Vol.2013-SE-179 No.29
2013/3/12

ツールによる形式モデルの成熟度測定法の提案

井上 心太[1]　大森 洋一[2]　荒木 啓二郎[2]

概要：ソフトウェアの仕様書を形式モデルに変換し，検証することでソフトウェアの品質を向上できる．本研究では、形式モデル作成の際に，モデルの成熟度を測るための指標を測定し，判断材料の提供を目的とする．測定のための指標の選定にはソフトウェアメトリクスと EASE プロジェクトの知見を参考にした．指標の入力はモデル作成者の主観を排し，かつ手間を減らすためにツールにより自動化と収集できるように考慮した．

Proposal of Formal Model's Maturity Measuring Method by a Tool

Abstract: The Software quality can be improved by conversion to the formal model from the specifications of software and verifying it. The objective of this research is the offer of indexes for measuring the maturity. When model makers create the formal models, the indexes are the judgement material of the formal model maturity. The software metrics and the knowledge of an EASE project was referd to selection of the indexes for measurement. It was considered that the input of the indexes can collect with automation with a tool in order to eliminate a model maker's subjectivity and reduce time and effort.

1. はじめに

ソフトウェアの品質向上の方法のひとつとして形式手法が注目されている．形式手法を適用し，ソフトウェアの仕様書を形式的なモデルに変換することで仕様書中の矛盾や曖昧さを取り除くことが可能となる．また、形式的なモデルはツールにより数学的な検証が可能であり，ソフトウェアの正しさを証明することもできる [1].

形式手法とはソフトウェアの仕様書を厳密に記述し，数学的な検証を行う技術である．形式手法では，仕様書を形式的なモデルとして記述する段階とそのモデルを検証する段階とに分けられる．記述段階では，形式仕様記述言語と呼ばれる形式モデル記述用の言語を用いて，ソフトウェアのモデルを記述する．形式仕様記述言語には状態モデルに基づくものや代数的仕様に基づくものなど様々な種類が存在する．これらの言語は，検証したい目的により選択して使用する．代表的な言語としては，Z 言語, B，VDM-SL, VDM++，Alloy，PROMELA などがある．また，検証段階には，以下の 3 つの方法が存在する．

- 証明
- モデル検査
- 仕様アニメーション

実際に形式手法を適用する場合は，これらのいずれかを選択して使用するか，これらを組み合わせて使用する．これらの検証の内容は**表 1** に示す [2].

本研究では，形式仕様記述言語として VDM++を使用した．VDM++は形式手法の一種である VDM（Vienna Development Method）の形式仕様記述言語であり，オブジェクト指向に対応している [3]．そのため，ソフトウェアシステムのモデルを柔軟に記述することが可能である．また，VDM を適用したプロジェクトの成果も報告されており，その有効性は実証されている [4].

形式手法を実際の開発へ用いる際の課題のひとつが，モデルをどこまで書けばよいのかの判断に知識や経験を必要となるという点である．すなわち，形式手法はソフトウェア開発の様々な工程へ応用することができるので，モデル化している対象が仕様であるか設計であるか，対象となるシステムに含まれているか，といった判断に属人性がある．ここで，実際の開発とは，複数の関係者が関わるような開

[1] 九州大学大学院システム情報科学府
Graduate School of Information Science and Electical Engineering, Kyushu University
[2] 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

表 1 形式手法における検証の種類
Table 1 Kind of verification in formal method

検証の種類	説明	利点	欠点
証明	モデル中の仕様に矛盾がないことを数学的に証明する.	数学的に証明することにより曖昧さが混入しない厳密な検証が可能であること.	多くの場合，証明に必要な論理式を仕様作成者がモデルとは別に作成しなければならない.
モデル検査	仕様書中の論理式を内部的に木構造に展開し，そのすべての組み合わせを実行して検証する.	モデルのすべての状態を網羅的に検証可能であること.	モデル検査できる論理式は，一般的な仕様記述用の論理式より適用範囲が狭く，対象システム全体を記述するのに向かない.
仕様アニメーション	仕様の実行を通して妥当性検査と検証を行う.	テストケースとその実行結果により検証を行うため，形式手法の知識を持たない人間にも分かりやすいこと.	証明やモデル検査に比べて「完全な検証」という点で劣る.

発を指す．仕様は顧客と開発者の合意であるが，顧客と開発者，あるいは開発者間や顧客間で万人が納得する合意形成は簡単ではない．どこかで仕様が十分成熟されたと判断し，次の段階へ移る決断を下さなくてはならない．

本研究は，こうした立場の違いを乗り越えるための，成熟した仕様の客観的な指標の提示を目的としている．これまでにこの目的のための確立した指標はないので，
(1) プログラムと同じ指標を応用する
(2) 仕様記述のプロセスの指標を応用する
という 2 通りの方法を検討する．前者は，特に機能アルゴリズムまで記述する陽仕様に，プログラムと同様のメトリクスを定義し，モデルを評価するものである．後者は記述された仕様そのものではなく，仕様記述プロセスの振興によりモデルの成熟度を評価する．

前提として，仕様の良し悪しは，対象となる問題領域によって大きく変わり，また関係者の技術力や生産設備や市場動向，派生開発の可能性といった非技術的要素も関係し，機械的な評価が困難であるという特性がある．この意味において，本研究で提案する指標は絶対的なものではなく，客観的かつ現状の開発よりコストを増やさずに計測できるべきであろう．したがって，自動的かつ時間をかけずに収集した情報を，関係者へ提示する手法が望ましいと考えられる．

本研究の以降の章の構成は以下のとおりである．まず，第 2 章では，本研究で参考にしたソフトウェアメトリクス及び．EASE プロジェクトについて述べる．第 3 章では，形式モデルの成熟度測定に用いる指標を提案する．第 4 章では，形式モデルの成熟度を自動で測定するためのツールについて述べる．最後に第 5 章では，本研究のまとめを述べる．

2. プログラムの測定指標

プログラムのソースコードコードを測定するための指標には様々なものがある．指標を使用して，ソースコードの状態を知ることで，ソフトウェアの品質を向上させたり，プロジェクトのマネジメントに利用したりする．本章では，ソフトウェアメトリクスと EASE プロジェクトで用いられた指標について述べる．

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を可視化するための指標の集合である．ソフトウェアのソースコードを測定し，指標を数値で示すことでソースコードの品質を定量的に評価できる．一般的にメトリクスの測定にはツールを使用し自動化する．大規模なソフトウェアに対して，様々なメトリクスを収集するためには膨大な時間がかかってしまうからである．代表的なメトリクスとしては，凝集度と結合度が挙げられる．以降では，この 2 つのメトリクスについて説明する．

2.1.1 凝集度

凝集度とはソフトウェアのクラスの構成要素の関連性を示す指標である．凝集度が高いほどクラスの独立性も高く優れた設計のコードといえる．凝集度の測定では，クラスやパッケージ内の機能要素と情報要素間の関連性を測る．着目しているクラスに互いに関連するメンバ変数やメソッドが十分に集まっていれば，そのクラスの変数 1 つあたりへアクセスする関数は多くなる．このような性質に着目したメトリクスの 1 つが，*LCOM**（Lack of Cohesion in Methods）である．*LCOM**は次の式で定義される．

$$LOCM* = \frac{\frac{1}{a}\sum_j^a \mu(A_j) - m}{1 - m} \quad (1)$$

式 (1) より *LCOM**は，メンバ変数 1 つあたりへアクセ

表 2　モジュールの凝集度の種類
Table 2　Kind of module cohesion

独立性	凝集度	種類	説明
低	弱	暗号的凝集度	プログラムを単純に分割しただけで、モジュールの機能を定義できない．または，複数の機能間にまったく関連はない．
		論理的凝集度	関連した複数の機能をもち，モジュールが呼び出されるときの引数（機能コード）で，モジュール内の 1 つの機能が選択，実行される．
		時間的凝集度	初期設定や終了設定モジュールのように，特定の時期に実行する機能をまとめたモジュール．モジュール内の機能間にあまり関連はない．
		手順的凝集度	複数の逐次的に実行する機能をまとめたモジュール．
		連絡的凝集度	手順的凝集度のうち，モジュール内の機能間にデータの関連性があるモジュール．
		情報的凝集度	同一のデータ構造や資源を扱う機能を 1 つにまとめ，機能ごとに入口点と出口点をもつモジュール．
高	強	機能的凝集度	1 つの機能だけからなるモジュール．

A_j　着目しているクラスの j 番目のメンバ変数
a　着目しているクラスのメンバ変数の個数
m　着目しているクラスのメソッドの個数
$\mu(A_j)$　メンバ変数 A_j にアクセスしているメソッドの個数

スするメソッドの数が少ないと 1 に近づき，多ければ 0 に近づく．つまり，*LCOM**の値が小さいほど凝集度が高いといえる．凝集度を高く保つためには，1 つのクラスが請け負う機能を関連の強い機能のみに絞ることが必要である．こうすることで，クラス毎の機能が明確になり，コードの可読性が上がる．また，凝集度の種類を表 2 に示す．

2.1.2　結合度

結合度とは，ソフトウェアのクラス間の関連性を示す指標である．結合度が低いほどクラスの独立性が高く，優れた設計のコードといえる．結合度の測定では，クラス内に定義されている型の種類の数を測る．具体的には，継承クラス，インターフェイス，メンバ属性，メソッドのパラメータなどである．つまり，結合度を低く保つために，クラス間の不要な関連をなくすことと，関連のあるクラス間の関連を可能な限り小さくすることが必要となる．こうすることで，他のクラスの実装を影響を受けずにクラスの追加や削除が可能となり，ソフトウェアへのバグの混入を防ぎやすくなる．結合度の種類を表 3 に示す．

このように，メトリクスをこまめに測定し，そのフィードバックを受けることで開発中のソースコードの状態を知ることができる．コードの状態に応じて，対策や改善を行うことができるためソフトウェア開発のマネジメントにも役立つ．

2.2　EASE プロジェクト

EASE プロジェクトとは 2003 年から開始された文部科学省のリーディングプロジェクト「e-Society：基盤ソフトウェアの総合開発」の中の「データ収集に基づくソフトウェア開発支援システム」をテーマにしたプロジェクトで，奈良先端大学，大阪大学が中心となった活動である [8]．EASE とは，Eprical Approach to Software Engineering（ソフトウェア工学へのエンピリカルアプローチ）の略であり，ソフトウェア開発において定量的なデータに基づく科学的手法を適用することで，生産性や品質の向上を目指そうというものである [9]．

2.2.1　先進ソフトウェア開発プロジェクト

先進ソフトウェア開発プロジェクトとは，独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センターが提唱する手法を実証するために実施されたプロジェクトである．このプロジェクトでは EASE が研究開発を行っている EPM, EPM Pro*と呼ばれるツールを使用してプロジェクトの計測・分析を行っている．

2.2.2　EPM

EPM（Emprical Project Monitor）は，エンピリカルアプローチを実践する環境の一つであり，「ソフトウェア開発における自動的なデータ収集と分析のためのプラットフォーム（エンピリカル環境）」である [9], [10]．この環境では，ソフトウェア開発データを収集するツールが用意されており，集められたデータは分析ツールにかけられ，分析結果から導き出された改善策がプロジェクト管理者を通して，開発現場にフィードバックされる．EPM では，ソフトウェア開発者の負担を小さくするため，ソフトウェア開発プロジェクトで広く使用されている

- 構成管理システム
- メール管理システム
- 障害管理システム

からデータが収集される．収集データはトランスレータによって，収集コンテキストを示すタグが付与され，XMK 形式の標準エンピリカルデータとなる．標準エンピリカルデータは，インポータによって EPM リポジトリと呼ばれ

表 3　モジュールの結合度の種類
Table 3　Kind of module coupling

独立性	結合度	種類	説明
低	強	内容結合	絶対番地を用いて直接相手モジュールを参照したり，相手モジュールに直接分岐する．
		共有結合	共通領域（グローバル領域）に定義されたデータを参照する．
		外部結合	必要なデータだけを外部宣言し，他のモジュールから参照を許可し共有する．
		制御結合	機能コードなど，モジュールを制御する要素を引数として相手モジュール渡し，モジュールの機能や実行を制御する．モジュール凝集度の論理的凝集度がこれに相当する．
		スタンプ結合	相手モジュールで，構造体データ（レコード）の一部を使用する場合でも，構造体データすべてを引数として相手モジュールに渡す．
高	弱	データ結合	相手モジュールでをブラックボックスとして扱い，必要なデータだけを引数として渡す．

るデータベースに格納される．EPM リポジトリのデータはアナライザによって分析され，その結果が Web ブラウザ上にグラフ表示される．アナライザによる表示例としては，以下のようなものがある．

- ソースコードの行数
- 障害検出数の時間推移
- 構成管理システム上でのチェックインとチェックアウトのタイミング
- 開発者によるメール投稿数の推移とプログラム更新時期との関係
- 障害検出数と構成管理システムへのチェックインの関係

EPM のグラフ表示の例を表 1 に示す [11].

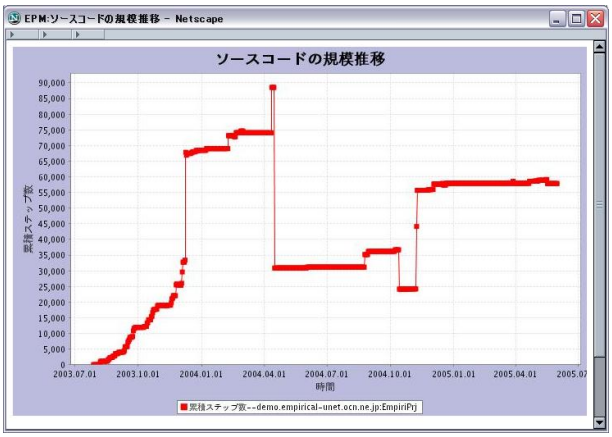

図 1　EPM のグラフ
Fig. 1　Graph of EPM

2.2.3　EPM Pro*

EPM で収集したデータの分析用の拡張機能として EPM Pro*がある [12]．分析機能は，プロジェクトの遅延リスクの検出を目的として，GQM（Goal/Question/Metric）パラダイム [13] を用いている．EPM Pro*は，大きく 2 つの分析機能を持つ．障害追跡ツールで管理されるバグ管理データを扱う ProQAV（Project Quality Assessment Viewer）と，構成管理システムで管理されるソースコードデータを扱う ProPFV（Project Process Flow Viewer）である．ProQAV は以下の項目を出力する．

- バグ累積件数
- 重要度や優先度の高い障害件数や未解決件数
- 平均滞留時間の推移

さらに，詳細分析機能として，バグ残存率や平均滞留時間の移動平均，乖離率なども出力できる．また，ソースコードデータとバグ管理データを関連付けて分析することも可能で，コーディングバグ密度（KLOC あたりのコーディングバグ数），設計バグ密度（KLOC あたりの設計バグ数）なども出力する．一方，ProPFV は以下の項目を出力する．

- 規模遷移としての LOC（Lines of Code）
- ファイル数
- ファイル更新回数
- 10 行以上削除された更新の回数
- 追加行数，削除行数

また，要員の管理や配置の適切さを計測するメトリクスとして，2 名以上の作業者によって変更が加えられたファイル数と 1 ファイルあたりの変更を加えた作業者数の平均も見ることができる．

2.2.4　プロジェクトの測定と分析

先進ソフトウェア開発プロジェクトでは，2 つのフェーズで開発が実施された．フェーズ 1 では EPM，フェーズ 2 では EPM Pro*を使用してプロジェクトの計測・分析が行われた．フェーズ 1 では以下に示すデータが計測された．

- ソースコードの行数の推移
- ファイルやソースコードの変更量
- ファイル更新履歴
- 障害件数の推移
- 平均障害滞留在日数の推移

- 障害の混入・発見工程
- 障害原因
- 社間メール件数の推移

一方，フェーズ 2 では，EPM によって収集された情報をもとに EPM Pro*を用いてより高度な分析が試みられた．この分析から開発作業の進捗一端をつかむことができる．分析の結果得られた知見を以下に示す [8].

- ファイルの更新回数を追うことで開発の完了状態を見ることができる
- ファイル数の変化から進捗状況を推測できる
- 追加行数と削除行数がともに大きく増加するときは大規模な追加が想定される
- 追加行数のみが増加する時は新規開発が想定される
- 削除量の大きい更新の比率が高い場合，あるいは削除行数の割合が高い場合，既存コードへの変更が頻繁，または大量で，変更の危険性や既存コードの修正の複雑さを推測できる

これらの活動は従来の管理手法で得ていた状況認識に対してエビデンスを提供した．また，従来の自己申告方式による進捗報告に比べて，自動収集により省力化され，かつ人手の介入がないことにより正確な報告データを提供することができ，円滑なプロジェクト運営を支援する効果があったと報告されている [8].

3. モデル成熟度の測定指標の提案

本研究では，モデルの成熟度を測定する形式仕様記述言語を VDM++[3] とする．VDM++では，クラスを構成単位として形式モデルを記述する．クラスの構成要素として以下の項目があり，モデル記述対象の状態を記述していくことで，モデルを形成する．

- 定数
- 型
- インスタンス変数
- 操作
- 関数
- スレッド
- 同期制約

VDM++の形式モデルの成熟度を測定する指標として以下の項目を考える．

- モデルの更新頻度
- モデルの行数の推移
- 削除・追加行数の推移
- 凝集度
- 結合度

モデルの成熟度の指標は，モデル作成者がどこまでモデルを書いたらよいかを判断する材料として使用される．この判断は人間がするが，そのための指標は人手をかけずに収集し，提示することでコストを上げないようにすることが重要である．また，これらの指標はそれ自身を単独で使用するだけでなく，全体の値の推移を総合的に判断して使用することを前提とする．これにより，形式モデル作成中にモデルを多角的に評価することができ，より詳細にモデルの成熟度を測ることが可能となる．さらに，１つの指標からは分かりにくい形式モデルの変化を読み取ることができ，モデル作成の支援も期待できる．以降では，それぞれの指標の使用目的とその利点について述べる．

3.1 モデルの更新頻度

モデルの更新頻度は，形式モデルを構成するファイル群が更新された日時を測定する．更新頻度が高い，つまり前回の更新日時と今回の更新日時の間隔が小さい場合は，モデルを作成中であることがうかがえる．逆に更新頻度が低くなるにつれてモデルが成熟していると判断することができる．更新頻度を測定し，その推移を追っていくことでモデルの安定度の把握に利用できる．また，プロジェクト中で発生した問題や仕様変更などがプロジェクトに与える影響を読み取ることも可能となる．

3.2 モデルの行数の推移

モデルの行数は，形式モデルを構成するファイル群の各ファイルにおける空白行およびコメントを除いた行数をカウントする．集計する際には，ファイルごとの行数およびモデル全体の行数を集計する．この行数はモデルの規模を表しておりその推移から 1 度の更新でファイルに加えられた変更の規模を知ることができる．例えば，この値が大きい場合には，1 度の更新で大量の変更・追加をモデルに加えたことが分かる．モデルの規模の変化はモデルが成熟するにつれ，小さくなると推測できるため，モデルの行数の推移もモデルの更新頻度と同様にモデルの安定度を把握できる．しかし，モデルの行数のみではモデルに与えられた正しい変更量を知ることはできないため，削除・更新行数の推移と合わせて観測することで安定度を測ることとなる．

3.3 削除・追加行数の推移

モデルの削除・追加行数の推移はモデルの行数の変化からは分からないモデルの変更規模を表す．例えば，モデルの行数が前回の更新の際と変わらないように，モデルに対して削除と追加を行った場合，モデルの行数の推移からは変化が読み取ることができない．このようなけーすではモデルの変化を削除・追加行数からモデルの変更規模の変化を知ることができる．追加・削除行に関しては，その値が小さくなるにつれモデルへの変更規模が小さくなる．そのため, 値が小さいほどモデルが安定したといえる．逆にその値が大きいほどモデルに対して大きな変更が加えられているため，注意してモデルを管理する必要がある．このようなプロジェクトの変化を実データの変化とともに追うこ

とで問題が発生したときにより早く対処できるようになる．形式モデルは仕様書の検証時に用いるため，設計や実装などの工程に入る前に問題を取り除くことが期待できる．

3.4 凝集度

凝集度は，ソフトウェアのソースコードに対して使用される測定指標である．VDM++では，オブジェクト指向に対応しており，クラスを定義してモデルを記述することから，そのクラス間での凝集度を測定することができる．以降では，VDM++モデルにおける凝集度を形式的凝集度と呼ぶこととする．その際には，$LCOM*$ を VDM++に適用した以下の式により形式的凝集度の値 V を算出する．

$$V = \frac{\frac{1}{i}\sum_{j}^{i}\mu(A_j) - o}{1 - o} \tag{2}$$

A_j	着目しているクラスの j 番目のインスタンス変数
i	着目しているクラスのインスタンス変数の個数
o	着目しているクラスの操作の個数
$\mu(A_j)$	インスタンス変数 A_j にアクセスしている操作の個数

この式では，$LCOM*$ 算出式におけるメンバ変数をインスタンス変数に，メソッドを操作に置き換えたものである．形式的凝集度は $LCOM*$ と同様にインスタンス変数 1 つあたりへアクセスする操作の数が少ないと 1 に近づき，多ければ 0 に近づく．V の値が小さいほど形式的凝集度が高いと定義する．また，VDM++のモデルでは操作に対して事条件と事後条件を付加できるが，これら 2 つに含まれるインスタンス変数に関しては $\mu(A_j)$ の数からは除外する．事前条件とは，操作（関数）が実行される前に満たされていなければならない制約をその操作（関数）に対して明示的に定義するものである [14]．そして，事後条件は操作が実行された後に満たされていなければならない制約を明示的に定義するものである．VDM++のモデルでは，このような制約を記述することでモデルが満たすべき性質を明らかにしていく．さらに，VDM++では，操作は陽定義と陰定義の 2 つ方法で定義することができる．陽定義では，操作の具体的な処理内容を記述する．陰定義では，操作の処理内容を記述しない．つまり，陰定義では V の値は一定となる．これにより，陰定義から陽定義に操作の定義を変更した時期や形式的凝集度の変化から読み取ることが可能となる．さらに，陽定義で定義された操作に関しては形式的凝集度から操作の処理内容の部分の変化を読み取ることができる．以上の観点から，モデルの成熟度評価の指標のひとつとして用いることができる．

3.5 結合度

VDM++のモデルを測定する際には，凝集度と同様に結合度も測定することが可能である．結合度はクラス中の型の種類の総数により評価される．VDM++では，ユーザーが定義するモデルの記述対象のドメイン固有の型を使用してモデルを構築する．したがって，VDM++モデルのクラスひとつに含まれる型の種類はソフトウェアのソースコードよりも多くなる可能性がある．したがって，VDM++モデルに対して結合度を測定すると結合度が低くなる．しかし，モデルの成熟していく段階で急激に型が増減することは考えにくい．そのため，もし結合度の値が急激に増減した場合は，仕様の変更や大きな問題が発生している可能性がある．結合度の推移を定期的に確認することでプロジェクト管理における重要な情報を与える．

4. ツール化

4 章で述べた形式モデルの成熟度を測定するための指標をモデルの作成中に収集することは，多大な労力が必要となる．また，人の手でデータを収集する場合，データの漏れなどの意図しないミスが発生する可能性がある．したがって，本研究では形式モデルの成熟度測定のための指標を自動収集し，その結果をモデル作成者にフィードバックするツールを考案した．このツールは，モデル作成者が形式モデルの作成などで実際に使用するフロントエンド部分と指標などを自動的に収集するためのバックエンド部分から成る．ツールの概念図を**図 2** に示す．

特に本稿に関係する部分として，独立したメトリクス評価サーバ上で動作し，評価を表示するためのインターフェースを用意している．これにより，ツール利用者あるいはプロジェクト毎のニーズに応じて，評価プログラムを簡単に追加，削除することができるようになっている．また，さまざまな指標を得るための評価プログラムを従来の形式手法で行なっていた仕様検証と並行してい実行可能であり，トリクス取得にかかる手間および時間の増加を最小限に抑えられる．

4.1 継続的インテグレーション

本研究では，複数の関係者間での仕様の共有に継続的インテグレーションの考え方を適用した．継続的インテグレーション（Continious Integration）とは，ソフトウェア開発手法の一種であるエクストリーム・プログラミング（XP）のプラクティスの 1 つである．Martin Fowler のエッセイ [5] の中で，継続的インテグレーションについて「チームのメンバーが各自の成果物を頻繁に統合するソフトウェア開発のプラクティスある．通常，最低でも 1 日に 1 度以上は各自でインテグレーション作業を行う．インテグレーション時のエラーをできるだけ早く検出できるよう，すべてのインテグレーションは，テストを含めた自動化されたビルドによって検証される．多くのチームが，このアプローチを用いることでインテグレーション上の問題を大幅に削減でき，動作するソフトウェアの開発をより

短期間で行うことができると考えている.」と述べられている． これから分かるように，継続的インテグレーションはソフトウェアの統合およびビルドを頻繁に行い，そのフィードバックから問題をいち早く発見し改善するためのプラクティスである．また，ソフトウェアの状態をビルドの度にチーム全体で共有することも可能となる．継続的インテグレーションでのビルドとは，ソフトウェアの生成，テスト，インスペクション，デプロイを行う一連の活動のことを指す．したがって，頻繁にビルドを行うことが品質の向上にもつながる．

文献 [6] によると，ソフトウェア開発において，継続的インテグレーションは以下のような価値をもたらす．

(1) リスクを軽減する
(2) 繰り返しが多い手作業を軽減する
(3) いつでも，どの環境にもデプロイできるソフトウェアを生成する
(4) プロジェクトの可視性を改善する
(5) ソフトウェア製品に対する開発チームの自信を深める

本研究では，以上の項目に着目し，継続的インテグレーションのプラクティスを複数の関係者が関わる形式モデルの作成に応用した．

また，継続的インテグレーションではそのプラクティスを実践するためのツールが非常に重要な役割を果たす．継続的インテグレーション実施のためのツールの代表的なものに，Jenkins[7] がある．Jenkins では，自動ビルド，自動テスト，テスト結果のメール通知など様々な機能を持っており，継続的インテグレーションのプラクティスを強力にサポートする．Jenkins は本研究のツール化の部分で使用した．

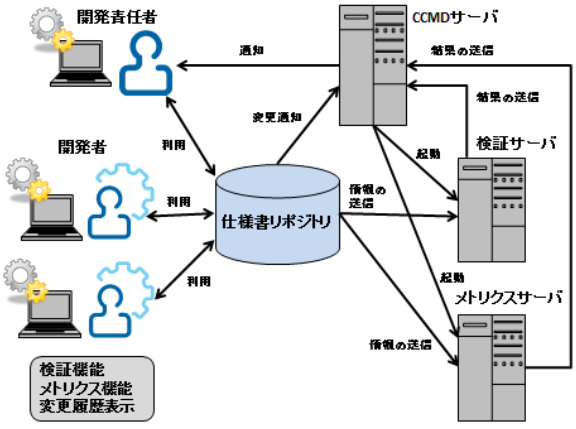

図 2 ツールの概念図
Fig. 2 Concept of tool

開発者は，ツール上で形式モデルを記述し，構成管理システムを使用して仕様書リポジトリにモデルのファイルを集約する．ツールのバックエンドでは，仕様書リポジトリが更新される度に各種サーバがデータの収集・分析を行う．サーバ上で収集・分析されたデータはメールにより開発者にフィードバックされる．以上のような仕組みにより，モデルの作成者はツール上でモデルを作成し構成管理ツールでそれらのモデルを管理するだけでモデルの成熟度を知ることができる．以降では，フロントエンド，バックエンド部分の詳細について述べる．

4.2 フロントエンド

ツールのフロントエンド部分は文献 [15] で提案されているツール（以降，ドメイン辞書管理ツールとする）をベースに構築されている．ドメイン辞書管理ツールは，IBM が開発した統合開発環境 Ecllipse のプラグインで，形式モデルの作成を支援するツールである．モデル作成者は仕様書中のキーワードをドメイン辞書管理ツール上で辞書の見出し語として登録する．ドメイン辞書管理ツールは，辞書に登録されたキーワードをもとに仕様書中の単語の検証や VDM++モデルの雛型の出力などを行う．ドメイン辞書管理ツールの外観を**図 3** に示す．ドメイン辞書管理

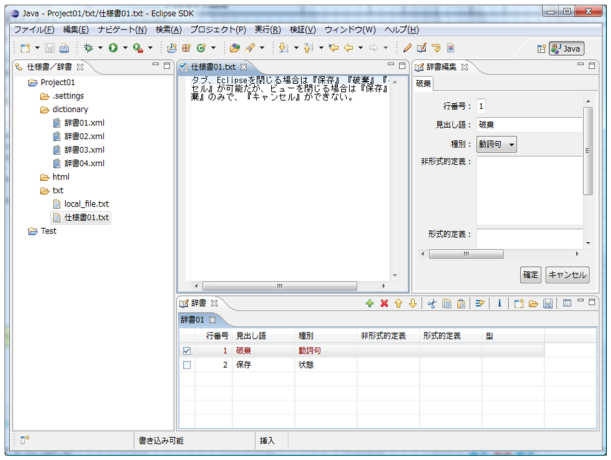

図 3 フロントエンド
Fig. 3 Frontend

ツールで作成・編集したモデルは構成管理システム上のリポジトリに補完する．構成管理システム上でモデルを管理することで複数人での作業を円滑に行うことができる．また，Eclipse には構成管理ツールの連携用プラグインや VDM++の構文チェックや型チェックなどの検証が可能な Overture[16] が存在する．そのため，これらのプラグインを導入することでモデル作成者は Eclipse のみでモデル作成のすべての作業を進めることができる．

4.3 バックエンド

ツールのバックエンド部分では，仕様が一ヶ所で管理されており，関係者が最新の仕様を参照できる．リポジトリでは，検証とモデルの成熟度評価が並列かつ自動的に行

われる．それぞれは，無矛盾性と成熟度の異なる観点での評価である．従来の方法に比べて，時間や人手のコストは増えない．これらの目的を達成するために，バックエンドでは継続的インテグレーションツールの一種である Jenkins[7] が使用されている．Jenkins は，ソフトウェアのビルドや cron で起動するジョブなどの繰り返しのジョブの実行を監視するツールである．cron とは，ジョブ（スクリプト）を自動実行するためのデーモンプロセスのことである．Jenkins はサーバ上で動作し，実行されたビルドやエラーの通知は，メールや Web ブラウザ上で確認することができる．Web ブラウザ上での Jenkins の外観を**図 4** に示す．ツールでは，仕様書リポジトリを Jenkins を使用し

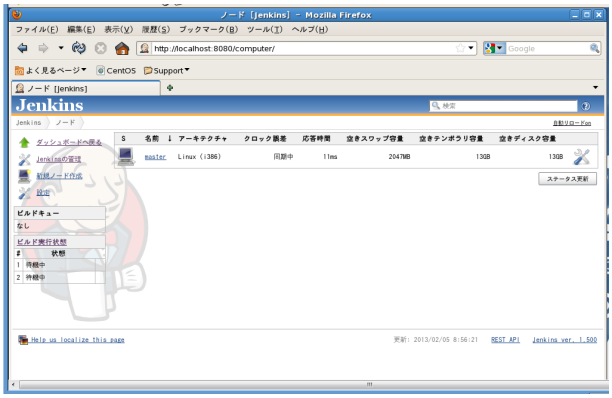

図 4 Jenkins
Fig. 4 Jenkins

て監視し，リポジトリの変更時に特定のディレクトリ下に配置された shell スクリプトを実行する．ツールは形式モデルの成熟度測定のための指標を得るためのデータを収集する処理が記述されたスクリプトやモデルの検証およびテスト用のスクリプトを提供する．また，ツールの仕様者は任意のスクリプトファイルを用いて Jenkins によりモデルの監視や検証を行うことが可能である．例えば，プロジェクト内で取り決められた命名規則などをチェックするためのスクリプトファイルを用意し，命名規則を守っていないモデルがリポジトリに更新された場合にメールにより通知することが可能である．このように自由度の高いカスタマイズが可能なため，あらゆるプロジェクトに柔軟に対応することができる．

5. おわりに

本研究では，形式モデルの作成の際に，モデルの成熟度を測定するための指標を提案した．これらの指標は，モデルの作成中に測定し記録されることで，その阿多愛の推移からモデルの成熟度を判断する．また，モデル作成者の負担を小さくするために，指標はツールにより自動収集されモデル作成者にフィードバックされる仕組みを開発した．これにより，プロジェクト内での形式モデル作成の管理が容易になると考える．一方，今後の課題として，提案した指標を実際の形式モデル作成で測定しその有効性を評価する必要がある．

謝辞 ツール開発に協力して下さった吉村康晴氏をはじめとする九州ビジネス株式会社のみなさまに感謝申し上げます．また，本研究の一部は，JSPS 科研費 24220001，基盤研究（S）「アーキテクチャ指向形式手法に基づく高品質ソフトウェア開発法の提案と実用化」の助成を受けたものです．

参考文献

[1] 中島 震：ソフトウェア工学の道具としての形式手法，*NII Technical Report* (2010).
[2] 佐原 伸：形式手法の技術講座，ソフト・リサーチ・センター (2008).
[3] ジョン・フィッツジェラルド，ピーター・ゴルム・ラーセン，ポール・マッカージー，ニコ・プラット，マーセル・バーホフ，酒匂 寛 訳：VDM++によるオブジェクト指向システムの高品質設計と検証，翔泳社 (2010).
[4] 栗田太郎：携帯電話組込み用モバイル Felica IC チップ開発における形式仕様記述手法の適用，情報処理， Vol. 49, No. 5, pp. 506–513 (2008).
[5] Martin Fowler: Continious Integration, Martin Fowler (online), available from ⟨http://www.martinfowler.com/articles/continuousIntegration.html⟩ (accessed 2013-1-25).
[6] ポール・M・デュバル，スティーブ・M・マティアス，アンドリュー・グローバー，大塚 康史, 丸山 大輔, 岡本 裕二, 亀村 圭助 訳.：継続的インテグレーション入門，日経 BP 社 (2009).
[7] Jenkins: http://jenkins-ci.org/.
[8] 独立行政法人 情報処理推進機構ソフトウェア・エンジニアリング・センター：ソフトウェアエンジニアリングの実践〜先進ソフトウェア開発プロジェクトの記録〜，翔泳社 (2007).
[9] EASE プロジェクト：http://www.empirical.jp/.
[10] 大平雅雄，横森励士，坂井 誠，岩村 聡，小野英治，新海 平，横川智教：ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム，電子情報通信学会論文誌 D-I，No. 2, pp. 228–239 (2005).
[11] EASE プロジェクト：EPM 利用者マニュアル，国立大学法人奈良先端科学技術大学院大学（オンライン），入手先 ⟨http://www.empirical.jp/EASE_DVD/Tools/EPM/Release/help/index-j.htm⟩（参照 2013-2-4）.
[12] 松村知子，勝又敏次，森崎修司，玉田春昭和，大杉直樹，門田暁人，楠本真二，松本健一：自動データ収集・可視化ツールを用いたリアルタイムフィードバックシステムの構築と試行，奈良先端科学技術大学院大学テクニカルレポート (2007).
[13] Basili, V.: Using Measurement to Build Core Competencies in Software, *Seminar sponsored by Data and Analysis Center for Software* (2005).
[14] CSK: VDM++言語マニュアル，CSK（オンライン），入手先 ⟨http://www.vdmtools.jp/uploads/manuals/langmanvice_a4J.pdf⟩（参照 2012-1-30）.
[15] 大森洋一，荒木啓二郎：自然言語による仕様記述の形式モデルへの変換を利用した品実向上に向けて，情報処理学会論文誌，Vol. 3, No. 5, pp. 18–28 (2010).
[16] Overture: http://www.overturetool.org/.