解　説

ソフトウェアメトリクスとデータ分析の基礎

阿萬 裕久　野中 誠　水野 修

ソフトウェアメトリクスは，ソフトウェアの品質マネジメントを実践する上で必要不可欠な存在である．しかしながら，実際のところ広く積極的に活用されているとまでは言い難い．その背景には“何を測り，どう活用するのか？”というシンプルではあるが容易でない問題がある．本論文はそのための一助として，ソフトウェアメトリクスとそこでのデータ分析の基礎，特に，どういったソフトウェアメトリクスや数理モデルがあり，分析で何に気を付けるべきかを中心に解説を行っている．また，ソフトウェアメトリクスの円滑な活用に役立つツールもいくつか紹介している．

Utilization of software metrics is essential to successful software quality management. However, software metrics are sometimes misused, abandoned or unconsidered in practice. To avoid these unwanted consequences, we need to answer the simple but difficult problem about “what to measure and how to utilize.” This article discusses fundamental topics which include classification of metrics, measurement theory, statistical models and their pitfalls. The article also provide some useful information about tools which help us make use of software metrics.

1　はじめに

ソフトウェアメトリクス (software metrics) (単にメトリクス，メトリックスとも呼ぶ) は，ソフトウェアライフサイクルのさまざまな特性から機械的あるいはアルゴリズム的に派生した定量的な計測規準 (尺度) である [53]．端的に言えば，ソフトウェアやその開発過程・環境を測定対象とした定量的尺度である．

ソフトウェア工学の分野において，ソフトウェアメトリクスの概念そのものは古くから存在しており，決して新しいトピックではない．実際，古典的な文献としては 1970 年代に発表された Halstead のソフトウェアサイエンス (software science) 理論 [29] や McCabe のサイクロマティック数 (cyclomatic number) [54] が挙げられる．(そして今もなお，それらの一部は現役のメトリクスとして使われている [33].) そのように比較的歴史の長い分野ではあるが，残念ながらソフトウェアメトリクスが開発現場で積極的に活用されているとは必ずしも言い難い†1．実際のところ，“何をどうやって測ればよいのか”，“どのように活用したらよいのか” といった疑問が常につきまとう．

メトリクスを効果的に活用するには，メトリクスとそのデータ分析に関する適切な知識が必要不可欠である．そこで本論文では，メトリクスに関する基礎知識やデータ分析で注意すべき事項を中心として，開発現場のデータ分析に従事する実務者や研究者向けの解説を行う．まず，メトリクスの基礎知識について説明し，その上で測定データ分析において注意すべき点，並びに関連する数理モデルについて解説する．また，メトリクスによる測定とモデルによる活用を円

Fundamentals of Software Metrics and Their Data Analysis.

Hirohisa Aman., 愛媛大学大学院理工学研究科, Graduate School of Sc. and Eng., Ehime University.

Makoto Nonaka, 東洋大学経営学部, Faculty of Business Administration, Toyo University.

Osamu Mizuno, 京都工芸繊維大学大学院工芸科学研究科, Graduate School of Sc. and Tech., Kyoto Institute of Technology.

コンピュータソフトウェア, Vol.28, No.3 (2011), pp.12–28.
[解説論文] 2010 年 12 月 17 日受付.

†1　より広義に “定量的手法” という観点で見ても，それを実践している企業は全体の約 32% にすぎないという報告もある [90].

滑に遂行するための関連ツールについてもいくつか紹介する．最後に，最近の研究で見られるデータ収集の傾向について簡単に紹介する．

2 メトリクスの基礎知識と注意点

本節ではメトリクスに関する基礎知識と注意すべき事項について説明する．まず，2.1 節で用語の定義を行い，2.2 節で代表的なメトリクスをいくつかの分類に沿って紹介する．そして，2.3 節では測定の前に重要な “対象属性の明確化” についてふれ，2.4 節では測定後でのデータの取り扱いについて，その注意すべき点を述べる．

2.1 用語の定義

ここでは，本論文の内容と関係の深いいくつかの用語について定義しておく．以下はいずれも “ソフトウェアやその開発，使用に関する” という文脈であることに注意されたい．

まず，人手又は自動的な手段によって，定量的又は定性的に識別できる実体の物理的又は概念的な特徴を属性 (attribute) という [35]．そして，属性に対して数値や記号 (分類) を対応付ける (決定する) 操作を測定 (measurement) といい，さらには測定の結果を測定値，測定値が割り当てられる変数を測定量 (measure[†2]) という [36]．なお，1 つ又は複数の属性をある観点から整理し，分類したものを特性 (characteristic) という [36]．

測定において，属性と測定値を対応付ける規準 (ものさし) のことを尺度 (scale) [36], [78] という．また，ある特定の尺度に関して，属性の定量化に使用する一連の操作の一般的な記述を測定方法 (measurement method)，具体的な操作の集合を測定手続き (measurement procedure) [36] という．

そして，ここでいう尺度と測定方法の両方を合わせた概念をメトリクス (metrics)[†3] という [35], [78]．つまり，メトリクスは測定規準 (尺度) のみならず，それを用いてどのように測定を行うのかという方法も含んでいる．その意味で 1 節で示したメトリクスの定義は用語の定義として限定的であったが，初学者に対して直感的な導入イメージを提供するという目的，さらには 2.2.4 節で後述する scale type としての尺度との混乱を避けるという目的で，あえて使用したものであった．以下の議論でも特に言及はしないが，メトリクスという用語には測定方法の概念も含まれているという意味であることに注意されたい．

その他，本論文では関連する用語としてモデル，品質マネジメント，品質管理という用語を使用している．

モデル (model) とは 1 つ以上の測定量をそれに関連する判断基準と結合するアルゴリズム又は計算のことであり [36]，本論文では主として統計学に基づいたモデルについて論じている (3 節)．

品質マネジメント (quality management) は品質管理 (quality control) を含む上位概念であり，前者が品質に関して組織を指揮・管理することであるのに対し，後者は設定された品質基準を満足させることである [37]．つまり，品質基準そのものを設定し，それを満足するように品質管理を実施するという活動全体が品質マネジメントである．

2.2 メトリクスとその分類

一般にメトリクスは，プロダクトメトリクス，プロセスメトリクス及びリソースメトリクスの三種類に大別される [53]．

1. プロダクトメトリクス (product metrics)
   ソフトウェア開発のプロダクト (成果物) について測定を行うメトリクスである．
2. プロセスメトリクス (process metrics)
   ソフトウェアライフサイクルの各フェーズあるいはライフサイクル全体における活動について測定を行うメトリクスである．
3. リソースメトリクス (resource metrics)
   ソフトウェア開発で投入・消費されるリソース (資源) について測定を行うメトリクスである．

概していえば，開発プロセスが続いていく中で，そこで投入・消費されるのがリソース，生み出されるの

†2 名詞扱いである．

†3 厳密には，単一のものがメトリック (metric)，その集合がメトリクスであるが，本論文では表記をメトリクスに統一している．

表 1　メトリクスの例

測定対象	メトリクス例
プロダクト	ユースケース数，ファンクションポイント，LOC，サイクロマティック数 (制御フローの複雑さ)，Halstead メトリクス，パブリックインスタンスメソッドの数，インスタンス変数の数，特殊化指数 [48]，CK メトリクス (WMC：複雑さで重み付けされたメソッド数，CBO：オブジェクト間の結合度，RFC：クラスに対する応答集合の大きさ，DIT：継承木の深さ，NOC：子クラスの数，LCOM：クラスにおける凝集度の欠如；バリエーションがいくつかある [15])，メソッド呼出しの数，クラス内に属性等の型として登場するクラス数・種類数 (クラス間結合度) [46]，モジュールのファンイン・ファンアウト数，情報フローの複雑さ (Henry-Kafura メトリクス)，コードクローンの個数・最大長・割合 [10], [57]，コメント文記述の密度・頻度 [5], [7]，テストケース数，テストケースの網羅率，欠陥密度
プロセス	工数，工期，生産性，顧客との対話時間，インタビュー実施回数，開発開始後の仕様変更回数・変更率，ソースコードの変更回数・変更率，プロトタイピング実施の有無，静的解析ツールの使用・不使用，コードレビュー実施の有無，レビュー効率，欠陥除去率
リソース	(投入可能な) 工数，工期，費用，(開発要員の) 生産性，経験年数

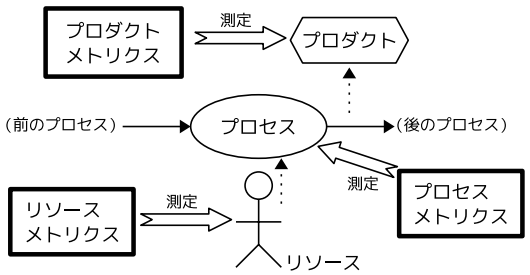

図 1　プロダクトメトリクス，プロセスメトリクス及びリソースメトリクスにおける測定の位置付け

がプロダクトであり，上述した三種のメトリクスはそれぞれを測定対象としている (図 1)．基本的なメトリクスの例を表 1 に示す．

また，メトリクスには “尺度の種類” 及び “現場での制御可能性” という属性もあり，詳しくは後述するが測定値を利用する上でそれぞれ重要な意味を持つ．

- 尺度の種類
  そのメトリクスは，名義尺度，順序尺度，間隔尺度並びに比率尺度のいずれであるか？
- 現場での制御可能性
  そのメトリクスは，現場での作業従事者が主体的かつ直接的に制御可能な量を測定しているのか？それとも観測可能な量にすぎないのか？

以下，2.2.1 ～ 2.2.3 節ではプロダクトメトリクス，プロセスメトリクス及びリソースメトリクスについてそれぞれより詳しく述べ，2.2.4 節及び 2.2.5 節で尺度の種類及び現場での制御可能性についてそれぞれ説明する．

2.2.1　プロダクトメトリクス

ソフトウェア開発のプロダクトを測定対象としたメトリクスであり，代表的なものとしては，

- ファンクションポイント (function point) 法 [1], [2]：機能仕様書を対象としてシステムの機能規模を測定†4，
- Henry-Kafura メトリクス [32]：モジュール設計を対象としてそこでのファンイン数及びファンアウト数をもとに情報フローの複雑さを測定，
- Chidamber-Kemerer(CK) メトリクス [19]：オブジェクト指向設計を対象としてその複雑さや継承木の深さ，結合度 (coupling)，凝集度 (cohesion) といった複数の属性を測定，
- McCabe メトリクス (サイクロマティック数) [54]：フローチャート (またはソースコード) を対象としてそこでの制御フローの複雑さを測定，
- Lines Of Code(LOC)：コードの長さ (規模) を

†4　正確には，ファンクションポイント法は特定のメトリクスを意味するものではない．実際，目的や分野に応じて IFPUG 法 [34]， SPR 法 [77]，NESMA 法 [61]，COSMIC-FFP 法 [20] といった複数の測定法があり，それらの測定法に従ってファンクションポイントが算出されるようになっている．

測定,
が知られている.

これまでに提案及び研究されてきているメトリクスの大半はプロダクトメトリクスであるといっても過言ではない[53]. 実際, “メトリクス　プロダクトメトリクス” という印象を持つ読者も少なくないであろう. 特にソースコードを測定対象としたメトリクスは数多く存在する. 通常, ソースコードは, 組織的管理の有無に関わらず記録・保管される成果物であり, ソフトウェア技術者・研究者にとって最も馴染み深い存在であること, さらにはそれを入力として機械的な測定作業 (ツール化) も比較的容易であるということがプロダクトメトリクス台頭の主な理由である.

プロダクトメトリクスは対象成果物の属性を定量的に把握するものであり, 得られた情報を解析することで品質の評価や予測が行われる. 例えば, あるモジュールのソースコードをいくつかのメトリクスで測定し, そこで得られた測定値をある種の統計モデルに入力として与えることで当該モジュールに “フォールトの潜在が疑われるか (Fault-prone であるか)” を判別したり, その疑惑度を算出したりといった研究が盛んに行われている[65]. さらにはその結果をテスト計画やレビュー計画に役立てる[8],[41] といった応用も研究されている.

2.2.2 プロセスメトリクス

ソフトウェアライフサイクルの各フェーズにおける活動を測定対象としたメトリクスであり, 主としてその活動における

- 時間
- コスト
- 工数
- 関連イベントの有無
- 関連イベントの発生回数
- 関連手法やツールの使用・不使用

などが測定される. 例えば, 顧客との対話時間やインタビュー回数, 顧客からの変更要求の回数, 仕様書のレビューに要した工数, レビューで指摘された欠陥数[†5], 仕様書完成までに費やした総コストといったものが挙げられる. また, “Yes/No” を問う (質的データである) ものとして, 要求分析段階でプロトタイピングを行ったかどうか, コーディング時に静的解析ツールを使用したかどうか, コードレビューを行ったかどうか等も挙げられる. ツール使用の有無といった質的データは工数や時間と比べて単純なものではあるが, 後工程でのテスト工数や成果物の品質に影響を及ぼす場面も珍しくなく[49], 有意義なメトリクスとなる場合もある.

端的に言えば, プロセスメトリクスは活動の記録であり, それまでの活動の定量的な把握・分析を通じて状況判断や評価, 予測 (見積り) に役立てられる. 代表的な例としては生産性の評価が挙げられる. 前述のプロダクトメトリクスによって成果物の規模が算出され, プロセスメトリクスによって時間や工数が算出される. そして, 単位時間あたりの規模や単位工数あたりの規模というかたちで生産性が評価される.

他にも, 1 つの活動, 例えばレビュー (インスペクション) に着目し, そこでの欠陥除去率やレビュー速度, 単位工数あたりでの指摘欠陥数といったメトリクスを用いて活動の効果と効率を定量的に評価し, 開発管理に役立てる[63] といった活用がある.

また, 開発プロジェクトそのものが目標を達成できずに失敗に終わったり, 失敗とまではいかないまでも内部で大変な調整[†6]が必要であったりといった状況について, アンケートを通じて要因を分析し, プロジェクトの混乱を未然に防ぐといった研究もある[30].

2.2.3 リソースメトリクス

ソフトウェア開発作業において投入・消費される資源を測定対象としたメトリクスであり, 人間あるいはその集まり (チーム), 開発環境, ツール等が対象となる[26]. 例えば,

- 開発要員数
- 開発要員の作業単価
- 開発要員のスキルレベル
- 開発用マシンのスペック
- ネットワーク環境の品質及びコスト
- コミュニケーションツールの品質及びコスト

†5 “仕様バグ” とも呼ばれる.

†6 俗にいう “火消し” を意味している.

表 2　主な尺度とメトリクスの例

尺度	表現可能な情報	測定値に許容される処理 (各行，それより上の処理を含む)	メトリクス例
名義尺度	対象間での 等価性	頻度の計上 (頻度，最頻値)	使用言語を “C: 1，C++: 2，Java: 3，COBOL: 4，VB: 5，その他: 6” で表現
順序尺度	対象間での 順序性	大小比較，順位付け (中央値，順位相関， パーセンタイル)	ソフトウェア故障に対してその深刻度を “致命的: 4，重大: 3，普通: 2，軽微: 1” で表現
間隔尺度	対象間での差	加算，減算 (平均，標準偏差，積率相関)	——
比率尺度	対象間での比率	乗算，除算 (幾何平均，変動係数)	コードの長さを物理行数で表現

- 労働環境・条件に関する各種項目 (部屋の広さ，明るさ，労働時間等)

などが挙げられる．またここでも，時間やコスト，工数を測定対象としてよいが，こちらは “投入可能な” 量になる．

2.2.4　尺度の種類

前小節まで代表的なメトリクスについて，それらを測定の対象ごとに大別して紹介してきた．次にどの種のメトリクスにも共通する重要な属性として “尺度の種類” (scale type) について説明する．

いま仮に，何らかのメトリクスに関するいくつかの数値データが得られたとする．そして，それらのデータを使って，当該ソフトウェアの開発規模や構造の複雑さを評価したり，開発工数やフォールトの潜在数を見積もったりといったことが期待されているとする．しかしここで安易に評価値や見積り値を出そうとしてはならない．最初にメトリクスによって得られた値がどういった尺度の下で得られたものであるのかを確認しておく必要がある．さもなくば測定値に対して誤った演算・処理を施してしまい，そこから誤った解釈を導き出してしまったり，不適当な見積り値を算出してしまったりする恐れがある．このような注意点については 2.4 節で後述する．

主な尺度としては，名義尺度，順序尺度，間隔尺度及び比率尺度がある．測定では対象物に数値または記号を割り当てるが，尺度によってそれらの数値・記号に対する解釈の可能性が異なる．それぞれの概要を表 2 に示す．より詳しくは文献 [26], [66], [86] を参照されたい．以下では便宜上，測定データを数値として説明する．“Yes/No” や “A, B, C” といった質的データも数値化されているものとして説明する†7.

- 名義尺度 (nominal scale)
  名義尺度は対象物に数値を名前として割り当てるものであり，値の大きさに意味はない．この場合，各数値の出現頻度には意味があるが，その合計や平均には意味がない．この種のメトリクスでは，何らかの “場合分け” を目的とした使い方は可能であるが，これらを加工して評価値を算出するといった使い方は一般に誤りである．
- 順序尺度 (ordinal scale)
  順序尺度は対象物の間で順序関係が成立するように数値を割り当てるものであり，それによって大小の比較や優劣の比較が可能である．ただし，数値の違いで得られる情報は順序性のみであり，違いの “大きさ” は反映されない．これも名義尺度と同様に，各数値の出現頻度には意味があるが，その合計や平均には意味がない．仮に表 2 のメトリクス例について数値の和を考えた場合，軽微な故障が 3 件と重大な故障 1 件の重要度は同じになるが，果たしてそのような数値の比較にどれほどの意味があるのか疑問である．同様に平均値による議論もできない．
- 間隔尺度 (interval scale)
  間隔尺度は順序尺度の性質を全て満たし，なおかつ値の差にも意味のあるものである．したがっ

†7　記号についても数値と同様に演算を定義すればよいが，議論に際して無用な混乱を招く恐れがあるため，ここでは数値に一本化して説明することにする．

て，測定値の差を使った議論も可能である．ただし，値の比には意味がなく，測定値間の除算を伴うような処理に対して適切な解釈を行うことは難しい．典型的な例としては摂氏や華氏による温度測定が挙げられる．例えば，A, B 及び C の三点がそれぞれ 10°C, 20°C 及び 30°C として測定されたとき，A と B の温度差は B と C の温度差に等しいといえる．測定尺度を華氏に換えても同様のことは成り立つが，C が A の 3 倍熱いとはいえない．

実際には，間隔尺度に相当するようなソフトウェアメトリクスを目にすることはほとんどない†8．

- 比率尺度 (ratio scale)

  比率尺度は間隔尺度の性質を全て満たし，なおかつ値の比にも意味がある．比率尺度の測定値に関しては通常の四則演算が全て許され，それゆえどのような数理モデルにも入力として利用できる．

2.2.5 現場での制御可能性

メトリクスに関してもう 1 つ重要な属性にふれておく．それは，現場での制御可能性であり，メトリクスで得られる情報が “観測可能なだけ” なのか，それとも “制御可能でもある” のかということである．

まず，メトリクスによって測定値が得られているということは，少なくともそれは観測可能な量である．しかし，それが現場で制御可能であるかどうかは別の話である．ここでいう制御可能とは，現場の作業従事者が意識することで直接的に制御できるような量であるかということである．

例えば，モジュールの規模や凝集度，サイクロマティック数，コメント記述の割合，他のモジュールとの結合度といった属性をメトリクスで測定するとすれば，それらの量は観測可能であって，なおかつ制御可能な場合が多い．開発者たちがそれらの値を意識して設計作業やコーディング作業を行ったり，やり直しを行ったりすることで測定値を (ある程度は) 主体的に変えることができる．しかし，例えば顧客から出された機能要求の件数や仕様変更の回数は，現場の開発者が直接関与できないところで意思決定がなされた結果であり，観測可能ではあっても制御可能ではない．

メトリクスを品質管理に活用しようとする場合，現場において作業従事者が制御可能な量とそうでない (観測可能なだけの) 量を分けて考えなければならない．単純な話ではあるが，意外な盲点となる場合もある．例えば，“モジュールの規模” が “リリース後に検出される欠陥数” と相関関係にあるようであれば，それを参考にモジュール規模を評価し，場合によってはモジュールの分割や再設計を行うこともできる．しかし，これが “モジュールの規模” ではなく “顧客による仕様変更の回数” であったとすると，現場での活動では変えることのできない量である．つまり，その種の相関関係を知見として与えられたとしても，現場での作業に直接反映できるものではない．仕様変更の回数からリリース後の欠陥数を予測できるという意味では有益な知見であるが，必ずしも現場での成果物評価や作業改善に活用できるというわけでもない．

いったん数値化されると，その後は数値の動向や関係の解析にのみ注目しがちであるが，目的によっては制御可能性という視点も重要となりうる．

2.3 対象属性の明確化

ここまでメトリクスとその属性について紹介してきた．次に，測定を行うにあたって重要な “対象属性の明確化” について述べる．

やや細かい話になるが，測定に際して具体的には測定対象物の “どの属性を” 測定しようとしているのかを明確にしておくべきである．例えば，あるモジュールを測定対象とした場合，漠然とそのモジュールの品質を測定したいというだけでは不十分である．より具体的なレベルに掘り下げる必要があり，例えばこれを “欠陥密度” というかたちで表すといった定義が必要である．さらにはそのためのモデルが必要であり，ここでは仮に

$$\text{欠陥密度} = \frac{\text{検出された欠陥数}}{\text{モジュール規模}}$$

という式を導入したとする．しかしながら，更なる疑問が浮かんでくる．

- どの段階で検出された欠陥を数えるのか？

†8 間隔尺度となるようなメトリクスを定義することはできる．しかし，筆者らの知る限り，一般に使われるようなメトリクスでこれにあたる例は見あたらない．

- モジュール規模はどういうメトリクスで測るのか？(物理行数なのか？　論理行数なのか？　また単位は LOC なのか？　KLOC なのか？)
- そもそも何をもって欠陥 1 個と数えるのか？[64]

このような疑問点に対して 1 つずつ明確な定義ないし規準を設定していくことが必要不可欠である．さもなければ “品質” というそれらしい言葉の下で実のところどうやって導出されたのかがあやふやな数値が導出され，それがその後の評価や見積りに使われてしまうという危険性がある．

測定における対象属性の明確化とそこでのメトリクスの定義に有効な手法として Goal-Question-Metric (GQM) [12] アプローチが知られている．これは抽象度の高い目的 (Goal) をより具体的な質問 (Question) に分解し，さらに各質問について，それに答えるための具体的なメトリクス (Metric) を定義するというトップダウン方式で対象属性とメトリクスを明確にしていくものである．

- 目的 (Goal)：測定実施者・分析者が，メトリクスによる測定を通じて最終的に示したい事項である．例えば，“モジュールの品質を評価する” 等が挙げられる．
- 質問 (Question)：上記の目的について，その達成度を評価・判定するための質問である．1 つの目的に対して複数の質問を設定してよい．例えば，“モジュールで検出された欠陥の数は適正な範囲のものか？” や “モジュールは複雑な構造になっていないか？” 等が挙げられる．
- メトリクス (Metric)：上記の質問への回答に必要なメトリクスである．1 つの質問に対して複数のメトリクスを定義してよい．例えば，“欠陥密度” や “サイクロマティック数” 等が挙げられる．もちろん，これらのメトリクスの定義においても前述したように詳細まで (何をもって欠陥 1 個と数えるか等) 明確に定義しなければならない．

2.4　測定データ分析における注意点

本節の最後に，測定後のデータ分析における注意点について述べる．メトリクスそのものの純粋な役割は，対象属性の数値化や記号化であるが，当然ながら測定データ (測定の結果) の取り扱いについても無視はできない．通常，現場では測定を目的としているわけではなく，測定データを分析して品質マネジメントやプロセス改善に役立てることを目的としている．それゆえ，測定データの適切な分析や解釈は重要な役割を担っている．ここではデータ分析・解釈において注意すべき事項として，前述した “尺度の種類” という視点から議論する．

前述したように，ソフトウェアメトリクスを利用する場合は，それがどのタイプの尺度であるのかを認識し，その上で測定値を解釈したり，数理モデル・統計モデルの入力として適切に利用すべきである．さもなくばメトリクスの利用は “安易な数値化” に成り下がってしまい，意味をなさなくなる恐れがある．一例として，表 2 に示した順序尺度でのメトリクス例について再び考える．この例では故障の深刻度を整数で表しているが，この場合は値どうしでの大小関係にのみ意味があり，数値が深刻さの度合いを表すものではない．仮に，あるシステムで発生した故障の深刻度が合計で 43 であり，別のシステムでは 35 であったとしても，そのような合計値でもって後者のシステムの方が故障の少ない優れたシステムであるとは言い切れない．合計値は一種の総合評価値のような見方をされがちであるが，安易な数値化の一例であり，誤った分析と言わざるをえない．

改めて演算を基点に述べると，メトリクスが比率尺度であれば四則演算は許されるため特別な配慮は不要であるが，名義尺度・順序尺度・間隔尺度の場合はそれぞれで許される演算に限りがある．特に，名義尺度や順序尺度の場合，測定結果がたとえ数値であったとしても，値の加算すら許されないため，一般の感覚でいうところの演算はほとんど適用不可となる．例えば，順序尺度で得られる測定値の平均を求めたり，相関係数[†9]を求めたり，t 検定を行ったりといった処理はいずれも誤りである．この場合はそれぞれ代わりとして，中央値を求める，順位相関係数を求め

†9　ピアソンの積率相関係数 (Pearson product-moment correlation coefficient) をさす．単に “相関係数 (correlation coefficient)” と呼ぶ場合は，スピアマンの順位相関係数 (Spearman rank correlation coefficient) ではなく，こちらをさすことが多い．

る，ノンパラメトリック検定 (例えばウィルコクソン検定 (Wilcoxon test) [9], [38]) を行うといった処理を検討すべきである．

なお，ここでの尺度タイプの議論の前提として，測定の信頼性が決定的に重要であるということを付け加えておく．測定の信頼性が損なわれると，比率尺度と考えられているものが，間隔尺度，順序尺度あるいは名義尺度へと転落してしまい，結果として限られた演算や解釈しか許されないようになってしまう場合がある．例えば，ソースコードの論理行数や命令数について考える．明確に測定法や測定規準が定められていればいずれも比率尺度であるが，定義に曖昧さが含まれると (例えば，変数宣言は数えるのか，ヘッダファイルの内容は展開するのか，が未定義の場合等)，測定値どうしの間隔が保証されないばかりか，順序性さえも損なわれる恐れがある．

また，いったん比率尺度として測定されたとしても，その後の数値変換次第では尺度タイプが変わる場合もあるので注意が必要である．例えば，比率尺度の例としてソースコードの物理行数を考える．この値域は $[0, \infty)$ であるが，これを処理の都合上 $[0, 1)$ に変換したとしよう．この場合，線形な変換は不可能なため[†10]，値の順序性は保持できたとしても，差や比率は保持されない．それゆえ，変換後の値は順序尺度と同じレベルになり，それらについて平均値や相関係数を使った議論は誤った解釈を導く恐れがある．

3 数理モデル

3.1 メトリクスの利用目的と数理モデル

前述したように，メトリクスを使用することでプロダクトやプロセス，リソースのさまざまな属性を測定可能である．ただし，測定作業や管理にもコストは必要であり，その目的，すなわち “測定値を使って何をしたいのか” を明確にしておく必要がある．やみくもに測定を行ってデータを集めるだけでは徒労に終わる場合も多い[†11]．

これまで研究・実践されているメトリクスの活用としては，品質の見える化とマネジメント，プロセスの評価と改善，工数の見積り，信頼性の評価，Fault-prone モジュールの判別等が挙げられる (表 3)．特に Fault-prone モジュールの判別 (予測) に関する研究は，プロダクトやプロセスの品質評価・改善との関わりも深く，これまでに (現在もなお) さまざまなアプローチが検討され続けている [65]．多くの場合，測定データに対して何らかの数理モデルと適用し，その上で目的に合わせた評価・予測が行われている．次小節では比較的広く用いられている数理モデルについて紹介する．

3.2 代表的なモデル

3.2.1 重回帰モデル

いま，N 個の変数 x_i $(i = 1, \ldots, N)$ の値を使って別の変数 y の値を予測するという問題を考える．換言すれば，x_i を原因とし，y をその結果とした因果関係を考え (期待し)，両者の間の関係式を構築するという問題である．この問題に対して，

$$y = \sum_{i=1}^{N} \alpha_i x_i + \beta \qquad (1)$$

という一次式の関係を仮定し，与えられたサンプルデータを使って適切な係数 α_i 及び β を見つけ出す手法を重回帰分析 (multiple regression analysis) という (図 2)．ただし，実際には (1) 式の等号が全データに対して成立することは希であり，多少なりとも誤差が含まれる．そのため，実際には残差平方和が最小となるように係数が決定される．なお，x_i を説明変数，y を目的変数と呼び，(1) 式を重回帰モデル (multiple regression model) (あるいは重回帰式) と呼ぶ．なお，このモデルはあくまでもサンプルデータの上で見られる説明変数と目的変数の間の関係性を示すものであり，両者の間で “原因-結果” という因果関係が存在していることまでを示すものではない．

†10 仮に線形変換可能として，コード A の変換後の値が 0.5，コード B の値が 0.6 であった場合，A と B を連結して作ったコードの値は線形性から 1.1 となり，値域を超えてしまい矛盾が生じる．

†11 まったく無駄というわけではなく，測定して記録を取ることで当該属性に対する意識が高まり，そのことが間接的・心理的に良いフィードバックとなる可能性も否定はできない．

表 3　メトリクスの主な活用

目的	概要	活用例
品質の見える化とマネジメント	ソフトウェア属性の状況を定量的に把握し，品質の総合的な評価や組織的なマネジメント [92] を行う.	開発中のプロダクトを測定し，メトリクスの一般的・平均的な値や範囲との比較 [48], [87] を通じて品質評価や改善点の洗い出しを行う.
プロセスの評価と改善	測定値の整理・分析を通じてプロセスでの活動内容や進捗状況を評価したり，継続的な評価に基づいてプロセスを改善したりする.	レビューの作業効率や効果を評価する [63]. 開発プロジェクトの統計データ [40] を参考にプロジェクトの管理・運営を見直す. CMMI [18] “測定と分析” プロセス領域のプラクティスを実践する.
工数の見積り	仕様や設計，開発リソースの属性を測定し，得られた測定値をもとに開発工数を見積もる [62], [84].	COCOMO モデル [13], [14] を用いて工数 (人月) を見積もる. 測定値に基づいて過去の類似プロジェクトを抽出し，工数を類推する [75], [85].
信頼性の評価	プロダクトやテストプロセスの属性を測定し，潜在フォールト数や平均ソフトウェア故障時間間隔 (MTBF) を予測する.	ソフトウェア信頼度成長モデル [91] に基づいて残存フォールト数を予測したり，最適なリリース時期を決定 [94] したりする.
Fault-prone モジュールの判別	レビューやテストに先立って，フォールトの潜在が疑われる (Fault-prone) モジュールを測定値によって判別 (予測) する [65].	CK メトリクス [19] とロジスティック回帰モデルを用いて Fault-prone モジュール (ここではクラス) を判別する [11]. 同様に複数のメトリクスとベイジアンネットワークを用いて Fault-prone モジュールを判別する [71].

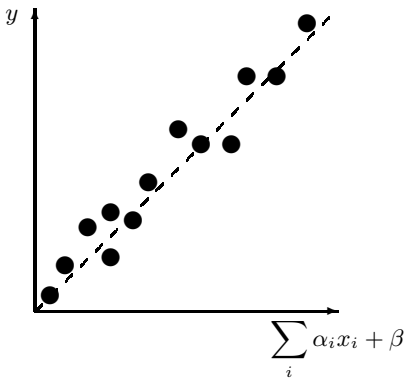

図 2　重回帰モデルのイメージ図

モデルの解釈ではこの点にも留意すべきである.

このモデルの具体的な活用例としては，いくつかのメトリクスを説明変数とし，それらのメトリクスが関係していると思われる変数 (属性) を目的変数に設定するといった例が挙げられる．例えば，開発プロジェクトの途中で測定されたいくつかのメトリクスを説明変数とし，そのプロジェクトの終了までに要する開発工数や最終的なコード規模を目的変数とした活用等がある [16], [62], [80]．ただし，文献 [56] のようにあらかじめ変数を対数変換して (例えば “$\log\ y$” として) 使用することも多い．統計分析モデルによってはデータの分布として正規分布を仮定している場合も多く，データ分布を正規分布の形に近づけることが対数変換の理由である.

なお，説明変数の中で互いに強い相関を持つような変数対が存在していたり，1 つの変数が他の変数の一次式で表現 (近似) 可能であったりすると重回帰モデルを適切に構築できない．これを多重共線性 (multicollinearity)[†12] という．重回帰モデルを構築する際には，多重共線性に注意しつつメトリクスの取捨選択を行う必要がある.

3.2.2　ロジスティック回帰モデル

上述の重回帰モデルと同様に，N 個の説明変数 x_i $(i = 1, \ldots, N)$ の値を使って目的変数 p の値を予測するという問題を考える．ただし，p は区間 $(0, 1)$ 内の値をとり，ある事象の発生確率 (あるいは割合) を表すものとする．この問題に対して,

†12　“マルチコ” と呼ばれることもある.

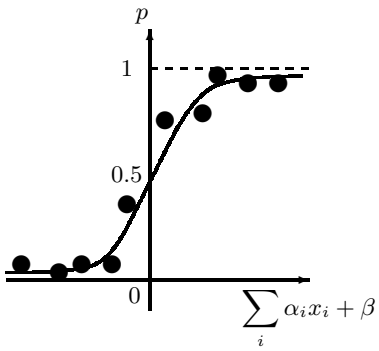

図 3 ロジスティック回帰モデルのイメージ図

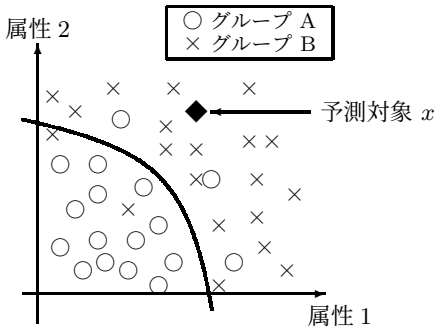

図 4 判別分析モデルのイメージ図

$$\log \frac{p}{1-p} = \sum_{i=1}^{N} \alpha_i x_i + \beta \qquad (2)$$

という重回帰モデルに似た式を仮定し，与えられたサンプルデータを使って適切な係数 α_i 及び β を見つけ出す手法をロジスティック回帰分析 (logistic regression analysis) という．そして，(2) 式をロジスティック回帰モデル (logistic regression model) と呼ぶ (図 3)．なお，この式は次のかたちで書かれることが多い：

$$p = \frac{1}{1+e^{-Z}}, \qquad Z = \sum_{i=1}^{N} \alpha_i x_i + \beta\,. \qquad (3)$$

ここで $F(Z) = 1/(1+e^{-Z})$ を Z のロジスティック関数という．ロジスティック回帰モデルは，説明変数の合成変量 Z とモデル化の目的である確率 (割合) p とをロジスティック関数で結合させたモデル [82] と解釈できる．

このモデルの活用例としては，重回帰モデルと同様にいくつかのメトリクスを説明変数とし，それらのメトリクスが関係していると思われる事象の発生確率や割合を目的変数に設定した例が見られる．代表的な活用例の 1 つとしては，ソフトウェアモジュールにおけるフォールト潜在の疑わしさ (fault-proneness) を目的変数として Fault-prone モジュールの予測を行う†13といったものが挙げられる [10], [11], [31], [45].

†13 ただし，Fautl-prone モジュールの予測モデルがロジスティック回帰モデルに限定されるというわけではないことに注意されたい．後述する文献 [58] のように，他のモデルを用いた Fautl-prone モジュール予測も行われている．

3.2.3 判別分析モデル

いま，N 個の属性を持った N 次元データがいくつか与えられ，それらが複数のグループに分類できるとする．その N 次元空間上でグループを区分けするための超平面ないし超曲面を (与えられたデータから) 見つけ出す手法を判別分析 (discriminant analysis) という．超平面・超曲面の方程式が得られることで，新たなデータ x が与えられた際に x がどのグループに属するのかを判別できる (図 4)．これは一種の予測法として活用できる．

例えば，モジュールを N 個のメトリクスで測定して N 次元データとして表現し，そこでのフォールトの有無でもってデータのグループ分けを行う．そして，過去の実績データを元にフォールト有りのグループとフォールト無しのグループとを分けることができるような超平面・超曲面を見つけ出す．その後，未検査のモジュールが与えられた場合，N 次元空間上でどちらのグループ側に位置することになるかによってそのモジュールのフォールト潜在を予測するという活用がある [58]．同様にフォールトの有無ではなく，後のコード変更量の大小について予測するといった活用もある [55].

一般的には各グループの平均となる点 (ベクトル) を求め，それらの中間に超平面ないし超曲面を設けることになる．中間位置はユークリッド (Euclid) 距離またはマハラノビス (Mahalanobis) 距離を使って求められる．マハラノビス距離はデータのばらつき

を考慮した距離概念であり，ユークリッド距離はマハラノビス距離の特殊ケース (グループ間で分散が等しい) と考えることもできる．

また，判別分析モデルの変形版として，正常グループ†14からのマハラノビス距離によって評価対象モジュールの異常度を算出するマハラノビスタグチ法 (Mahalanobis-Taguchi method) [79], [6] という手法もある．

3.2.4 成長曲線モデル

フォールトの検出数や故障の発生回数を時間の経過とともに観測し，その変化を成長曲線 (growth curve) としてモデル化するという手法がある (図 5)．代表的なものとしてソフトウェア信頼度成長モデル (software reliability growth model) [91] があり，フォールト検出数 (あるいは故障発生回数) の予測や平均故障発生時間間隔 (MTBF:mean time between failures) の算出といった信頼度の定量的評価に利用できる．

モデルとしてロジスティック曲線 (logistic curve) やゴンペルツ曲線 (Gompertz curve) が使われる場合もある [93] が，それらはフォールト検出等の傾向を経験的に表したものであって理論的な妥当性には疑問もあり，一般には確率的な計数過程のモデルが使われている [70]．例えば，フォールトの検出 (あるいは故障の発生) 事象を 1 つの確率事象と見なし，その生起の様子をポアソン過程といったかたちでモデル化するものがある．ただし，単位時間に一定のペースでフォールト検出 (故障発生) が起こり，それが持続するという状況はソフトウェア開発では考えにくい．そのため，そのような検出 (発生) ペースが時間とともに変化する非同次ポアソン過程 (NHPP:non-homogeneous Poisson process) モデルが一般的である．最も基本的なモデルとして，時刻 t の時点で検出済みのフォールト数 (期待値) を $H(t)$，総フォールト数を a とし，単位時間あたりに検出されるフォールト数がその時点での残存フォールト数に比例する，つまり

$$\frac{dH(t)}{td} = b(a - H(t))$$

という微分方程式を解いて得られる指数形ソフトウェア信頼度成長モデル (exponential software reliability growth model) [28]

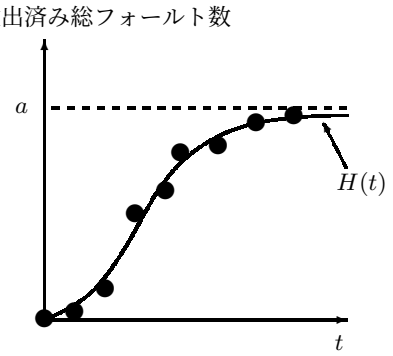

図 5　成長曲線モデルのイメージ図

$$H(t) = a(1 - e^{-bt}) \ , \qquad (4)$$

がある†15(ただし，b は 1 個あたりのフォールト検出率)．また，フォールト間の依存関係を考慮してフォールトの検出能力が時間とともに変化するという概念を採り入れた，より複合的な習熟 S 次形ソフトウェア信頼度成長モデル (inflection S-shaped software reliability growth model) [67]

$$H(t) = \frac{a(1 - e^{-bt})}{1 + ce^{-bt}} \qquad (5)$$

といったものもある (c はテスト能力の習熟傾向を表すパラメータ)．近年では，これまでの代表的なソフトウェア信頼度成長モデルを包括した，統合モデル [27] や一般化ガンマソフトウェア信頼性モデル [68] 等も研究されている．

3.2.5 計数データの解析モデル

上述の成長曲線モデルとも関係するが，メトリクスによって得られるデータは，フォールト検出数のようにある種の件数や個数であることも多い．そういったデータは計数データ (count data) と呼ばれる．ソフトウェア開発で得られる計数データの分布では，歪みのある，右裾の長い分布 (歪度 > 0) が見られることも多く [40]，そのままでは重回帰モデル等の変数にできない．そのため，通常は対数変換といった前処理が行われるが，それでもなおデータにおける分散の均一

†14　フォールト無しであったり，高品質なものであったりするものを想定している．

†15　正確には (4) 式は同モデルでの平均値関数である．(5) 式についても同様である．

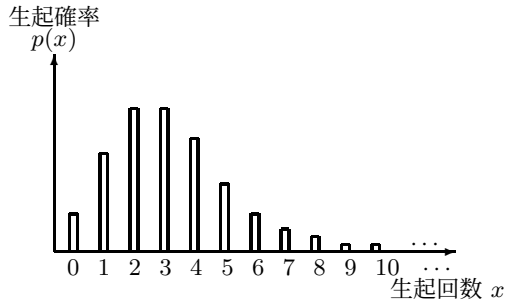

図 6 ポアソン分布の例

性や誤差の正規性が保証されるとは限らず，厳密には重回帰モデルの使用が不適となることもある．

この種の問題への対策として，計数データを対象とした統計解析モデル[39] の活用が 1 つの選択肢として挙げられる．例えばフォールトの検出数といった目的変数についてポアソン分布 (Poisson distribution) (図 6) を仮定し，そのパラメータを回帰モデルによって推定するポアソン回帰モデル[81] がある．また，ポアソン分布では期待値と分散が等しいことが条件として求められるが，これが成立しない場合には (二項分布を拡張した) 負の二項分布 (negative binomial distribution) [39] による負の二項回帰モデル[44] の利用も可能である．

3.2.6 ベイズ統計モデル

これまで紹介してきた統計モデルは，いずれも，特定母集団におけるパラメータを定数として，それをサンプルデータによって推定するというモデルであった．一方，ベイズ統計 (Bayesian statistics) モデル[52] は，推定したいパラメータが何らかの確率分布 (事前分布) に従うと仮定し，事前分布を考慮した上で，特定のインスタンスに関するパラメータの確率分布 (事後分布) をそのデータによって推定するというモデルである．また，複数のパラメータの構造をネットワークモデル (ベイジアンネットワーク (Bayesian network) モデル[76]) で表現し，パラメータ同士の関連を考慮した上で推定することも可能である (図 7)．このようにベイズ統計モデルは他の統計モデルとその特徴が大きく異なる．

事前分布を持ったパラメータをベイジアンネットワークモデルとして表現することで，評価対象のデータに欠損値があった場合でも予測値を得ることができる．例えば，独立テストで検出されるフォールト数を予測するにあたって，ソフトウェア規模に加えて要員スキル等のリソースメトリクスと手戻り度等のプロセスメトリクスを含んだネットワークモデルが得られているとする[25]．このとき，評価対象のプロジェクトデータにおいて，一部のプロセスメトリクスでの測定値が欠落していたとする．この場合にあっても，当該メトリクスの事前分布及びこれに関連するメトリクスとの関係性に基づいて，最終的な関心事である“独立テストで検出されるフォールト数” の予測値を得ることはできる．ただし，ベイジアンネットワークモデルを構築するにあたっても，他の多くの統計モデルと同様に多量のデータが必要であったり，精緻な因果関係を構築すればするほど適用可能なデータのクラスが狭まるという課題もある．実際には，経験的に妥当と考えられる因果関係に基づいてネットワークモデルを作成し，データを用いながらパラメータの分布を学習していく方法がとられる場合がある．

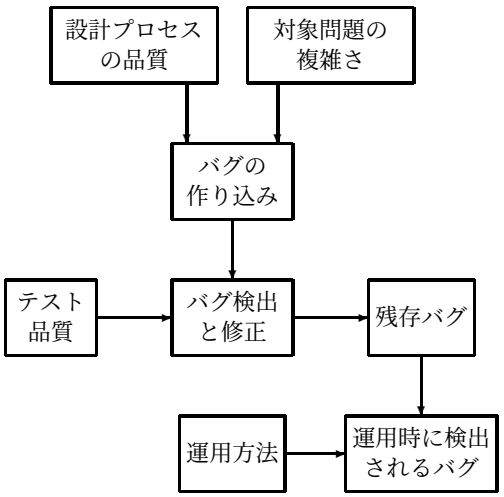

図 7 ベイジアンネットワークの例
(文献 [24] より抜粋; 著者らで表現を一部変更)

4 関連ツール

本節では，前述したメトリクスの測定や活用に関連したいくつかのツールについて紹介する．

表 4　メトリクス測定ツール (一部)

名称	概要	メトリクス
Eclipse Metrics Plugin [23]	統合開発環境 Eclipse [22] のプラグイン．開発中のコードを測定し，測定値を適正範囲の観点から可視化.	LOC，サイクロマティック数，フィールド数，CK メトリクス (一部) 等.
CCCC [47]	C 及び C++ ソースコード向けの測定アプリケーション．ソースファイルを測定し，測定値を適正範囲の観点から可視化.	LOC，サイクロマティック数，コメント行数，ファンイン・ファンアウト数，Henry-Kafra メトリクス等.
SourceMonitor [17]	C，C++，C#，Java，VisualBasic 等のソースコード向け測定アプリケーション．指定フォルダ内のソースファイルを測定し，測定値の一覧表を生成.	LOC，文の数，コメント密度，実行パス数，ネスト数，メソッド呼出し数等.
DependencyFinder [83]	Java のソースコード向けの測定アプリケーション (入力にはコンパイル済みのバイトコードを使用)．クラス間，メソッド間での依存関係を可視化.	メソッド呼出し，他クラスの使用といった各種の依存関係の件数等.
CCFinderX [42]	C，C++，C#，Java，VisualBasic，COBOL のソースコード向けのコードクローン [43] 検出アプリケーション．指定されたフォルダ内のソースファイルを測定し，コードクローンの位置情報と関連する測定値の一覧表を生成.	ソースファイルにおけるコードクローンの占有率，コードクローンの長さや個数等.
EPM [21]	開発プロジェクトにおけるデータの自動収集と分析を支援するツール．ソースコードだけでなく，障害検出やメール投稿といったアクティビティについてもデータ収集が可能.	LOC，障害検出数，メール投稿数等.

4.1　測定ツール

メトリクスによる測定は，単に測るだけとはいえ，そこにも作業コストが必要となる．実際のところ，原理上は人手で測定可能であっても，作業や処理が煩雑であったり，扱うデータ量が膨大であったりして，測定ツールが必要不可欠となる場合も考えられる．それゆえ，メトリクスの円滑な活用のためには，測定ツールの積極的な利用が望ましい．

既存のツールは，その “自動測定” という性質上，測定対象がソースコードに限定されるものがほとんどではあるが，プロジェクト管理を支援する機能を有したツールもいくつか存在する．ここでは既存ツールの一部を表 4 に示す†16．多くのツールでは測定可能なメトリクスがあらかじめ固定されているが，測定機能部分をプラグイン化して拡張可能性を持たせたツールやプラットフォーム (例えば MASU [51]) もある．

†16　有用な有償ツールも多数存在するが，ここではオープンソースないしフリーウェアに限定して紹介している．

4.2　データ解析ツール

測定データを有効に活用する上で，データの整理と解析は必要不可欠である．通常，収集されたデータは表計算ソフトウェアで管理される場合が多い．実際，表計算ソフトウェアではデータの統計処理やグラフ化が容易であり，現実的な解析ツールといえる．

しかしながら，複雑な統計処理や大量のデータ処理を行う場合，一般の表計算ソフトウェアでは限界がある．例えば，前述のロジスティック回帰モデルの表計算ソフトウェア上での構築は容易でない．また，1 つの表計算シートに記載できるデータ件数には上限があり，やむなく複数のシートに分割して記載される場合もある．

そういった場合は統計処理ソフトウェアの利用を推奨する．例えばオープンソースの統計処理ソフトウェア R [72] ではプログラミング感覚で統計処理やグラフ化が実行可能であり，そのためのライブラリも豊富である．R では図 8 に示す簡単なスクリプトで

```
model <- glm(FAULT ~ CC + CALL + LOC,
             data = d, family=binomial )
```

図 8 R でのロジスティック回帰モデル構築例

ロジスティック回帰モデルを構築できる†17．図 8 に示した glm 関数では一般化線形モデル (generalized linear model) [81] の構築を行うことができ，ロジスティック回帰モデルに限らずさまざまな線形・非線形モデルの構築に活用できる．前述のポアソン回帰モデルや負の二項回帰モデルも構築できる [44]．R を使った統計処理・データ処理に関して，近年では Web 上の情報 [73] も豊富であり，初心者向けの解説本も多数出版されるようになってきている [9], [50], [59]．

成長曲線モデルについても，上述した R の glm 関数である程度は対応可能である [3] が，そこでのモデル構築は最尤法ではなく最小自乗法によるものであるため，信頼性に関する確率的な議論を十分にできない [89]．より厳密には，信頼度評価専用のツール (例えば SRATS [69]) を使うべきである．

また，最近はデータマイニング技術を使ったデータ解析も盛んに行われるようになってきている．これについては例えば Weka [88] というオープンソースソフトウェアがあり，どういったメトリクスがフォールト潜在の疑わしさ (Fault-proneness) に強く影響するのかといった解析 [65] が容易に実施できる．

5 データ収集における最近の傾向：オープンソースと公開データリポジトリの利用

著者らの知る限り，近年では国内外を問わずメトリクスそのものを新たに提案するような研究は少なくなり，メトリクスに関する数理モデルやデータ解析 (関連するツール開発を含む) に関する研究が多数派となっている．数理モデルやデータ解析の研究において “実際の開発データ” は貴重な研究材料であるが，最近ではそういったデータの収集源としてオープンソースソフトウェアや公開データリポジトリが多く見られるようになってきている．以前は産学連携といったかたちで大学の研究者が企業から実データを提供してもらうという構図が一般的であったが，その傾向が変化してきている．

まず，オープンソースソフトウェアについては，大量かつ多種多様なソースコードを誰でも自由に入手・測定できるだけでなく，リポジトリを通じてその発展・改変の様子を観察したり，バグ管理システムを通じて不具合の報告・対応状況を確認したりといった解析が可能である点がデータ収集源として選ばれている主な理由である．実際，筆者らが過去約三年間 (2007 年 4 月から 2010 年 8 月まで) に国内のソフトウェア工学関連論文誌及び研究会報告†18で発表された論文を調査したところ，開発の実データを利用・解析した論文が 139 件あり，そのうちの 59 件 (42%) がオープンソースプロジェクトを活用したものであった．それらの論文では，ソースコードをプロダクトメトリクスで測定して分析するだけでなく，リポジトリや変更履歴を対象とした一種のプロセスメトリクスによる測定・分析やバグ管理システム上での不具合報告の動向分析といった研究が報告されていた．

次に，公開データリポジトリについてであるが，これは実験データや測定データを Web 上で公開・共有したものであり，著名なものとして NASA MDP [60] や PROMISE リポジトリ [74] がある．例えば，PROMISE リポジトリでは，Eclipse 等の著名なオープンソースソフトウェアについて，どのコードでいくつのフォールトが検出されていたかといったデータが公開されており，Fault-prone モジュール予測といった研究で広く活用されている．

6 おわりに

本論文では，ソフトウェアメトリクスの基礎についてふれ，その上でデータ分析で注意すべき点や代表的な数理モデル，ツールについて概説した．また，最近の傾向の 1 つとして，オープンソースプロジェクト

†17 詳細は誌面の都合上割愛するが，d というオブジェクト (表計算シートに相当) に，サイクロマティック数 (CC)，関数呼出し回数 (CALL)，LOC 及びフォールトの有無 (FAULT) があり，そこからモデル (オブジェクト名 = model) を構築している．

†18 日本ソフトウェア科学会，電子情報通信学会及び情報処理学会の論文誌，査読付ワークショップ論文及び研究会報告をさす．全部で 1157 件を調査した．

や公開データリポジトリをデータ収集源とした研究が活発に行われるようになってきたことについてもふれた．

近年ではさまざまな測定ツールや解析ツールをオープンソースといったかたちで容易に入手・利用できるようになってきており，以前に比べてソフトウェアメトリクスを容易に利用できるようになってきている．メトリクス活用の第一歩は“まず測ること”に尽きるが，本論文で述べたように目的や対象，属性の明確化も忘れてはならない．そして，その上で適切にデータを分析することが効果的なメトリクスの活用へとつながる．本論文が今後の積極的なメトリクスの導入や研究の足がかりとなれば幸いである．

なお，本論文における参考文献の選択は系統的なレビューによるものではないため，メトリクスに関する文献を必ずしも網羅するものではないことを付記しておく．

謝辞 本解説論文の執筆機会を与えてくださった権藤克彦副編集委員長 (東京工業大) 並びに本論文の初版につきまして有益なコメントをくださった査読者の皆様に感謝いたします．

参 考 文 献

[1] Albrecht, A.J.: Measuring Application Development, in *Proc. IBM Applications Development Joint SHARE/GUIDE Symp.*, 1979, pp. 83–92.
[2] Albrecht, A.J. and Gaffney, J.: Software Function, Source Lines of Code and Development Effort Prediction, *IEEE Trans. Softw. Eng.*, Vol. SE-9, No. 6 (1983), pp. 639–648.
[3] 阿萬裕久：オープンソース開発におけるコード変更量の推移予測に関する考察，電子情報通信学会技術報告，Vol. 109, No. 343 (2009), pp. 73–78.
[4] 阿萬裕久：エンピリカルソフトウェア工学のすすめ，ソフトウェアエンジニアリング最前線 2009，鵜林尚靖，岸知二 (編)，近代科学社，2009, pp. 13–20.
[5] 阿萬裕久：オープンソースソフトウェアにおけるコメント文記述とフォールト潜在率との関係に関する実証的考察，ソフトウェアエンジニアリング最前線 2010，松下誠，紫合治 (編)，近代科学社，2010, pp. 97–100.
[6] Aman, H., Mochiduki, N. and Yamada, H.: A Model for Detecting Cost-Prone Classes Based on Mahalanobis-Taguchi Method, *IEICE Trans. Inf. & Syst.*, Vol. E89-D, No. 4 (2006), pp. 1347–1358.
[7] Aman, H. and Okazaki, H.: Impact of Comment Statement on Code Stability in Open Source Development, in *Knowledge-Based Softw. Eng.*, Virvou, M. and Nakamura, T. (Eds.), IOS Press, 2008, pp. 415–419.
[8] 阿萬裕久，山下裕也：整数計画法を用いた重点レビュー対象モジュールの選択，コンピュータソフトウェア，Vol. 27, No. 4 (2010), pp. 240–245.
[9] 青木繁伸：R による統計解析，オーム社，2009.
[10] 馬場慎太郎，吉田則裕，楠本真二，井上克郎：Fault-Prone モジュール予測へのコードクローン情報の適用，電子情報通信学会論文誌 D，Vol. J91-D, No. 10 (2008), pp. 2559–2561.
[11] Basili, V.R., Briand, L.C. and Melo, W.L.: A Validation of Object-Oriented Design Metrics as Quality Indicators, *IEEE Trans. Softw. Eng.*, Vol. 22, No. 10 (1996), pp. 751–761.
[12] Basili, V.R. Caldiera, G. and Rombach, D.: Goal, Question, Metric Paradigm, in *Encyclopedia of Software Engineering*, Vol. 1, Marciniak, J.J. (ed.), John Wiley & Sons, 1994, pp. 528–532.
[13] Boehm, B.W.: *Software Engineering Economics*, Prentice Hall, 1981.
[14] Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D. and Steece, B.: *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
[15] Briand, L.C., Daly, J.W. and Wüst, J.: A Unified Framework for Cohesion Measurement in Object-Oriented Systems, *Empirical Software Eng.*, Vol. 3, No. 1 (1998), pp. 65–117.
[16] Briand, L.C., Eman, K.E., Surmann, D., Wieczorek, I. and Maxwell, K.D.: An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques, in *Proc. 21st Int'l Conf. Softw. Eng.*, 1999, pp. 313–322.
[17] Campwood Softw.: SourceMonitor Version 2.6, http://www.campwoodsw.com/sourcemonitor.html.
[18] Carnegie Mellon University: Capability Maturity Model Integration (CMMI), http://www.sei.cmu.edu/cmmi/.
[19] Chidamber, S.R. and Kemerer, C.F.: A Metrics Suite for Object Oriented Design, *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6 (1994), pp. 476–498.
[20] COSMIC: Common Software Measurement International Consortium, http://www.cosmicon.com/.
[21] ease プロジェクト: EASE ツール - EPM, http://www.empirical.jp/EASE_DVD/Tools/.
[22] The Eclipse Foundation open source community website, http://www.eclipse.org/.
[23] Eclipse Metrics Plugin, http://eclipse-metrics.sourceforge.net/.
[24] Fenton, N., Neil, M. and Marquez, D.: Using Bayesian Networks to Predict Software Defects and Reliability, in *Proc. IMechE Vol. 222 Part O: J. Risk and Reliability*, 2008, pp. 701–712.
[25] Fenton, N., Neil, M., Marsh, W., Hearty, P., Radlinski, L. and Krause P.: Project Data Incorporating Qualitative Facts for Improved Software

Defect Prediction, in *Proc. 3rd Int'l Workshop on Predictor Models in Softw. Eng.*, 2007, p. 2.
[26] Fenton, N.E. and Pfleeger, S.L.: *Software Metrics: A Rigorous and Practical Approach, Second Edition*, PWS Publishing, 1997.
[27] Furuyama. T. and Nakagawa, Y.: A Manifold Growth Model that Unifies Software Reliability Growth Models, *Int'l J. Reliability, Quality and Safety Eng.*, Vol. 1, No. 2 (1994), pp. 161–184.
[28] Goel, A.L. and Okumoto, K.: Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures, *IEEE Trans. Reliability*, Vol. R-28, No. 3 (1979), pp. 206–211.
[29] Halstead, M.H.: *Elements of Software Science*, Elservier North Holland, 1977.
[30] 浜野康裕，天嵜聡介，水野修，菊野亨：相関ルールマイニングによるソフトウェア開発プロジェクト中のリスク要因の分析，コンピュータソフトウェア，Vol. 24, No. 2 (2007), pp. 79–87.
[31] Hata, H., Mizuno, O. and Kikuno, T.: Fault-Prone Module Detection Using Large-Scale Text Features Based on Spam Filtering, *Empirical Software Eng.*, Vol. 15, No. 2 (2010), pp. 147–165.
[32] Henry, S. and Kafura, D.: Softwre Structure Metrics based on Information Flow, *IEEE Trans. Softw. Eng.*, Vol. SE-7, No. 5 (1981), pp. 510–518.
[33] 細川宣啓：続・一歩先行くメトリクス測定の始め方・生かし方，ソフトウェア・テスト PRESS，Vol. 10 (2010), pp. 57–78.
[34] IFPUG: International Function Point Users Group, http://www.ifpug.org/.
[35] ISO/IEC 14598-1:1998: Information technology – Software product evaluation – Part1: General overview (JIS X 0133-1:1999 ソフトウェア製品の評価 – 第 1 部：全体的概観).
[36] ISO/IEC 15939:2002: Software engineering – Software measurement process (JIS X 0141:2004 ソフトウェア測定プロセス).
[37] ISO 9000: Quality management systems – Fundamentals and vocabulary (JIS Q 9000:2006 品質マネジメントシステム – 基本及び用語).
[38] 岩崎学：統計的データ解析入門ノンパラメトリック法，東京図書，2006.
[39] 岩崎学：カウントデータの統計解析，朝倉書店，2010.
[40] 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 (編)：ソフトウェア開発データ白書 2009，日経 BP，2009.
[41] 柿元健，門田暁人，亀井靖高，まつ本真佑，松本健一，楠本真二：Fault-prone モジュール判別におけるテスト工数割当てとソフトウェア信頼性のモデル化，情報処理学会論文誌，Vol. 50, No. 7 (2009), pp. 1716–1724.
[42] Kamiya, T.: the archive of CCFinder Official Site, http://www.ccfinder.net/.
[43] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7 (2002) pp. 654–670.
[44] 金明哲：R とカテゴリカルデータのモデリング (2), *ESTRELA*, No. 160 (2007), pp. 58–63.
[45] 木浦幹雄，まつ本真佑，亀井靖高，門田暁人，松本健一：異なるプロジェクト間における Fault-Prone モジュール判別の評価，ソフトウェア工学の基礎 XIV，岸知二，野田夏子 (編)，近代科学社，2007, pp. 131–136.
[46] Li, W. and Henry, S.: Object-Oriented Metrics That Predict Maintainability, *J. Systems and Software*, Vol. 23, No. 2 (1993), pp. 111–122.
[47] Littlefair, T.: CCCC - C and C++ Code Counter, http://cccc.sourceforge.net/.
[48] Lorenz, M. and Kidd, J.: *Object-Oriented Software Metrics*, PTR Prentice Hall, 1994.
[49] 町田欣史：ソースコード品質向上のための全社的な取り組み，プロジェクトマネジメント学会 2007 年度秋季研究発表大会予稿集, 2007，pp. 174–176.
[50] 間瀬茂：R プログラミングマニュアル，数理工学社，2007.
[51] MASU, http://sourceforge.net/projects/masu/.
[52] 松原望：ベイズ統計学概説，培風館，2010.
[53] 松本健一 (訳): メトリクス，ソフトウェア工学大事典，片山卓也，土居範久，鳥居宏次 (監訳)，朝倉書店，1998, pp. 1420–1430.
[54] McCabe, T.: A Software Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol. SE-2, No. 4 (1986), pp. 308–320.
[55] 望月尚美，阿萬裕久，山田宏之：クラスサイズメトリクスを用いたソフトウェア変更量の予測判別に関する考察，ソフトウェア工学の基礎 XI，野呂昌満，山本晋一郎 (編)，近代科学社，2004, pp. 93–96.
[56] 門田暁人，小林健一：線形重回帰モデルを用いたソフトウェア開発工数予測における対数変換の効果，コンピュータソフトウェア，Vol. 27, No. 4 (2010), pp. 234–239.
[57] 門田暁人，佐藤慎一，神谷年洋，松本健一：コードクローンに基づくレガシーソフトウェアの品質分析，情報処理学会論文誌，Vol. 44, No. 8 (2003), pp. 2178–2188.
[58] Munson, J.C. and Khoshgoftaar, T.M.: The Detection of Fault-Prone Programs, *IEEE Trans. Softw. Eng.*, Vol. 18, No. 5 (1992), pp. 423–433.
[59] 長畑秀和：R で学ぶ統計学，共立出版，2009.
[60] NASA IV&V Facility Metrics Data Program, http://mdp.ivv.nasa.gov/.
[61] NESMA: Netherlands Software Metrics Association, http://www.nesma.nl/.
[62] 野中誠：ソフトウェア開発コスト予測研究の動向と課題，ソフトウェアエンジニアリング最前線 2008，飯田元，山本里枝子 (編)，近代科学社，2008, pp. 33–40.
[63] 野中誠：ソフトウェアインスペクションの効果と効率，情報処理，Vol. 50, No. 5 (2009), pp. 385–390.
[64] 野中誠：ソフトウェア品質の定量的管理における曖昧さ—ソフトウェア欠陥測定の原則—，東洋大学経営論集，No. 76 (2010), pp. 99—109.
[65] 野中誠，水野修：fault-prone モジュール予測技法の基礎と研究動向，ソフトウェアエンジニアリングシンポジウム 2010 チュートリアル資料，情報処理学会シンポジウムシリーズ，Vol. 2010, No. 2, 2010.
[66] 野中誠，鷲崎弘宜 (訳)：演習で学ぶソフトウェアメトリクスの基礎，日経 BP 社，2009.

[67] Ohba, M.: Software Reliability Analysis Models, *IBM J. Res. Develop.*, Vol. 28, No. 4 (1984), pp. 428–443.
[68] 岡村寛之，安藤光昭，土肥正：一般化ガンマソフトウェア信頼性モデル，電子情報通信学会論文誌 D-I, Vol. J87-D-I, No. 8 (2004), pp. 805–814.
[69] 岡村寛之，安藤光昭，土肥正：表計算ソフトウェアによるソフトウェア信頼性評価ツール (SRATS) の開発，電子情報通信学会論文誌 D-I，Vol. J88-D-I, No. 2 (2005), pp. 205–214.
[70] 岡村寛之，古村仁志，土肥正：傾向局線に基づいたソフトウェア信頼性モデルに対するパラメータ推定，情報処理学会論文誌，Vol. 47, No. 3 (2006), pp. 897–905.
[71] Pai, G.J. and Dugan, J.B.: Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods, *IEEE Trans. Softw. Eng.*, Vol. 33, No. 10 (2007), pp. 675–686.
[72] R Foundation: The R Foundation for Statistical Computing, http://www.r-project.org/.
[73] RjpWiki, http://www.okada.jp.org/RWiki/.
[74] Sayyad, S.J. and Menzies, T.J.: PROMISE Software Engineering Repository, http://promise.site.uottawa.ca/SERepository/.
[75] Shepperd, M. and Schofield, C.: Estimating Software Project Effort Using Analogies, *IEEE Trans. Softw. Eng.*, Vol. 23, No. 11 (1997), pp. 736–743.
[76] 繁桝算男，植野真臣，本村陽一：ベイジアンネットワーク概説，培風館，2006.
[77] Software Productivity Research, http://www.spr.com/.
[78] SQuBOK 策定部会 (編)：ソフトウェア品質知識体系ガイド—SQuBOK Guide—，オーム社，2007.
[79] Taguchi, G., Chwdhury, S. and Wu, Y.: *The Mahalanobis-Taguchi System*, McGraw-Hill, 2000.
[80] 田村晃一，柿元健，戸田航史，角田雅照，門田暁人，松本健一，大杉直樹：工数予測における類似性に基づく欠陥値補完法の実験的評価，コンピュータソフトウェア，Vol. 26, No. 3 (2009), pp. 44–55.
[81] 田中豊，森川敏彦，山中竹春，冨田誠 (訳)：一般化線形モデル入門，共立出版，2008.
[82] 丹後俊郎，山岡和枝，高木晴良：ロジスティック回帰分析，朝倉書店，1996.
[83] Tessier, J.: Dependency Finder, http://depfind.sourceforge.net/.
[84] 富野壽，荒木貞雄 (監訳)：ソフトウェアの規模決定，見積り，リスク管理，共立出版，2008.
[85] 角田雅照，大杉直樹，門田暁人，松本健一，佐藤慎一：協調フィルタリングを用いたソフトウェア開発工数予測方法，情報処理学会論文誌，Vol. 46, No. 5 (2005), pp. 1155–1164.
[86] 鷲尾隆，元田浩：尺度の理論，日本ファジィ学会誌，Vol. 10, No. 3 (1998), pp. 401–413.
[87] 鷲崎弘宜，波木理恵子，福岡呂之，原田陽子，渡辺博之：プログラムソースコードのための実用的な品質評価枠組み，情報処理学会論文誌，Vol. 48, No. 8 (2007), pp. 2637–2650.
[88] Weka 3: Data Mining Software in Java, http://www.cs.waikato.ac.nz/ml/weka/.
[89] Wood, A.: Predicting Software Reliability, *IEEE Computer*, Vol. 29, No. 11 (1996), pp. 69–77.
[90] 矢口竜太郎，吉田洋平：成功率は 31.1%，日経コンピュータ，12 月 1 日号 (2008)，pp. 36–53.
[91] 山田茂：ソフトウェア信頼性モデル，日科技連，1994.
[92] 山田茂，木村光宏，高橋宗雄：TQM のための統計的品質管理，コロナ社，1998.
[93] 山田茂，高橋宗雄：ソフトウェアマネジメントモデル入門，共立出版，1993.
[94] 山田茂，得能貢一，井上圭：コスト評価基準に基づくソフトウェア信頼性/安全性を考慮した最適リリース問題，電子情報通信学会論文誌 A，Vol. J82-A, No. 1 (1999), pp. 64–72.