メトリクス計測プラグインプラットフォームMASUの開発

三宅達也[†]　肥後芳樹[†]　井上克郎[†]

ソフトウェアメトリクスとは，ソフトウェアのさまざまな特性を評価する尺度である．近年，ソフトウェアの大規模化・複雑化に伴い，ソフトウェアの特性評価はますます重要になり，さまざまなソフトウェアメトリクスが提案されている．しかし，メトリクスの計測は大きなコストを必要とする．とくにソースコードを対象としたメトリクスは計測のためにソースコード解析を行う必要があるが，ソースコード解析器の実装は容易ではない．また，計測するメトリクスが異なれば，ソースコード解析器が解析しなければならない情報は異なる．さらに，通常ソフトウェアメトリクスは，ファイルやクラス，メソッドといったソフトウェアの概念的な要素に対して定義されるものであり，同じ概念を共有する言語に共通して適用できるが，ソースコード解析器は言語ごとに用意しなければならない．このため，メトリクスを計測するツールの多くは単一の言語を対象としており，異なる言語で記述された複数のソフトウェアから統一的にメトリクス値を計測することは難しい．そこで我々は，複数の言語に対して適用可能であり，かつ，必要最低限のビジネスロジックを記述するだけでメトリクスの計測を可能とするメトリクス計測プラグインプラットフォーム MASU を開発している．本稿ではMASU の概要と今後の展望について述べる．

MASU: Metrics Assessment plugin-plutform for Software Unit of multiple programming languages

Tatsuya Miyake,[†] Yoshiki Higo[†] and Katsuro Inoue[†]

Software metrics are quantitative measures of software quality. Recently, software evaluation is getting an important role because the size and complexity of software systems are getting increased. Software metric is a tool for evaluating software systems. However, metrics measurement requires substantial cost because it needs source code analysis; developing a source code analyzer is an arduous task. Moreover, each metric requires different information for computing it. Additionaly, software metrics are measured from conceptual units of software systems such as file, class, and method ,so that it can be applied to multiple programming languages sharing the same consept. However, source code analyzers have to be developed for each programming languages because the source code written in different language have different grammers. Almost existing tools target only a single programming languages, which makes it difficult to measure software metrics from multiple programming languages in a unified way. In this paper, we introduce a software platform, MASU (Metrics Assesment plugin platform for Software Unit of multiple programming languages). The platform enables us to measure software metrics only by implementing the bussiness logics.

1. はじめに

ソフトウェアメトリクスとは，ソフトウェアの品質評価に用いられる尺度であり[8)]，ソースコードなどのソフトウェアプロダクトから計測される．ソフトウェアメトリクスには単純にソースコードの行数を表すLOC から，オブジェクト指向言語におけるクラスやメソッド，フィールド間の関連性を評価する CK メトリクス[4)]，関数やメソッド内の処理の複雑さを表すサイクロマチック数[7)] などのさまざまな種類のメトリクスが存在する．これらのメトリクスは，ファイルやクラス，メソッド，関数，制御フローといったソフトウェアの概念的な要素に対して定義されているため，言語間の記述様式の差異にとらわれることなく，同じ概念を共有する言語に共通して適用することができる．

しかし，メトリクスを計測するために，ソースコードを解析しソフトウェアの概念的な要素の情報を取得するには大きなコストを必要とする．これまでに種々のコンパイラコンパイラが開発されているが，それらが支援しているのは抽象構文木構築などの構文解析までであり，変数の参照・代入や関数の呼び出しなど，

† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

どのようにプログラム内の要素が関連しているかを調査するための意味解析を行うためには，開発者自らが実装しなければならない．また，メトリクスを計測するツールの多くは単一言語を対象としているため[5]，さまざまな言語のソースコードのメトリクスを計測するには，それぞれの言語に対応した計測ツールを個別に用意する必要がある．そのような場合，個々のツールが対応しているメトリクスの種類が異なっていたり，同一のメトリクスであっても定義が曖昧な部分の解釈や計測方法などが異なることがあり，複数の言語から統一的にメトリクス値を計測することは難しい．これらのことを踏まえると，以下の特徴をもつメトリクス計測ツールが必要であるといえる．

- 複数のプログラミング言語のソースコードに対して適用可能である．
- 多言語の解析から得た結果を統一的に扱える．
- ユーザは，必要最低限のビジネスロジックを記述するだけで，新たなメトリクスを計測可能である．

本稿では，これらの要求を満たすべく我々が開発しているメトリクス計測用プラグインプラットフォーム MASU を紹介する．

2. 背　　景

2.1 ソフトウェアメトリクス

ソフトウェアメトリクスとは，ソフトウェアの品質評価や工数・保守コスト予測などに用いられる尺度である[8]．代表的なソフトウェアメトリクスには，オブジェクト指向言語におけるクラスやメソッド，フィールド間の関連性からソフトウェアの複雑度を評価する CK メトリクス，関数やメソッド内の処理の複雑さを表すサイクロマチック数などが存在する．

2.1.1 CK メトリクス

CK メトリクスは，Chidamber と Kemerer によって提案されたオブジェクト指向設計を対象とする複雑度メトリクスである[4]．Basili らによって，従来のコードメトリクスの組み合わせよりもフォールトの発生を予測するための良い指標となることが示されている．次に示す 6 つのメトリクスにより構成されている．

WMC (Weighted Methods per Class)　クラスあたりの重み付きメソッド数．通常，重み付けにはサイクロマチック数[7] やソフトウェアサイエンス[6] が用いられるが，厳密な定義はされていない．

DIT (Depth of Inheritance Tree)　計測対象のクラスの継承木内での深さ．多重継承が可能である場合は，継承木における計測対象クラス (を表す節点) からそれ以上基底クラスが存在しないクラス (根) に至る最長経路の長さとなる．

NOC (Number Of Children)　計測対象クラスから直接派生しているサブクラスの数．

CBO (Coupling Between Object classes)　計測対象クラスが結合しているクラスの数．クラス間の結合とは，あるクラスが他のクラスの属性やメソッド，インスタンス変数を使用していることを意味する．

RFC (Response For a Class)　計測対象クラスに対する参照．ローカルメソッドの数とリモートメソッドの数で定義される．ローカルメソッドは計測対象のクラスに定義されているメソッド，リモートメソッドは計測対象のクラス以外で定義されているメソッドのうちローカルメソッドから呼び出されているメソッドである．

LCOM (Lack of Cohesion in Methods)　メソッドの凝集の欠如．メソッドの凝集度とはクラスのメソッドと属性の関連性を意味する．凝集性が高いほど，意味的に「閉じた」クラス設計がなされていることになり，良い設計であることを示す．LCOM は計測対象クラスのメソッドのすべての組み合わせのうち，共通した属性を参照していないメソッドの組み合わせの数から，共通した属性を参照しているメソッドの組み合わせの数を引いた値となる．ただし，計測値が 0 より小さいさくなるときは，0 となる．

2.1.2 サイクロマチック数

サイクロマチック数とは，McCabe によって提案されたメトリクスであり[7]，メソッドの複雑度を表す．プログラムの制御フローを有効グラフで表現したときの枝の数を e，節点の数を n と定義すると $e-n+2$ で表される．この値は直観的にはソースコード上の分岐の数に 1 を加えた数を表す．サイクロマチック数が大きいほど，そのメソッドの制御フローは複雑になり，保守性や可読性が低いことを意味する．

2.1.3 ソフトウェアメトリクス計測

ソースコードを対象としたメトリクス計測は基本的に次の 2 ステップに分けることができる．

ソースコード解析　メトリクス値の計測に必要な情報をソースコードから抽出．

メトリクス値計測　ソースコードから抽出した情報を利用してメトリクス値を計測．

多くのソフトウェアメトリクスは，CK メトリクスやサイクロマチック数のようにファイルやクラス，メソッド，関数，制御フローといったソフトウェアの概念的な要素に対して定義されるものであるため，言語

間の記述様式の差異にとらわれることなく，同じ概念を共有する言語に共通して適用できる．つまり，メトリクス値計測のロジックは言語が異なっても共通して利用することができる．一方，異なる記述様式に対応するためには，各記述様式用のソースコード解析器を用意しなければならない．

ソフトウェアメトリクスの計測に必要なコストの大部分はソースコード解析が占めている．このため，既存のメトリクス計測ツールの多くは単一の言語を対象としており，複数の言語で記述されたソフトウェアから統一的にメトリクス値を計測することは難しい．

2.2 ソースコード解析

ソースコード解析とは，ソースコードを解析器と呼ばれるプログラムを用いて，自動で必要な情報を抽出する技術である．抽出された情報は，ソフトウェアメトリクスやその組織独自の方法を用いて調査され，ソフトウェアの品質を把握するためなどに用いられる．

しかし，ソースコードを解析し，その情報からメトリクスなどを算出するには，例えば，関数間の呼び出し関係などの，意味解析以上の深い解析をしなければ得ることのできない情報を必要とするため，解析器自体を作成することに大きなコストが必要である．既に，JavaCC[1] や ANTLR[2] などに代表される多くのコンパイラコンパイラが開発されているが，それらが支援するのは構文解析までであり，それ以上の深い解析を行う場合は，多大な時間と労力を必要とする．

一方，ソフトウェアのソースコードが調査される機会は増加しており，ソースコード解析技術の重要性は増している．たとえば，ソフトウェア開発企業では，QA（Quality Assurance) のために，開発したソフトウェアをソフトウェア工学的な手法を用いて調査することが増えてきた．この調査の一環として，ソースコード解析を行い，その品質や今後の保守コストなどを予測することが多い．しかし，近年顕著に見られる開発期間の短縮化や，対象ソフトウェアによって適用する解析手法が異なることから，満足に QA が行われているとは言えない．

また，情報化社会のニーズに応えるために，国内の多くの大学において情報系学部・学科が新設されている．そのため，卒業論文や修士論文のテーマとしてソースコード解析が行われることが増えてきている．しかし，Java や C++など，実際に社会で広く利用されているプログラミング言語で記述されたソースコードを解析するのは，多くの学生にとって敷居が高いと言わざるを得ない．そのため，良いアイデア（解析手法やメトリクスなど）を思い付いたとしても，それが実現されることなく，単なるアイデアとして学位論文にまとめられてしまっている．

これらの問題は，深い解析を行う解析器を実装するコストが高いことに起因している．解析者は，ビジネスロジックの実装ではなく，そのビジネスロジックで用いる情報を抽出する解析器の実装に悩まされている．

3. MASU の概要

ソフトウェアメトリクスの計測にともなう問題を解決するため，我々は複数言語の解析結果から統一的にメトリクス計測を行えるプラグインプラットフォーム MASU(Metircs Assesment plugin platform for Software Unit) を開発した．MASU は次の特徴をもつ．

- 複数のプログラミング言語のソースコードに対して適用可能である．
- 多言語の解析から得た結果を統一的に扱える．
- 言語依存な要素にも対応するため，解析対象ソフトウェアの構成要素の追加や変更を柔軟に行える．
- ユーザは，必要最低限のビジネスロジックを記述するだけで，新たなメトリクスを計測可能である．

MASU は Java を用いて実装されており，現在，規模は約 42000 行，クラス数は約 390 である．

3.1 入　出　力

MASU は，広く使われているオブジェクト指向言語のソースコードからメトリクスを計測する．出力は対象ソースコードに含まれるすべてのクラスやメソッドなどのメトリクスの値やソースコード解析情報である．

3.2 アーキテクチャ

MASU の構成図を図 1 に示す．本ツールは次の 2 種類のモジュールから構成される．

メインモジュール　ソースコードを解析し共通データを構築する．

プラグイン　共通データを用いて個別のメトリクス値を計測する．

以下にそれぞれの詳細を示す．

3.2.1 メインモジュール

メインモジュールは図 1 が示すようにプロダクト解析部，プラグイン制御部，メトリクス集計部から構成される．以下に各部の役割を示す．

- **プロダクト解析部**
  プロダクト解析部は言語非依存な AST(Abstract Syntax Tree：抽象構文木) 生成部と AST 解析部，データ構造構築部に分けられる．図 2 にプロダクト解析部の構成を示す．
  AST 構築部: 入力として与えられたソースコードをもとに言語非依存な AST を生成する．図 2

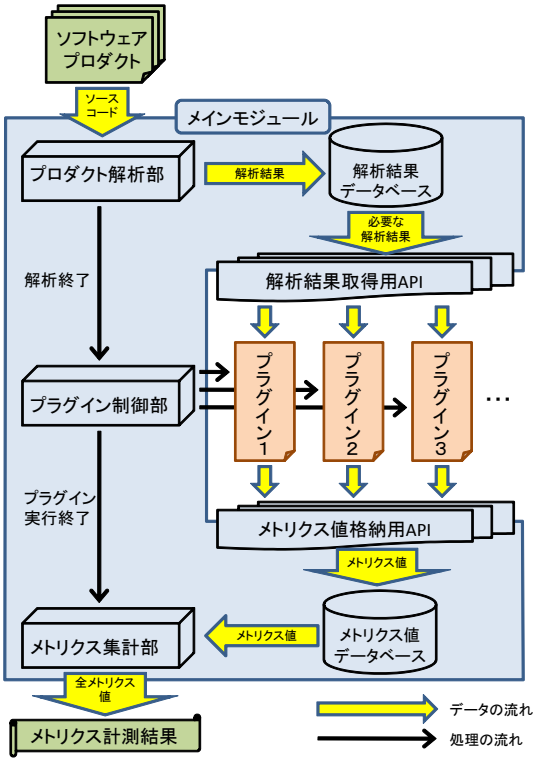

図 1　MASU の構成図

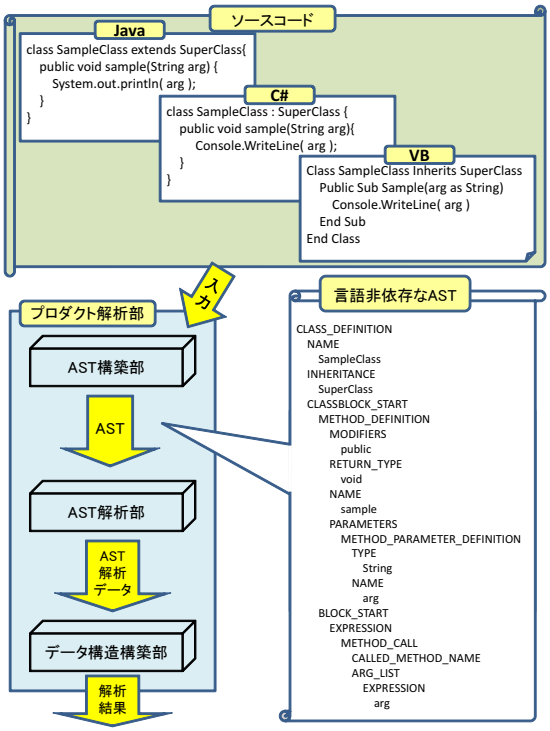

図 2　プロダクト解析部の構成

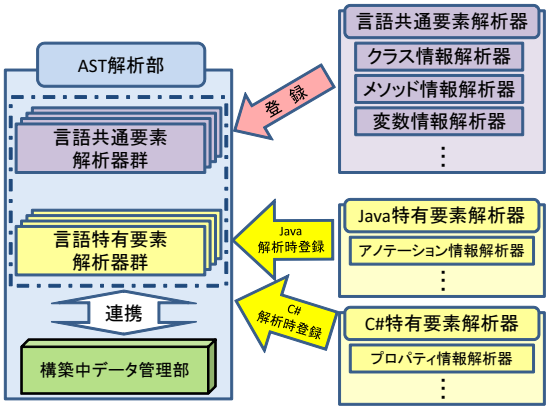

図 3　AST 解析部の構成

に示されてある 3 つのソースコードは意味的には同じであるが，異なるプログラミング言語（Java, C#，Visual Basic）で記述されているため，構文的には異なっている．たとえば，3 つの言語ではクラスの継承関係の定義の方法が異なっている．また，Visual Basic は引数の型の定義の仕方やブロックの開始・終了要素の記述方法が Java や C#とは異なっている．AST 構築部ではこのような各言語間の構文的な違いを吸収し，図 2 の左部に示してあるような共通の AST に変換する．

AST 解析部: AST を解析し言語非依存な共通データを構築する．ここで言語非依存な共通データとは複数の言語で一般的に利用されるようなソフトウェアの要素に対するデータを意味する．具体的には，ファイルやクラス，メソッド，フィールド，制御フロー，名前空間，型などがある．

図 3 に AST 解析部の構成を示す．AST 解析部は複数の解析器と構築中データ管理部で構成される．解析器はソフトウェアを構成するそれぞれの要素に対して 1 つずつ用意され，構築中データ管理部や他の解析器と連携しながら対応する要素の情報を解析する．

例えば，メソッド情報解析器はメソッドの情報のみを解析する．具体的には，変数情報解析器や識別子情報解析器と連携し，メソッドの引数やメソッド名などの情報を解析する．また，構築中データ管理部から構築中のクラス情報を取得し，解析対象のメソッドを宣言しているクラスとの所有関係を解析する．

異なる言語間で共通している要素の構文的な違いは AST 構築部で吸収されるため，共通要素に対する解析器は言語が異なっても正常に動作する．

データ構造構築部: AST 解析部で抽出した各要素

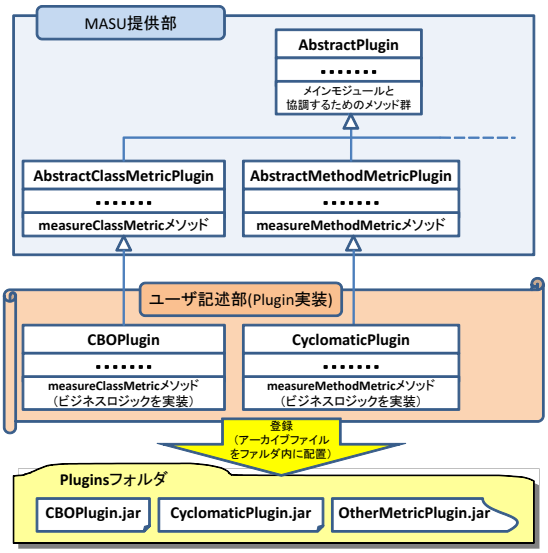

図 4　プラグインの概略

の関係を解析し，データ構造を構築する．例えば，「メソッド間の呼び出し関係」や「変数宣言情報とその参照情報」などの要素間の情報を構築する．

- **プラグイン制御部**
  登録されているプラグイン群を自動的，あるいはユーザの操作によって実行する．
- **メトリクス集計部**
  各プラグインによって計測されたメトリクス値を集計しファイルに出力する．

3.2.2　プラグイン

各プラグインは基本的に，1 つのメトリクスを計測するように実装される．個々のプラグインはメインモジュールが提供する解析結果取得用 API を用いて，メインモジュールが構築したデータを取得する．そして，計測対象要素のメトリクスを計測し，メトリクス値格納用 API を用いて計測結果を登録する．

各プラグインの作成手順を図 4 に示す． MASU はメインモジュールと協調するための API を持った抽象クラスを提供する．ユーザはメトリクスの計測対象となる要素に対応した抽象クラスのサブクラスを実装すればよい．現在，MASU がメトリクス計測対象としている要素はファイル，クラス，メソッド，フィールドである．例えば，CK メトリクスを計測する場合は計測対象となる要素はクラスであるので，AbstractClassMetricPlugin クラスのサブクラスを実装する．基本的にはメトリクス計測用のメソッド (measureClassMetric メソッドなど) にビジネスロジックを記述し，必要に応じてメインモジュールと協調するためのメソッドをオーバーライドすればよい．

作成された Plugin クラスはアーカイブ化して専用の plugins フォルダに配置すればメインモジュールが自動的にプラグインを認識し，プラグイン制御部で使用可能になる．

3.3　特　　徴

3.3.1　ビジネスロジックの分離

メインモジュールは，メトリクス計測のためのデータ構造を提供し，個々のメトリクス値を計算するためのビジネスロジックはプラグインとして記述される．

このように，メインモジュールとプラグインを分離することにより，新しいメトリクスを計測する場合でも，プラグインのみを作成するだけでよく，ツール作成のコストを大幅に削減することが可能となる．

3.3.2　高い拡張性

MASU のメインモジュールの設計の特徴として高い拡張性があげられる．

複数の言語間にはファイルやクラスのような共通した要素も多いが，そうでない要素も存在する．たとえば，C#には JAVA には存在しない”プロパティ”と呼ばれる要素が存在する．また，同じ言語でもそのバージョンが異なれば，ソフトウェアを構成する要素が異なることがある．たとえば，JAVA1.5 以降には JAVA1.4 までには存在しなかった”generics”という要素が存在する．このような言語やバージョンに依存した情報もメトリクスの計測には必要な場合が多い．

MASU の AST 解析部はソフトウェアの構成要素ごとに解析器をもつため，図 3 に示すように言語依存な要素に対応した解析器を切り替えるだけで各言語特有の要素に対応できる．例えば，C#を解析する際は，言語共通要素解析器とは別にプロパティ情報解析器を登録すれば，C#特有の要素であるプロパティの情報を解析することができる．

より深い解析を行うために解析する要素を増やしたい場合は，その要素に対応した解析器を作成，登録するだけでよく，既存のコードを変更する必要はない．不要な要素の解析を行いたくないときは，その要素に対応した解析器を無効にすればよい．例えば，CK メトリクスの計測には，制御フローの情報は必要ないため，制御フロー情報解析器を無効にすれば，ソースコード解析のパフォーマンスが向上する．

3.3.3　セキュリティの安全性

MASU はアクセス制御をスレッド単位で動的に行うセキュリティマネージャを提供する．具体的には，次の処理を行う．

- 各スレッドの権限の動的な変更．
- 他のスレッドに勝手に譲渡されないスレッド単位

での動的な権限管理.

- メインモジュール・GUI・プラグインに共通して許可されるグローバルな権限の動的な管理.
- 各プラグインのファイルシステムへのアクセス制御.

簡単に述べると，MASU はメインモジュールが構築したデータ構造をユーザ (プラグイン側) が勝手に変更できないよう管理している.

3.4 言語依存要素と非依存要素

MASU の AST 解析部が高い拡張性を提供したとしても，言語非依存要素の実装にかかるコストが高すぎるようであれば，言語非依存 AST および共通要素解析器の効果は低くなる．そこで，我々は共通要素解析器の有効性を示すために，オブジェクト指向言語共通の要素を解析するコードと JAVA 特有の要素を解析するコードの実装に必要としたコストを，クラス数を用いて定量的に評価した．各要素に対する「解析器に必要としたクラス数」と「要素の情報を保存するために必要としたクラス数」,「その他の処理に必要としたクラス数」を表 1 に示す．表 1 より，少なくとも JAVA 特有要素の解析にかかる実装はオブジェクト指向言語共通要素の解析にかかるコストに比べ非常に小さく，共通要素解析器の効果は高いと言える.

また，MASU の AST 解析部は言語特有の要素を，意味的に同じであれば言語共通要素としてみなし，共通要素解析器を用いて解析する．例えば，C#のプロパティは意味的にはアクセサメソッドと等しいため，MASU ではメソッドとして扱うことを考えている．これにより，オブジェクト指向言語共通の概念に対して定義されたメトリクス全般を計測することを可能とする．言語特有の要素に関するメトリクスを計測したいという要求も考えられるが，既存のメトリクスの多くはオブジェクト指向言語共通の概念に対して定義されいる．MASU の目的は共通の概念に対して定義されたメトリクスを，異なる言語で記述された複数のソフトウェアから統一的に計測することであるため，特定の言語のためのメトリクスは計測対象としない.

4. MASU を用いたプラグイン実装

MASU を用いたメトリクス計測プラグインの試作として，CK メトリクスとサイクロマチック数を計測するプラグインを実装した.

表 1 オブジェクト指向言語共通要素と JAVA 特有要素の実装に要したコスト (クラス数)

対象要素	解析器	情報保存	その他
オブジェクト指向共通要素	118	157	19
JAVA 特有要素	12	2	5

4.1 プラグインの規模

各メトリクス計測プラグインの規模と実装に要した時間を表 2 に示す．いずれのプラグインも実装は 1 人で行われた．表の列「メトリクス」は計測されるメトリクス名を,「総行数」は実装されたプラグインの総行数 (カッコ内は空白・コメント込みの総行数) を,「BL 行数」はビジネスロジックの行数を,「時間」は実装に要したおおよその時間を示す．また，WMC は全てのメソッドの複雑度が一律であると仮定して実装した.

4.2 プラグイン実装例

MASU のプラグイン実装の例として，CK メトリクスの 1 つである RFC と WMC を計測するためのプラグインの例を示す.

RFC 計測プラグイン RFC を計測するには，各クラスに定義されているメソッドと，それらのメソッドの中で呼び出されているメソッドの情報を抽出しなければならず，高いコストを必要とする．しかし，MASU のプラグインとして実装することにより，表 2 や図 5 に示す程度のソースコードで実装することができる．図 5 からわかるように，メインモジュールと協調するための API はほとんど 1 行で実装することができ，プラグイン作成に必要なコストは実質的にはビジネスロジックを記述するメソッドの実装のみである．ビジネスロジックの記述もメインモジュールが提供する情報を利用することにより，低コストで実装することができる.

WMC 計測プラグイン クラスに定義されているメソッドの複雑度がすべて等しいと仮定すると，WMC 計測プラグインは図 6 の WmcPlugin クラスのように，非常に低コストで実装することができる．MASU を利用しない場合でも，クラスのメソッド数を計算するだけでよいためそれほど高いコストは必要としない.

しかし，WMC は，各メソッドの重み計算については定義していないが，各メソッドが一律の重み

表 2 試作プラグインの規模と要した時間

メトリクス	総行数	BL 行数	時間 (分)
WMC	31(74)	2	10
DIT	35(81)	8	20
NOC	36(73)	1	10
CBO	61(121)	29	20
RFC	56(117)	7	15
LCOM	114(221)	48	40
サイクロマチック数	52(115)	21	25

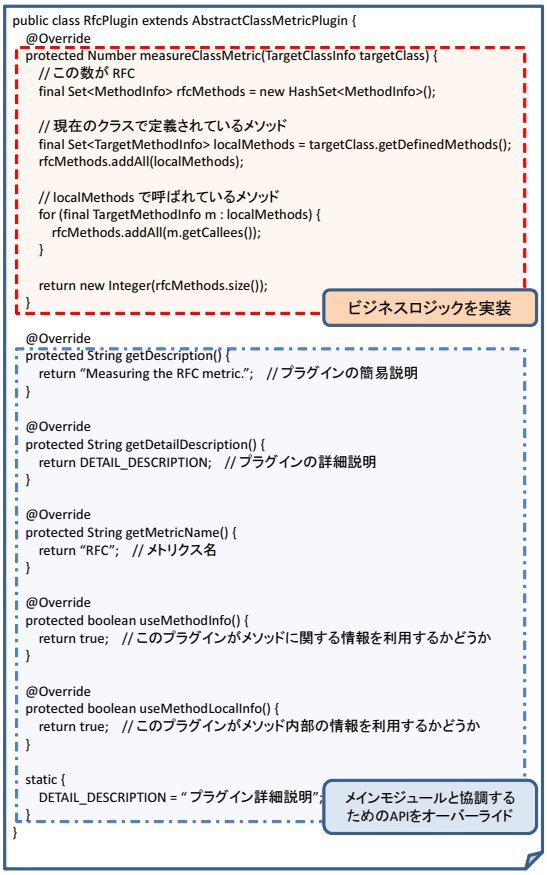

図 5 RFC 計測プラグイン実装例

を持つとして計測されることは少ない．メソッドの複雑度を示すメトリクスとしてサイクロマチック数などが有名であるが，厳密な定義が存在しない以上，ユーザーによって要求するメトリクスは異なるであろうし，さまざまなメトリクスを用いて計測結果を比較したいという要求も考えられる．このような場合，要求されるメトリクスそれぞれに対応した計測ツールを作成することは容易ではない．しかし，MASU の解析情報を利用すれば図 6 の CustomWmcPlugin クラスに示してあるように，ユーザー独自のメソッド複雑度計測ロジックを自由に記述することができる．

例が示すように，MASU を利用することにより，ユーザーは低コストかつ自由にメトリクス計測ツールを実装することが可能となる．

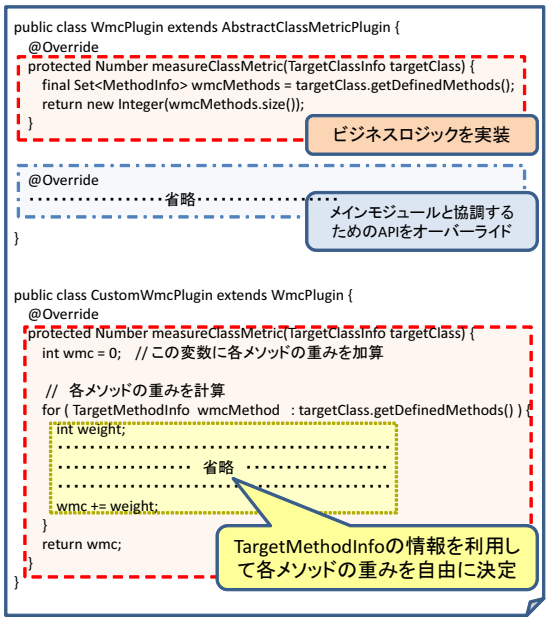

図 6 WMC 計測プラグイン実装例

5. 関 連 研 究

5.1 バイトコード解析ツール

バイトコード解析ツールとしては，以下の 2 つが有名である．

Soot: a java Optimization Framework[9] McGill 大学のプロジェクトとして開発されている Java バイトコードを開発者の定義どおりに最適化するためのフレームワーク．Soot は最適化のための情報としてバイトコード解析情報を提供するため，プログラム解析ツールとしても使用されている．

WALA[10] IBM Research が開発したオープンソースの Java バイトコード解析ライブラリ．

バイトコードから得られる情報はソフトウェアの潜在的な情報を多く含むが，制御文の種類などのようにソースコードからでなければ得られない情報も多く存在する．また，バイトコードはコンパイラが生成したコードであり，人間が保守しなければないらないのはソースコードである．そして，メトリクスが示すのは，人間が作成したプロダクトの品質評価や，人間が行う保守のコスト予測などに使われる尺度である．ゆえに，メトリクスの計測は，バイトコードの特徴ではなく，ソースコードの特徴を測ることに意味があるといえる．

また，Soot や WALA は Java のバイトコードのみを対象としており，多言語対応はされていない．

5.2 MOOSE

Ducasse らは言語非依存のリエンジニアリングプ

ラットフォームとして MOOSE を開発している[3]．MOOSE は MASU と同様にソースコードを入力としたソフトウェアメトリクスの評価機能を提供する．

しかし，MOOSE はソースコードを直接解析するわけではなく，CDIF や XMI などに変換された情報を解析するため，提供される情報はモデルベースの情報となる．モデルベースの情報はソフトウェアの視覚化などには有用であるが，ソースコードから直接得られる情報と比較すると情報量が減少する．

また，ソースコードから CDIF や XMI への変換部はサードパーティーのツールに依存しているため，MASU が提供するような拡張・変更容易性は備えていない．

6. 現状と今後

6.1 入力ソフトウェアプロダクトの拡大

MASU の開発はまだ初期段階であり，ソースコードの解析部は現在は Java 用の実装が用意されているのみである．今後，C++,C#などのほかのオブジェクト指向言語に順次対応していく予定である．

また，解析対象をソースコード以外に広げることも考えている．例えば，実行履歴を解析して共通データを構築するような解析部を作成することで，動的なデータを基にしたメトリクス計測が可能になる．

6.2 解析部のプラグイン化

現在，MASU の解析部は拡張性の高い設計を適用することにより，解析する言語非依存な共通データを拡張しやすく，また，言語依存なデータの解析にも柔軟に対応できる．しかし，解析部はメインモジュールの一部として設計されているため，ユーザが独自に解析部を改良し，データ構造の構築の方法を変更したり，対応言語を拡張することは難しい．

そこで解析部をプラグイン化することにより，ユーザーが独自にさまざまなソフトウェア要素に対するメトリクスに対応したり，データ構造を自由に構築することができるようにし，多様な入力データに柔軟に対応できるようにすることも検討していきたいと考えている．具体的には，図 3 に示してある個別要素ごとの情報解析器をプラグイン化する予定である．

6.3 GUI の作成

メインモジュールとプラグインの他に GUI を用いたユーザーインターフェースモジュールを作成することを考えている．現在のところ，Swing を用いたスタンドアロンアプリケーション用のインターフェース，SWT を用いた Eclipse プラグイン用のインターフェース，GWT を用いた Web ベースのインターフェースの 3 つを検討している．

Eclipse プラグインに関しては，既に開発に着手しており，メトリクス計測用のインターフェースだけでなく，Eclipse における MASU プラグイン開発プロジェクト作成インターフェースの実装も検討している．

7. ま と め

本稿では，メトリクス計測用プラグインプラットフォーム MASU について述べた．MASU は以下の特徴を持っており，MASU を用いることによって，従来に比べてはるかに低いコストでソースコードからメトリクスを計測することが可能になる．

- 複数のプログラミング言語のソースコードに対して適用可能である．
- 多言語の解析から得た結果を統一的に扱える．
- 言語依存な要素にも対応するため，解析対象ソフトウェアの構成要素の追加や変更を柔軟に行える．
- 必要最低限のビジネスロジックを記述するだけで，新たなメトリクスを計測可能である．

MASU の開発はまだ初期段階であり，他言語への対応や GUI などは十分ではない．今後さまざまな拡張を行う予定である．

参 考 文 献

1) JavaCC. `http://javacc.dev.java.net/`.
2) ANTLR. `http://www.antlr.org/`.
3) A. L. Baroni and F. B. Abreu. An OCL-Based Formalization of the MOOSE Metric Suite. In *Proc. of the 7th ECOOP Workshop on Quantitative Approaches in Object-Orietend Software Engineering*, 2003.
4) S. Chidamber and C. Kemerer. A Metric Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, 25(5):476–493, Jun 1994.
5) Eclipse Metrics Plugin. `http://www.teaminabox.co.uk/`.
6) M. H. Halstead. *Elements of Software Science*. Elsevier Science Inc., 1977.
7) T. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, Dec 1976.
8) P. Oman and S. L. Pleeger. Applying Software Metrics. *IEEE Computer Society Press*, 1997.
9) Soot: a Java Optimization Framework. `http://www.sable.mcgill.ca/soot/`.
10) WALA. `http://wala.sourceforge.net/wiki/index.php/Main_Page`.