情報処理学会論文誌 Vol.59 No.4 1175–1191 (Apr. 2018)

コンカレントフィードバック開発方法の自動車ソフトウェア開発への適用

林 健吾[1,a)] 青山 幹雄[2] 古畑 慶次[3]

受付日 2017年8月2日, 採録日 2018年1月15日

概要：本稿では，不確定要素の多いイノベーティブな新ソフトウェア製品開発において，プロトタイプに迅速にフィードバックを繰り返すことにより進化させるコンカレントフィードバック開発方法を提案する．著者らが従事している自動車ソフトウェア開発では，自動走行などに関わるまったく新たなシステムを，研究開発部門と共同で要素技術を開発しながら，かつ，短期間で高品質なシステム化を行うことが求められている．このため，実車両上にプロトタイプを構築する要求開発と，仕様を基にしたソフトウェア製品開発の 2 フェーズの開発方法を採用していた．しかし，この開発モデルでは，開発期間短縮と品質保証に問題があった．これに対して，要求開発におけるプロトタイプに迅速にフィードバックを繰り返して進化させる進化型プロトタイプの新たなモデルとして，コンカレントフィードバック開発方法を提案する．さらに，製品開発におけるコンカレントエンジニアリングの協調のアプローチを発展させ，協働してソフトウェア製品を進化させる 3 パターンのフィードバックループで構成するコンカレントフィードバックプロセスモデルを提案する．提案方法を，超音波センサを用いた自動車ソフトウェア開発に適用し，開発期間を短縮して外部品質を改善した結果から提案方法の有効性を示す．

キーワード：コンカレント開発，進化型プロトタイピング，自動車ソフトウェア，アジャイル開発，フィードバック制御

A Concurrent Feedback Development Method and Its Application to Automotive Software Development

Kengo Hayashi[1,a)] Mikio Aoyama[2] Keiji Kobata[3]

Received: August 2, 2017, Accepted: January 15, 2018

Abstract: This paper proposes an approach to concurrent feedback development method for innovative software products under uncertainty. In automotive software development, innovative new software products need to develop with high-quality while the development team works together with the research team to create new technologies and components. However, it has become more difficult to ensure quality. At the same time, automobile manufacturers demand to shorten the development time. Conventionally, we developed the software in the two phases of “prototype development for developing requirements specifications” and “product software development based on the requirements specifications”. In this article, we propose the concurrent feedback development method using the prototype in the specification development. The proposed concurrent feedback loop model consists of three patterns of feedback loop to evolve the product concurrently. We demonstrated a reduction of the development time and improved the internal quality of the product from the application to our automotive system development.

Keywords: concurrent development, evolutionary prototyping, automotive software, agile development, feedback control

[1] 株式会社デンソー
DENSO CORPORATION, Kariya, Aichi 448–8661, Japan
[2] 南山大学
Nanzan University, Nagoya, Aichi 466–8673, Japan
[3] 株式会社デンソー技研センター
DENSO E & TS TRAINING CENTER CORPORATION, Ohbu, Aichi 474–0011, Japan
[a)] kengo_hayashi@denso.co.jp

1. はじめに

情報システム開発では，高品質な要求を満たし，かつ，短期間に開発することが求められている．さらに，イノベーティブな新規システムの開発では，不確実性の高い要求をプロトタイピングにより具現化し，それを成長させる進化型プロトタイピングが実践されている [6]．たとえば，自動車ソフトウェアシステムの開発では，自動走行などの従来とは異なった高度なシステムを高い安全性などの品質を満たして開発することが求められている [10], [24]．

自動車システム開発の分野において，筆者らは，車両周囲の環境をセンシングして車両制御する自動車システムを開発している．このシステムは，ユーザの実使用環境の影響を受けやすく，製品の要素技術もいまだ確立していない．そこで，2 つの部門で要求開発と製品開発のフェーズを分担して開発している（**図 1**）．要求開発フェーズでは，研究開発部門が車両メーカと連携しながら実車両上にシステムのプロトタイプを構築する．要素技術と要求仕様，制御仕様を開発して利用品質と外部品質 [12] を明らかにする．製品開発フェーズでは，製品開発部門がソフトウェア製品を開発する．要求仕様書と制御仕様書に基づき，V 字型モデルを適用した開発で製品の外部品質を確保する．

近年，システムの高度化，複雑化が進む一方，高品質かつ短期間での開発が求められている [10]．その結果，製品開発フェーズの開発期間の短縮が求められた．この要求に対応すべく，使い捨て型プロトタイピングに基づいた 2 フェーズ開発方法から進化型プロトタイピングへの転換が必要となった [9], [11], [14]．

しかし，開発スコープと必要な技術の違いから，要求開発フェーズでは製品に求められるすべての外部品質を確保することは難しい．たとえば，要求開発フェーズでは，保守性の指標である複雑度は開発規模に比例して大きくなり，品質の低下が見られた [11]．製品開発フェーズでは開発期間の制約から，品質確保の期間が十分にとれない可能性がある．外部品質の保守性や性能効率が低下すると，製品展開時の派生開発において品質リスクや納期リスクを生じ，製品展開を妨げる．

我々はコンカレントエンジニアリングのアプローチにより，要求開発のプロトタイプを利用して製品開発フェーズを早期に開始する方法を提案し，開発期間の短縮を果たした [11]．本稿では，さらに外部品質の確保を狙って，コンカレントエンジニアリングのフィードバック機構に着目したコンカレントフィードバック開発方法を提案する．本方法では，要求開発フェーズから製品開発部門がプロトタイプ開発に参画する．そして，3 パターンのフィードバックループで高品質を保証しプロトタイプを製品に進化させる．本開発方法を超音波センサシステムの新製品開発に適用した．開発期間短縮効果と外部品質確保効果により提案方法の有効性を示す．

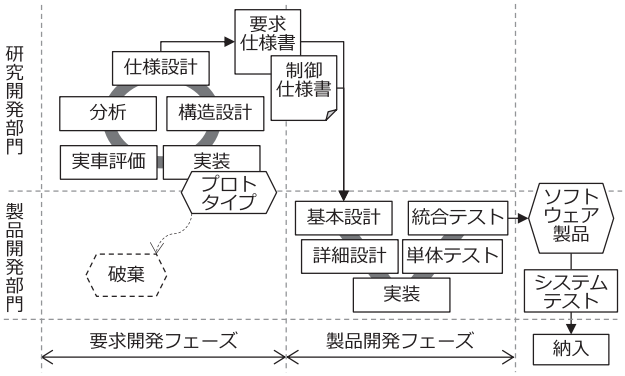


図 1 従来の 2 フェーズ開発方法
Fig. 1 Conventional 2-phase development method.

2. 関連研究

2.1 モデルベース開発

モデルベース開発は，構築したモデル上で検証を自動化し，検証済みのコードを自動生成することで開発期間を短縮する [22], [29]．しかし，精度の良いモデルを得るためには，開発対象のドメイン技術の確立が前提となる．

2.2 進化型プロトタイピング

使い捨て型プロトタイピングに対して，継続的に改良して製品化する進化型プロトタイピングは，開発期間の短縮に有効である [9], [14]．しかし，プロトタイプを顧客からのフィードバックを得ることを主目的とした場合，顧客向けの機能構築が優先され，ソフトウェアアーキテクチャの劣化が起こりうる [25]．その結果，外部品質の確保が課題となる．

2.3 スパイラルモデル/アジャイル開発

ソフトウェアプロセスモデルとして，使い捨て型プロトタイプを適用することで，顧客から要求を引き出し，未確立の技術を検証しながら，後戻りリスクを軽減してソフトウェアを開発するスパイラルモデルが提案されている [4]．

また，短期間にフィードバックを繰り返す開発モデルとしてアジャイル開発が提案されている [16]．代表的なアジャイル開発として XP [3]，Scrum [23] が知られている．アジャイル開発では，製品をインクリメンタルに開発することで，安定しない顧客の要求を短期間で追従して製品リリースする [3], [16], [23]．

いずれの開発方法でも，アーキテクチャ設計の期間を計画的に確保し，繰返しごとの品質スコープを適切に定めることで，品質確保と短期開発が両立可能である [5]．

しかし，現在の自動車システムのソフトウェア開発では，要素技術を開発してプロトタイプを構築する技術者が所属する組織と，外部品質を確保する技術者が所属する組織は

異なっている．組織構造が適合しないことから，これらの開発方法を導入することが難しい．

2.4 コンカレントエンジニアリング（CE）

CE は開発期間を短縮するために工程間や部門間の技術差を統合して製品を開発するアプローチである [7], [13]．このアプローチはソフトウェア製品開発にも適用され，その有効性が示されている [1]．近年，自動車業界におけるシステム開発においても，CE を導入することで，開発期間と開発コストを低減する提案がある [15]．この提案では，異なる部門間の共同作業を通して，研究開発時から仕様，設計，試作が製品開発に最適化するようフィードバック機構を設けて修正を繰り返す．

しかし，顧客と協調して要求仕様を開発する場合，素早いフィードバックが最も優先される．要求開発を遅延させないフィードバック機構が必要である．

2.5 フィードバックモデル

フィードバック機構のモデルとして，プロセスの継続的改善を目的とした，Plan（計画），Do（実行），Check（評価），Act（改善）の PDCA サイクルが広く知られている [26]．評価における学びと知識の共有を重視して Check を Study と置き換えた PDSA サイクルも提案されている [26]．

また，顧客要求への素早い学習を目的とした，Build（構築），Measure（計測），Learn（学習）の BML ループ [20] が提案されているほか，Observe（観察），Orient（情勢判断），Decide（意思決定），Act（行動）の OODA サイクルといった意思決定プロセスも提案されている [18]．

いずれのモデルも，共通の活動を前提とした組織を対象としている．開発スコープが異なる組織が共同作業する場合は，異なる作業を調整する仕組みが必要である．

3. アプローチ：多重フィードバックの統合

3.1 研究課題とアプローチの概要

本稿では，自動車ソフトウェア開発に CE を適用して開発期間を短縮したうえで，顧客と協調した要求開発に遅延を発生させず，展開製品開発のための外部品質を確保する開発方法を確立することを研究課題とする．

本稿のアプローチは，自動車ソフトウェア開発でプロトタイプをプロダクトに進化させるために，3 パターンのフィードバックが必要であることに着目する．

本章では，3 パターンのフィードバックが必要である理由と，フィードバックの多重化で生じる問題を示し，フィードバック機構を統合するためのフィードバックモデルの開発が必要であることを示す．

3.2 フィードバックの 3 パターンへの分類

自動車製品の開発では，プロトタイプ開発に基づき，プロトタイプを段階的に進化させるインクリメンタル開発，シリーズ開発へ発展させる 3 つのフェーズからなる開発形態をとることで，開発費用の投資を回収する戦略を採用している [27]．そのため，自動車ソフトウェア開発で進化型プロトタイピングを採用する場合，これら 3 つのフェーズに対応すべく，プロトタイプに与えるフィードバックを 3 つのパターンに分類できると考えた．これらのフィードバックは，商品性と安定性，保守性の 3 つの品質をソフトウェア製品に与える．各品質を作り込める技術者は異なっている（**図 2**）．

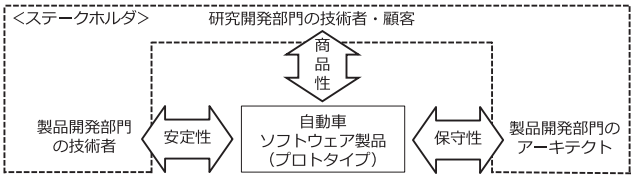


図 2 ステークホルダを起点とする 3 パターンのフィードバック
Fig. 2 Feedback of three patterns from stakeholders.

(1) 商品性

商品性は，製品の価値を向上させ製品を売るための品質である．自動車ソフトウェア製品では様々な外部環境において利用者に安全で快適な製品を提供したい [10]．

ソフトウェアの品質モデル [12] における，利用品質の有効性（Effectiveness），満足性（Satisfaction），リスク回避性（Freedom from risk），利用状況網羅性（Context coverage）に加えて，外部品質の使用性（Usability），機能適合性（Functional Suitability）の副特性である機能完全性（Functional Completeness）や機能妥当性（Functional appropriateness），信頼性（Reliability）の副特性である障害許容性（Fault tolerance）が対象となる．

これらの品質特性は，製品の機能と機能開発を支える要素技術の開発によって実現される．要求開発において，顧客と協調した研究開発部門の技術者がプロトタイプにフィードバックすることができる．

(2) 安定性

安定性は，製品が安定稼働し販売可能であることを保証する品質である．自動車ソフトウェア製品として不定な動作や欠陥が残留していることは許容されない [27]．

ソフトウェアの品質モデルにおける，外部品質の機能適合性の副特性である機能正確性（Functional Correctness）や信頼性の副特性である成熟性（Maturity），セキュリティ（Security）が対象となる．

これらの品質特性を，製品開発部門の技術者が，要求仕様や制御仕様における機能要求を分析した設計，テスト，検証作業を通してプロトタイプにフィードバックすることができる．

(3) 保守性

保守性は，製品を継続して開発できる状態を保つための品質である．自動車ソフトウェア製品は利用するハード

ウェアの世代交代が少なく，ソフトウェアで自動車プラットフォームの変更や機能拡張に対応する必要がある．機能拡張の余地があり，改造しやすいソフトウェアを構築したい．

ソフトウェアの品質モデルにおける，外部品質の性能効率性（Performance Efficiency）や保守性（Maintainability）が対象となる．

これらの品質特性を，非機能要求を定めてアーキテクチャ設計することで，製品開発部門のアーキテクトがプロトタイプにフィードバックすることができる．

3.3　多重フィードバック構造で生じる問題

多重化したフィードバック構造を形成することで，プロトタイプの開発コントロールとコミュニケーションの複雑化に起因した様々な問題が生じる（図 3）．

(1)　開発コントロールの複雑化

ソフトウェアの開発規模が大きくなったことで，個別の担当者が把握できる領域は限られる．フィードバックが多重に実行された場合，プロトタイプの開発コントロールが複雑化し，以下の問題が生じる．

フィードバック対象のソフトウェア部品が変化して作業がムダになる．たとえば，成熟性向上のために単体検査で欠陥を除去した部品をプロトタイプに反映しようとしたとき，要求開発における仕様変更で部品そのものが削除されていることが起こりうる．

フィードバックの相互作用でソフトウェアが予定外の動作を示す．たとえば，機能正確性を向上させた部品をプロトタイプに反映させたとき，機能向上していたとしても，ソフトウェアの挙動は変化しており，要求開発における顧客デモで担当者が把握していない挙動を起こしうる．

また，要求開発で部品が変更されたことに気付かずに，他のフィードバックで変更前の部品に無秩序にロールバックすると，原因不明の挙動変化だけでなく，原因解析のためのムダな作業を引き起こす．

(2)　コミュニケーションの複雑化

顧客からフィードバックを得て商品性を高めるために，要求開発におけるプロトタイプの更新頻度は高く，更新間隔も短い．フィードバックを多重に実行する場合，開発者の増加でコミュニケーションが複雑化し，互いに背反した以下の問題が生じる．

要求開発された要求仕様や制御仕様は開発末期に文書化される．参照文書が存在しないことで，安定性を向上させるフィードバックにおいて，機能正確性や成熟性を高めるための効果的な検証やテスト設計ができずに，作業のムダや品質低下を引き起こす．

一方，安定性，保守性を確保するための情報収集で，各担当が個別に要求開発担当者にアクセスすると，アクセス過多によって要求開発の作業が妨げられ，開発速度の低下と商品性の低下を引き起こす．

これらのプロトタイプの開発コントロールとコミュニケーションの複雑化を軽減して各フィードバックの効果を最大化するために，多重化したフィードバック機構を統合するフィードバックモデルを開発する必要がある．

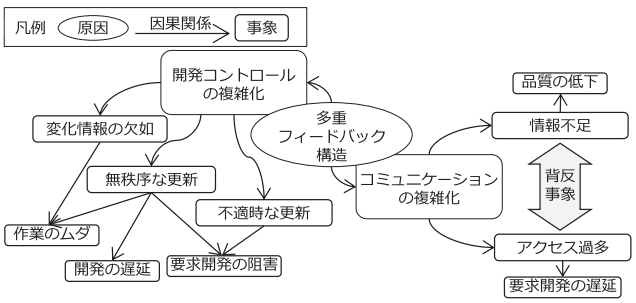


図 3　多重フィードバック構造で生じる問題

Fig. 3　Problems with multiple feedback structure.

4.　コンカレントフィードバックループモデル

4.1　コンカレントフィードバックループモデルの概要

本稿で提案する，進化型プロトタイピングにおけるコンカレントフィードバック開発方法の中核技術として，異なる組織が協働するためのコンカレントフィードバックループモデルを提案する．本モデルを組織，開発プロセス，アーキテクチャに展開することで，コンカレントフィードバック開発方法を設計する．

本ループモデルは，図 4 に示すように 3 パターンの異なるフィードバックループから構成される．プロトタイピングループとエボリューションループは，プロダクトを成長させるうえで対照的に作用するフィードバックループである．アーキテクチャループは，アーキテクチャを介してコンカレントなループ活動を支えるフィードバックループで

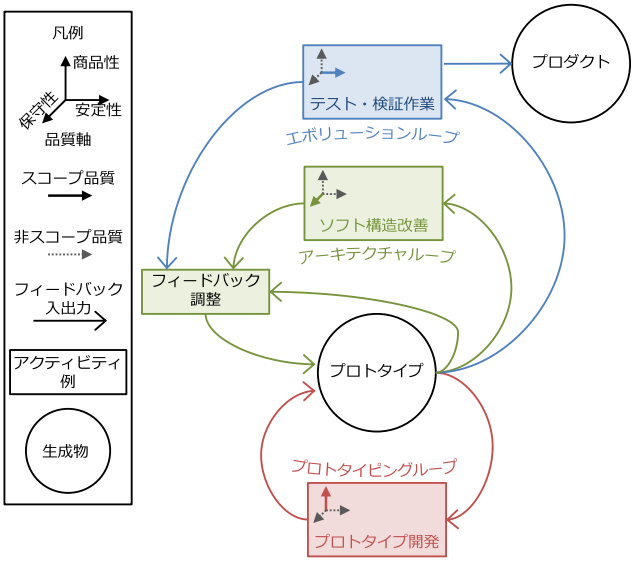


図 4　コンカレントフィードバックループモデル

Fig. 4　Concurrent feedback loop model.

表 1 ループパターンの特徴
Table 1 The characteristic of loop patterns.

	ループパターン		
	プロトタイピングループ	アーキテクチャループ	エボリューションループ
周期	短い	変則	長い
人数	少数	1〜2人	多数
ループ数	1以上	1	1以上
価値	商品性	保守性	安定性
品質特性	<利用品質> 有効性，満足性 リスク回避性 利用状況網羅性	–	–
	<外部品質> 使用性，障害許容性 機能完全性 機能妥当性	<外部品質> 性能効率性 保守性	<外部品質> 機能正確性 成熟性 セキュリティ
担当指向性	研究開発部門	アーキテクト	製品開発部門
適したループモデル	BML	OODA PDCA BML	PDSA
主な活動	・顧客と協調してフィーチャと技術を開発する	・アーキテクチャ設計や品質指標を計測して改善する ・コンカレントなループ活動を支援する	・テストや検証作業を通してプロダクトに負荷を掛ける

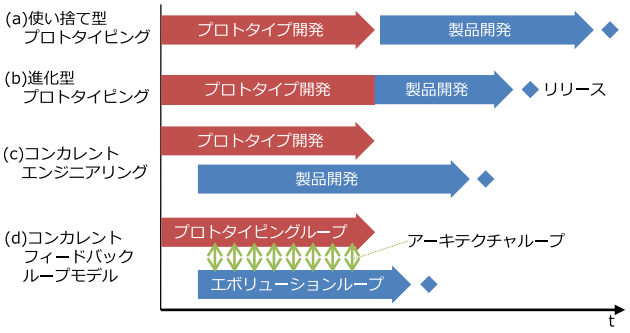


図 5 開発期間短縮メカニズム
Fig. 5 Development period shortening mechanism.

ある．**表 1** に各ループの特徴を示す．

プロトタイピングを前提とした開発形態による開発期間短縮メカニズムの差異を**図 5** に示す．従来のプロトタイピング開発においても，(a) 使い捨て型プロトタイピングから (b) 進化型プロトタイピングに切り替えることで，一定の開発期間短縮効果が期待できる．使い捨て型プロトタイピングでは，一部もしくはすべての仕様開発のために構築したプロトタイプソフトウェアはその時点で破棄して，製品開発ではアーキテクチャ設計からやり直す．一方，進化型プロトタイピングではプロトタイプソフトウェアを継続して開発することで，製品開発での開発量を低減させることができる．開発量が小さくなることで，プロトタイプ開発後に必要となる開発期間を短縮することができる．

さらに，(c) コンカレントエンジニアリングの概念を導入することによって開発期間のいっそうの短縮が期待できる．進化型プロトタイピングに切り替えても，プロトタイプ開発では自動車製品に求められる安定性や保守性が確保される保証がない．プロトタイプソフトウェアを引き継いで製品開発するには，これらの品質を確保するための開発や検証が必要となる．コンカレントエンジニアリングを適用した場合，プロトタイプ開発において安定性や保守性をあらかじめ保証することで，品質確保のための開発や検証をフロントローディングできる．この結果，製品開発で必要な工数を低減させ，プロトタイプ開発後に必要となる開発期間をさらに短縮することができる．

しかしながら，プロトタイプ開発と並行してプロトタイプを進化させようとすると，3.3 節で示した問題によって開発期間短縮効果が制限される．(d) 本ループモデルでは，これらの問題を軽減することで，コンカレント化した開発期間短縮効果を極大化できる．

4.2 プロトタイピングループ

プロトタイピングループ（以下，P ループと略記）は，利用品質や外部品質を明らかにし，商品性の向上をスコープとして，少人数の開発者が短い周期でループを回して製品のプロトタイプを開発する．素早いフィードバックで顧客から要求を引き出し，試行を繰り返して製品の要素技術を開発する．

本ループは開発対象に対して開発要員数だけ同時並行に駆動される．1 つ 1 つのループは，顧客からのフィードバックや要素技術の評価結果を受け，各開発要員が自律的に実行する．アジャイル開発におけるストーリ開発 [3], [23] と同様に，個々のアイテム開発が個々のループに相当する．

本ループはプロジェクトの開始と同時に起動する．顧客からの要求と，アーキテクチャループからのアーキテクチャ設計と設計改善方針を入力とし，プロトタイプを更新する．要求仕様と制御仕様を出力することでループ活動を終了する．研究開発部門による BML ループでの実行が適している．

4.3 エボリューションループ

エボリューションループ（以下，E ループと略記）は，安定性の向上をスコープとして，多人数の開発者が時間をかけてプロトタイプをプロダクトに進化させる．P ループで構築したプロトタイプに対して，テスト・検証を繰り返して欠陥を抽出することで機能正確性や成熟性を確保する．

単体テストや結合テスト，設計検証など，必要な要員数と周期が活動ごとに異なることから，本ループは開発対象に対して複数駆動される．1 つ 1 つのループは，確保したい対象品質ごとにチームを割り当てて計画的に実行する．1 つのループは計画した品質保証活動を完了し，アーキテクトに結果を引き継いだ時点でループを閉じる．

本ループはアーキテクトからプロトタイプに関する最初の情報を入手した時点で起動する．P ループによって更新されるプロトタイプと，アーキテクチャループからの要求仕様や設計，開発状況などの諸情報を入力とする．品質向上のための改善情報をアーキテクチャループに出力し，

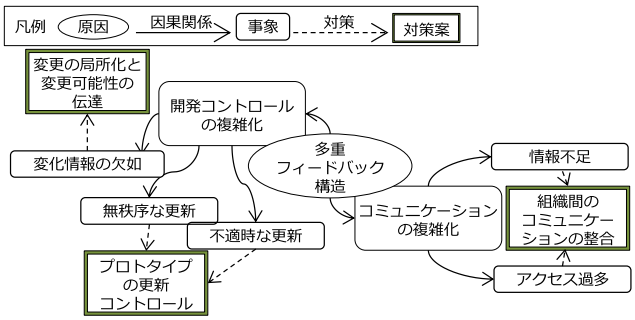


図 6 A ループにおけるフィードバック構造問題への対処
Fig. 6 Dealing with feedback structure problem in A-loop.

ソフトウェア製品のリリースでループ活動を終了する．E ループは基本的には自律的に活動し，アーキテクトからの情報入力で活動計画を更新する．製品開発部門による PDSA サイクルでの実行が適している．E ループは，アーキテクチャループを介して間接的にプロトタイプコードを改造する．

4.4 アーキテクチャループ

アーキテクチャループ（以下，A ループと略記）は，性能効率性や保守性などの外部品質を確保し，保守性の向上をスコープとして，コンカレントなループ活動を維持するためのアーキテクチャを提供する．3.3 節で示した問題に対処するため，ソフトウェア構造の変更を局所化したうえで変更する可能性を伝達する機能と，プロトタイプの更新をコントロールする機能，ループを担う異なる組織間のコミュニケーションを整合する機能を，A ループは備える（**図 6**）．

本ループは開発対象に対してアーキテクトによって 1 つだけ駆動される．1 つ 1 つのループは，外部品質を確保するための自律的なプロトタイプ改造か，E ループからの入力を受けた他律的な調整をアーキテクトが実行して閉じる．

本ループはプロジェクトの開始と同時に起動し，ソフトウェア製品のリリースでループ活動を終了する．本ループは P ループの活動と E ループの活動の両方の状況に対応しながら，アーキテクトがループの周期を定めずに実行する．

A ループは，開発進捗とプロジェクトの状況に応じて適切なフィードバックモデルを選択して実行する．たとえば，計画に基づいてアーキテクチャ設計などのアクティビティを実行できる場合は PDSA サイクルを選択する．品質指標を計測しながら設計を改善する場合は BML ループを選択する．また，P ループと E ループの調整を進める場合は OODA ループを選択して実行する．

4.4.1 外部品質の確保活動

要求された品質特性をシステムが示すことを保証するアクティビティと戦術の提供を目的に，アーキテクチャパースペクティブが提案されている [21]．アーキテクチャパースペクティブを適用することで，機能などの複数の独立した関心事を横断して，アーキテクトがアーキテクチャを設計する．

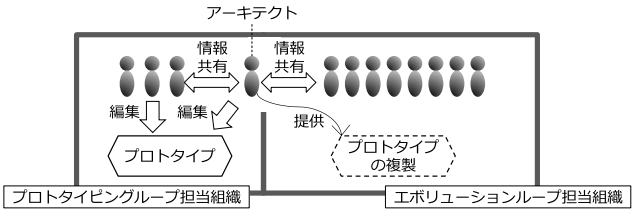


図 7 アーキテクトによるコミュニケーション摩擦の軽減
Fig. 7 Reduction of communication friction by the architect.

この考え方に基づき，A ループでは，性能効率性や保守性などの外部品質をパースペクティブとして，これらの品質特性を評価することで，アーキテクチャを分析，検討，改善して品質を確保する．アーキテクトは P ループによる実装結果を基に，要求仕様や技術などの関心事をその実現方法とともに理解し，プロトタイプの進化に合わせて外部品質を確保する．

4.4.2 変更の局所化と変更可能性の伝達

A ループは，プロトタイプのソースコードから共通コード，汎用コードを探して部品化して，安定性の高い部分を構造的に分離する．部品化された関数は変更される可能性が低くなる．部品の変更可能性をアーキテクトが伝達し，これらの関数を活動対象とすることで E ループの活動のムダが低減される．その結果，活動の効率が上がり，フィードバック効果が向上する．

4.4.3 プロトタイプの更新コントロール

A ループは E ループから改造コードや欠陥指摘をプロトタイプにすぐには反映させず，一時的に蓄える．プロトタイプの変化を追跡して E ループへの入力時と同じソフトウェア部品に改造を反映させることで，E ループの改造を適合して作業のムダを低減する．

また，P ループの活動に合わせて適切なタイミングで改造を反映させることで，P ループの顧客デモなどの活動を阻害する要因を除去する．これらのプロトタイプを更新コントロールする活動によって，A ループは P ループと E ループの活動の衝突を緩衝するバッファとして機能する．

4.4.4 組織間のコミュニケーションの整合

複数組織が交わると，人員増加にともないコミュニケーション摩擦を起因とした開発力低下を生じる．これを軽減するため，組織間のコミュニケーションのインタフェースをアーキテクトに限定する（**図 7**）．この役割は，Coplien の組織パターンにおける防火壁（Fire Wall）と門番（Gate Keeper）の 2 つのロールに基づく [8]．

E ループ担当組織に対してアーキテクトは門番として働き，P ループの開発進捗や状況を適切なタイミングで提供する．たとえば，E ループで単体テストに仕掛かっている関数部品が削除・変更された場合には，速やかに連絡することでムダを最小限に抑えるよう働く．また，テストや設

計検証に必要な要求仕様の情報をEループに伝えることで，プロトタイプの理解を助ける．

一方，Pループに対してアーキテクトは防火壁として働く．Eループで発生する不明点の問合せをアーキテクトでフィルタすることで，アクセス過多によるAループの集中力の低下を防ぐ．また，先に述べたようにEループで生じたプロトタイプの改造をバッファリングすることで，Pループが把握できていない変更が生じることを防ぐ．

5. コンカレントフィードバック開発方法

提案するコンカレントフィードバック開発方法（以下，CF開発方法）を，自動車システムの新製品開発に適用した．この適用例を用いて提案するCF開発方法を説明する．

5.1 CF開発方法のフレームワーク

提案するCF開発方法は，進化型プロトタイピングのコンテキスト上に，前章で提案したコンカレントフィードバックループモデルを展開して設計する．設計したフレームワークは，次に示す組織構成，アーキテクチャ設計，開発プロセスの3つの要素で成り立つ．

組織構成において，コンカレントフィードバックループモデルで中核をなすAループを実行するアーキテクトを規定し，調整活動を実行するアーキテクチャの構成管理方法を規定する．

アーキテクチャ設計では，筆者らの組織で採用している既存のアーキテクチャを説明し，そのうえでAループの活動方針におけるアーキテクチャ戦術を定義する．

開発プロセスにおいて，3つの各ループの具体的なアクティビティを展開する．

これらの役割と仕組みとプロセスにコンカレントフィードバックループモデルを展開し，研究開発と製品開発を同時並行に実行することをCF開発方法の特徴とする．

5.2 CF開発方法の組織構成

図8に2フェーズ開発方法と比較したCF開発方法の組織構成を示す．開発組織は，研究開発部門と製品開発部門の2部門で構成される．マルチアプリケーション開発に基づき，各アプリケーションに開発チームとプロジェクトリーダ（以下，PL）が存在し，PLを束ねるプロジェクトマネージャ（以下，PM）が存在する．各部門の開発チームは，部門ごとに方針を定めてチームを分割している．

CF開発方法として，コンカレントフィードバックループモデルを，アーキテクトとソフトウェア構成管理の2つの要素に展開する．

5.2.1 アーキテクト

アーキテクトは，コンカレントフィードバックループモデルにおけるAループを担う．そのために，Aループを実行するためのスキルと，従事するための独立性，Pループに対する情報レベルの確保が必要となる．

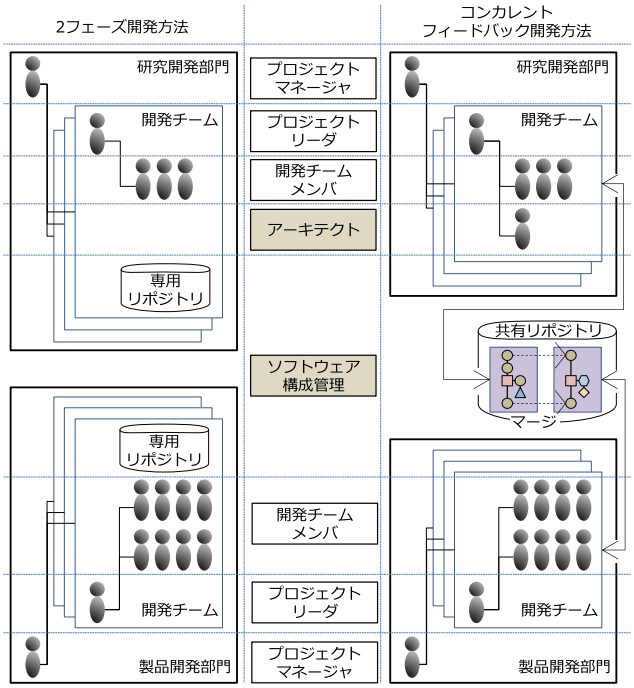


図8 CF開発方法の組織構成
Fig. 8 Organization structure of the CF development method.

スキルを確保するため，製品開発部門のメンバ，あるいは製品開発フェーズ経験者から担当を割り当てる．

独立性を確保するため，PM，PLとは異なる担当者を割り当てる．顧客向けのデモなどの活動への参加は必須としないほか，Pループ，Eループに参加して工数不足を補うことも禁止する．

情報レベルを確保するため，研究開発部門の開発チームと同じ配属とする．

5.2.2 ソフトウェア構成管理

プロトタイプを両部門で共有し，Aループでプロトタイプの更新をコントロールする仕組みを設けるため，リポジトリを共有化し，部門ごとにストリームを分割する構成管理とする．

各部門のループ活動を独立させて，異なるループによるストリーム操作の混入を防ぐことを目的とする．また，アーキテクトの意図で時間差を設けてストリームをマージすることで，プロセス上の混乱を軽減する．Eループの活動はアーキテクトを介してPループのストリームに反映されて後，Eループのストリームへと循環して反映される．

プロトタイプが進化したプロダクトは製品開発部門が所有するEループのストリームからリリースする．プロトタイプ開発における顧客デモはPループのストリームからリリースする．

5.3 CF開発方法のアーキテクチャ設計

CF開発方法のアーキテクチャ設計の説明の前提とし

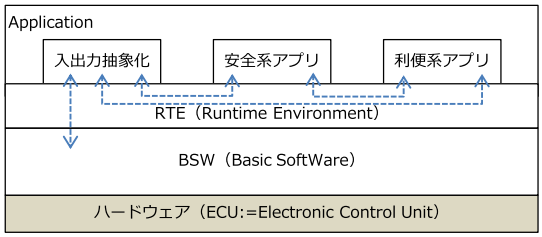


図 9 ソフトウェアアーキテクチャの開発ビュー
Fig. 9 Deployment view of software architecture.

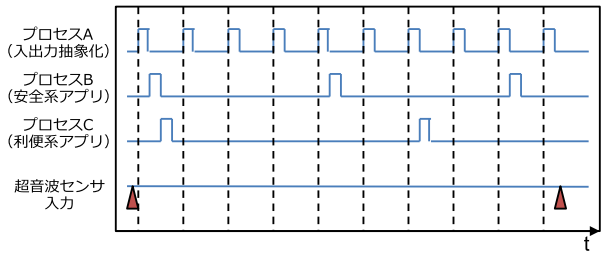


図 10 ソフトウェアアーキテクチャの並行性ビュー
Fig. 10 Concurrency view of software architecture.

て，筆者らの組織で採用しているソフトウェアのアーキテクチャを説明する．本アーキテクチャに則った場合の，A ループにおけるパースペクティブに対するアーキテクチャデザイン戦術を示す．

5.3.1 ソフトウェアアーキテクチャ

プロトタイプと製品で共通の，ソフトウェアアーキテクチャにおけるモジュール構造モデルと並行性モデルを示す．本アーキテクチャに基づいて，進化型プロトタイピングとコンポーネントのアーキテクチャデザイン戦術を実現する．

図 9 に開発ビューポイントのモジュール構造モデルを示す．モジュール構造は，Basic Software（以下，BSW）層と Application 層の 2 層のレイヤ構造からなる．BSW 層には自動車ソフトウェアの標準規格である AUTOSAR を採用している．Application 層は入出力抽象化と安全系アプリ，利便系アプリの開発対象パッケージが存在している．これらのパッケージは，AUTOSAR で提供される通信インタフェースである RTE（Runtime Environment）で接続されており，開発ビュー上の依存関係を排除している．AUTOSAR を採用し，コンポーネント間を疎結合に保つことで，アプリケーション開発の独立性を高め，プロトタイプからプロダクトに進化させるべき対象コンポーネントを容易に製品に結合することができる．

図 10 に並行性ビューポイントの Application 層に対する並行性モデルを示す．Application 層のコンポーネントは，入出力抽象化プロセス A と，安全系アプリプロセス B，利便系アプリプロセス C の 3 つの周期駆動プロセスに分類される．アプリケーションの実現に不可欠なプロセス A は最も短い周期で駆動し，実行優先度が高い．安全系アプリを実現するプロセス B は 2 番目の長さの周期と実行優先度を持つ．利便系アプリを実現するプロセス C は，安全系アプリの動作を阻害しないよう，最も長い周期と最低優先度で実行される．すべてのプロセスはシステムにおけるアプリケーション動作のトリガとなる超音波センサの入力周期よりも短く設計している．駆動する周期回数に余裕を設けることで，処理時間の負荷を分散するバッファを設けている．

5.3.2 アーキテクチャデザイン戦術

CF 開発方法におけるパースペクティブとして，性能効率性の指標として処理時間とメモリ使用量を，保守性の指標として関数部品のソースコード行数と複雑度 [17] を選択した．

ソフトウェアのアーキテクチャデザインで確保する必要のある品質特性には，信頼性（Availability）や相互運用性（Interoperability），変更容易性（Modifiability）など多くの特性が存在する [2]．近年の自動車システムのソフトウェアでは，短期間に機能拡張を求められるようになっていることから，3.2 節で述べたように，機能拡張が容易であるソフトウェアを構築したい．

組込ソフトウェアでは，製品リリース後の資源拡張は一般的に困難である．このため，適用対象製品において機能拡張容易性を保証するために CPU 資源とメモリ資源に余裕が必要であることから，性能効率性の指標に処理時間とメモリ使用量を選択した．対象製品が通信バスや共有メモリを資源として用いる場合は，レイテンシやスループットを指標に追加する必要がある．

また，機能拡張容易性の評価指標として関数部品のソースコード行数や凝集度，結合度，複雑度，ファンイン・ファンアウト数などの様々な指標が存在する．本適用では，定量的にツールによる測定が容易で，リファクタリングの指標としての活用しやすさを考慮し，関数部品のソースコード行数と複雑度の組合せを選択した．

これらのパースペクティブと E ループでのムダの削減方針に基づいて，下記のアーキテクチャデザイン戦術を採用した．

(1) 処理の時間的分散

5.3.1 項で示したとおり，プロセスの駆動周期にはバッファが設けられている．コンポーネントが実行する処理の時間的な優先度を分析し，実行周期をずらせる処理を分散することで，平均的な CPU の稼働率を低下し，性能効率性を向上させる．

(2) 反復処理・汎用処理の最適化

共通部品，汎用部品を括り出し，不確実性の低い部品と高い部品の分離を優先する．共通化することで実行モジュールの ROM 使用量を削減できる．また，部品化を促すことで，複雑度の低下にも寄与する．

(3) 変更の局所化

要求開発に依存する処理についても，入出力処理のよう

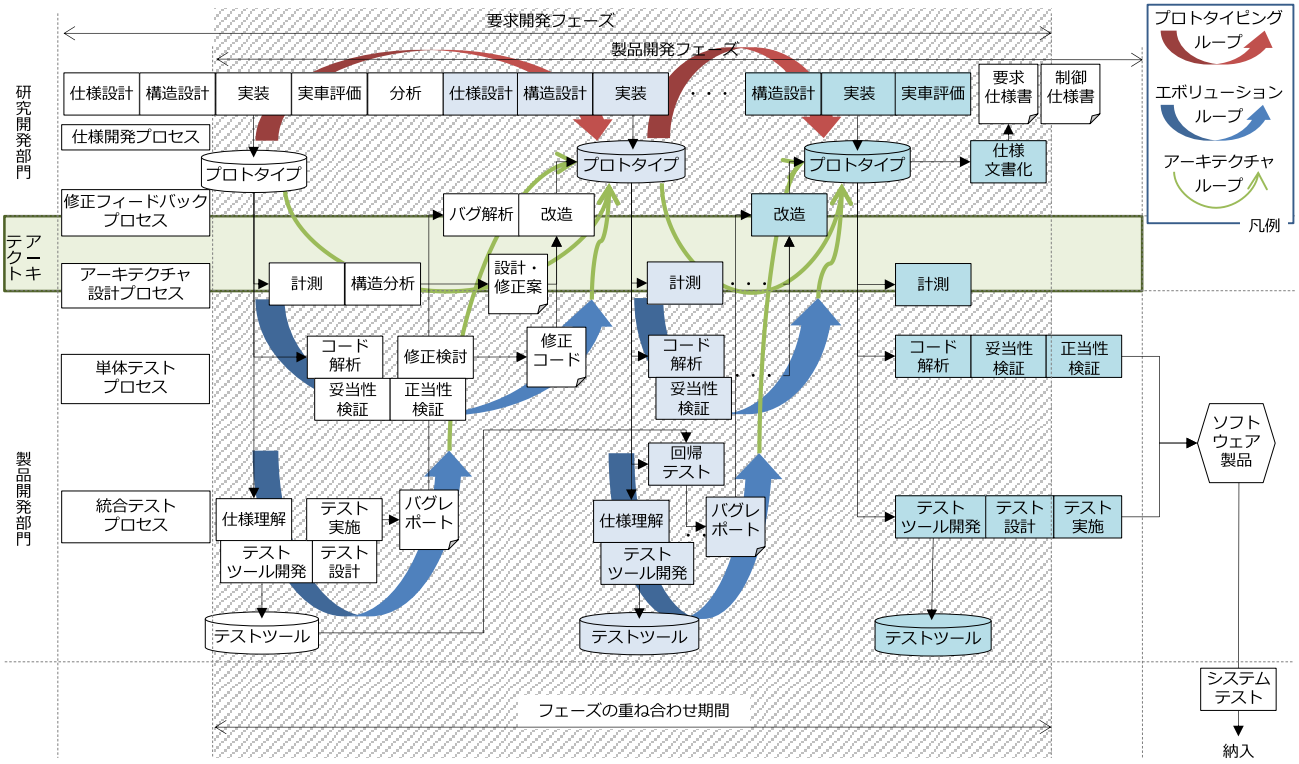


図 11　コンカレントフィードバック開発方法のプロセスフロー
Fig. 11　Process flow of concurrent feedback development method.

に，仕様変更の可能性が低いと判断される処理部分については，複雑度を考慮して小さな粒度で部品化する．

これらの部品について，アーキテクトが変更可能性の大，中，小を判断して，E ループへ判断結果を入力する．E ループは変更可能性の小さい部品から優先してテスト対象とすることで，部品変更によるムダのリスクを軽減できる．

5.4　CF 開発方法の開発プロセス

図 11 に CF 開発方法のプロセスフローを示す．従来プロセスを分解し，コンカレントフィードバックループモデルの各ループに則って展開した．開発プロセスは，仕様開発，アーキテクチャ設計，単体テスト，統合テスト，遅延フィードバックの 5 つのプロセスから成り立つ．

5.4.1　仕様開発プロセス

仕様開発プロセスは従来プロセスと同じく仕様設計，構造設計，実装，実車評価，分析のイテレーションを繰り返してインクリメンタルにプロトタイプを開発する．このプロセスは研究開発部門が担当し，P ループを BML ループの考え方に基づき反復する．

5.4.2　アーキテクチャ設計プロセス

アーキテクチャ設計プロセスは，特定の品質指標を計測し，ソフトウェア構造を分析してアーキテクチャを設計，修正する．過去のプロトタイプ開発を分析した結果から，外部品質として低い傾向が見られた性能効率性と保守性を今回開発の計測対象とした．性能効率性の指標として CPU 使用量，メモリ使用量を，保守性の指標として複雑度を選択した．このプロセスはアーキテクトが担当し，A ループを回す．変更規模が大きく分析作業にコストを要するときは，製品開発部門に分析作業やコード修正の一部を委託する．

5.4.3　単体テストプロセス

単体テストプロセスは，プロトタイプのソースコードを解析し，妥当性確認（LSB，オーバフロー/アンダフロー，動作範囲内の冗長性確認）と正当性確認（静的解析・カバレッジ検査）を通して欠陥を抽出して修正コードを生成する．このプロセスは製品開発部門が担当し，E ループを PDSA サイクルで回す．

5.4.4　統合テストプロセス

統合テストプロセスは，仕様理解から統合テストツール開発，テスト設計/実施と，ソフト変更時の回帰テストを通してバグレポートを生成する．このプロセスは単体テストと同様に製品開発部門で E ループを形成する．

5.4.5　修正フィードバックプロセス

修正フィードバックプロセスは，報告された修正案・バグレポートの内容を判断し，解析の要否，変更リスクを加味して，プロトタイプを更新する．このプロセスはアーキテクトが担当し，P ループと E ループの活動の衝突を緩衝する A ループを形成する．E ループからの修正コードをプロトタイプにすぐに反映させるのではなく，アーキテクトで 1 度蓄えることで，P ループの顧客デモなどの活動を阻

害しないタイミングで，E ループの改造が P ループの改造と衝突しないマージ方法で，プロトタイプを更新する．仕様依存が大きくアーキテクトでは修正是非を判断しきれない案件については，研究開発部門と検討・修正を協議する．

6. 超音波センサシステム開発への適用と評価方法

6.1 適用事例の背景

CF 開発方法を，超音波センサ応用システム開発プロジェクトの 2 つのアプリケーション開発に適用した．1 つ目は開発初期からの適用事例である．2 つ目は開発末期での適用事例である．筆者らの 1 人はアーキテクトとして開発に参加している．

適用事例は，車両周囲の環境を超音波センサによってセンシングし，車両の駆動ならびにステアリング操作を制御する自動車システムの新製品開発である．製品の市場投入後は，小変更をともなう複数車両への展開が計画されている．

本システムは，複数の独立したアプリケーションをコンポーネント分割して実現する．2 つの事例は異なるアプリケーションを構成する異なるコンポーネント開発への適用事例である．各事例の概要と，評価方法について示す．

6.2 事例 1：開発初期からの適用事例

事例 1 のプロジェクトは，仕様開発開始から車両試作納入までが 7 カ月間あり，納入の 1.5 カ月前に要求仕様書の発行マイルストーンが設定されていた．製品開発フェーズは，要求開発フェーズから 1 カ月後に開始する計画とした．

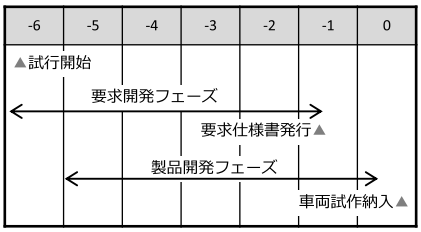
計画を立案するうえで開発規模の見積りが必要となるが，組織における類似システムの新製品開発には参考とできる実績がない．そこで，ソフトウェアの開発において，工数，開発期間，開発規模，生産性の関係から，最短開発期間を求める見積手法として知られる SLIM [19] を利用した．同規模の開発コンポーネントのソースコード行数から，最短の開発期間を SLIM で見積もり，開発期間の参考値とした．

計画と見積りについて**図 12** に示す．

6.3 事例 2：開発末期の適用事例

事例 2 のプロジェクトは，開発が末期まで進んでいる途中で CF 開発方法を適用した事例である．事例 1 の次の車両試作納入フェーズが開発期間にあたる．適用開始時点で，あらかじめ計画されていたプロトタイプ開発の期間を大幅に延長しており，適用後 1 カ月半の期間の中でプロトタイプ開発の収束と製品リリースの実現が求められていた．

事例 1 と同様に筆者らの 1 人がアーキテクトとして開発に参加しているが，CF 開発の A ループでの取決めとは異なり，研究開発部門の開発チームの PL も兼任している．



	SLIM 見積もり
規模	16KLOC
生産性指標 (社会平均)	13.0
組織構築率 (Max値)	5
最短開発期間 (月)	6.41

図 12 試行プロジェクトの計画と見積り
Fig. 12 Planning and estimation of the pilot project.

6.4 評価方法

提案する開発方法の効果を確認するために，開発期間の短縮効果と，アーキテクトによるフィードバック効果の評価を実施した．評価データはアーキテクトとして開発に参加した筆者らの 1 人が，実開発中に計測した結果に基づく．

7. 事例 1（開発初期からの適用）の評価結果

事例 1 の評価結果を示す．事例 1 では開発初期から試行プロジェクトとして計画的に取り組んでいるため，すべての評価結果が得られている．

7.1 開発期間の短縮効果

試行プロジェクトに CF 開発方法を適用した結果，納期遅れなく車両試作納入にソフトウェアをリリースできた．開発期間の短縮効果について，従来の 2 フェーズ開発方法，CE と，本稿で提案した CF 開発方法による結果とを比較し，その効果を示す．

従来の 2 フェーズ開発方法では要求仕様書発行から製品開発フェーズを開始している．SLIM での見積りを加算すると，約 12 カ月の開発期間を要する．CF 開発方法では約 7 カ月で開発を終えていることから，2 フェーズ開発方法に対して，CF 開発方法は約 5 カ月，41.7%の開発期間短縮効果が得られた．

CE が理想どおり働けば，要求仕様書発行時点で基本設計と詳細設計は完了していると見なせる．また，並行活動による冗長なテストは生じない．7.2.2 項で述べるように，検査におけるやり直しと無効な検査の比率は合計で 25%であった．そこで，CE で開発した場合には，製品開発部門の開発期間はこの 25%を減らして 5.5 カ月の 75%である 4.13 カ月と推定できる．この結果，研究開発部門の開発期間である 5.5 カ月に 4.13 カ月を加えて，CE では 9.63 カ月の開発期間を要すると推定できる．よって，CE に対して CF 開発方法では 9.63 カ月が 7 カ月へ短縮されることから，約 2.63 カ月，すなわち，約 27%の開発期間短縮効果が期待できる．

7.2 アーキテクトによるフィードバック効果

試行プロジェクトは，関数の新規追加，変更，削除を繰り返し，機能とアーキテクチャの開発，不具合修正を重ね

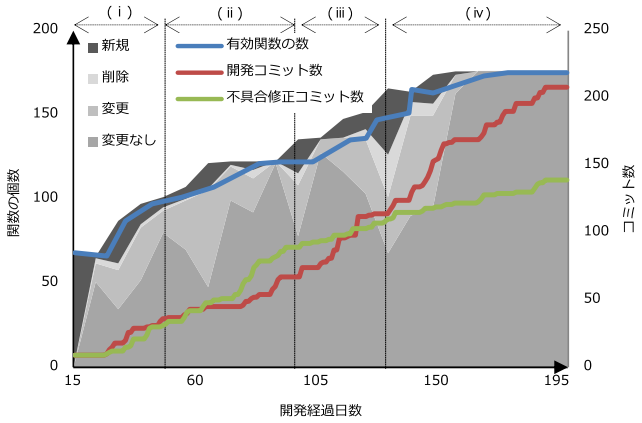


図 13 プロトタイプの進化の推移
Fig. 13 Transition in prototype evolution.

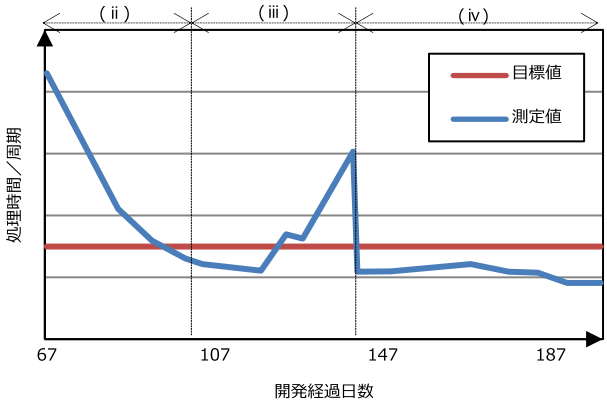


図 14 処理時間の推移
Fig. 14 Transition of CPU process times.

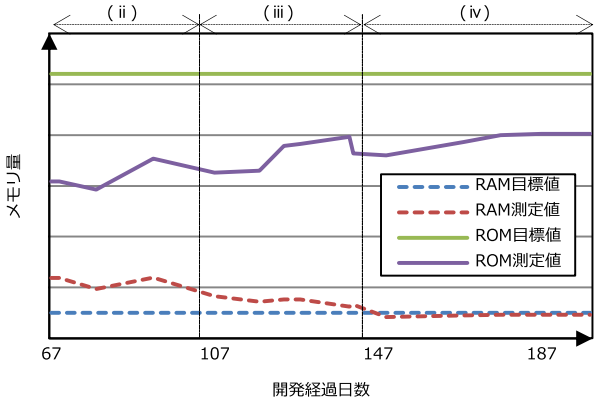


図 15 メモリ使用量の推移
Fig. 15 Transition of use of memories.

てプロトタイプをプロダクトに進化させた．顧客デモや納入イベントを大きな区切りとし，さらに 2 週間を目安に区切って計測した関数構成の変化と，日ごとのコミット数の推移を，開発のためのコミットと不具合修正のためのコミットに分けて図 13 に示す．

開発コミットの 85%は P ループによる機能開発，15%は A ループによるアーキテクチャ開発が占めていた．不具合修正コミットの 60%は P ループ，40%は A ループを通した E ループが占めていた．P ループの周期は 1〜2 週間で 15 イテレーション，E ループの周期は 3〜4 週間で 6 イテレーション回っている．A ループはアーキテクチャ設計の 1〜2 週間の周期に，E ループに合わせた整合活動を加え，開発を通して 12 イテレーション相当の活動を行った．

(i) の期間では，P ループによりプロトタイプ開発が進行した．(ii) の期間では，各ループが活動しているが，不具合修正の比重が高く開発が停滞している．(iii) の期間では顧客へのデモを優先して，不具合修正を後回しに開発を進めている．(iv) の期間に移るときにアーキテクチャに大きな変更を加え，その結果，機能開発と不具合修正が安定し，ソフトの変更規模が収束に向かっている．

これらの時期と並べて，アーキテクトによるフィードバック効果を，振舞いと構造，組織間のコミュニケーションの整合の 2 つの観点で評価する．

7.2.1 振舞いと構造への効果

振舞いへのフィードバック効果として，処理時間とメモリ使用量を評価し，構造へのフィードバック効果として，保守性の指標であるサイクロマチック複雑度を評価する．

(1) 処理時間

開発コンポーネントは，周期駆動するタスクである．1 周期にかかる処理時間を評価した．類似規模，類似機能の実績値を基に，システムが成立する許容値として設定した目標値とともに，測定値の推移結果を図 14 に示す．

開発コンポーネントは，主要な 2 つのサブコンポーネント，α，β で構成されている．計測開始時点では，α の処理時間が高く，目標値の約 3 倍の値を示していた．(ii) の期間において，A ループで繰返し処理の最適化や共通処理の括り出しを通してアーキテクチャを改善して目標値を達成した．(iii) の期間では β の構造劣化により処理時間が悪化したが，β の再構築で大きく処理時間を改善した．(iv) の期間では安定推移して目標値を達成して開発を終了した．(iii) の期間では，構造劣化により処理時間が悪化することが予想されていたが，アーキテクトによる再構築を条件に，機能開発を優先する開発方針を採択した．

(2) メモリ使用量

開発コンポーネントの ROM 使用量と RAM 使用量の推移を図 15 に示す．計測開始時点では，RAM 使用量が目標値に対して 2 倍以上の値を示していた．A ループで処理の汎用化や共通処理の括り出しを行い，E ループで保証精度に対して冗長だった変数サイズを最適化するなど連携してメモリ削減に向けて活動をした．その結果，ROM 使用量は低水準に保たれ，RAM 使用量も低減され，両使用量は目標値を達成して開発を終了した．

以上より，E ループと協調した A ループのアーキテクトによるフィードバックで，アーキテクチャの振舞いが改善され，外部品質のうち，性能効率性が確保できる結果が得られた．

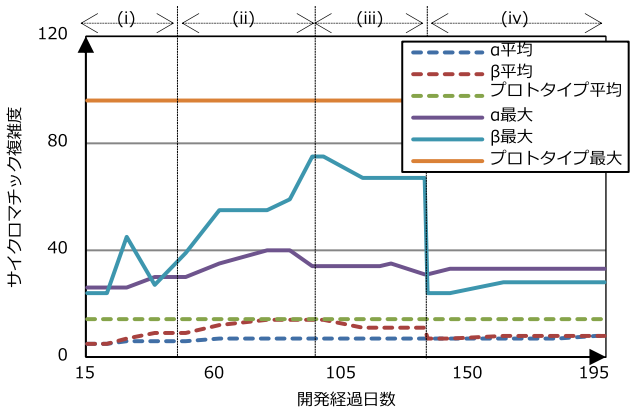


図 16 サブコンポーネント別の複雑度の推移
Fig. 16 Transition of complexities with sub-components.

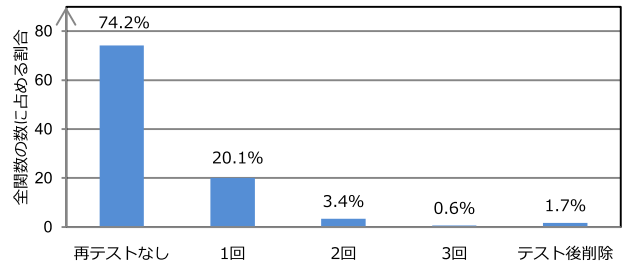


図 17 単体テストの実施回数
Fig. 17 The running number of unit tests.

(3) 複雑度

図 16 にサブコンポーネント別の複雑度の最大値と平均値の推移を，過去のプロトタイプ開発における類似規模，類似機能の実績値を比較対象にして示す．

(i) の期間では，α を優先して A ループで高凝集度を保つように関数分割した．その結果，α の複雑度は安定に保たれ，開発終了に至るまで一定の水準で推移した．

(ii) の期間に，β の複雑度が悪化した．開発進捗を確認したところ，追加予定の機能の残存数も多く，開発困難に陥ることが見込まれた．そこで，プロトタイプのストリームを分離し，仕様開発は継続してアーキテクトが β の再構築に着手した．

(iii) の期間は，処理時間は悪化したが機能開発は進められた．アーキテクトがソフトウェアを再構築し，再構築したソフトウェアに置き換えることで複雑度の水準は下がり，(iv) の期間は低水準を維持して開発を終了した．アーキテクトによる β の再構築がなければ，(ii) での不具合修正作業と同等の作業が継続して発生し，P ループの活動が停滞した可能性が高い．

以上より，A ループのフィードバックで，アーキテクチャの構造が改善され，外部品質のうち，保守性が確保できる結果が得られた．

7.2.2 組織間のコミュニケーションの整合への評価

組織間のコミュニケーションの整合効果として，図 17 に示すように単体テストの実施回数を記録し評価した．

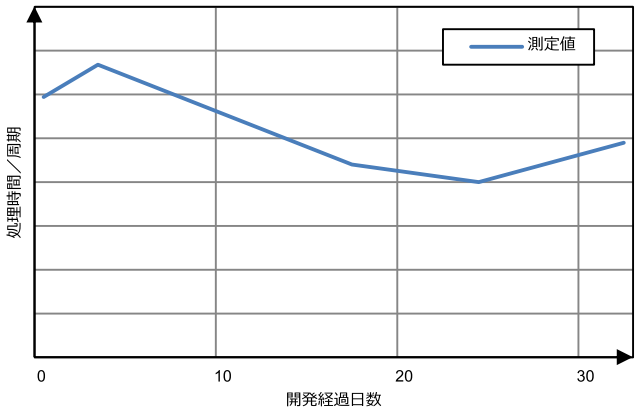


図 18 処理時間の推移
Fig. 18 Transition of CPU process times.

単体テストを最大 3 回やり直した関数が存在したが，約 75%の関数は再テストなしで完了している．また，単体テストを実施したもののその後削除されてしまった関数の割合は 2%以下と少ない．アーキテクトによる不確実性の情報伝達が機能したことで，E ループにおけるムダが軽減された．特に，β の再構築を早くから予告し，E ループの活動対象から外したことは効果として大きい．

A ループのアーキテクトによるフィードバックで，整合活動が有効に働いたことが確認できた．

8. 事例 2（開発末期での適用）の評価

事例 2 の評価結果を示す．事例 2 では適用期間が 1 カ月と短く，見積りや目標値が設定できていない．アーキテクトのフィードバック効果のうち，振舞いと構造への効果のみ評価し，その結果を示す．

8.1 処理時間

開発初期からの適用事例と同様に，1 周期にかかる処理時間を評価した．測定値の推移結果を図 18 に示す．

適用開始時点では，まだ追加すべき機能が残っていたが，処理時間は増加傾向をみせていた．E ループの協力を得て，計算用の変数サイズを全面的に見直し，A ループによりフィードバックをかけて処理時間を削減した．この間も機能追加は継続されたが，A ループによる処理時間の分散もあわせて実施することで，取組み開始時より 15%低減して開発を終えることができた．

8.2 メモリ使用量

開発コンポーネントの ROM 使用量と RAM 使用量の推移を図 19 に示す．機能がある程度搭載された開発中期のメモリ使用量を目標値としている．処理時間と同様に，計算用の変数サイズの見直しにより RAM 使用量は維持された．A ループによる処理の汎用化や共通処理の括り出しにより，ROM 使用量は低減され，機能追加を実施したうえ

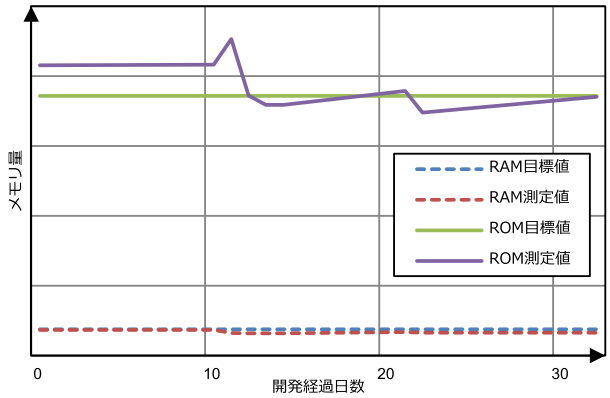


図 19 メモリ使用量の推移
Fig. 19 Transition of use of memories.

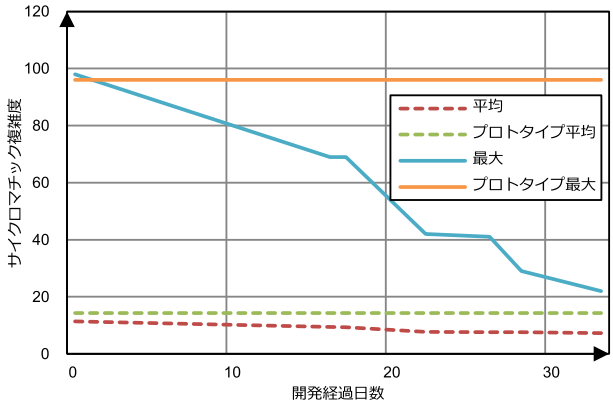


図 20 複雑度の推移
Fig. 20 Transition of complexities.

で，両使用量は目標値を達成して開発を終了した．

以上より，事例 1 と同様に，E ループと協調した A ループのアーキテクトによるフィードバックで，アーキテクチャの振舞いが改善され，外部品質のうち，性能効率性が確保できる結果が得られた．

8.3 複雑度

図 20 に複雑度の最大値と平均値の推移を，事例 1 と同様の比較値とあわせて示す．

事例 2 の開発コンポーネントは事例 1 の半分程度の開発規模であるが，取組み時点ですでに複雑度の最大値は過去の最大値を超えていた．プロトタイプ開発における仕様の急速な開発により，制御構造が悪化していたことが要因であった．制御構造を順番に整理して見直すことで，複雑度の最大値を低減させ，平均値もあわせて低減させ，いずれも過去の指標を大幅に下回ることができた．

以上より，事例 1 と同様に，A ループのフィードバックで，アーキテクチャの構造が改善され，外部品質のうち，保守性が確保できる結果が得られた．

9. 考察

9.1 開発期間短縮効果に対する考察

CF 開発方法の開発期間短縮効果は，進化型プロトタイピングに移行した効果が最も大きい．テスト・検証プロセスの多くがコンカレント化された効果も大きく累積されている．この効果は，A ループにより E ループの活動のムダが軽減されたことで，さらに有効に働いている．

しかし，仕様書発行から検証プロセス完了までにまだ 1.5 カ月を要している．開発終盤に仕様発行に向けて機能の追加・変更が頻発したことが要因としてあげられる．P ループにおける変更管理を強化するなど，まだ開発期間を短縮する改善が見込める．

9.2 外部品質確保効果に対する考察

事例 1 では，CF 開発方法により，プロトタイプ開発の早期から外部品質の確保活動を開始した．その結果，性能効率性と保守性の 2 つの品質指標の目標値を達成することができた．

外部品質を確保するためには，アーキテクチャの設計とともに，アーキテクチャの崩壊を防ぐための施策が重要となる [25]．CF 開発方法では，アーキテクトによりはじめのアーキテクチャが設計された後，アーキテクトによるアーキテクチャの直接的保全と，P ループ担当者によるアーキテクチャパターン模倣による設計スキルの向上，測定値の定期提供によるアーキテクチャ保全のための意識促進の 3 点が働いていた．

(1) アーキテクトによるアーキテクチャの直接的保全

A ループでは，アーキテクトによって継続的にアーキテクチャが見直され修復する保全活動が繰り返された．

直接的保全の最大の活動は，事例 1 のサブコンポーネント β に対しての活動である．開発当初，図 16 に示すように，β の複雑度は大きく，その影響でコンポーネント全体の効率も図 14 に示すように劣化傾向にあった．β の再構築により 2 つの指標値が改善されたが，性能と複雑度を測定し，β の知識を共有しているアーキテクトでなければこの施策は実行しえない．また，開発後期の性能改善は，計測と詳細な分析が必要であり，機能開発に工数を集中させている研究開発部門だけでは同様に実現困難である．

事例 2 でも短期間ながら同様の効果を示している．P ループしか存在しない取組み開始時点では，コンポーネントのアーキテクチャはすでに劣化しており，図 18，図 19，図 20 に示すように，外部品質の各指標はすべて悪化の傾向にあった．特に，保守性の指標である複雑度は開発規模が事例 1 の半分程度にもかかわらず，プロトタイプの最悪値を超えていた．

図 21 に，従来の要求開発フェーズで開発しているプロトタイプにおけるコンポーネントの複雑度と，CF 開発方

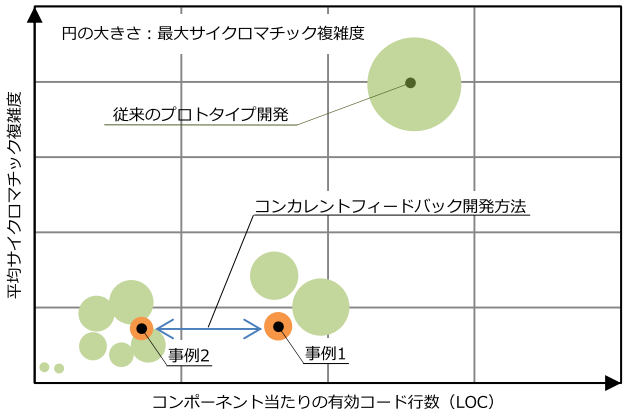


図 21 サイクロマチック複雑度の比較
Fig. 21 Comparison of cyclomatic complexity.

法で開発したコンポーネントの複雑度の比較結果を示す．従来のプロトタイプ開発では，開発規模が大きくなると複雑度の最大値は大きく（グラフ上の円の直径が大きく）なり，複雑度の平均値も反比例するように大きく（グラフ上の上方向の高さ）なっている．一方，CF 開発方法では，開発規模が大きくなっても，複雑度の平均，最大ともに同程度の水準を保っている．アーキテクトが A ループでアーキテクチャを直接的に保全し続けることで，複雑度を小さく抑え，保守性を向上させる結果が得られた．

(2) アーキテクチャパターン再利用による設計スキル向上

アーキテクトにより再構築されたサブコンポーネント β は，図 16 に示すように複雑度が小さく保たれ，保守性が安定した．同様に，図 14，図 15 に示すようにコンポーネント全体の性能も安定している．

アーキテクトがアーキテクチャを直接的に保全することで品質が確保されたことが主要因であるが，設計されたアーキテクチャパターンを P ループの開発者が再利用することで，設計が保全されたことも作用している．

再構築後のソースコードを手本とすることで，P ループの開発担当者はアーキテクチャの設計，実装パターンを学ぶことができる．パターンに沿って開発することで設計スキルを向上し，アーキテクチャの保全に寄与することができた．

(3) アーキテクチャ保全のための意識促進

性能の監視を開始するまでは，P ループの開発担当者における処理時間と使用メモリ量への関心は低く，開発担当者が元々備えている設計，実装スキルに依存した結果でしかなかった．

これに対して，指標値が定期的にレポートされるようになると，設計や実装の結果が性能に与える影響が各開発担当者にフィードバックされることで，開発担当者の意識が変容した．どのような設計，実装をすると性能が悪化するのかがフィードバックで学べることで，アーキテクチャを保全する意識が促進したと考えらえる．

その結果，開発終盤では，機能開発において，どの資源を消費すべきか，P ループ担当者からアーキテクトに意見が求められるようになった．アーキテクトも開発を通して分析・計測した結果から，適切なトレードオフを提示できている．協働と計測・学習によって，外部品質の確保効果が促進されたといえる．

9.3 アーキテクトの組織間調整に対する考察

実践におけるアーキテクトの役割に関するサーベイによれば，アーキテクトは単なるアーキテクチャ設計にとどまらず，調整などの役割を果たすことが求められている [28]．

本提案では，アーキテクトを研究開発部門の開発チーム内に配置した．この結果，研究開発部門はアーキテクトからソフトウェア再構築などのサポートが受けられた．関数部品の品質確保や回帰テストによるデグレード確認が製品開発部門で実行されたこととあわせて，開発を阻害する性能劣化や構造劣化を防いで，安定して商品性向上のための要求開発に集中できるようになった．

製品開発部門では，アーキテクトからソフト構造の変更可能性や開発状況を得られることで，仕様不明点が解消された．その結果，十分な期間を得て開発ドメインの知識を蓄えながら無理なくテスト工程を進めることができた．アーキテクトをプロトタイプ開発のロケーションから外して配置すると，情報レベルの低下を引き起こし，これらの効果は期待できない．

一方，アーキテクトには大きな負担がかかる．開発序盤から中盤は素早い変化に合わせたアーキテクチャの設計が求められる．中盤から終盤は多様な情報を把握して計測・改善活動を指揮しなくてはならない．開発を通して高い能力の発揮が求められる．CF 開発方法を採用する場合，アーキテクトは未経験者ではなく，スキルを有した人材を割り当て，かつその他の役割との兼任を避けて選任化させることが望ましい．

本開発方法において，アーキテクトに期待されるスキルセットを**表 2** に示す．このスキルセットは，適用事例 1，2 を通して筆者らの経験に基づいて抽出している．

アーキテクトにはアーキテクチャ設計に関するスキルのほか，目的が異なる複数の組織体を協働させるためのスキルが必要となる．協働のためのスキルが低いと，組織間の調整で衝突が生じ，各組織への混乱が生じうる．また，アーキテクト自身の負担が増えて活動の継続を妨げる要因となる．これらのスキルは，アジャイルプロジェクトのアーキテクトに求められる重要なアクティビティ [21] を実行するスキルと共通性が高い．

事例 2 においては，CF 開発方法の設計では推奨していない，アーキテクトと PL の兼任を試行した．その結果，アーキテクトの本来責務ではない機能開発や要素技術開発のために割かれる工数が増大した．これらのタスクは集中力を必要とする．実際，顧客対応に工数が費やされたこと

表 2 アーキテクトのスキルセット
Table 2 Skill set of architect.

スキル分類	スキル
アーキテクチャ設計	ソフトウェアの全体アーキテクチャ構造を表すビューを利用してモデルを作成するか，もしくはモデルを理解しており，構造の変更がシステムに与える影響を把握することができる．
	分割されたソフトウェア構成要素の単位で設計と品質特性を計測することでき，変更の影響を特定することができる．
	システムに必要な品質特性を特定して，パースペクティブを選択し，アーキテクチャに適用することができる．
コラボレーション	協働すべき組織体の開発プロセスを理解し，開発状況や開発要員の負荷状況を把握することができる．
	協働すべき組織体の開発目標と開発プロセス，アクティビティの重要性を理解し，組織間のアクティビティが衝突しないように干渉タイミングと手段を調整することができる．
	組織体の構成要員のスキル，性質を把握して，協働のためのアクティビティが推進されるよう適切な手段を選択して各要員に働きかけることができる．
	組織横断的に行動することができ，各組織体の信任を得ている．

で，アーキテクトとしての整合機能が停滞する場面が見られた．A ループが機能しなくなると，各ループ活動が整合されなくなり，開発が混沌とした状態に陥る．開発方法の設計どおり，アーキテクトはプロジェクト管理や技術開発する責務を兼任すべきではないという教訓を実体験として得ることができた．

9.4 関連研究との比較

提案方法を進化型プロトタイピングと CE，スパイラルモデル，アジャイル開発と比較する．

(1) 進化型プロトタイピングとの比較

進化型プロトタイピングは，プロトタイプを継続的に進化させるアプローチであり，外部品質の確保方法を規定していない [6], [9]．そのため，外部品質を確保するためのアーキテクチャの質は，開発メンバの備えるスキルへの依存度が高くなる（9.2 節参照）．

CF 開発方法では研究開発するロールと外部品質を確保するためのロールを分離することによって適切なスキルが割り当てられるように工夫することで，外部品質の確保を実現できた．

(2) CE との比較

従来の CE では，部門間，工程間が協働するための情報の取扱いを重視しつつも，フィードバック機構の機能差に着目した情報の整合方法までは考慮されていない [1], [7], [13]．ソフトウェア開発では，フィードバックの対象は設計だけでなく，プロトタイプのソースコードまで及ぶ．製造分野であれば，抽象的な設計だけでなく鋳型の直接的な加工も含めて情報として操作できることと同義である．

CF 開発方法では，情報操作の範囲の広がりに適応して，ソフトウェア開発で情報をコントロールして CE に取り組むことが可能となる．要求開発と実装も含めた CE の実運用モデルを提示し，開発を通して全体をフィードバックしていく具体的な方法を提供した．

(3) スパイラルモデルとの比較

スパイラルモデルは，開発の各フェーズでプロトタイプを開発する使い捨て型プロトタイピングである [4]．アーキテクチャ設計のフェーズで外部品質を対象にプロトタイプ開発することで品質を確保することは可能となる．

しかし，後戻りによる開発遅延は軽減できるものの，開発期間の短縮の面では使い捨て型を採用しているため効果が小さい．また，要求仕様や要素技術の確立に時間を要する場合には追従しきれない．CF 開発方法では研究開発に時間を要する場合でも追従して外部品質を確保することが可能となる．

(4) アジャイル開発の比較

アジャイル開発では，機能を横断したクロスファンクショナルチームが推奨されている [16], [23]．このチームにはアーキテクトも含まれ，十分なスキルを有したメンバを集められれば納期を短縮したうえで品質を確保することが可能である．

本稿の提案方法では，各組織にスキルが分散されている場合でも，各組織が有するスキルを統合して補完し強め合いながら納期短縮と市場要求への変化への対応，ならびに今後の製品展開までを視野に入れて開発活動に取り組むことが可能となる．複合的なチームの協働方法を提示し，組織的に活動を統合していく具体的な方法を提供した．

9.5 CF 開発方法の適用可能領域と前提条件

CF 開発方法を適用可能な情報システム領域について考察する．CF 開発方法のメリットとデメリットを述べ，適用可能な領域と適用の前提条件を示す．

(1) CF 開発方法のメリット

CF 開発方法のメリットは開発期間短縮効果と外部品質確保効果が高いこと，スキルレベルの異なる開発要員の開発活動を統合できることである．

要求仕様開発に時間を要しても，同時並行に外部品質を確保し，その後の展開開発に備えることができる．反対に，要求仕様が明確な開発領域では，CF 開発方法ではなく，従来の V 字モデルに従った開発方法で十分といえる．

研究開発や製品開発のスキルが組織として分散していても，各スキルを活用した活動を最大限に発揮することができる．反対に，開発に必要なスキルが 1 つの組織にまとまっている場合は，アジャイル開発に取り組み，開発活動を完結させる方が有効であろう．

(2) CF 開発方法のデメリット

CF 開発方法のデメリットは，アーキテクトを担える人材が必要であることと，開発組織間の協働関係の構築が必要であることである．

アーキテクトを担える人材，特に調整能力を備えた人材が欠如している場合，CF 開発方法の取り組みが困難となる．その場合は，開発期間の短縮を割り切り，スパイラル

モデルなどのリスクベースによる開発が有効であろう．

開発組織間の協調関係が構築されておらず，研究開発と製品開発が完全に独立している場合，CF 開発方法の取り組みは困難である．アーキテクトを研究開発組織に配置しても，人工として吸収されてしまうリスクが生じる．先進技術の開発担当が文化の異なる競合ベンダである場合や，開示の許可されていない特殊技術を扱う場合も適さない．

(3) CF 開発方法の適用可能領域

CF 開発方法は，プロトタイプ開発で要求仕様や要素技術を開発しており，製品に求められる品質レベルの高い領域において，開発期間の短縮が必要な情報システムが対象となる．すなわち，実現性の不確実性と要求品質が高く，開発期間の短い領域である．たとえば，まったく新しい製品で，かつ，高い品質が要求される情報システムを短期に開発する場合である．現在，新規システム開発の期間は短くなっており，このような開発条件は多くの情報システム開発で一般的となっている．

本稿では，このような領域で，実現性を解決するスキルと，要求品質を作り込むスキルを有する組織体が分散している場合の解決策を提供している．

(4) CF 開発方法の前提条件

CF 開発方法のメリット，デメリットと適用可能領域から，CF 開発方法を採用するための前提条件をまとめる．

製品の性質としては，機能面や技術面で不確実さを有することがあげられる．一方，開発組織として，これらの不確実さを低減させるためのスキルが，1 つの組織ではなく複数の組織に分散していること，組織間の開発活動はオープンであり，組織間の地理的分散が小さい，もしくはツールによる支援や協調関係の構築によりコミュニケーションの密度が大きく円滑であることがあげられる．さらに，アーキテクトを担える人材を有していることも必要である．

地理的な分散が大きく，コミュニケーションの密度が小さい，あるいは，遅い場合には CF 開発方法の効果は限定されるおそれがある．また，開発活動がクローズドな組織を含む場合は，CF 開発方法の効果は著しく限定されるおそれがある．

10. 今後の課題

本開発方法は新規開発向けに設計している．開発は次の納入のためのソフトウェアリリースに向けて継続される．しかし，製品開発部門の担当者は今回納入に集中することから，次リリースに向けた CF 開発に追従して工数を割くことが難しい．今後は，連続した開発に対応した CF 開発方法の設計を検討する．

11. まとめ

自動車システムにおいて開発期間短縮と品質確保を両立させるために，使い捨て型プロトタイピングから進化型プロトタイピングに移行した．そのうえで，協働のためのコンカレントフィードバックループモデルを提案・展開し，コンカレントフィードバック開発方法を設計して適用した．

その結果，製品開発の開発期間短縮に成功し，外部品質の確保を達成した．また，複数のフィードバックループを回して部門間で協働することで，ソフトウェアの外部品質活動が強化され，部門別の独立作業では得られないスキル向上と柔軟な改善活動に取り組むことが可能となった．

参考文献

[1] Aoyama, M.: Beyond Software Factories, *Information and Software Technology*, Vol.38, pp.133–143 (May 1996).

[2] Bass, L. et al.: *Software Architecture in Practice, 3rd ed.*, Addison-Wesley (2012).

[3] Beck, K.: *Extreme Programming Explained*, Addison-Wesley (2000).

[4] Boehm, B.: A Spiral Model of Software Development and Enhancement, *Computer*, Vol.21, No.5, pp.61–72 (1988).

[5] Boehm, B.: Some Future Software Engineering Opportunities and Challenges, *The Future of Software Engineering*, pp.1–32, Springer (2011).

[6] Carter, R.A. et al.: Evolving Beyond Requirements Creep, *Proc. RE '01*, pp.94–101, IEEE Computer Society (Aug. 2001).

[7] Carter, D.E. and Baker, B.S.: *Concurrent Engineering*, Addison-Wesley (1992).

[8] Coplien, J.O. and Harrison, N.B.: *Organizational Patterns of Agile Software Development*, Prentice-Hall (2004).

[9] Davis, A.M.: Operational Prototyping, *IEEE Software*, Vol.9, No.5, pp.70–78 (1992).

[10] Ebert, C. and Favaro, J.: Automotive Software, *IEEE Software*, Vol.34, No.3, pp.33–39 (2017).

[11] 林 健吾：コンカレントフィードバック開発方法の車載ソフトウェア開発への適用，ソフトウェアエンジニアリングシンポジウム 2015 論文集，情報処理学会，pp.48–56 (Sep. 2015).

[12] ISO/IEC 25010:2011, Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models (2011).

[13] Jo, H.H. et al.: Concurrent Engineering, *Computers & Industrial Engineering*, Vol.21, No.1-4, pp.35–39 (1991).

[14] John, C.: *Evolutionary Systems Development*, Plenum Press (1991).

[15] Katzenbach, A.: Automotive, *Concurrent Engineering in the 21st Century*, pp.607–638, Springer (2015).

[16] Martin, R.C.: *Agile Software Development*, Prentice Hall PTR (2003).

[17] McCabe, T.J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol.2, No.4, pp.308–320 (1976).

[18] Moon, T. et al.: Analyzing the OODA Cycle, *PHALANX Online*, Vol.35, No.2, pp.9–13, 34–35 (2002).

[19] Pendharkar, P.C.: A Probabilistic Model for Predicting Software Development Effort, *IEEE Trans. Softw. Eng.*, Vol.31, No.7, pp.615–624 (2005).

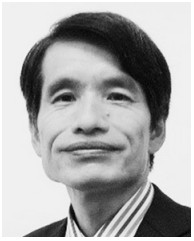
[20] Ries, E.: *The Lean Startup*, Crown Business (2011).

[21] Rozanski, N. and Woods, E.: *Software Systems Architecture, 2nd ed.*, Addison Wesley (2012). 榊原 彰（監訳）：ソフトウェアシステムアーキテクチャ構築の原理，

第 2 版，ソフトバンククリエイティブ (2014).
[22] Schätz, B. et al.: Model-Based Development of Embedded Systems, *Advances in Object-Oriented Information Systems*, LNCS, Vol.2426, pp.298–311, Springer (2002).
[23] Schwaber, K.: Scrum Development Process, *Business Object Design and Implementation*, pp.117–134, Springer (1997).
[24] Schäuffele J. and Zurewka, T.: *Automotive Software Engineering*, *2nd ed.*, SAE International (2016).
[25] De Silva, L. et al.: Controlling Software Architecture Erosion, *J. Systems and Software*, Vol.85, No.1, pp.132–151 (2012).
[26] Sollecito, W. and Johnson, J.K.: *McLaughlin and Kaluzny's Continuous Quality Improvement in Health Care*, Jones & Bartlett Pub. (2011).
[27] Weber, J.: *Automotive Development Processes*, Springer (2009).
[28] Weinreich, R. et al.: The Architect's Role in Practice, *IEEE Software*, Vol.33, No.6, pp.63–69 (2016).
[29] Zhang, J. and Cheng, B.H.C.: Model-Based Development of Dynamically Adaptive Software, *Proc. ICSE '06*, pp.371–380, ACM (May 2006).

林 健吾 （正会員）

2003 年名古屋大学大学院人間情報学研究科修士課程修了．同年（株）デンソー入社．ナビゲーションシステム，超音波センサを利用した先進安全システムのソフトウェア開発に従事．2018 年南山大学大学院ソフトウェア工学専攻博士後期課程修了．博士（ソフトウェア工学）．2016 年度情報処理学会山下記念研究賞受賞．



青山 幹雄 （正会員）

1980 年岡山大学大学院工学研究科修士課程修了．同年富士通株式会社入社．大規模ソフトウェア開発とプロジェクト管理，ソフトウェア工学の実践に従事．1986〜1988 年米国イリノイ大学客員研究員．1995 年 4 月〜2001 年 3 月新潟工科大学情報電子工学科教授．2001 年より南山大学数理情報学部情報通信学科教授，2009 年より情報理工学部ソフトウェア工学科教授．博士（工学）．クラウドコンピューティング，自動車組込みソフトウェア，機械学習ソフトウェア等を対象として，要求工学，ソフトウェアアーキテクチャ技術の研究と教育，人材育成に取り組む．著書『要求工学知識体系』（2011 年刊：共著）ほか多数．IEEE Software，IEEE Transactions on Services Computing 等の編集委員，本会理事を歴任．1993 年情報処理学会研究賞受賞．ソフトウェア科学会，自動車技術会，IEEE，ACM，SAE 各会員．

古畑 慶次

1988 年名古屋大学大学院電子機械工学専攻博士課程前期課程修了．同年（株）デンソー入社．デジタル信号処理，音声認識の研究開発，携帯電話，ナビゲーションシステムのソフトウェア開発に従事．2004 年より（株）デンソー技研センターに勤務．技術研修所にて高度ソフトウェア専門技術者の育成研修，現場の技術指導を担当．2015 年南山大学大学院数理情報専攻博士後期課程修了．博士（数理情報学）．