モジュール化によるホワイトカラー労働の生産性向上のための基盤ソフトウェア「MIP」の提案

金澤圭吾[†1]　佐藤哲也[†2]

キーワード：生産性，ホワイトカラー，バージョン管理，Git

The Proposal of Productivity Improvement Software “MIP” for White Collar working

KEIGO KANAZAWA[†1]　TETSUYA SATO[†2]

Keywords: Productivity, White-Collar, Version Control, Git

1. 解決する課題とその背景

ホワイトカラーの低生産性は我が国の大きな課題である．一般にホワイトカラー労働は非定型性や属人性を有するため，製造業現場で有効とされる生産管理手法が有効ではないとされる．また，労働人口の減少局面にあって，リモートワークや外部の知的生産リソースの活用が求められているが，管理の困難さや従前の勤務管理の方法とのミスマッチから存分に普及しているとはいえない．また，我が国固有の勤労文化や商習慣などが影響して生産性を押し下げているという指摘もある．

本論ではその課題解決のためのソフトウェアを設計・提案することを目的とする．

2. 提案の理論的背景

本提案の動機としては，インターネット関連産業におけるソフトウェア開発方法論で採用されている方法論を一般的なホワイトカラー労働に導入することにある．それにより，成果物の品質保持や進捗の可視化の大幅な改善が期待できる．具体的には，バージョン管理ソフトウェアの導入による生産プロセスの可視化と継続的な生産物の品質維持等の要素を主に考えている．以下に幾つかの論点について整理する．

2.1 ソフトウェア開発におけるバージョン管理の重要性

一般にソフトウェアの開発では,顧客のニーズ対応やセキュリティ対策などのためにバージョンアップが頻繁に行われる．つまり，生産物として僅かな複数の違いをもつ派生品が同時並行的に多数存在している．また，開発のプロセスにおいても過去のコードの状態に戻す必要がある．また作業者の詳細な作業履歴を記録することで，将来的な不具合の発生原因の追求を行う．これらの課題を解決するために，バージョン管理システム（VCS）と呼ばれるソフトウェアを導入することが一般的である．

様々な VCS が存在するが，近年 Git が事実上の標準になっている．Git の特徴としては分散型リポジトリを採用しているために，開発者が原本リポジトリの汚染を気にすることなく作業を記録できることがある．また，最終的に優れた改善パッチが完成した場合は，そのパッチを大元のリポジトリに反映するよう依頼することも可能である．この特徴は多くのオープンソースソフトウェアの開発において有効に活用されており，際立った効率性を実現している．しかし，VCS はホワイトカラー労働一般でみるとまだまだ普及しているとはいえない．これらの普及により大幅な生産性の向上が見込める．

2.2 修正プロセスのモジュールとしての commit

Git では修正履歴は commit と呼ばれる．システム理論的にいえば，この commit は修正プロセスのモジュラリティを高めて可搬性を実現したものである. commit 情報はその内部で修正前の状態と修正後の状態を保持している．そのため，修正前の状態が同一なファイルであるリポジトリであれば，当該 commit を異なるリポジトリに適用できることになる．また，修正によっては原理的に別ファイルの複数箇所の修正を統一的に扱う必要があるが，コミットとしてまとめることで問題を回避することができると言った特徴がある．

この修正プロセスをモジュール化した commit は情報の変更作業に可搬性をを提供し，オープンソースの高い生産性

†1 （株）デザインルール
Designrule Inc.
†2 （株）デザインルール
Designrule Inc.

を実現する鍵となる技術である．この commit を用いた情報生産プロセスのモジュール化が非ソフトウェアの生産にも適用できれば，高い生産性が実現できる可能性がある．そこで，MIP では Git の活用を考えたい．

2.3 ホワイトカラー労働の整理

MIP の検討に入る前に，本論における幾つかの前提概念を整理しよう．本論におけるホワイトカラー労働の定義として，抽象的な情報学的シンボルの取得と作成及びそのための操作をあげたい．別の言い方をすれば物理的な行為を除く，何らかのコミュニケーションやデザインなどの行為の総称がホワイトカラー労働である．この定義はやや曖昧であるが，一般的なホワイトカラー労働の持たれるイメージと大きく変わることはないと考えている．

ところで，情報をストックとフローに分けて考えることもしばしば行われる．それをホワイトカラー労働に適用すれば，労働者内部に蓄えられた知識がストックであり，目先の必要な業務処理のために流通する情報がフローである．さらにフローもインプットとアウトプットに分けることができるだろう．一般にインプットはコミュニケーションを通じて得られた情報や，知識情報を書籍や Web サイトなどから収集する行為に相当するであろう．一方，アウトプットは業務の遂行のために作成された文書類の作成が相当するといえる．

ホワイトカラー労働の生産性を検討する上では，本来これらの 3 要素であるストック，インプット・フローおよびアウトプットフローのそれぞれが適切に計量できることが望ましい．しかし，人間のもつストック情報量を適切に計測することは現実的ではない．いくつかの有効性の乏しい資格制度が代替的に存在しているに過ぎない．インプット・フローの計測はパソコンを視聴デバイスとしている活動についてはおそらく可能であろう．だが，それ以外のデバイスによる認知を計量することは現時点では難しい．

一方，比較的容易と思われるのはアウトプット・フローである．特に情報の有効寿命の長い重要な情報であればしっかりとした記録を行うことは必定である．実際にソフトウェア開発における VCS のログはソフトウェア開発における生産性の計量メトリックスとして活用されている．[1]

3. MIP の提案の概要

3.1 Git 操作の自動化

Git が優れた VCS であり，多くの開発者に支持されていることは既に述べた．しかし，一方で大きな問題としては操作が難解であることがあげられる．もとより開発者にとっても使いにくいと言われることもしばしばである．また，Git を一般的に利用しているチームでは，やってはいけない操作などが存在している．これらの操作を初心者が実施してしまうことに対する恐れも大きな問題点である．

そこで，MIP では，これらの Git の利用方法のうち標準的で安全とされる利用方法をパッケージ化して提供することをめざす．これらのローカルでの履歴保存と履歴の共有操作を自動化することにより，複数人での制作物のリアルタイム共有を実現することが可能である．

3.2 標準的な利用方法としての Github-flow

今日の開発現場では単一のソースコード群に対する複数メンバー(チーム)による同時並行的開発が中心であるが，チームで一つの制作物を制作する以上特定箇所の重複修正は避けられない．VCS ではこのような現象をコンフリクトが発生したと表現する．コンフリクトが発生すると，人手でファイルの修正を行うコストが必要になり歓迎されない．そこで，Git を基盤としてコンフリクトの発生を最低限に抑えつつ，同時並行的に複数人数で編集を進めていく方法論が開発されている．代表的な方法論として Github-flow が挙げられる[2].

- 管理者がマスタ（原本）ブランチ(※)を準備する．
- 変更する作業者はマスタのコピーを作成し，作業者独自のブランチ（名）を設定する．
- 変更は作業者独自のブランチ上で行う．
- 変更作業が終了したら，当該作業をあらわすコミット集合をマスタブランチに取り込むよう要求を行う．
- 要求がある場合，管理者ないしは確認を委託された者は当該変更作業が正当な内容であり，取り込みに問題がないことを確認する．
- 確認の上，問題がない場合はマスタ上に取り込む．

※）ブランチとは，VCS における幾つかの派生バージョンの存在を示す．

その際に，変更者はサイクル内での変更の修正サイズがなるべく小さくなるように配慮する．つまりマスタからコピーを作成して，取り込み要求を作成するまでのサイクル時間をなるべく短くする．その時間が短ければマスタ側で別の変更要求が受け入れられて重複修正の発生するリスクを減らすことができる．もちろんそのためには変更者に対する作業依頼のレベルで十分に小さいタスクとなっていることが必要である．

もっとも，修正のサイクル時間を短くしたとしても，変更作業中に変更作業を行っている箇所が他作業者によって変更されるリスクは常に存在している．その場合は作業者の固有ブランチ上でマスタに生じた変更を吸収し，最新のマスタに対する変更要求として作成する義務がある．

これらの Github-flow とよばれる方法論は筆者らの研究開発により部分的に自動化することが可能である．これらの処理により難解とされる Git の利用を自動化するソフトウェアが実現することになる．

3.3 自動化に伴う commit 粒度の問題

MIP では，commit 作業を機械的に記録することになる．実

際にはファイルの保存が行われるたびに commit を記録することになる．しかし，本来の Github-flow を始めとした Git の流儀では，commit は意味のあるまとまり毎に作成されることが望ましいとされる．それと比較すると明らかに細かい粒度で保存することになる．その点は今後の課題としてあげられる．しかし，Git における commit は複数の commits を合成して一つの commit にすることが可能である．将来的には commit の内容を知能的に判断して意味のあるまとまりを実現できることが望ましい．

3.4 継続的改善作業の適用

今日的なアジャイル開発とよばれる，ソフトウェア開発方法論の大きな特徴として，継続的なインスペクション（ソフトウェア・テスト）とビルド（デプロイ）をあげることができる[3]．これは，共通リポジトリで管理されるソースコードが変更されるたびに，当該ソースコードの品質を機械的・ソフトウェア的に確認することである．これによりバグの混入を早期発見することができ，コードの品質を担保することが可能になる．

ところで，ソースコードの品質については，実行結果が正しくまた安定して動作をするというソフトウェアに本来求められる性質に加えて，他の開発者にとっても読みやすくメンテナンスや修正が行いやすいという性質も重要である．その 2 つの性質のうち，前者の性質は一般的なホワイトカラーの成果物については定義が困難であるが，後者の性質は極めて重要である．

読みやすいコードは適切なインデントの有無や一行の長さが一定範囲内で収まっているといった，幾つかのルールに従っているかどうかで判定されることが一般的である．そして，そのルールチェックは一般的に lint と呼ばれて，コードの執筆ルールに準じているかを判定する仕組みを使うことが多い．

近年に入り，この lint のメカニズムを校正作業のように一般的な文章構成に用いようとする技術が登場している．[4][5]まだ現段階では明らかな文法の誤りなど，比較的シンプルな校正にとどまっているが，自然ご処理技術や人工知能技術の進展とともにいずれ高度な校正処理がソフトウェア的に実現する可能性は高い．そこで，本ソフトウェアでは，これらの lint 技術を一般的なホワイトカラー労働の成果物に対して適用する事を検討する．

4. MIP 導入による効果と応用可能性

4.1 モジュール化による効果

commit が修正作業を意味するモジュールであることは既に述べた．このモジュール化によって従来の直線的な作業経路に限らないホワイトカラー労働，知的生産が実現できる．例えば，校正を要する文書に対して校正を複数案用意し，最も優れた構成案を事後的に採用することなどが極めて容易に可能となる．また，提出先と形式が異なるが内容の類似した文書にする変更を一つの作業で済ませることなどが可能となる．このような効果は VCS を導入した場合に得られる重要な変化である．しかし，分散型 VCS であり，継続的改善の基盤として用いられることの多い Git 技術を導入することのメリットはそれだけにととどまらず，様々な波及効果を生み出す．本章ではこれらの効果と応用可能性について整理する．

4.2 作業ログとしての commit と可視化

リポジトリに記録されている commit を確認することで，作業者の作業ログを取得することができる．これにより，作業の進捗のリアルタイム可視化が可能になる．つまり作業者がいつの時点でどの程度の作業をしているのかを客観的に計測できることにつながり，作業者の生産性を把握することが可能である．

4.3 文書作成・データ作成プロセスの証跡記録としての活用

ログを逐次記録できる性質を利用することで，文書の真正性や作業プロセスの正当性を担保することが可能である．例えば，不正な論文作成プロセスが研究コミュニティにおける問題化していく中で，これらの記録を保持していくことで自身の正当性を保護することが可能になる．

4.4 自動校正等の自動化処理

また，commit が行われた動作をトリガーとして，様々な自動化処理が可能になる．先に述べた継続的な校正作業などはその性質を利用している．その他にも，変更があったことを関係者に通知することなどは一般的に考えられる．

4.5 知的生産の自動化に向けた履歴データとしての活用

文書の加筆や修正などの作業を細かい粒度で把握できるため，作業者の作業を変更差分として定義することが可能である．これらのデータが十分に集まることで，校正ルールの機械的学習が可能になることは十分に有り得る．これにより，ヒューリスティックに与えられない暗黙的修正ルールが生成され，いずれ機械的な校正案が自動生成される可能性は十分にある．

5. おわりに

本論では，ホワイトカラー労働の生産性向上のためのソフトウェアとして，MIP を提案する．MIP は現代的なソフトウェア開発方法論の中で生み出されてきた知的労働を効率化する幾つかの枠組みを，非ソフトウェア開発以外の分野にも適用することで，ワークフローの革新的改善を目指すソフトウェアである．発表では作成中のアプリケーションのデモを含めてディスカッションできればと考えている．

なお，この論文は mip アプリケーションを使って作成された．

参考文献

[1] 門田暁人, 伊原彰紀, 松本健一. ソフトウェアリポジトリマイニング. コンピュータソフトウェア. May 2013, vol. 30, no.

2, p.52-65.
[2] Scott Chacon. “Github Flow”. https://gist.Github.com/Gab-km/3705015, (参照 2017-02-07)
この手の議論はブランチ戦略と呼ばれる．有名なものとして Git Flow,Github Flow, Gitlab Flow が挙げられる．
[3] Martin Fowler, Matthew Foemmel. “継続的インテグレーション”. http://objectclub.jp/community/XP-jp/xp_relate/cont-j, (参照 2017-02-07)
[4] “Textlint”. https://github.com/textlint/textlint, (参照 2017-02-07)
[5] “Redpen”. http://redpen.cc/, (参照 2017-02-07)