Vol.2022-SE-210 No.2
2022/3/11

複合コミット分割支援のための対話型ステージングツールの試作

古賀 碧[1,a)]　小林 隆志[1,b)]

概要：複数のタスクを同時に含んだコミットは複合コミットと呼ばれ、開発履歴の理解や分析に悪影響を与えることが知られている．複合コミットを自動で分割する手法の研究は数多く行われてきたが、この技術を開発に応用するためには開発者が分割案を確認し部分的に手動修正できるようなツールが必要となる．一方で既存の分割ツールは分割ロジックと操作インターフェースの結合度が高く拡張性に課題がある．この問題を解決するために本研究では任意のコミット分割手法の出力結果を利用可能であり対話的にステージングを行えるコマンドラインツールを試作した．

キーワード：コミット分割，Single Task Commit, 対話的ツール，版管理システム

1. はじめに

共同的なソフトウェア開発では機能追加やバグ修正を積み重ねることによってソースコードは継続的に進化してゆく．そのようなソフトウェアの継続的な進化を支援する環境として Git や Subversion などのバージョン管理システム (VCS) が用いられる．VCS の導入は変更の段階的な保存だけでなく，変更内容の複数人でのレビューやリリースバージョンの柔軟な切り替えなどソフトウェア開発を支援する様々な機能を提供する．そのためソフトウェアの共同開発において VCS はなくてはならないものとなっている．

VCS においてソフトウェアの変更履歴を構成する基本的な単位はコミットであり，コミットはソースコードの最新バージョンからの変更の差分 (diff) とその変更の内容や意図を記述したコミットメッセージからなる．コミットの適切な粒度として機能追加やバグ修正といった単一の作業 (タスク) 単位で行うことが推奨されており [1]，これに従って作成されたコミットは Single Task Commit(以降 STC) と呼ばれる．STC となるように変更を記録することで，個々の変更内容が明確な状態で変更履歴が維持され，変更内容のレビューや差し戻し (revert) が簡単に行えるようになる [2,3]．このようにソフトウェア開発において開発者が STC を行うことの意義は大きい．

一方で実際の開発では複数のタスクを含んだ複合コミットまたは Complex Commit(以降 CC) が多く作られてしまうことが知られている．Herzig らは 5 つのオープンソースプロジェクトにおいてバグ修正コミットの最大 15%が複合コミットであったことを報告し [4], Tao らは 4 つのオープンソースプロジェクトにおいて，全コミットの平均 17%，最大 29%が CC であると結論付けている [5]．

変更履歴に CC が含まれることは開発に悪影響を与えることが知られている [6]．これは CC が変更履歴の可読性や再利用性を減少させることに加えて，変更履歴を解析して利用するような支援手法においてノイズとなるためである [7]．

これを受けて，変更箇所同士の関連度に基づいてクラスタリングを行うことで CC を STC に分割するような手法が多数提案されてきた [4,5,8]．しかしそれらの手法を使って実際にコミット分割を行うには，その出力結果を元にコミットが作成できるだけでなく，そのコミットを開発者が確認しながら修正できるようなツールが必要である．これは STC の基準はプロジェクトの規模や開発方針によって大きく左右されるため，常に開発者が求める STC を出力し続けることは不可能なためである [9,10]．

自動分割されたコミットを開発者が対話的に編集できるツールは既にいくつか存在するが [9–11]，本研究ではそのような対話的な CC 分割ツールに更に二つの要件を求める．

一つは分割にかかる工程を可能な限り少なく抑えることである．開発者が実際に CC を分割しようとするとき，一般的にはネイティブの git コマンドや SourceTree[*1]，GitKraken[*2]といった広く普及している Git クライアントツールを使用することが考えられる．分割結果を修正するツールが開発者に広く受け入れられるためには，対象としている CC の分割というタスクにおいて，そのような一般的な手法よりも工数の減少という観点で優位であることが

[1] 東京工業大学 情報理工学院
School of Computing, Tokyo Institute of Technology
[a)] koga@sa.cs.titech.ac.jp
[b)] tkobaya@c.titech.ac.jp

[*1] https://www.sourcetreeapp.com/
[*2] https://www.gitkraken.com/

重要である．

もう一つは分割に必要な情報をツールが提供し，どのようにし CC を分割するべきかを開発者に示すことである．変更の内容や規模によっては開発者がそもそも自分の行った変更の内容を完全に把握できず，どのように変更を分割するべきかに行き詰まることが先行研究によって報告されている [12]．そのため開発者が自身の行った変更の意図を正しく認識することを支援する機能が必要である．

本研究ではこれらの要件を整理し，要件を満たす機能を明らかにする．また，それらの機能を実現する複合コミット分割支援用の対話型ステージングツール C-four[*3] を試作する．

以降では，まず 2 節でコミット分割ツールの関連研究を紹介する．3 節では前述の 2 つの要件を満たすためにどのような機能が必要となるかを説明する．4 節では実装したツール C-four の機能や使用方法について説明する．5 節では実際のプロジェクトにおいて CC を STC に分割する使用シーンを想定しながら，本ツールの有用性を確認する．

2. 関連研究

既存のコミット分割支援ツールはコミット作成前に用いるか，作成後に用いるかというという観点から大きく二種類に分けられ，これらを以降それぞれ事前型，事後型と呼ぶ．

事前型のツールは開発者が CC を作成することを未然に防ぐことを目的としている．Dias らは SmallTalk 用のコミット分割ツールとして EpiceaUntangler を開発した [3]．このツールでは IDE の操作履歴に基づいた変更内容のクラスタリング結果を開発者が確認することができる．Sarocha らは変更内容をリファクタリング単位で分割し，それらを対話的に調整してコミットを作成できるツールである ChTree を開発した [11]．Zhang らはクラスタリングされた変更内容を，閾値の変更による大まかな調整とドラッグ &ドロップによる細かな調整という異なる 2 つの粒度で調整することのできるツールとして SmartCommit を提案した [10]．

一方，事後型のツールは主にコードレビュー時のレビュワーの負担を減らすことを目的としている．Tao らはプログラムスライシングに基づいて変更箇所の適切な分割案を提示する手法を提案し，開発者が実際にレビューが円滑に行えるようになったことを示した [5]．また CC を分割することではなく，CC を発見して警告することに焦点を当てたツールも存在し，それもこの事後型のツールと言える．Muylaert らはプログラムスライシングを用いて入力コミットが CC であるか否かを判別できる高速な手法な提案し，CC を検出して警告することのできるツールを開発した [8]．

今回試作した C-four は事前型のツールであるが，既存の事前型ツールの特徴として分割アルゴリズムを提供するロジックサイドと渡された情報を元に操作画面を提供するクライアントサイドの結合度が高いことが挙げられる．いずれのツールも同研究内で開発した独自の分割アルゴリズムに沿った操作インターフェースを提供しており，これは特定の利用シーンに特化した独自の機能を提供できる反面，使用言語や変更内容など分割アルゴリズムの制約を受けやすく，利用シーンが限定されやすいという欠点が存在する．

これに対し C-four はロジックサイドとクライアントサイドの結合度を弱めることで，他の分割アルゴリズムの出力情報を元にコミット分割を行うことを可能にした初めてのツールである．

3. 提案手法: C-four

3.1 概要

本研究では複合コミット分割支援用の対話型ステージングツールである C-four を提案する．本ツールは 1 でも述べたように，CC 分割ツールに求められる要件として既存研究 [9,10] でも指摘されているコミットの対話的な編集が可能であるという要件に加えて，更に以下の二つの要件を満たすことを目的としている．

- **工数の削減**: CC の分割にかかる工数を最小限に抑えることができる
- **変更意図の認識の支援**: CC に含まれる各変更意図を開発者が正しく認識することを支援する

本研究では，これらを実現するために開発者がツールを使用して以下のように作業できることが最適と考える．

(1) 与えられた変更内容が変更意図に基づいて分割される
(2) 各変更内容がコミットの変更意図に基づいているかを判断しながら変更内容を移動させる

このように，まず変更意図の明確な初期分割がなされた後，開発者がそれぞれの変更意図に各変更内容が基づいているか否かを確認しながら，基づいていない変更内容のみを移動させることで，工数の削減と変更意図の認識の支援の両者が実現できる．

以下では，開発者がこのような使い方を行うことを支援するためにツールが提供するべき機能について説明する．

3.2 変更意図の明確な初期分割

C-four では起動時にあらかじめコミットが分割された状態で用意されており，利用者はそこから chunk(後述) を移動させることで作成したいコミットに近づけていく．これをコミットの初期分割と呼ぶ．本ツールでは，この初期分割が変更内容に含まれる各変更意図に基づいて行われることを目的としている．

[*3] https://github.com/tklab-group/c-four

表 1　config.ini の例

[SECTION1]
CMD = hoge -o

3.2.1　他ツールの出力結果の利用

これを実現するためには，既に多数提案されている CC の自動分割技術 [4,5,8] を有効に活用することが最も重要であると本研究では考えた．これによりプロジェクトの規模や言語に適した技術，精度の高い最先端の技術などを簡単に導入できるためである．そのため本ツールでは他ツールの出力結果を初期分割に利用できる機能を提供する．

具体的にはコミットをどのように初期分割するべきかという情報が入った指定のフォーマットの json を受け取ることでそれに基づいた初期分割を行うことができる．この機能は初期分割に用いる自動分割の手法を簡単に切り替えることを可能としており，2 節でも述べたようにこのような機能を備えたコミット分割ツールは C-four が初めてである．

この機能を用いるための起動オプションについて説明する．このオプションには出力された json をパスを指定することで参照する path オプションと，config ファイルに指定したコマンドの出力結果を直接入力として用いる config オプションの二つがある．まず path オプションの方は入力を正しく受け取れるかや画面が正しく表示されるかなどを確認するために試験的に使うことを想定している．

一方，実際に自動分割と C-four を組み合わせて日常的に運用したい場合は config オプションが便利である．config オプションを用いるには下の事前準備が必要である．

(1) config.ini ファイルを適当なパスに作成する
(2) 1 で作成した config.ini ファイルに**表 1** のような形式で適当なセクション名と初期分割が標準出力されるコマンドを記載する
(3) configini のパスを環境変数 C_FOUR_CONFIG_PATH に設定する
(4) c-four –config hoge のように 2 で設定したセクション名を指定して起動する

起動時の引数によって config.ini に設定した複数のコマンドを自由に選択することができるため，使用する言語や変更の規模に応じて初期分割のアルゴリズムを柔軟に切り替えることが可能である．

3.2.2　デフォルトの初期分割

本ツールでは前述したような他ツールとの連携を行わなくとも，デフォルトでファイル単位と分割なしという二つの初期分割を行える．

ファイル単位の分割でも CC を高い精度で STC に分割できることは既存研究 [10] でも報告されており，これだけでも十分に変更意図が明確な初期分割を行えるものと考えている．なおオプションを指定せずに起動した場合はこの初期分割が行われる．

次に分割なしのオプションは主に現在の変更内容を確認し，場合によってはその中の一部をコミットしたいようなときに有効である．例えばある大きなタスクを行っている最中にそのタスクとは無関係な些細なバグを発見したときを考える．このとき開発者はそのバグの修正だけを一旦コミットしてしまい，元のタスクに集中したいと思うだろう．そのときこの分割なしのオプションを選択し変更されたコードを一覧で確認することで，その中からそのバグの修正コードだけを見つけ出し，元のタスクに影響を与えることなくコミットを作成することが可能である．

3.3　変更意図に基づいた変更内容の移動

初期分割が行われた後，開発者は初期分割で用意されたコミットの変更意図と，そこに含まれる各変更内容の変更意図が一致しているかを検討し，一致していない変更内容を適切なコミットに移動させることでコミットを編集する．この際に重要となるのは，変更内容をどのような粒度で解釈するか，変更内容の移動をどのように支援するのかの二点点が重要となる．以下でそれぞれについて述べる．

3.3.1　変更の粒度：chunk 単位のコミット操作

変更内容の粒度が小さすぎる場合，変更内容の個数自体が多くなることに加え，単独では明確な変更意図を持たない変更内容が増え，それぞれの変更意図を把握し続けることが困難となる．加えて操作できる変更内容が多くなるため，これは工数の増加にも繋がる．粒度が大きすぎる場合，それぞれの変更内容が複数の変更意図を含む可能性が上がる．この場合そもそも STC を作成すること自体が不可能となってしまう．以上を踏まえると単独でその変更意図が解釈可能であり，かつ複数の変更意図を含まないような適切な粒度を選択する必要がある．

以上の議論を踏まえたうえで，本ツールではコミットを構成する最小単位を連続する追加行または削除行のまとまりである chunk とした．この妥当性を検証するため，まず一般的な Git クライアントツールで用いることのできる行単位，Hunk 単位，ファイル単位という三つの粒度を考える．

行単位でコミットを解釈することは，メソッドのスコープを示す括弧だけの文や可読性を高めるための空行など単独では明確な意味を持たない変更についても操作粒度として認めることになる．それらは CC を分割するにあたってノイズとなるため，分割に必要な工数が増えるだけでなく，明確な意味を持たない変更内容が画面上に多数表示されることになり，開発者が変更意図を正しく認識することを阻害される．一方でファイル単位，Hunk 単位は同時に複数の変更意図を含む可能性があり，コミットの構成単位として用いるには粒度が大きい．

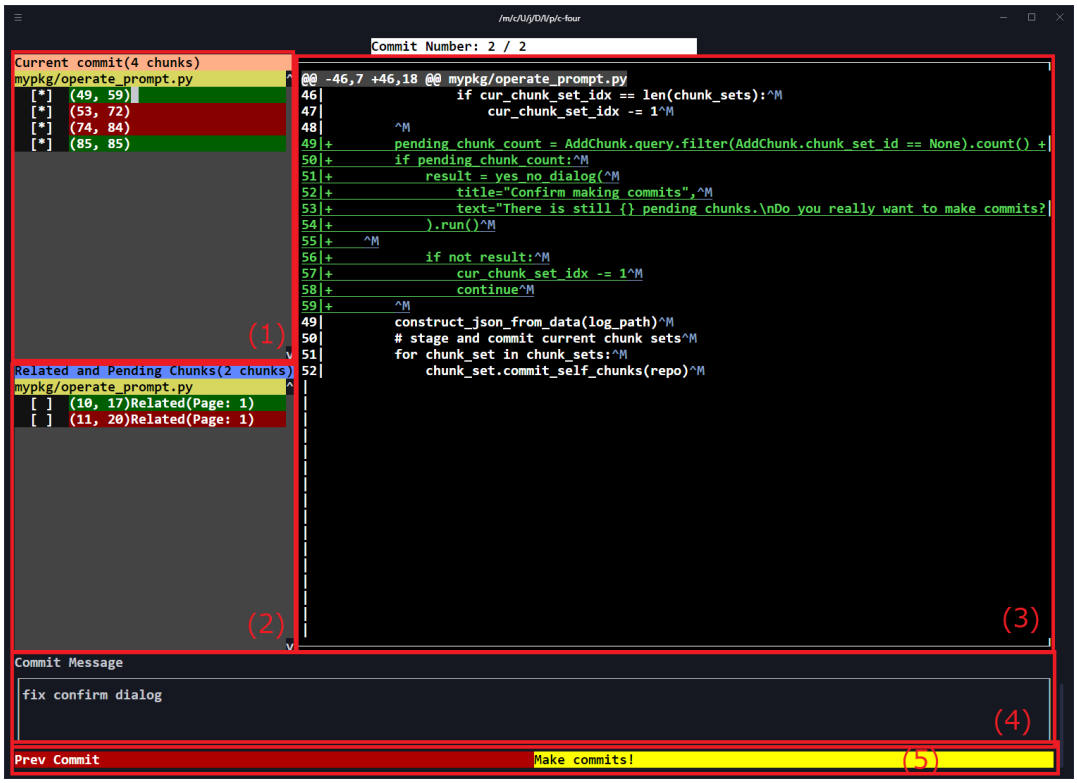

図 1　コミット編集画面

そのため，行単位とファイル・Hunk 単位の中間の粒度である chunk 単位が最も適切な粒度であると考えた．実際に chunk 単位で CC の分割を行った既存研究が存在しており [13, 14]，これは chunk がコミットを構成する上での必要十分な粒度であることが示している．よって本ツールにおけるコミットの最小の構成単位を chunk としている．

3.3.2　移動の支援：関連 chunk の表示

変更内容同士には様々な関連性があり，開発者はそれらを確認しながら変更を移動する必要がある．各 chunk と同じコミットに含まれる可能性が高い chunk を常に確認することができる機能を提供することで，移動の際に考慮するべき chunk を容易に識別することが可能となる．これは chunk 同士の関係性が見落とされることで CC が正しく分割されない問題を防ぐことも目的としており，実際に変更箇所同士の関係性を可視化することの重要性は先行研究 [11] でも言及されている．この機能により，高い関連度を持つが初期分割では同じコミットに含まれていないという chunk 同士の関係性が見落とされることがなくなり，変更意図に基づいた変更内容の移動が正しく行われる可能性が高まることが期待される．

3.3.3　移動の支援：chunk の保留

コミットの変更意図と各 chunk の変更意図の一致を確認するにあたって，今のコミットに含めるべきではないがどのコミットに含めるべきかはまだ分からないという chunk が出てくることが考えられる．そのような chunk に対応するため，本ツールではそのような chunk をどのコミットにも含めない状態にすることができる．これが本ツールが変更意図に基づいた変更内容の移動を支援するために提供するもう一つの機能である chunk の保留である．これにより chunk をどのコミットに割り当てるべきかという整理を開発者が行いやすくなることが期待される．

4.　ツール詳細：*C-four*

本節では，3 節の提案手法を実装した対話型ステージングツール C-four について説明する．C-four は python 言語で実装されており，Python Prompt Toolkit を用いてコンソール上での対話的な操作を実現している．Git で版管理されているディレクトリ下で `c-four` コマンドをコマンドラインから実行することで起動する．

C-four の起動時にはまず，3 節で説明したように起動のオプションに基づいてコミットの初期分割が行われ，**図 1** のような画面がコンソールに表示される．この画面を以後コミット編集画面と呼ぶ．

1 つのコミット編集画面は 1 つのコミットに対応しており，作成するコミット数分のコミット編集画面が存在する．コミット編集画面では chunk をコミット間で移動したり，コミットを追加・削除していきながらコミットの内容を編集していく．以後，コミットを構成している chunk を構成 chunk，コミット編集画面で現在編集しているコミットを対象コミットと呼ぶ．コミット編集画面は以下の 5 つの部分から構成される．

(1) 構成 chunk 一覧

(2) 関連, 保留状態の chunk 一覧
(3) diff 確認画面
(4) コミットメッセージ編集画面
(5) コミット移動ボタン
各部分の役割と機能について説明する.

構成 chunk 一覧

この部分には対象コミットの構成 chunk が一覧で表示される. 各 chunk 上で Enter キーを押すとその chunk の内容が diff 画面に表示されるほか，対象の chunk 上で対応するキーを押すことでその chunk の割り当てを変更することができる. chunk の割り当てと対応するキーは以下の通りである.

- a キー: chunk を対象コミットに割り当てる，デフォルトの状態
- d キー: chunk を保留状態 (後述) に割り当てる
- p キー: chunk を 1 つ前のコミットに割り当てる，対象コミットが最初のコミットのときは a キーと同様
- n キー: chunk を 1 つ後のコミットに割り当てられる，対象コミットが最後のコミットのときは a キーと同様

実際に chunk の割り当てが変更されるのは別のコミット編集画面を移動するときなので，それまでは chunk の割り当ては何回でも変更可能である.

関連，保留状態の chunk 一覧

関連 chunk および保留 chunk の両方が一覧で表示される. 関連 chunk, 保留 chunk のどちらとも，chunk 上で a キーを押すと対象コミットに割り当てられ，d キーを押すことで割り当てが解除される.

diff 画面

選択した chunk の diff の内容が表示される.

コミットメッセージ編集画面

対象コミットのコミットメッセージの確認・編集が行える.

コミット移動ボタン

一つ前，または一つ後のコミット編集画面に移動できる. 対象コミットが最後の場合，1 つ後のコミット編集画面に移動するためのボタンはコミットの作成開始のためのボタンに置き換わる.

またコミット編集画面では ctrl + v キーで空のコミットの追加，ctrl + s キーで対象コミットの削除が行える.

5. 評価

ここでは実際の CC 分割を想定したケーススタディを行い，本ツールの提唱する，変更意図の明確な初期分割と変更意図に基づいた変更内容の移動という 2 つのステップで CC 分割が正しく行えることを確認する.

5.1 実験内容

本実験ではプルリクエストを CC, それらに含まれる各コミットを STC とみなし，あるプルリクエストに含まれる変更内容を C-four を用いてそれらを元のコミットらに分割するというタスクを行う.

まず実際の CC では粒度の高い共通の変更意図が存在することが考えられる. 同一プルリクエスト内に含まれるコミットは共通の変更目的を持っており，そのためそれらを統合したものは実際の CC を模倣していることが期待できる.

次にそれらに含まれるコミットを STC とみなすためには，それらが単一の変更意図のみを含むことが保証されていることが望ましい. そこで使用するリポジトリは Antangler[*4]を選択した. これは当リポジトリの Contribute のガイドライン[*5]にて単一の変更意図のみを含むコミットを作成することが規定されており，各コミットが STC であることが保証されていると判断したためである.

本実験では対象リポジトリの PR#44862[*6]を使用する. 当 PR は変更行数が 74 行，変更ファイル数が 2，コミット数が 2 であり，二つの変更ファイルはある実行ファイルとそのテストコードという関係にあるためこれらをそれぞれ script, test と呼ぶ. そして二つのコミットはどちらとも script, test の両方を変更している.

1 つ目のコミットでは script にある条件文を追加し，test ではそれに伴うテストコードの変更を行っている. 2 つ目のコミットでは'strictNullChecks' という定数を script に導入し，test では同じくそれに伴うテストコードの変更を行っている. 説明のため以降これら二つのコミットをそれぞれ first, second と呼ぶ.

5.2 実験結果

まず初期分割はファイル単位で行う. このとき二つのコミットが作成されそれらはそれぞれ A, B の変更内容のみを含む.

次に実際の分割を見ていく. まず一つ目の A の変更内容を含んだコミットには**図 2** のように 3 つの chunk が含まれる. それらの内容を確認すると，一番上の chunk では first のメインの変更意図である条件文の追加，下の二つの chunk では second のメインの変更意図である'strictNullChecks' という定数の追加を行っていることが分かる. そのためこのコミットでは first のメインの変更意図である一番上の chunk のみを残し，下の二つの chunk を一旦保留する. 本コミットは一旦これ以上編集が行えないので，コミットメッセージを入力して二つ目のコミットの編集に移る.

二つ目のコミットには**図 3** のように 19 個の chunk が含まれるが，このうち 9 個の chunk に'strictNullChecks' と

[*4] https://github.com/angular/angular
[*5] https://github.com/angular/angular/blob/master/CONTRIBUTING.md
[*6] https://github.com/angular/angular/pull/44862

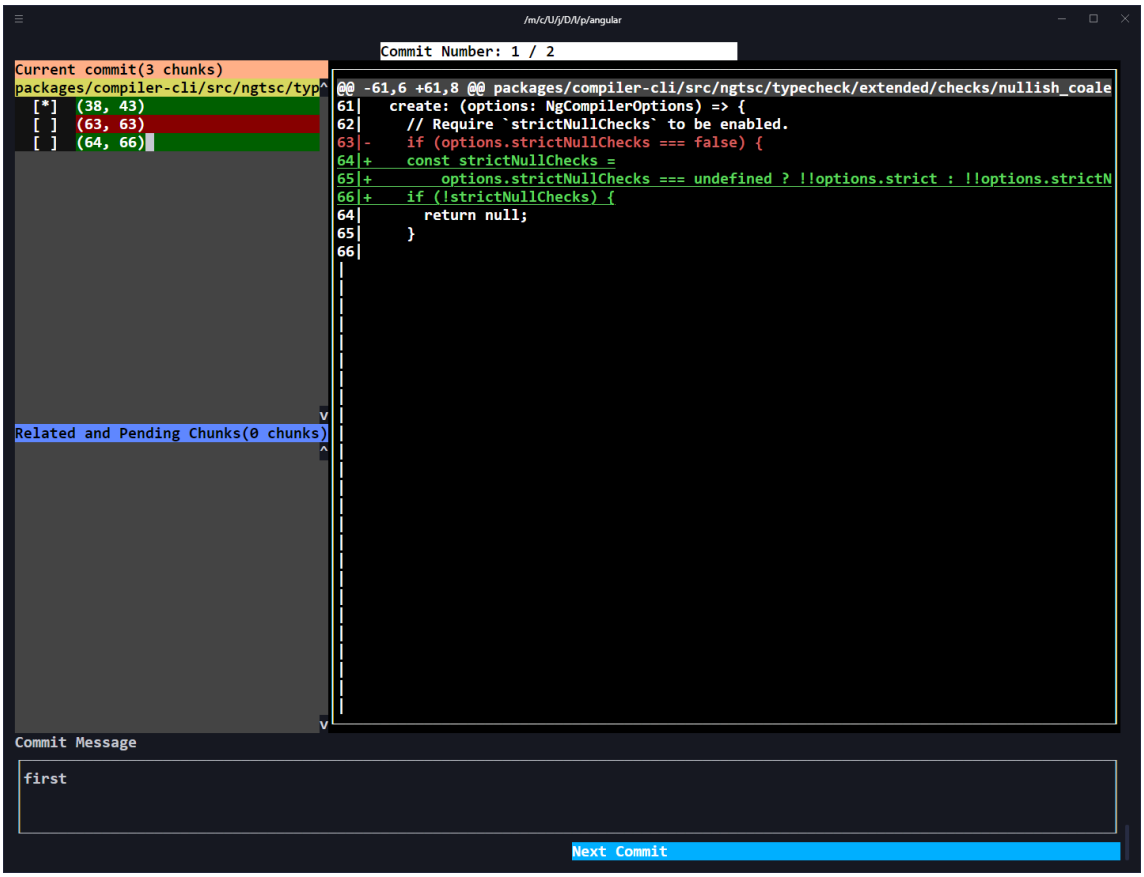

図 2　first の編集画面

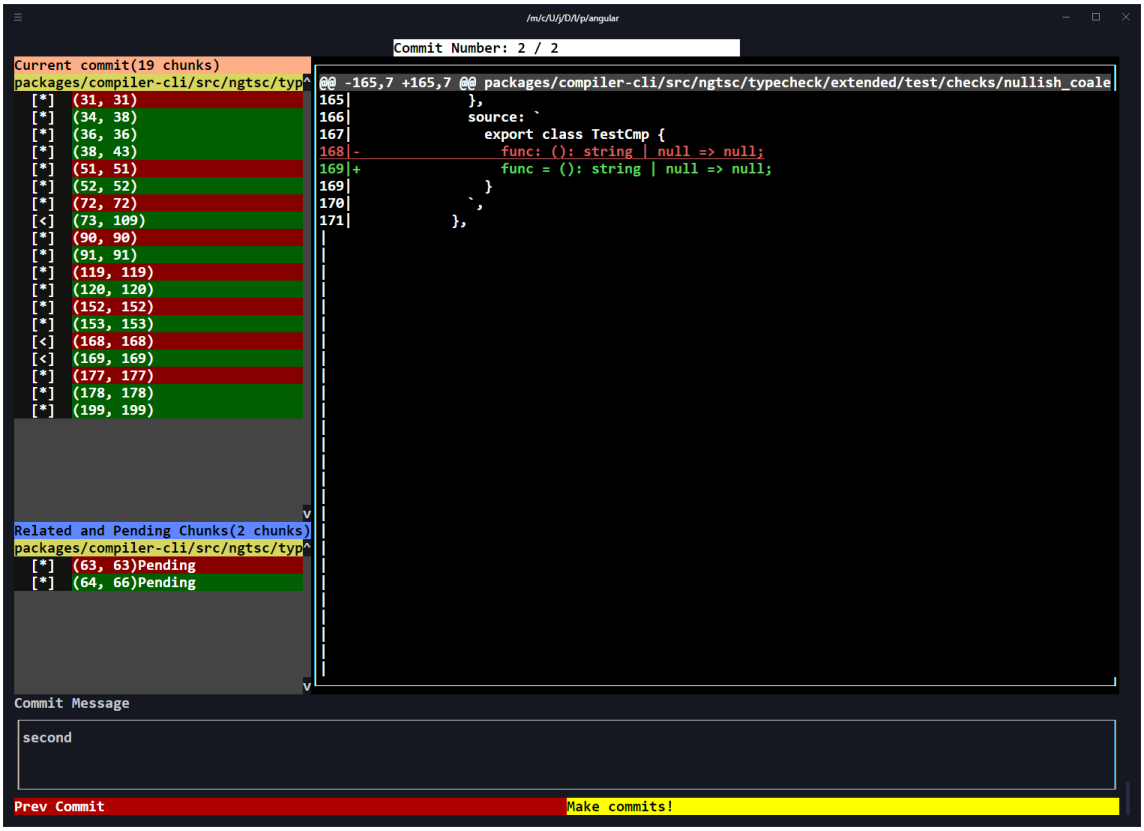

図 3　second の編集画面

いう単語が含まれており，このことからこのコミットのメインの変更意図は定数'strictNullChecks' の導入とそれに伴うテストコードの変更だと認識することができる．つまり先ほど保留にしていた二つの chunk はこのコミットの変更意図と一致しているため，このコミットに含めるという判断が可能である．同様に second ではなく first の変更意図に一致する 3 つの chunk も見つけ出すことができ，前のコミットに移動させることができる．

この時点で second だけでなく first も同時に目的の変更内容になるため，コミットメッセージを入力してから右下の"Make Commits!"のボタン上で Enter を押すことでコミット作成が開始され正しくコミットを分割することができる．

このように C-four を用いることで開発者は全ての変更意図を自分で認識するのではなく，初期分割によって示された変更意図に各 chunk が従っているかという観点からコミットを整理することが可能である．そして本 PR では 74 行の変更を行っているが，本ツールを用いることで計 5 回の chunk の移動のみで正しい分割を行うことができている．

これは変更意図の明確な初期分割と変更意図に基づいた変更内容の移動という 2 つのステップで CC 分割が正しく行えたことで，工数の削減と変更意図の認識の支援という両者が実現したことにほかならない．そしてこれはデフォルトの初期分割や chunk の保留といった各機能がそれらの実現に有効に機能したことを示している．

6. おわりに

本研究ではまずソフトウェア開発において発生する複合コミット (CC) という問題について，その分割を支援するようなツールには分割における工数の削減と変更意図の認識の支援という二つの要件が求められることを示した．そしてそれを実現するために変更意図の明確な初期分割と変更意図に基づいた変更内容の移動という二つの手順で CC 分割を行える CC 分割支援ツールである C-four を提案した．

その後 C-four がそれらの実現のために持つ各機能を提示し，それらの機能が実際に有効に働くことを確認するために実際の CC 分割を模倣したケーススタディを行った．結果として，変更意図の明確な初期分割と変更意図に基づいた変更内容の移動という二つの手順で正しく CC を分割することが可能であり，工数の削減と変更意図の認識の支援という二つが実現できていることを確認した．

今後の展望として，まず実際に CC の自動分割技術との連携を行うことが挙げられる．chunk 単位での CC 分割が可能な既存研究 [13, 14] を改良し本ツールで利用可能な json を出力できるようにすることで，より高い精度で初期分割を行うことが可能になり本ツールの有用性は更に向上するものと考えられる．

また本ツールの利用ログから，実際の分割時の操作履歴や分割にかかった時間などの利用情報を得ることが考えられる．実際の CC 分割に基づいて得られたこれらの情報は非常に実用的なものであり，分割に用いる新たなメトリクスやより実践的なデータセットとして本分野の更なる発展に有効に活用できることが期待される．

謝辞 本研究の一部は科研費（#18H03221）の助成を受けた．

参考文献

[1] Berczuk, S. P. and Appleton, B.: *Software Configuration Management Patterns*, Addison-Wesley Professional (2002).
[2] Barnett, M. et al.: Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets, *Proc. ICSE* (2015).
[3] Dias, M., Bacchelli, A., Gousios, G., Cassou, D. and Ducasse, S.: Untangling fine-grained code changes, *Proc. SANER* (2015).
[4] Herzig, K. and Zeller, A.: The impact of tangled code changes, *Proc. MSR* (2013).
[5] Tao, Y. and Kim, S.: Partitioning Composite Code Changes to Facilitate Code Review, *Proc. MSR* (2015).
[6] Herzig, K.: Mining and untangling change genealogies (2012).
[7] Nguyen, H. A., Nguyen, A. T. and Nguyen, T. N.: Filtering noise in mixed-purpose fixing commits to improve defect prediction and localization, *Proc. ISSRE* (2013).
[8] Muylaert, W. and Roover, C. D.: Untangling composite commits using program slicing, *Proc. SCAM* (2018).
[9] Yamashita, S., Hayashi, S. and Saeki, M.: ChangeBeadsThreader: An Interactive Environment for Tailoring Automatically Untangled Changes, *Proc. SANER* (2020).
[10] Bo, Zhang et al.: SmartCommit: A Graph-Based Interactive Assistant for Activity-Oriented Commits, *Proc. FSE* (2021).
[11] Sothornprapakorn, S., Hayashi, S. and Saeki, M.: Visualizing a Tangled Change for Supporting Its Decomposition and Commit Construction, *Proc. COMPSAC* (2018).
[12] Tao, Y., Dang, Y., Xie, T., Zhang, D. and Kim, S.: How Do Software Engineers Understand Code Changes? An Exploratory Study in Industry, *Proc. FSE* (2012).
[13] 眞田, 小林：変更個所の構造的特徴の学習に基づく複合コミットの分割，信学技報 SS2020-27，電子情報通信学会 (2021).
[14] Li, C. and Kobayashi, T.: Untangling Composite Changes Using Tree-based Convolution Neural Network, Technical Report SS2020-46, IEICE (2021).