B-01

ソフトウェア開発のグループワーク演習における各個人のタスクの自動計測

A tool for recording the group development activities in a programming class

五田 篤志†　井上知奈津†　大堂 哲也†　福本大†　玉田春昭‡
Atsushi Itsuda　Chinatsu Inoue　Tetsuya Ohodo　Dai Hukumoto　Haruaki Tamada

1. 研究の背景

ソフトウェア開発の現場において，開発作業はグループワークであり，多人数で協調して作業をする。それにより納期までにソフトウェアを完成・出荷できるようになる。このような多人数での開発は，一人での開発と比べると，グループワーク特有の作業が生じる。例えば，作業の分担や，作業者間のコミュニケーション，マネージャによる進捗管理などが挙げられる。そのため，学生は実社会に出る前に，グループワークのプロセスやその重要性を学ぶ必要がある。

多くの大学の情報系学部でソフトウェア工学教育の一環として，大学の学生を対象にグループ開発の体験学習が行われている。学生は実際の開発経験を通じて，その重要性について学習する。ただし，学生が必ずしもその重要性を習得できるとは限らない。また教員にとってもグループワークの評価を行うことは容易ではなく，成果物であるソフトウェアの出来栄えが良いからといって，作業者間の協調がうまく行われていたことの証にはならない。学生の理解を促すには，グループ開発のプロセスについて何らかの定量的なデータを収集・分析し，グループワークの重要性を示す客観的な分析結果を学生に示さなければならない。データ収集として，学生に演習の実績を記録してもらう取り組みが行われているが，学生による手入力では，記録漏れなどがありデータの値として信頼性に欠けている。

そこで，学生の手入力に頼らない実績を自動的に記録する方法を提案する。実績として収集するデータは，作業時間，打鍵数，クリック数である。これらのデータは実績に直接結びつき，また分析も容易であると考えられる。これにより，通常手入力では記録することのできない実績に直接結びつくデータを収集することができ，客観的分析結果を学生に示すことができる。

2. 提案手法

2.1. 計測対象とする講義科目

前提として，本稿ではソフトウェア開発のグループワーク演習として，京都産業大学のプロジェクト演習という講義科目とする。このプロジェクト演習は学部３年生を対象としたグループワーク演習で，４カ月の開発期間（２時限／週×７週）で行われる。１グループあたり，4～5 名で，担当教員は 1 名,TA1 名で行われる。開発環境は学生個人が所有する MacBook (Mac OS X)である。

この演習では１グループ内の学生それぞれに役割を持たせ，１つのプロダクトを作成する。作成するプロダクトの要求仕様が与えられ，設計，実装，テストのプロセスを経て，最後は納品という形で完成したプロダクトを提出させる。作成するプロダクトは年度ごとに決まっており，各グループで同じプロダクトを作成する。もちろん，設計の違いなどにより，実現方法はグループごとに異なる。なお，本年度(2012)年度はフーリエ変換器，作年度は SpiroDesign であった。

また，この演習では，各グループのメンバの作業履歴を GanttProject[1]を利用して，ガントチャート上に予定と実績を記録することを課している。まず各グループで，サンプルを参考にし，作業内容を WBS に分解する。このサンプルは TA がこの演習のサンプルのために作成したものである。次に分解された作業ごとに予定をガントチャート上に入力する。その後，それぞれの学生の作業が進むごとに実際に費やした時間を実績としてガントチャートに入力する。演習で用いたガントチャートのサンプルを図 1 に示す。しかし，実績の入力は納品の直前に思い出しながら一括で入力する学生が多く，信頼性の低い値であることが多い。ガントチャートに実績を入力する部分を支援できれば，それぞれの学生の振り返りにもつながり，ソフトウェア開発へのより深い理解を促すことができると期待できる。

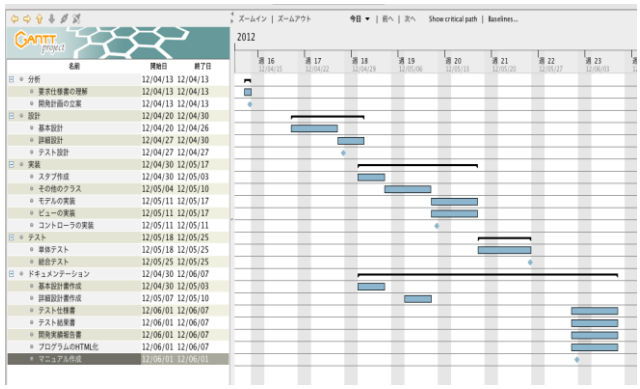

図 1.演習で用いたガントチャートのサンプル

2.2 計測対象

プロジェクトの評価には何らかの定量的なデータを収集し，分析をする必要がある。そこで，プロジェクトの作業履歴として，アプリケーションごとの起動時間を記録するものとする。多くの概念は門田らが提案する計測システム[2]を参考にしている。しかし，門田らの作成した具体的なツール Taskpit[3]は Windows 上で動作するため，本稿で対象とするプラットフォームでは動作しない。そのため，ここで Mac OS X 上で動作する計測システムを改めて述べる。

†京都産業大学，Kyoto Sangyo University of Information Technology
‡京都産業大学，Kyoto Sangyo University

プロジェクトでの作業内容を記録するとき，手作業による記録では計測データの記録漏れや，計測が正確でないなど，客観的なデータとして信用性に欠ける。そこで，アプリケーションがアクティブになった時間を自動的に計測することにより，定量的なデータを収集することにした。アプリケーションがアクティブになるということはそのアプリケーションで作業をしていることを表している。アプリケーションを切り替えた場合，今まで作業していたアプリケーションがアクティブでなくなり，切り替えたアプリケーションがアクティブになる。このように，アクティブになったアプリケーションを検知することで，作業履歴が記録できるようになる。

また，具体的な作業を行っていることを確認する指標として，マウスのクリック数，キーボードの打鍵数も記録する。ただし，このツールをインストールするのは学生個人が所有する MacBook である。そのため，プライバシにも配慮しなければならない。そのため，入力したキーはパスワードなどの秘密情報にもつながり，また，私的利用時のプライバシの侵害にもつながるため，記録しない。

コンピュータ上の作業履歴を自動計測する既存のツールとして ManicTime[4]，TaskPit[3]がリリースされている。しかし,これらはいずれも Windows 用のツールである。そのため，Mac を用いている本学の演習では利用できない。そこで，作業履歴を自動計測し，なおかつ Mac OS X 上で動作する MALIMO を開発した。

3. 作業履歴の計測

3.1 タスクへの分解

対象とする演習の要件には，各学生が用いるアプリケーションの指定は無い。例えば，コーディング作業では emacs，Xcode，Eclipse といったアプリケーションの候補があげられ，それぞれの学生は好みのアプリケーションを使用する。そのため，アプリケーション名からは，作業内容を特定できない。そこで作業者の作業内容を特定するためタスクという概念を導入する。つまり，コーディングという作業を行うために利用するアプリケーションを列挙しておく。そして，それらのアプリケーションがアクティブになったとき，コーディング作業が行われているものとする。このように，複数のアプリケーションを１つのタスクに割り当てて，タスク単位でデータを収集する。これにより，各作業者に異なる開発環境でも同じような計測ができる。

一方，同じアプリケーションであっても，異なるタスクが実行される場合も考えられる。例えば，Eclipse ではコーディングの他に，UML 設計が行える。そこで，アクティブになったウィンドウで開いているファイル名に着目する。多くのアプリケーションでは，ウィンドウタイトルにファイル名が表示される。そのファイル名を取得し，タスク割り当ての情報として用いる。

3.2 作業内容の記録

3.1 で述べた情報を記録するため，MALIMO はアプリケーションがアクティブになったときに，以下の５種類の情報を CSV 形式でファイルに記録する。

- アクティブになったアプリケーション名
- アクティブになったアプリケーションのウィンドウタイトル
- アクティブになった時間
- 前回アクティブになってからのキーストローク数
- 前回アクティブになってからのクリック数

記録のときは，本来行うべき作業に影響がでないよう，最小限の処理で済むようにする。タスクへの分解や作業内容の分析，また，ガントチャートへの入力は後に別途行うものとする。

記録したデータを分析するために，２つのグラフを出力する。１つはタスクごとの作業時間を表すグラフであり，もう一方のグラフは日ごとの作業時間を表すグラフである。

3.3 作業履歴の分析・閲覧

作業履歴を分析・閲覧するため，記録した内容をタスクに分解する。タスクの設定は WBS に依存するものであり，グループごとに異なる。そのため，設定ファイルを利用してタスクを設定する。設定ファイルは INI 形式を採用しており，キーと値のペアが一行に記される。キーにはタスク名が記され，値には，そのタスクに対応するアプリケーション名，ファイル名が記される。キーと値は=で区切られ，アプリケーション名が複数続く場合にはコンマで区切られる。また，タスク割り当てにファイル名を用いる場合は，アプリケーション名にコンマを続けて，ファイル名を記す。ワイルドカードとしてアスタリスク(*)をアプリケーション名，ファイル名で受け入れる。アスタリスクは任意の文字列にマッチする。図 2 に設定ファイルの例を示す。

このサンプルではアプリケーションの Eclipse と Xcode をコーディングタスクにしている。またその他のコーディングに使うアプリケーションにも対応できるように、拡張子が c と java のファイルもコーディングタスクにしている。コーディングと UML 設計で同じアプリケーションの Eclise を指定しているが、ファイルの拡張子で異なるタスクに絞り込むことが可能である。

```
文書作成=Word,TextEdit,Pages
コーディング=Eclipse,Xcode,*:*.c,*:*.java
UML 設計=Eclipse:*.uxf
ブラウザ=Firefox,Sfari,Chrome
その他=*:*
```

図 2. タスク設定ファイルのサンプル

4. ケーススタディ

4.1 作業履歴計測システム

第３章で述べた内容を確認するため，作業履歴計測システム MALIMO を作成した。MALIMO は Mac OS X Snow Leopard 以降，Java SE 6 以降の環境で動作する。作業履歴の記録部分は Objective-C で作成し，分析・閲覧部分は Java で作成した。

作業履歴の記録には，Mac OS X の通知機構を利用する。NSNotificationCenter クラスに通知してほしいイベントをイベントハンドラとともに登録してすると，イベント発生時にイベントハンドラが呼び出される機構である[5]。

MALIMO では，ウィンドウがアクティブになったときに通知するよう通知機構に登録を行うことで，作業履歴を記録するようにした。

また，マウスのクリック，打鍵を検知するためにイベントモニタ機構を利用する。通常，マウスクリックや打鍵を検知するにはイベントモニタ機構の LocalMonitor を利用する。しかし，ここで行いたいことは，MALIMO 起動中のすべてのマウスクリック，打鍵の検知である。すなわち，MALIMO 以外のアプリケーションに対するイベントも検知しなければならない。そのために GlobalMonitor が利用できる。ただし，GlobalMonitor を利用するとキーロガーなどの危険なアプリケーションも作成できてしまうため，Mac OS X Lion 以降では無条件に GlobalMonitor から情報を取り出せないようになっている。GlobalMonitor を利用するためには，環境設定のユニバーサルアクセスで補助装置にアクセスできるようにしなければならない。

4.2 作業履歴の計測

4.1 で作成した MALIMO を情報系学部の 4 年生に適用し，どのような作業履歴が計測できるか確かめた。また，計測した作業履歴を確認し，被験者にインタビューを行い，実際の作業がどのように進められたのかを確認した。実行環境は Mac OS X Lion 10.7.4，Java SE 6，Intel Core i5, 1.6GHz*2，2GB RAM の iMac である。計測期間は 2012 年 2 月 21 日から 2 月 27 日までの 7 日間である。今回の計測で用いた設定ファイルを図 3 に示す。

```
文書作成=Word,TextEdit,Pages
論文=*:ronbun.docx
コーディング=*:*.c,*:*.java
ブラウザ=Firefox,Sfari,Chrome
Twitter=YoruFukurou
スリープ=loginwindow,ScreenSaverEngine
その他=*:*
```

図 3.タスクの設定ファイル

計測の結果，7 日間で計 32 時間当該マシンを利用しており，1 日当たり平均して，4 時間，クリック数 1,228，打鍵数 4,528 であった。ただし，これにはスクリーンセーバーが動作している時間も含まれている。スクリーンセーバーが動作していた時間を除けば 7 日間で計 22 時間であった。

図 4 にタスクごとに集計した作業時間のグラフを示す。縦軸はタスク名，横軸は作業時間を分単位で示している。バーの色は日付ごとに色分けされている。このグラフから被験者はスリープ以外には，コーディング，ブラウジングと Twitter に多くの作業を割いていることがわかる。

インタビューにより作業内容を確認してみると，自身の研究のためのプログラミングに多くの時間を費やしていたこと，わからなかった箇所は検索エンジンなどで検索していたことがわかった。また，常日頃から頻繁に Twitter で友人との連絡をとっているため，Twitter に費やす時間が自然と多くなったことがわかった。

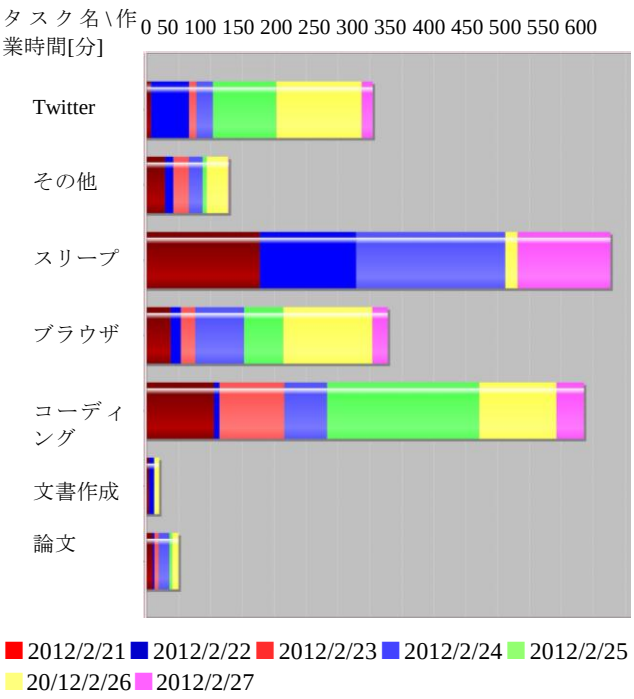

図 4.タスク毎の 1 週間の作業時間の集計

図 5 に日付ごとに各タスクの作業量を表すグラフを示す。横軸は日付，主縦軸は作業時間，副縦軸は打鍵数，クリック数を表す。図 2 からは，コーディングのタスクに費やした時間が一番多いのは 2 月 25 日であると読み取れる。しかし，図 5 の打鍵数，クリック数を見てみると，2 月 25 日より，2 月 26 日のほうがともに多くなっている。これはコーディングを行うアプリケーションの前で悩んでいる時間が 25 日のほうが 26 日より多かったことが考えられる。つまり，26 日のほうが実際に手を動かし，作業が進んでいると考えられる。インタビューにより，この仮説が正しい ことが確認できた。

これらの結果から，プログラミング演習において，学生の作業履歴を把握するには，十分であるといえる。

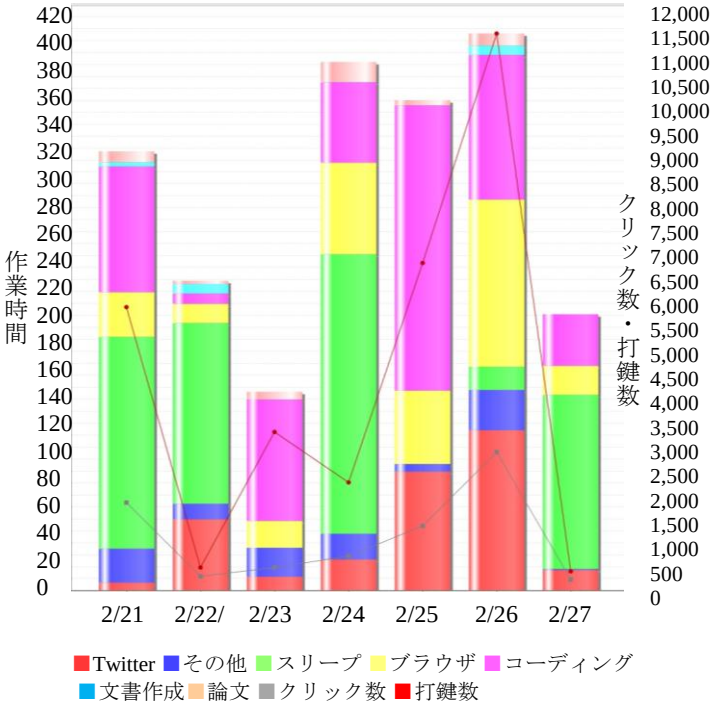

図 5.日付ごとの 1 週間の集計

5. 関連研究

玉田らはグループワーク学習を対象にメンバ間でのコミュニケーション量と回数，また，開発時間に着目し，と成果物の評価との関係を分析した[6]。分析内容は興味深いものの，分析の基礎となるデータをアンケートで収集しているため，正確さに疑問が残る。特に開発時間はグループによりばらつきが大きく，正確に答えているグループとそうでないグループの区別がつかないことが問題である。

齋藤らはプログラミング演習時のコーディング過程を可視化するためのシステムとその評価のためのメトリクスを提案している[7]。このシステムはグループワークでの開発過程を可視化するために使える可能性がある。ただし，齋藤らの手法はコンパイルエラーの修正に手間のかかる初心者を対象としているため，そのまま適用することはできない。

大平らは企業での開発現場の問題を可視化するためのツール EPM を提案している[8]。これはソースコード管理システム，バグ管理システム，そして，メーリングリストの情報をまとめ，1 つのバグに対してコミットやメンバ間でのメールのやりとりを関連付けるツールである。本ツールはソースコードに関する定量的データを収集するために有用であるが，本稿で問題としている WBS データについては収集の対象外である。また，タスクという単位でのデータ計測が難しい点が問題である。

6. まとめ

本稿では，ソフトウェア開発のグループワーク演習における作業履歴を信頼できるデータとして記録することを目的として，タスクの自動計測ツールを開発した。

ケーススタディでは，記録したデータをもとに学生がどのような作業に従事していたか，またどれだけの作業量であったかを示し，実験後のインタビューによってデータが値として信頼できるかの確認を行った。

参考文献

[1] GanntProject Team, ``Gantt Project: Free project scheduling and management,'' http://www.ganttproject.biz/, (Last Accessed: July 23, 2012).

[2] 門田暁人，亀井靖高，上野秀剛，松本健一，``プロセス改善のためのソフトウェア開発タスク計測システム''，ソフトウェア工学の基礎 XV，日本ソフトウェア科学会 FOSE2008, pp.123-128, Nov. 2008.

[3] 奈良先端科学技術大学院大学，``Taskpit'', http://taskpit.jpn.org/index.html, (Last Accessed: July 23, 2012).

[4], ManicTime, ``Time management software – ManicTime,'' ,http://www.manictime.com/ (Last Accessed: July 23, 2012)

[5]荻原剛志，``詳解 Objective-C 2.0 第 3 版''，ソフトバンククリエイティブ，Dec. 2011.

[6]玉田 春昭　井垣 宏　引地 一将　門田 暁人　松本 健一 “プログラミング実習におけるグループ開発プロセスの分析”,ソフトウェア工学の基礎XI,日本ソフトウェア科学会 FOSE2005,pp.123--128,November2005.

[7] 齋藤俊，山田誠，井垣宏，楠本真二，井上亮文，星徹，プログラミング演習における受講生支援のためのコーディング過程可視化システムの提案，信学技報 ソフトウェアサイエンス研究会, Vol.2012-03-SS, pp.61--66, March 2012.

[8]大平雅雄, 横森励士, 坂井誠, 岩村聡, 小野英治, 新海平, 横川智教, “ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム,“ 電子情報通信学会論文誌 D-Ｉ, Vol.J88-D-Ｉ, No.2, pp.228-239, Feb. 2005.