や中間成果物に対しリアルタイムにフィルタリングをかけ、スライダ変化によって OK/NG がヒートマップのように移り変わるようにするなど、本ツールを改善していくことも考慮していきたい。

参考文献

[1] Victor R. BASILI:“Using Measurement to Build Core Competencies in Software”, Seminar sponsored by Data and Analysis Center for Software, 2005.

[2] Akito MONDEN,Tomoko MATSUMURA,Mike BARKER,Koji TORII,Victor R. BASILI:“Customizing GQM Models for Software Project Monitoring”, IEICE TRANSACTIONS on Information and Systems, Vol.E95-D No.9 pp.2169-2182, 2012.

[3] 纐纈伸子,川村真弥, 野村准一, 野中誠:“プロセスおよびプロダクトメトリクスを用いた Fault-Prone クラス予測の適用事例”,ソフトウェア工学研究会報告 2010-SE-168(6) 1-8, 2010.