堆積型ファイルシステムMoraineとメトリクス環境MAMEへの適用

山本 哲男　松下 誠　井上 克郎

近年，ハードディスクの容量は飛躍的に大きくなり，その値段は急速に低下してきている．ディスク容量が十分に大きければ，不要なファイルを消す必要がなく，変化する全てのファイルのバージョンを保存出来ると考えられる．本論文では，まず自動的にすべてのファイルの変更を保存する堆積型ファイルシステムMoraineを提案する．Moraineでは常に最新のバージョンが透過的に操作可能であり，記録されたバージョンを容易に取り出すことが可能である．また，Moraineを用いたソフトウェアメトリクス環境であるMAME(Moraine As a Metrics Environment)を提案する．MAMEはソフトウェア開発に用いることにより，開発の途中あるいは終了後にさまざまなメトリクスデータを収集し分析することを可能にする．

1 はじめに

近年，ハードディスクの容量は飛躍的に大きくなり，その値段は急速に低下してきている．数十Gバイトの容量のハードディスクが数万円という低コストで入手でき，OSやツール，アプリケーションソフトなどをインストールしたとしても，何十Gバイトという残り容量をユーザが使うことができる．

一方，現在，我々がソフトウェア開発を行う際，用いているファイル管理やバージョン管理の方法は，過去，ディスクの容量が限られていた時期に考えられ，開発されたものが主である．不要になったファイルはユーザの手によって消去され，その領域は再利用される．バージョン管理システムでは、ユーザが陽にcheck-inの操作を行うことによって始めて保存されるバージョンが作成される．

しかし，ディスク容量が十分に大きければ，不要なファイルを消す必要はないであろう．また，バージョン管理においても，特定のファイルのみをバージョンとして登録するのではなく，一時的に作成されたものを含めて，変化する全てのファイルのバージョンを保存することができるのではないだろうか．

本研究では，ソフトウェアの開発作業に用いることを前提とした，堆積型ファイルシステムMoraineと呼ぶ新たなファイルシステムを提案する[17]．これは，ソフトウェア開発者の作成するプロダクト(ファイル)の全てを自動的に採取し，それらを一つのバージョンとして保存する。

Moraineでは，ソフトウェア開発者はcheck-inやcheck-outといったバージョンを管理するための作業を意識せず開発を進めることが出来る．また，Moraineを導入する際には，導入前の開発環境を変更する必要がない．特に指定しない限り，保存される各バージョンの

Accumulative File System Moraine and A Metrics Environment MAME.

Tetsuo Yamamoto, 大阪大学大学院基礎工学研究科, Graduate School of Engineering Science, Osaka University.

Makoto Matsushita, 大阪大学大学院基礎工学研究科, Graduate School of Engineering Science, Osaka University.

Katsuro Inoue, 大阪大学大学院基礎工学研究科 / 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Engineering Science, Osaka University / Graduate School of Information Science, Nara Institute of Science and Technology.

中で，常に最新バージョンのみが操作対象となる．

ソフトウェア開発におけるプロダクト管理のため、種々のバージョン管理システムが開発され，販売されているが，これらはMoraineのような全プロダクトの全ての変更を自動的に保存するという目的では設計されていない．また，一部の商用OSで用いられている自動バージョン番号付与ファイルシステム [9] は，Moraineと似た機能を提供するが，ディスクの容量の制限から保存できるバージョン数や廃棄期限が決められるのが通常である．

このような堆積型ファイルシステムを用いることによって，ソフトウェアを開発に関して，いろいろ新たな考えやそれに基づくシステム開発・支援環境を考えることができる．本研究では，Morainenの応用の一例として、メトリクス環境MAME(Moraine As a Metrics Environment)の提案とその実現を行った [17]．

現在まで、数多くのメトリクス環境が提案され，実装されている [3] [8] [14] [16]．しかし，これらは事前に収集するメトリクスを決め，そのためのツールを用意する必要がある．一般に，メトリクスを計測するためには，プロジェクトを開始する前にデータを収集する計画を立てる．例えば，GQMパラダイム [2] では目標に対して質問を決めることが必要である．それらを決めた後に収集するメトリクスを選ぶ．このようなトップダウン的な手法は，プロジェクトを開始する前にプロジェクトが詳細に決まっていれば成功する．そして，プロジェクトの開始時からデータの収集を行う事が出来る．

しかし，この手法では、プロジェクト管理の方針の変更に対応することは困難である．プロジェクトの進行中や終了後に，計画したもの以外の新たなメトリクスデータを収集することはできない．失われたプロダクトや終了したプロセスからのデータの収集は現在のソフトウェア開発環境では実行不可能である．

一方，Moraineはすべてのファイルのバージョンを細粒度で保存するため，MAMEでは，プロジェクトの開始後でさえもメトリクスを収集する方針を変更することが出来る．

本研究では，MAMEを実際の学生のコンパイラ作成演習に適用した．収集するメトリクスは演習の終了後に決定し，その後メトリクスの収集を行った．そして，これらのトリクスの解釈から学生の演習活動の特徴が理解出来た．

本論文では，堆積型ファイルシステムMoraineと，Moraineの有効性，有用性を示すアプリケーションであるソフトウェアメトリクス環境MAMEを提案する．以降，2節では，堆積型ファイルシステムMoraineの概要について述べ，3節ではMoraineの評価について述べる．4節ではMoraineを用いたソフトウェアメトリクス環境MAMEに付いて述べ，MAMEを学生実験に適用した結果を5節で述べる．6節で，MoraineとMAMEについてそれぞれ考察を行い，最後に7節でまとめと今後の課題について述べる．

2 Moraineの概要

本節では，堆積型ファイルシステムMoraineについて述べる．Moraineはファイルの更新を監視し，更新されたファイルを新バージョンとして記録するシステムである．バージョンを記録する際に既存のツールを使うことで，それらのツールに対応した他のツールを適用することも可能になる．また，ファイルシステムとして実装することで既存の開発環境を一切変更することなく導入が可能になる．

2.1 設計方針

近年，ディスクの大容量化やCPUの性能は急速に向上している．開発管理，プロダクトの品質の向上など，ソフトウェア開発におけるさまざまな観点を考えた場合，ソフトウェア開発環境上で行われた開発者全員の作業は記録するべきであると我々は考えている．また，記録した作業履歴は容易に取得可能でなければならない．

このような開発環境では，作成されるすべてのファイルのすべてのバージョンが記録される．また，開発者はバージョンの保存について何も行う必要がない．ファイルが必要無ければ消去することも通常の操作で行うことが出来き，消去されたファイルもタイムスタンプやユーザによって指定されたタグによって後から取得できる．

これらの考えを元にMoraineの設計方針を以下に示す．

- 容易な操作
  ユーザはMoraineの操作を事前に学ぶ必要がない．Moraineは通常のファイル操作を自動的に監視し，開発者の特別な操作を必要としない．

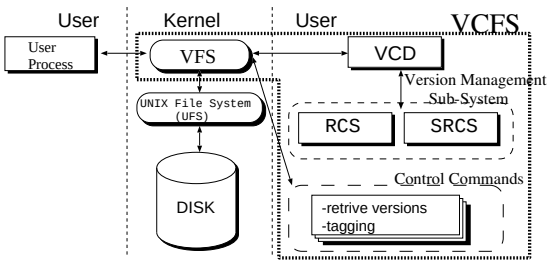

図1 Moraineのアーキテクチャ

- オープンな構造
  Moraineはデータリポジトリやバージョンの管理に独自の構造を持たない．そのため，容易に多くのシステムに移植が可能である．

以上の設計方針に基づいてUNIXのファイルシステム上にバージョン管理機能を組み込んだシステムを設計した．この手法では，通常のファイルに対する読み出し(readシステムコール)，書き込み(writeシステムコール)動作を直接バージョン管理作業に対応付ける．また，既存のファイルシステムを用いたスタッカブルファイルシステム(Stackable File System) [4]として実装しており，既存の環境に容易に導入可能である．

2.2 Moraineのアーキテクチャ

図1は我々が提案するMoraineのアーキテクチャを表している．このシステムの基本部分はVCFS(Version Control File System)である．MoraineはVCFS，VFS(Virtual File System)，VCD(Version Control Daemon)，バージョン管理サブシステム，管理用コマンド群から構成される．今回の実装においては，バージョン管理サブシステムとしてRCS(Revision Control System) [15]とSRCS(Simple Revesion Control System)が利用可能である．SRCSは，我々が作成した，ファイルを圧縮せず単純にバージョンを記録していくサブシステムである．

バージョンが記録されるのは，新規にファイルを作成した場合や既存のファイルを変更した場合である．これらの場合，ファイルを新規バージョンとして記録する．また，ファイルを読み出す場合は最新のバージョンを読み出すことになる．さらに，VCFSはファイルのロックを行い，ファイルのバージョンとしての記録作業が終わるまで，他のプロセスからはそのファイルに対して読み出しのみが行なえる．

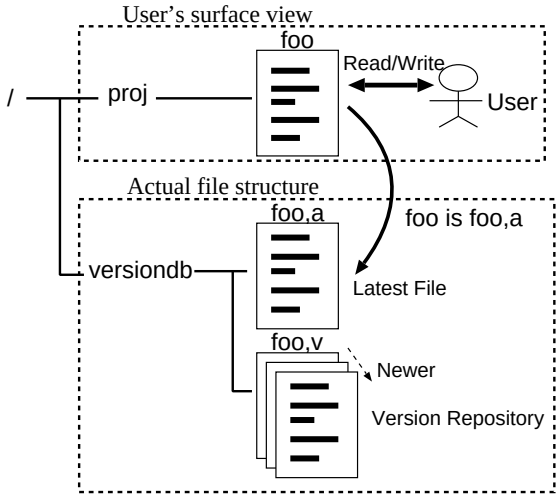

図2 VCFSと実ファイルシステムの関係

VCFSではVFSによりスタッカブルファイルシステムとして実装されている．そのため，VCFSによって管理されているファイルはUNIXの通常のファイルシステムに直接保存されず，VFSを通して保存される．

各ファイルは最新バージョンのファイルと過去のバージョンを記録したファイルからなる(図2)．通常，VCFSでは最新バージョンのファイルのみ操作可能である．過去のバージョンは直接参照することが不可能である．例として，VCFSで*/versiondb*以下に存在するすべてのファイルやディレクトリを */proj* からも読み書き出来るように接続し，*/proj/foo* というファイルを作成した場合，VCFSは *foo* (*/proj/foo*自身)の最新バージョンのファイルとして *"/versiondb/foo,a"* を作成し，過去のバージョン記録ファイルとして *"/versiondb/foo,v"* を作成する．*/proj/foo* は */versiondb/foo,a* というファイルを指すことになる．また，*/versiondb/foo,v* は */proj* の下では参照出来ない．ユーザは */proj* の下のファイルのみを操作することになる．

VCFSは常に最新バージョンのファイルをファイルシステム上に用意しているため(上記の例では */versiondb/foo,a*)，ファイルの読み出しはそのファイルを単に読み出すだけであるので高速に動作する．ファイルを読み出し専用で開いた場合は，NULLファイルシ

ステム [13](何も変更せずに通常のファイルシステムに読み書きを行うファイルシステム) と同じ動作をする．それに対し，書き込みフラグを付けてファイルを開いた場合，読み出しや書き込みといった操作は従来と同じであるが，ファイルに対して閉じる動作 (close システムコール) を行った際，そのファイルに対して check-in 動作を行う．この check-in の動作は VCD が行う．

VCD はカーネル内の VFS とバージョン管理サブシステムとの中継を行うデーモンプログラムである．VCD はカーネルから依頼されるファイルのバージョン管理を引き受け，バージョン管理サブシステムを起動する．

バージョン管理サブシステムは，実際にバージョンを記録する部分である．バージョン管理サブシステムは複数のバージョン管理ツールの集合体である．今回は RCS [15] と SRCS を用いたが，他のツールを使用することも出来る．

管理用コマンド群は，通常のファイル操作では行えない操作をするためのコマンド群である．コマンドの例として，任意のバージョンのファイルの取り出し，ブランチ作成，バージョン間の差分の閲覧などがある．

Moraine の実装は BSD UNIX [7] [10] 系のオペレーティングシステムである FreeBSD 3.0-RELEASE [5] を対象にして行った．実装はすべて C 言語で記述し，全体で 5000 行程度である．

3 Moraine の評価

本節では，システムの性能とファイルシステムに必要な容量の点から Moraine の評価を行う．システムを導入しても，システムの動作が遅く開発に支障を来たしてしまう場合には実際の開発環境へに導入することは難しい．そこで，ファイルの読み書きに関してファイルシステムの性能評価を行った．

本評価では，ファイルシステムとして通常用いられる UNIX ファイルシステム (UFS) とスタッカブルファイルシステムの一つである NULLFS と VCFS の比較を行った．以下で述べるテストは，バージョン管理サブシステムとしてバージョン間の差分を計算して圧縮して保存するのみ RCS を用い，Pentium 166MHz/48MBRAM の計算機で行った．

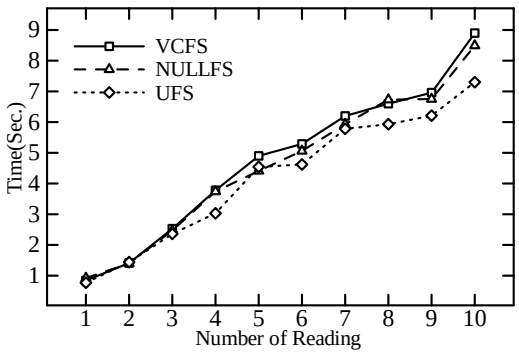

図 3 ファイルの読みだし性能

3.1 ファイルの読み書きテスト

ファイルの大きさが 1MB の異なるファイルを，単一プロセスで繰り返し連続して読み出す時の処理時間を計測した．ここで，ファイルの内容は処理時間に影響を与えない．読み出し実験は回数を変化させて各回数毎に処理時間を測定した．複数回読み出しを行う時は，同一ファイルを指定された回数読み出すのではなく，異なるファイルを読み出す．そのため，ファイルの個数を読み出す回数分だけあらかじめ用意しておく．ここで，処理時間とは連続した読み出しを行うプロセスの実行時間である．つまり，プロセスの生成から消滅までの時間となる．測定した結果を図 3 に示す．縦軸は実行時間を示し，横軸は読み出した回数を示す．

いずれの回数においても VCFS と NULLFS はほとんど違いが見られない．このことからファイルの読み出しに関して，バージョン管理機能に要する時間はわずかであることが分かる．VCFS や NULLFS は，UFS と比べると約 10% 余計に時間を要している．これはスタッカブルファイルシステムとして実装されていることによると考えられる．

同様のテストを，書き込みに関しても行った．ファイルの大きさを 1MB とし，異なるファイル名で単一プロセスで繰り返し連続して書き込む時の処理時間の計測を行った．測定結果を図 4 に示す．“VCFS+” は VCD と RCS の間で同期をとり完全にバージョンの記録が完了するまでの時間を計測したものである．“VCFS” の場合は RCS の終了を待たない．

UFS と比べ，NULLFS は約 10% 余計に時間を要する．

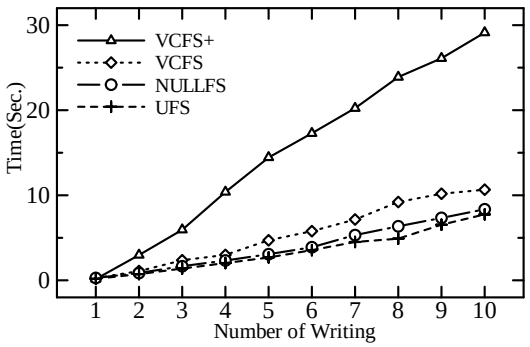

図4 ファイルの書き込み性能

これは読み出しにおけるUFSとNULLFSとの違いと同じである．VCFSとUFSを比べると，書き込みの際には新たなバージョンを保存する時間を要するため，約30%VCFSのほうが遅い．VCFS+は，UFS，NULLFS，VCFSと比べると数倍の時間を要する．これはバージョンの記録を行うためのRCSの処理時間を含んでいるためである．しかし，実際はバックグランドでバージョン管理サブシステムが動作し，バージョンの記録を行う．このため，ファイルを書き込む利用者に対する応答時間は，VCFS+ではなくVCFSの時間になる．VCFSの時間は，利用者に対してファイルの保存に関する動作の支障をきたすことはないと考える．

最後に，実際のアプリケーションのコンパイルに要する処理時間を測定した．測定対象となるアプリケーションはFreeBSDに付属の“tar”と“dump”のプログラムとした．アプリケーションのソースファイルは各ファイルシステム上に用意しておき，“make”の開始から終了までの時間を処理時間とした．測定結果を表1に示す．コンパイルが行う作業は，異なるソースファイルのコンパイルの繰り返しである．コンパイル時に行われる作業は，ソースファイルの読み出しとそのソースファイルから生成されたオブジェクトファイルの書き込みであり，ファイルシステムに対して読み書きを行う．VCFSはUFSと比べると20%程度の時間を要する．これは，実用上問題ない程度の時間である．

表1 アプリケーションのコンパイル時間(秒)

	dump	tar
VCFS	12.79	33.46
NULLFS	11.3	32.01
UFS	10.9	28.27

表3 ディスク使用容量(Kバイト)

	UFS	VCFS
student1	225	1388
student2	117	546
student3	73	604

3.2 容量テスト

筆者の所属する学科で行われるPascalサブセットコンパイラ作成演習の作業をMoraineに適用した．適用したデータを表2に示す．“全行数”は最終バージョンのC言語のソースファイルの全行数を表す．“総ファイル数”は最終バージョンC言語のソースファイルの総数を示す．“総バージョン数”はC言語のソースファイル，オブジェクトファイルなどすべてのファイルのすべてのバージョンの総数を示す．括弧内の値はC言語のソースファイルのみのバージョンの総数を示す．例えば，student2は全行数が4067行で，20個のソースファイルを作成し，総バージョン数は249個にであることを示す．そのうち，C言語のソースファイルのバージョン数は147個であった．

UFS(NULLFSはUFSと同じとなる)とVCFS上で最終的なファイルの総容量を表3に示す．この値はソースファイルだけでなく，コンパイルした結果のオブジェクトファイルも含んでいる．

UFS上の容量が通常のファイルシステムで保存した時の大きさであり，この大きさと比べるとVCFSは6–8倍の容量を必要とする．しかしながら，最も大きい場合でも1.4Mバイトであり，バージョン管理サブシステムとして差分圧縮を行わないSRCSを用いた実験でも，最も大きい場合で20倍程度の3.1Mバイトとなった．

表2 プロジェクトの内容

	全行数	総ファイル数	総バージョン数(ソースファイルのみ)
student1	9339	45	533 (311)
student2	4067	20	249 (147)
student3	2543	18	357 (247)

4 MAMEの概要

本章では，Moraineの応用の一例として、メトリクス環境MAME(Moraine As a Metrics Environment)について述べる．これまでのメトリクス環境は，事前に収集するメトリクスを決め，そのためのツールを用意する必要がある．一方，MAMEは，全てのファイルのバージョンを漏れなく保存するファイルシステムMoraineを用いて実現されているため，プロジェクトの開始後でさえもメトリクスを収集する方針を変更し，過去のプロダクトの情報を収集することが可能である．

4.1 メトリクス環境の必要条件

本節では，定量的計測を行うメトリクス環境として必要な事項について考える．メトリクス環境とは，一般のソフトウェア開発の定量的プロセスやプロダクトの計測のための環境である．メトリクス環境では，開発者の活動から定性的ではなく定量的なメトリクスを収集する．また，データを保存し分析するための機能を提供する．メトリクス環境によって収集されたデータは，ソフトウェアプロセス評価の際にも使用出来る．我々は，メトリクス環境には以下の4つが必要だと考える．

1. 開発者に余計な負担をかけない．
   開発者は，“特別なツールを使うように” や “あらかじめ決められた方法で開発してくれ” といった，データを収集するために特別な操作を強要されるのを好まない．そのため，これらの余分な作業について考える必要がある．さらに，開発者にこれらの作業を課すことは収集するデータの品質に問題を引き起こすかもしれない．
2. さまざまな種類のメトリクスデータを容易に収集出来る．
   あるメトリクスデータを収集するたびに単一のプログラムを利用するのではなく，汎用性のあるツールからなるツール群を構築する．これらのツール群は多くのメトリクスデータを収集するための一般的な環境を持つ．
3. さまざまな粒度のメトリクスデータを容易に収集出来る．
   ソフトウェア開発活動には，開発者や管理者といった多くの側面からの活動がある．さまざまな側面に応じて，異なる粒度のデータが必要なる場合がある．さらに，あるメトリクスにおいてさまざまな粒度を考えることで，メトリクスの抽象化をより多くの角度から行うことが出来る．
4. 収集したメトリクスデータの構造は独自なものではない．
   近年，オープンソースの考え方 [6] が広がりつつある．ソフトウェアアーキテクチャ，設計，実装などを隠すことなく，世の中へ公開する動きがある．このような状況では，収集するメトリクスデータに一般的に用いられる構造を採用することはとても重要である．メトリクス環境の質を高めることにもなる．

4.2 MAMEアーキテクチャの設計

図5にMAME(Moraine As a Metrics Environment) のアーキテクチャを示す．MAMEはMoraineを利用したメトリクスデータを容易に収集出来るメトリクス環境である．Moraineが収集したファイルの更新履歴を用いて，さまざまなメトリクスデータを提供する．さらに，任意のファイルごとの詳細な更新履歴を，Webブラウザを用いて参照出来るツールも提供する．Moraine上に開発環境を構築することにより，開発環境中のファイルに対する全ての操作はMoraineによって記録される．開発者はデータの分析ツールなどで，記録された履歴をMoraineから取得することが出来る．これらの情報を得るにあたって，開発環境自身を変更する必要はなく，単にMoraine上でこれまでと同様の開発を行うだけである．開発者はメトリクスをMAMEで取得可能かどうかを気にすること無く，開発作業を続けることが出来る．

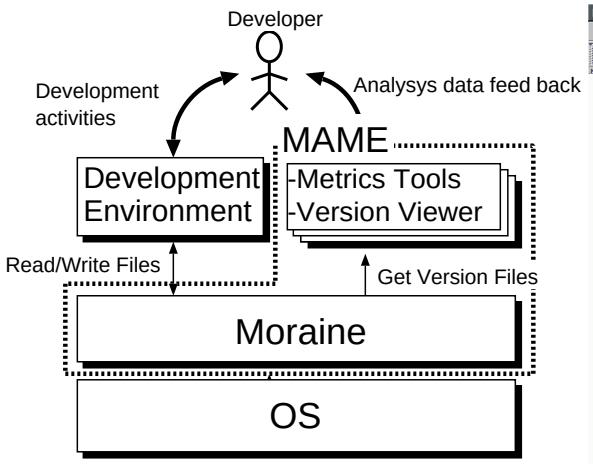

図5 MAMEアーキテクチャ

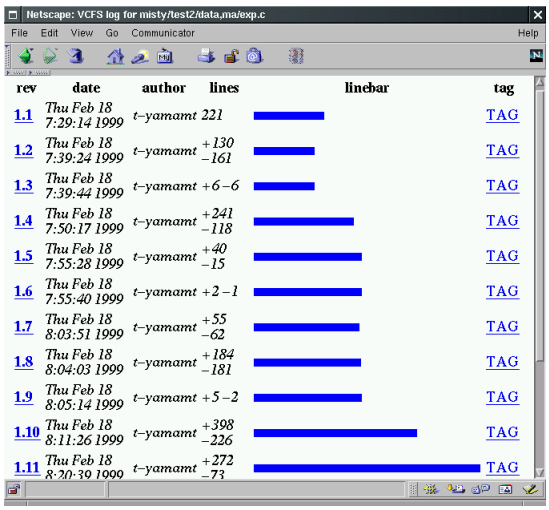

図6 バージョンビューア

MAMEにはさまざまな種類のメトリクスツールが含まれている．現在の実装には，メトリクスツールが出力する情報はバージョンの数，ファイルの更新時間，ファイルの更新者，ファイルの行数，ファイルがC言語で記述されている場合は関数の個数，ディレクトリ内のファイル一覧がある．また，これらの情報を組み合わせた出力も可能である．これらのデータはテキスト形式で出力され，Perlなどの言語を用いて容易に加工が出来る．

図6はバージョンビューアの画面イメージである．バージョンビューアはWebブラウザを用いている．画面の一番左の列はバージョン番号を表す．バージョンは最初のバージョンである1.1から始まり，最新のバージョンまですべてを表示する．各バージョンは，バージョン番号，更新した日付，更新した開発者，前バージョンからの追加もしくは削除された行数，行数を棒グラフ化したもの，もしタグがあればそのタグから構成される．このツールを用いることでバージョンの変更履歴が視覚的に表現できる．

4.3 MAMEの機能

MAMEは，4.1節に示したメトリクス環境に必要な機能を満たしている．以下にMAMEにおけるそれぞれの機能について説明する．

1. MAMEでは，メトリクスのデータの収集にMoraineを用い，従来の開発環境をMoraineの上にそのまま構築している．このため，開発者は開発者自身の環境を変更する必要がなく，MAMEを導入する以前の開発形態でそのまま開発が続けられる．つまり，開発者に対してメトリクスを収集するための特別な作業を指示する必要はなく，余計な負担をかけない．
2. MAMEは，開発環境上のファイルに対する更新履歴を収集する．我々はソフトウェア開発作業の多くはファイルに対する操作だと考える．MAMEはファイルの内容の変更や，ファイルの作成，削除といったファイルに対するすべての操作を記録する．ファイルの更新時間や更新者，ファイルの行数といったデータが容易に取得可能であり，これらのデータを加工することによってメトリクスデータを容易に収集出来る．
3. MAMEは，開発者が保存したファイルの更新履歴をすべて保存している．そのため，ファイルに関する細粒度のデータが取得出来る．また，開発時に行った操作を別途に記録し，共に用いることで，より高い抽象度のデータを得ることも可能であろう．
4. MAMEで取得可能なデータはファイルに関する情報をテキスト形式である．出力されたデータをそのまま使用することも出来，複数のデータを独自に加工するこも可能である．

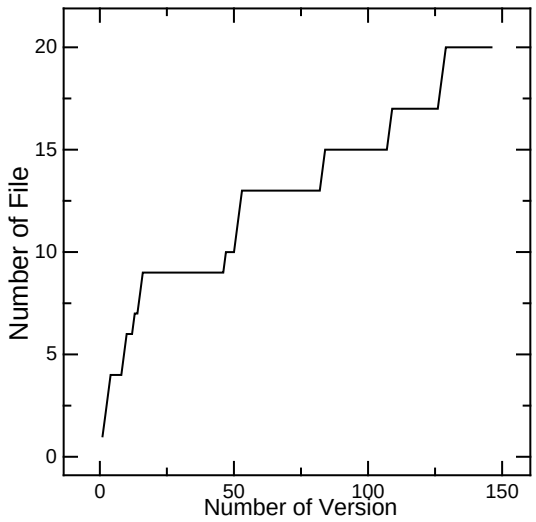

図7 ファイル数の推移 (Student 2)

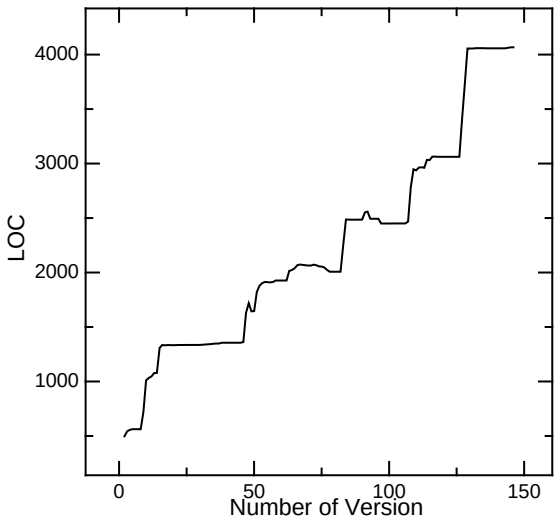

図8 行数の推移 (Student 2)

5 MAMEを用いた実験

開発作業後に開発中に作成されたファイルに関するメトリクスデータが，MAMEで実際に取得可能であることを示す．

5.1 実験の概要

3節で示したPascalサブセットコンパイラ作成演習をMAMEに適用した．データ収集は演習後に行った．今回は，学生の開発動向を知るために，3つのメトリクスを選んだ．それぞれ，ファイルの総数(同時に存在する数)，ソースファイルの全行数，C言語のソースファイル中の関数の総数である．

図7，図8，図9はstudent 2の各メトリクス値を表している．これらのグラフの横軸はC言語のソースファイル(147個のバージョン)の累積したバージョン数を表している．横軸にはバージョン数ではなくファイルの更新時間を用いることも可能であるが，学生の演習の場合は実時間よりもバージョン数での推移の方が良いと考える．なぜならば，学生の場合は連続した開発は行わず，実時間には関係ない時間で開発するためである．

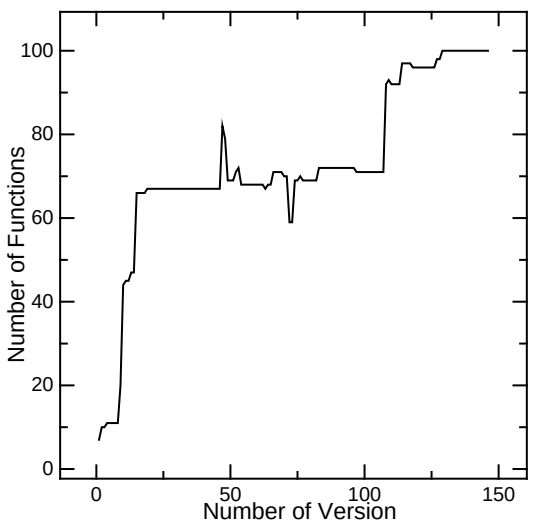

図9 関数の数の推移 (Student 2)

5.2 メトリクスデータの解釈

図7と図8から，開発は特に問題なく順調に進んだと考えられる．両方のグラフ共に，ほとんど単調増加を示している．バージョン数が20から50の間では，すべてのグラフにおいてほとんど変化は見られない．これは，バージョン数は増えているがファイルの行数はほぼ一定であることを示している．これらのバージョンに対して作業を行っている間，学生はファイル内の細かな変更を

行っていると考えられる．

図9は初期の段階(バージョン数が約20)で大きな増加を示している．そして，しばらくその状態が続いている．学生は初期の段階で必要な関数を作成し，後にその関数内を記述していることが分かる．また，最後の段階(バージョン数が約100)で多くの関数を追加している．

このように，学生が行った開発の動向は，MAMEの出力から容易に得ることが出来る．

6 考察

6.1 Moraine

Moraineは，現在の大容量なハードディスクがある環境下では有用である．作成したすべてのファイルのすべてのバージョンを開発者に負担をかけることなく自動的に記録するため，開発者はバージョン管理の機構やコマンドなどを学ぶ必要がない．Moraineを導入しても，開発者の環境を一切変更することなく保存されたファイルの履歴をすべて保存出来きる．

Moraineをバージョン管理システムとして見た場合，効率的に任意のバージョンを記録し，取得できるという機能を持っている．また，任意のバージョンに明示的なタグを用いることで指定のバージョンの特定を容易に行うことも出来る．

現在のMoraineの実装では，タイムスタンプや指定したタグを用いることで過去のバージョンの取得を行う．これはCVSやRCSと同等の機能である．しかし，Moraineで記録したバージョンの粒度はCVSやRCSで記録したバージョンより細かい．そのため，過去のバージョンを検索し取得するための，より洗練された機構が必要である．

Moraineは利用者が保存したファイルを自動的にかつ蓄積してすべてのバージョンを記録する．バージョン管理システムとして関連したツールとしてClearCase [1]，PVCS [11]，Visual Source Safe [12]などがある．これらのツールは，check-in/check-outモデルを採用しており，分散開発や同時並行開発やビルド管理といった機能を実現している．さらに，グラフィカルユーザインターフェースを用いて操作が行うことができ，特定の開発環境と統合された場合にのみ使用出来る．これらのツールはさまざまな粒度でオブジェクトのバージョンを記録する．しかし，開発者はシステムの利用方法を知る必要があり，check-in/check-outコマンドを利用しなければならない．また，特定の環境を想定しているため，汎用性に欠ける．一方，Moraineではバージョンの記録は完全に自動化されており，ファイルシステムとして実装することで特定の環境に依存しない．しかしながら，Moraineはバージョンを取得する機能に中心がおかれており，ClearCaseやPVCSなどのツールに比べファイル間の関係の管理や複数人での開発を支援するなどの機能が現在の実装には備わっていない．また，一部のOSで用いられている自動バージョン番号付与ファイルシステム [9]は，ディスクの容量による制約から，保存できるバージョン数や廃棄期限が決められるのが通常である．一方，Moraineはファイルの全ての保存保存作業で生じたすべてのバージョンを保存している．また，これらのファイルシステムはOSと不可分な形でて実現されており，可汎性も問題がある．

Moraineの性能は，実際のソフトウェア開発で十分実用的と言える．バージョンを記録する際の性能の低下はあまり存在しない．ファイルの差分を計算することによって，システムの負荷は上昇するが，この計算はバックグラウンドで実行されるため，実際には計算を終える前にファイルの操作は終了する．これにより，ユーザに対する負荷はほとんど存在しない．

Moraineは新しいソフトウェア開発環境モデルの基礎となるであろう．現在のソフトウェア開発環境はファイルなどに対して多くの管理操作が必要である．また，システムにすべての必要なファイルを保存しなければならない．Moraineを用いることで，必要なくなったファイルを完全に消去することも出来る．消去されたファイルもMoraineに保存されているため，将来またそのファイルが必要になった場合に，ファイルを復元することが容易に行える．

6.2 MAME

MAMEはMoraineを用いて実現されており，ソフトウェアメトリクス環境として新しいアーキテクチャである．MAMEは4節で示したメトリクス環境にとって必要な機能を全て持っている．このため，開発者に特別な強制を強いることなく，メトリクスデータを収集するた

めの負荷もほとんどない．メトリクスデータを集める際，蓄積されたバージョンは後で容易に取り出すことが出来る．MAMEを導入して開発を行った場合，開発作業後でもファイルの行数などのファイルに関するメトリクスデータを，細かい粒度で取得可能となる．

MAMEの本質は過去のバージョンや消去されたファイルに対するメトリクスデータを取得出来る所にある．そのため，プロジェクトの途中や終了後でさえもメトリクスの収集方針を設定したり変更したりすることが出来る．一般的なメトリクス環境ではあらかじめ決められた収集方針があるが，MAMEでは開発前にあらかじめ収集方針を決める必要がない．この特性はすべてのファイルのすべてのバージョンを記録するシステムによって成り立っている．また，3節や5節で示した実験データによって，MAMEは十分実用的であることが示されている．ディスクの容量は通常のファイル使用容量よりも必要であるが，近年のディスクの大容量化により問題にはならない．

ファイルの更新をバージョンとして記録しているため，ファイルの更新履歴のみが取得出来る．そのため，なぜファイルを更新したかを知るにはファイルの更新履歴から分析するしかない．たとえば，オブジェクトファイルが更新された場合，コンパイルを行ったであろうといった分析を行う．このような分析から開発者の作業過程の推測が可能であろうと考えている．

メトリクス環境として使われている環境としてMETKIT (Metrics Education Toolkit) [14]がある．METKITは通常の開発でメトリクスデータを収集するために使われる．開発者は必要なデータを収集するためにモジュールを導入し作成する．つまり，METKITを利用する際には事前に取得するメトリクスを定める必要がある．Crocodile [8]も既存のツールと組み合わせた環境である．しかし，これもメトリクスの方針は開発前に定める必要がある．TAME [3]はメトリクス環境を構築するためのプロジェクトである．TAMEはメトリクスデータを収集するのにGQMパラダイム [2]を導入している．そのため，収集するメトリクスを決定するのにトップダウン的な手法が用いられている．Ginger2 [16]もまたメトリクス環境を構築している．しかし，独自のツール群を用いているため，実際の開発環境に適用することは困難である．

7 まとめ

本論文では，ソフトウェア開発環境の基盤として用いることが出来る，すべてのファイルの全バージョンを自動的に記録する堆積型ファイルシステムMoraineを提案した．Moraineはバージョン管理を容易な操作で提供し，オープンシステムな構造を持っている．そして，さまざまなソフトウェア開発プロジェクトに適用可能である．また，Moraineを性能とディスク容量の点から評価した．

また，我々はMoraineの有用性を示すために，容易にメトリクスデータを収集可能な環境であるMAMEを提案した．MAMEはMoraineを用いており，開発者に負担をかけることなくメトリクスデータの収集を行う．また，MAMEをある開発事例に適用し，MAMEから得られたメトリクスデータを用いて学生の活動の分析を行った．MAMEを用いることでメトリクスデータの収集及び解析は容易なものとなった．

今後の課題として，MoraineやMAMEを大規模なソフトウェア開発プロジェクトに適用することが挙げられる．また，システムの信頼性を高めるために，我々はカーネルとMoraineのインタフェースの実装を見直している．

参 考 文 献

[1] Atria Software Inc.: ClearCase Product Summary, Technical report, Atria Software Inc., 24 Prime park Way, Natick, Massachusetts 01760, 1994.

[2] Basili, V. R., Caldiera, G., and Rombach, H. D.: Goal Question Metric Paradigm, *Encyclopedia of Software Engineering*(Marciniak, J. J.(ed.)), Vol. 1, John Wiley & Sons, 1994, pp. 528–532.

[3] Basili, V. R. and Rombach, H. D.: The TAME Project: Towards Improvement–Oriented Software Environments, *IEEE Transactions on Software Engineering*, Vol. SE-14,No. 6(1988), pp. 758–773.

[4] Heidemann, J. and Popek, G.: File-System Development with Stackable Layers, *ACM Transactions on Computer Systems*, Vol. 12,No. 1(1994), pp. 58–89.

[5] Hubbard, J. K.: RELEASE NOTES FreeBSD Release 3.0-RELEASE. “http://www.freebsd.org/releases/3.0R/notes.html”.

[6] Laymond, E. S.: The Cathedral and the Bazaar. “http://www.tuxedo.org/%7Eesr/writings/cathedral-

bazaar/cathedral-bazaar.ps".

[7] Leffler, S., McKusick, M., karels, M., and Quarterman, J.: *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.

[8] Lewerents, C. and F, S.: A Product Metrics Tool Integrated Into a Software Development Environment, *In Proc. European Software Measurement Conference FESMA98*, 1998, pp. 603–608.

[9] McCoy, K.: *VMS File System Internals*, Digital Press, 1990.

[10] McKusick, M., Bostic, K., karels, M., and Quarterman, J.: *The Design and Implementation of the 4.4BSD UNIX Operating System*, Addison-Wesley, 1996.

[11] Merant, Inc.: Merant PVCS Version Manager & Configuration Builder. "http://www.merant.com/products/pvcs/".

[12] Microsoft, Inc.: Visual Source Safe. "http://msdn.microsoft.com/ssafe/".

[13] Pendry, J.-S. and McKusick, M.: Union Mounts in 4.4BSD-Lite, *Proceedings of the USENIX 1995 Technical Conference*, New Orleans, LA, USA, January 16–20 1995, pp. 25–33.

[14] Russell, M. and Bush, M.: Introduction to METKIT, *Journal of Information and Software Technology*, Vol. 35,No. 2(1993), pp. 108–110.

[15] Tichy, W. F.: RCS – A System for Version Control, *Software–Practice and Experience*, Vol. 15,No. 7(1985), pp. 637–654.

[16] Torii, K., Matsumoto, K., Nakakoji, K., Takada, Y., Takada, S., and Shima, K.: Ginger2: An Environment for CAESE (Computer-Aided Empirical Software Engineering), *IEEE Transactions on Software Engineering*, Vol. 25,No. 4(1999), pp. 474–492.

[17] Yamamoto, T., Matsushita, M., and Inoue, K.: Accumulative Versioning File System Moraine and Its Application to Metrics Environment MAME, *Proceedings of the ACM SIGSOFT Eighth International Symposium on the Foundation of Software Engineering*(Rosenblum, D. S.(ed.)), San Diego, CA, November 2000, pp. 80–87. Published as ACM SIGSOFT Software Engineering Notes, vol. 25, No. 6, November 2000.