情報処理学会論文誌　デジタルプラクティス　Vol.3　No.1　11–18 (Jan. 2022)

空調機ソフトウェアを対象としたSPLE開発における ブランチ・マージプロセスの改善と考察

長峯 基[1,a)]　徳本 修一[1]　中島 毅[2]　井上 克郎[3]

受付日 2021年7月2日, 再受付日 2021年8月20日,
採録日 2021年9月17日

概要：ソフトウェアプロダクトラインエンジニアリング（SPLE）を成功させる重要なプロセスの1つに構成管理があり，構成管理の技術領域の1つにソフトウェアのブランチ・マージがある．ソフトウェアのブランチ・マージは，複数の並行した開発において，プロジェクト間でコア資産への変更影響を受けないようにするために，開発開始時にコア資産を派生させ，開発完了後に変更をコア資産に統合することであり，コア資産の再利用性や製品開発の品質・生産性に大きな影響を及ぼす．我々は，並行開発が多い業務用空調機への5年のSPLE適用経験におけるコア資産の派生（統合されない）の原因を分析し，製品開発組織によるコア資産の固有化および未評価部品の利用回避の2つの問題によることを見出し，これらを解決するためにブランチ・マージプロセスの改善を行った．改善後のプロセスは，ブランチ時に衝突回避のための計画を立てる作業と，マージ時に部品の適合性を確認する作業からなる．この改善プロセスを実践し，マージの衝突およびコア資産ブランチ数にどのような効果があるかを改善前後で比較検証した．その結果，マージの衝突を回避しコア資産のブランチを抑制できていることを確認した．

キーワード：ソフトウェアプロダクトライン，SPLE，ブランチ・マージ，構成管理

Improvement of the Branch and Merge Process for Software Product Line Engineering in the Air-Conditioner Domain

Motoi Nagamine[1,a)]　Shuich Tokumoto[1]　Tsuyoshi Nakajima[2]　Katsuro Inoue[3]

Received: July 2, 2021, Revised: August 20, 2021,
Accepted: September 17, 2021

Abstract: Configuration management is one of the key processes for successful software product line engineering (SPLE), in which the software branch and merge process is an important technique. Software developer branches core assets at the start of development and at the end of development merges the changes to the core assets in the branches. Such a branch and merge process has a significant impact on the reusability of core assets, and the quality and productivity of the product development. This paper proposes an improved branch and merge process that fits the parallel development with the difficulties of merging. The process consists of a planning activity at the start of the development to reduce the merge conflicts and an activity at the time of merging changes in the branches to check the compatibility with other products. We have applied the configuration management processes to the projects before and after the improvement to find out that we could avoid merge conflicts and suppress the number of core asset branches.

Keywords: software product line engineering (SPLE), branch and merge process, configuration management

[1] 三菱電機（株）
MITSUBISHI ELECTRIC Co., Chiyoda, Tokyo 100–8310, Japan
[2] 芝浦工業大学
Shibaura Institute of Technology, Koto, Tokyo 135–8548, Japan
[3] 大阪大学
Osaka University, Suita, Osaka 565–0871, Japan
[a)] Nagamine.Motoi@ce.MitsubishiElectric.co.jp

1. はじめに

近年，家電製品等の組込み製品においては，他社差別化のための高機能化，多様なユーザニーズに応えるための製品バリエーションの充実化，タイムリな市場投入が強い事

© 2022 Information Processing Society of Japan

業要求となっている．

このような事業要求に対応するため，ソフトウェアプロダクトラインエンジニアリング（以下，SPLE）[1]-[3] が広く実践されている．SPLE とは，ある製品群において，共通かつ管理された機能を共有するソフトウェア・システム群を活用して体系的な再利用をする開発手法のことである [2], [4].

SPLE において，コア資産（再利用対象となる設計書やソースコード等のソフトウェア資産）の維持と発展のためには，構成管理を確実に行うことが重要である [5]．なぜならば，SPLE においては，コア資産への変更の影響は個々の製品の品質や生産性に多大な影響を及ぼすためである [6]．ここでコア資産の構成管理とは，コア資産自体のバージョン管理やコア資産の各製品への適用状況を管理することを指す [5], [7].

複数プロジェクトによる並行開発でのコア資産の維持・発展を確実に行うための構成管理の技術領域の 1 つとしてソフトウェアのブランチ・マージがある．ソフトウェアのブランチ・マージとは，複数の並行した開発において互いの影響を受けずに開発するためにコア資産を派生させること（この作業を「ブランチを切る」と呼ぶ），および開発されたソフトウェア部品（再利用対象であるソフトウェアのモジュール）の変更差分を，コア資産として統合すること（この作業を「マージする」と呼ぶ）を指す [8]．ブランチ・マージの各作業をまとめた一連の活動をブランチ・マージプロセスと呼ぶ.

ブランチ・マージにおける主要な問題として，マージの衝突がある．マージの衝突とは，並行する開発で同じソフトウェア部品に，異なる開発者が異なる変更を別々に実施し，その変更をコア資産に戻す際に，2 つの変更が衝突することを指す．マージの衝突は，多大な設計のやり直しや修正誤りを引き起こす [8].

我々は，業務用空調機（室外機）を対象に 2010 年から SPLE を実践している．業務用空調機は，製品の並行開発が多く，ソフトウェア開発も並行しやすいためマージの衝突が生じやすいという特徴がある.

適用当初は，1 つのトランク（コア資産を保管する場所）でコア資産を保守・開発し，個々のリリース時にコア資産のブランチを切り，製品固有の仕様変更や不具合修正があった場合，開発後にマージする方法を採用していた．この方法では，同時並行の開発プロジェクト間で独立してコア資産に変更を加えるため，複数の開発が並行する中でマージの衝突が起こりやすくなっていた．このため，製品開発部門において，自部門の製品開発を最適化しようとしてコア資産の固有化（製品特化）が進んでいた．また，製品部門では他製品からの変更がマージされたコア資産の品質を懸念して利用を控える事態も生じていた.

この分析結果に基づき，マージの衝突回避ならびにコア資産の信頼性向上および再利用率の向上を狙い，①アプリケーション開発開始時に開発予定のソフトウェア部品のブランチ状況を確認しマージ計画を検討すること（部品開発計画策定），②開発終了後に将来の製品適用を見据え変更したソフトウェア部品を使用する他製品系列の動作保証を行うこと（マージ試験），をプロセスに組み込む改善を行った.

本稿は，SPLE 適用経験から，複数組織による並行開発でのコア資産のブランチ・マージにおける問題点を明らかにし，その問題点を解決する改善手法を提案する．本改善手法を実開発に適用し，マージの衝突およびコア資産ブランチ数にどのような効果があるかを改善前後で比較検証することにより有効性の評価を行った．その結果，並行開発時の部品開発状況を可視化し信頼性を確保することがコア資産の派生の抑制やマージにかかるコストの抑制に効果があることが分かった.

本稿 2 章では，適用対象の特徴および複数組織による並行開発でのコア資産のブランチ・マージの問題点を述べるとともに，ブランチ・マージプロセスの改善手法を提案する．3 章で，改善前後のプロセスを定量的に比較検証することで改善手法の有効性を評価し，4 章にて本評価に対する妥当性への脅威を示す．5 章では，本稿で研究対象としたソフトウェアマージの衝突回避および SPLE の品質保証に関する関連研究を紹介し，6 章で，今回の実践によって得られた知見をまとめる.

2. ブランチ・マージの改善手法の提案

2.1 SPLE 適用対象

我々は，業務用空調機（室外機）の製品群に対して，SPLE の本格適用を 2010 年より実施してきている．この製品群は，日本国内のみならず世界各国へ出荷され，冷暖房能力や機能の搭載有無，ハードウェア構成の違いがあるため，様々な製品バリエーションを持っている．ハードウェア起因の小さな仕様の差異をいかにソフトウェアでカバーするかが開発戦略上重要であり，多品種小変更型開発として，以下の特徴を有する.

（ア）製品の機能・性能がハードウェアに大きく依存するため，製品群の仕様および開発計画は，機械や電気部門（まとめて製品部門と呼ぶ）が決定する [9]．製品部門は用途や出荷地域が異なる固有の市場ごとに組織（部や課）が分かれており，互いの仕様変更を把握していない.

（イ）製品部門からの仕様は，コア資産に基づくものでなく，以前開発した製品に基づいたものとなる．例：前年製品をベースに，ある機能を変更するなど.

（ウ）個々の仕様変更は小規模であり，短期に並行して実施される.

適用対象の製品群において，製品部門は市場ごとに組織

© 2022 Information Processing Society of Japan

が分かれているのに対し，ソフトウェア部門は，1つの部門が各製品向けにソフトウェアを開発し提供している．

2.2　改善前のプロセスとその問題点

2.1節で述べた特徴を有する業務用空調機において，2010年から5年間SPLE適用を継続した．この際，コア資産のブランチ・マージプロセスは，1つのトランクでコア資産であるソフトウェア部品を保守・開発し，個々のリリース時にコア資産のブランチを切って変更を加え，開発後にトランクにマージする方法を採用していた．このプロセスを5年間継続したことにより，コア資産のブランチ数が増加するという問題が生じた[10]．

改善前のプロセスにおけるブランチ・マージの進み方を**図1**に示す．

このプロセスの詳細手順は以下となる．

①コア資産はトランクに保管
②特定製品向けに部品をトランクで開発
③部品開発完了後，トランクからブランチを切って製品リリースブランチを作成する．製品リリースブランチは，ある製品向けの評価やチューニングを目的に，トランクで開発したソフトウェア部品を固定化するためのものである
④製品リリースブランチを使い，製品ソフトウェアを生成・リリース（リリース作業）
⑤製品ソフトウェアに不具合や仕様変更が生じた場合は，製品リリースブランチ上で部品を修正する（チューニング目的の仕様変更は頻発する）
⑥製品ソフトウェアを出荷したのちに，確定した変更差分をトランクにマージする

改善前のプロセスは，ソフトウェア部品をトランクで開発する．これは，マージの衝突がほとんど生じないことを前提としている．この当時，並行開発を実践するソフトウェア開発者は5人程度であり，緊密なコミュニケーションやスケジュール調整によって同一トランクでソフトウェア部品を保守・開発することが可能であると考えていた．

しかし実際には，製品の並行開発の多さゆえに，同一のソフトウェア部品に同時に異なる仕様変更が発生することが多かった．この場合，並行に行われた他製品向けの部品開発との衝突を避けるため，ある製品のために切ったブランチ（製品リリースブランチ）がトランクにマージされず，継続的に製品固有のコア資産として維持される問題が生じた（**図2**）．この派生したコア資産を製品メインブランチと呼ぶ．製品メインブランチをベースに開発が進んでしまうことで，製品群全体の再利用性が低下する，変更が累積しトランクに戻すためのマージコストが生じるという問題がある．

製品メインブランチが発生する原因は，以下である．

問題①：製品部門によるコア資産の固有化（製品特化）

多品種小変更型開発においては，同一ソフトウェア部品に対する開発の衝突が生じやすい．さらに，特徴（ア）および特徴（イ）から，製品部門は，自部門の製品開発を最適化するため，コア資産を製品部門固有にする，つまり製品メインブランチを積極的に作ることで他の製品開発の影響を最小化しようとする．製品メインブランチ上で開発を継続することで，変更が積み上がっていき，ますますマージが困難になる．

問題②：未評価部品の利用回避

製品部門は用途や市場ごとに異なる開発組織が担当する．ある製品開発は，ベースとなる以前開発した製品に基づき実施される．その際，以下が問題となる．

- ベース製品は，その製品系列における評価済のコア資産，すなわち製品リリースブランチを選択するのがベストである．対してトランクには，その製品系列における前回開発での後，他の製品系列の部品開発による変更もマージされており，他の製品系列の開発で変更された部品は，当該製品系列では未評価である．

この問題のため，製品部門は，コア資産を自部門で開発している製品固有のものとしたがる（製品メインブランチを積極的に作りたがる）．

問題①および②を解決するためには，他製品からの部品変更の影響を受けずに開発しつつ，コア資産をマージする仕組み，およびマージしたコア資産が他製品系列に組み込んでも動作することを保証する仕組みを構築することが課題である．

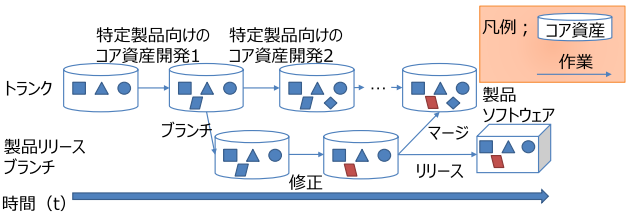

図1　改善前のプロセスにおけるブランチ・マージの進み方

Fig. 1　Progress of branch and merge in the pre-improvement process.

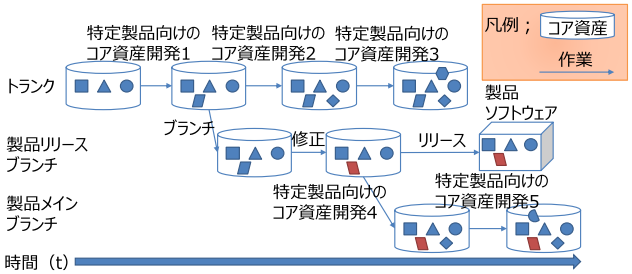

図2　改善前のプロセスにおける製品メインブランチ

Fig. 2　The main branch of a product in the pre-improvement process.

© 2022 Information Processing Society of Japan

2.3　改善プロセスの提案

2.3.1　改善後のプロセスの概要

前節で述べた課題を解決するためにブランチ・マージプロセスの改善を行った．改善後のプロセスにおけるブランチ・マージの進み方を**図 3** に示す．

改善後のブランチ・マージプロセスは，製品開発ごとに製品開発ブランチと呼ぶブランチを切って保守・開発し，製品開発完了後に開発したソフトウェア部品をコア資産にマージする方式を採用している（問題①への対応）．さらに，ソフトウェア部品をトランクからブランチする際に「部品開発計画策定」を実施する（問題①への対応）点と，開発したソフトウェア部品をマージする際に，「マージ試験」を実施する（問題②への対応）点を改善前のプロセスに追加している．このプロセス変更によって，マージの衝突を回避することと，開発したソフトウェア部品が他製品でも利用できることを狙っている．以下に改善プロセスを詳述する．

2.3.2　製品開発ブランチでの部品開発

改善前のプロセスでは，すべてのソフトウェア部品を 1 つのトランク上で保守・開発するプロセスを採用していたため，複数の製品の開発が重なると製品メインブランチが作成されやすく変更が累積しやすい（マージがされにくい）という問題があった．この問題を解決するため，各製品開発のプロジェクト開始時にトランクからブランチを切り，そのブランチ上で当該製品対応のソフトウェア部品の変更開発を行い，開発後にトランクへのマージを行うプロセスとした．

2.3.3　部品開発計画策定

上記のプロセス変更は，複数の製品の開発が重なっても部品の開発は可能だが，マージ時の衝突という課題は残っている．そこで，「部品開発計画策定」というプロセスを設けることとした．

部品開発計画策定とは，当該製品で変更するソフトウェア部品に対し，他製品での開発計画の有無や開発内容を確認し，当該の製品開発ブランチにおける各ソフトウェア部品のマージ種別を定めるものである．

ここで，マージ種別は，更新（同じソフトウェア部品としてマージ），派生（異なるソフトウェア部品としてマージ），廃棄（当該製品専用の部品としてマージしない）の 3 つから選択する．各種別の選択基準は以下のとおりである；

- 更新：他製品での展開予定があり，衝突がない
- 派生：他製品での展開予定があるが，衝突している
- 廃棄：他製品での展開予定がない（当該製品固有）

更新を選んだソフトウェア部品はトランク上の開発元部品に単純マージ，派生を選んだソフトウェア部品はトランクに類似部品として新規登録，廃棄を選んだソフトウェア部品は製品開発ブランチに残し，コア資産とはしない．マージ方式で派生を選択した場合に，類似したソフトウェア部品が増えるという問題はある．しかし，本製品のプロダクトラインアーキテクチャがソフトウェア部品を選択したものをビルド・リンクする方式を採用しているため [10]，類似部品が数個程度であれば部品選択の間違いは生じにくい．

部品開発計画策定では，製品開発ブランチごとに，**表 1** に示すブランチ帳票を作成する．ブランチ帳票は，縦軸にトランクに登録されている全ソフトウェア部品，横軸にソフトウェア部品ごとの版，前版からの変更点，ブランチ時点での他製品での利用状況（開発計画）を示した表である．本情報をもとに各ソフトウェア部品の当該製品での利用有無とマージ方式を定める．たとえば，部品 A は他製品での利用状況はない（表中で”-”）ため，当該製品での開発内容をそのままマージする（更新とする），部品 C は製品 1 で開発中であり変更内容も異なるため，派生部品としてマージ（新規登録）する，ということをブランチ時に計画する．

ソフトウェア部品の開発状況を他の開発者に周知することは，マージの衝突回避に有効である．本改善プロセスでは，上記のブランチ帳票をこの目的に使用している．具体的には，開発中の各製品のブランチ帳票をもとにソフトウェア部品ごとの開発状況を把握し，部品開発計画策定時にブランチ帳票の「他製品での利用状況」欄を自動更新する機能を実現している．同様の作業を手動で行う場合，各製品開発ブランチを調査し，開発実績および開発予定のあるソフトウェア部品を確認することとなるが，並行開発が多いためブランチ数も多くなり調査のための時間を要する．また，変更要求の確定が遅いため部品のマージ計画を

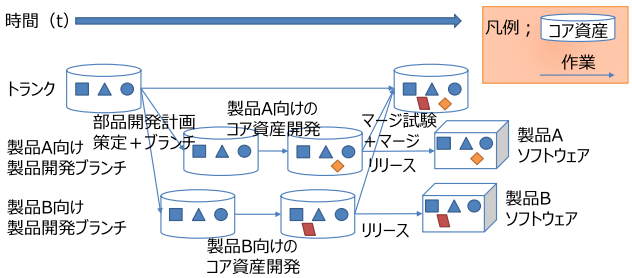

図 3　改善後のプロセスにおけるブランチ・マージの進み方

Fig. 3　Progress of branch and merge in the pre-improvement process.

表 1　ブランチマージ帳票の例

Table 1　Example of the branch and merge form.

部品名称	版	前版からの変更点	ブランチ時			マージ時	
			他製品での利用状況	利用有無	開発形態	開発実績	マージ可否
部品A	1.00	-	-	有	更新		
部品B-1	1.01	***	製品1	無	-		
部品B-2	2.00	部品B-1から派生	製品2	有	更新		
部品C	1.01	***	製品1	有	派生		

© 2022 Information Processing Society of Japan

見直すことも多いため，この作業を手作業で実行するのは現実的ではない．本作業を自動化することで，手動時と比べ調査および周知にかかる工数を 97.9% 削減と試算している．

このように製品の開発開始時に製品開発ブランチを切り，ブランチ帳票を用いて部品開発計画を策定することで，①他製品の影響をあらかじめ考慮してソフトウェア部品を開発する，②他製品開発でのソフトウェア部品の開発状況を把握し開発完了後にできる限り衝突させずにマージする，の 2 点を可能とする．

2.3.4　マージ試験

多品種小変更型開発においては，問題②のとおり，開発したソフトウェア部品を他製品に組み込んだ場合の動作保証をする必要があった．

そこで，開発したソフトウェア部品をマージする際に将来的に展開予定のある他製品への適合性を評価する「マージ試験」を設けている．**図 4** はそのための具体的な環境である．

マージ試験では，トランクに保管されているあらかじめ準備された製品の基本試験仕様書と，製品開発ブランチにて開発した当該ソフトウェア部品用の試験仕様書に示された 2 種の試験を実施する．試験対象となるソフトウェアは，当該製品で開発した仕様を展開する可能性のある製品系列の最新版ソフトウェアとする．試験対象となる製品系列の選定は，ロードマップ（仕様の展開計画）や元となったソフトウェア部品の適用製品をもとに判断する．全製品系列への展開を想定するソフトウェア部品の場合，最大で 10 程度の製品で試験を実施する．たとえば，製品 A で開発したソフトウェア部品 P を製品 B に将来展開する見込みがある場合は，製品 B の部品構成に，開発した部品を組み込んで先行的に試験を実施する．

マージ試験を製品の基本動作と当該ソフトウェア部品用の試験の 2 種を実施することで，他の製品部門に対し，開発したソフトウェア部品 P を組み込んでも製品が正常に動作し当該ソフトウェア部品が期待どおりの機能を果たすことを保証している．これらの試験は事前に作成された試験仕様書に基づく自動実行を原則としているため，動作の保証にはほとんど人手を要しない．

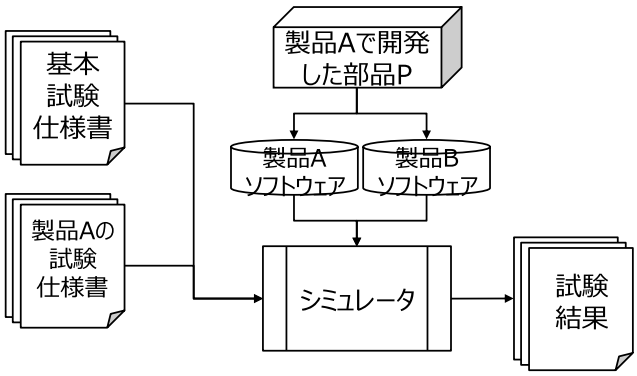

図 4　マージ試験環境
Fig. 4　Merge test environment.

マージ試験完了後に，対象となるソフトウェア部品のマージを実施する．マージに際しては，①マージしようとするソフトウェア部品が当初の部品開発計画に従い開発されているか，②他の製品開発ブランチにおいて，部品開発計画時に確認したとおりにソフトウェア部品が開発されているか（マージの衝突が発生しうる割り込み開発がないか）を確認し，マージの衝突がない場合に速やかにリポジトリにマージする．マージの衝突が生じた場合は，再設計のうえマージするか，別のソフトウェア部品として派生させて登録するかを検討し個別に対応する．

なお，本プロセスの導入後もソフトウェア部品の適合性に起因する不具合は 0 件を維持できており，マージを積極的に採り入れることによる品質への悪影響が生じていない．

3.　改善手法の評価

2.2 節で述べた改善前のブランチ・マージプロセスを 2010 年から適用し，2015 年から 2.3 節で改善したプロセスを導入している．改善手法適用開始直前の 2014 年度の各指数を基準として，各指標の推移を示し，適用の効果を示す．各年度のプロットは，各年度の開発で得られたデータの平均値とした．

3.1　マージコスト比率 Rc

問題①に示すように，多品種小変更型開発においては，製品メインブランチ上で開発が継続しやすく，本来コア資産とすべき変更がマージされず累積するという問題がある．本改善手法の構成管理プロセスとしての有効性を評価するため，改善前後の開発費に占めるコア資産のマージコストの割合に基づいて，マージ衝突を回避できているかという観点から評価する．ただし，改善前のプロセスでは，コア資産は先述のとおり直接的にはマージされず，製品メインブランチとして変更が累積していたため，これらがマージされたとした場合のコストを仮想的にマージコストとする．仮想的なマージコストの算出は，製品メインブランチ上にあるマージ対象（開発形態を「更新」としたソフトウェア部品のうち衝突しているもの）の変更量と，変更量あたりのマージコストの平均値（実績値）から算出する．本マージコストにはマージ試験にかかるコストは含まない*．マージコスト比率は，次式とする．

$$\mathrm{Rc} = \mathrm{A}/\mathrm{B} \tag{1}$$

A：コア資産マージコスト

* ここでのコストは人の工数のみを考えており，マージ試験は自動実行のため人の工数は 0 としている．

© 2022 Information Processing Society of Japan

B：当年度のソフトウェア開発費

コア資産マージコスト A は改善前後のプロセスにおいて以下のとおりである．

改善前：ブランチ・マージプロセスに実際にかけたコスト＋仮想的なマージコスト

改善後：ブランチ・マージプロセスに実際にかけたコスト（マージ試験含む）

なお Rc を年度ごとに比較するため，2014 年を 1.00 とした相対値 Rc-yy で示す．Rc-yy は次式と定義する．

$$\text{Rc-yy} = \text{yy 年の Rc}/\text{2014 年の Rc} \quad (2)$$

Rc-yy の推移を**図 5** に示す．

当初プロセスにおいては，毎年一定の割合で，並行開発間での衝突が生じ，トランクへのマージが行われなかったためコア資産にマージすべき変更が累積していたが，改善プロセス導入後は，15 年に 28% 減，16 年に 61% 減となっている．これは，当初プロセスにおいては，ソフトウェア開発者の緊密なコミュニケーションや日程調整だけでは開発の衝突を回避できずマージすべき内容が累積してしまったのに対し，改善プロセス導入後は，開発の衝突を防止することができているといえる．また，マージすべきソフトウェア部品は当年度中にマージできており，本改善手法が有効であると評価できる．

3.2　製品メインブランチ数 Nb

プロセスの改善を行った結果，コア資産が派生しなかったことを確認する．改善前のプロセスにおける派生したコア資産は製品メインブランチであり，改善後のプロセスにおける派生したコア資産はマージされなかった場合の製品開発ブランチが製品メインブランチとなる．このことから改善前後の製品メインブランチ数 Nb が増加しなかったことを確認することで，コア資産が 1 つ（トランクのみ）に維持できたことを検証する．

製品メインブランチ数の推移を**図 6** に示す．

当初プロセスにおいては，年を追うごとにブランチ数が増加し，2014 年に 3 本の製品メインブランチが存在していた．これに対し，プロセス改善以降は，製品メインブランチ（マージされなかった製品開発ブランチ）は存在しておらず 2 年後も 0 本のままであり，目論見どおりコア資産の派生を防ぐことができている．これは，各製品部門が製品メインブランチを囲い込み，固有のコア資産を形成する必要がなくなったことを意味している．

このようにコア資産の派生を 0 で維持できた要因は，3.1 節でも議論したとおり，マージ計画を立てることでマージの衝突が回避しやすくなったことに加え，他製品で開発したソフトウェア部品の品質が保証されているためコア資産に統合しやすくなったことが主要因と考える．

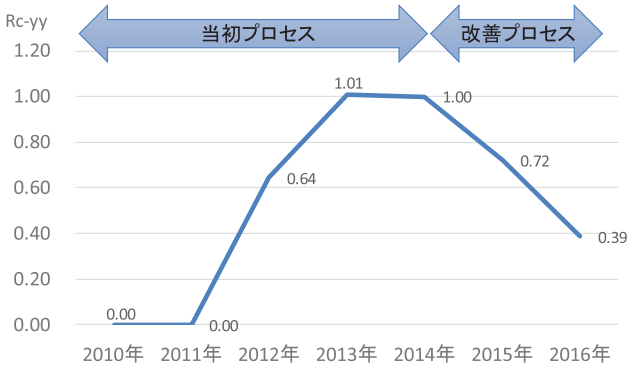

図 5　マージコスト比率

Fig. 5　Merge cost ratio.

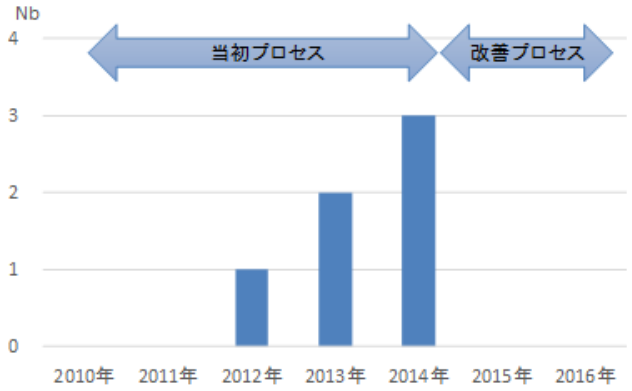

図 6　製品メインブランチ数

Fig. 6　The number of the main branches of products.

4.　妥当性への脅威

本稿で実践した他開発の状況を可視化してブランチ・マージする手法は，コア資産開発の計画性の高さや（計画どおり開発できること，計画外の開発が発生しないこと）が重要となる．本稿の対象である空調機は，他社動向を見ながら仕様変更する特性を持ちあわせているため，計画外の開発が生じやすく，マージ時に想定外の衝突が発生しているケースも散見された．このような状況を解決するためには，Fine-grained revision control [11] のような細やかにブランチ・マージを繰り返す手法も有効と考える．本適用対象では，製品部門が製品ごとにソフトウェアを管理したいという要望があるため製品ごとのブランチ・マージという手法を採用したが，マージの衝突が一定数残っているため，先述の Fine-grained revision control と組み合わせた手法も検討の余地がある．

また，マージ試験では，開発したソフトウェア部品を将来どの製品系列に組み込むかを想定して，事前に適合性を評価している．製品への組み込みが予想できない，もしくは製品系列が多すぎて評価が困難な場合に本手法の適用が難しいという事態も想定されるが，本適用対象では，製品系列は 10 程度と大きく増加しておらず，製品開発のたびにビルド対象の製品系列が増えるといった問題は発生して

© 2022 Information Processing Society of Japan

いない．

5. 関連研究

5.1 ソフトウェアマージの衝突回避

ソフトウェアマージの衝突については，衝突の検知・回避 [5], [12]，マージ技術およびツール [13], [14] の各分野で様々な研究がなされており，Mens らが各手法を整理している [8].

衝突の検知に，グラフ分割という手法がある [8]．グラフ分割とは，ソフトウェア構成要素（クラスやソフトウェア部品）間の依存関係をグラフによって可視化する手法である．衝突する可能性のある構成要素を見つけ出すことに優れるが，頻繁に依存関係が変更される場合のグラフ更新に負荷が生じる．また，衝突の回避の技術にカプセル化がある [8]．カプセル化とは変数やメソッドを隠蔽することで，変更の伝播を局所化しマージ時の衝突を減らすものである．メソッド等の境界を越えた変更の場合，効果は限定的である．

このほかにも衝突を回避する手法に Fine-grained revision control がある [11]．Fine-grained revision control は，一回あたりの変更量をごく小さくし，小さな開発を積み上げていくことで，マージの衝突を回避する．開発規模の大きな変更が並行した場合に，マージが衝突しやすく多大なコストがかかる傾向にある．

Adams らはワークステーションの開発において，Fine-grained revision control をベースとした手法を用い，短期間に小さな開発を実施しマージすることの有効性を述べている [11]．機能ごとにブランチを設けて開発を進めマージをする手法は，Git-flow や Github-flow でも用いられている [15], [16]．これらの手法は，コア資産のブランチを製品ごとに設ける点において我々の手法と異なっている．

マージ時の衝突を回避する手法の 1 つにお互いの変更を周知するという方法がある．Brun らは，マージの衝突を特定・管理・回避することを支援する Crystal というツールを提案し，複数のエンジニアが協調的に開発する場合のコストや難易度を下げる実践的な取り組みであると評価している [17]．本稿での取り組みは，変更を周知することで衝突を回避する点において Brun らの取組みと類似しているが，製品開発ごとにコア資産をブランチし，ブランチ時点での情報をもとにマージ計画を立てる点が異なっている．

5.2 SPLE の品質保証

SPLE のアプリケーション開発における品質保証はテストを中心とした事例が数多く報告されている [18].

SPLE におけるテスト手法の 1 つに，ドメイン開発やアプリケーション開発にて開発したテストケースを再利用する手法がある．この手法は，ドメイン開発や他アプリケーション開発で開発したテストケースを固有のアプリケーション開発に適合させて再利用するもので，多くの事例がある [19].

いずれの手法も，将来の部品適用を想定して先行的に試験している点が我々の取組みと異なっている．

6. おわりに

本稿では，並行開発が多い業務用空調機への 5 年の SPLE 適用経験におけるコア資産の派生（統合されない）の原因を分析し，製品部門によるコア資産の固有化および未評価部品の利用回避の 2 つの問題によることを見出し，これらを解決するためにブランチ・マージプロセスの改善を行った．

改善プロセスは，ブランチ時に衝突回避のための計画を立てる作業と，マージ時に部品の適合性を確認する作業からなる．この改善プロセスを適用した結果，マージの衝突を回避することに成功し，マージコスト比率を 61% 削減，コア資産ブランチ（製品メインブランチ）も 0 本を維持できている．この結果，ソフトウェア部品再利用率の向上も実現できた [20].

これらの結果に基づき，構成管理プロセスは，製品開発組織および製品開発プロセスとの親和性があるブランチ・マージプロセスを構築することが重要であるとの知見を得た．

参考文献

[1] 吉村健太郎，ダルマリンガム ガネサン，ディルク ムーティック：プロダクトライン導入に向けたレガシーソフトウェアの共通性・可変性分析法，情報処理学会論文誌，Vol.48, No.8, pp.2482–2491 (2007).

[2] Clements, P. and Northrop, L.: Software Product Lines: Practices and Patterns and Addison-Wesley (2002).

[3] Van der Linden, F. J., Schmid, K. and Rommes, E.: Software product lines in action: the best industrial practice in product line engineering, Springer (2007).

[4] Deelstra, S., Sinnema, M. and Bosch, J.: Product derivation in software product families: a case study, *Journal of Systems and Software*. Vol.74, No.2, pp.173–194 (2005).

[5] Tichy, W. F.: Tools for software configuration management, *Proc. of the International Workshop on Software Version and Configuration Control*, pp.1–20 (1988).

[6] Munson, J. P. and Dewan, P.: A flexible object merging framework, *Proc. of the 1994 ACM conference on Computer supported cooperative work, ACM*, pp.231–242 (1994).

[7] Pohl, K., Bockle, G. and van der Linden, F.: ソフトウェアプロダクトラインエンジニアリング，株式会社エスアイビー・アクセス (2009).

[8] Mens, T.: A state-of-the-art survey on software merging, *IEEE Trans. Softw. Eng.*, Vol.28, No.5, pp.449–462 (2002).

[9] 西　康晴：製品品質の決定要因としての組込みソフトウェアと組込みソフトウェア・クライシス，品質，日本品質管理学会誌，Vol.34, No.4, pp.343–349 (2004).

[10] Nagamine, M., Nakajima, T. and Kuno, N.: A Case

© 2022 Information Processing Society of Japan

Study of Applying Software Product Line Engineering to the Air Conditioner Domain, the 20th International Systems and Software Product Line Conference, pp.220–226, ACM (2016).
[11] Adams, E., Gramlich, W., Muchnick, S. S. and Tirfing, S.: SunPro engineering a practical program development environment. In Advanced Programming Environments, pp.86–96, Springer (1987).
[12] Feather, M. S.: Detecting interference when merging specification evolutions, *Proc. of the 5th international workshop on Software specification and design*, pp.169–176 (1989).
[13] Leßenich, O., Apel, S. and Lengauer, C.: Balancing precision and performance in structured merge, *Automated Software Engineering*, Vol.22, No.3, pp.367–397 (2015).
[14] Menezes, G. G. L., Murta, L. G. P., Barros, M. O. and Van Der Hoek, A.: On the nature of merge conflicts: A study of 2,731 open source Java projects hosted by GitHub, *IEEE Trans. Softw. Eng.*, (2018).
[15] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M. and Damian, D.: The promises and perils of mining GitHub, *Proc. of the 11th working conference on mining software repositories*, pp.92–101 (2014).
[16] Dwaraki, A., Seetharaman, S., Natarajan, S. and Wolf, T.: GitFlow: Flow revision management for software-defined networks, *Proc. of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pp.1–6 (2015).
[17] Brun, Y., Holmes, R., Ernst, M. D. and Notkin, D.: Proactive detection of collaboration conflicts, *Proc. of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp.168–178 (2011).
[18] Metzger, A. and Pohl, K.: Software product line engineering and variability management: achievements and challenges, *In Future of Software Engineering Proceedings*, pp.70–84 (2014).
[19] Engström, E. and Runeson, P.: Software product line testing-a systematic mapping study, *Information and Software Technology*, Vol.53, No.1, pp.2–13 (2011).
[20] 長峯　基，中島　毅：多品種小変更型開発におけるコア資産保守・製品導出手法の改善と実践，デジタルプラクティス，Vol.10, No.3, pp.639–655 (2019).

長峯 基（非会員）

2006 年筑波大学第三学群工学システム学類卒業．同年三菱電機（株）入社．現在，同社静岡製作所に勤務．

徳本 修一（正会員）

1995 年北海道大学大学院工学研究科精密工学専攻修士課程修了．同年三菱電機（株）入社．現在，同社情報技術総合研究所に勤務．情報処理学会，電子情報通信学会各会員．

中島 毅（正会員）

1984 年早稲田大学大学院修士課程修了．同年三菱電機（株）入社．2008 年早稲田大学大学院博士課程修了，博士（工学）．現在，芝浦工業大学情報工学科教授，ソフトウェア工学に関する研究に従事．著書に『IT Text ソフトウェア開発改訂第 2 版』（共著，オーム社）．技術士（情報工学/総合技術監理）．IEEE CS，電子情報通信学会，電気学会各会員．

井上 克郎（フェロー）

1984 年大阪大学大学院基礎工学研究科博士後期課程修了（工学博士）．同年同大学基礎工学部情報工学科助手．1984 年〜1986 年，ハワイ大学マノア校コンピュータサイエンス学科助教授．1991 年大阪大学基礎工学部助教授．1995 年同学部教授．2002 年より大阪大学大学院情報科学研究科教授．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．

© 2022 Information Processing Society of Japan