Vol.2010-SE-167 No.6
2010/3/18

バリエーション並行開発のための版管理ツールと統合開発環境

横山祐司[†1] 日高隆博[†2] 山本晋一郎[†1]
小林隆志[†2] 手嶋茂晴[†2] 阿草清滋[†2]

C 言語により多品種開発を行う際に条件付きコンパイルを用いる．このようなソースコードはバリエーションの増加に伴い，各バリエーションを区別するための，コンパイル条件や条件分岐が増え，可読性や保守性が低下する．本論文では，上記のようなソースコードの可読性や保守性を向上させることを目的とし，版管理ツールのバージョン軸とブランチ軸に対してバリエーション軸を追加することによる新たなバリエーション管理手法の提案する．さらに，提案手法に基づく支援ツールに関して述べ提案手法の有効性を議論する．

Version management tool and integration development environment for multi-variant development

Yuji Yokoyama,[†1] Takahiro Hidaka,[†2]
Shinnichiro Yamamoto,[†1] Takashi Kobayashi,[†2]
Shigeharu Teshima[†2] and Kiyoshi Agusa [†2]

The conditional compilation is usually used in multi-variant development by C language. The increase of variations often lead to a decline in readability and maintainability of its source code because variables for compilation conditions and conditional branches in its source codes to discriminate variants are increased. In this paper, it aims to improve the readability and the maintainability of the multi-variant source code. We proposed a new variation management method based on a new management axis of the version management tool. We add the “variation axis” into existing two axis, the version axis and the branch axis. We also developed a supporting tool based on our proposal method and discuss about efficiency of proposal method.

1. はじめに

近年，大規模かつ複雑なソフトウェアが開発されており，それに伴う開発コストの増大は企業にとって非常に深刻な問題の一つとなっている．特に組込みソフトウェア分野においては、多くの製品に対応するための多品種開発が重要となってきており、ソフトウェア部品を効率的に再利用しながら製品ごとの構成を管理するための手法が求められている．C 言語により多品種開発を行う際は，開発の効率化のために一般的に条件付きコンパイルを用いる．ソースコード上の部品がコンパイラに取り込まれるための条件をマクロを用いて設定し，プリプロセッサ命令で切り分ける．本論文では，このマクロをコンパイルスイッチと呼ぶ．コンパイル時に目的のバリエーションに対応するコンパイルスイッチの組み合わせを与えることで，1 組のソースコードから複数バリエーションのソフトウェアを生成することが可能となる．このようにして，既存のソフトウェアを再利用し，その一部と新規開発したソフトウェア部品を組み合わせて多品種開発を実現している．

しかし，上記のような方法で記述されたソースコードは，バリエーションの増加に伴い，プリプロセッサ命令の切り分けが多くなる．このとき，あるバリエーションに着目して開発を行うとすると，関係のない他のバリエーションのソースコードが混在して記述されていて，可読性や保守性が低下してしまう．このことは，C 言語の多品種開発において，大きな問題となっている[1)]．

本論文では，上記のようなソースコードに対して，可読性や保守性の低下という問題を解決することを目的とし，バリエーション管理手法を提案する．従来の版管理ツールでは時間軸としてバージョンを扱い，管理対象の分岐をブランチとして扱っているが，我々は新たにバリエーションを管理対象として追加する．本提案を適用することによって従来はソースコード中に埋め込まれていたバリエーションを明示的に分離することで保守性を高め，編集時には着目したいバリエーションのソースコードのみを抽出して編集を行い，編集完了後に統合することによって可読性の問題を解決することが可能であると考えている．

これにより，バリエーションごとの版管理が可能となる上に，可読性の高いソースコードでの編集が可能となる．

†1 愛知県立大学大学 Aichi Prefectural University
†2 名古屋大学 Nagoya University

2. 多品種開発における構成管理技術

2.1 コンパイルスイッチによる構成管理

C 言語で記述されたソフトウェアにおいて多品種開発を行う場合，条件付コンパイルを用いる方法が一般的である．C 言語のソースコードは，コンパイルの前にプリプロセッサにより前処理が施される．プリプロセッサで行われる処理の 1 つに条件付きコンパイルがあり，1 組のソースコードから複数種類のソフトウェアを生成するためによく用いられる．

例えば，あるバリエーションでは，関数 `foo()` が，別のバリエーションでは関数 `bar()` が呼び出される場合には，バリエーションを表すマクロを定義しているかどうかを条件として，呼び出される関数を切り替える記述を行う．この条件付コンパイルの条件として用いられているマクロのことを本論文ではコンパイルスイッチと呼ぶ．コンパイルスイッチを適宜切り替えることで，任意のバリエーションのソースコードの生成を一組のソースコードから生成できる．車載ソフトウェア等の多品種開発では，このコンパイルスイッチを利用し，開発対象の機能や仕様，実行環境に合わせた多品種のソフトウェアを一組のソースコードから生成することで実現している．

2.2 条件付きコンパイルの問題点

条件付コンパイルは，多品種開発をおこなう上で非常に有効な機能であるが，開発対象ソフトウェアの規模とバリエーション数に比例してコンパイルスイッチが増加していき，それに伴い以下のような問題が発生してくる．

- **可読性の低下**
  バリエーション間の差の箇所のみをコンパイルスイッチで分岐する記述を行うため，プログラムの流れを追うことや，変更すべき箇所を特定することが困難となる．例えば，**図 2.2** のコードでは，if 文がどのようなコードになるかを理解することが難しくなってしまっている．
- **保守時の安全性の低下**
  変更箇所によっては他のバリエーションに影響を及ぼす可能性がある．仮に**図 2** において，コンパイルスイッチ B が ON の時のみ，statement2 を statement2' と変更を行いたいとする．このような変更は**図 3** のように変えるべきである．しかしながら，コンパイルスイッチを記述せずに変更すると，全てのバリエーションの処理が変更されてしまうことになる．他のバリエーションについて熟知していないユーザが保守する場合は，このように他のバリエーションに影響を与えてしまう可能性があり，危険である．

```
1    case ACC_STATUS_ADAPT:
2  #ifdef WIDE_SPEED_RANGE
3        if (!aucUserACCOn
4            ||  astVehicleStatus.b1BrakeControlOn
5  #ifndef KEEP_STOP
6            ||  assMyVehicleSpd <= ACCSTOP_CURRENT_SPEED_VALUE
7  #endif
8            ||  assUserTgtSpd <= ACCOFF_TARGET_SPEED_VALUE
9  #ifdef ACC_CANCEL_BY_WIPER
10           ||  astVehicleStatus.ucWiperSpeed >= ACC_CANCEL_WIPER_SPEED;
11 #endif
12       ) {
13           aucNextStatus = ACC_STATUS_OFF;
14 #ifdef KEEP_STOP
15       } else if(assMyVehicleSpd <= ACCLOW_CURRENT_SPEED_VALUE){
16           aucNextStatus = ACC_STATUS_LOW;
17 #endif
18       ...
19 #else
20       ...
```

図 1 可読性が低下しているコード
Fig. 1 Code to which readability has decreased

```
1  #ifdef A
2        statement1;
3        statement2;
4        statement3;
5  #endif
```

図 2 変更前のコード
Fig. 2 Code before change

```
1  #ifdef A
2        statement1;
3  #ifdef B
4        statement2';
5  #else
6        statement2;
7  #endif
8        statement3;
9  #endif
```

図 3 変更後のコード
Fig. 3 Code after change

2.3 版管理システムの利用

ソフトウェア開発において，ソースコードやその他のデータを管理するシステムとして版管理システムがある．これは管理されているファイルに対して,「いつ」「誰が」「どんな」変更を行ったかを記録しておくものである．開発者はその記録を後から参照でき，最新版だけでなく古い版も取りだすことができる．中でも広く使用されている Subversion[3] をここでは取り上げ，今後，版管理システムとは Subversion を指す．

Subversion を用いてバリエーション開発を行う場合，考えられる管理手法として次に述べる二つがある．一つ目に，条件付きコンパイルを用いて，trunk のみで管理する方法，二つ目に，バリエーションごとにブランチを作成し，バリエーションごとに版管理を行う方法である．一つ目の方法は，前節で述べた方法であり，結果として既に述べた問題が起こる．また，trunk のみで管理するため，バリエーションごとの版管理は不可能である．二つ目の方法は，バリエーションごとにブランチを作成するため，バリエーションごとに版管理ができるというメリットがある．しかしながら，複数のバリエーションに関わる共通部分に変更が生じたと場合，その変更内容を関係する全てのソースコード群に反映させなければならない．しかしながら，バリエーション毎にブランチを作成してからの期間がたつと，変更の波及範囲を正確に把握し，正しく変更を行うことは非常に困難であり，新たなバグを生む原因にもなる．それぞれメリットとデメリットがあるが，二つ目の方法はバリエーションが多くなると作業量が多くなり，バリエーションが多くなる多品種開発には不向きである．結果として，多品種開発では，2.2 で述べた条件付きコンパイルが多く用いられる．

2.4 プリプロセッサを利用したバリエーション管理

バリエーション管理を行う方法としてプリプロセッサを利用するものがある[2]．プリプロセス処理を改善し，クエリ言語 FQL を用いてバリエーションを指定することで，特定のバリエーションがどのように他のバリエーションに影響を及ぼすかの分析を行う．分析を行うことで，管理コストの削減，保守性の向上を目指している．しかし，バリエーションに対するアノテーションを記述しなければならず，従来の記法と差があるため，導入コストがかかるという問題点がある．また，プリプロセッサでバリエーション管理を行う場合の問題として、共通部分とバリエーション部分の関連を見ながら編集するために保守時の可読性が低いことが挙げられる．

3. バリエーション並行開発手法

本論文では，前節で述べたようにプリプロセッサ命令の切り分けが随所に現れる可読性の低い C ソースコードに対して，ソースコードの可読性を高めて，バリエーション開発を行う手法を提案する．

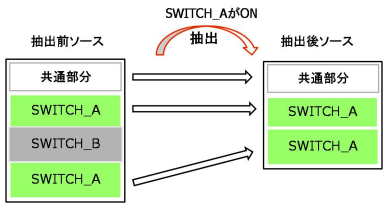

図 4 バリエーション抽出
Fig. 4 Variation Extract

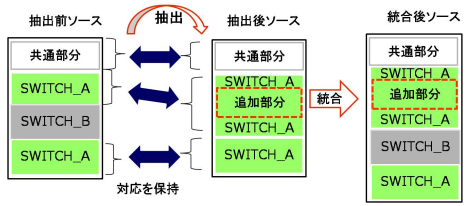

図 5 バリエーション統合
Fig. 5 Variation Integrate

3.1 バリエーション管理への要求

まず，バリエーション並行開発において必要となる，バリエーション抽出とバリエーション統合について述べ，必要な機能を議論する．

バリエーション抽出 ソースコードには複数のバリエーションのソースコードが記述されているため，あるバリエーションに着目したい場合に，その処理の流れが追いにくくなる．また，着目しているバリエーションのソースコードにおいて検索をしたときに，関係のないバリエーションのソースコードは検索対象から除外したいという要求がある．そこで，着目したいバリエーションに関係のあるソースコードのみが現れるソースコードを生成する．これをバリエーション抽出と呼ぶ．例を**図 4** に示す．

バリエーション統合 バリエーション抽出により生成されたソースコードを変更した後，その変更内容を抽出前のソースコードに反映させる必要がある．これをバリエーション統合と呼び，例を**図 5** に示す．これを実現するためには，抽出後のソースの行と抽出前のソースコードの行の対応が必要である．この情報は抽出後のソースコードのみから得ることはできない．そこで，抽出時に行の対応を記録し，そのファイルを参照することで統合を行う．行の対応などの情報が記録されるファイルをコード構成ファイルと呼ぶ．なお，コード構成ファイルは抽出後のソースコードの変更に伴って，変更されなければならない．

3.2 バリエーション管理とブランチ管理の類似性

我々は 3.1 で述べた「バリエーション抽出，編集，バリエーション統合」といった流れが，版管理システムにおける「Checkout，編集，Commit」の流れと類似していることに着目した．版管理システムでは，ブランチを作成することにより，新たな系列を作られる．バリ

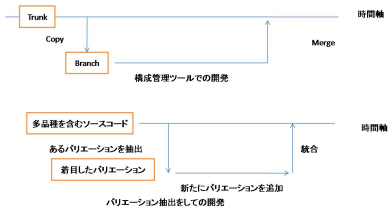

図 6 「ブランチ作成～マージ」と「バリエーション追加～統合」の比較
Fig. 6 Comparison between「branch create-merge」and「variation add-Integrate」

エーションに対しても，同様に考えることができ，新たにバリエーションを追加することは，ブランチとしての管理ではなく別の区分において新しい系列を作ることと見なすことができる．ブランチとバージョンの比較を**図 6** に示す．

3.3 バリエーション並行開発手法の提案

前節までの議論を元に，我々はバリエーション並行開発手法を提案する．提案手法の概要図を**図 7** に示す．版管理ツールではバージョン軸とブランチ軸の二つのみであったが，提案手法ではそこにバリエーション軸を追加する．

開発を行う際は，着目したバリエーションを抽出して開発を行う．バリエーションを抽出することで，可読性，保守性の低下は避けられる．また，このように版管理ツールにバリエーション軸を追加することで，ソースコードを一元管理したままバリエーションごとの版管理を行うことが可能となる．

本研究が提案するバリエーション管理手法では，開発パターンは三つになる．以下にそれらの開発パターンについて述べる．

3.3.1 開発パターン 1：バリエーションに固有の処理変更

この開発パターンは，あるバリエーションにおける処理を変更する場合に適用する．

まず，リポジトリからソースコードを Checkout して，処理を変更したいバリエーションの抽出を行う．Checkout から抽出までの操作を VariationCheckOut と呼ぶ．その後，ローカルで編集を行う．この編集には，コード構成ファイルにアクセスする機能を持ったエディタを用いる．編集が終わったら，リポジトリのソースコードとバリエーション統合をし，Commit を行う．バリエーション統合から Commit までの操作を VariationCommit と呼ぶ．バリエーションを指定してチェックアウトを行うため，バリエーションごとの版管理が可能となる．

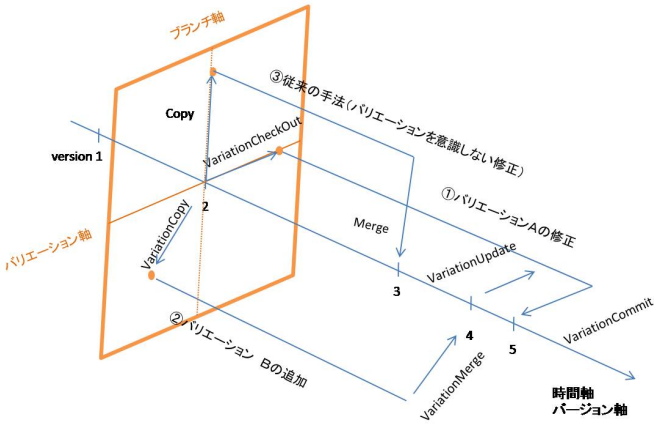

図 7 バリエーション並行開発
Fig. 7 Variation Parallel development

3.3.2 開発パターン 2：新規バリエーションの追加

この開発パターンは，新たにバリエーションを増やす場合に適用する．ブランチ開発を行いたい場合は最初にブランチを作成する．その後はバリエーションの処理変更と同様に，VariationCheckout を行う．そして新たにコンパイルスイッチを追記することでバリエーションを増やし，VariationCommit を行う．

3.3.3 開発パターン 3：従来の開発手法

その他バリエーションに着目せず修正や開発したい場合などは，通常の開発どおり，ブランチを作成しマージする．

抽出したソースコードでは，ソースコード上にコンパイルスイッチは記述されておらず，各行の条件はコード構成ファイルを元にエディタの機能で確認する．

4. バリエーション管理ツールと統合開発環境

本研究では，提案手法での開発パターンを支援するために，ソースコードの抽出や統合を行う**バリエーション管理ツール**と，抽出されたコンパイルスイッチの見えないソースコードで編集を行うための**統合開発環境**を開発した．**図 8** にそれらの概要を示し，各機能について説明する．

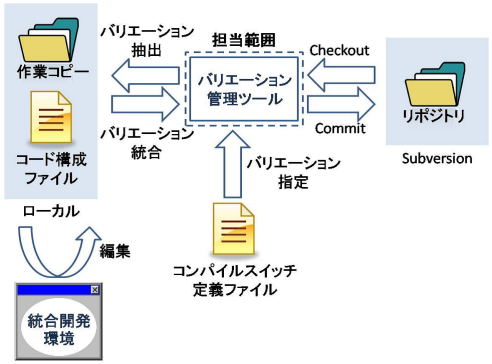

図 8 バリエーション管理ツール及び統合開発環境の概要
Fig. 8 Outline of variation management tool and integrated development environment

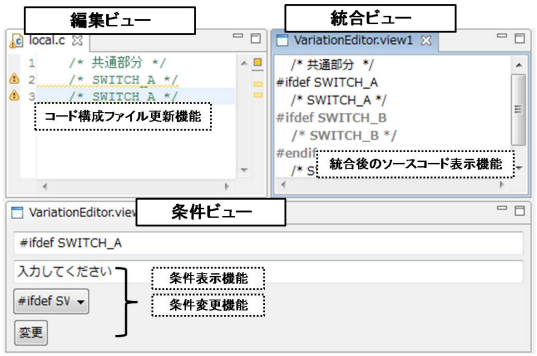

図 9 統合開発環境の概要
Fig. 9 Outline of integrated development environment

4.1 バリエーション管理ツール

バリエーション管理ツールは，C 言語とシェルスクリプトで実装されており，ソースコードの規模は約 2,500 行である．本ツールは機能毎にシェルスクリプトを実行することで使用する．バリエーション管理ツールの機能である Variation Checkout と Variation Commit の詳細を以下に示す．

4.1.1 Variation Checkout

リポジトリからソースコードを Checkout し，そこから着目したいバリエーションのソースコードのみを抽出する．コンパイルスイッチ定義ファイルにより指定されたコンパイルスイッチが真であり，指定されなかったコンパイルスイッチが偽のときに有効となるソースコードを抽出する．

- **入力**
  リポジトリのソースコード，コンパイルスイッチ定義ファイル，コピー先のアドレス (任意)，ユーザレベル (任意)
- **出力**
  抽出後のソースファイル，リポジトリのソースファイル，コード構成ファイル

4.1.2 Variation Commit

Variation Checkout により抽出されたソースコードをコード構成ファイルを参照しながら，抽出前のソースコードに統合し，Commit する．着目していたバリエーションに関する部分は変更を反映させ，その他には何も変更しない．

- **入力**
  抽出後のソースファイル，抽出前のソースファイル，コード構成ファイル
- **出力**
  統合後のソースファイルが Commit される

4.2 統合開発環境

統合開発環境は Java 言語で記述されており，Eclipse プラグインとして実装した．ソースコードの規模は約 2,000 行となっている．また外部ツールとして，Eclipse CDT 3.1.2[4] を利用した．**図 9** に統合開発の概要を示す．統合開発環境は編集ビュー，統合ビュー，条件ビューの 3 つのビューから成り立っている．

4.2.1 編集ビュー

編集ビューではリポジトリとの対応を保持するコード構成ファイルを自動で更新する機能を持つ．

- **コード構成ファイル更新機能**
  新しく改行コードが挿入された場合，リポジトリとの対応を更新しなければならないが，どのように統合していいか分からない場合がある．図 10 に例を示す．
  図 10 では，統合の際に抽出されていないコードが間にあるため，どちらに統合していいかがわからなくなる．そこで，統合する位置は，改行コードが挿入された位置で判断する．新しく生成されたコードは，改行コードが挿入された位置と同じブロックに

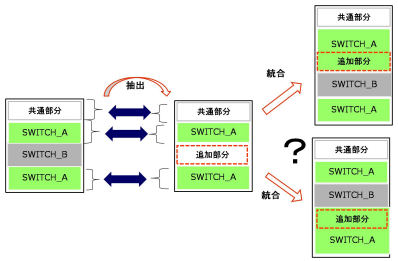

図 10 対応を取る際の問題点
Fig. 10 Ploblem when take correspondence

なる．図 10 では，上のブロックで改行コードが挿入された場合は上のブロックと同じブロックになり，下のブロックで改行コードが挿入された場合は下のブロックと同じブロックになる．

また改行コードが削除された場合，削除された行の条件は削除される．

4.2.2 条件ビュー

条件ビューは条件の表示機能，条件の変更機能を持つ．

- **条件の表示**
  抽出されたソースコードではコンパイルスイッチはソースコード上に記述されていないため，カーソルの位置の行のコンパイルスイッチの条件をビューに表示する．また，コンパイルスイッチの条件が変わる境界値，間にリポジトリのソースコードが統合される場合はマーカーにより表示される．このマーカーにより，意図した編集や，条件変更と違う結果になることを防ぐことができる．
- **条件の変更**
  カーソルで選択された行のコンパイルスイッチの条件を入力された条件に変更する．但し，条件境界，リポジトリのソースコードを間に含む場合は変更できない．条件を入れ子関係にしたい場合は，「親の設定」のリストから親を選んで変更する．

4.2.3 統合ビュー

統合ビューでは VariationCommit 後のソースコードを表示する．

- **統合後のソースコード表示**
  リポジトリと統合した後のソースコードを表示する．編集を行っている最中に逐次更新され，リポジトリの部分は灰色で表示される．このビューにより，抽出されていないソースコードも確認可能である．また，変更が意図したものになっているか確認が可能である．

表 1 ユーザによる権限
Table 1 Authority by user

ユーザ		編集	条件変更機能
ユーザレベル 1	他のバリエーションとの共通部分	×	×
	バリエーション特有の部分	○	×
ユーザレベル 2	他のバリエーションとの共通部分	○	○
	バリエーション特有の部分	○	○

図 11 ユーザレベル 1
Fig. 11 User level 1

4.3 ユーザ権限による機能制限

ここまで提案した機能を全てのユーザが使えてしまうと 2.2 で述べたように，他のバリエーションとの共通部分を変更してしまった場合，他のバリエーションに予想もしない影響が出てしまう可能性がある．この問題を解決するために，ユーザの熟練度によって使用できる機能を制限する権限を区別することが必要である．権限を導入することで，他のバリエーションに対する影響を小さくすることができ，安全に保守を行うことができる．本研究で提案するユーザ権限と制限される機能の関係を**表 1** に示す．

表 1 では，バリエーションについて熟知している度合いにより，ユーザを分けて考えている．ユーザがバリエーションに関して理解が浅い場合（ユーザレベル 1）は，他のバリエーションに影響が出るようなことはできない．よって共通部分の処理に関しては一切編集はできず，条件変更もできない．編集可能な部分は**図??**の通りである．

ユーザが他のバリエーションやコンパイルスイッチについて熟知している場合（ユーザレベル 2）は，使用できる機能に制限はない．

5. 適 用 実 験

5.1 車載ソフトウェアへの適用

本研究では，車載ソフトウェア[5] を対象に提案手法を適用し，その有効性を確認する実

験を行った．適用対象は，名古屋大学と愛知県立大学が共同で実施する OJL(実践的ソフトウェア工学研究) の一環で大塚らが作成した車載ソフトウェア ACC（Adaptive Cruise Control）システムを用いた．この ACC は，CarSim[6] 及び Matlab/Simlink[7] 上で動作し，コンパイルスイッチによって以下の複数のバリエーションに展開されている[⋆1]．これらは，全てソースコード内で前処理命令を用いてプログラムの分岐を行っており，適切なコンパイルスイッチの構成を与えてコンパイルを行うことで，目的のソフトウェア構成を持つプログラムを生成する．

- ACC が有効となる速度領域の違い
- ACC 制御中におけるブレーキランプ点灯の有無
- 速度制御演算方法
- 停止保持機能の修正
- 操舵角を考慮した制御機能

本研究では，この ACC に対して，あるバリエーションを抽出した結果，可読性が向上したかの抽出実験と，あるバリエーションの修正と追加作業を提案手法で行った場合，保守性が向上したかの評価を行った．

5.2 抽 出 結 果

バリエーション展開後の ACCStatusControl.c にバリエーションチェックアウトを行い，どの程度違いがあるかを比較する．抽出条件は以下の通りである．

- ユーザレベル 1
- WIDE_SPEED_RANGE
- KEEP_STOP

図 12 での左側のソースコードが抽出前，右側のソースコードが抽出後である．見て分かる通り，コンパイルスイッチのが全て消えるため，可読性が向上している．また，コンパイルスイッチの条件が変わる場合や，抽出されていないソースコードが間にある場合も，マーカーにより判別が可能である．

また getNextStatus という関数に着目すると，抽出前のコードでは 341 行であったのが，抽出後は 217 行と約 3 割ほどコード行数が減少しており，可読性が向上している．また，バリエーション固有のコードである，条件”WIDE_SPEED_RANGE&&KEEP_STOP” の部分は 62 行であり，他のバリエーションに影響を与える箇所が大部分であった．この部分を

⋆1 後者 2 つの機能は，2008 年度生の OJL の成果である

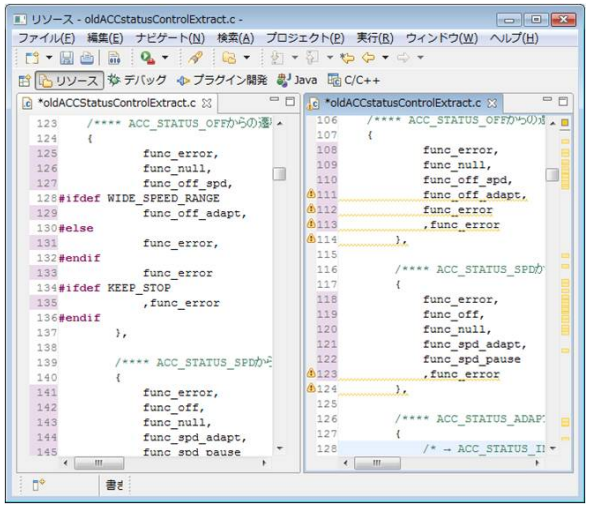

図 12 抽出前との比較
Fig. 12 Comparison with before extract

制限することにより，ユーザレベル 1 の場合であれば他のバリエーションに影響を与えることなく保守が可能となる．

5.3 停止保持機能の修正

5.3.1 バリエーションに固有のコード修正

変更箇所は以下の通りであった．

- バリエーション特有のコードの箇所: 15 箇所
- 他のバリエーションとの共通箇所: 3 箇所

ユーザレベル 1 で全ての作業は行えないが，他のバリエーションとの共通部分のコードの変更箇所は非常に少なかったため，その部分は他のバリエーションを理解しているユーザが変更することで，安全に保守が可能である．

5.4 新たなバリエーション追加作業

操舵角を考慮した制御機能追加作業では，新たにコンパイルスイッチ CONSIDER_ HANDLE_ANGLE が追加され，その条件部分のコードが追加されただけであり，元のコードに変更は見られなかった．このような場合は，提案手法ではユーザレベル 1 で作業が行えるべきである．しかし，現段階ではユーザレベル 1 では条件変更機能はなく，3.3.2 章で述べ

た新しいバリエーション追加が行えない．そこで，新しいコンパイルスイッチ追加機能が全ユーザレベルにおいて必要であるということが分かった．その場合，他のバリエーションとの共通部分を変更される心配がなく，保守性が向上する．

5.5 考　　察

5.2 章で示した通り，バリエーション抽出を行うことで，可読性が向上したソースで編集を行うことができるようになった．また，抽出時にバリエーションの理解度によってユーザレベルを設定することにより，その他のバリエーションに影響を与えないような制約をあたえることで，保守性を向上することができた．

また，ユーザのレベル分けをより細分化する必要が実験の結果判明した．提案手法では他のバリエーションについて知識のないユーザは，他のバリエーションに影響のない，抽出されたバリエーションに特有の条件の行のみしか編集できないようにして保守性を高めているが，それでは不十分であり検討する必要がある．例えば，コンパイルスイッチによる分岐が A-B の入れ子になっており， B が排他的な機能を選択するスイッチであり，選択が必須であるという状況が発生した．この場合，B の全ての機能に共通する箇所を編集しようとしても，B は選択が必須となりレベル 1 のユーザでは編集することができなくなる．

これらのことから，少なくとも，共通部分が編集できないことで，安全に保守できるものの，このように，ある部分集合については熟知しているユーザに対する権限を作成する必要があり，様々なユーザ権限が必要となると考える．

6. おわりに

6.1 ま と め

本論文では，版管理ツールへのバリエーション管理機能の拡張と、バリエーション編集のための統合開発環境について提案し、評価のための実装を行った提案手法を車載システムのサンプルとして開発した ACC のソースコードに適用した結果，可読性の高いソースコードを編集することができ，保守の際に他のバリエーションに影響が出る危険性をなくすことができた．

6.2 今後の課題

現段階ではノーマルユーザとスーパーユーザの 2 段階しか実装されていないが，ユーザによる権限の細分化が必要である．また， 現在は他のバリエーションをコンパイルスイッチの ON と OFF の組み合わせ全通りで判断しているが，実際はコンパイルスイッチの依存関係などにより少なくなる．このバリエーションを全て把握できれば，安全性は変わらずに編集できる部分が増やせると考えている．そのためには全バリエーションをユーザが定義することや，コンパイルスイッチ依存構造解析器の使用などが考えられる．

5.4 で述べた様に，新しいコンパイルスイッチを追加する機能が現在まだ実装できていない．他のバリエーションについて理解していないユーザでも，他のバリエーションに影響が出ないよう新しいコンパイルスイッチを追加する機能と共に，他のバリエーションとマージする機能が必要である．

現段階ではバリエーションごとの版管理が実現できていない．バリエーションが抽出された際の情報を元に，実現する必要がある．また，誰が，いつ，どのバリエーションを抽出したかを把握することにより，現在どのバリエーションについて変更が行われているかや，変更箇所によりどれだけのバリエーションに影響が及ぶかなど，バリエーションの並行管理をすることができると考えられる．

6.3 謝　　辞

本研究は，文部科学省研究拠点整備等補助金（先導的 IT スペシャリスト育成推進プログラム）による助成のもとで実施された OCEAN プロジェクト教育カリキュラムの OJL の一部として実施された．本研究を進めるにあたりご協力いただいたトヨタ自動車株式会社の細谷伊知郎様，西川賢司様，城戸滋之様，加藤光晴様，小嶌隆志様に，謹んで感謝の意を表する．最後に，本研究において共に研究や議論を重ねた，名古屋大学の太田創君にも感謝する．

参 考 文 献

1) H. Spencer, and G. Collyer: #ifdef Considered Harmful, or Portability Experience with C News, Proc. USENIX Summer 1992 Technical Conference, pp.185-197. 1992.
2) Stan Jarzabek,Yinxing Xue,Hongyu Zhang and Youpeng Lee: Avoiding Some Common Preprocessing Pitfalls with Feature Queries, Proc APSEC2009 (2009)
3) COLLABNet: Subversion http://subversion.tigris.org/
4) Eclipse C/C++ Development Tooling - CDT. http://www.eclipse.org/cdt/.
5) 大塚直也:車載システムを例題としたコンパイルスイッチ間の依存関係抽出. 愛知県立大学, 修士論文,2009
6) Mechanical Simulation: CarSim Overview. http://www.carsim.com/products/carsim/.
7) MATLAB：サイバネットシステム. http://www.cybernet.co.jp/matlab/.