特別研究報告

題目

ソースコード更新履歴を利用した
変更危険度計測システムの試作

指導教員
井上 克郎 教授

報告者
村尾 憲治

平成 18 年 2 月 20 日

大阪大学 基礎工学部 情報科学科

平成 17 年度 特別研究報告

ソースコード更新履歴を利用した変更危険度計測システムの試作

村尾 憲治

内容梗概

近年のソフトウェアの大規模化，複雑化に伴い，その開発形態は多人数化，分散化しており，開発過程で新規開発者が参入することも多い．しかし，新規の開発者にとって，大規模複雑なソフトウェアの全体像を把握することは容易ではない．また，ソフトウェアに対して不具合の修正や新機能の追加等の変更を行う際，その変更により新たな不具合が発生することがある．ソフトウェアに対する理解が十分でなければ，このような不具合の発生する可能性を事前に知ることは難しい．

一方，ソフトウェアの開発を行う際には，開発されたプロダクトを効率良く管理する為に版管理システムを利用することが多い．このシステムに存在するリポジトリと呼ばれるデータベースには開発履歴情報が蓄積されている．ソフトウェア開発において，この履歴情報は以前の開発についての理解に役立つことが知られている．従って，この開発履歴情報を解析することにより，ソフトウェアに対して変更を行う際に注意を必要とする箇所の特定が可能であると期待できる．

そこで，本研究では版管理システムに蓄積された開発履歴情報を利用し，ソフトウェアのソースコードに対して変更危険度を計測する手法の提案を行う．変更危険度とは「変更によって問題が発生する可能性の大きさ」を表す．また，提案手法を実装するシステムの試作を行う．さらに，実際に開発に版管理システムを使用したソフトウェアに対して本システムを適用し，評価を行なった．その結果，本システムを用いることで開発者はソフトウェアの変更作業において注意すべき箇所を知ることができることが確認できた．このことにより，ソフトウェア開発者は注意すべき箇所に対する変更を行う際により慎重な行動を喚起され，ソフトウェアの開発生産性の向上が期待できる．

主な用語

版管理システム
開発履歴情報

目 次

1　はじめに

オープンソースソフトウェアの開発規模の増大に伴い，その開発形態は多人数化，分散化している．大規模なオープンソースソフトウェア開発では，複数の開発者が互いにソースコードを共有しながら同時に一つの開発作業に携わることが一般的になりつつある．開発過程で新規の開発者が参加することも珍しくない．ソフトウェアの大規模化，複雑化が進む近年，新規の開発者にとってソフトウェアの全体像を把握することは困難になってきている．

また，開発過程で機能の追加，拡張あるいは不具合の修正等の必要から，開発者はしばしばソースコードに何らかの変更を施す．変更を施した際，本来変更すべきでない機能を変更してしまったり，変更内容に誤りや不備があるなど，変更によって新たな不具合を引き起こしてしまうことがある．例えば，あるソースコードに対して変更を行ったとき，ソースコードの振る舞い自体には何の問題も無くとも，その内容が定められた仕様に反してしまっている場合などが考えられる．ソースコードに対してこのような問題を引き起こす変更を行ってしまう可能性は変更を施す箇所によって異なるが，このような可能性を事前に知ることはソフトウェアに対する十分な理解を必要とする．ソフトウェアの全体像の把握が困難な新規開発者にとって，これは大きな問題である．

一方，ソフトウェアを効率よく管理する為に，近年のオープンソースソフトウェア開発では，版管理システムを用いることが多くなっている．版管理システムは，プロダクトの開発履歴をリポジトリと呼ばれるデータベースに格納して管理する．版管理システムでは開発者がソフトウェアに対して行った変更の履歴が全て個別のアーカイブに保存されており，その履歴の中には将来の開発に活用することの出来る情報が多く蓄積されている．版管理システムのアーカイブ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，開発の手助けになることが知られている [1]．従って，この版管理システムに蓄積された変更の履歴を解析することで，ソフトウェアに対して変更を行う際に注意を必要とする箇所の特定が可能であると期待できる．

本研究では版管理システムを利用し，変更によって問題が発生する可能性を事前に把握する為の手法を提案する．具体的には，リポジトリに蓄積されたソフトウェアの開発履歴を解析し，ソースコードの行毎の「変更危険度」を計測し，計測結果の表示を行う．変更危険度とは「変更によって問題が発生する可能性の大きさ」を表す．変更危険度を示すことにより，ソースコードの変更を行う際，開発者に注意を喚起することを目的としている．また，手法を実装するシステムの試作を行う．

また，実際に開発で版管理システムを使用したソフトウェア開発に対して本システムを適用し，評価を行なった．その結果，本システムを用いることで開発者はソースコードの変更時において注意すべき箇所を知ることができることが確認できた．このことにより，開発者

は注意すべき箇所に対する変更を施す際に，より慎重な行動を喚起され，ソフトウェアの開発生産性の向上が期待できる．

以降，2 節ではオープンソースソフトウェア開発とその開発環境及びその問題点について述べ，3 節ではソースコードの危険度の計測についての関連研究とその問題点について述べる．4 節では変更危険度の定義や計算法など，本研究で提案する手法について述べる．5 節では試作したシステムの概要と実装について述べ，6 節では評価実験について述べる．最後に 7 節で本研究のまとめと今後の課題について述べる．

2　オープンソースソフトウェアとその開発環境

本節では，オープンソースソフトウェアとその開発環境について述べ，オープンソースソフトウェアの持つ問題点を説明する．

2.1　オープンソースソフトウェア

開発中のソースコードやドキュメント等のプロダクトを広く公開して，複数の開発者が並列的にソフトウェアの開発作業を行う開発手法はオープンソースソフトウェア開発と呼ばれている [2][3]．また，オープンソースソフトウェア開発によって開発されたソフトウェアをオープンソースソフトウェアと言う．オープンソースソフトウェア開発は，高品質で多機能なソフトウェアを開発できるとして注目を集めている．

オープンソースソフトウェア開発では，世界中に分散した各開発者が，インターネットに代表される大規模ネットワークを使って開発作業を行う．そのため，開発者はいつでも自由に開発作業に参加することが可能である．FreeBSD[4] や Linux[5]，Apache[6] 等は，オープンソースソフトウェアの代表である．

2.2　オープンソースソフトウェア開発環境

オープンソースソフトウェア開発では，各開発者がそれぞれ分散して並列的に開発作業を行うことが可能である．その一方で，開発中のソースコードやドキュメント等のプロダクトを広く公開するため，それらの管理を行う必要がある．そこで，オープンソースソフトウェア開発に参加する開発者は，オープンソースソフトウェア開発環境と呼ばれる環境の中でプロダクトの管理を行う．

オープンソースソフトウェア開発環境の構成例を図 1 に示す．オープンソースソフトウェア開発環境は，一般に複数のシステムから構成される．図 1 の構成例の場合，ソースコードやドキュメント等のプロダクトは，版管理システム [7] の一つである Concurrent Versions System(CVS)[8] [9][10] を用いて管理される．それらのプロダクトは，rsync や ftp を利用して，各開発者に複製，配布される．また，開発者間で相互に行われる意志疎通の手段として，電子メールやメーリングリストが用いられる．その内容はアーカイブとして保存され，World Wide Web(WWW) を用いた検索エンジンによって自由に検索や閲覧が可能である．開発者からのバグ報告等フィードバックは，GNU Problem Report Management System(GNATS) を用いたバグデータベースによって管理される．

以下では，これらのシステムの中から，オープンソースソフトウェア開発で用いられる版管理システムとバグ追跡システムについて説明する．

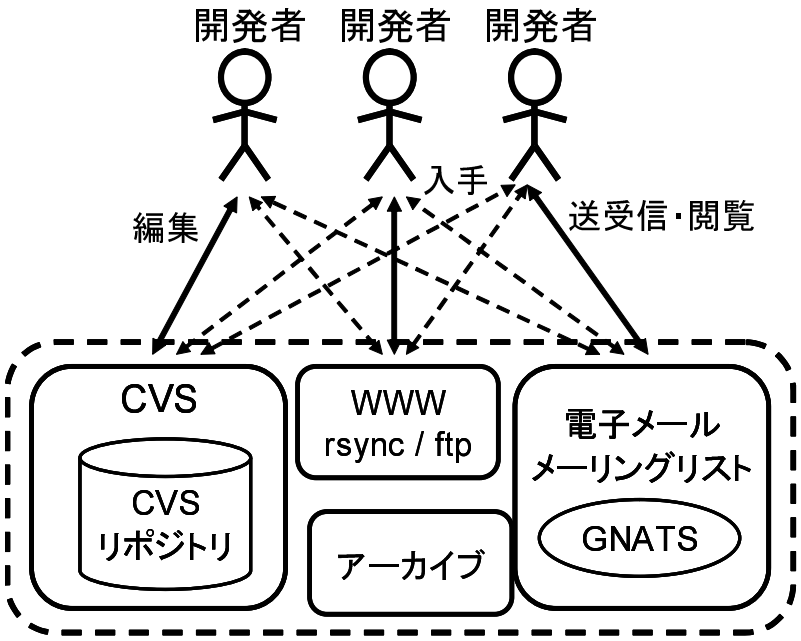

図 1: オープンソースソフトウェア開発環境の構成例

2.2.1 版管理システム

版管理とは，主として以下の 3 つの役割を提供する機構である．

- 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．

各プロダクト (ソースコード，リソースなど) の履歴データは，**リポジトリ (Repository)** と呼ばれるデータ格納庫に蓄積される．その内部では，プロダクトのある時点における状態である**リビジョン (Revision)** を単位として管理する．１つのリビジョンには，ソースコードやリソースなどの実データと，作成日時やメッセージログなどの属性データが格納されている．

また，リポジトリとのデータ授受をする為に，開発者はシステムに依存したオペレーション (operation) を利用する必要がある．

版管理手法を述べるにあたり，その基礎となるモデルが数多く存在する [11][12] ．本節では，多くの版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．なお，以降本文において，プロダクトのある時点における状態のことをリビジョンと呼ぶことにする．

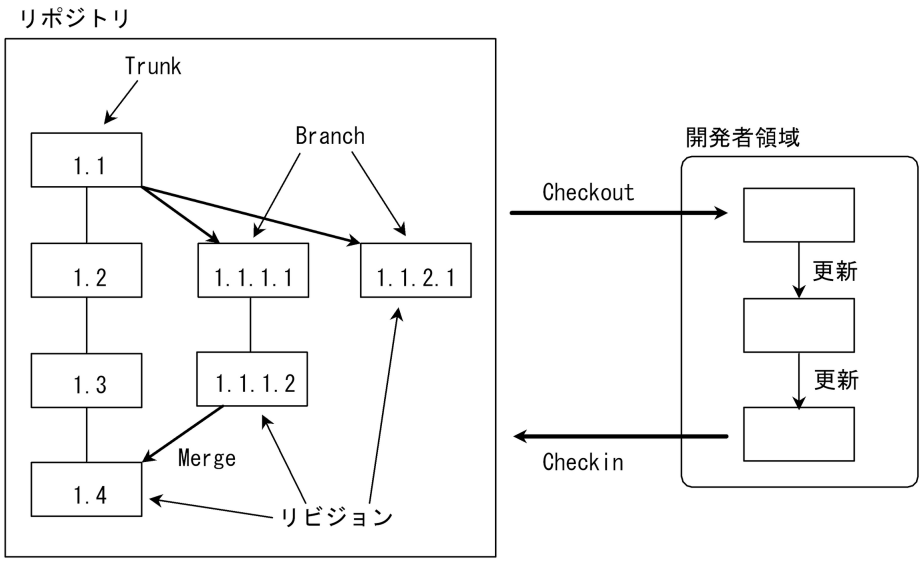

図 2: Checkout/Checkin Model

The Checkout/Checkin Model

このモデルは，ファイルを単位としたリビジョン制御に関して定義されている (図 2 参照).

リビジョン管理下にあるコンポーネントは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．開発者のそれらのファイルを直接操作するのでなく，各システムに実装されているオペレーションを介して，リポジトリとのデータ授受を行う．リポジトリより特定のリビジョンのコンポーネントを取得する操作を チェックアウト (Checkout) という．逆に，データをリビジョン に格納し，新たなリビジョンを作成する操作を チェックイン (Checkin) という．

単純にリビジョンを作成するのみでは，分岐のない一直線状のリビジョン列を生成することになる．しかし，過去のリビジョンに遡り，別の工程で開発を行う場合 (例えば，デバッグ) 等の為，リビジョン列を分岐させるには，ブランチ (Branch) という操作により，ブランチを生成し，その上にリビジョンを作成するという手法を採る．図 2 では，リビジョン 1.1.1.1, 1.1.1.2 およびリビジョン 1.1.2.1 がブランチである．このブランチに対して，元のリビジョン列のことを**トランク (Trunk)** という．図 2 では，リビジョン 1.1，1.2，1.3，1.4 がトランクである．また，ブランチ上での作業内容 (デバッグ修正部分等) を，別のリビジョンに統合する作業を マージ (Merge) という．このように，リビジョン列は木構造になることからリビジョンツリー (RevisionTree) という．リビジョンツリーの例を図 3 に示す.

版管理システムと呼ばれるものは，多数存在する．UNIX 系 OS では，多くの場合，RCS[13]

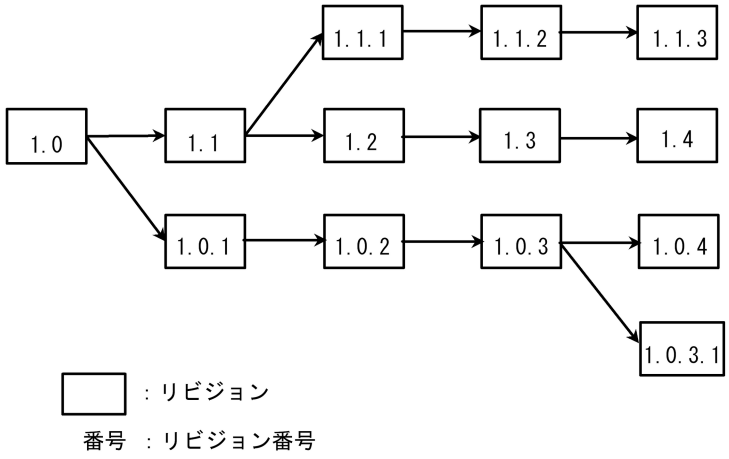

図 3: リビジョンツリーの例

や CVS 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，版管理システムのうちの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，版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 同様，UNIX 上で動作するシステムとして構築された版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，開発者間で変更の内容が衝突した場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やグループウェアの場面で活躍の場が多い．その最たる例が，FreeBSD や OpenBSD 等のオペレーティングシステムの開発である．

CVS は，GUI やウェブインターフェイス，エディタ等様々な目的に応じて，その関連ツールが数多く開発されており，実際に利用されている．例えば，CGI を利用した CVS のウェブインターフェイスとして CVSweb[18] がある．CVSweb を利用することにより，リポジトリ内に存在するファイル一覧や，各リビジョンのデータ，リビジョン間の差分等を既存のウェブブラウザで閲覧することが可能である．同様のシステムとして，Bonsai[19] 等もある．

2.2.2 バグ追跡システム

バグ追跡システムは開発されたプロダクトのバグ報告や機能の追加要求を管理するためのシステムである．これらのシステムでは，開発者が投稿した障害情報をデータベースに蓄積する．他の開発者らは，それらの問題を閲覧することで，修正が行えると思えばそれを引き受け，修正作業を行う．さらに，それ以外の開発者も，過去の解決された事例を閲覧することで，自身の問題解決に役立てることが可能である．

UNIX 系 OS 上での開発で用いられるバグ追跡システムには，GNATS や Bugzilla といったものが挙げられる．これらのシステムでは，まず，起草者がバグ報告や作成したプロダクトの機能に対して追加要求を行う．それらを基に担当の管理人が決定され，彼らが修正担当者を割り当て，修正担当者が実際に要求の解決を図る．これらのシステムも，先の述べた版管理システムと同様にネットワークを通してバグ情報の投稿や抽出を行うことが一般的に行われており，開発作業の促進に繋がっているといえる．

2.3 オープンソースソフトウェア開発の問題点

ここまでオープンソースソフトウェア開発について説明してきたが，オープンソースソフトウェア開発には問題点がある．

オープンソースソフトウェア開発では，複数の開発者が自由に各々の開発作業を行う．これらの開発者は，普段から開発作業に専念するのではなく，個人的な時間を利用して作業を行うことが多い．従って，ソフトウェアの理解に費やす時間が限られており，十分な理解をすることが難しい．さらにソフトウェアが複雑化するに従い，より理解に必要な情報量が増大する．つまり，開発が大規模で，新規に参加した開発者であるほど開発の全体像を把握しにくくなる傾向がある．

また，開発過程で機能の追加・変更あるいは不具合の修正等の必要から，開発者はしばしばソースコードに何らかの変更を施す．その際，変更によって新たな問題が引き起こされて

しまうことがある．例えば，仕様で定められた変数の初期値を変更してしまったり，他の開発者が新しい機能を実装するため頻繁に更新を行っている箇所に別の意図で変更を行ってしまう等の事態が起こり得る．このようなソースコードに対する変更によって問題が発生する可能性は変更を施す箇所によって異なるが，変更が難しい箇所を事前に知ることはソフトウェアに対する十分な理解を必要とする．ソフトウェアの全体像の把握が困難な新規開発者にとって，このことは大きな問題である．

3　関連研究

本節では，本研究の関連としてソフトウェアを変更する際に参考となる指標を求める既存手法を紹介し，それぞれの手法が抱える問題点を挙げる．

3.1　ソースコードのみを利用した手法

ソースコードのみを利用した手法には，ソフトウェアメトリクス [23] と呼ばれるものがある．メトリクスの種類は多岐に渡り，またメトリクスツールも Eclipse[24] のプラグインとして開発された Eclipse Metrics Plugin を始め様々に存在する．サポートするメトリクス，対応プログラミング言語，出力形式等はメトリクスツールによって異なるが，ほとんどのメトリクスツールがコード行数 (Line of Code)，メソッドの凝集度の欠落 (Lack Of Cohesion Of Methods)，複雑度 (Complexity)，結合度 (Couplings) といった基本となるメトリクスに対応している．

これらのメトリクスが示す値を参考にすることで，開発者はソースコードの複雑さや機能修正や追加のしやすさ等の目安を知ることができる．例えば，凝集度とはクラスやパッケージ内の機能要素と情報要素間の関連性の強さを表す指標であるが，この凝集度が高ければ保守性が高いと判断できる．

3.1.1　ソースコードのみを利用した手法の問題点

以下にソースコードのみを利用した既存手法の問題点を挙げる．

- 実際の開発状況を反映できない．

  ソースコードのみを利用することは，手法を適用できるソフトウェアが非常に多いという大きな利点に繋がっている．しかし，必然的に開発の表面しか見ることができないため，対象のソースコードに至る過程を一切反映できないという問題がある．例えば，メトリクスを用いた結果が同じであったとしても，開発開始以来一度も変更されていないソースコードと頻繁に更新が行われたソースコードの危険度が同じであるとは考えにくい．

- 対応プログラミング言語が限定されている．

  対応プログラミング言語が限定されるのは，構文解析を行うためや，手法自体がオブジェクト指向言語にしか適用できないといった理由による．

  しかし，対応プログラミング言語が限定されていると，対応外のプログラミング言語で記述されたソースコードに手法を適用できないという問題点がある．

3.2 版管理システムを利用した手法

ここでは IVA[25] と HATARI[26] というツールを紹介する．

IVA

IVA は版管理システムを利用し，ソフトウェア中の不安定な箇所を特定することを目的としたツールである．不安定な箇所とは，ソフトウェア開発において度々同時に更新されるメソッド群やクラス群を意味する．

IVA が用いる大まかな手法は，版管理システム CVS から更新履歴情報を解析し，2 つ以上の要素が繰り返し同時に更新されている回数を測定するというものである．ここで言う要素とは，オブジェクト指向ソフトウェアのメソッドやクラス等を指す．このようにして測定した要素間の同時更新回数が特に多いものを不安定な箇所と定めている．

不安定な箇所は将来においても頻繁に更新されることが考えられる．IVA は不安定な箇所を開発者に示すことにより，不安定な箇所の改善を促し，将来の保守効率の向上が期待できるとしている．

HATARI

HATARI は Eclipse のプラグインとして開発され，ソフトウェアの要素についてのバグ発生危険度を測定することを目的としたツールである．ここでいう要素とは，IVA と同様にオブジェクト指向ソフトウェアのメソッドやクラスのことを指す．

HATARI は版管理システムとバグ追跡システムを関連付けることによってソフトウェアの要素についてバグ発生危険度を測定している．大まかな手法の手順は次のようになる．

1. バグ追跡システムを解析し，バグが報告された日時とバグが修正された日時を取得
2. 版管理システムのソースコード更新履歴情報を解析し，バグが修正された日時の情報を基に，バグ修正の為に変更された要素の候補を選定
3. 版管理システムのソースコード更新履歴情報を解析し，バグが報告された日時の情報を基に，バグを発生させる原因となった変更が行われた要素の候補を選定
4. バグ修正の為に変更された要素の候補とバグを発生させる原因となった変更の要素の候補から，共通するものを特定
5. 特定された要素をバグを発生させた変更が行われた要素とし，その要素のバグ発生危険度を次の数式を用いて測定

$$\text{バグ発生危険度} = \frac{\text{その要素のバグを発生させた更新回数}}{\text{その要素の全体の更新回数}}$$

HATARI はバグ発生危険度を開発者に提示することにより，バグ発生危険度の高い要素に対する変更を避けたり，より慎重に変更を行うように注意を喚起できるとしている．

3.2.1 版管理システムを利用した手法の問題点

以下に版管理システムを利用した手法の内，上記 2 つの既存手法についての問題点を挙げる．

- 古い情報と新しい情報が区別されていない．

  IVA と HATARI はどちらも古い情報と新しい情報の重要さを区別していない．例えば，HATARI において数年前にバグを発生させた変更と 1 週間前にバグを発生させた変更の重要さは特に区別されていない．新しい情報の方が重要さは大きいと考えられることから，時間的な要素は考慮すべきである．

- 手法を適用できるソフトウェアが少ない．

  これは HATARI の手法について言えることである．

  HATARI はバグ追跡システムという特別な情報源を用いているが，バグ追跡システムを開発作業に使用していないソフトウェアは珍しくない．つまり，バグ追跡システムを利用していることを前提条件とすると手法を適用できるソフトウェアが少なくなるという問題がある．

4　提案手法

本節では，本研究で提案する変更危険度を計測する手法について述べる．以下では，まず「変更危険度」など本研究で用いる用語について説明し，次に手法の概要を述べる．

4.1　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．

- 更新履歴情報
- 変更危険度
- 更新パターン

4.1.1　更新履歴情報

更新履歴情報とは，現行のソースコードに至るまでの全ての更新についての情報である．この情報は，版管理システムのリポジトリに蓄積された開発履歴情報から得られる．

更新履歴情報には全てのリビジョンについての情報が含まれ，その内容は以下のようになる．

- リビジョン番号
- 更新を行う 1 つ前のリビジョン番号
- 更新を行う 1 つ前のリビジョンとの差分情報
- ブランチ情報
- 更新日時
- 更新者

4.1.2　変更危険度

変更危険度とは「変更によって問題が発生する可能性の大きさ」を表す．

しかし，変更危険度は単にソースコードが複雑であったり，バグが発生しやすいことを示すものではない．例えば，開発者によってある変更がなされたとき，ソースコードの振る舞いとしては問題がなくとも，仕様と異なるために問題となることがある．あるいは，ある開発者が新規の機能を追加するため現在頻繁に更新作業を行っている箇所に，他の開発者が別

の意図で変更を加えると開発を混乱させてしまう．変更危険度はこのような問題も含めて，「変更によって問題が発生する可能性の大きさ」を表している．

本手法ではトランク上の最新リビジョンにおけるソースコードの各行に対して変更危険度を計測する．

4.1.3 更新パターン

更新パターンとは現行のソースコードに至るまで「どのような更新を行われてきたか」という履歴にみられるパターンである．

本手法では更新パターンを分析することにより変更危険度を計測する．今回用いる更新パターン以下の 3 種である．

- 一度変更した内容が後になって戻されているパターン (Undoing pattern)

  以後，このパターンを**更新パターン U** と記載する．

  このパターンが見られるとき，一度変更を行ったものの，その変更内容に問題があり，変更前のソースコードの状態に戻さなければならなかった，あるいは結果的に戻すことになってしまったという事態が発生したと考えられる．

  すなわち，過去に問題のある変更を行ってしまったと考えられる．そのため，将来変更を行った場合，同様の問題を引き起こす可能性がある．

  従って，このパターンが見られる箇所では変更危険度の値は大きいと考える．

- 変更されている量が多いパターン (large change Amount pattern)

  以後，このパターンを**更新パターン A** と記載する．

  このパターンが見られるとき，単純に更新が頻繁に行われため，結果的に変更された量が多くなっている場合がある．このようなときは更新パターン F が同時に見られる．

  しかし，更新が頻繁に行われている訳ではない場合にこの更新パターン A が見られると，何度か大きな変更が行われていると考えられる．つまり，重大な不具合が発生してしまったか，あるいは大きな機能の追加や修正などのためにソースコードを大きく変更する必要があった可能性がある．一般にこのような箇所に対する変更は注意を要すと考えられる．

  従って，このパターンが見られる箇所では変更危険度の値は大きいと考える．

- 頻繁に更新されているパターン (high change Frequency pattern)

  以後，このパターンを**更新パターン F** と記載する．

  このパターンが見られるとき，その箇所では今後も変更が行われる可能性が高い．

特にこのパターンが最近の開発過程において見られる場合は，何らかの機能が開発途中であると考えられ，安易に変更を行うことは危険である．

従って，このパターンが見られる箇所では変更危険度の値は大きいと考える．

本手法ではこれらの更新パターンそれぞれについて変更危険度を計測する．

以降では更新パターン U から計測される変更危険度を **CR(U)**，更新パターン A から計測される変更危険度を **CR(A)**，更新パターン F から計測される変更危険度を **CR(F)** と記載する．

4.2 提案手法の概要

本手法は，更新作業において特に注意を要す箇所を示し，開発者を支援することを目指す．

具体的には，トランク上の最新リビジョンにおけるソースコードの各行に対し，その変更危険度を示す．そして開発者に変更危険度を示すことにより，変更を行う際に注意を喚起する．例えば，ある開発者が変更を行おうとする箇所の変更危険度の値が大きければ，他の変更手段を検討したり，他の開発者と相談する等，慎重な行動の喚起が期待できる．

提案手法の概要を図 4 に示す．前提となるのは，対象のソフトウェアが開発に版管理システムを利用していることである．

本手法の手順は次のようになる．

1. 分析対象のソースコードファイルについて，リポジトリを解析し，更新履歴情報を得る．また，更新履歴から過去の更新の過程を導出する．
2. 過去の更新の過程から 3 種の更新パターン，すなわち更新パターン U, 更新パターン A, 更新パターン F の検出を行う．
3. 更新パターンの検出結果を 3 種の更新パターンそれぞれに統合し，各更新パターンに対応した 3 種類の変更危険度 CR(U),CR(A),CR(F) を計測する．このとき，3 種の変更危険度はトランク上の最新リビジョンにおけるソースコードの各行に対して計測される．

以下では，これらの手順についてそれぞれ説明する．

4.2.1 更新履歴情報の取得

リポジトリを解析し，分析対象のソースコードファイルに対応した更新履歴情報を取得する．

また，取得した更新履歴情報の内，リビジョン間の差分情報から削除，追加，変更が行われた箇所を特定し，トランク上の全てのリビジョンの更新過程を追跡可能にしておく．具体的には，更新が行われた前後のリビジョン間で行の対応付けとその行の内容を保存する．

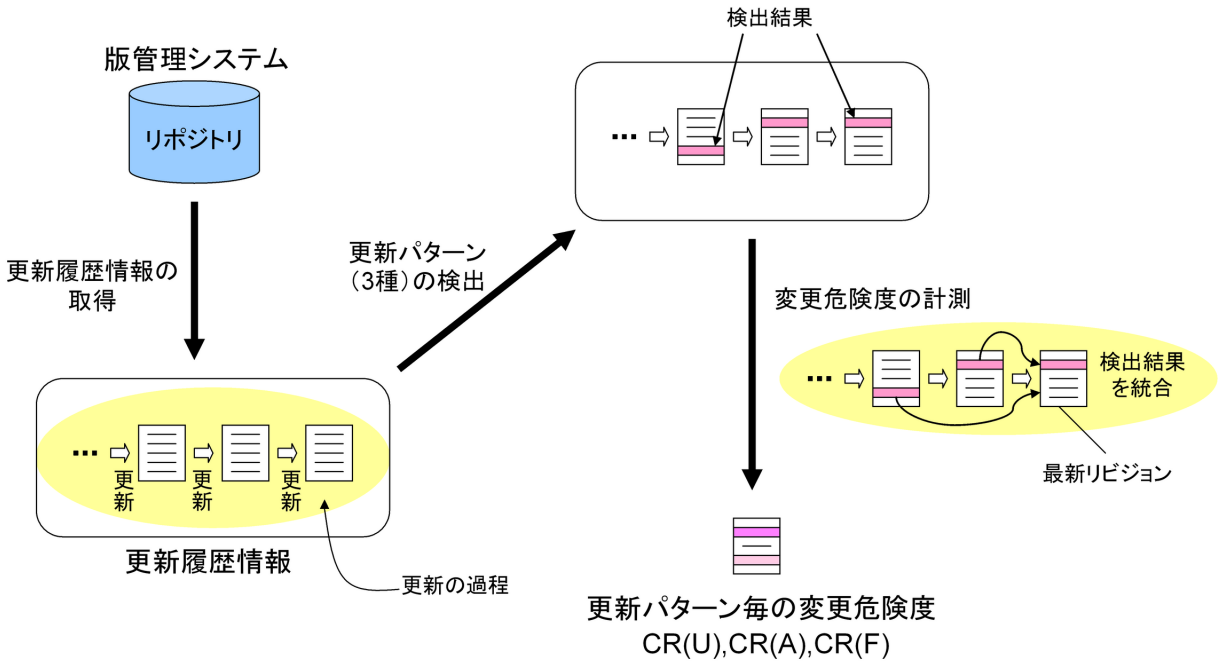

図 4: 提案手法の概要

行の対応付けは，実際は行の対応だけでなく行と行間の対応を考える．行間の対応も考慮するのは，リビジョン間の対応関係を正確に追跡できるようにするためである．

行（及び行間）の対応関係の例を図 5 に示す．図 5 はあるソースコードファイルのリビジョン 1.5 から 1.6 への更新を表している．この更新では，追加，削除，変更が行われている．

以下で図 5 を例として行の対応関係の説明を行うが，その際に用いる表記方法について説明しておく．なお，この表記方法は本節以降でも用いる．

- 行 a（a は行番号）

  行 a とはソースコードの a 行目を意味する．

  例えば，行 10 とはソースコードの 10 行目のことである．

- 行間 a+（a は行番号）

  行間 a+は行 a とその次の行 (a+1) の行間を意味する．

  例えば，行 10 と行 11 の間は行間 10+である．

- 行 a〜行 b（a,b は行番号）

  行間を含め，行 a から行 b までの範囲を意味する．

  例えば，行 10〜行 15 と記載した場合，その範囲には行 10，行間 10+，行 11，行間 11+，行 12，行間 12+，行 13，行間 13+，行 14，行間 14+，行 15 を含む．

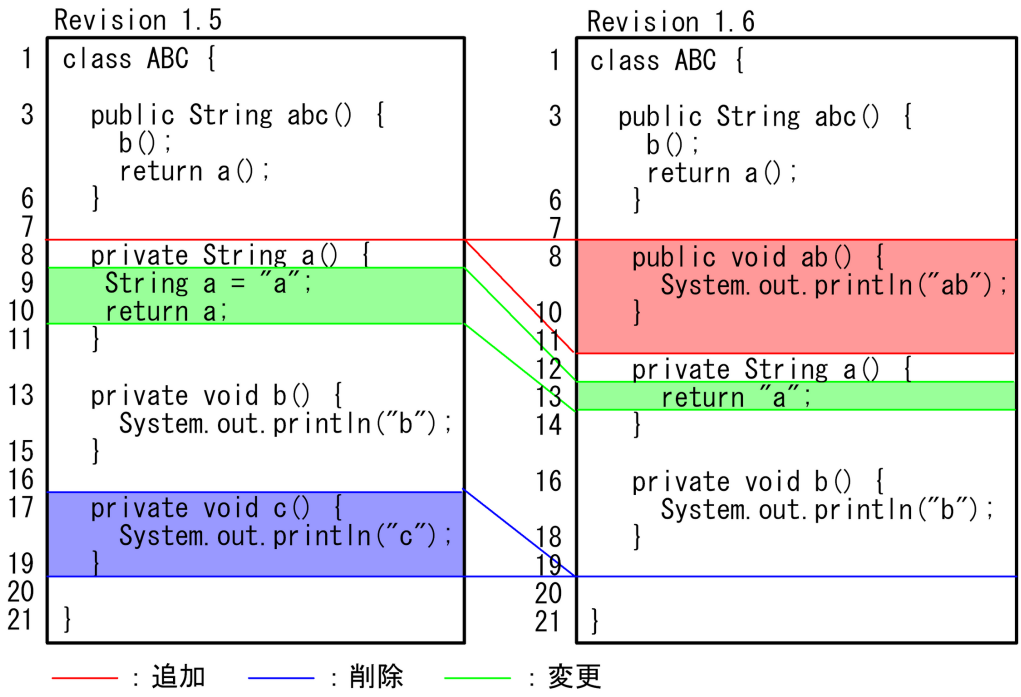

図 5: 行（及び行間）の対応関係の例

- 行間 a+～行間 b+（a は行番号）

  行 (a+1)～行 b に行間 a+と行間 b+を加えた範囲を意味する．

  例えば，行間 10+～行間 15+と記載した場合，その範囲には行間 10+，行 11，行間 11+，行 12，行間 12+，行 13，行間 13+，行 14，行間 14+，行 15，行間 15+を含む．

以下では図 5 を例として追加，削除，変更された箇所，及び変化のない箇所のそれぞれについて行の対応関係の説明を行う．

- 追加

  図 5 における追加は，リビジョン 1.5 の行間 7+に対して行われている．

  このリビジョン 1.5 の行間 7+は，図 5 が示すようにリビジョン 1.6 の行 8～行 11 に対応している．しかし，リビジョン 1.5 の行間 7+の対応範囲をリビジョン 1.6 の行 8～行 11 と定めてしまうと問題がある．この場合，リビジョン 1.6 の行間 7+，行間 11+に対応するリビジョン 1.5 の行または行間が存在しないことになってしまう．

  これを回避するために本システムでは，図 5 の追加において「リビジョン 1.5 の行間 7+」に対応する範囲は「リビジョン 1.6 の行間 7+～行間 11+」と定める．

- 削除

図 5 における削除は，リビジョン 1.5 の行 17〜行 19 に対して行われている．

このリビジョン 1.5 の行 17〜行 19 は，図 5 が示すようにリビジョン 1.6 の行間 19+に対応している．しかし，追加の場合と同様，このまま対応付けると問題がある．この場合，リビジョン 1.5 の行間 16+と行間 19+に対応するリビジョン 1.6 の行や行間が存在しないことになってしまう．

したがって本システムでは，図 5 の削除において「リビジョン 1.5 の行間 16+〜行間 19+」に対応するのは「リビジョン 1.6 の行間 19+」と定める．

- 変更

  図 5 における変更は，リビジョン 1.5 の行 9〜行 10 に対して行われている．また，変更後はリビジョン 1.6 の行 13 になっている．

  本システムでは，変更は削除と追加の組み合わせだと考える．すなわち，図 5 における変更は，まずリビジョン 1.5 の行間 8+〜行間 10+が削除されたと見なす．そして削除された行間に対してリビジョン 1.6 の行間 12+〜行間 13+が追加されたと考える．

  したがって本システムでは，図 5 の変更において,「リビジョン 1.5 の行間 8+〜行間 10+」に対応するのは「リビジョン 1.6 の行間 12+〜行間 13+」と定める．

- 変化なし

  追加も削除も変更も行われていない行でも，対応する行番号がずれることがある．

  例えば，図 5 におけるリビジョン 1.5 の行 13 に対応しているのはリビジョン 1.6 の行 13 ではない．実際に対応しているのはリビジョン 1.6 の行 16 である．

  このような行番号のずれは，その行より上で追加や削除，変更が行われている場合に発生する．

4.2.2　更新パターンの検出

更新履歴情報から得られた過去の更新過程を追跡することにより，更新パターン U，更新パターン A，更新パターン F の検出及び検出箇所に変更危険度を計測する際に用いる値の記録を行う．

以下ではそれぞれの更新パターンを検出する方法及び値の記録方法について説明する．

- 更新パターン U

  図 6 及び図 7 はこのパターンの例を示したものである．

  過去の更新過程のある箇所に次のような手順の更新が見られたとき，一度変更した内容が後になって戻されていると判断する（図 6）．

  1. あるリビジョン A において行間 a+〜行間 (a+i)+までが全て変更または削除される．

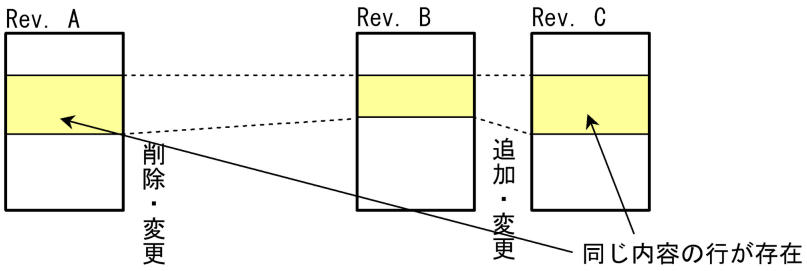

図 6: 一度変更した内容が後になって戻されているパターンの例 1

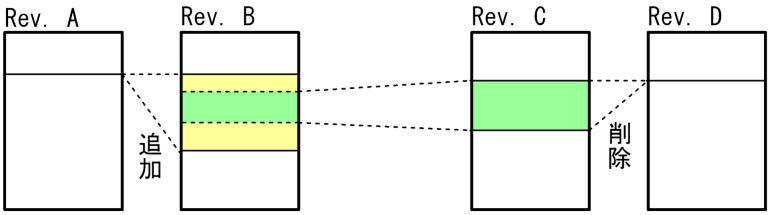

図 7: 一度変更した内容が後になって戻されているパターンの例 2

2. リビジョン A の行間 a+～行間 (a+i)+が，リビジョン A 以降のリビジョン B の行間 b～行間 (b+j)+に対応している．このとき，リビジョン B の行間 b+～行間 (b+j)+の範囲に対して変更または追加が行われる．
3. リビジョン B 行間 b+～行間 (b+j)+が，リビジョン B の次のリビジョン C の行間 c+～行間 (c+k)+に対応している．このとき，リビジョン C の行間 c+～行間 (c+k)+の範囲に含まれる行で，リビジョン A の行間 a+～行間 (a+i)+に含まれる行と同じ内容のものが存在している．

この場合，リビジョン A の行間 a+～行間 (a+i)+に含まれる行と同じ内容を持つリビジョン C の行間 c+～行間 (c+k)+の範囲に含まれる行それぞれに，値を 1 として記録する．

また，次のような手順の更新が見られたときも一度変更した内容が後になって戻されていると判断する（図 7）．

1. あるリビジョン A において行間 a+に対して追加が行われる．追加後はリビジョン A の次のリビジョン B の行間 b+～行間 (b+i)+となる．
2. リビジョン B の行間 (b+j)+～行間 (b+k)+が，リビジョン B 以降のリビジョン C の行間 c+～行間 (c+l)+に対応している．リビジョン B の行間 (b+j)+～行間 (b+k)+

表 1: 変更量と値の記録

更新作業	更新前	更新後	総変更量	値の記録
追加	行間	m 行	m	更新後のリビジョンの m 行 それぞれに値 1
削除	n 行	行間	n	更新後のリビジョンの行間に値 n
変更	n 行	m 行	$n+m$	更新後のリビジョンの m 行 それぞれに値 $\frac{n+m}{m}$

※ n,m は正数

は，行間 b+〜行間 (b+i)+の一部あるいは全てである．(ただし 0≤j<k≤i)

3. リビジョン C の行間 c+〜行間 (c+l)+が全て削除され，リビジョン D の行間 d+になる．

この場合，リビジョン D の行間 d+に，値を $\frac{k-j}{i}$ として記録する．i はリビジョン A の行間 a+に対応するリビジョン B の行数，$k-j$ はリビジョン D の行間 d+に対応するリビジョン B の行数である．

- 更新パターン A

  このパターンは，更新パターン U と異なり，パターンの存在の有無を明確に決定できない．したがって本手法では，過去の更新過程から全ての変更においてその変更量の多さを求める．これを用いて更新パターン A による変更危険度を測定する．

  更新パターン A を検出した値は表 1 に示すように記録する．

- 更新パターン F

  このパターンも更新パターン A と同様，パターンの存在の有無を明確に決定できない．したがって本手法では，過去の更新過程において，ある箇所の更新が行われたかどうかを判定する．判定には，ある箇所の範囲内で更新パターン A を検出した値の合計が 0 より大きいかどうかを調べる．更新が行われていれば値を 1 とし，行われていなければ値を 0 とする．

  このように更新パターン F の検出した値は更新パターン A の検出した値から求める．

4.2.3 変更危険度の計測

変更危険度はトランク上の最新リビジョンの各行について，各更新パターン毎にそれぞれ CR(U),CR(A),CR(F) が計測される．これらの変更危険度の計測において，本手法では時間的

な重みを考慮する．具体的な変更危険度の計測方法を説明する前に，この時間的な重みについて説明する．

時間的な重みを用いるのは以下のような理由による．

- 更新パターンが最近のリビジョンで検出された場合と昔のリビジョンで検出された場合では，最近のリビジョンで検出された場合のほうが最新のソースコードに与えている影響が大きいと考える．
- 過去にリファクタリング等，ソースコードの改善作業が行われた場合，その箇所の変更危険度は低下したと考えられる．時間的な重みを用いることで，この変更危険度の低下を表現することができる．

このような時間的重みの必要条件は以下のようになる．

- 昔であるほど重みが小さく，無限大に昔であれば時間的な重みは 0 に収束する．
- 時間的な重みは乗算で用いるので，例えば次のような数式が成り立つ必要がある．
  (1ヵ月前の重み) × (2ヶ月前の重み) = (3ヶ月前の重み)

本手法では時間的な重みを以下のように設定した．更新時間については更新履歴情報から得る．

$$
\begin{aligned}
&\text{時間的な重み} = (1-d)^{\frac{x}{t}} \\
&d = 0.2 \\
&x = (\text{トランク上の最新リビジョンの更新時間 [秒]}) \\
&\qquad - (\text{重み付けを行う対象のリビジョンの更新時間 [秒]}) \\
&t = \frac{60 \times 60 \times 24 \times 365}{10} [\text{秒}]
\end{aligned}
$$

この時間的な重みの意味は「重みの最大値を 1 とし，$\frac{1}{10}$ 年毎に 2 割重みが減少する」となる．

以下では，各パターン毎の変更危険度 CR(U),CR(A),CR(F) の計測方法を説明する．基本となる計測方法を図 8 に示す．CR(U),CR(A),CR(F) では，それぞれ図 8 の検出した値 v_i が異なるのみで，検出した値 v_i の統合手法は同じである．

以下の説明ではトランク上の最新リビジョンを $i = 0$ とし，トランク上の過去のリビジョンを新しいものから順に $1, 2, 3, 4, \cdots$ とする．

CR(U) の計測

基本となる計測方法は図 8 に示した通りである．具体的な手順は次のようになる．

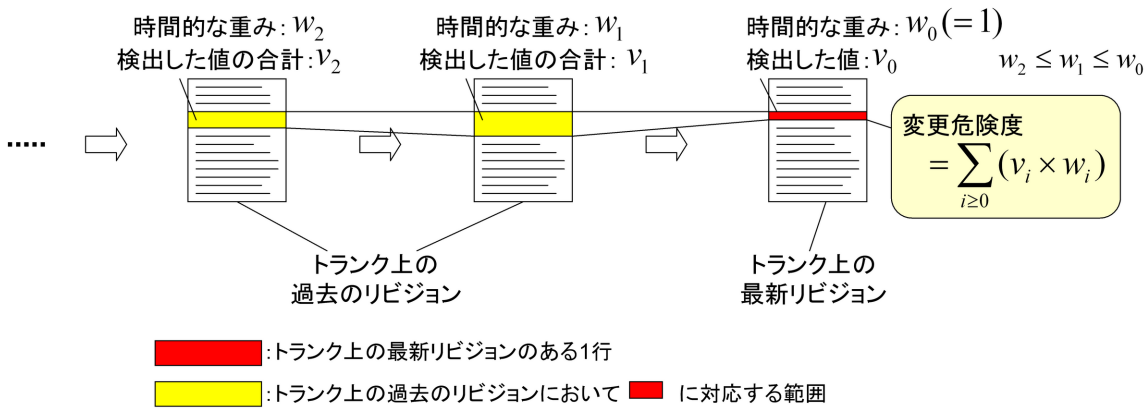

図 8: 基本となる変更危険度の計測方法

1. トランク上の各リビジョン i について時間的な重み w_i を設定する．このとき，最新リビジョンの時間的な重み w_0 は 1 とし，古いリビジョンほど時間的な重みの値を小さくする．

2. CR(U) を最新リビジョンのある 1 行に対して計測するが，以下ではその行を $target$ と記載する．$target$ に記録された更新パターン U の値を v_0 とする．値が記録されていなければ $v_0 = 0$ とする．

3. トランク上の過去のリビジョンにおいて，$target$ に対応する範囲を定める．対応する範囲は，各リビジョン間の行と行間の対応を追跡することにより求める．

4. 過去のリビジョン i の対応する範囲内の行と行間に記録された更新パターン U の値を合計する．合計した値を v_i とする．

5. 以上の値 v_i と時間的な重み w_i から，$target$ の CR(U) を以下のように計測する．

$$CR(U) = \sum_{i \ge 0} (v_i \times w_i)$$

このようにして計測された CR(U) は，ある行の内容がそれ以前の内容に戻されたことがある行数に，それぞれの時間的な重みを掛け合わせた値を意味している．

CR(A) の計測

基本となる計測方法は図 8 に示した通りである．具体的な手順は次のようになる．

1. トランク上の各リビジョン i について時間的な重み w_i を設定する．このとき，最新リビジョンの時間的な重み w_0 は 1 とし，古いリビジョンほど時間的な重みの値を小さくする．

2. CR(A) を最新リビジョンのある 1 行に対して計測するが，以下ではその行を $target$ と記載する．$target$ に記録された更新パターン A の値を v_0 とする．値が記録されていなければ $v_0 = 0$ とする．

3. トランク上の過去のリビジョンにおいて，最新リビジョンの行に対応する範囲を定める．対応する範囲は，各リビジョン間の行と行間の対応を追跡することにより求める．

4. 過去のリビジョン i の対応する範囲内の行と行間に記録された更新パターン A の値を合計する．合計した値を v_i とする．

5. 以上の値 v_i と時間的な重み w_i から，$target$ の CR(A) を以下のように計測する．

$$CR(A) = \sum_{i \geq 0} (v_i \times w_i)$$

このようにして計測された CR(A) は，最新のソースコードの行が現在の内容に至るまでに変更された行数に，それぞれの時間的な重みを掛け合わせた値を意味している．

CR(F) の計測

基本となる計測方法は図 8 に示した通りである．具体的な手順は次のようになる．

1. トランク上の各リビジョン i について時間的な重み w_i を設定する．このとき，最新リビジョンの時間的な重み w_0 は 1 とし，古いリビジョンほど時間的な重みの値を小さくする．

2. CR(F) を最新リビジョンのある 1 行に対して計測するが，以下ではその行を $target$ と記載する．4.2.2 節で述べたように，更新パターン F の検出した値は更新パターン A の検出した値を用いることによって求める．従って，まず $target$ に記録された更新パターン A の値を取得し，v_0 を以下のように定める．

$$v_0 = \begin{cases} 1 & (target \text{ に記録された更新パターン A の値が 0 より大きい}) \\ 0 & (target \text{ に記録された更新パターン A の値が 0 }) \end{cases}$$

3. トランク上の過去のリビジョンにおいて，最新リビジョンの行に対応する範囲を定める．対応する範囲は，各リビジョン間の行と行間の対応を追跡することにより求める．

4. 過去のリビジョン i の対応する範囲内の行と行間に記録された更新パターン A の値を合計し，その値に応じて v_i を以下のように求める．

$$v_i = \begin{cases} 1 & (target \text{ に対応するリビジョン i の範囲内の更新パターン A の合計値が 0 より大きい}) \\ 0 & (target \text{ に対応するリビジョン i の範囲内の更新パターン A の合計値が 0 }) \end{cases}$$

5. 以上の値 v_i と時間的な重み w_i から，$target$ の CR(F) を以下のように計測する．

$$CR(F) = \sum_{i \geq 0} (v_i \times w_i)$$

このようにして計測された CR(F) は，最新のソースコードの行が現在の内容に至るまでに変更された回数にそれぞれの時間的な重みを考慮した値を意味している．

4.3 提案手法の特徴

本提案手法において，既存の関連研究と異なる特徴として次のようなものが挙げられる．

- 更新履歴情報を利用することで，実際の開発過程を反映した変更危険度を計測する．
- ソースコードファイルがプレーンテキスト形式ならばプログラミング言語を問わず適用可能である．
- 時間的な重みを導入することにより，古い情報と新しい情報の重要さを区別する．
- 版管理システムを用いていれば適用可能であり，多くのソフトウェアに対応できる．

5　変更危険度計測システム

本節では，4 節で述べた提案手法を実現するため試作した変更危険度計測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，全体のソースコードの行数は約 6000 行となった．開発環境は以下のとおりである．

- CPU : Pentium4 2.00GHz
- RAM : 512MB
- OS : Microsoft Windows XP Professional, Version 2002, Service Pack 2
- JDK 5.0

なお，本システムで対象とするソフトウェアは開発に版管理システムとして CVS を利用している必要がある．

本システムの処理手順の概要は以下のようになる．

1. 作業用コピーやリポジトリ，分析結果の出力先等の必要な情報を設定.
2. 作業用コピー内にあるバージョン制御情報を参照．分析対象とするリポジトリに登録されたソースコードを決定する．
3. 変更危険度の計測が終了していない分析対象のソースコードファイル 1 つについて，リポジトリ内の版管理ファイルを解析．以下の情報を抽出する．
    - ソースコードの全リビジョン番号
    - リビジョン間の差分情報
    - 更新日時
    - 更新者
4. 3 でリポジトリを解析して得られた差分情報を基に，行写像追跡情報（5.2 節参照）を構築.
5. 3,4 で得られた情報を用いて更新パターンを分析．各パターン毎の変更危険度 CR(U),CR(A),CR(F) を測定する．
6. 5 で得られた変更危険度の測定結果を HTML 形式で出力．分析対象の全てのソースコードファイルの計測が終了するまで 3〜6 を繰り返す.

7. 分析対象のディレクトリ構造を基に，各ディレクトリに対応する分析結果閲覧用 HTML ファイルを出力．

本システムは大きく次の 4 つの部分から構成される．括弧内の数字は上記の処理手順に対応しており，行われる処理内容を示している．

- ユーザインターフェイス部 (1)
- データベース生成部 (2,3,4)
- 更新パターン分析部 (5)
- 変更危険度出力部 (6,7)

以下ではこれらについて説明を行う．

5.1 ユーザインターフェイス部

処理を行うのに必要な情報を設定するため，変更危険度計測システムは図 9 に示すような GUI を持つ．

この GUI は以下の 6 つの部分に分けられる．

- Input 部

  Input 部では，作業用コピーとリポジトリの場所をそれぞれ Target Directory と Repository の項目で指定する．テキストフィールドにパスを入力して場所を指定するが，ファイル選択ダイアログから選ぶこともできる．ファイル選択ダイアログは Browse ボタンを押すことで表示される．

- Output 部

  Output 部では，HTML 形式で出力される分析結果の格納先とトップディレクトリ名をそれぞれ Target Directory と Top Directory Name の項目で指定する．分析結果の格納先の入力にはファイル選択ダイアログを利用することもできる．Input の場合と同様，ファイル選択ダイアログは Browse ボタンを押すことで表示される．指定されたディレクトリが存在しない場合は自動的にディレクトリを作成する．

- Analysis Option 部

  Analysis Option 部では，分析に対するオプション機能のオン／オフを指定する．図 9 で示されるように，以下のオプション機能を設定できる．

  – 空白の数の違いを無視

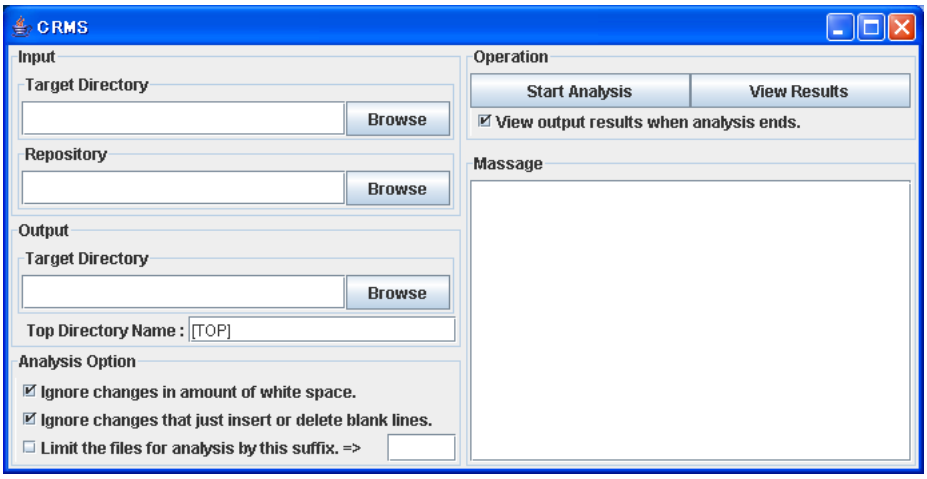

図 9: 変更危険度計測システムの GUI

- 空行の数の変化を無視
- 拡張子による分析対象の絞込み

チェックを付けた場合はその機能を使用する．なお，拡張子による分析対象の絞込みの機能を使用する場合は，拡張子の種類指定する必要がある．拡張子の種類はチェックボックスの右のテキストフィールドに入力する．

- Operation 部

  Operation 部では，Start Analysis と View Results のボタンを押すことでそれぞれ分析の実行と結果の閲覧を行うことができる．分析結果は HTML 形式で出力されるので，分析結果の閲覧はブラウザを起動させて行う．また，2 つのボタンの下のオプションにチェックを入れておくと，分析終了後自動的に分析結果の閲覧に移行する．

- Massage 部

  Message 部では，テキストエリアに分析の経過状況や入力情報の不備による警告文等を表示する．

分析結果は HTML 形式で提供されることは既に述べたが，そちらのユーザインターフェイスについては 5.4 節で述べる．

5.2 データベース生成部

データベース生成部で行う主な処理は次の 3 つである．

- 分析対象ファイルの決定
- リビジョン情報データベースの生成
- 行変遷追跡情報データベースの生成

以下ではこの 3 つの処理の実装について説明を行う．

5.2.1 分析対象ファイルの決定

分析対象ファイルはユーザが所持する作業用コピー内のバージョン制御情報から決定する．バージョン制御情報は作業用コピー内にある全ての CVS サブディレクトリの Entories ファイルから得ることができる．バージョン制御情報とは CVS で管理されているファイル名およびディレクトリ名の情報である．

なおこの分析対象ファイルの決定には，拡張子による分析対象の絞込みを行うことができる．絞込みを行う場合，指定された拡張子を持たないファイルは分析対象としない．

5.2.2 リビジョン情報データベースの生成

リビジョン情報データベースには，全ての分析対象ファイルについて，以下の情報を格納する．

- ファイル名
- 全てのリビジョンについての以下の情報
  - リビジョン番号
  - 更新を行う 1 つ前のリビジョン番号
  - ブランチ情報
  - 更新日時
  - 更新者

ファイル名以外の情報は，分析対象ファイルに対応するリポジトリの管理情報を参照し，更新履歴情報を得ることによって格納する．なおこの際，リビジョン情報データベースには更新履歴情報の「リビジョン間の差分情報」は格納しない．リビジョン間の差分情報は次節の行変遷追跡情報データベースの生成に用いられる．

5.2.3　行変遷追跡情報データベースの生成

行変遷追跡情報データベースは，1 つの分析対象ファイルに対して差分情報を基に生成される．この差分情報は，リビジョン情報データベースを生成時にリポジトリを解析した際に得られたものである．

このデータベースにはトランク上の全てのリビジョンにおける全ての行と行間の変化の過程が保存される．また，行変遷追跡情報データベースでは，行と行間の変化の過程の他に追加や変更された行の文字列も保存する．すなわち，このデータベースは過去の更新過程を保存するものである．

5.3　更新パターン分析部

更新パターン分析部で実装されている処理は以下の 2 つである．

- 更新パターンの検出
- 変更危険度の計測

以下ではこの 2 つの処理の実装について説明を行う．

5.3.1　更新パターンの検出

更新パターンの検出には，データベース生成部で生成された行変遷追跡情報データベースを利用する．

検出された箇所にはそれぞれの値を行変遷追跡情報データベースに記録する．

5.3.2　変更危険度の計測

変更危険度の計測は，行変遷追跡情報データベースを利用する．

各更新パターン毎に最新リビジョンから行と行間の対応関係を追跡し，検出時に記録された値を取得する．取得した値と時間的な重みから各変更危険度を計測する．

5.4　変更危険度出力部

変更危険度の出力は HTML 形式で行われる．変更危険度出力部で行われる処理は次の 3 つである．

- 分析対象ファイルの各行に対する変更危険度の出力
- 分析対象ファイルの各行に対する変更危険度の詳細情報の出力

- 分析結果閲覧用 HTML ファイルの出力

以下ではこれらの処理について説明する．

5.4.1 分析対象ファイルの各行に対する変更危険度の出力

更新パターン分析部で計測した変更危険度を出力する．
出力例を図 10 に示す．ソースコードの各行について出力されるのは以下の情報である．

- 行番号
- 更新パターン毎の変更危険度 CR(U),CR(A),CR(F)
- 行の文字列

また，各変更危険度はその値だけでなく，パターン毎に値の大きさに応じた色を付けている．この色により，変更危険度の値の大きい箇所が視覚的に分かりやすいように配慮されている．
彩色の基準は図 11 のようになる．この基準は更新パターン毎に個別に設けられる．CR(U),CR(A),CR(F) 毎に，ソースコードファイル内における変更危険度の最大値を赤とし，最小値を緑とする．
なお，行番号にはリンクが貼られている．リンク先の HTML ファイルでは，分析対象ファイルの各行に対する変更危険度の詳細情報が記載されている．

5.4.2 分析対象ファイルの各行に対する変更危険度の詳細情報の出力

分析対象ファイルの各行に対する変更危険度の詳細情報は更新パターンの分析情報とリビジョン情報データベース，行変遷追跡情報データベースを用いて出力される．
出力例を図 12 に示す．詳細情報は各行それぞれに対して出力されるが，その内容は以下の通りである．

- 行の文字列
- 各リビジョンのリビジョン番号
- 各リビジョンにおける CR(U),CR(A),CR(F) の値（時間的な重みを無視したもの）
- 各リビジョンにおいて対応する行と行間の範囲
- 各リビジョンにおけるソースコード全体の行数
- 各リビジョンにおける全体の変更量

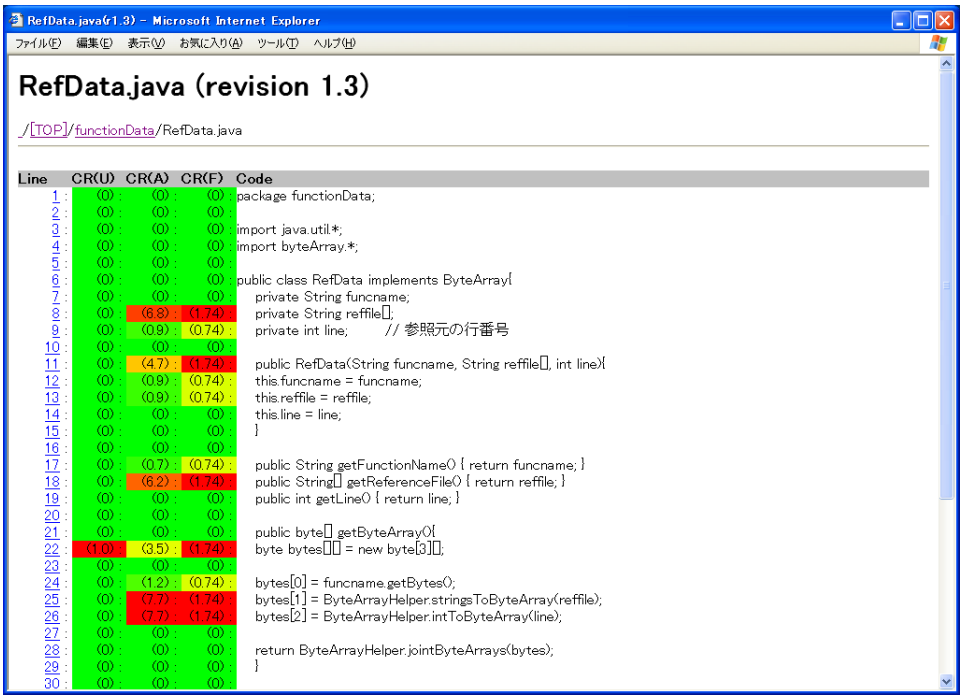

図 10: 分析対象ファイルの各行に対する変更危険度の出力の例

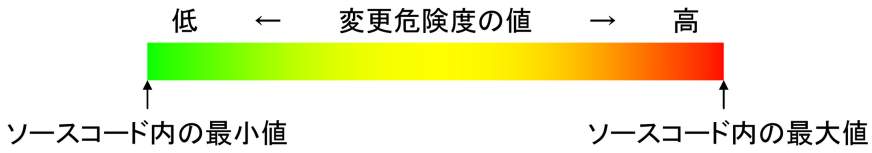

図 11: 変更危険度の彩色基準

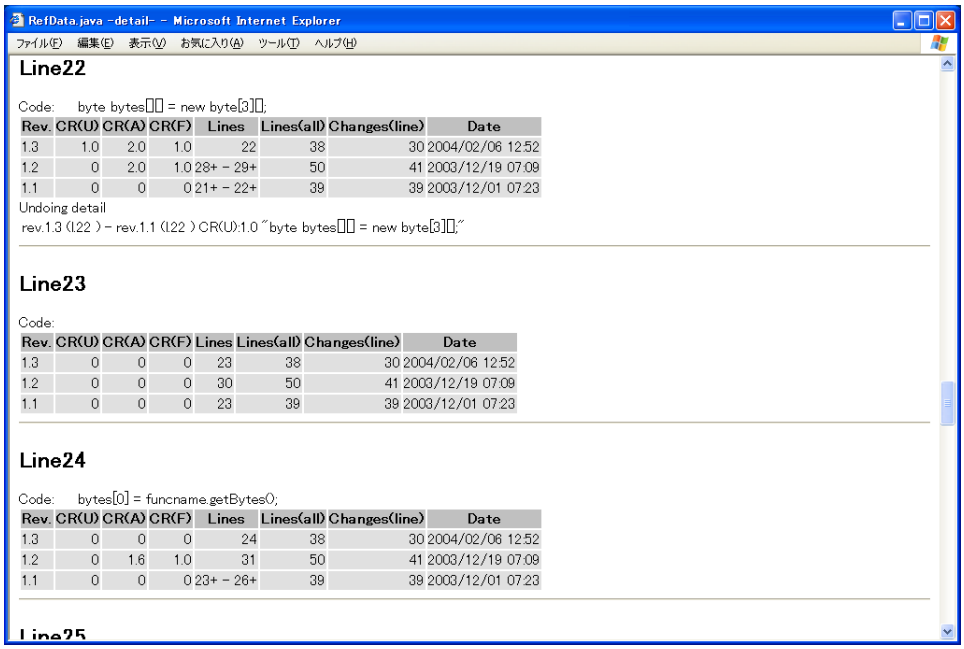

図 12: 分析対象ファイルの各行に対する変更危険度の詳細情報の出力の例

- 各リビジョンの更新日時
- 戻されたパターンについての戻された文字列
- 戻されたパターンについての戻されたリビジョンとその期間の内容

5.4.3 分析結果閲覧用 HTML ファイルの出力

この分析結果閲覧用 HTML ファイルはディレクトリ構造を反映させたものである．閲覧するファイルの決定に利用することを目的としている．

出力例を図 13 に示す．対象ディレクトリが持つファイルやディレクトリについて以下の情報が出力される．

- 対象ディレクトリ内のディレクトリ名またはファイル名
- 最新リビジョン番号（ファイルの場合のみ）
- CR(U),CR(A),CR(F) の値

  CR(U),CR(A),CR(F) の値は，ファイルの場合はそのソースコードの CR(U),CR(A),CR(F) の最大値を使用する．ディレクトリの場合は，そのディレクトリ内のファイルの CR(U),CR(A),CR(F)

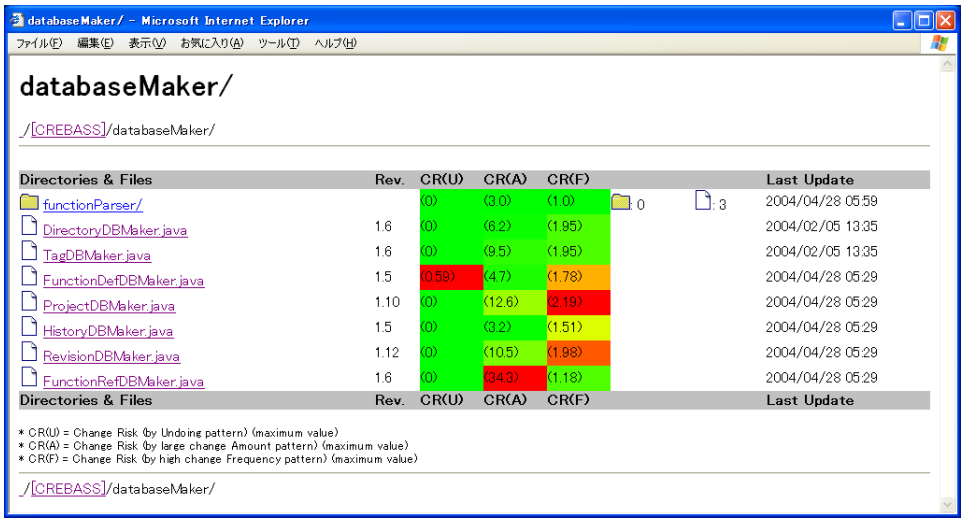

図 13: 分析結果閲覧用 HTML ファイルの出力の例

の値と，サブディレクトリの CR(U),CR(A),CR(F) の値からそれぞれの最大値を求め，その値を使用する．

- 内部のディレクトリ数（ディレクトリの場合のみ）
- 内部のファイル数（ディレクトリの場合のみ）
- 最終更新日時

  ディレクトリの最終更新日時は，そのディレクトリ内で最も新しい日付のものを使用する．

また，ファイル及びディレクトリの CR(U),CR(A),CR(F) には色を付け，大きな値を持つものを視覚的に分かりやすくしている．

ただし，各ファイル及びディレクトリの CR(U),CR(A),CR(F) はそれぞれ計測の際に基準となった最終更新日時が異なることがある．そのため，CR(U),CR(A),CR(F) に時間的な重みを掛けることで基準時刻を統一し，その値を用いて彩色を行う．彩色の方法は図 11 示したものと同様である．

6　評価実験

本節では実際に変更危険度計測システムを用い，得られた結果について述べ，考察を行う．

6.1　実験目的

本実験の目的は，変更危険度計測システムが出力する変更危険度の値を参考にすることにより，変更することによって問題が発生する危険性のある箇所を知ることが可能であるかを調べることである．

6.2　実験対象

実験対象には我々の研究チームで開発された開発履歴理解支援システム CREBASS[27] を用いる．本実験では CREBASS のソースコードの内，Java で記述されたものに対して変更危険度計測システムを適用した．

6.3　実験方法

本実験では，データベース作成の機能を実装する 8 つのソースコードファイルについて評価を行う．

まず本システムの適用結果から CR(U),CR(A),CR(F) のいずれかの値が大きい箇所を抽出した．抽出の際には，コメント部などソースコードの危険性には関係がないと判断される箇所を除いた．その結果，値の大きな箇所は合計で 26 箇所となった．

次に抽出を行った 26 箇所について，CREBASS を熟知する開発者 1 人にアンケート調査を行った．アンケートの内容は以下の通りである．

- 過去にバグが発生したことがあった
- アルゴリズムが複雑である
- 他のモジュールに影響を及ぼしやすい
- 実装方法を何度も変更した
- 上記以外の問題が過去に存在した

開発者は，このアンケートの項目に当てはまるものがあれば，そのうち最も適切であるものを選ぶ．アンケートのいずれかの項目に該当する箇所は，変更することによって問題が発生する危険性があると判断した．

表 2: アンケートの結果

	箇所数	内訳						
		U	A	F	U,F	U,A	A,F	U,A,F
過去にバグが発生したことがあった	6	0	1	5	0	0	0	0
アルゴリズムが複雑である	0	0	0	0	0	0	0	0
他のモジュールに影響を及ぼしやすい	0	0	0	0	0	0	0	0
実装方法を何度も変更した	12	0	1	3	1	0	6	1
上記以外の問題が過去に存在した	1	0	0	0	0	0	1	0
小計	19	0	2	8	1	0	7	1
該当無し	7	0	0	7	0	0	0	0
合計	26	0	2	15	1	0	7	1

上記以外の問題が過去に存在した…仕様の変更

U…CR(U) の値のみが大きい箇所

A…CR(A) の値のみが大きい箇所

F…CR(F) の値のみが大きい箇所

U,F…CR(U) と CR(F) の 2 つの値のみが大きい箇所

U,A…CR(U) と CR(A) の 2 つの値のみが大きい箇所

A,F…CR(A) と CR(F) の 2 つの値のみが大きい箇所

U,A,F…CR(U),CR(A),CR(F) の 3 つの値全てが大きい箇所

6.4 実験結果

アンケートの結果を表 2 に示す.

全 26 箇所の内，19 箇所がアンケートの項目に該当している．これより適合率は $\frac{19}{26}$ となり，およそ 0.73 となる．

6.5 考察

実験結果を見ると，適合率が 0.73 と比較的高いことが分かる．このことより，本手法が概ね有効であることが示された.

以下では各更新パターンの変更危険度 CR(U),CR(A),CR(F) について，実験結果から分かる事柄について記載する.

- CR(U)

CR(U) の値が大きい箇所は，表 2 における内訳 U, 内訳 U,F, 内訳 U,A, 内訳 U,A,F に該当する箇所である．

今回の実験では，CR(U) の値のみが大きい箇所，または CR(U) と CR(A) の値のみが大きい箇所はなかった．すなわち，CR(U) の値のみが大きい箇所では必ず CR(F) の値も大きかった．

このことは，更新パターン U が見られる可能性が低いことを表している．すなわち，更新を行った後に更新前の状態に戻すような事態になることは珍しく，またそのような事態が発生した箇所は頻繁に更新されている傾向にあると考えられる．本手法ではそのような珍しい出来事が発生した箇所を CR(U) の値として検出できていることが分かる．

また，本実験において CR(U) の値が大きい箇所は，実装を何度も変更した箇所であることから，ソースコードの振る舞いに問題があったのではなく，仕様や設計上の問題により更新を以前の状態に置き換えた箇所であると考えられる．

本実験では，CR(U) の値は過去にバグを発生した箇所の抽出に役立ってはいない．これが偶然の結果なのかどうかは更なる実験によって確認する必要がある．しかし上記の事柄より，CR(U) の値は仕様や設計上の問題による危険性を把握するのに有用であることが示されている．

- CR(A)

  CR(A) の値が大きい箇所は，表 2 における内訳 A, 内訳 U,A, 内訳 A,F, 内訳 U,A,F に該当する箇所である．

  本実験において CR(A) の値が大きい箇所は全部で 10 箇所あるが，その内 8 箇所では CR(A) だけでなく CF(F) の値も大きい．このことから，更新パターン A の見られる箇所では，同時に更新パターン F が見られやすいことが確認できる．また本実験において，このような箇所は，いずれも危険性のある箇所に該当している．

  CR(A) の値のみが大きい箇所については，本実験結果を見ると過去にバグが発生していたり，実装方法を何度も変更していることが分かる．

  しかし，全体的に CR(A) の値は CF(F) の値が大きい箇所と一致する傾向が強く，CR(A) 単体の特徴を把握するには更なる実験が必要である．

- CR(F)

  CR(F) の値が大きい箇所は，表 2 における内訳 F, 内訳 U,F, 内訳 A,F, 内訳 U,A,F に該当する箇所である．

  本実験において CR(F) の値が大きい箇所は全部で 24 箇所と最も多い．また，CR(F) の値のみが大きい箇所は 15 箇所であるが，この内 7 箇所がアンケートのどの項目にも

該当しておらず，しかもアンケートのどの項目にも該当しなかったのはこの 7 箇所だけである．

これら 7 箇所について開発者に意見を求めたところ，これらの箇所は設計やアルゴリズムの変更に伴い，最近更新を行った箇所であることが分かった．アンケート調査を行った開発者は，実質 1 人で CREBASS の開発を行っているため，CREBASS の開発過程を十分に理解しており，これらの 7 箇所には危険性がないと判断した．

しかし，最近更新を行った箇所は，開発過程を十分に理解していない新規の開発者にとっても危険性が無いとは言えない．更新が行われた意図を理解せずにこれらの箇所に変更を加えると，開発を混乱させる可能性は十分にあると考えられる．

以上のことから，CR(F) の値が大きい箇所は全ての開発者にとって変更すると問題が発生する危険性のある箇所を示している訳ではないが，CR(F) から得られる情報は十分に有用であると言える．

また，アンケート調査を行った際に「データベースを実装したソースコードファイルには，今回抽出した 26 箇所以外にも存在する」という意見を開発者から得ることができた．

これより，本手法で抽出できていない箇所が存在することが分かる．

この問題を解決するには，更新パターン U, 更新パターン A, 更新パターン F 以外に新しい更新パターンを考案する必要がある．

7　まとめと今後の課題

本研究では版管理システムのリポジトリを解析し，更新パターンを分析することで計測する変更危険度をユーザに提示することにより，更新作業において特に注意を要す箇所を特定し，ソフトウェア開発を支援する手法を提案した．また，本手法を実装する変更危険度計測システムの作成を行った．

さらに，作成したシステムを用いて評価実験を行った．本実験において，高い適合率で変更に注意を要する箇所を特定することができた．これにより，本手法はユーザのソフトウェア開発において有益な情報を提供できることが確認された．

今後の課題としては，以下のものが挙げられる．

- 構文解析の導入

  今回の評価実験においては，変更危険度の高い箇所を選択する際，コメント部などのソースコードの危険性に関係のない箇所は除いた．プログラミング言語に応じた構文解析を実行できるように変更危険度計測システムを拡張すれば，コメント部などのソースコードの危険性に関係のない箇所を判別可能になる．これにより，変更危険度計測の精度向上が期待できる．

- 新たな更新パターンの考案

  今回の評価実験時に得られた開発者の意見から，本来は変更危険度が大きいと判定されるべきである箇所が本手法では完全には抽出できていないことが分かっている．このような箇所を抽出するため，新たな更新パターンの考案が望まれる．

- 多人数で開発されているソフトウェアに対する評価実験

  今回の評価実験において実験対象とした CREBASS は実質 1 人で開発が行われていたため，今後は多人数で開発されているソフトウェアに対する評価実験も行う必要がある．

謝辞

本研究において，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 教授に深く感謝致します.

本研究において，常に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 助教授に深く感謝致します.

本研究において，様々な御指導や御助言，および評価実験への御協力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 中山 崇 氏に深く感謝いたします.

本研究において，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 楠田 泰三 氏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 井上研究室の皆様に深く感謝いたします.

参考文献

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

[2] 落水浩一郎, “分散共同ソフトウェア開発に対するソフトウェアプロセスモデルに関する基礎考察”, 電子情報通信学会技術研究報告, SS2000-48(2001-01), pp.49–56, 2001.

[3] Eric S. Raymond, “The Cathedral & the Bazaar”, O’REILLY, 1999.

[4] The FreeBSD Project, The FreeBSD Project, http://www.freebsd.org/.

[5] Linux Online Inc., The Linux Home Page, http://www.linux.org/.

[6] The Apache Software Foundation, Apache Projects, http://www.apache.org/.

[7] Jacky Estublier, “Software Configuration Management: A Roadmap”. The Future of Software Engineering in 22nd ICSE, pp.281–289, 2000.

[8] Brian Berliner, “CVS II:Parallelizing Software Development”, In USENIX Association, editor, Proceedings of the Winter 1990 USENIX Conference, pages 341–352, Berkeley, CA, USA, 1990.

[9] Karl Fogel, “Open Source Development with CVS”, The Coriolis Group, 2000.

[10] 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, December, 2000

[11] Ulf Asklund, Lars Bendix, Henrik B Christensen, and Boris Magnusson, “The Unified Extensional Versioning Model”, 9th International Symposium, SCM-9, LNCS1675, pp.100–122, 1999.

[12] Reidar Conradi and Berbard Westfechtel, “Version models for software configuration management”, ACM Computing Surveys, Vol. 30, No.2, pp.232–280, June 1998.

[13] Walter F. Tichy, “RCS - A System for Version Control”, SOFTWARE - PRACTICE AND EXPERIENCE, VOL.15(7), pp.637–654, 1985.

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

[15] Microsoft Corporation, Microsoft Visual SourceSafe, http://msdn.microsoft.com/ssafe/.

[16] Merant, Inc., PVCS Home Page, http://www.merant.com/pvcs/.

[17] Peter Fröhlich and Woflgang Nejdl, “WebRC Configuration Management for a Cooperation Tool”, SCM-7, LNCS 1235, pp.175–185, 1997.

[18] CVSWeb, http://www.freebsd.org/projects/cvsweb.html/.

[19] Bonsai, http://www.mozilla.org/bonsai.html.

[20] VA Linux Systems, Inc., SourceForge, http://sourceforge.net/.

[21] Collab. Net, Inc., SourceCast, http://www.collab.net/products/sourcecast/.

[22] Open Source Development Lab, Inc., Open Source Development Lab, http://www.osdlab.org/.

[23] B.Henderson-Sellers, ”Object-oriented metrics : measures of complexity”, Prentice Hall, 1996

[24] Eclipse, http://www.eclipse.org/.

[25] J. Bevan, E. James Whitehead, Jr. ”Identification of Software Instabilities”, Proceedings of the Tenth Working Conference on Reverse Engineering (WCRE’03), IEEE, November 2003, pp.134-143.

[26] J. Śliwerski, T. Zimmermann, A. Zeller, “HATARI: Raising Risk Awareness” In Proc. European Software Engineering Conference/ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), Lisbon, Portugal, September 2005.

[27] 中山崇, 松下誠, 井上克郎, “関数の変更履歴と呼び出し関係に基づいた開発履歴理解支援システム”, 電子情報通信学会技術研究報告 pp.7-12,2004