Vol.2013-SE-179 No.14
2013/3/11

ソースコード更新情報を用いたテスト着手時期の予測

田島 峻[1]　森崎 修司[1]　小池 利和[2]

概要：ソースコードリポジトリから自動収集できる更新情報からシステムテスト着手時期の予測手法を 2 つ提案する．1 つは更新間隔が狭い時期から広い時期への移行によってシステムテスト着手と判断する．もう 1 つはソースコードの変更・削除行数に対する追加行数の比率が小さくなる時期への以降によってシステムテスト着手可能と判断する．8 プロジェクトのソースコードリポジトリを用いて提案手法を評価したところ，6 プロジェクトにおいて更新間隔が移行する結果が得られた．追加行数の比率を用いた手法では，比率の時系列遷移を 3 パターンに分類でき，うち 1 パターンでシステム開始時期の予測ができることが期待できる結果を得た．

1. はじめに

ソフトウェア開発の大規模化，短納期化に伴い，開発完了時期を予測することが難しくなってきている．開発完了時期を予測するためには，ソフトウェア開発の最終段階であるシステムテストの開始時期の予測が重要となる．プロジェクトマネージャは，同時進行して進むコーディング，単体テスト，統合テストの状況や情報を収集し，それらを統括して予測する．

プロジェクトマネージャがシステムテストの着手時期を予測するための有力な情報の一つは，開発担当者やサブシステム／コンポーネントのリーダの進捗報告である．多くのソフトウェア開発プロジェクトにおいて，進捗情報をもとに，プロジェクト進行の妥当性や健全性が判断されており，進捗報告をスムーズに支援するためのツールやシステムも提供されている．

システムテストの着手時期を決定する場合に，進捗報告以外の情報があれば，妥当性の確認や多面的な判断が可能になることが期待されるが，開発担当者やリーダにとって，進捗報告以外の情報提供や進捗報告により詳細な情報を要求することは難しい．進捗報告は主に人手によるものであり，さらなる情報の要求は開発作業を圧迫することにつながるからである．

オープンソースプロジェクトを対象として，ソースコードリポジトリの更新履歴を用いて，不具合予測をする研究が盛んに行われている [1]．これらの研究では，オープンソースの開発者から人手によって情報を集めることなく，ソフトウェアの品質や不具合が含まれている可能性の高いモジュールを予測している．

しかしながら，不具合モジュールの予測，更新の予測をはじめとして，モジュールの局所的な予測が中心であり，システムテストの着手時期を予測する研究は少ない．本稿では，プロジェクトマネージャがシステムテストの開着手時期を予測，決定するための補助情報の一つとして，ソースコードリポジトリから自動収集可能な更新情報を使った手法を提案する．具体的には，コミット間の日付間隔を用いて開発フェーズを三段階に推定する方法，及び，累積追加行数と累積削除・変更行数の増減の比率により，コーディングの完了を予測する方法である．

本稿では，以降，2 章で予測を行う上での前提，2 手法の詳細を述べ，3 章でソースコードリポジトリに適用した結果を述べる．4 章で考察し 5 章でまとめる．

2. 前提

2.1 バージョン管理システム

ソフトウェア開発におけるバージョン管理システムでは，開発における様々な情報が記録される．そうした情報の中にはコードの遷移とともにコミットを行った日付も含まれている．バージョン管理システムでは，チェックアウトとコミット（チェックイン）という操作を行うことができる．チェックアウトはバージョン管理システムのデータベースであるリポジトリから以前コミットしたデータを取り出すことを示し，コミットはリポジトリ内のデータの更新を表している．コミットはソースコードを更新し，機能の追加や修正がある際におこなわれるため，進行している

[1] 静岡大学
Shizuoka University
[2] ヤマハ株式会社
Yamaha Corporation

表 1 コミット間の日付間隔による予測値と実績値の比較

プロジェクト	要求・設計終了日の予測	要求・設計終了日の実績	コーディング終了日の予測	コーディング終了日の実績
A-1	73	77	305	231
A-2	75	108	該当なし	290
B-1	69	74	372	284
B-2	42	該当なし	276	154
C-1	66	49	234	189
C-2	53	該当なし	205	154
C-3	60	該当なし	203	161
D-1	44	13	該当なし	167

工程によっては更新の間隔が表れる．そして 1 回のコミットにはコミット日付，コミット者の情報，コミットに対するコメント等の自由記述の情報，追加行数，削除行数，変更行数，総行数が記録される．

バージョン管理システムのコミット記録から得られるこれらの情報を用いて今回の研究を行った．

2.2 コミット間の日付間隔からの予測

バージョン管理システムからはコミットの際にコミットをした日時の情報がソースコードの情報とともに記録される．ソフトウェア開発におけるコーディングのおおよその終了時期を予測することを目的とし，ここではバージョン管理システムから得られるコミット日付から求められるコミット間の日付間隔の情報を元に分析を加えた．

今回は，コミット間の日付間隔にの動向により，コーディング終了時期を予測するために，バージョン管理システムから得られるコミット日付の情報を主に用いて分析を行った．コーディング終了時期を予測することにより，テスト着手時期を客観的に予測することがで可能になる．機能の追加や更新が複数名でおこなわれるプロジェクトにおいては 1 日に複数回コミットが行われることがある．ここでは，その 1 日に複数回ある更新を 1 日に 1 回とまとめて考えることでコミット間の日付間隔を考える．つまり，コミットの記録が 1 回以上存在する日は，その日にコミットされているとし，コミットの記録が存在しない日はコミットなしとする．

コミット間の日付間隔が開発工程ごとに差があると仮定すると，それはいくつかの段階で分けられると考えられる．この方法を以下のようにして考えた．

バージョン管理システムにおける 1 コミットを c とおいたとき，n を定数として，1 つのソフトウェア開発におけるリポジトリの情報は c_1，c_2，c_3，...，c_{n-1}，c_n のコミットの集合で表すことができる．これにおいて，最初のコミット日付を 1 とした時からの経過日数 x，x の直後のコミット日付の経過日数を y とした時，コミット間の日付間隔 I を

$$f(c_x, c_y) = I_y \tag{1}$$

により表す．

この I_y について次のように段階分けをする．

(1) コミット間の日付間隔が I_d の期間
(2) コミット間の日付間隔が I_c の期間
(3) コミット間の日付間隔が I_t の期間

ここで，I_d は要求・設計の期間であり，I_c はコーディングの期間，I_t はテストの期間である．このうち，コーディングの期間の終了時である I_c の終了日を予測する．

また，検証の際，外れ値について考慮した．プロジェクトの運営の方法によって外れ値の考え方は様々であると思うが，今回のようなプロジェクト運営においては，コミット間の日付間隔を考えるにあたって，プロジェクトにおける開発ではない期間や，ソフトウェア開発における例外的なコミットとして今回は以下の二つを外れ値として挙げた．

- ソフトウェアにおけるソースコードにおいて行数遷移の大きすぎるコミット
- 休暇によるコミット間の日付間隔

上記の二項目のうち一つ目は，開発中ソフトウェアのソースコードの行数に対して，コミットした際の行数の遷移が明らかに大きいものを外れ値として除外することである．これは，そのプロジェクト内におけるソフトウェア開発のコード遷移に主に焦点を当てるために除外している．以前のプロジェクトからの引き継ぎのベースデータであったり，他のソフトウェアから機能を移植された際のことを考えるよりも，その開発プロジェクト内におけるコミットの動向を考慮するためである．

外れ値として除外した二つ目の項目は，長期休暇などの社内での休暇によると考えられるコミット間の日付間隔を考慮することである．コミット間の日付間隔を考慮するにあたって，休日の間隔を含めて考慮してしまうことで結果に影響が出てしまうことを防ぐためである．

2.3 行数遷移による予測

開発の進行に伴い，コーディング段階では機能追加や既存機能の変更が実施される．そのため，コーディング序盤ではソースコードの追加の比率が大きくなることが予想さ

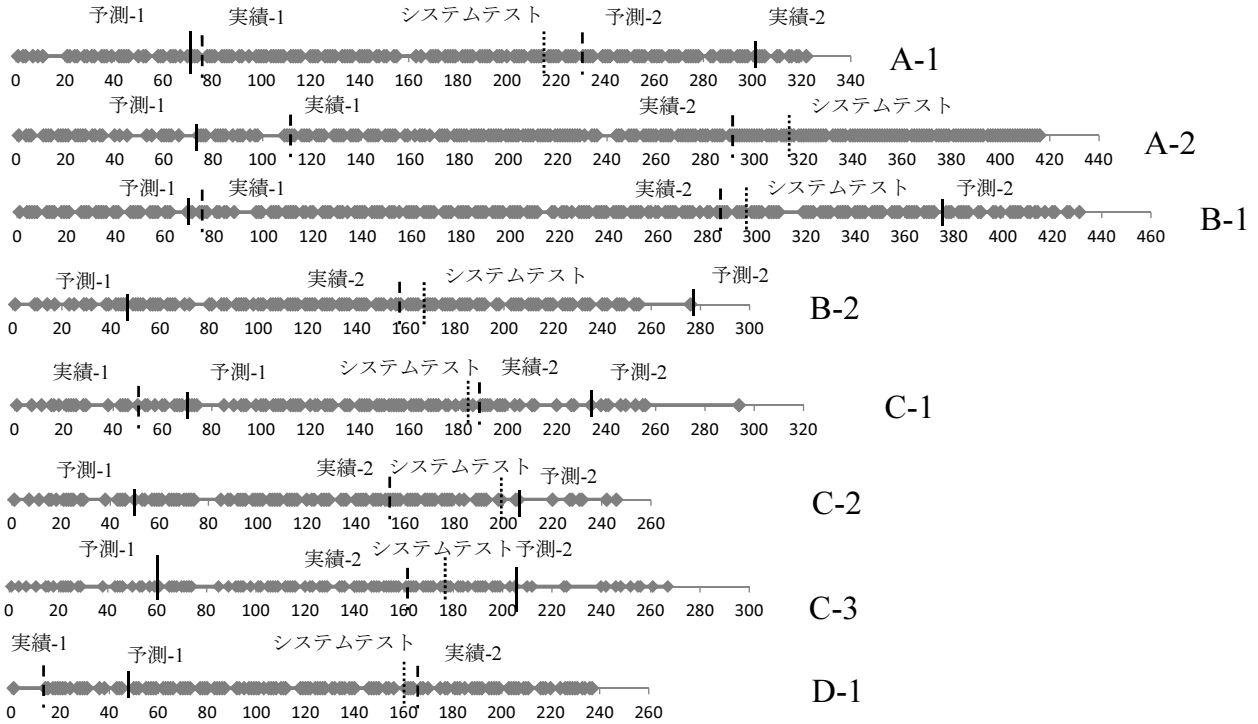

図 1 コミット間の日付間隔

れる．コーディング終盤に近づくにつれ，開発者がプログラムが期待通り動作するかを確認する単体テストが実施されはじめると，追加行数の比率は小さくなり，削除・変更行数の比率が大きくなると予想される．

そこで，ある時点までの累積追加行数と累積削除・変更行数の比率を求めることにより，

テスト着手時期を予測するために下の式を用いる．追加行数は，あるコミット k とその次のコミット $k+1$ の差分から追加となったソースコード行数を計測して求める．削除・変更行数は，あるコミット k とその次のコミット $k+1$ の差分から，削除されたソースコード行数，変更されたソースコード行数を計測して求める．最初のコミット 1 からコミット k までの追加行数の和が累積追加行数，最初のコミット 1 からコミット k までの削除行数の和に変更行数の和を加算したものが，累積削除・変更行数である．

最初のコミットからあるコミット n までの累積追加行数 $T_A(n)$ と累積削除行数 $T_M(n)$ とすると，

$$T_A(n) = \sum_{i=1}^{n} A(i) \tag{2}$$

$$T_D(n) = \sum_{i=1}^{n} D(i) \tag{3}$$

ここで得られた累計追加行数 $T_A(n)$ と，累計削除・変更行数 $T_D(n)$ から累計追加行数と累計削除・変更行数の比率 P_n を求める．

$$P_n = \frac{T_D(n)}{T_A(n)} \tag{4}$$

この P_n の値が一定の値 S よりも小さくなったとき，テスト着手が可能な時期とする．

$$P_n > S \tag{5}$$

となる S の値を設定することにより，テスト着手時期を予測する．

3. 対象データと試行結果

3.1 対象データとなるプロジェクト概要

今回はある企業において実際に実施されたプロジェクトにおけるソースコード更新情報と実績値をもとに分析している．合計 8 つのプロジェクトについて分析を行ったが，その際，プロジェクトの規模や概要によってグループ分けを行った．

グループは複雑度の高いプロジェクトの分類から順に A，B，C とし，プロジェクト A の分類に 2 つのプロジェクトがある場合はプロジェクト A-1，A-2 としている．これら 3 つの分類については全てウォーターフォール型の開発が実施された．

また，プロジェクト D はイテレーション開発が実施された分類である．

3.2 コミット間の日付間隔による予測の評価

表 1 に具体的なコミット間の日付間隔による第 1 段階終了日と考えている要求・設計終了日の予測・実績値と，第 2 段階終了日として考えているコーディング終了日の予測・実績値を一覧にしたものを挙げている．表においては，1

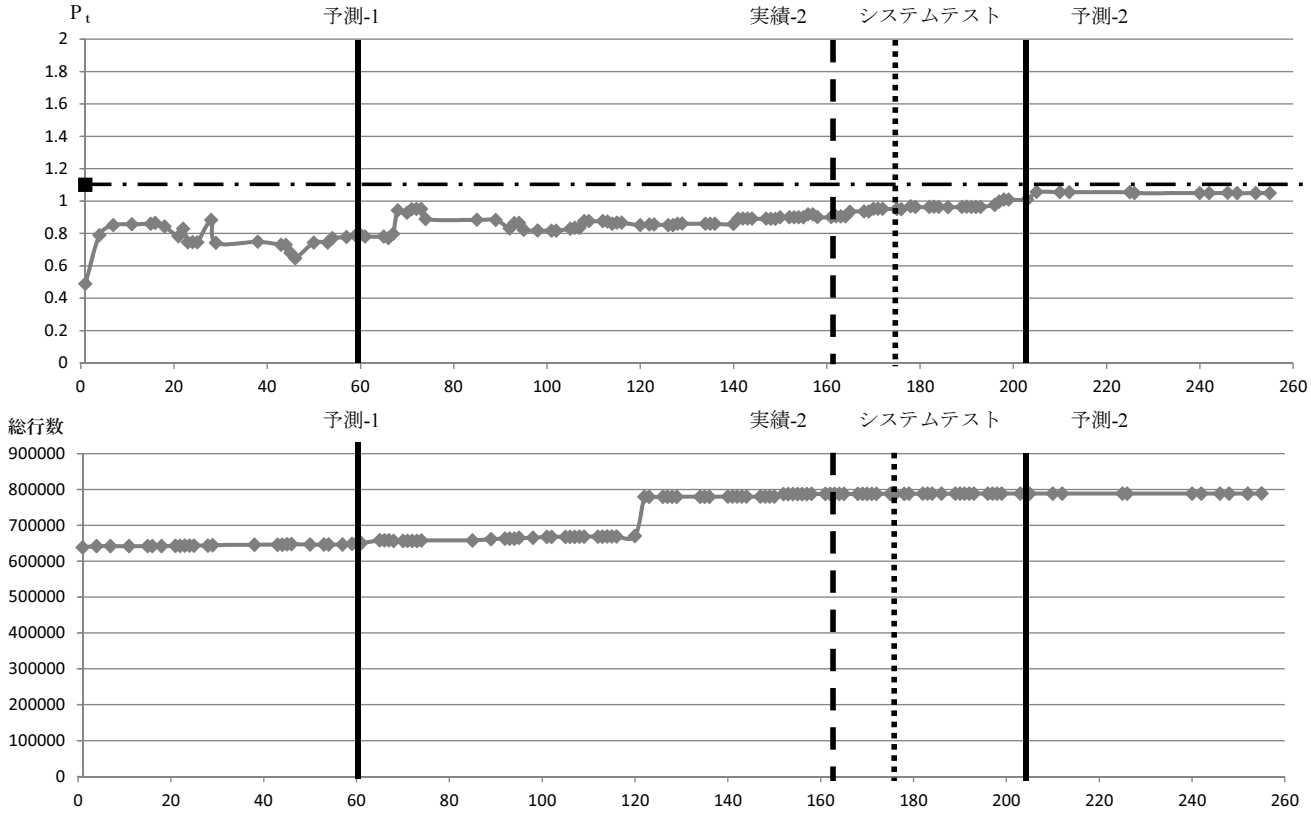

図 2 プロジェクト C-3

行が 1 プロジェクトに対応しており，各数値は順に，そのプロジェクトにおける要求・設計終了日の予測値を経過日数で求めたもの，要求・設計終了日の実績値，コーディング終了日の予測値を経過日数で求めたもの，コーディング終了日の実績値になっている．

表中の表示が該当なし，となっている部分は，第 1 段階に該当する期間が存在しなかったものであり，検出できずとなっているところは，今回の手法からは段階の移行が求められなかったものである．コミット間の日付間隔による予測では，複数のプロジェクトについて，3 段階に分かれる傾向が見られた．

図 1 は各プロジェクトのコミットの日付を，数直線上に並べたものである．この内のひとつの数直線が実際に行われたあるひとつのプロジェクトにおけるコミット間の日付間隔を表している．数直線の横軸は最初のコミット日を基準にそれからの経過日数を示している．各数直線の右側に記載されているのはプロジェクト名である．

図 1 の数直線には垂直に 3 種類の線が引いてあり，このうち実線はコミット間の日付間隔から予測した前述した 3 つの段階のそれぞれの境界線を示している．すなわち，2 本垂直な線がある場合は，1 つ目の実線が 1 段階目と 2 段階目の境界線，2 つ目の実線が 2 段階目と 3 段階目の予測による境界線となっている．図 1 中の間隔の長い点線は実績値のデータから導いたそれぞれの段階の境界線を表している．最後に，間隔の短い点線が数直線に対し垂直方向にそれぞれ入っているが，これはプロジェクトにおける実際のシステムテストの開始日である．

図 1 に示されるように，プロジェクト B-2，C-1，C-2，C-3 では，第 1 段階はコミット間の日付間隔が 2 日以上である期間がある程度続く状態が見られた．第 2 段階では，コミット間の日付間隔が 1 日である状態が続いている．図 1 に示されるように，第 1 段階でのコミット頻度よりも第 2 段階でのコミット頻度の方が多くなっていることがわかる．第 3 段階では，第 1，第 2 段階よりもコミット間隔が広くなり，およそ 5 日以上のコミット間隔がつづく結果となった．図 1 のプロジェクト C-3 において，0〜60 の間では，日付間隔が 2 日以上空いている傾向があり，61〜203 の範囲では日付間隔が 1 日である期間が続き，204〜は日付の間隔が約 5 日以上である傾向が見られた．

プロジェクト A-1，A-2，B-1 では，前述の B-2，C-1，C-2，C-3 のプロジェクトほどではないが，第 1 段階においてコミット間隔が空く傾向が見られた．また，第 2 段階においてもプロジェクト A と同様にコミット間隔がほぼ空いていない状態が続いている．第 3 段階においては，それを判定して適応できなかったプロジェクトもあり，プロジェクト B の分類のものは全体的にコミット間隔が狭い傾向が見られた．図 1 のプロジェクト B-1 において，0〜70 のあいだでは日付間隔が 2 日程度空いている傾向があり，71〜377 ではコミット間の日付間隔が 1 日である期間が続き，378 以降は 5 日程度の空きが出る傾向があった．

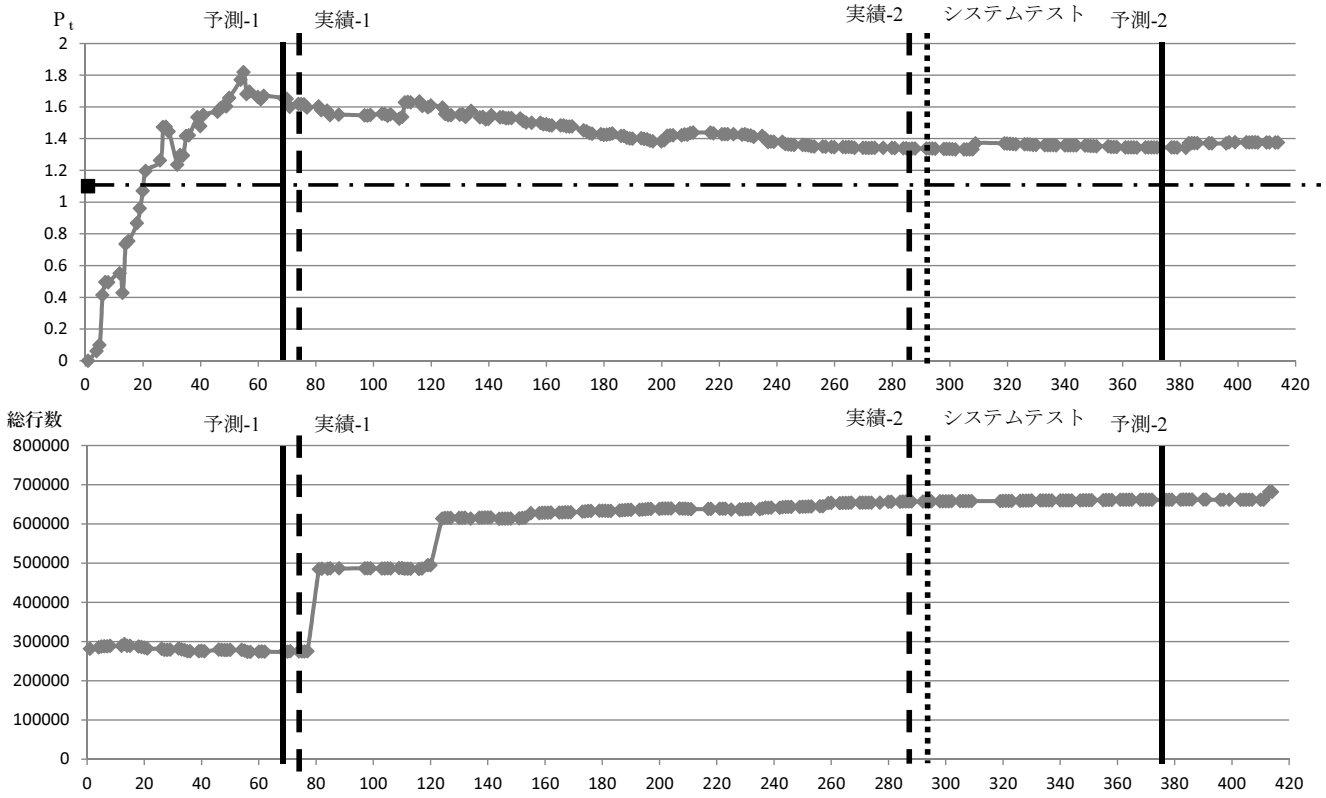

図 3 プロジェクト B-1

プロジェクト D-1 では，全体的にコミット頻度の変化が見られにくい状況であった．プロジェクト D-1 では 0〜44 では 2 日程度の空きがコミット間の日付間隔に見られたが，それ以降の期間ではコミット間の日付間隔は 1 日である状態がプロジェクト終了まで続いた．

3.3 行数遷移による予測の評価

8 件のプロジェクトの，P_n を求めた．結果は，次のタイプ A, B, C のいずれかに分類することができた．

- タイプ A
  P_n が単調増加し，S に漸次的に近づく．(プロジェクト B-2, C-1, C-3)
- タイプ B
  P_n が初期段階で大きな値をとり，S に漸次的に近づく．(プロジェクト A-1, A-2, B-1, C-2)
- タイプ C
  P_n に変化がまありない．(プロジェクト D-1)

図 2，図 3，図 4 は，それぞれ，タイプ A, B, C のプロジェクトでの P_n の値の遷移 (上半分) とソースコード総行数の遷移 (下半分) を示している．両グラフともに横軸は経過日数 t を示している．下側の総行数の遷移のグラフの縦軸は総行数である．開発終了段階である横軸の終点はリリース日となっている．垂直に引いてある実線，間隔の長い点線，間隔の短い点線の関係は前章の関係と対応しており，それぞれ実線はコミット間の日付間隔による段階の境界線，間隔の長い点線は実績のデータによる段階の境界線，間隔の長い点線は実際のシステムテスト開始時点を表している．

タイプ A, B, C の全てにおいてテスト着手可能時期を予測するための共通した値 S は得られなかった．タイプ A においては，S の値を 1.1 とすることで，テスト着手可能時期を予測できる可能性がある．

4. 考察

4.1 コミット間の日付間隔による予測

コミット間の日付間隔については，第 1 段階においてコミット間の間隔が空くのは，2 章でも述べたが，今回のようなプロジェクト運営の中において，ベースプログラムのコミットやデータの引継ぎによるコミットを外れ値として設定していることが 1 つである．また，コミットの初期段階では，プロジェクト全体を見渡した時にまだ本格的にコーディングに入っていない期間だと考えられるので，コミット間に日が空く傾向がある．このことから考えると，プロジェクト B-2，C-2，C-3 については第 1 段階において，どのプロジェクトもある程度のコミット間隔が空いているため，後半の工程に適切に作用しているのではないかと予測される．また，プロジェクト A-1，A-2，B-1，C-1 については，第 1 段階で大きくコミット間の間隔が空いているわけではなく，ソフトウェアの設計などとコーディングを並行して行っていることが推察される．

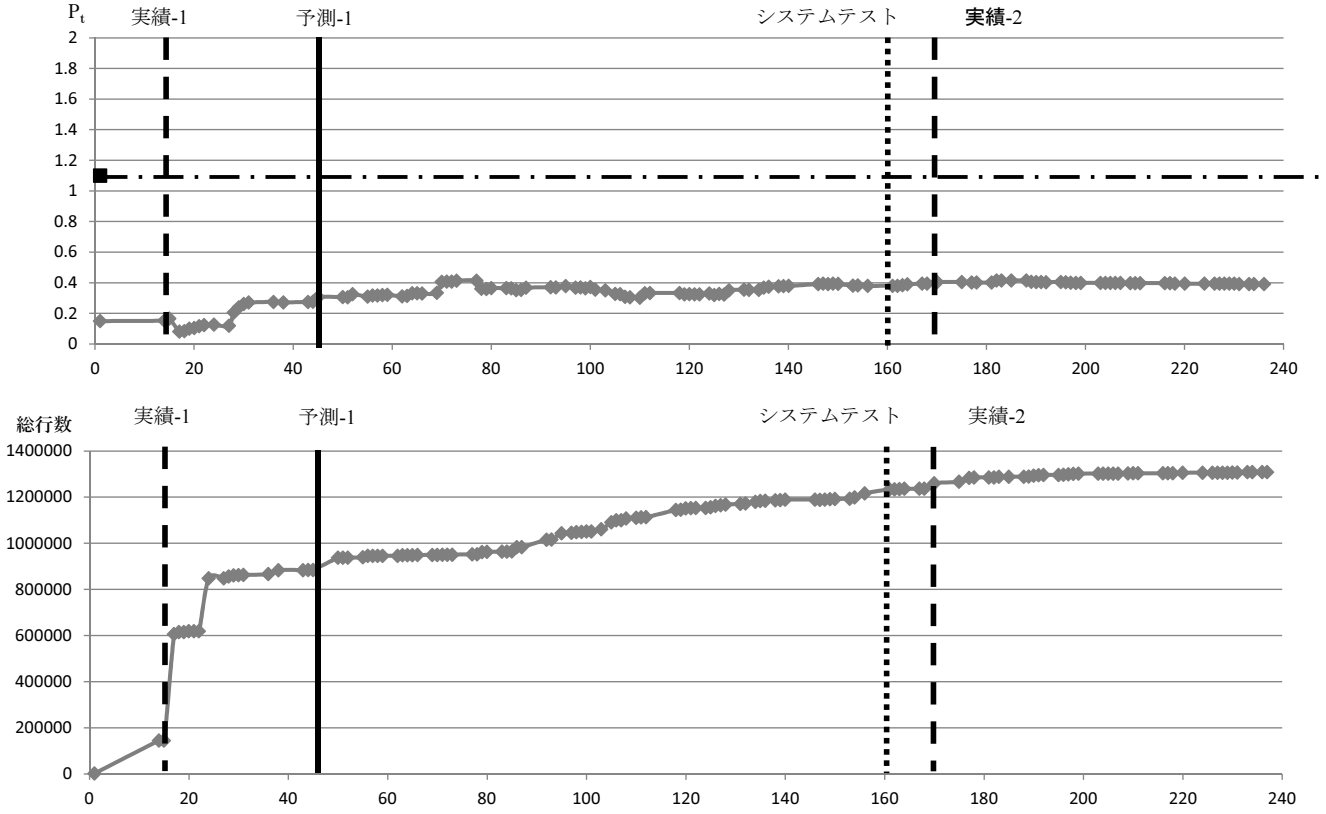

図 4 プロジェクト D-1

第 2 段階では，どのプロジェクトにおいてもコミット間隔が小さくなっている．これは開発が進行するにつれて，コーディングが本格化してきたからであると推測される．バージョン管理システムは，今回のようなプロジェクト運用をしていれば，コーディングの時期に頻繁に用いられるため，コーディングが本格化している時期は毎日のようにコミットが行われ，なおかつそれが外れ値として除外されないコード遷移であることが多いため，第 2 段階においてはコミット間隔が狭くなり，その要因は開発工程がコーディングが主となっていることが考えられる．

次に第 3 段階では，プロジェクト D-1 以外のものは再びコミット間隔が広がり始める．特に，プロジェクト B-2，C-2，C-3 の分類のものについてはその傾向が顕著に見られ，プロジェクト A-1，A-2，B-1，C-1 のものについても第 3 段階が存在し，コミット間隔が空いている傾向が見られる．コミット間の日付間隔が大きくなってくるということは，コーディングが終盤に移り，次の段階であるテスト段階へ移行したものだと考えられる．テストを主に行う段階になると，順調に進んでいるプロジェクトであれば，コミット回数は減っていき，機能開発や機能追加を目的としたコミットではなく，修正のためのコミットが主になってくる．このため，コミット回数が減少し，コミット間隔が広くなると考えられる．

また，プロジェクト D-1 のものについては，イテレーション開発であるということから，ほかのプロジェクト分類のようなウォーターフォール型開発モデルとは違う傾向が現れたと考えられる．

表 1 はコミット間の日付間隔による予測値と，プロジェクトを終えてからの実績値とを比較したものである．各項目に挙げられている数値は 1 回目のコミットからの経過日数を示している．要求・設計終了日の予測の実績値の差は 4〜33 日間，コーディング終了日の予測と実績値の差は 42〜122 日間と，予測の結果に大きく差が出た．このことから，第 1 段階である要求・設計終了日は，コミット間の日付間隔から予測できうるが，コーディング終了時期を的確に予測できておらず，結果として大きな予実差が生じている．

プロジェクトの実際の様子を知る開発者から次のような意見を得た．

- 第 2 期間への移行時期の予測はコーディング本格開始時点の予測に活用できそうである．
- 第 3 期間への移行時期の予測時期は実際にシステムテスト開始時点と近く予測に使えそうである．たとえば，システムテスト移行の妥当性を判断する材料として使える可能性がある．
- 第 3 期間への移行時期の予測は品質安定時期 (リリース可能時期) の予測に活用できそうである．たとえば，リリース時期の予測や判定の妥当性を判断する材料として使える可能性がある．ただし，プロジェクトによっては，グラフからは品質が安定しつつあると推測

されるが実際には異なるものもあるため，さらなる議論が必要である．

4.2 行数遷移による予測

タイプ A のプロジェクトでは，一定の値 S を与えることによって，テスト着手可能時期を予測することができる可能性を示す結果が得られた．提案手法で仮定している，コーディング初期段階ではソースコードが追加され，中盤から終盤にかけて追加の割合が減り，削除・変更の行数の割合が増える傾向を示すプロジェクトが存在することが確認できた．タイプ A に分類されるプロジェクトの条件を検討することは今後の重要な課題の 1 つである．

タイプ B, C のプロジェクトでは，提案手法で想定する P_n の遷移とは異なるものがあった．タイプ C はイテレーション型プロセスによる開発であった．タイプ B に分類されるプロジェクトはタイプ A と同一の開発プロセスでありながら，P_n の遷移が異なっていた．

プロジェクトの実際の様子を知る開発者から，次のような意見を得た．

- P_n のような累積で算出する指標では，変化を捉えづらく，その推移からの予測は難しいと考える．
- プロジェクト完了後にプロジェクト評価の目的で使う指標として使えると考える．
- テスト着手時期の予測としては，累積ではなく，移動平均等で遷移を観察するほうがよいのではないか．

5. おわりに

ソースコードリポジトリから自動収集できる更新情報からテスト着手時期の予測手法を提案した．1 つはコミット日付を用いた手法であり，コミット間の日付間隔を用いること間隔の狭い時期と広い時期を区別することによって，システムテストの着手可能時期を予測する．もう 1 つは，コミットごとのソースコードの追加行数と削除・変更行数の比率を用いた手法であり，追加行数の比率が小さくなる時期を区別することによって，システムテスト着手時期を予測する．

8 プロジェクト分の商用ソフトウェアのソースコードリポジトリの履歴とシステムテスト着手時期の実績を用いて，提案手法を評価したところ，コミット間の日付間隔による予測手法において 8 プロジェクト中 6 プロジェクトでシステムテスト着手日を予測できた．ソースコードの累積追加行数，削除・変更行数の比率の時系列の遷移を調べたところ，遷移を 3 パターンに分類することができ，そのうち 1 パターンでは，システムテスト着手の実績値周辺で一定の値に近づく結果が得られた．

評価に用いたプロジェクトをよく知る実務者との議論により，既存のシステムテスト着手時期の判断材料と提案手法を併用すれば，多面的な判断が期待されることを確認した．コミットの日付間隔による手法では，予測精度の向上，累積追加，削除・変更行数による手法では，3 パターンの条件や閾値の精査が今後の課題である．

謝辞

データ収集に携わった方々に御礼申し上げる．本研究は文部科学省科学研究補助費（基盤研究 B:課題番号 23300009）による助成を受けた．

参考文献

[1] B. Kenmei, G. Antoniol and M. Penta, “Trend Analysis and Issue Prediction in Large-Scale Open Source Systems, ” In Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering, pp. 73-82(2008)

[2] T. M. Khoshgoftaar, E. B. Allen, “Ordering Fault-Prone Software Modules, ” Software Quality Control, vol.11 no.1, pp.19-37(2003)

[3] S. Kim , T. Zimmermann , E. J. Whitehead Jr. , A. Zeller, “Predicting Faults from Cached History,” Proceedings of the 29th international conference on Software Engineering, p.489-498(2007)

[4] 山田悠太，藤原賢二，吉田則裕，飯田元，トピック抽出に基づく開発者の活動に着目したリポジトリ可視化手法，情報処理学会研究報告. ソフトウェア工学研究会報告 2012-SE-178(17), 1-7, 2012-10-25