修士学位論文

題目

ソースコードの変更履歴におけるメトリクス値の変化を用いたソフトウェアの特性分析

指導教員
井上 克郎 教授

報告者
村尾 憲治

平成 20 年 2 月 8 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

平成 19 年度 修士学位論文

ソースコードの変更履歴におけるメトリクス値の変化を用いたソフトウェアの特性分析

村尾 憲治

内容梗概

ソフトウェアに関する深い知識は，その開発や保守を行うにあたり有益である．開発者がソフトウェアに関する知識を得るには，対象ソフトウェアについての調査が必要となるが，これを手動で行う場合，対象とするソフトウェアの規模に応じて膨大な時間と労力を費やすことになる．そのため，開発者に負担を掛けずにソフトウェアに関する知識を提供することが求められる．

本研究では，ソースコードにおけるソフトウェアメトリクスの値の変遷に基づき，ソフトウェアの特性を分析する手法の提案を行う．提案する手法は，バージョン管理システムのリポジトリに蓄積されたソフトウェアの開発履歴に関する情報を用いてメトリクス値の変遷を分析することにより，注目すべきモジュールやメトリクス，変更など，ソフトウェアの様々な特性に関する情報を得ることができるというものである．本手法によって得られる情報は，効率的なソフトウェアの開発や保守に役立つと期待される．実際に本手法を実装したシステムを複数のオープンソースソフトウェアに対して適用し，将来問題の発生しやすいモジュールや，開発過程において特に注目すべき変更など，有益なソフトウェアの特性が得られることを確認した．

主な用語

バージョン管理システム
ソフトウェアメトリクス

目 次

1　まえがき

ソフトウェアに関する深い知識は，その開発や保守を行うにあたり有益である．例えば，問題が発生しやすいモジュールを知っていればそのモジュールに労力を注ぐことで効率的な開発や保守が行え，またソフトウェアがどういった過程を経て開発されてきたかを知っていれば今後の開発の見通しも立てやすい．開発者がこのような知識を得るには，対象ソフトウェアについての調査が必要となるが，これを手動で行う場合，対象とするソフトウェアの規模に応じて膨大な時間と労力を費やすことになる．そこで，開発者にソフトウェアに関する知識を提供するため，数多くの研究がなされている．

ソフトウェアの開発履歴を基に知識を提供する研究が既に行われている．多くのものはCVS[1][2][3] や Subversion[4] などのバージョン管理システムを情報源としている．バージョン管理システムは本来ソフトウェアの開発を効率よく管理するために用いられる．管理されるソフトウェアの開発履歴に関する情報は，バージョン管理システムのリポジトリというアーカイブに蓄積されている．このリポジトリに蓄積された開発履歴を閲覧することによって，ソフトウェアの開発過程についてより深い理解が得られることが知られている [5].

一方，ソフトウェアのソースコードそのものを基に知識を提供する研究も行われている．これに関しては，McCabe のサイクロマチック数 [6] や CK メトリクス [7] に代表されるソフトウェアメトリクスが有名である．特に，CK メトリクスは多くの研究において問題の発生しやすいモジュールを特定するのに有効であると示されている [8][9].

本研究ではバージョン管理システムとソフトウェアメトリクスに基づいてソフトウェアの特性を分析する手法を提案する．具体的には，バージョン管理システムに蓄積された開発履歴情報を用いてソフトウェアメトリクスの値の変遷を求めることにより，以下に示す 6 つの変動度というソフトウェアメトリクスの値が変動する激しさの指標を導出する．この変動度という指標を用いて，ソフトウェアの傾向や性質を分析する．この手法により様々な観点からのソフトウェアの傾向や性質を知ることができる．

- モジュールの変動度
- メトリクスの変動度
- 変更の変動度
- （モジュール，メトリクス）の変動度
- （モジュール，変更）の変動度
- （モジュール，メトリクス，変更）の変動度

また，提案手法を実装したシステムを作成し，複数のオープンソースソフトウェアに対して適用実験を行った．その結果，各変動度を用いて分析を行うことにより，後に問題の発生しやすいモジュールや，開発過程において特に注目すべき変更など，ソフトウェアの開発や保守を行うにあたり有益である様々なソフトウェアの傾向や性質が取得できた．

以降，2 節ではバージョン管理システムとソフトウェアメトリクスについて説明し，3 節では本稿における用語と変動度の計測手段について述べる．4 節では提案手法の説明を行い，5 節では提案手法を実装したシステムについて述べる．6 節では提案手法を実装したシステムを用いて行った適用事例について，その結果と考察を述べる．7 節では関連する研究について述べ，最後にまとめと今後の課題を 8 節に記す．

2　バージョン管理システムとソフトウェアメトリクス

本節では，本研究の根幹を成すバージョン管理システムとソフトウェアメトリクスについて説明する．

2.1　バージョン管理システム

本節では，まずバージョン管理システムの概要について説明し，次にバージョン管理システムをいくつか紹介する．

2.1.1　バージョン管理システムの概要

現在，非常に多くのソフトウェアの開発や保守においてバージョン管理システム [5][10] が利用されている．バージョン管理システムとは，主として以下の 3 つの機能を提供するものである．

- プロダクトに対して施された追加・削除・変更などの作業を履歴として蓄積する．
- 蓄積した履歴を開発者に提供する．
- 蓄積したデータを編集する．

バージョン管理システムの仕組みを説明するにあたり，その基礎となるモデルが数多く存在する [11][12]．以降では，多くのバージョン管理システムが採用している Checkout/Checkin モデルについて概要を述べる．

まず，Checkout/Checkin モデルにおける用語を説明する．次に挙げる用語は，本節以降でも用いる．

- リビジョン
  各プロダクト（ソースコード，ドキュメントなど）のある時点における状態．1 つのリビジョンには，その時点でのソースコードやリソースなどの実データと，リビジョン作成時刻や作成時のメッセージログなどの属性データが格納されている．
- リポジトリ
  1 つのプロジェクトの全てのリビジョンを格納する，データ格納庫．
- チェックアウト
  リポジトリから特定のリビジョンのプロダクトを開発者の手元にコピーする操作．

- チェックイン
  プロダクトに対して行なった変更内容をリポジトリに送り，新たなリビジョンを作成する操作.

- リビジョン列
  連続するリビジョンの系列.

- トランク
  リビジョン列の内，主となる一直線状のリビジョン列．単純にリビジョンを作成するだけならば，リビジョンはトランク上に生成される．

- ブランチ
  リビジョン列の内，トランクから分岐したリビジョン列．新しいリビジョンをトランクから分岐させるのは，デバッグを行う場合など，主となる作業工程とは別に開発を行う場合である．

- マージ
  ブランチ上で行われた変更内容を他のリビジョン列におけるリビジョンに反映させる操作.

Checkout/Checkin モデルを図 1 に示す．Checkout/Checkin モデルは，ファイルを単位としたリビジョンの制御に関して定義されている．リビジョン管理下にあるプロダクトは，それぞれのバージョン管理システムに依存したフォーマット形式のファイルとしてリポジトリに格納されている．開発者はそれらのファイルを直接操作するのでなく，各バージョン管理システムに実装されているコマンドを介してリポジトリとのデータ授受を行う．図 1 に示されるように，Checkout/Checkin モデルでは，各開発者が以下の 3 つの作業を繰り返すことによってソフトウェアの開発を行う．

1. リポジトリから必要なプロダクトをチェックアウトする．
2. チェックアウトしたプロダクトに対し，開発者の手元で変更を加える．
3. 変更を行ったプロダクトをリポジトリへチェックインする．この作業により，リポジトリ内に新たなリビジョンが生成される．

2.1.2 バージョン管理システムの紹介

バージョン管理システムと呼ばれるものは多数存在する．

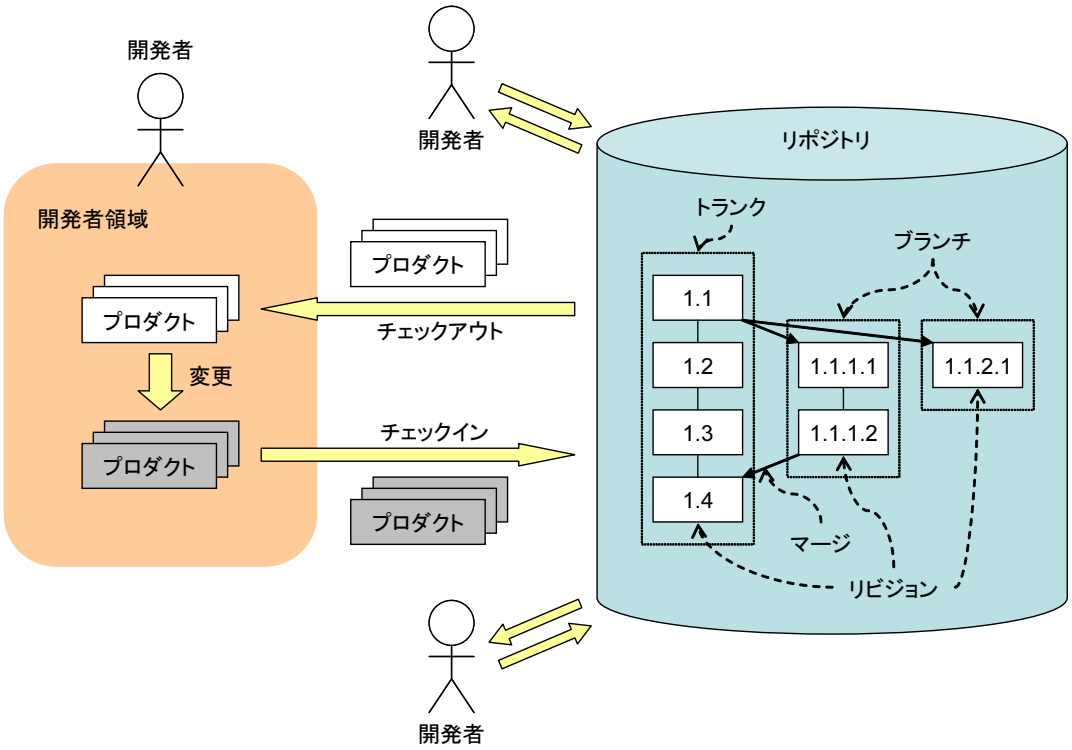


図 1: Checkout/Checkin モデル

UNIX 系 OS では，多くの場合 RCS[13] や CVS[1][2][3] といったシステムが標準で利用可能となっている．ClearCase[14] のように商用ものも存在する．また，UNIX 系 OS だけではなく，Windows 系 OS においても，SourceSafe[15] や PVCS[16] をはじめ，数多く存在する．さらに，ローカルネットワーク内のみではなく，よりグローバルなネットワークを介したシステムも存在する [17].

ここでは，それらバージョン管理システムの内，主だったものをいくつかを紹介する．

RCS

RCS は UNIX 上で動作するツールとして作成された版管理システムであり，現在でもよく使用されているシステムである．単体で使用される他，システム内部に組み込み，バージョン管理機構を持たせる場合などの用途もある．RCS ではプロダクトをそれぞれ UNIX 上のファイルとして扱い，1 ファイルに対する記録は 1 つのファイルに行われる．

RCS におけるリビジョンは，管理対象となるファイルの中身がそれ自身によって定義され，リビジョン間の差分は diff コマンドの出力として定義される．各リビジョンに対する識別子は数字の組で表記され，数え上げ可能な識別子である．新規リビジョンの登録や，任意のリビジョンの取り出しは，RCS の持つツールを利用する．

CVS

CVS は，RCS と同様に UNIX 上で動作するシステムとして構築されたバージョン管理システムであり，近年最も良く使われるシステムの１つである．

RCS と大きく異なるのは，複数のファイルを処理する点である．また，リポジトリを複数の開発者で利用することも考慮し，開発者間の競合にも対処可能となっている．さらに，ネットワーク環境（ssh，rsh 等）を利用することも可能である為，オープンソースによるソフトウェア開発に用いられることが多い．その最たる例が，FreeBSD[18] や OpenBSD[19] などのオペレーティングシステムの開発である．

Subversion

Subversion[4] は，CVS の持ついくつかの問題点を解決するために開発されたバージョン管理システムで，CVS の後継として注目されている．実際に，ソフトウェア開発に用いるバージョン管理システムを CVS から Subversion に移行するケースも現れている．

CVS はプロダクトをファイル単位で管理していたが，Subversion は全てのプロダクトを一つのデータベースで管理する．Subversion では，CVS では考慮していなかったファイルの移動の処理を扱うことができたり，リポジトリ内のディレクトリを削除したりすることが可能になっている．また，ネットワークを介した利用を想定して開発

されており，リポジトリへチェックインを行なう際は，新しいリビジョンを作成するのに必要な差分の情報だけが送られるため，効率が良い．

2.2 ソフトウェアメトリクス

本節では，まずソフトウェアメトリクスの概要について説明し，次にソフトウェアメトリクスをいくつか紹介する．

2.2.1 ソフトウェアメトリクスの概要

ソフトウェアメトリクスは，ソフトウェアの様々な特性（複雑度，信頼性，効率など）を判別する客観的な数学的尺度であり，ソフトウェアを統計的な視点から見ることを可能にする [20]．例えば，ソフトウェア開発の設計や実装の段階において，設計書やソースコードからソフトウェアの複雑さを測定し，それによって後のエラーの発生を予測するための複雑度メトリクスがよく用いられている [21]．ソフトウェアメトリクスには定性的なものと定量的なものがあるが，本研究において言及するのは，例として挙げたようなソースコードを対象とする定量的なソフトウェアメトリクスのみとする．

今までに提唱されたソースコードを対象とするソフトウェアメトリクスとしては，Halsted のメトリクス [22]，McCabe のサイクロマチック数 [6]，Chidamber と Kemerer が提案した CK メトリクス [7] などが挙げられる．

2.2.2 ソフトウェアメトリクスの紹介

ここではソースコードを対象としたソフトウェアメトリクスをいくつか紹介する．以降，ソフトウェアメトリクスを単にメトリクスを記載する．

ソースコードを対象としたメトリクスの中で，最も定義と計算が簡単であり，長い歴史を持つものの 1 つとしてソースコードの行数を示す LOC が挙げられる．LOC の定義は次のようになる．

LOC （ソースコードの行数，lines of code）：
LOC は計測対象のソースコードの行数である．
計測対象のソースコード C の行数を l とする．このとき $LOC(C) = l$ である．

Chidamber と Kemerer が提案した CK メトリクス [7] は，オブジェクト指向ソフトウェアにおけるクラスを対象とするメトリクスとして有名である．CK メトリクスは広く利用されているため，複数の研究者による様々な実験に用いられている [23][24]．また，CK メトリクスは他のメトリクスに比べ，よりエラーの予測に有効であるとの報告がなされている [8][9].

CK メトリクスは，WMC，DIT，NOC，CBO，RFC，LCOM の 6 種類のメトリクスから成る．これらのメトリクスは，いずれも値が大きいほど複雑であることを意味している．以下にそれぞれのメトリクスの説明を記載する．

RFC （クラスへのレスポンス，response for a class）：

RFC は計測対象クラスのメソッドと，それらのメソッドから呼び出されるメソッドの数の総和である．

計測対象クラス C が n 個のメソッド $M_1, M_2, \ldots, M_n$ を持つとする．また，メソッド $M_1, M_2 \ldots, M_n$ は C 以外で定義されているメソッドをそれぞれ $m_1, m_2, \ldots, m_n$ 個だけ呼び出すとする．このとき，$RFC(C) = n + \sum_{i=1}^{n} m_i$ である．

CBO （クラス間の結合，coupling between object classes）：

あるクラス C_a が，異なるクラス C_b のメソッドやフィールドを使用するとき，C_a は C_b と結合しているという．

CBO は計測対象クラスと結合しているクラスの数である．

計測対象クラス C が結合しているクラスの数を cc とすると，$CBO(C) = cc$ である．

LCOM （メソッドの凝集の欠如，lack of cohesion in methods）：

メソッドの凝集度とは，クラスのメソッドがお互いに関連している程度を指す．全てのメソッドが同一のインスタンス変数にアクセスする場合，高い凝集度を持つ．メソッドがアクセスするインスタンス変数の集合が互いに素ならば，凝集度は低くなる．

LCOM はメソッドの凝集度の低さを表すメトリクスである．

計測対象クラス C が n 個のメソッド $M_1, M_2, \ldots, M_n$ を持つとする．$I_1, I_2, \ldots, I_n$ をそれぞれメソッド M_i によって用いられるインスタンス変数の集合とする．$P = \{(I_i, I_j) | I_i \cap I_j = \phi\}$，$Q = \{(I_i, I_j) | I_i \cap I_j \neq \phi\}$ と定義する．ただし，$I_1, I_2, \ldots, I_n$ が全て ϕ の時は，$P = \phi$ とする．このとき，$LCOM(C) = |P| - |Q|$ である．ただし，$LCOM(C) < 0$ となる場合は $LCOM(C) = 0$ とする．

NOC （子クラスの数，number of children）：

NOC は計測対象クラスを直接継承しているクラスの数である．

計測対象クラス C を直接継承している子クラスの数を sc とする．このとき $NOC(C) = sc$ である．

DIT （継承木における深さ，depth of inheritance tree）：

DIT は計測対象クラスの継承木内での深さである．多重継承が許される場合は，DIT は継承木におけるそのクラスを表す節点から，それ以上親クラスが存在しないクラス，すなわち根に至る最長パスの長さとなる．

例えば，開発言語が Java であるソフトウェアを対象とする場合，全てのクラスは java.lang.Object を最上位の親クラスとして持つので，java.lang.Object をクラス C_0 とおくと，$DIT(C_0) = 1$ と定義する．C_0 を直接継承したクラスを C_1，C_1 を直接継承したクラスを C_2 とおくと，それぞれ $DIT(C_1) = 2$，$DIT(C_2) = 3$ となる．

WMC （クラスの重み付きメソッド数，weighted methods per class）：

WMC は計測対象クラスにおける各メソッドの複雑度の総和である．

計測対象クラス C が n 個のメソッド $M_1, M_2, \ldots, M_n$ を持つとする．これらのメソッドの複雑度をそれぞれ $c_1, c_2, \ldots, c_n$ とする．このとき，$WMC(C) = \sum_{i=1}^{n} c_i$ である．

メソッドの複雑度の具体的な算出方法は定められていないが，Halsted のメトリクス [22] や McCabe のサイクロマチック数 [6] などを用いる方法が考えられる．

3　準備

本節では，本研究における用語や手法で用いる変動度の計測手段を定義する．

3.1　用語

本節では本研究における以下の用語の意味を説明する．

- 変更履歴情報
- スナップショット
- 変動度
- ソフトウェアの特性

3.1.1　変更履歴情報

変更履歴情報とは，現在のソースコードに至るまでの全ての変更についての情報である．
本稿において「ソースコードの変更履歴情報」あるいは単に「変更履歴情報」と記載した場合，特に断りのない限り，全てのリビジョンおける以下の情報の集合を意味する．

- リビジョン番号
- 変更を行った開発者名
- 変更の日時
- 変更を行った開発者によるメッセージログ
- 変更を行う 1 つ前のリビジョン番号
- 変更を行う 1 つ前のリビジョンからの変更内容

変更履歴情報はバージョン管理システムのリポジトリに蓄積された開発履歴から取得することができる．

3.1.2　スナップショット

スナップショットとは，ある時点におけるソフトウェア全体の状態である．
ソフトウェアに対して何らかの変更が加わると，その時点で今までのスナップショットとは異なる新しいスナップショットが生まれる．

本稿において「ソースコードのスナップショット」あるいは単に「スナップショット」と記載した場合，特に断りのない限り，ある時点におけるソフトウェアの状態でのソースコード全てを意味する．

スナップショットはバージョン管理システムから得ることができる．しかし，ブランチを持つソースコードが存在する場合，ある時点におけるそのソースコードの状態をトランク上のものとするかブランチ上のものとするか，あるいは両方とするかで複数種類のスナップショットが考えられる．本研究においては，スナップショットにはブランチを含めない．スナップショットはトランク上のソースコードのみを対象とする．

3.1.3 変動度

変動度とは，変動の激しさを表す指標である．

本稿において「メトリクス値の変動度」あるいは単に「変動度」と記載した場合，特に断りのない限り，ソースコードのメトリクス値の変動の激しさを意味する．

変動度の計測手段など, 詳しくは 3.2 節で述べる．

3.1.4 ソフトウェアの特性

ソフトウェアの特性とは，対象のソフトウェアに存在する何らかの傾向や性質である．

広い意味の言葉であるため，様々なものがソフトウェアの特性として挙げられる．例えば，ソースコードの規模や開発日数，開発者の人数，用いるプログラミング言語，コーディング規約や開発者の士気の高さなどもその言葉の意味に含めることができる．もちろん，最新の状態のソースコードのメトリクス値もソフトウェアの特性として挙げることができる．

その中でも本研究が注目するソフトウェアの特性は，ソフトウェアの開発や保守の過程において発生した傾向や性質である．したがって，本稿において「ソフトウェアの特性」と記載した場合，特に断りのない限り，ソフトウェアの進化において生み出された傾向や性質を意味する．

本研究では，以下のソフトウェアの特性を分析できる．

- 対象ソフトウェアにおいてメトリクス値の安定しないモジュール
- 対象ソフトウェアにおいて値の安定しないメトリクス
- それぞれの変更がソフトウェアに与えた影響の程度
- それぞれの開発者がソフトウェアに与えた影響の程度

これらの特性を知ることは，ソフトウェア理解や開発管理，開発者の評価など幅広い分野で役立つと期待される [25][26].

3.2 ばらつき，不確かさ，変化の指標

対象データのばらつきを表す指標として散布度，不確かさを表す指標としてエントロピー，変化を表す指標として距離などが挙げられる．これらは主として数学や統計学で用いられる指標である．

本研究では，これらのばらつきや不確かさ，変化の指標を変動度の計測手段として用いる．以降，これらの指標を用いた変動度の計測手段について説明する．

説明に当たっては以下の設定を用いる．

- 対象のモジュールの集合を $MD = \{md_1, md_2, \cdots, md_x\}$ とする．ただし，x は総モジュール数である．
- 用いるメトリクスの集合を $MT = \{mt_1, mt_2, \cdots, mt_y\}$ とする．ただし，y は用いるメトリクスの数である．
- 対象のソフトウェアにおける変更時刻の集合を $CT = \{ct_1, ct_2, \cdots, ct_z\}$ とする．ただし，z は変更時刻の総数であり，添字の値が小さいほど古い変更の時刻を表すものとする．すなわち ct_1 を最初の変更時刻，ct_2 を 2 回目の変更時刻とし，ct_z を z 回目の変更時刻，つまり最新の変更の時刻とする．
- 変更時刻 ct_k におけるモジュール md_i のメトリクス mt_j の値を v_{md_i,mt_j,ct_k} とする．ただし，変更時刻 ct_k においてモジュール md_i が存在しない場合，メトリクス値は空であるとし，$v_{md_i,mt_j,ct_k} = null$ とする．

3.2.1 変動度の計測手段としての応用

基本的な変動度の計測手段としては**エントロピー**，**正規化エントロピー**，**四分位偏差**，**四分位分散係数**，**ハミング距離**，**ユークリッド距離**，**マハラノビス距離**，**メトリクス値の変化量**の 8 つを用いる．以下，これら 8 つの変動度の計測方法について説明する．

エントロピー

Shannon の提唱した情報理論 [27] に依れば，エントロピーとは不確かさを表す指標である．メトリクス値の変動の激しさをメトリクス値の不確かさと捉えることによって，エントロピーを変動度の指標として用いることができる．

エントロピー H は対象モジュールの各メトリクスに対して導出される．ただし，モジュール md_i がメトリクス mt_j に関して取るメトリクス値 $v_{md_i,mt_j,ct_1}, v_{md_i,mt_j,ct_2}, \cdots, v_{md_i,mt_j,ct_z}$ が全て $null$ 値である場合，エントロピー H_{md_i,mt_j} は定義されない．それ以外の場合，メトリクス値 $v_{md_i,mt_j,ct_1}, v_{md_i,mt_j,ct_2}, \cdots, v_{md_i,mt_j,ct_z}$ において値が

$null$ でないものの数を $z'(1 \leq z' \leq z)$ とし，z' 個のそれぞれの値を $v'_1, v'_2, \cdots, v'_{z'}$ と表すことにする．さらにその z' 個の値のうち，異なる値の種類数を $z''(1 \leq z'' \leq z')$ とする．z'' 種類の値をそれぞれ $v''_1, v''_2, \cdots, v''_{z''}$ とおくと，エントロピー H_{md_i,mt_j} の計算式は以下のようになる．

$$H_{md_i,mt_j} = -\sum_{l=1}^{z''} p_l \log_2 p_l$$

ここで p_l は，値 v'_l の出現頻度を表し，次の式で定義される．

$$p_l = \frac{\sum_{k=1}^{z'} \text{equal}(v''_l, v'_k)}{z''} \quad (1 \leq l \leq z'')$$

$$\text{equal}(v''_l, v'_k) = \begin{cases} 1 & (v''_l = v'_k \text{のとき}) \\ 0 & (v''_l \neq v'_k \text{のとき}) \end{cases}$$

メトリクス値 $v_{md_i,mt_j,ct_1}, v_{md_i,mt_j,ct_2}, \cdots, v_{md_i,mt_j,ct_z}$ が $null$ を除き全て異なる値を持つ，すなわち $z'' = z'$ のときエントロピーは $H_{md_i,mt_j} = -\log_2 \frac{1}{z'}$ となり最大である．逆にメトリクス値 $v_{md_i,mt_j,ct_1}, v_{md_i,mt_j,ct_2}, \cdots, v_{md_i,mt_j,ct_z}$ が $null$ を除いて全て同じ，すなわち $z'' = 1$ のときエントロピーは $H_{md_i,mt_j} = -\log_2 1 = 0$ となり最小である．

正規化エントロピー

エントロピーの項目で述べたように，エントロピーの最大値は $H_{md_i,mt_j} = -\log_2 \frac{1}{z'}$ であり，変更時刻の総数 z から $null$ 値を持つ変更を除いた数 z' に依存するため，異なるソフトウェア間では正しく値を比較することが難しい．この問題を解決するため，新たに正規化エントロピー H' を定義する．

正規化エントロピー H' はエントロピー H を用いて導出されるため，エントロピー H と同様に対象モジュールの各メトリクスに対して値が得られる．上記のエントロピー H で用いた設定に準じると，正規化エントロピー H'_{md_i,mt_j} の計算式は以下のようになる．

$$H'_{md_i,mt_j} = \frac{H_{md_i,mt_j}}{-\log_2 \frac{1}{z'}}$$

エントロピー H_{md_i,mt_j} をその最大値で除算するため，正規化エントロピーは必ず $0 \leq H'_{md_i,mt_j} \leq 1$ となる．

四分位偏差

統計分野で用いられる散布度は，データのばらつきの程度を表現するものである [28]．メトリクス値の変動の激しさをメトリクス値のばらつきの程度と捉えることで，

変動度の指標として散布度を用いることができる．一般にメトリクス値が正規分布に従うとは言えないので，散布度としては四分位偏差を用いる．

メトリクス値 $v_{md_i,mt_j,ct_1}, v_{md_i,mt_j,ct_2}, \cdots, v_{md_i,mt_j,ct_z}$ から $null$ 値を除いたものを昇順に並び替えた値の並びに対し，その値の並びを $1:3$ に分割する点の値を第 1 四分位数 q_1，$3:1$ に分割する点の値を第 3 四分位数 q_3 とすると，四分位偏差 Q_{md_i,mt_j} の計算式は以下のようになる．

$$Q_{md_i,mt_j} = \frac{q_3 - q_1}{2}$$

四分位分散係数

四分位偏差 Q の値は絶対値であり，用いるメトリクスのスケール差に影響されてしまう．このため，異なるメトリクス間では正しく値を比較することが難しい．そこでこの問題への対処として四分位分散係数 Q' を用いる．四分位分散係数も四分位偏差と同様，統計分野で用いられる散布度の一種である [28].

四分位分散係数 Q' は四分位偏差 Q を用いて導出されるため，四分位偏差 Q と同様に対象モジュールの各メトリクスに対して値が得られる．メトリクス値 $v_{md_i,mt_j,ct_1}, v_{md_i,mt_j,ct_2}, \cdots, v_{md_i,mt_j,ct_z}$ から $null$ 値を除いたものを昇順あるいは降順に並び替えたものを $1:1$ に分割する点の値を中位数 m とすると，四分位分散係数 Q'_{md_i,mt_j} の計算式は以下のようになる．

$$Q'_{md_i,mt_j} = \frac{Q_{md_i,mt_j}}{m}$$

ただし $m = 0$ の場合は，$Q_{md_i,mt_j} = 0$ ならば $Q'_{md_i,mt_j} = 0$，$Q_{md_i,mt_j} \neq 0$ ならば $Q'_{md_i,mt_j} = \infty$ とする．

ハミング距離

エントロピーや四分位偏差などは，それぞれのモジュールの各メトリクスに対して変動度が得られるため，ある変更間の変動度を知るのには利用できない．そこで，変更間の変動度を知るために距離を用いる．まずはハミング距離について述べる．

情報理論 [27] においてハミング距離は，等しい文字数を持つ 2 つの文字列の中で対応する位置にある異なった文字の個数を意味する．文字列をメトリクス値の列に置き換えることで，ハミング距離を 2 つの変更間における変動の激しさを示す指標として用いることができる．

ハミング距離を用いた変動度は連続する 2 つの変更間について導出される．このため，最初の変更時刻 ct_1 についてはハミング距離が導出できないので，$null$ とする．最初の変更以外の時刻 $ct_k (1 < k \leq z)$ におけるモジュール md_i のハミング距離 DH_{md_i,ct_k}

は以下の計算式で表される．

$$DH_{md_i,ct_k} = \sum_{j=1}^{y} \text{different}(v_{md_i,mt_j,ct_{k-1}}, v_{md_i,mt_j,ct_k})$$

$$\text{different}(v_{md_i,mt_j,ct_{k-1}}, v_{md_i,mt_j,ct_k}) = \begin{cases} 1 & (v_{md_i,mt_j,ct_{k-1}} \neq v_{md_i,mt_j,ct_k} \text{のとき}) \\ 0 & (v_{md_i,mt_j,ct_{k-1}} = v_{md_i,mt_j,ct_k} \text{のとき}) \end{cases}$$

この計算式が示すように，ハミング距離 DH_{md_i,ct_k} は変更時刻 ct_{k-1} と ct_k 間において値が変化したメトリクスの数を表している．したがって，$1 < k \leq z$ となる k の内，$(v_{md_i,mt_j,ct_{k-1}} = null) \vee (v_{md_i,mt_j,ct_k} = null)$ となる k についてはハミング距離が計測できないので $DH_{md_i,ct_k} = null$ とする．まとめると，上記の計算式が成り立つの条件は $\forall k((1 < k \leq z) \wedge (v_{md_i,mt_j,ct_{k-1}} \neq null) \wedge (v_{md_i,mt_j,ct_k} \neq null))$ となる．この条件が成り立たない場合は，$DH_{md_i,ct_k} = null$ とする．

ユークリッド距離

上記のハミング距離 DH は変更前と変更後でメトリクス値が変化したかどうかのみを考慮しており，値がどの程度変化したかということは反映されない．このハミング距離 DH に対し，ユークリッド距離 DE はメトリクス値の変化の大きさを反映した変動度を表す.

ユークリッド距離は，ピタゴラスの定理に基づいて定義されるユークリッド空間における 2 点間の距離である [29]．変動度としてのユークリッド距離 DE は，用いるメトリクスの数 y 次元ユークリッド空間上の距離ということになる．

ハミング距離 DH と同様に，ユークリッド距離 DE は連続する 2 つの変更間について導出されるので，最初の変更時刻 ct_1 については $null$ となる．モジュール md_i の変更時刻 ct_k におけるメトリクス値をベクトル $\vec{v}_{md_i,ct_k} = [v_{md_i,mt_1,ct_k}, v_{md_i,mt_2,ct_k}, \cdots, v_{md_i,mt_y,ct_k}]^T$ と表すことにすると，最初の変更以外の時刻 $ct_k(1 < k \leq z)$ でのモジュール md_i のユークリッド距離 DE_{md_i,ct_k} は以下の計算式で表される．

$$DE_{md_i,ct_k} = \sqrt{(\vec{v}_{md_i,ct_k} - \vec{v}_{md_i,ct_{k-1}})^T \cdot (\vec{v}_{md_i,ct_k} - \vec{v}_{md_i,ct_{k-1}})}$$

この計算式が示すように，ユークリッド距離 DE_{md_i,ct_k} は変更時刻 ct_{k-1} と ct_k 間の y 次元ユークリッド空間上の距離を表している．したがって，ハミング距離と同様に上記の計算式が成り立つの条件は $\forall k((1 < k \leq z) \wedge (v_{md_i,mt_j,ct_{k-1}} \neq null) \wedge (v_{md_i,mt_j,ct_k} \neq null))$ となる．この条件が成り立たない場合は，$DE_{md_i,ct_k} = null$ とする．

マハラノビス距離

ユークリッド距離が示す変動度は，メトリクス間に相関がなく，かつスケール差も

ないと見なした状態での値となっている．しかし，実際にはメトリクス間に相関やスケール差の存在することが多い．例えば，異なる定義によりモジュールの結合度を求める 2 つのメトリクスがあった場合，目的を同じくするこの 2 つのメトリクスには強い正の相関関係があると考えられる．また，ソースコードの行数を表すメトリクスなどは変更によって絶対値が大きく変化することも珍しくないが，サブクラスの数を表すメトリクスなどは変更によって変化する絶対値があまり大きくなることは少ないと考えられる．そこで，相関やスケール差を考慮した値を求めることのできるマハラノビス距離も利用する．

マハラノビス距離は多変量解析など統計学でよく用いられる距離の一種であり，情報分野においてもクラスタリング手法などにしばしば利用される [30][31][32][33].

ハミング距離 DH やユークリッド距離 DE と同様に，マハラノビス距離 DM は連続する 2 つの変更間について導出されるので，最初の変更時刻 ct_1 については $null$ となる．また，マハラノビス距離 DM はその計測に対象のソフトウェア全体でのメトリクスの共分散行列 Σ を必要とする．この共分散行列は $y \times y$ の正方行列である．モジュール md_i の変更時刻 ct_k におけるメトリクス値をベクトル $\vec{v}_{md_i,ct_k} = [v_{md_i,mt_1,ct_k}, v_{md_i,mt_2,ct_k}, \cdots, v_{md_i,mt_y,ct_k}]^T$ と表し，その内の要素に $null$ を含まないベクトルの個数を n $(1 \leq n \leq (x \times z))$ とする．この n 個のベクトルをそれぞれ $\vec{v}'_1, \vec{v}'_2, \cdots, \vec{v}'_n$ とおくと，メトリクスの共分散行列 Σ は以下の式で表すことができる．

$$\Sigma = \frac{1}{n} \sum_{l=1}^{n} (\vec{v}'_l - \vec{\mu}) \cdot (\vec{v}'_l - \vec{\mu})^T$$

$$\vec{\mu} = \frac{1}{n} \sum_{l=1}^{n} \vec{v}'_l$$

この共分散行列 Σ を用いて，最初の変更以外の時刻 $ct_k (1 < k \leq z)$ におけるモジュール md_i のマハラノビス距離 DM_{md_i,ct_k} は以下の計算式で表される．

$$DM_{md_i,ct_k} = \sqrt{(\vec{v}_{md_i,ct_k} - \vec{v}_{md_i,ct_{k-1}})^T \Sigma^{-1} (\vec{v}_{md_i,ct_k} - \vec{v}_{md_i,ct_{k-1}})}$$

計算式が示すように，マハラノビス距離 DM はメトリクスの共分散行列 Σ の逆行列を計算に用いる．したがって，共分散行列 Σ が特異行列（非正則行列）となる場合は，逆行列が存在しないため一切のマハラノビス距離が計測できない．このような場合にはマハラノビス距離 DM の計測自体を行わないものとする．例えば，あるメトリクス mt_j の値が常に 0 となるようなソフトウェアでは，共分散行列 Σ が特異行列となるので，マハラノビス距離 DM の計測は行わない．

また，ハミング距離 DH やユークリッド距離 DE と同様に，上記の計算式が成り立

つの条件は $\forall k((1 < k \leq z) \wedge (v_{md_i,mt_j,ct_{k-1}} \neq null) \wedge (v_{md_i,mt_j,ct_k} \neq null))$ となる．この条件が成り立たない場合は，$DM_{md_i,ct_k} = null$ とする．

メトリクス値の変化量

エントロピーからマハラノビス距離までの 7 つの変動度計測方法では，変更によってメトリクス値が変化したか，あるいはどの程度の大きさで変化したかということを用いて計測を行っているが，メトリクス値の変化が増加であるか減少であるかは区別していない．そのため，メトリクス値の増加と減少を区別したい場合，紹介した 7 つの方法は利用できない．そこで，メトリクス値の増減を区別できる変動度の計測方法であるメトリクス値の変化量 CV を新たに定義する．

メトリクス値の変化量が示す変動度は他の計測方法の変動度とは異なり，それぞれのモジュールの各メトリクスについて，最初の変更を除くそれぞれの変更毎に値が得られ，粒度が最も小さくなっている．距離と同様に，メトリクス値の変化量 CV は連続する 2 つの変更間について導出されるので，最初の変更時刻 ct_1 については $null$ となる．最初の変更以外の時刻 $ct_k(1 < k \leq z)$ でのモジュール md_i のメトリクス mt_j におけるメトリクス値の変化量 CV_{md_i,mt_j,ct_k} は以下の計算式で表される．

$$CV_{md_i,mt_j,ct_k} = v_{md_i,mt_j,ct_k} - v_{md_i,mt_j,ct_{k-1}}$$

この計算式が示すように，メトリクス値の変化量 CV_{md_i,mt_j,ct_k} は変更時刻 ct_{k-1} と ct_k 間でメトリクス mt_j の値がどの程度増減したかを表している．したがって，距離と同様に上記の計算式が成り立つの条件は $(1 < k \leq z) \wedge (v_{md_i,mt_j,ct_{k-1}} \neq null) \wedge (v_{md_i,mt_j,ct_k} \neq null)$ となる．この条件が成り立たない場合は，$CV_{md_i,mt_j,ct_k} = null$ とする．

以上の 8 つの方法は，表 1 及び表 2 に示すように，それぞれ得られる結果の意味など特徴が異なる．表 1 及び表 2 で用いている項目の意味はそれぞれ次の通りである．

- 粒度（次元）
  変動度が何を対象にした値を示すかということを表す．次元が高いほど粒度としては小さいことになる．

- 変更順序
  メトリクス値の変化の順序が変動度に影響するかどうかを表す．

- スケールの変化
  いずれかのメトリクスのスケールが全て n 倍されたときなど，あるメトリクスのスケールが変化したときに変動度の値に影響が出るかどうかを表す．メトリクス間のスケー

表 1: 求める変動度の特徴（1/2）

エントロピー H	
粒度（次元）	2 次元（モジュール，メトリクス）
変更順序	影響しない
スケールの変化	影響しない
備考	最小値は 0，最大値は変更回数に依存
正規化エントロピー H'	
粒度（次元）	2 次元（モジュール，メトリクス）
変更順序	影響しない
スケールの変化	影響しない
備考	最小値は 0，最大値は 1
四分位偏差 Q	
粒度（次元）	2 次元（モジュール，メトリクス）
変更順序	影響しない
スケールの変化	影響する
備考	異なるメトリクス間の比較には向かない
四分位分散係数 Q'	
粒度（次元）	2 次元（モジュール，メトリクス）
変更順序	影響しない
スケールの変化	影響しない
備考	-

表 2: 求める変動度の特徴（2/2）

ハミング距離 DH	
粒度（次元）	2 次元（モジュール，変更）
変更順序	影響する
スケールの変化	影響しない
備考	-
ユークリッド距離 DE	
粒度（次元）	2 次元（モジュール，変更）
変更順序	影響する
スケールの変化	影響する
備考	メトリクス間の相関やスケール差はないものと見なしている
マハラノビス距離 DM	
粒度（次元）	2 次元（モジュール，変更）
変更順序	影響する
スケールの変化	影響しない
備考	計測が不可能な場合がある
メトリクス値の変化量 CV	
粒度（次元）	3 次元（モジュール，メトリクス，変更）
変更順序	影響する
スケールの変化	影響する
備考	メトリクス値の変動の正負を区別する

ルの変化が変動度の計測に影響するかどうかであって，メトリクス値の変化の大きさが変動度の計測に影響するかどうかではない．

- 備考
  その他の特徴を記載している．

3.2.2 変動度の計測手段の拡張

3.2.1 節の計測手段によって導かれる変動度は，2 つあるいは 3 つの次元を持つ．例えばエントロピーはモジュールとメトリクスの 2 つの次元を持ち，ハミング距離はモジュールと変更時刻の 2 つの次元を持つ．これらの変動度をモジュール，メトリクス，あるいは変更時刻のいずれかの次元を 1 つ落とせば，残った 1 つあるいは 2 つの次元に対する変動度を新たに求めることができる．エントロピーを例にとると，モジュールの次元を落とせばメトリクスについての変動度が，逆にメトリクスの次元を落とせばモジュールについての変動度が求まる．すなわち，3.2.1 節の計測手段によって導出される変動度は注目する側面を変えることにより，モジュールの変動度や変更の変動度，メトリクスの変動度など，新たな変動度として捉えることができる．3.2.1 節の計測手段による変動度に加え，これらの新たな変動度を用いてソフトウェアの特性を分析する．

以下では，計測手段の拡張によって導出される**モジュールの変動度**，**メトリクスの変動度**，**変更の変動度**，**（モジュール，メトリクス）の変動度**，**（モジュール，変更）の変動度**，**（モジュール，メトリクス，変更）の変動度**について述べる．

モジュールの変動度

モジュールの変動度は 1 次元であり，それぞれのモジュールについて 1 つの値を持つものである．

この変動度を用いれば，対象のソフトウェアにおいて注目すべきモジュールを知ることができる．変動度の大きいモジュールはメトリクス値が変化しやすいので，他のモジュールに影響を与えやすい，あるいは逆に影響を受けやすい，または頻繁に変更が行われていると考えられる．この変動度の大きさに応じて力を割くモジュールを決定すれば，ソフトウェアの開発や保守を効率的に行える [34].

モジュールの変動度としては以下のものを用いる．ただし，$FMD(H)$ や $FMD(HD)$ などの FMD とは fluctuation of module の略である．

- **エントロピーによるモジュールの変動度** $FMD(H)$

  $FMD(H)$ はエントロピー H のモジュールの側面について注目したものである．具体的には，あるモジュールにおける全てのメトリクスについてエントロ

ピーの総和を取った値で表現される．モジュール md_i の変動度 $FMD(H)_{md_i}$ は以下の式で表される．

$$FMD(H)_{md_i} = \sum_{j=1}^{y} H_{md_i,mt_j}$$

- **正規化エントロピーによるモジュールの変動度** $FMD(H')$

  $FMD(H')$ は正規化エントロピー H' のモジュールの側面について注目したものである．具体的には，あるモジュールにおける全てのメトリクスについて正規化エントロピーの総和を取った値で表現される．モジュール md_i の変動度 $FMD(H')_{md_i}$ は以下の式で表される．

$$FMD(H')_{md_i} = \sum_{j=1}^{y} H'_{md_i,mt_j}$$

- **四分位分散係数によるモジュールの変動度** $FMD(Q')$

  $FMD(Q')$ は四分位分散係数 Q' のモジュールの側面について注目したものである．具体的には，あるモジュールにおける全てのメトリクスについて四分位分散係数の総和を取った値で表現される．モジュール md_i の変動度 $FMD(H')_{md_i}$ は以下の式で表される．

$$FMD(Q')_{md_i} = \sum_{j=1}^{y} Q'_{md_i,mt_j}$$

- **ハミング距離によるモジュールの変動度** $FMD(DH)$

  $FMD(DH)$ はハミング距離 DH のモジュールの側面について注目したものである．具体的には，あるモジュールにおける全ての変更時刻についてハミング距離の総和を取った値で表現される．モジュール md_i の変動度 $FMD(DH)_{md_i}$ は以下の式で表される．

$$FMD(DH)_{md_i} = \sum_{k=1}^{z} DH_{md_i,ct_k}$$

  ただし，$DH_{md_i,ct_k} = null$ となっている場合は $DH_{md_i,ct_k} = 0$ として扱う．

- **ユークリッド距離によるモジュールの変動度** $FMD(DE)$

  $FMD(DE)$ はユークリッド距離 DE のモジュールの側面について注目したものである．具体的には，あるモジュールにおける全ての変更時刻についてユークリッド距離の総和を取った値で表現される．モジュール md_i の変動度 $FMD(DE)_{md_i}$ は以下の式で表される．

$$FMD(DE)_{md_i} = \sum_{k=1}^{z} DE_{md_i,ct_k}$$

ただし，$DE_{md_i,ct_k} = null$ となっている場合は $DE_{md_i,ct_k} = 0$ として扱う．

- **マハラノビス距離によるモジュールの変動度** $FMD(DM)$

  $FMD(DM)$ はマハラノビス距離 DM のモジュールの側面について注目したものである．具体的には，あるモジュールにおける全ての変更時刻についてマハラノビス距離の総和を取った値で表現される．当然ながらマハラノビス距離が計測不能で存在しない場合，$FMD(DM)$ も導出しない．モジュール md_i の変動度 $FMD(DM)_{md_i}$ は以下の式で表される．

$$FMD(DM)_{md_i} = \sum_{k=1}^{z} DM_{md_i,ct_k}$$

  ただし，$DM_{md_i,ct_k} = null$ となっている場合は $DM_{md_i,ct_k} = 0$ として扱う．

四分位偏差 Q のメトリクスの次元を落としたものは用いない．なぜならば，各メトリクスにスケール差が存在する場合，四分位偏差にも各メトリクスのスケール差が影響しているので，単純にメトリクス間の総和を取って変動度とするには問題があるからである．

メトリクスの変動度

メトリクスの変動度は 1 次元であり，それぞれのメトリクスについて 1 つの値を持つものである．

この変動度を用いれば，対象のソフトウェアにおいてどのメトリクスの値が変化しやすく，どのメトリクスの値が変化しにくいか，というソフトウェアの特性を知ることができる．

メトリクスの変動度としては以下のものを用いる．ただし，$FMT(H)$ などの FMT とは fluctuation of metric の略である．

- **エントロピーによるメトリクスの変動度** $FMT(H)$

  $FMT(H)$ はエントロピー H のメトリクスの側面について注目したものである．具体的には，あるメトリクスについて全てのモジュールのエントロピーの総和を取った値で表現される．メトリクス mt_j の変動度 $FMT(H)_{mt_j}$ は以下の式で表される．

$$FMT(H)_{mt_j} = \sum_{i=1}^{x} H_{md_i,mt_j}$$

- **正規化エントロピーによるメトリクスの変動度** $FMT(H')$

  $FMT(H')$ は正規化エントロピー H' のメトリクスの側面について注目したものである．具体的には，あるメトリクスについて全てのモジュールの正規化エン

トロピーの総和を取った値で表現される．メトリクス mt_j の変動度 $FMT(H')_{mt_j}$ は以下の式で表される．

$$FMT(H')_{mt_j} = \sum_{i=1}^{x} H'_{md_i,mt_j}$$

- **四分位偏差によるメトリクスの変動度** $FMT(Q)$

  $FMT(Q)$ は四分位偏差 Q のメトリクスの側面について注目したものである．具体的には，あるメトリクスについて全てのモジュールの四分位偏差の総和を取った値で表現される．メトリクス mt_j の変動度 $FMT(Q)_{mt_j}$ は以下の式で表される．

$$FMT(Q)_{mt_j} = \sum_{i=1}^{x} Q_{md_i,mt_j}$$

- **四分位分散係数によるメトリクスの変動度** $FMT(Q')$

  $FMT(Q')$ は四分位分散係数 Q' のメトリクスの側面について注目したものである．具体的には，あるメトリクスについて全てのモジュールの四分位分散係数の総和を取った値で表現される．メトリクス mt_j の変動度 $FMT(Q')_{mt_j}$ は以下の式で表される．

$$FMT(Q')_{mt_j} = \sum_{i=1}^{x} Q'_{md_i,mt_j}$$

変更の変動度

変更の変動度は 1 次元であり，それぞれの変更時刻について 1 つの値を持つものである．

この変動度を用いれば，対象のソフトウェアにおいていつどのような変更が行われたのか，というソフトウェアの特性を知ることができる．この特性はソフトウェアの開発履歴を理解するのにも有効であるが，開発を行った意図に反した影響がソフトウェアに出ていないか確かめるのに用いることもできる．例えば，小規模な影響しか与えないつもりで行った変更の変動度が大きければ，意図しない影響が発生していたり，変更自体に問題があった可能性がある．

また，この変更の変動度と変更履歴情報から得られる変更を行った開発者名を対応付ければ，対象のソフトウェアに対してそれぞれの開発者がどの程度の変動度を発生させたのか知ることも可能である．

変更の変動度としては以下のものを用いる．ただし，$FCT(DH)$ などの FCT とは fluctuation of change time の略である．

- **ハミング距離による変更の変動度** $FCT(DH)$

  $FCT(DH)$ はハミング距離 DH の変更時刻の側面について注目したものであ

る．具体的には，ある変更時刻について全てのモジュールのハミング距離の総和を取った値で表現される．変更時刻 ct_k の変動度 $FCT(DH)_{ct_k}$ は以下の式で表される．

$$FCT(DH)_{ct_k} = \sum_{i=1}^{x} DH_{md_i,ct_k}$$

ただし，$DH_{md_i,ct_k} = null$ となっている場合は $DH_{md_i,ct_k} = 0$ として扱う．

- **ユークリッド距離による変更の変動度** $FCT(DE)$

  $FCT(DE)$ はユークリッド距離 DE の変更時刻の側面について注目したものである．具体的には，ある変更時刻について全てのモジュールのユークリッド距離の総和を取った値で表現される．変更時刻 ct_k の変動度 $FCT(DE)_{ct_k}$ は以下の式で表される．

$$FCT(DE)_{ct_k} = \sum_{i=1}^{x} DE_{md_i,ct_k}$$

ただし，$DE_{md_i,ct_k} = null$ となっている場合は $DE_{md_i,ct_k} = 0$ として扱う．

- **マハラノビス距離による変更の変動度** $FCT(DM)$

  $FCT(DM)$ はマハラノビス距離 DM の変更時刻の側面について注目したものである．具体的には，ある変更時刻について全てのモジュールのハミング距離の総和を取った値で表現される．変更時刻 ct_k の変動度 $FCT(DM)_{ct_k}$ は以下の式で表される．

$$FCT(DM)_{ct_k} = \sum_{i=1}^{x} DM_{md_i,ct_k}$$

ただし，$DM_{md_i,ct_k} = null$ となっている場合は $DM_{md_i,ct_k} = 0$ として扱う．

（モジュール，メトリクス）の変動度

（モジュール，メトリクス）の変動度は 2 次元であり，それぞれのモジュールの各メトリクスについて 1 つの値を持つものである．

モジュールの変動度をメトリクス別に見たい場合や，メトリクスの変動度をモジュール別に見たい場合にこの変動度を用いる．

（モジュール，メトリクス）の変動度としては以下のものを用いる．ただし，$FMDMT(CV)$ の $FMDMT$ とは fluctuation of module and metric の略である．

- **エントロピー** H

  3.2.1 節で述べたエントロピー H を用いる．

- **正規化エントロピー** H'

  3.2.1 節で述べた正規化エントロピー H' を用いる．

- **四分位偏差 Q**

  3.2.1 節で述べた四分位偏差 Q を用いる．

- **四分位分散係数 Q'**

  3.2.1 節で述べた四分位分散係数 Q' を用いる．

- **メトリクス値の変化量による（モジュール，メトリクス）の変動度 $FMDMT(CV)$**

  $FMDMT(CV)$ は，メトリクス値の変化量 CV のモジュールとメトリクスの側面について注目したものである．具体的には，メトリクス値の変化量の変更時刻の次元をその総和を取ることによって落とした値で表現される．モジュール md_i，メトリクス mt_j の変動度 $FMDMT(CV)_{md_i,mt_j}$ は以下の式で表される．

$$FMDMT(CV)_{md_i,mt_j} = \sum_{k=1}^{z} CV_{md_i,mt,j,ct_k}$$

  ただし，$CV_{md_i,mt,j,ct_k} = null$ となっている場合は $CV_{md_i,mt,j,ct_k} = 0$ として扱う．

（モジュール，変更）の変動度

（モジュール，変更）の変動度は 2 次元であり，それぞれのモジュールの各メトリクスについて 1 つの値を持つものである．

モジュールの変動度をそれぞれの変更別に見たい場合や，変更の変動度をモジュール別に見たい場合にこの変動度を用いる．

（モジュール，変更）の変動度としては以下のものを用いる．

- **ハミング距離 DH**

  3.2.1 節で述べたハミング距離 DH を用いる．

- **ユークリッド距離 DE**

  3.2.1 節で述べたユークリッド距離 DE を用いる．

- **マハラノビス距離 DM**

  3.2.1 節で述べたマハラノビス距離 DM を用いる．ただし，マハラノビス距離が計測可能であった場合に限る．

メトリクス値の変化量 CV のメトリクスの次元を落としたものは用いない．なぜならば，メトリクス値の変化量 CV は各メトリクスのスケール差の影響を直接受けるため，単純にメトリクス間の総和を取って変動度とするには問題があるからである．

（モジュール，メトリクス，変更）の変動度

（モジュール，メトリクス，変更）の変動度は 3 次元であり，それぞれのモジュー

ルの各メトリクスについて，変更時刻ごとに 1 つの値を持つものである．

（モジュール，メトリクス）の変動度をそれぞれの変更別に見たい場合や，（モジュール，変更）の変動度をメトリクス別に見たい場合にこの変動度を用いる．

（モジュール，メトリクス，変更）の変動度としてはメトリクス値の変化量 CV を用いる．

以上をまとめると，表 3 のようになる．

表 3: 変動度とその値の種類

変動度	値の種類
モジュール	$FMD(H), FMD(H'), FMD(Q'),$ $FMD(DH), FMD(DE), FMD(DM)$
メトリクス	$FMT(H), FMT(H'), FMT(Q), FMT(Q')$
変更	$FCT(DH), FCT(DE), FCT(DM)$
（モジュール，メトリクス）	$H, H', Q, Q', FMDMT(CV)$
（モジュール，変更）	DH, DE, DH
（モジュール，メトリクス，変更）	CV

4　メトリクス値の変遷に基づくソフトウェアの特性分析手法

本節では，ソフトウェアの理解などの面で有益な情報であるソフトウェアの特性を分析する手法について述べる．

提案手法の概要を図 2 に示す．前提となるのは，対象のソフトウェアの開発にバージョン管理システムが利用されていることである．

本手法は次の手順から構成される．

手順 1：過去のスナップショットの取得

バージョン管理システムに保存された，分析対象のソフトウェアの変更履歴情報を利用し，過去のソースコード全てのスナップショットを取得する．

手順 2：スナップショットに対するメトリクス値の計測

全ての変更時に対応する過去のスナップショットを対象にメトリクス値の計測を行う．これにより，任意のモジュールに対するメトリクス値の変遷が得られる．

手順 3：変動度の計測

メトリクス値の変遷を用い，各種の変動度を計測する．

手順 4：変動度を用いたソフトウェアの特性分析

手順 3 で求めた変動度を用い，ユーザによるソフトウェアの特性分析を行う．

以下では，これらの手順についてそれぞれ説明する．

4.1　過去のスナップショットの取得

バージョン管理システムに保存された分析対象のソフトウェアの変更履歴情報を利用し，過去全てのソースコードのスナップショットを取得する．

本手法ではソースコードのスナップショットが変化する毎に，すなわちいずれかのソースファイルが変更される毎に，その時点でのスナップショットを取得する．ただし，複数のソースファイルが同時に変更されている，つまり 1 回のチェックインによって変更が行われている場合は，それら複数の変更をまとめて 1 回の変更と見なす．

図 3 はソフトウェアの変更履歴の例を示したものである．図 3 では，このソフトウェアに存在する 4 つのソースファイル F_1, F_2, F_3, F_4 が縦軸に配置されている．横軸は時刻を表し，時刻 ct_1, ct_2, ct_3, ct_4 において変更が行われている．このとき，ct_1 がソフトウェアの最初の変更，ct_4 がソフトウェアの最新の変更を指すものとする．また，$F_{1,ver.1}$ や $F_{4,ver.3}$ などはソースファイルとそのバージョンを示している．図 3 に示すようなソフトウェアの変更履

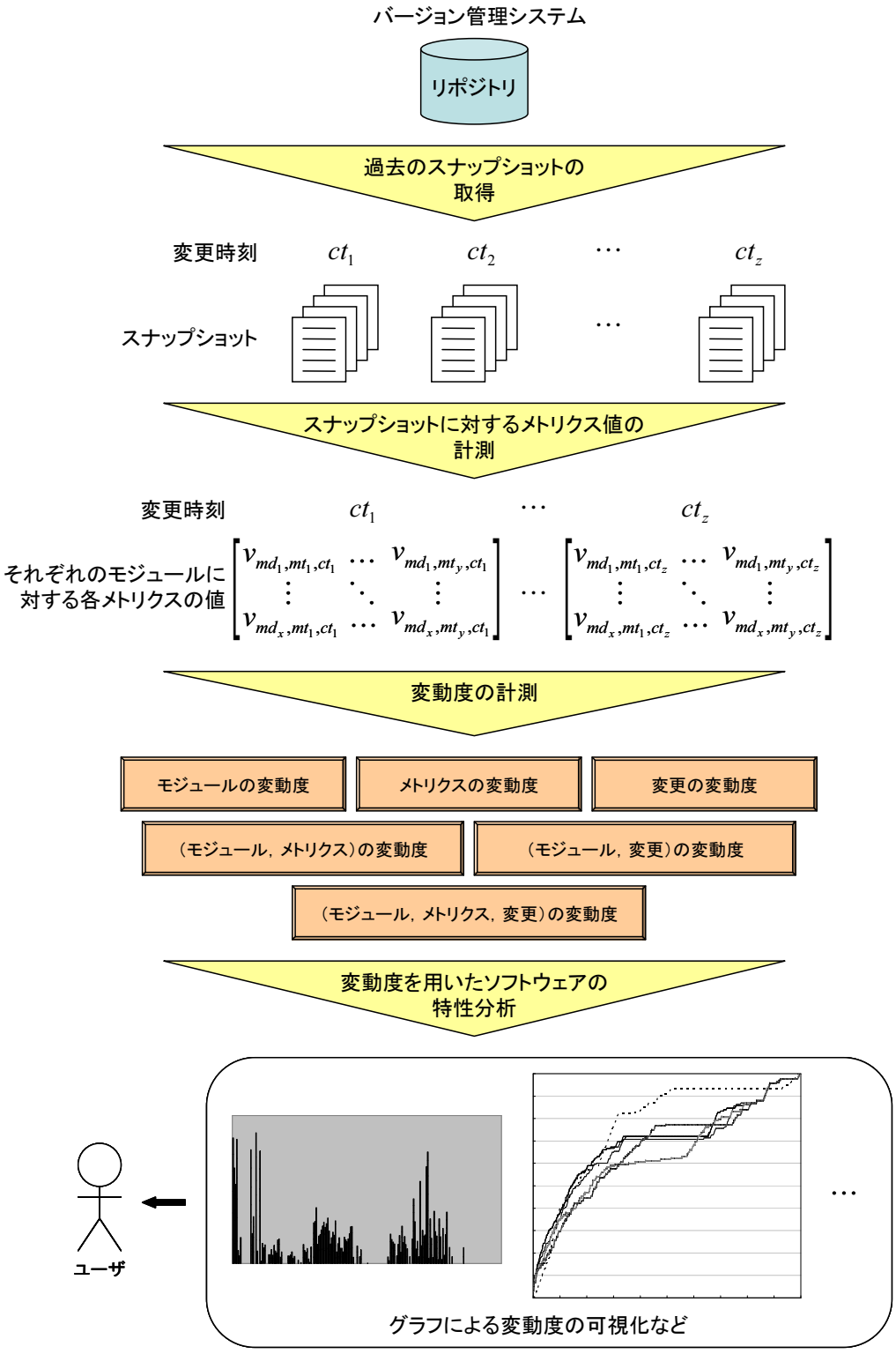

バージョン管理システム
リポジトリ
過去のスナップショットの
取得
変更時刻
ct_1
ct_2
…
ct_z
スナップショット
スナップショットに対するメトリクス値の
計測
変更時刻
ct_1
…
ct_z
それぞれのモジュールに
対する各メトリクスの値
v_{md_1,mt_1,ct_1} … v_{md_1,mt_y,ct_1}
v_{md_x,mt_1,ct_1} … v_{md_x,mt_y,ct_1}
v_{md_1,mt_1,ct_z} … v_{md_1,mt_y,ct_z}
v_{md_x,mt_1,ct_z} … v_{md_x,mt_y,ct_z}
変動度の計測
モジュールの変動度
メトリクスの変動度
変更の変動度
（モジュール，メトリクス）の変動度
（モジュール，変更）の変動度
（モジュール，メトリクス，変更）の変動度
変動度を用いたソフトウェアの
特性分析
ユーザ
…
グラフによる変動度の可視化など

図 2: 提案手法の概要

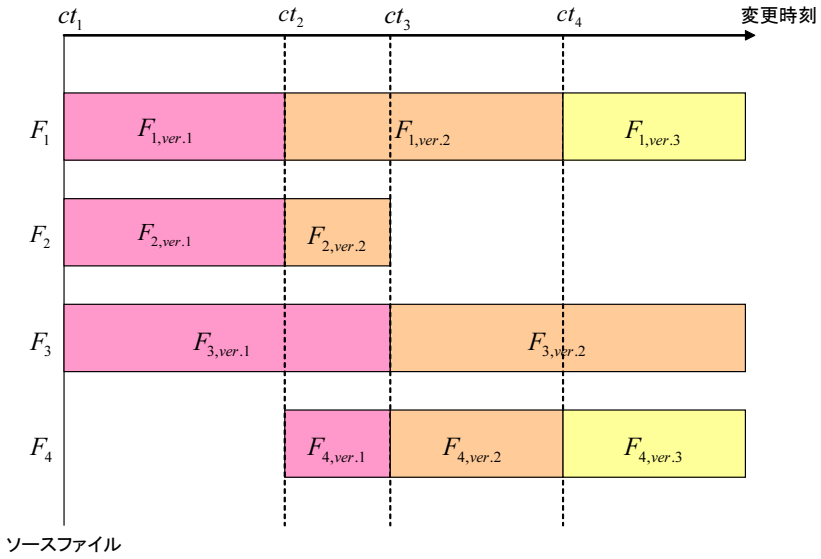


図 3: ソフトウェアの変更履歴の例

歴があったとき，変更時刻 ct_1, ct_2, ct_3, ct_4 におけるスナップショットを取得する．変更時刻 ct_1 では，3 つのソースファイル F_1, F_2, F_3 が変更（作成）されている．変更時刻 ct_2 では，F_1, F_2 が変更され，F_4 が作成されている．変更時刻 ct_3 では，F_3 が変更され，F_2 が消去されている．そして変更時刻 ct_4 では F_1, F_4 が変更されている．このとき，それぞれの変更時刻におけるソースコードのスナップショットは表 4 に示す通りになる．

バージョン管理システムを用いて過去のスナップショットのを取得する方法はいくつか考えられる．取得方法の例を以下に挙げる．

- バージョン管理システムのコマンドを用い，変更履歴情報を取得．次に全ての変更日時について，その時点でのソフトウェアを全てチェックアウトする．
- バージョン管理システムのコマンドを用い，変更履歴情報を取得．次に最新あるいは

表 4: 図 3 に対応するスナップショット

変更時刻	スナップショット
ct_1	$(F_{1,ver.1}, F_{2,ver.1}, F_{3,ver.1})$
ct_2	$(F_{1,ver.2}, F_{2,ver.2}, F_{3,ver.1}, F_{4,ver.1})$
ct_3	$(F_{1,ver.2}, F_{3,ver.2}, F_{4,ver.2})$
ct_4	$(F_{1,ver.3}, F_{3,ver.2}, F_{4,ver.3})$

最初の状態のソフトウェアをチェックアウトし，全ての変更における変更前と変更後の差分を利用して各時点でのソフトウェアのスナップショットを導出する．

- バージョン管理システムのリポジトリを直接解析し，全ての変更時点におけるソフトウェアのスナップショットを導出する．

バージョン管理システムのコマンドを利用する方法は，リポジトリが直接参照できない場合でも用いることができるのが利点であるが，ネットワーク越しにバージョン管理システムと何度もやり取りを行うため非常に時間がかかるのが難点である．

一方，バージョン管理システムのリポジトリを直接解析する方法は，比較的高速に過去全てのスナップショットを得られるのが利点である．しかし，リポジトリを直接参照できる環境でなければならないというのが問題となる．

後述の実験においては，実行時間を重視し，バージョン管理システムのリポジトリを直接解析する方法用いる．

4.2 スナップショットに対するメトリクス値の計測

全ての変更時に対応するスナップショットを対象にメトリクス値の計測を行う．これにより，任意のモジュールに対するメトリクス値の変遷が得られる．

メトリクス値 v は，任意のモジュール md_i と用いるメトリクス mt_j，そして任意の変更時刻 ct_k に対して得られるので v_{md_i,mt_j,ct_k} と表すことができる．

モジュールの単位は目的に合わせて選ぶことができる．ソフトウェアを構成する機能単位で注目したいのであればモジュールの単位を機能に，ファイルに注目したいのであればモジュールの単位をファイルにする．対象ソフトウェアがオブジェクト指向言語で作られている場合はクラスやメソッドをモジュールの単位とすることも可能である．

一方，用いるメトリクスはモジュールの単位や目的に合わせて選ぶ必要がある．例えば，モジュールの単位をクラスとしたいのであればクラスを対象としたメトリクスを，ファイルとしたいのであればファイルを対象としたメトリクスを用いる．あるいは，結合度に注目したいのであれば結合度を表すメトリクスを，凝集度に注目したいのであれば凝集度を表すメトリクスを用いる．

後述の実験においては，モジュールの単位をクラスとし，2.2.2 節で述べた CK メトリクス [7] などの広く利用されているメトリクスを用いる．

4.3 変動度の計測

メトリクス値の変遷を用い，メトリクス値の変動度を計測する．

変動度の導出手段は 3.2 節で述べた通りである．
3.2 節で述べたように，導出される変動度の種類は大別すると以下の 5 つとなる．

- モジュールの変動度
- メトリクスの変動度
- 変更の変動度
- （モジュール，メトリクス）の変動度
- （モジュール，変更）の変動度
- （モジュール，メトリクス，変更）の変動度

4.4　変動度を用いたソフトウェアの特性分析

4.3 節で求めた変動度を用いてソフトウェアの特性分析を行う．
ソフトウェアの特性分析は，変動度の値やそれをグラフなどの形式で可視化したものを用いてユーザが実行する．
例えば，モジュールの変動度からどのモジュールのメトリクス値が変化しやすいのか判断したり，メトリクスの変動度から対象のソフトウェアにおいて変化しやすいメトリクスはどれかを把握したりする．

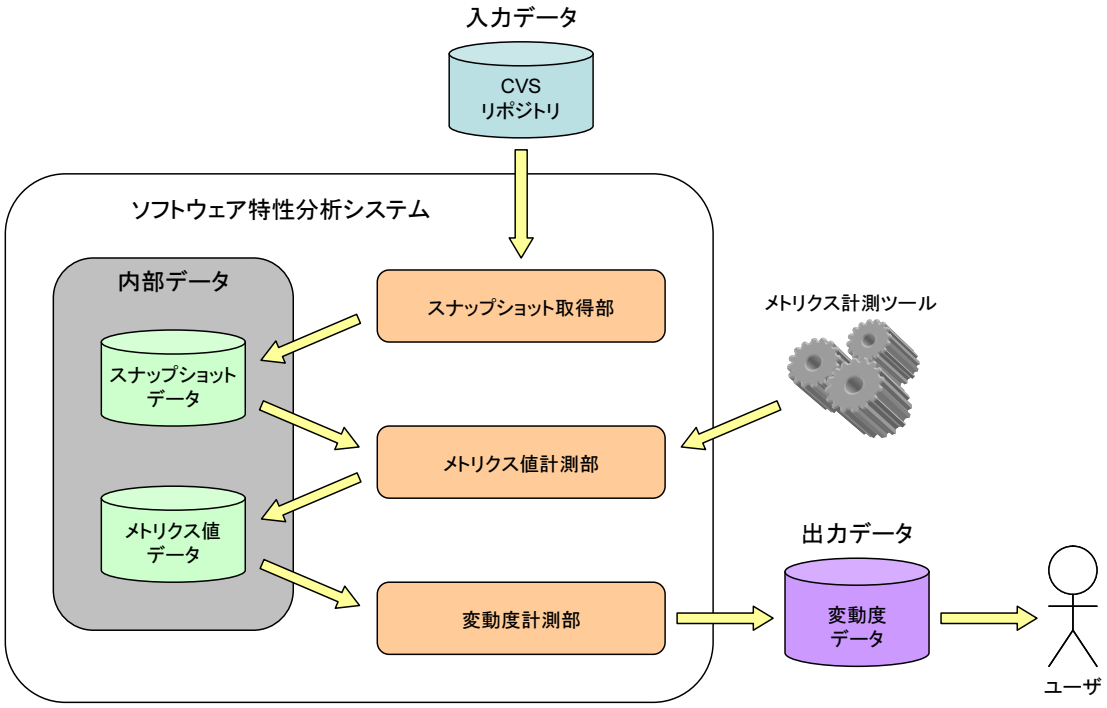


図 4: ソフトウェア特性分析システムの概要

5 実装

本節では，4 節で述べた提案手法を実現したシステムについての説明を行う．開発に用いた言語は Java で，開発環境は以下の通りである．

- CPU : Intel(R) Core(TM) 2 Duo 1.86GHz
- RAM : 2GB
- OS : Microsoft Windows XP Professional, Version 2002, Service Pack 2
- JDK 5.0

本システムの概要を図 4 に示す．

本システムが対象とするソフトウェアは，バージョン管理システムとして CVS を利用している必要がある．

本システムは大別して内部データ，スナップショット取得部，メトリクス値計測部，変動度計測部の 4 つの部分から構成される．以下ではこの 4 つについて説明を行う．

5.1 内部データ

内部データにはスナップショットデータ，メトリクス値データの 2 つがある．どちらのデータも，システム実行中はファイルとしてハードディスク上に保存される．

- スナップショットデータ

  スナップショットデータは，過去全てのスナップショットに関する情報を持つ．具体的には，過去全ての変更時刻に関して以下の情報を持つ．

  - スナップショットに含まれるソースコードファイルの名前
  - スナップショットに含まれるソースコードファイルの内容
  - 変更時刻
  - 変更者の名前

  スナップショットデータは，スナップショット取得部の出力であり，メトリクス値計測部の入力である．

  データはテキストベースのファイルとして保存される．

- メトリクス値データ

  メトリクス値データは，用いるメトリクスそれぞれについて，過去全ての変更時刻における各モジュールのメトリクス値を持つ．

  メトリクス値データは，メトリクス値計測部の出力であり，変動度計測部の入力である．

  データは CSV 形式のファイルとして保存される．

5.2 スナップショット取得部

スナップショット取得部では，CVS リポジトリを入力とし，スナップショットデータを出力する．

手順は以下のようになる．

1. CVS リポジトリを直接解析し，変更履歴情報を取得
2. 変更履歴情報から全ての変更時刻におけるスナップショットを構成
3. スナップショットと一部の変更履歴情報をスナップショットデータとして出力

変更時刻の単位は秒であるが，同時に複数のファイルを変更した場合，CVS の処理速度によっては 1 つの変更の時刻が複数存在する場合がある．そこで本システムでは，連続する

複数の変更を同じ変更と見なす「隣接する変更間の最大時間」が設定可能になっている．デフォルトでは，隣接する変更間の最大時間は 2 秒である．

複数の変更を 1 つの変更と見なした場合，その変更の時刻には最も新しい時刻を用いる．

5.3 メトリクス値計測部

メトリクス値計測部では，スナップショットデータを入力とし，メトリクス値データを出力する．

手順は以下のようになる．

1. スナップショットデータにおける各変更時刻のスナップショットに対して外部のメトリクス計測ツールを実行
2. メトリクス計測ツールの実行結果をメトリクス値データとして出力

外部のメトリクス計測ツールには，我々の研究チームが開発しているメトリクス計測ツールを用いる．

5.4 変動度計測部

変動度計測部では，メトリクス値データを入力とし，変動度データを出力する．

手順は以下のようになる．

1. メトリクス値データからモジュールの変動度，メトリクスの変動度，変更の変動度，（モジュール，メトリクス）の変動度，(モジュール，変更）の変動度，(モジュール，メトリクス，変更）の変動度を導出
2. それぞれの変動度を変動度データとして出力

変動度データは CSV 形式のファイルとして出力される．

ソフトウェアの特性分析は，ユーザが表計算ソフトなどを用いて出力された変動度データを可視化することによって行う．

6　適用事例

本節では，5 節で紹介したシステムを実存するオープンソースソフトウェアに適用した事例について述べる．以降，本節においては 5 節で紹介したシステムを単にシステムと呼ぶ．

まず，システムの実行環境や適用対象，用いるメトリクス，モジュールの粒度といった適用実験の概要について説明し，その次に適用結果と考察を述べる．

6.1　概要

適用実験を行ったシステムの実行環境，適用事例に用いる対象ソフトウェア，メトリクス，そしてモジュールの粒度について説明する．

6.1.1　実行環境

適用事例において，システムの実行環境は以下の通りである．

- CPU : Intel(R) Core(TM) 2 Duo 1.86GHz
- RAM : 2GB
- OS : Microsoft Windows XP Professional, Version 2002, Service Pack 2

6.1.2　適用対象ソフトウェア

適用対象として，SourceForge[35] にて公開されている以下の 3 つのオープンソースソフトウェアを用いる．

- FreeMind[36]
- JHotDraw[37]
- HelpSetMaker[38]

各対象ソフトウェアの概要を表 5，6，7 に示す．表 5，6，7 で用いている項目の意味はそれぞれ次の通りである．

- ソフトウェア名
  対象とするソフトウェアの名称を表す．
- 種別
  対象ソフトウェアがどのようなソフトウェアであるのかということを示す大まかな分類名を表す．

表 5: FreeMind の概要

ソフトウェア名	FreeMind
種別	ダイアグラム生成ツール
開発言語	Java
開発者数	12
総スナップショット数 z	225
最初の変更時刻 ct_1	2000/08/01 19:56:09
最後の変更時刻 ct_z	2008/01/13 20:55:35
ct_1 におけるソースファイル数	67
ct_z におけるソースファイル数	221
ct_1 におけるソースコードの総行数	3,882
ct_z におけるソースコードの総行数	39,350

表 6: JHotDraw の概要

ソフトウェア名	JHotDraw
種別	ドローツール
開発言語	Java
開発者数	24
総スナップショット数 z	196
最初の変更時刻 ct_1	2000/10/12 14:57:10
最後の変更時刻 ct_z	2005/04/25 22:35:57
ct_1 におけるソースファイル数	144
ct_z におけるソースファイル数	484
ct_1 におけるソースコードの総行数	12,781
ct_z におけるソースコードの総行数	60,430

表 7: HelpSetMaker の概要

ソフトウェア名	HelpSetMaker
種別	ドキュメント生成ツール
開発言語	Java
開発者数	2
総スナップショット数 z	260
最初の変更時刻 ct_1	2003/10/20 13:05:47
最後の変更時刻 ct_z	2006/01/07 15:08:41
ct_1 におけるソースファイル数	14
ct_z におけるソースファイル数	36
ct_1 におけるソースコードの総行数	797
ct_z におけるソースコードの総行数	9,167

- 開発言語
  対象ソフトウェアのソースコードが開発されている言語を表す．現在のところシステムが対応しているのは Java 言語のみであるため，全ての対象ソフトウェアの開発言語は Java である．

- 開発者数
  対象ソフトウェアに携わる開発者の数を表す．実装を行っている開発者だけでなく，設計やテストを行っている開発者も含む．

- 総スナップショット数 z
  システムが対象とする全ての変更時刻の数を表す．

- 最初の変更時刻 ct_1
  最初の変更を CVS に登録した時刻を表す．

- 最後の変更時刻 ct_z
  システムが用いる CVS リポジトリにおいて最後の変更を登録した時刻を表す．

- ct_1 におけるソースファイル数
  最初の変更時刻 ct_1 でのスナップショットにおけるソースファイルの数を表す．

- ct_z におけるソースファイル数
  最後の変更時刻 ct_z でのスナップショットにおけるソースファイルの数を表す．

- ct_1 におけるソースコードの総行数
  最初の変更時刻 ct_1 でのスナップショットにおけるソースコードの行数の総和を表す．

- ct_z におけるソースコードの総行数
  最後の変更時刻 ct_z でのスナップショットにおけるソースコードの行数の総和を表す．

6.1.3 用いるメトリクス

適用実験においては，2.2.2 節で紹介したメトリクスの内，以下の 6 つのメトリクスを用いる．

- LOC
- RFC
- CBO

- LCOM
- NOC
- DIT

これらのメトリクスを用いる理由は，一般的に用いられるメトリクスであり，特に CK メトリクス（RFC, CBO, LCOM, NOC, DIT）についてはその有効性が報告されているためである [8].

6.1.4　モジュールの粒度

適用実験におけるモジュールの粒度はクラスとする．

モジュールをクラスとする理由は，6.1.3 節で述べたメトリクスのほとんどがクラスを対象としているためである．

6.2　適用結果

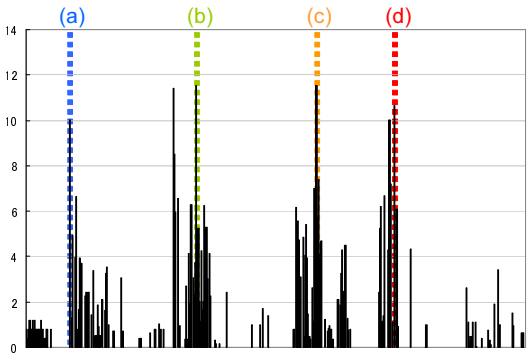
それぞれの対象ソフトウェアに対しシステムを適用した．ソフトウェア FreeMind, JHotDraw, HelpSetMaker に対するシステムの実行時間は，それぞれ約 28 分，40 分，17 分となった．

以下では，FreeMind に対するシステムの適用結果と，システムの適用結果として得られる各種変動度を用いたソフトウェアの特性分析についての考察を述べる．JHotDraw, HelpSetMaker の 2 つについてはその適用結果を本論文末に付録として掲載する．

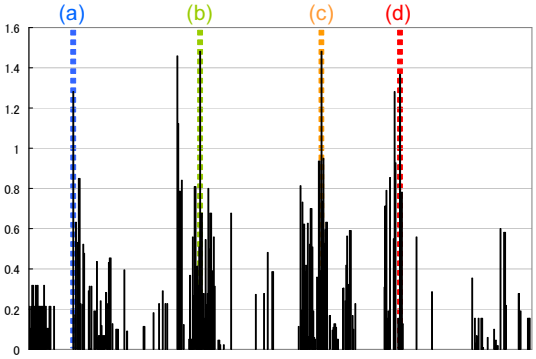
6.2.1　モジュールの変動度

3.2 節で述べたように，モジュールの変動度として用いる指標は $FMD(H)$, $FMD(H')$, $FMD(Q')$, $FMD(DH)$, $FMD(DE)$, $FMD(DM)$ の 6 つである．

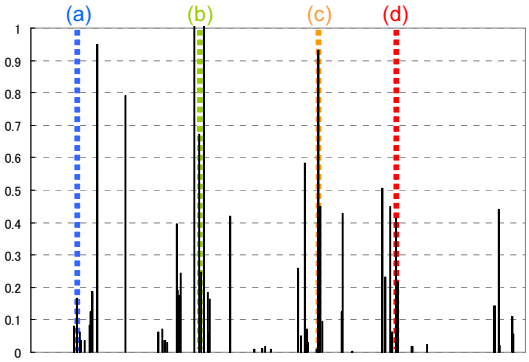
それぞれの指標のグラフを図 5 に示す．グラフの横軸はモジュールであり，合計 486 クラスが全てのグラフにおいて同順で並んでいる．グラフの縦軸はそれぞれの指標の値を表している．また，図 5 における点線 (a), (b), (c), (d) はほとんどのグラフにおいて特に値の大きいモジュールを示したものである．(a) の点線が示すクラスは `freemind.controller.Controller` クラス，(b) は `freemind.modes.MapAdapter` クラス，(c) は `freemind.modes.mindmapmode.MindMapMapModel` クラス，(d) は `freemind.view.mindmapview.NodeView` クラスとなっている．



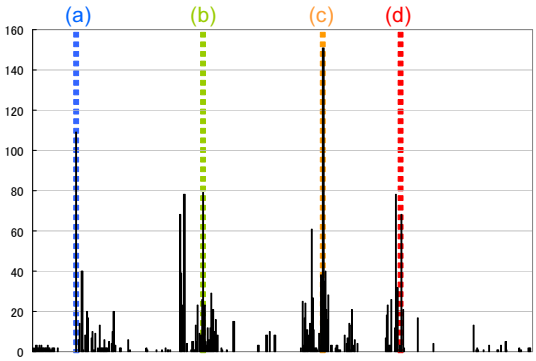
(1) $FMD(H)$



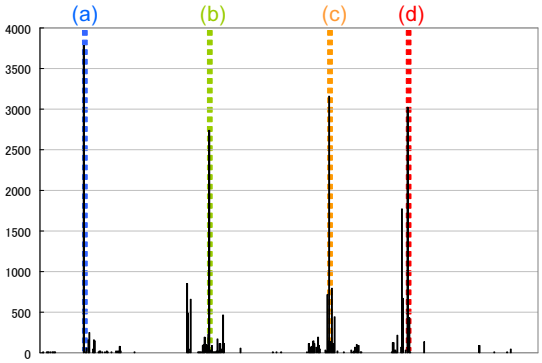
(2) $FMD(H')$



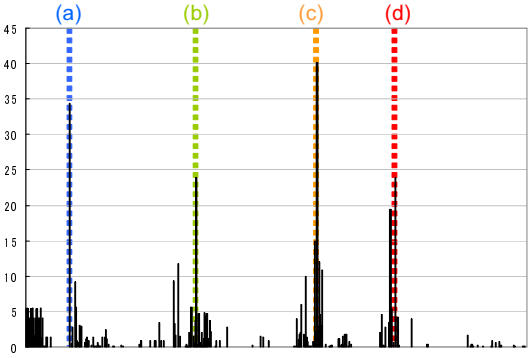
(3) $FMD(Q')$



(4) $FMD(DH)$

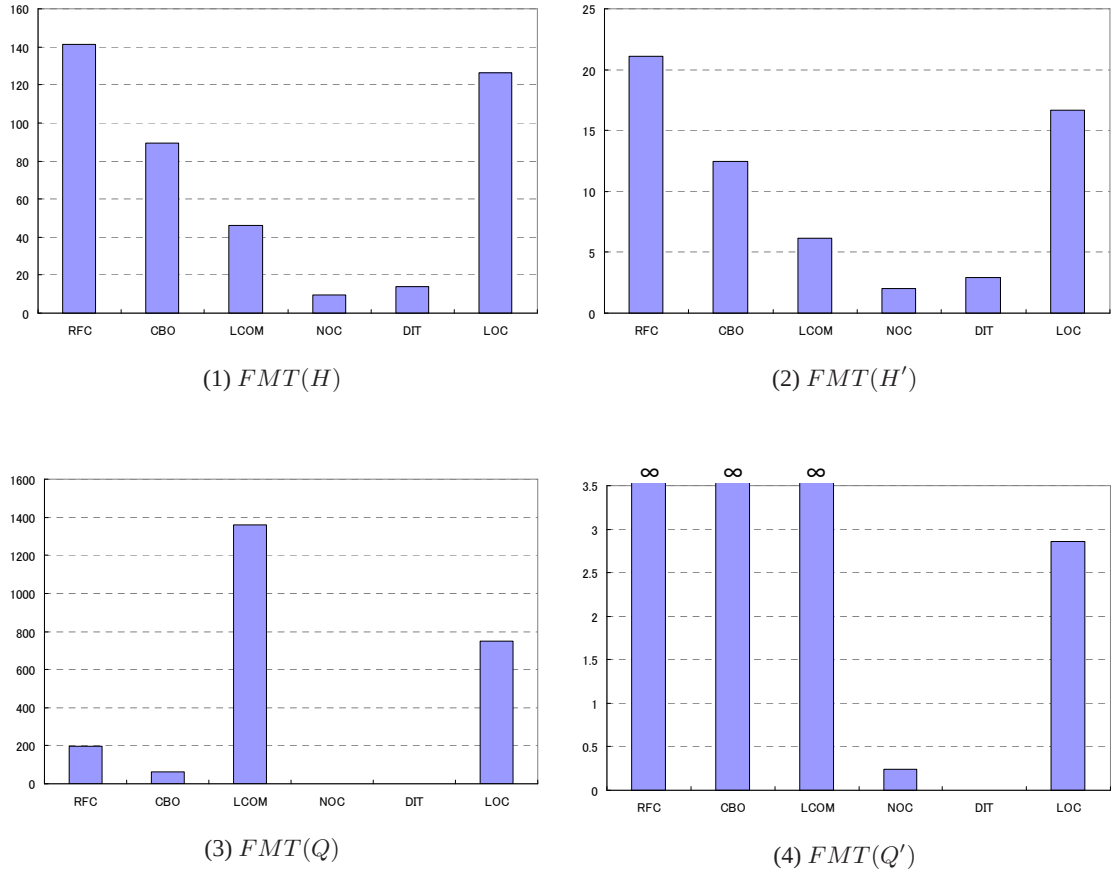


(5) $FMD(DE)$



(6) $FMD(DM)$

図 5: FreeMind におけるモジュールの変動度



(1) $FMT(H)$ (2) $FMT(H')$

(3) $FMT(Q)$ (4) $FMT(Q')$

図 6: FreeMind におけるメトリクスの変動度

6.2.2 メトリクスの変動度

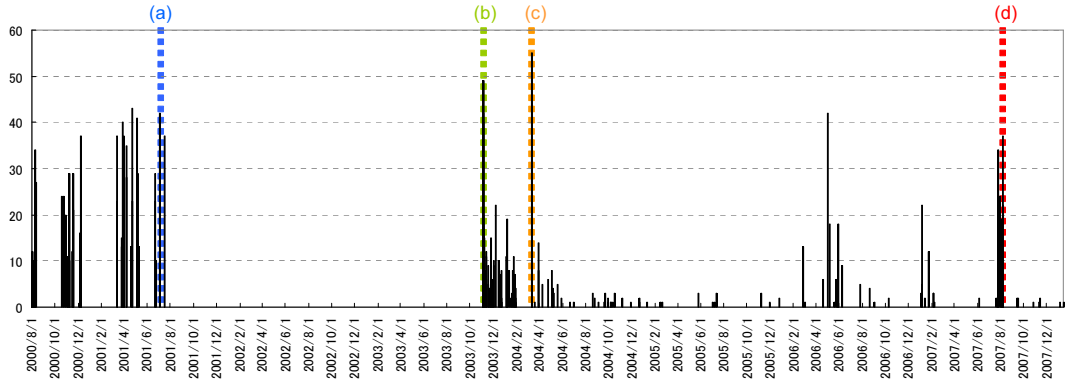
3.2 節で述べたように，メトリクスの変動度として用いる指標は $FMT(H), FMT(H'), FMT(Q), FMT(Q')$ の 4 つである．

それぞれの指標のグラフを図 6 に示す．グラフの横軸はメトリクスであり，グラフの縦軸はそれぞれの指標の値を表している．

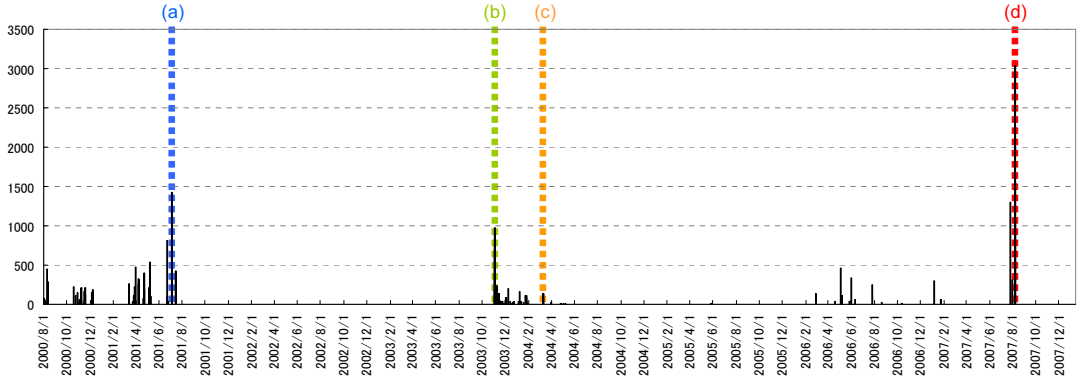
6.2.3 変更の変動度

3.2 節で述べたように，変更の変動度として用いる指標は $FCT(DH), FCT(DE), FCT(DM)$ の 3 つである．

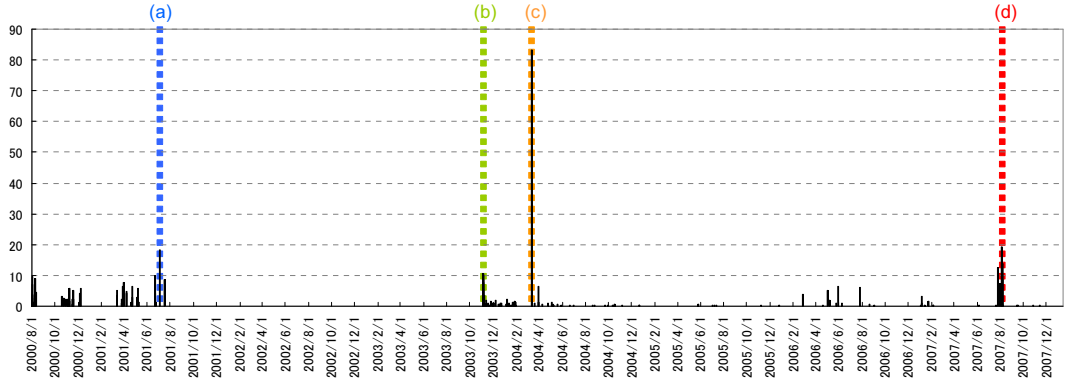
それぞれの指標のグラフを図 7 に示す．グラフの横軸は日時を表しており，グラフの縦軸は変更時刻におけるそれぞれの指標の値を表している．図 7 中の点線 (a), (b), (c), (d) で示した箇所は，全体的に変動度が高い変更を表している．



(1) $FCT(DH)$



(2) $FCT(DE)$



(3) $FCT(DM)$

図 7: FreeMind における変更の変動度

6.2.4 （モジュール，メトリクス）の変動度

3.2 節で述べたように,（モジュール，メトリクス）の変動度として用いる指標は $H, H', Q, Q', FMDMT(CV)$ の 5 つである．

モジュールの変動度の項にて挙げた，モジュールの変動度の大きい 4 つのモジュール `freemind.controller.Controller` クラス，`freemind.modes.MapAdapter` クラス，`freemind.modes.mindmapmode.MindMapMapModel` クラス，`freemind.view.mindmapview.NodeView` クラスにおけるそれぞれの指標のグラフを図 8 に示す．グラフの横軸は各メトリクスを表しており，グラフの縦軸はそれぞれの指標の値を表している．

6.2.5 （モジュール，変更）の変動度

3.2 節で述べたように,（モジュール，変更）の変動度として用いる指標は DH, DE, DH の 3 つである．

モジュールの変動度の項にて挙げた，モジュールの変動度の大きい 4 つのモジュールの内 1 つ，`freemind.modes.MapAdapter` クラスにおけるそれぞれの指標のグラフを図 9 に示す．グラフの横軸は日時を表しており，グラフの縦軸はそれぞれの指標の値を表している．また，図 9 中の点線 (a), (b), (c), (d) は変更の変動度の項で挙げた図 7 における点線と同じものである．

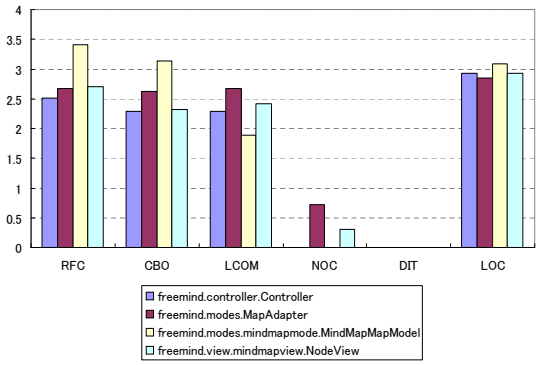
6.2.6 （モジュール，メトリクス，変更）の変動度

3.2 節で述べたように,（モジュール，メトリクス，変更）の変動度として用いる指標は CV の 1 つである．

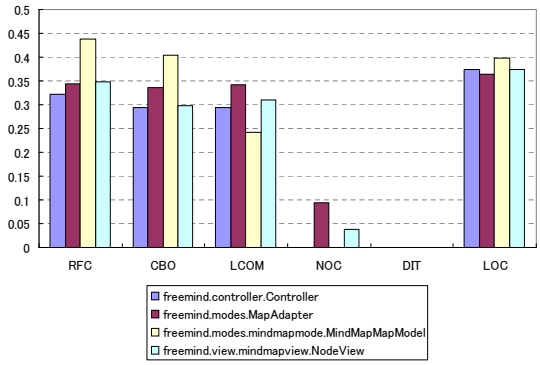
モジュールの変動度の項にて挙げた，モジュールの変動度の大きい 4 つのモジュールの内 1 つ，`freemind.modes.MapAdapter` クラスにおける各メトリクスのグラフを図 10 に示す．グラフの横軸は日時を表しており，グラフの縦軸はそれぞれのメトリクスにおけるメトリクス値の変化量を表している．また，図 10 中の点線 (a), (b), (c), (d) は変更の変動度の項で挙げた図 7 や（モジュール，変更）の変動度の項で挙げた図 9 における点線と同じものである．

6.3 考察

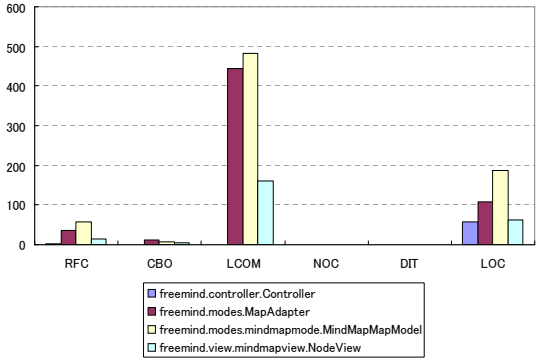
ここでは 6.2 節で記載した適用結果に対し，それぞれの変動度毎にどのようなソフトウェアの特性が分析できるかについて考察を行う．



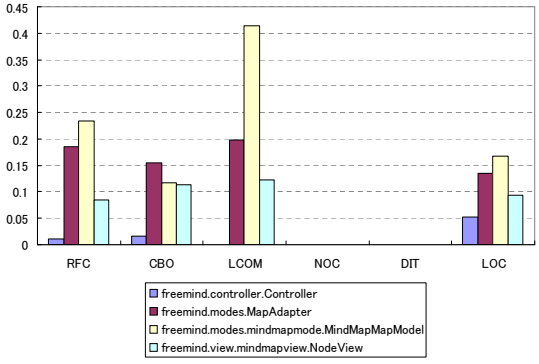
(1) H



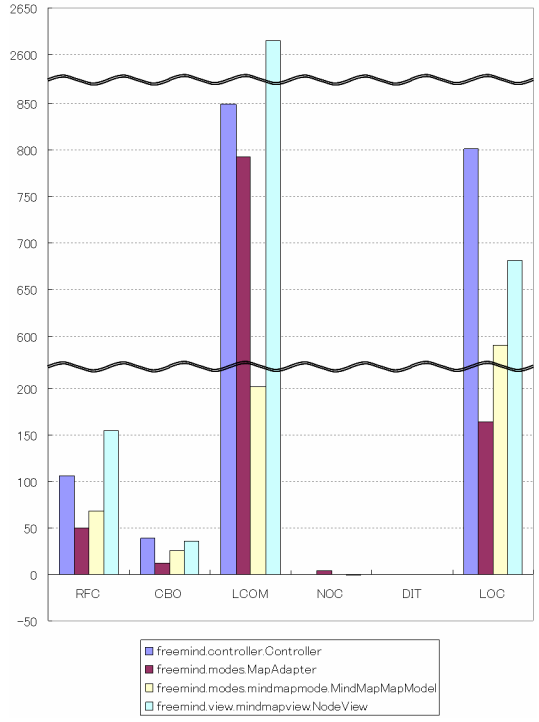
(2) H'



(3) Q

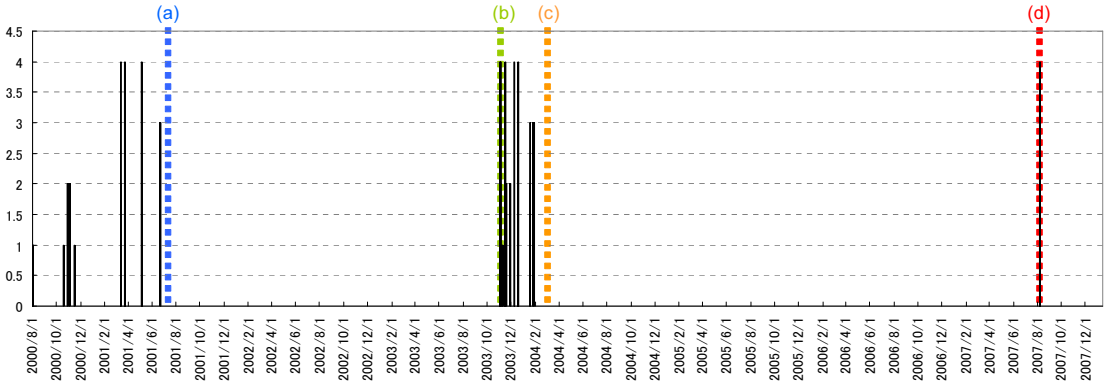


(4) Q'

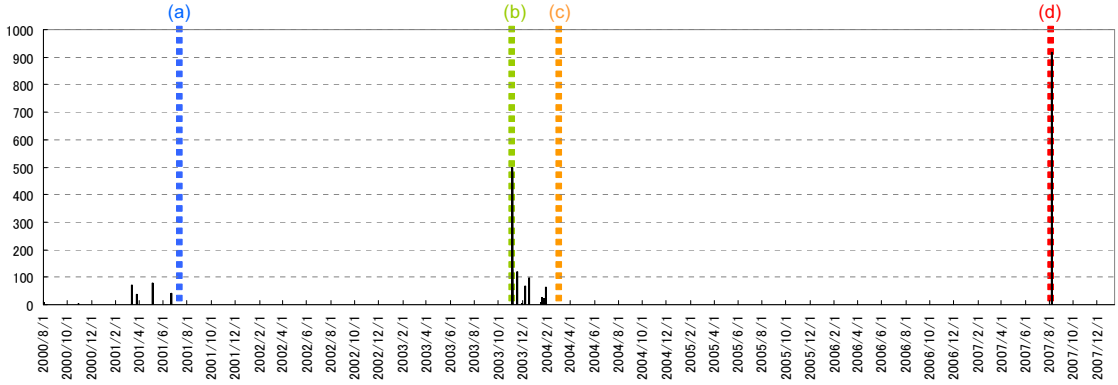


(5) $FMDMT(CV)$

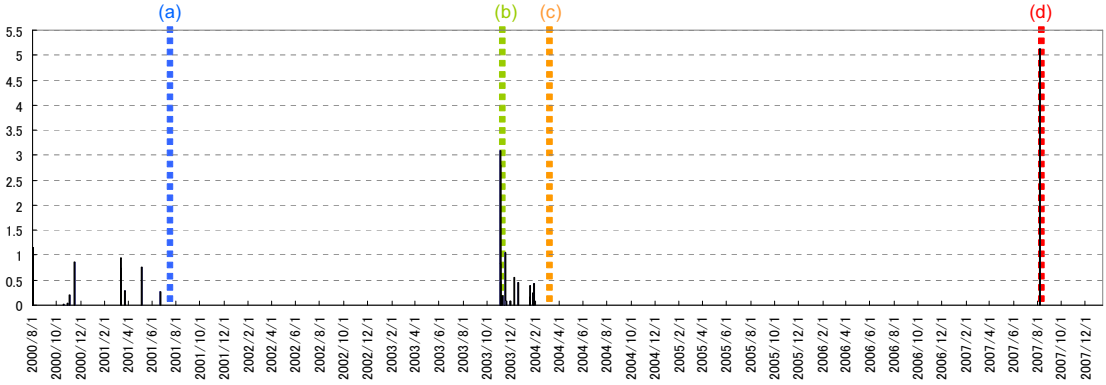
図 8: FreeMind の 4 つのクラスにおける（モジュール，メトリクス）の変動度



(1) DH



(2) DE



(3) DM

図 9: `freemind.modes.MapAdapter` クラスにおける（モジュール，変更）の変動度

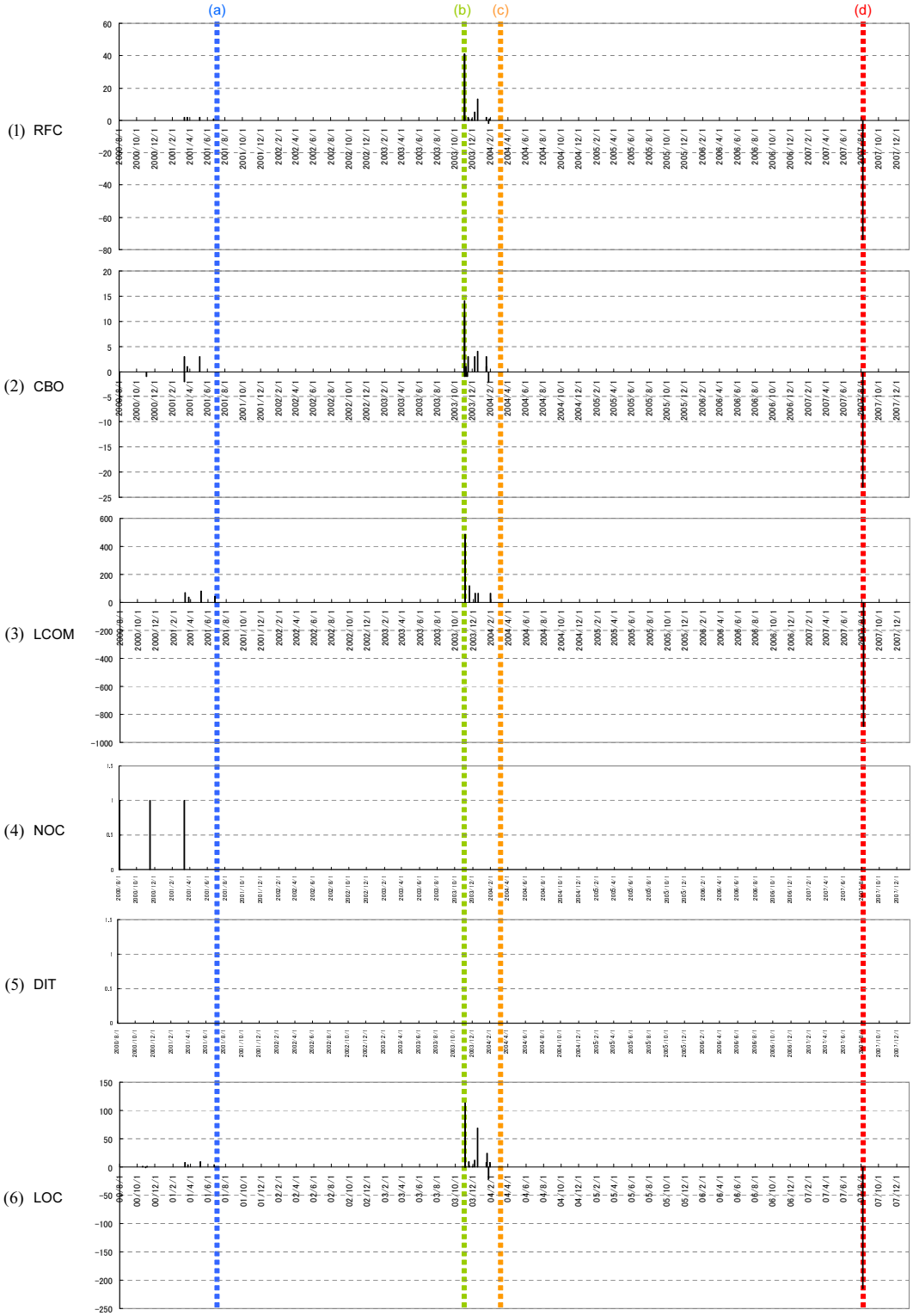


図 10: `freemind.modes.MapAdapter` クラスにおける
（モジュール，メトリクス，変更）の変動度

6.3.1 モジュールの変動度

図 5 を見ると，各指標毎のスケール差はあるものの，全体的にグラフの形状が似ていることが分かる．特に，図 5(1) の $FMD(H)$ と図 5(2) の $FMD(H')$，図 5(4) の $FMD(DH)$ と図 5(5) の $FMD(DE)$ と図 5(6) の $FMD(DM)$ は非常に良く似ている．すなわち，どの種類の指標を用いても変動度の高いモジュールはそれほど大きく異ならないと言える．

図 5 において特に変動度の大きい 4 つのモジュールが点線 (a), (b), (c), (d) で記されている．(a) の点線が示すクラスは `freemind.controller.Controller` クラス，(b) は `freemind.modes.MapAdapter` クラス，(c) は `freemind.modes.mindmapmode.MindMapMapModel` クラス，(d) は `freemind.view.mindmapview.NodeView` クラスとなっている．これらのクラスを始めとし，変動度の高いクラスに着目して開発を進めれば，効率的な開発になると期待できる．

変動度の高いモジュールに力を割くことが効率的な開発に繋がるということを確かめるため，変動度の高いモジュールとバグとの関係を調べる実験を行った．実験は次に示す手順で実行した．以降では，バグを修正するための変更を**バグ修正変更**，バグ修正変更の対象となったファイルの延べ数を**バグ修正数**と呼ぶ．

1. FreeMind の変更時刻の集合 $CT = \{ct_1, ct2, \cdots, ct_{225}\}$ を前 3 分の 1$\{ct_1, ct_2, \cdots, ct_{75}\}$ と後 3 分の 2$\{ct_{76}, ct_{77}, \cdots, ct_{225}\}$ に分割する．

2. CT の前 3 分の 1$\{ct_1, ct_2, \cdots, ct_{75}\}$ を用いてモジュール（クラス）の変動度を計測する．このとき，内部クラス（クラス内で宣言されているクラス）は除く．これにより，ソースファイルとクラスが 1 対 1 の関係になる．

3. 変更履歴情報に含まれる変更を行った開発者によるメッセージログを用いて，CT の後 3 分の 2$\{ct_{76}, ct_{77}, \cdots, ct_{225}\}$ におけるバグ修正変更の時刻を特定する．

4. 変動度の高いクラスのランキングによる，バグ修正数に対する被覆率を求める．

FreeMind の開発においては，バグ修正変更の際のメッセージログには，その先頭に「Bug fix: 」と記載する習慣がある．したがって上記の手順 3 においては，変更を行った開発者によるメッセージログに「bug」と「fix」の文字列（大文字小文字は区別しない）が含まれているかどうかでバグ修正変更の時刻を特定する．このようにして特定されたバグ修正数は 36 となった．

変動度の高いクラスのランキングによる，バグ修正数に対する被覆率を図 11 に示す．図 11 を見れば，どのモジュールの変動度の指標を用いても，ランキングの上位 20%（図 11 の点線で示された箇所）におよそ 97% 〜100% もの後のバグ修正変更が含まれていることが分

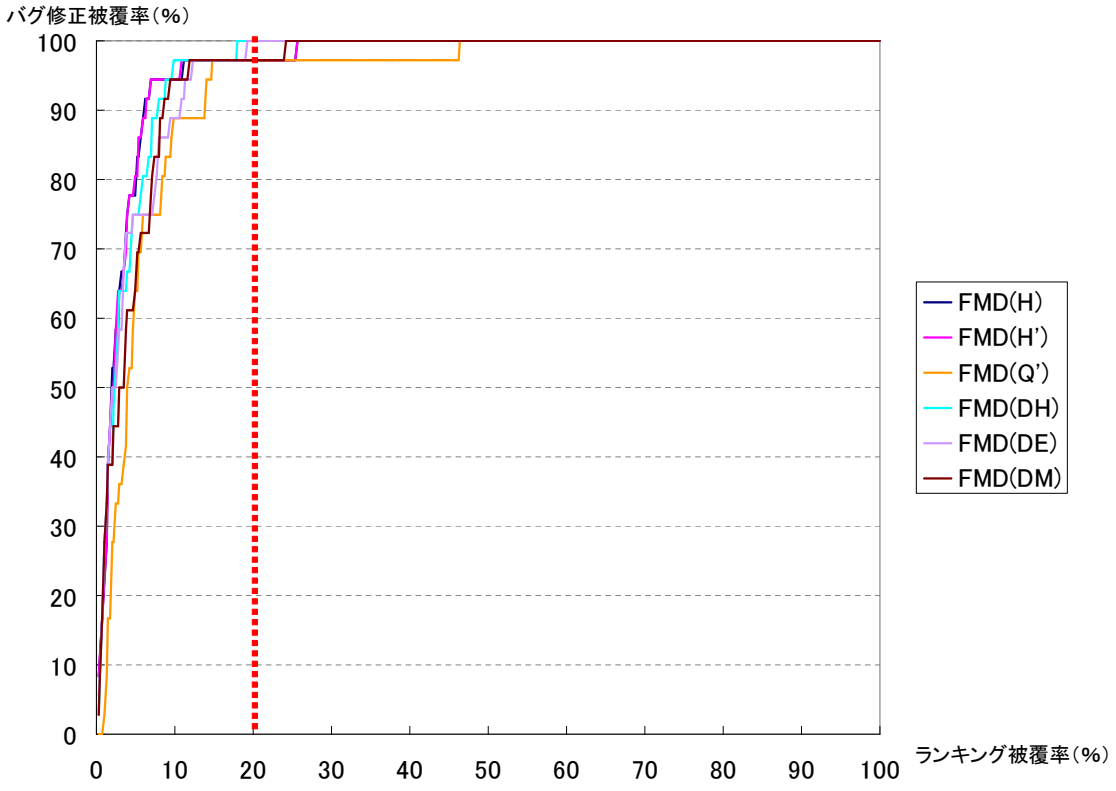


図 11: FreeMind におけるモジュールの変動度のランキングによるバグ修正数の被覆率

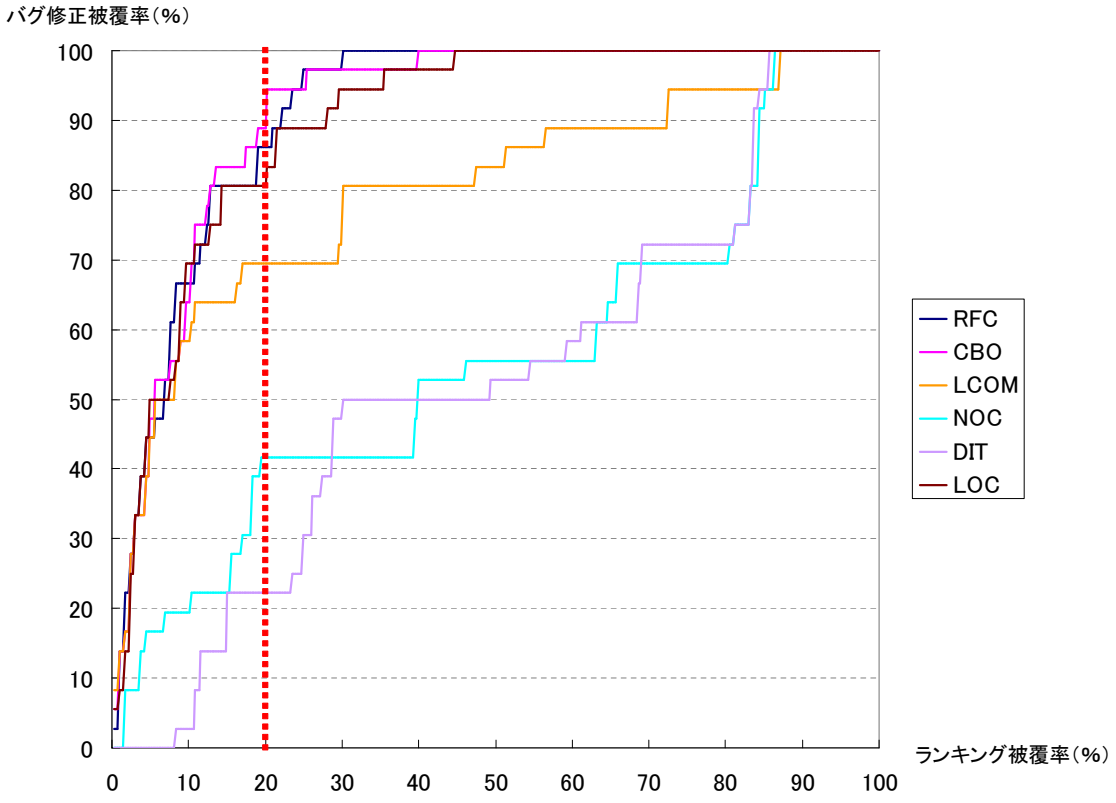


図 12: FreeMind におけるメトリクス値のランキングによるバグ修正数の被覆率

かる．各指標のグラフの形状が似ていることは，図 5 に示した棒グラフの形状が似ているためであると考えられる．

さらに，このモジュールの変動度の比較対象として，変更の前 3 分の 1$\{ct_1, ct_2, \cdots, ct_{75}\}$ の最後の変更時刻 ct_{75} におけるスナップショットに対するメトリクス $RFC, CBO, LCOM, NOC, DIT, LOC$ の値を求め，各メトリクス値のランキングによるバグ修正数の被覆率を求める．用いるメトリクスの内，LOC を除く CK メトリクス（$RFC, CBO, LCOM, NOC, DIT$）については，特にバグの予測に有効であるとの報告もなされている [8].

このようにして求めたメトリクス値のランキングによるバグ修正数の被覆率を図 12 に示す．図 12 を見ると，メトリクスの種類によってバグ修正数の被覆率が大きく異なることが分かる．ランキングの上位 20%（図 12 の点線で示された箇所）では，最大 89% 程の被覆率を示すが，最小では 22% 程度の被覆率に留まっている．しかし，ランダムにモジュールを選択したときのバグ修正数の被覆率の期待値がランキングの被覆率と等しくなることを考えれば，確かにメトリクスを用いたバグの予測はある程度有効であると言える．

図 11 と図 12 を比較する上で 1 つ問題がある．図 12 の各メトリクスの粒度（次元）は（モ

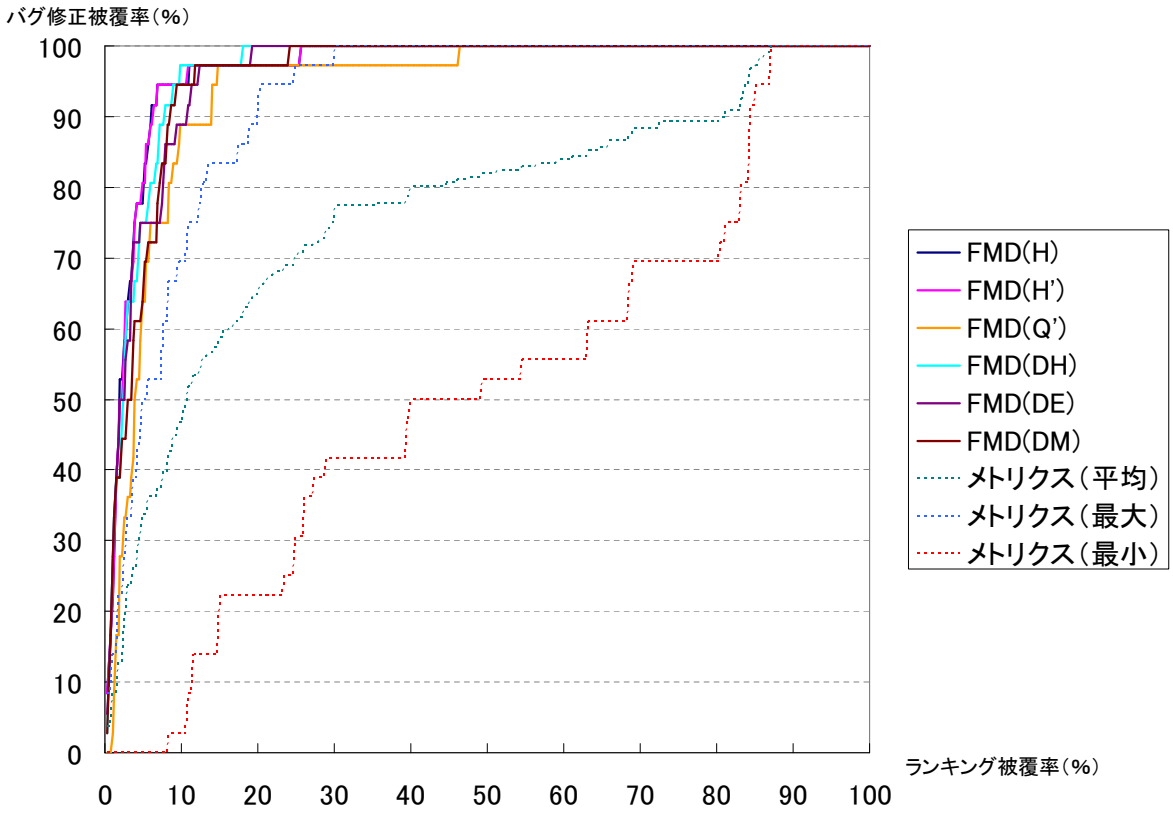


図 13: 図 11 と図 12 の比較

ジュール，メトリクス）の変動度と対応したものであるため，モジュールの変動度である図 11 と図 12 における個々のメトリクスを直接比較することは適当でない．したがって，メトリクス値のランキングによるバグ修正数の被覆率の平均値，最大値，および最小値を用いて比較を行う．図 11 のグラフにメトリクス値のランキングによるバグ修正数の被覆率の平均値，最大値，および最小値のグラフを加えたものを図 13 に示す．

図 13 から，次のことが言える．

- モジュールの変動度のどの指標を用いても，後のバグ修正の予測に極めて有効であり，また指標毎の差異もそれほど大きくはない．
- メトリクス値を用いた後のバグ修正の予測は有効であるが，用いるメトリクスによる差異が極めて大きい．
- 各メトリクスを用いた場合の後のバグ修正の予測効果の平均より，モジュールの変動度を用いた後のバグ修正の予測の方がより有効である．

- 後のバグ修正を最も有効に予測できるメトリクスを用いた場合でも，モジュールの変動度を用いた後のバグ修正の予測の方が一般に有効である．

以上のことより，ソフトウェアの開発においてモジュールの変動度が高いモジュールに力を割くことは，効率的な開発の実現に繋がると大きく期待できる．

6.3.2 メトリクスの変動度

図 6 からは様々なソフトウェアの特性が読み取れる．

例えば，図 6(1) の $FMT(H)$，図 6(2) の $FMT(H')$ のグラフより，メトリクス $RFC, LOC, CBO, LCOM, DIT, NOC$ の順でメトリクスの値が頻繁に変化していることが分かる．ソースコードの行数を表す LOC の値は，そのモジュール自体に変更が行われたときに高い確率で変化する．ソフトウェアにおけるモジュール間の関係の強さを表す RFC が LOC よりも頻繁に変化しているということは，対象のモジュール以外のモジュールが変更されたときにも値が変化しているということであり，FreeMind におけるモジュール間の独立性が弱いことを示唆していると考えられる．

また，図 6(1) の $FMT(H)$，図 6(2) の $FMT(H')$，図 6(3) の $FMT(Q)$ のグラフにおける $LCOM$ の変動度を比較すれば，$FMT(H), FMT(H')$ のグラフでは RFC, LOC, CBO の変動度より小さい値であったのに，$FMT(Q)$ のグラフでは $LCOM$ の変動度が最も大きくなっていることが分かる．エントロピー H や正規化エントロピー H' では値が変化したかどうかしか影響せず，四分位偏差 Q では変化した値の大きさも影響することを考慮すると，$LCOM$ は RFC, LOC, CBO に比べると値の変化は頻繁でないが，値が変化するときは大きく値が変化するということを表していると言える．

なお，図 6(4) の $FMT(Q')$ のグラフにおいて $RFC, CBO, LCOM$ の変動度が ∞ となっているのは，あるモジュールにおける半分以上の変更時にそれぞれのメトリクス値が 0 であったことを意味する．このような場合，個々のモジュールの持つ変動度が $FMT(Q)$ のグラフに反映されないので，図 6(3) の $FMT(Q)$ のグラフを代わりに利用するか，図 8(4) のような（モジュール，メトリクス）の変動度における個々のモジュールごとの四分位偏差 Q' のグラフを参考にしたほうがよい．

6.3.3 変更の変動度

図 7 から，いつ行われた変更がどの程度ソフトウェア FreeMind に影響を与えたのかというソフトウェアの特性を知ることができる．この特性を知ることは，対象ソフトウェアの開発過程の理解に役立つ．

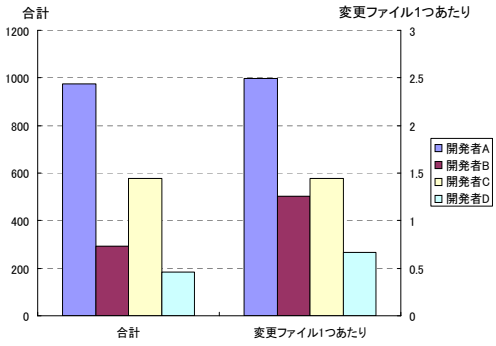
例えば，図 7 中の点線 (a), (b), (c), (d) で示した全体的に変動度の高い箇所ではソフトウェア全体に大きく影響するような変更が行われたと考えられる．実際にそれぞれの変更の内容を確認してみると，点線 (a) における変更では Ver. 0.4 のリリースに向けての多量のバグ修正が行われ，(b) においては Ver. 0.5 から Ver. 0.6 へ移行するために数多くのソースファイルが変更されていた．また，点線 (c) における変更では Ver. 0.8 に実装する新しい機能を持った様々なプラグインが追加され，(d) では Ver. 0.9 に実装する新しい機能を追加すると同時に不要なソースコードの削除や統合などの変更が行われていた．

さらに，図 7 中の点線 (a) における変更からしばらくたった後，点線 (b) で示される変更までのおよそ 2 年間，全てのグラフにおいて変動度が 0 になっている．つまり，この期間は開発を休止していたと考えられる．実際に開発の履歴を詳しく調査してみると，確かにこの期間では一切の開発を行っていなかったことが分かった．約 2 年にわたる開発の空白期間の影響として，図 7 中の点線 (b) と (c) の間に行われた変更が挙げられる．多くのバグ修正変更が点線 (b) と (c) の間に存在している．これは，もちろん (b) における大規模な変更の影響も強いであろうが，開発を長期間にわたって行わなかったために FreeMind というソフトウェアに対する理解度が低下し，その結果スムーズな変更ができなかったという要因も十分に考えられる．

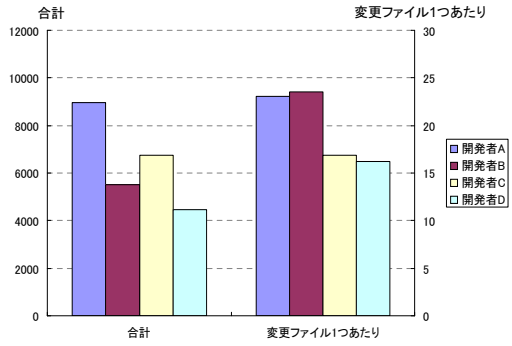
一方，変更の変動度を変更を行った開発者別に分け，その合計値を見ることで，各開発者による変更がどの程度ソフトウェアに影響を与えたのか知ることができる．FreeMind において，ソースコードの変更を担当する開発者は 4 人存在する．この 4 人の開発者 A, B, C, D について，変更を行った回数とその変更の対象になったファイル数の合計を表 8 に示す．表 8 を見ると，開発者によって変更する頻度や一度に変更するファイルの数が異なることが分かる．変更したファイル数が多いほど変更の変動度の合計は大きくなりやすいので，変更の変動度の合計と共に，その合計を各変更の対象となったファイル数の合計で割った値のグラフを図 14 に示す．図 14 において，左側の縦軸はそれぞれの開発者についての変更の変動度の合計値を表し，右側の縦軸は左側の縦軸が示す値を各開発者の変更の対象となったファイル数の合計で割った値を表す．図 14 のグラフは，それぞれの開発者の能力や担当した仕事

表 8: FreeMind における各開発者の変更回数と総変更対象ファイル数

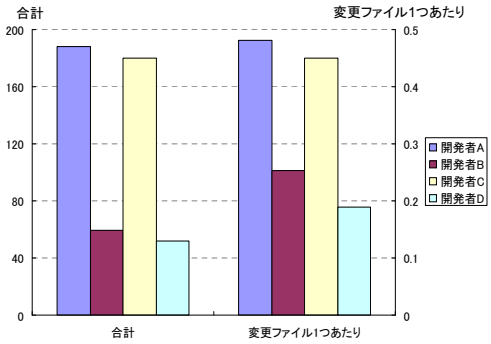
開発者	変更回数	総変更対象ファイル数
開発者 A	51	390
開発者 B	11	129
開発者 C	130	400
開発者 D	33	275



(1) 開発者別 $FCT(DH)$



(2) 開発者別 $FCT(DE)$



(3) 開発者別 $FCT(DM)$

図 14: FreeMind におけるメトリクスの変動度

の難しさや分量を知る手がかりになる．例えば，それぞれの開発者に割り振られた仕事の分量や難しさが同程度だと仮定するなら，図 14 を見る限りでは開発者 D が最も効率よく仕事をこなしていると判断することができよう．

6.3.4 （モジュール，メトリクス）の変動度

図 8(1) のエントロピー H，図 8(2) の正規化エントロピー H'，図 8(3) の四分位偏差 Q，図 8(4) の四分位分散係数 Q' のグラフは，それぞれメトリクスの変動度の項で示した図 6(1)，図 6(2)，図 6(3)，図 6(4) における各グラフのモジュール別の詳細を表していると言える．

また，図 8(3) の四分位偏差 Q のグラフと図 8(5) の $FMDMT(CV)$ のグラフを比較すれば，Q のグラフでは変動度の小さかった `freemind.controller.Controller` クラスと `freemind.view.mindmapview.NodeView` クラスが，$FMDMT(CV)$ のグラフでは大きな変動度を示していることがわかる．四分位偏差では例外的な値が影響しにくいという特徴を考慮すると，このことから，この 2 つのクラスはある変更で大きくメトリクス値が変化したことがあったものの，それ以外の変更ではあまりメトリクス値が変化しなかったという知見を得ることができる．

6.3.5 （モジュール，変更）の変動度

図 9 中の点線 (a), (b), (c), (d) は変更の変動度の項で挙げた図 7 における点線と同じものを表している．すなわち，点線 (a) では Ver. 0.4 のリリースに向けての多量のバグ修正が，(b) では Ver. 0.5 から Ver. 0.6 へ移行するための複数のソースファイルに対する変更が，(c) では Ver. 0.8 に実装する新しい機能を持った様々なプラグインの追加が，そして (d) では Ver. 0.9 に実装する新しい機能の追加と不要なソースコードの削除や統合が行われている．図 9 を見ると，`freemind.modes.MapAdapter` クラスは点線 (b) 及び (d) における変更とは深い関わりを持つが，点線 (a) 及び (c) における変更とはほとんど関わりを持たなかったことが分かる．

6.3.6 （モジュール，メトリクス，変更）の変動度

図 10 は,（モジュール，変更）の変動度の項で挙げた図 9 のような（モジュール，変更）の変動度をメトリクス別に詳細化したものと捉えることができる．また，図 10 中の点線 (a), (b), (c), (d) は変更の変動度の項で挙げた図 7 や（モジュール，変更）の変動度の項で挙げた図 9 における点線と同じものを表している．すなわち，点線 (a) では Ver. 0.4 のリリースに向けての多量のバグ修正が，(b) では Ver. 0.5 から Ver. 0.6 へ移行するための複数のソースファイルに対する変更が，(c) では Ver. 0.8 に実装する新しい機能を持った様々なプラグ

インの追加が，そして (d) では Ver. 0.9 に実装する新しい機能の追加と不要なソースコードの削除や統合が行われている．図 10 を見ると，`freemind.modes.MapAdapter` クラスは点線 (b) 及び (d) における変更と関わりを持つことが分かるが，(b) における変更ではメトリクス値が上昇し，(d) における変更ではメトリクス値が減少している．このことから，(b) における Ver. 0.5 から Ver. 0.6 への移行により `freemind.modes.MapAdapter` クラスの複雑さは上昇し，(d) における Ver. 0.9 に向けての不要なソースコードの削除や統合の結果 `freemind.modes.MapAdapter` クラスの複雑さは減少したということを知ることができる．

6.3.7 総括

ここまで，FreeMind にシステムを適用した結果，得られるモジュールの変動度，メトリクスの変動度，変更の変動度,（モジュール，メトリクス）の変動度,（モジュール，変更）の変動度,（モジュール，メトリクス，変更）の変動度について詳しく述べた．

それぞれの変動度から有益な情報が得られ，またいくつかの変動度から得られる情報についてはその有効性を実験により確かめることができた．以上の事柄から，本手法により得られる情報は，FreeMind というソフトウェアの効率的な開発や保守，およびそれらの管理に役立つと充分に期待できる．

7　関連研究

ソフトウェアの特性を得るための研究は数多く行われている．本節では，本研究の関連としてソフトウェアメトリクスを用いた研究とバージョン管理システムを利用した研究をいくつか紹介する．以下では本研究で提案した手法を本提案手法と呼ぶ.

7.1　ソフトウェアメトリクスを用いた研究

3.1.4 節で述べたように，ソフトウェアメトリクスもソフトウェアの特性を表すものである.

6 節の適用事例において用いた CK メトリクスは，Chidamber と Kemerer[7] によって生み出されたオブジェクト指向プログラムを対象としたメトリクスである．Basili ら [8] は，8 つの学生チームによるオブジェクト指向プログラミング言語 C++を用いて開発された情報管理システムを対象とし，CK メトリクスと他のメトリクスとの比較実験を行った．その実験結果から，他のメトリクスに比べ，CK メトリクスはより多くのソフトウェアの欠陥を，よりソフトウェア開発の早期において予測できることを示した．また，Subramanyam と Krishnan[9] は，産業界で開発された 8 つ以上のソフトウェアに対して CK メトリクスの有効性を調査した．その結果，Basili らと同様，CK メトリクスとソフトウェアの欠陥に大きな関係があることを見出した．同時に，ソフトウェアによってメトリクスの有効性が異なることも示した.

Nagappan ら [26] はソフトウェアのバグを予測する最も良いメトリクスがソフトウェアによって異なることを発見し，対象ソフトウェアにおける過去のバグに関する情報からソフトウェアメトリクスによってバグを予測する回帰モデルを構築した．彼らはいくつかの大規模な商用ソフトウェアに手法を適用し，その有効性を評価している.

CK メトリクスと Nagappan らによる手法，及び本提案手法との比較を表 9 に示す．表 9 で用いている項目の意味はそれぞれ以下に示す通りである.

- 手法
  手法の名称，あるいは手法の提案者を表す.
- 情報源

表 9: CK メトリクス，Nagappan らによる手法との比較

手法	情報源	ソフトウェアの特性が得られる対象
CK メトリクス	ソースコード	モジュール
Nagappan ら	ソースコード，バグ情報	モジュール
本提案手法	バージョン管理システム	モジュール，メトリクス，変更

手法の適用に必要な情報源を表す.

- ソフトウェアの特性が得られる対象
  手法によって得られるソフトウェアの特性が何に対して得られるのかを表す.

本提案手法とこれらの手法との違いは，情報源が異なること，そして得られるソフトウェアの特性の対象が異なることである．Nagappan らの手法はバグに関する情報が管理されている必要があるが，正確にバグに関する情報を管理しているソフトウェアが少ないのが問題となっている．適用可能なソフトウェアが多いという点では，ソースコードのみを必要とする CK メトリクスが最も優れていると言える．しかし，近年バージョン管理システムを用いているソフトウェア開発は非常に多いため，本提案手法が適用できるソフトウェアも非常に多い．また，6 節の適用事例で述べたように，本提案手法は CK メトリクスよりも有効なバグ修正の予測が可能であることを実験で確かめた．さらに，本提案手法は他の手法と異なり，モジュールに関するソフトウェアの特性以外についての情報も得られるのが利点となっている．

7.2 バージョン管理システムを用いた研究

近年ではバージョン管理システムなどの開発履歴に関する情報を利用してソフトウェアの特性を把握する試みも増えている．

Baven ら [39] はバージョン管理システムのリポジトリを解析することで同時に変更が行われたモジュール群を特定している．このように同時に変更されることが多いモジュール群は将来においても頻繁に変更されることが予想されるので，このモジュール群を開発者に示すことにより，将来の保守効率の向上が期待できるとしている．

また，Hassan と Holt[25] は情報理論 [27] によるエントロピーをソフトウェアの開発過程に当てはめ，ソフトウェアの開発履歴に含まれる情報量の計測を試みた．彼らはこの情報によりソフトウェアの開発過程の監視が行えるとし，特にプロジェクトマネージャにとって有益な手法であることを述べた．

Śliwerski ら [34] はバージョン管理システムと過去のバグに関する情報とを関連付け，バグを発生させる原因となった変更の特定を行っている．このような変更が頻繁に行われるモジュールは，今後もバグを発生させやすいと考えられる．開発者このモジュールを提示することで，将来の開発や保守を効率的に行うことが望めるとしている．

花川 [40] はソフトウェアメトリクスを用いたモジュール間の結合度と論理的なモジュール間の結合，すなわち同時に変更が施されるモジュール群に着目し，新たな複雑度の指標を定義した．また，ソフトウェアの開発過程におけるこの複雑度の変化を可視化し，ソフトウェアの複雑さの変化を視覚的に把握するツールを作成している．

Vasa ら [41] はソフトウェアの開発過程におけるリリース間でのモジュール毎の利用数，被利用数の変化を調査している．彼らは複数のオープンソースソフトウェアに対して実験を行い，利用数，被利用数の変化から得られるソフトウェアの特性の取得を試みた．その結果，ほとんどのモジュールは開発が経過するにつれ利用数，被利用数の変化が見られなくなってくるが，利用数，被利用数が非常に多い一部のモジュールは開発が経過しても利用数，被利用数が変化し続ける，などといったソフトウェアの特性を分析した．

これらの手法と本提案手法との比較を表 10 に示す．表 10 で用いている項目の意味は 7.1 節の表 9 と同様である．

本提案手法は，Baven らによる手法，Hassan と Holt による手法，Śliwerski らによる手法に比べ，モジュールについてだけでなく様々なソフトウェアの特性を得られる点で異なる．また，Śliwerski らによる手法はバグに関する情報が管理されているソフトウェアを対象とするため，手法を適用できるソフトウェアが少ないのが問題となっている．花川による手法と Vasa らによる手法は，本提案手法と同じ情報源を用い，得られるソフトウェアの特性の対象も本提案手法に近い．しかし，花川による手法ではソフトウェアの特性の対象となるメトリクスは結合度のみであり，Vasa らによる手法でも対象となるメトリクスは利用数，被利用数という一種の結合度のみである．本提案手法では，適用事例においては CK メトリクスを中心としたメトリクスを用いたが，手法としてはソースコードを対象とする定量的なメトリクスであればどんなメトリクスであっても用いることができる．また，本提案手法ではメトリクス値の変遷を変動度という数値として出力する点も既存の研究には見られなかった特長である．

表 10: バージョン管理システムを用いた手法との比較

手法	情報源	ソフトウェアの特性が得られる対象
Baven ら	バージョン管理システム	モジュール
Hassan, Holt	バージョン管理システム	モジュール
Śliwerski ら	バージョン管理システム，バグ情報	モジュール
花川	バージョン管理システム	モジュール，メトリクス（結合度），変更
Vasa ら	バージョン管理システム	モジュール，メトリクス（結合度），変更
本提案手法	バージョン管理システム	モジュール，メトリクス，変更

8　まとめ

本稿では，バージョン管理システムのリポジトリに蓄積された開発履歴情報を用いてメトリクス値の変遷を求め，以下の 6 種類の変動度を導出することによりソフトウェアの特性を分析する手法を提案した．

- モジュールの変動度
- メトリクスの変動度
- 変更の変動度
- （モジュール，メトリクス）の変動度
- （モジュール，変更）の変動度
- （モジュール，メトリクス，変更）の変動度

さらに，提案手法を実装し，実存する複数のソフトウェアに対して手法を適用した．その結果，ソフトウェアの開発や保守を行うにあたり有益なソフトウェアの特性を本手法によって得られることが確認できた．

今後の課題として重要であるのは，手法を実装したツールの拡張である．現状では制限が多く，以下にあげるような機能が望まれる．

- C++や C# など，他のプログラミング言語への対応
- Subversion など，他のバージョン管理システムへの対応
- 他の様々なソフトウェアメトリクスへの対応
- 変動度を可視化する作業の自動化

謝辞

本研究の全過程を通して，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本研究を通して，随時適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授に深く感謝いたします.

本研究において，適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾 隆 助教に深く感謝いたします.

本研究を通して，終始適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後 芳樹 助教に心より深く感謝いたします.

本研究に対して，適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 早瀬 康裕 特任助教に深く感謝いたします.

最後に，その他様々な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 ソフトウェア工学講座の皆様に深謝いたします.

参考文献

[1] Brian Berliner. CVS II:Parallelizing Software Development. In *USENIX Association, editor, Proceedings of the Winter 1990 USENIX Conference*, pp.341-352, Berkeley, CA, USA, 1990.

[2] Karl Fogel. Open Source Development with CVS. The Coriolis Group, 2000.

[3] 鯉江英隆, 西本卓也, 馬場肇. バージョン管理システム (CVS) の導入と活用. SOFT BANK, Dec 2000.

[4] Version Control with Subversion.
Available online at <http://svnbook.red-bean.com/>

[5] Peter H. Feiler. Configuration Management Models in Commercial Environments. CMU/SEI-91-TR-7 ESD-9-TR-7, 1991.

[6] Thomas J. McCabe. A complexity measure. In *Proceedings of the 2nd International Conference on Software Engineering (ICSE '76)*, p.407, Los Alamitos, CA, USA, 1976.

[7] Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. In *IEEE Transactions on Software Engineering*, Vol.20, No.6, pp.476-493, 1994.

[8] Victor R. Basili, Lionel C. Briand, and Walcelio L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. In *IEEE Transactions on Software Engineering*, Vol.22, No.10, pp.751-761, 1996.

[9] Ramanath Subramanyam and M.S. Krishnan. Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. In *IEEE Transactions on Software Engineering*, Vol.29, No.4, pp.297-310, Apr 2003.

[10] Jacky Estublier. Software Configuration Management: A Roadmap, In *The Future of Software Engineering in 22nd ICSE*, pp.281-289, 2000.

[11] Ulf Asklund, Lars Bendix, Henrik B. Christensen, and Boris Magnusson. The Unified Extensional Versioning Model, In *9th International Symposium*, pp.100-122, 1999.

[12] Reidar Conradi and Berbard Westfechtel. Version models for software configuration management, In *ACM Computing Surveys*, Vol.30, No.2, pp.232-280, Jun 1998.

[13] Walter F. Tichy. RCS - A System for Version Control. In *SOFTWARE - PRACTICE AND EXPERIENCE*, Vol.15, No.7, pp.637-654, 1985.

[14] Rational Software Corporation, Software configuration management and effective team development with Rational ClearCase.
Available online at <http://www.rational.com/products/clearcase/>

[15] Microsoft Corporation, Microsoft Visual SourceSafe,
Available online at <http://msdn.microsoft.com/ssafe/>

[16] Merant, Inc., PVCS Homepage.
Available online at <http://www.merant.com/pvcs/>

[17] Peter Fröhlich and Woflgang Nejdl. WebRC Configuration Management for a Cooperation Tool. SCM-7, pp.175-185, 1997.

[18] The FreeBSD Project.
Available online at <http://www.freebsd.org/>

[19] The OpenBSD Project.
Available online at <http://www.openbsd.org/>

[20] Mark Lorenz and Jeff Kidd. Object-Oriented Software Metrics: A Practical Guide. Prentice Hall, 1994.

[21] Maurizio Pighin and Roberto Zamolo. A Predictive Metric Based on Discriminant Statistical Analysis. In *Proceedings of the 19th International Conference on Software Engineering*, Boston, Massachusetts, USA, pp.262-270, 1997.

[22] Maurice H. Halstead. Elements of Software Science. Elsevier Science Inc., New York, NY, USA, 1977.

[23] Lionel C. Briand, John Daly, Victor Porter, and Jürgen Wüst. Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems. In *Proceedings of the The Ninth International Symposium on Software Reliability Engineering (ISSRE '98)*, p.334, USA, 1998.

[24] Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer. Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. In *IEEE Transactions on Software Engineering*, Vol.24, No.8, pp.629-639, 1998.

[25] Ahmed E. Hassan and Richard C. Holt. Studying the Chaos of Code Development. In *Proceedings of 10th Working Conference on Reverse Engineering (WCRE '03)*, pp.123-133, 2003.

[26] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining Metrics to Predict Component Failures. In *Proceeding of the 28th International Conference on Software Engineering (ICSE '06)*, pp.452-461, 2006.

[27] Claud E. Shannon. The Mathematical Theory of Communication. In *Bell System Technical Journal*, Vol.27, pp.379-423 & 623-656, Jul & Oct 1948.

[28] 宮川 公男. 基本統計学. 有斐閣, 1999.

[29] 中村 幸四郎, 寺阪 英孝, 伊東 俊太郎, 池田 美恵 (翻訳). ユークリッド原論. 共立出版, 1996.

[30] Didier H. Besset. Object-Oriented Implementation of Numerical Methods: An Introduction With Java and Smalltalk. Morgan Kaufmann Pub., Oct 2003.

[31] Francoise Fessant, Patrice Aknin, Latifa Oukhellou, and Sophie Midenet. Comparison of Supervised Self-Organizing Maps Using Euclidian or Mahalanobis Distance in Classification Context. In *Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks (IWANN '01): Connectionist Models of Neurons, Learning Processes and Artificial Intelligence-Part I*, pp.637-644, 2001.

[32] Camilo P. Tenorio, Francisco de A. T. de Carvalho, and Julio T. Pimentel. A Partitioning Fuzzy Clustering Algorithm for Symbolic Interval Data based on Adaptive Mahalanobis Distances. In *Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS '07)*, pp.174-179, 2007.

[33] 村上 正憲, 松島 正直, 山田 博章. Mahalanobis' 特徴平面による識別. 電子情報通信学会総合大会講演論文集, Vol.1995 情報・システム, No.2, p.213, 1995.

[34] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. HATARI: Raising Risk Awareness. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC-FSE '05)*, Lisbon, Portugal, Sep 2005.

[35] SourceForge Homepage.
Available online at <http://sourceforge.net/>

[36] FreeMind Homepage.
Available online at <http://freemind.sourceforge.net/wiki/>

[37] JHotDraw Homepage.
Available online at <http://www.jhotdraw.org/>

[38] HelpSetMaker Homepage.
Available online at <http://www.cantamen.com/helpsetmaker.php>

[39] Jennifer Bevan, E. James Whitehead, Jr. Identification of Software Instabilities. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03)*, pp.134-143, Nov 2003.

[40] 花川 典子. 論理結合とモジュール結合による複雑度を用いたソフトウェア進化の可視化ツールの提案. 第 14 回ソフトウェア工学の基礎ワークショップ *(FOSE '07)*, pp.65-74, 2007.

[41] Rajesh Vasa, Jean-Guy Schneider, Oscar Nierstrasz. The Inevitable Stability of Software Change. In *IEEE International Conference on Software Maintenance 2007 (ICSM '07)*, pp.4-13, 2007.

付録

A　適用事例：JHotDraw

ここでは，6.1.2 節で挙げた手法の適用対象ソフトウェアである JHotDraw についての適用結果を掲載する．

掲載するのはモジュールの変動度，メトリクスの変動度，変更の変動度に関する適用結果である．(モジュール，メトリクス）の変動度，(モジュール，変更）の変動度，(モジュール，メトリクス，変更）の変動度については省略する．

A.1　モジュールの変動度

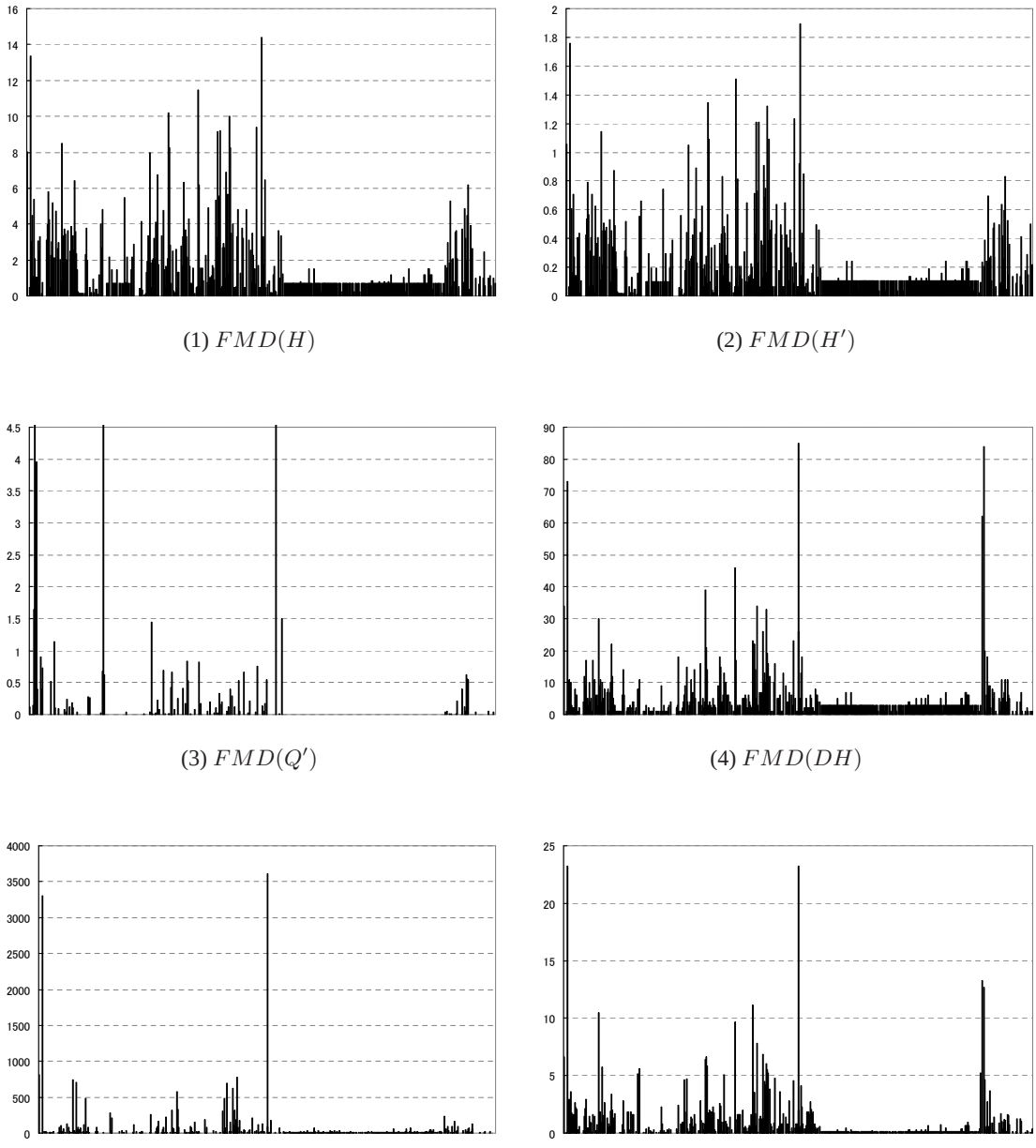
モジュールの変動度のグラフを図 15 に示す．グラフの横軸はモジュールであり，合計 569 クラスが全てのグラフにおいて同順で並んでいる．グラフの縦軸はそれぞれの指標の値を表している．

A.2　メトリクスの変動度

メトリクスの変動度のグラフを図 16 に示す．グラフの横軸はメトリクスであり，グラフの縦軸はそれぞれの指標の値を表している．

A.3　変更の変動度

変更の変動度のグラフを図 17 に示す．グラフの横軸は日時を表しており，グラフの縦軸は変更時刻におけるそれぞれの指標の値を表している．

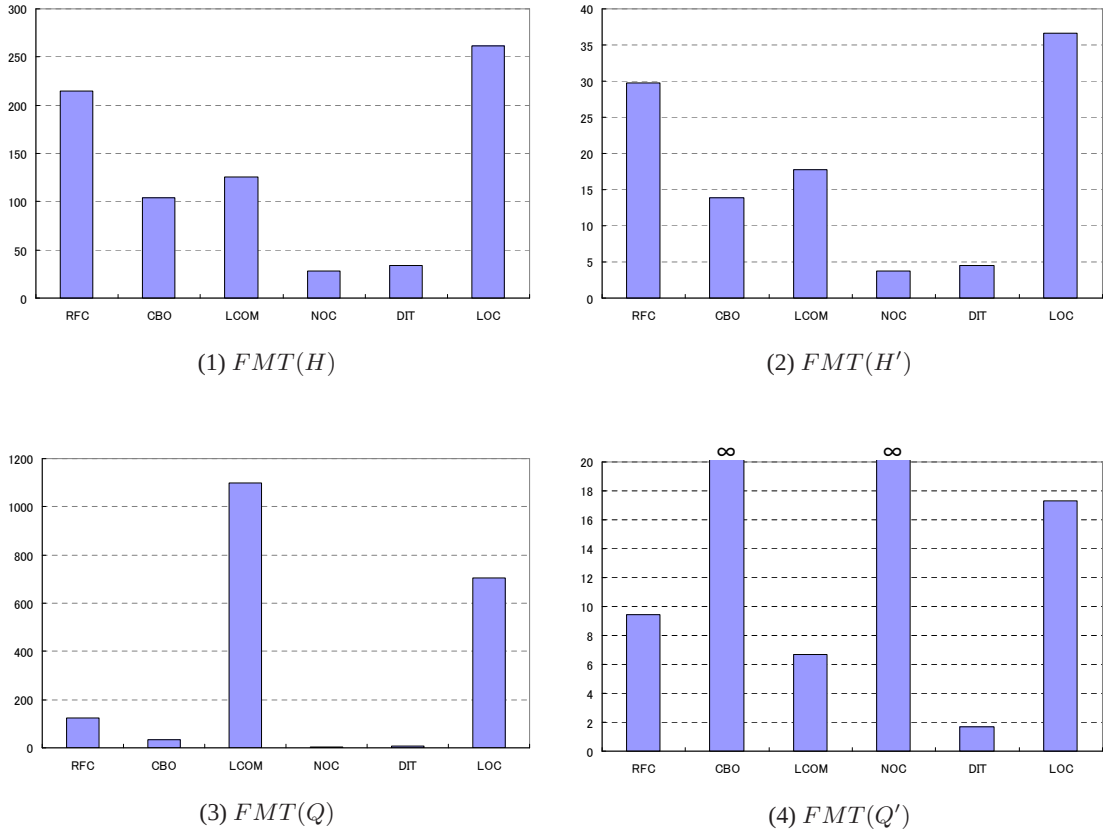


(1) $FMD(H)$ (2) $FMD(H')$

(3) $FMD(Q')$ (4) $FMD(DH)$

(5) $FMD(DE)$ (6) $FMD(DM)$

図 15: JHotDraw におけるモジュールの変動度



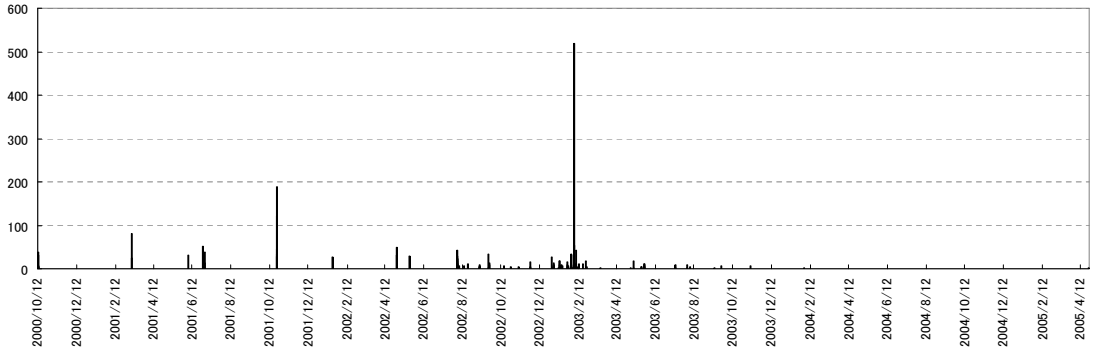
(1) $FMT(H)$

(2) $FMT(H')$

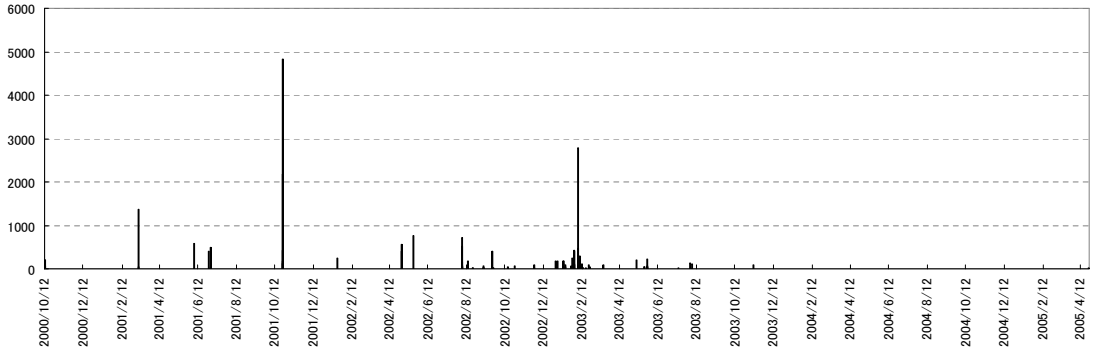
(3) $FMT(Q)$

(4) $FMT(Q')$

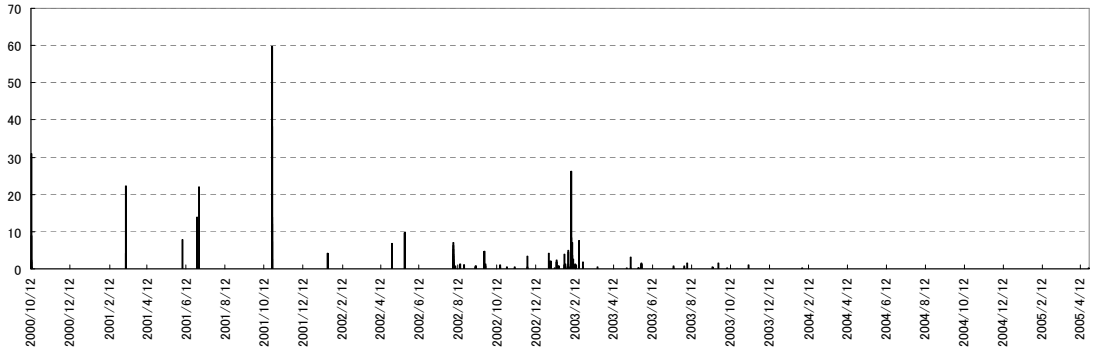
図 16: JHotDraw におけるメトリクスの変動度



(1) $FCT(DH)$



(2) $FCT(DE)$



(3) $FCT(DM)$

図 17: JHotDraw における変更の変動度

B　適用事例：HelpSetMaker

ここでは，6.1.2 節で挙げた手法の適用対象ソフトウェアである HelpSetMaker についての適用結果を掲載する．

掲載するのはモジュールの変動度，メトリクスの変動度，変更の変動度に関する適用結果である．(モジュール，メトリクス）の変動度，(モジュール，変更）の変動度，(モジュール，メトリクス，変更）の変動度については省略する．

B.1　モジュールの変動度

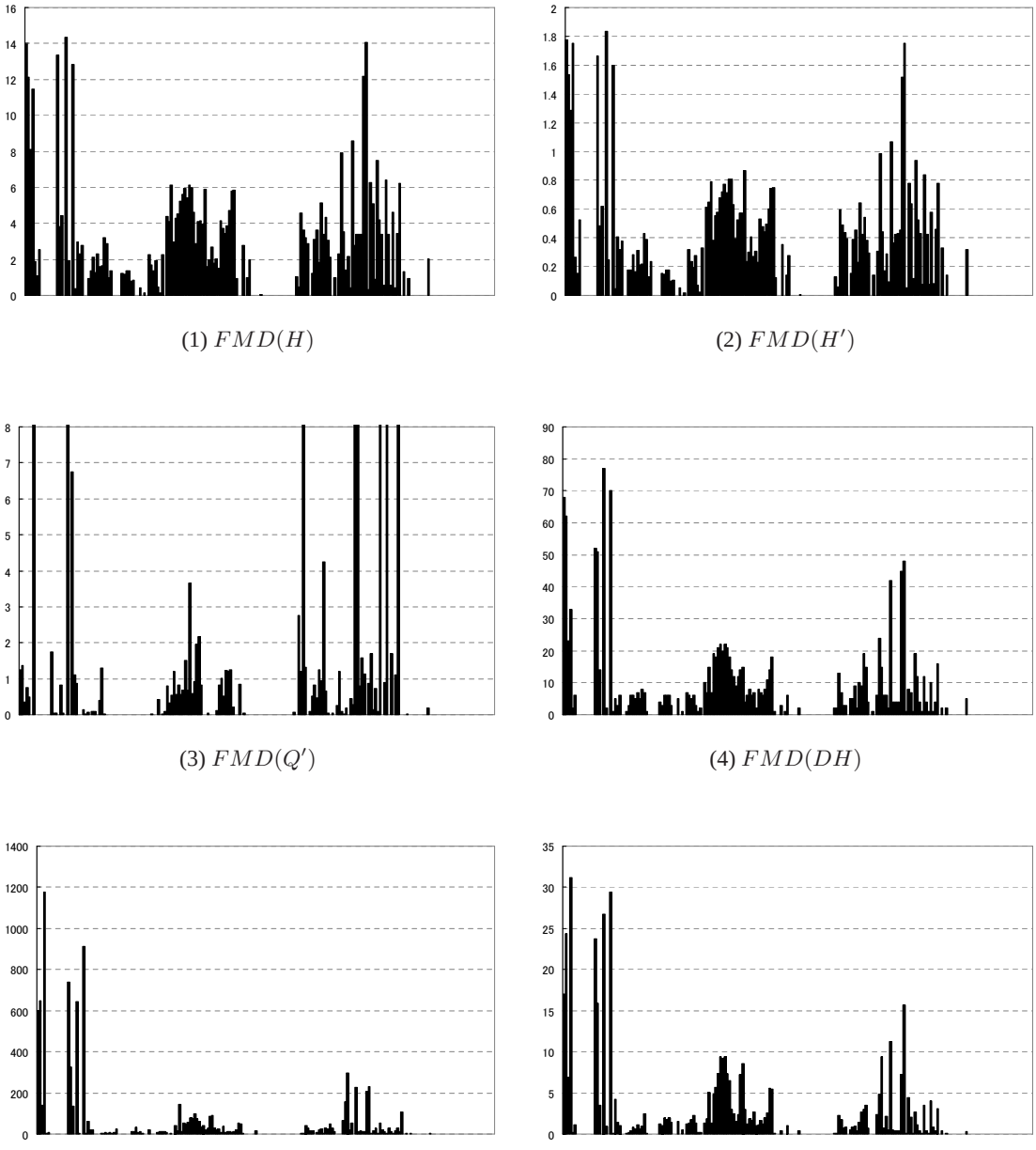
モジュールの変動度のグラフを図 18 に示す．グラフの横軸はモジュールであり，合計 210 クラスが全てのグラフにおいて同順で並んでいる．グラフの縦軸はそれぞれの指標の値を表している．

B.2　メトリクスの変動度

メトリクスの変動度のグラフを図 19 に示す．グラフの横軸はメトリクスであり，グラフの縦軸はそれぞれの指標の値を表している．

B.3　変更の変動度

変更の変動度のグラフを図 20 に示す．グラフの横軸は日時を表しており，グラフの縦軸は変更時刻におけるそれぞれの指標の値を表している．

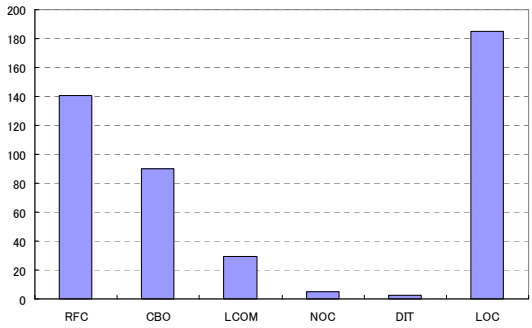


(1) $FMD(H)$ (2) $FMD(H')$

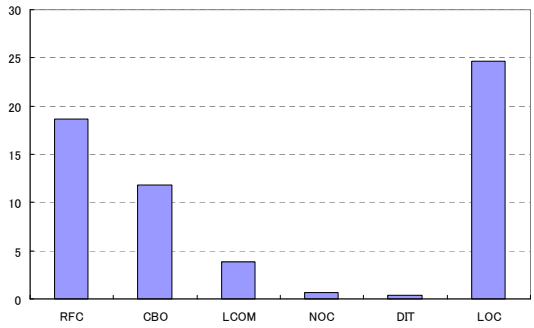
(3) $FMD(Q')$ (4) $FMD(DH)$

(5) $FMD(DE)$ (6) $FMD(DM)$

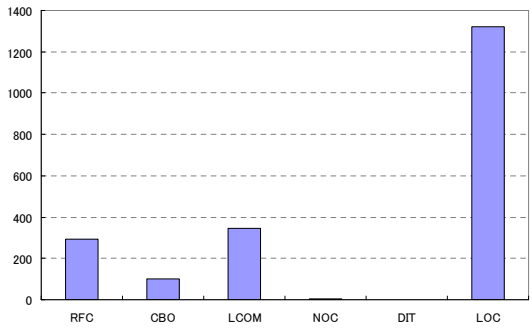
図 18: HelpSetMaker におけるモジュールの変動度



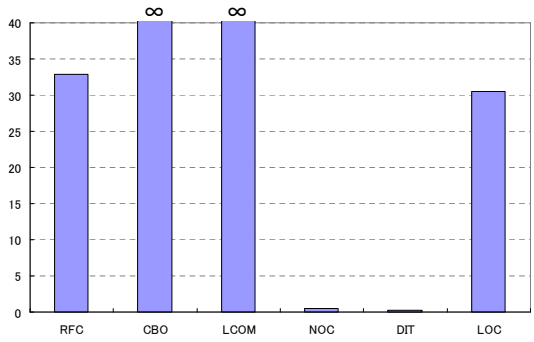
(1) $FMT(H)$



(2) $FMT(H')$

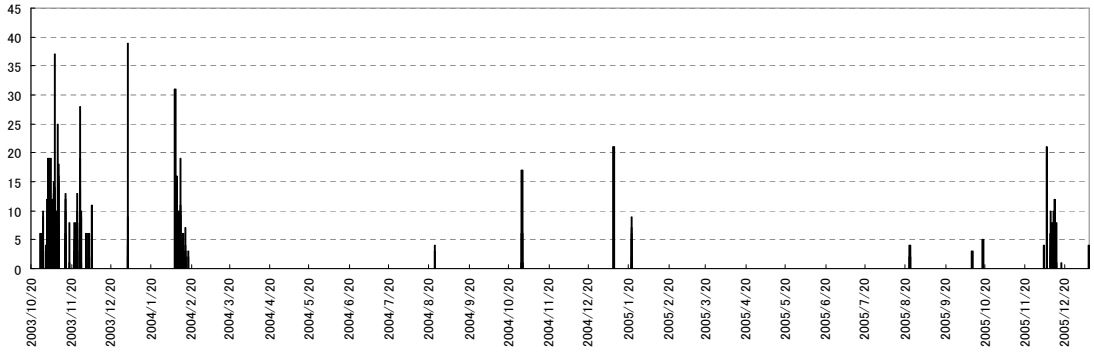


(3) $FMT(Q)$

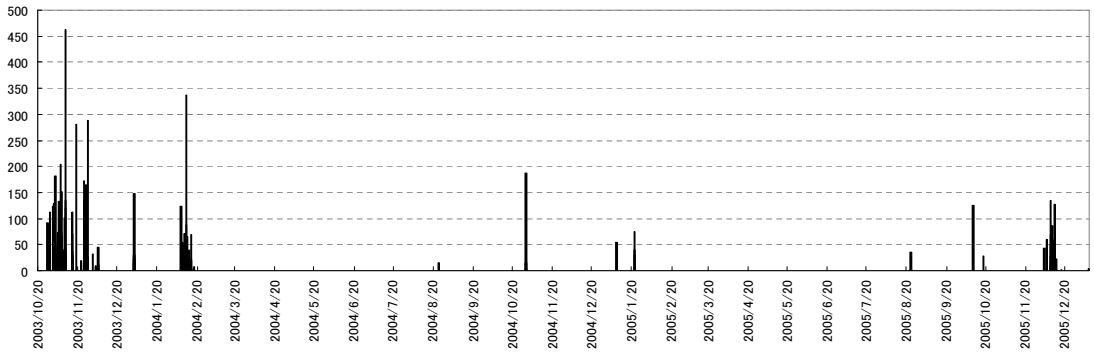


(4) $FMT(Q')$

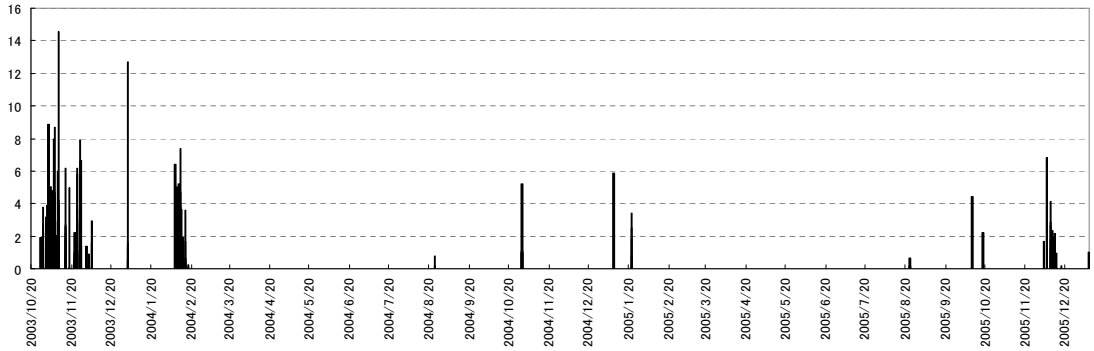
図 19: HelpSetMaker におけるメトリクスの変動度



(1) $FCT(DH)$



(2) $FCT(DE)$



(3) $FCT(DM)$

図 20: HelpSetMaker における変更の変動度