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 *

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

林 健吾
Kengo HAYASHI

This paper proposes an approach to concurrent feedback development in automotive systems software development. As the size and complexity of automotive software increase, it has become more difficult to ensure quality. At the same time, automobile manufacturers allow less time for development. Conventionally, we developed software in the two phases of “prototype development for developing requirements specifications” and “product software development based on requirements specifications”. In this article, we propose using concurrent feedback development using the prototype in the specification development. The proposed concurrent feedback loop model consists of three patterns of feedback loop to concurrently evolve the product. In our automotive system software development we demonstrated reduced development time and improved the internal product quality.

Key words :
concurrent development, evolutionary prototyping, automotive software, agile development, feedback control

1. はじめに

我々は，車両周囲の環境をセンシングして車両制御する車載システムの新製品を開発している．このシステムは，ユーザの実使用環境の影響を受け易く，製品の基礎技術も未だ確立していない．そこで我々は，**Fig. 1** に示すように 2 つの部門で仕様開発と量産開発のフェーズを分担して開発している．仕様開発フェーズでは，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・制御仕様を開発することで利用時品質と外部品質 [1] を確保している．量産開発フェーズでは，製品開発部門が要求仕様書と制御仕様書から製品版ソフトウェアを開発する．V 字型モデルに従って開発することで内部品質 [1] を確保している．

近年，システムの高度化，複雑化が進む一方，車両メーカからは商品力向上のために高品質かつ短期間での開発が求められている．この状況が量産開発フェーズの開発期間の短縮を引き起こしている．顧客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的プロトタイピングへの転換が必要となった．

しかし，開発スコープと必要とされる技術の違いから，仕様開発フェーズでは製品としての内部品質を確保することは難しい．例えば，仕様開発フェーズでは，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，

＊情報処理学会の了解を得て，ソフトウェアエンジニアリングシンポジウム 2015 論文集，pp.48-56 より一部編集して転載

品質の低下がみられた．量産開発フェーズでは開発期間の制約から，確保できる品質レベルは限られる．内部品質が低下すると，車種展開時の派生開発における品質リスクと納期リスクとなり，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，仕様開発のプロトタイプを利用して量産開発フェーズを早期に開始する方法を提案して取り組み，開発期間の短縮を果たした[2]．本稿では，内部品質の確保を狙いに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，仕様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が高品質を保って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内部品質確保効果により提案方法の有効性を示す．

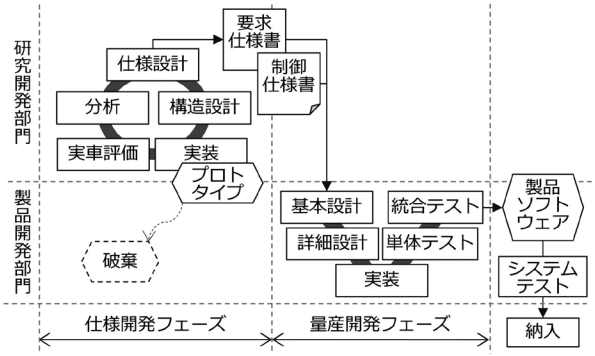

Fig. 1　Conventional 2-Phase Development Method

2. 関連研究

2.1 モデルベース開発

モデルベース開発は，モデル上で検証してコードを自動生成することで開発期間を短縮化する[3) 4)]．しかし，精度のよ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 進化的プロトタイピング

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的プロトタイピングは，開発期間の短縮に有効である[5)]．しかし，プロトタイプの目的は顧客からのフィードバックを得ることである．利用時品質や外部品質の確保には有効だが，短い開発期間では内部品質の確保が課題となる．

2.3 Agile 開発

XP, Scrum に代表される Agile 開発手法では，イテレーションをインクリメンタルに開発することで，製品のリードタイムを短縮化している[6) 7) 8)]．イテレーションごとの品質スコープが適切に定義されていれば，品質確保と短期開発が両立できる．しかし，車載システムの従来開発では，組織構造が適合しておらず，Agile 開発手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

コンカレントエンジニアリングは開発期間を短縮するために工程間・部門間の技術差を統合して開発する手法として提案されている[9) 10) 11)]．このアプローチでは，他部門同士の共同作業を通して，研究開発時から仕様，設計，試作品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，仕様開発を顧客と進める開発では，素早いフィードバックが第一であり，仕様開発を遅延させないフィードバック機構の構築が必要である．

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[12)]．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[12)]．また，顧客目線で迅速に不確実性へ対処する意図を強めた Build（構築）－ Measure（計測）－ Learn（学習）の BML ループ[13)]が提案されている他，Observe（観察）－ Orient（情勢判断）－ Decide（意思決定）－ Act（行動）の OODA ループといった意思決定プロセスも提案されている[14)]．いずれのモデルも，共通の活動を前提とした組織が対象である．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. コンカレントフィードバックループモデル

本稿で提案する，進化的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，**Fig. 2** 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上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ある．**Table 1** に各ループの特徴を示す．

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	エボリューションループ	アーキテクチャループ
周期	短い	長い	変則
人数	少数	多数	1〜2人
ループ数	1	1以上	1
品質スコープ	利用時・外部品質	内部品質	内部品質
価値スコープ	商品力	成熟度	保守性
担当指向性	研究開発部門	製品開発部門	アーキテクト
適したループモデル	BML	PDSA	OODA PDCA BML
主な活動	・顧客と協調してフィーチャと技術を開発する	・テストや検証作業を通してプロダクトに負荷を掛ける	・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

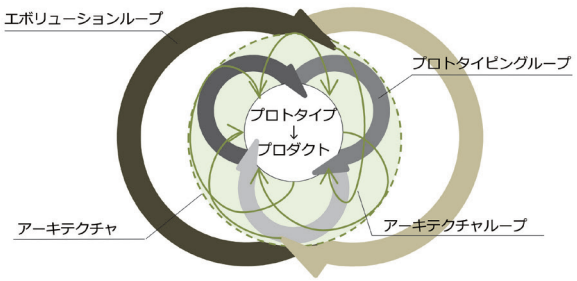

Fig. 2　Concurrent Feedback Loop Model

3.1 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）は，利用時品質や外部品質の確保と商品力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基礎技術を確立する．本ループは開発対象に対してひとつだけ駆動される．

本ループはプロジェクトの開始と共に起動し，要求仕様と制御仕様を完成させ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．

3.2 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）は，成熟度の向上をスコープとして，多人数の開発者が時間を掛けてプロトタイプをプロダクトに進化させる．P ループで構築され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内部品質を確保する．多角的に活動することから，本ループは開発対象に対して複数駆動される．

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し，製品ソフトウェア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．

3.3 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）は，保守性，効率性などの内部品質の確保をスコープとして，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．フィードバックが効果的に働くソフトウェア構造への誘導と，多重ループ構造によって生じるループ間の速度差の整合，ループを担う異なる組織間のコミュニケーションの整合を，アーキテクトが駆動する．本ループは開発対象に対してひとつだけ駆動される．

本ループはプロジェクトの開始と共に起動し，製品ソフトウェア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を基に，アーキテクトがループの周期を変えながら実行する．A ループは，開発進捗とプロジェクトの状況に応じてフィードバックモデルも変えながら機能する．

3.3.1 内部品質の確保活動

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という概念が提案されている[15]．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考慮して，アーキテクトがアーキテクチャを設計する．

この考え方に基づき，A ループでは，保守性や効率性などの内部品質を関心事として，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，フィーチャや技術などの関心事をその実現方法と共に理解し，プロトタイプの進化に合わせて内部品質を確保する．

3.3.2 フィードバック効果を高める構造への誘導

P ループは軽量に回すことを優先としているため，P ループで開発するプロトタイプのソフトウェア構造は変化しやすい．構造の不確実性が高く安定性が低いと，E ループでテスト・検証活動の対象としている関数部品が削除・変更されることで，E ループの活動がムダになり易い．

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上分離する．部品化された関数は変更される可能性が低くなり，これらの関数を対象とすることで E ループの活動がムダとなり難くなる．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．

3.3.3 ループ間の速度差を整合するプロセスの実行

P ループの開発周期は短くコードの変更量が大きい．E ループで改造する対象コードが変更されていた場合，前述の通り，改造が反映できず作業のムダが生じる．

また，P ループにおける顧客デモに対するソフトリリースの直前で，E ループによってプロトタイプが改造されたとする．P ループでは改造前の状態のプロトタイプで各種パラメータ値を調整しているため，P ループで認識できない改造が施されると，プロトタイプの挙動が把握できず，顧客デモに支障が生じ得る．

そこで，A ループは E ループから受信した改造コード，欠陥指摘を一時的に蓄える役目を担う．その上で，プロトタイプの変化を追跡して適切な箇所に改造を反映することで，E ループの改造を追従させてムダとなり得た活動を補完する．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することで，P ループの顧客でもなどの活動を阻害する要因を除去する．これらの活動によって，A ループは P ループと E ループの速度差を整合するバッファとして機能する．

3.3.4 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，人員増加に伴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，**Fig. 3** に示すように，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．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[16]．

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．例えば，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，プロトタイプの複製を部品の変更可能性と共に提供することで，E ループで気にしなくてはならない，プロトタイプの開発状態の理解を助ける．

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 ループが把握できない不意な変更が生じることを防ぐ．

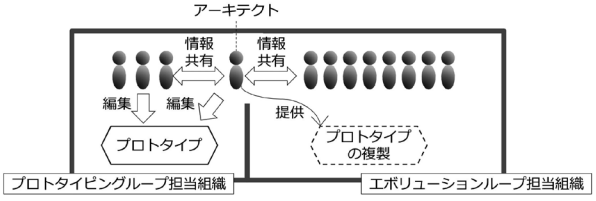

Fig. 3　Reduction of Communication Friction by the Architect

4. 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，車載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．

4.1 CF 開発方法のフレームワーク

提案する CF 開発方法は，進化的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．

4.2 CF 開発方法の組織構成

Fig. 4 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．

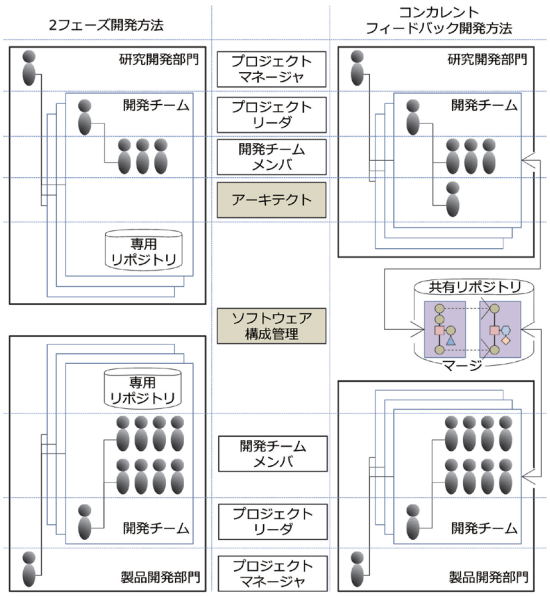

Fig. 4　Organizational Structure of the CF Development Method

4.2.1 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．

スキルを確保するため，製品開発部門のメンバ，或いは量産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 と担当者を分ける．顧客向けのデモなどの活動への参加は必須としない他，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．

4.2.2 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 ループで速度差を吸収するための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される．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．

4.3 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は，コンカレントフィードバックループモデルの A ループの方針に従い，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設計方針とした．仕様依存処理についても，入出力処理のように，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方針とした．

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．

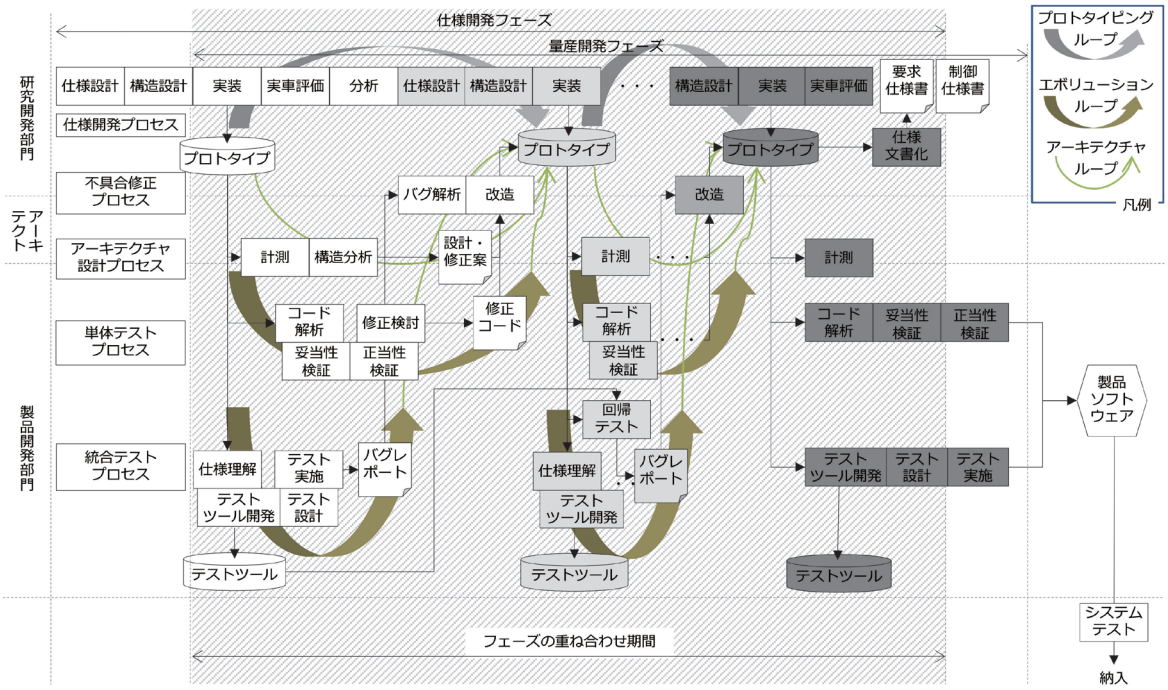

Fig. 5　Process Flow of Concurrent Feedback Development Method

4.4 CF 開発方法の開発プロセス

Fig. 5 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，不具合修正の 5 つのプロセスから成り立つ．

4.4.1 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 ループで回す．

4.4.2 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，内部品質として低い傾向がみられた保守性と効率性を今回開発の計測対象とした．保守性の指標として複雑度を，効率性の指標として CPU 使用量，メモリ使用量を選択した．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．

4.4.3 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，妥当性検証（LSB，オーバーフロー／アンダーフロー，動作範囲内の冗長性確認）と正当性検証（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．

4.4.4 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．

4.4.5 不具合修正プロセス

不具合修正プロセスは，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，プロトタイプソフトを修正する．このプロセスはアーキテクトが担当し，P ループと E ループの速度差を吸収する A ループを形成する．仕様依存が大きくアーキテクト

では修正是非を判断し切れない案件については，研究開発部門に検討・修正を委託する．

5. 超音波センサシステム開発への導入

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に導入した．筆者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車載システムの新製品開発である．製品の市場投入後は，小変更を伴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．本稿ではステアリング操作制御を伴うアプリケーションを構成するコンポーネント開発への適用事例について述べる．

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 SLIM[17] が知られている．組織における類似システムの新製品開発は参考とできる実績がないため，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 にて見積もり，開発期間の参考値とした．

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 ヶ月間あり，納入の 1.5 ヶ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る．量産開発フェーズは，仕様開発フェーズから 1 ヶ月後に開始する計画とした．計画と見積もりについて **Fig. 6** に示す．

	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

Fig. 6　Planning and Estimation of the Pilot Project

6. 評価

6.1 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が，実開発中に計測した結果に基づく．

6.2 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．

従来の 2 フェーズ開発方法では要求仕様書発行から量産開発フェーズを開始しており，SLIM での見積もりを加算すると，約 12 ヶ月の開発期間を要する．約 7 ヶ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 ヶ月，41.7% の開発期間短縮効果が得られた．

6.3 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，フィーチャとアーキテクチャの開発，不具合修正を重ね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 **Fig. 7** に示す．

開発コミットの 85% は P ループによるフィーチャ開発，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，フィーチャ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．

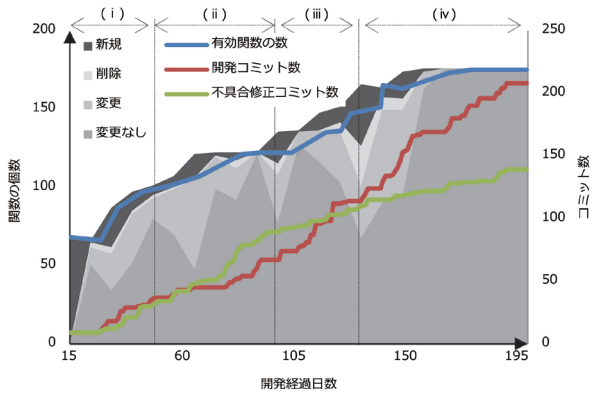

Fig. 7　Transition in Prototype Evolution

6.3.1 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 周期に掛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共に，測定値の推移結果を **Fig. 8** 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 ループで繰り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担保に，フィーチャ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 **Fig. 9** 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し，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，内部品質の内，効率性が確保できる結果が得られた．

(3)　複雑度

Fig. 10 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，追加予定のフィーチャの残存数も多く，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ブランチし，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フィーチャ開発は進められた．アーキテクトが手掛けた再構築ソフトの置き換え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．

Fig. 11 に，従来の仕様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の比較結果を示す．平均値，最大値共に有効コード行数で示される傾向に対して，複雑度が低く保守性が向上する結果が得られた．

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し，内部品質の内，保守性が確保できる結果が得られた．

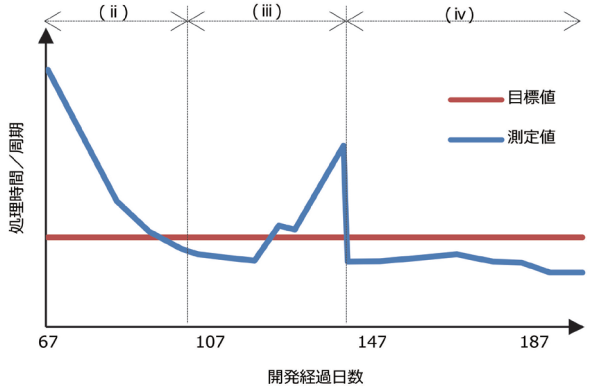

Fig. 8 Transition of CPU Process Times

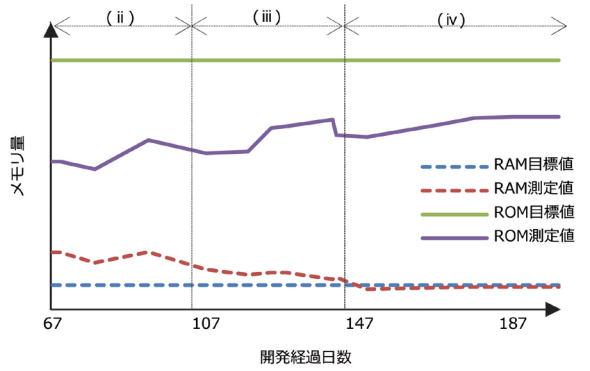

Fig. 9 Transition of Use of Memories

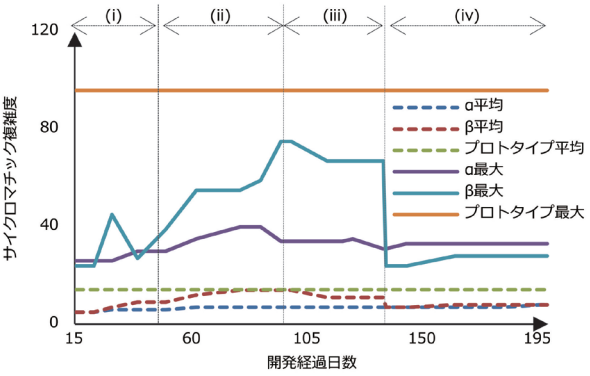

Fig. 10 Transition of Complexities with Sub-Components

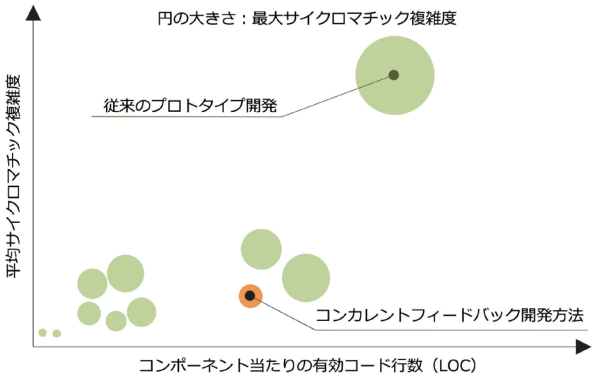

Fig. 11 Comparison of Cyclomatic Complexity

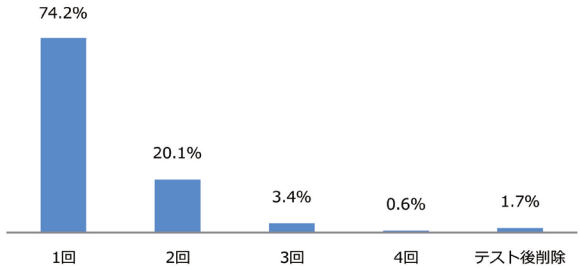

Fig. 12 The Running Number of Unit Tests

6.3.2 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，**Fig. 12** に示すように単体テストの実施回数を記録し評価した．

単体テストのやり直し回数として，最大 4 回実施した関数が存在したが，約 75% の関数は 1 回の実施で完了している．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．

7. 考察

7.1 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，進化的プロトタイピングに移行した効果が最も大きい．前述の通り，A ループにより E ループの活動のムダが軽減された上で，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い．

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 ヶ月を要している．開発終盤に仕様発行に向けてフィーチャの追加・変更が頻発したことが要因として挙げられる．P ループにおける変更管理など，まだ開発期間を短縮する改善の余地がある．

7.2 内部品質確保効果に対する考察

CF 開発方法により，プロトタイプ開発の早期から内部品質の確保活動を開始した．その結果，保守性，効率性の両品質指標の目標値を達成することができた．その

要因として，A ループによる活動効果と，P ループ担当者の設計スキル向上効果，測定値の定期提供による意識向上効果の 3 点が働いたものと推察される．

1 点目の活動効果について．開発当初，βの複雑度は大きく，効率性も劣化傾向にあった．βの再構築により両指標値の改善を果たしたが，複雑度と効率性を測定し，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得ない．また，開発後期の効率性の改善は，計測と詳細な分析が必要であり，フィーチャ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．

2 点目のスキル向上効果について．再構築後のβは，保守性，効率性共に安定している．これは，設計で品質が確保されたことが主要因である．さらに，改善後のソースコードが手本となり，開発担当者が設計と実装されたソースコードを基に開発を進めることで，学びが促されたものと推察される．

3 点目の意識向上について．効率性の監視開始までは，開発担当者における処理時間とメモリ量への意識は低く，開発担当者の設計・実装スキルに依存していた．指標値が定期的にレポートされることで，設計・実装結果のフィードバックが得られるようになり，開発担当者の意識が向上した．開発終盤では，フィーチャ開発において，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，内部品質の確保効果が促進したといえる．

7.3 アーキテクトの影響に対する考察

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も含めて，開発力が向上したと言える．

製品開発部門では，アーキテクトからソフト構造の不確実性や開発状況を得られ，仕様不明点も解消された．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．アーキテクトを開発ロケーションから離して配置すると，情報レベルが下がり，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掛かる．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．本開発方法を採用する場合，アーキテクトの割り当てが重要である．

8. おわりに

車載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的プロトタイピングに移行した．その上で，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試行した．その結果，製品開発の期間短縮に成功し，内部品質の指標目標値を達成した．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，ソフトウェアの内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．

本開発方法は新規開発向けに設計している．開発は次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．

参考文献

1) B. Zeiss, et al.: Applying the ISO 9126 Quality Model to Test Specifications, Proc. of Software Engineering 2007 LNI, Vol. 105, pp.231-242, 2007.
2) 林健吾 : 車載システム製品へのコンカレント開発の適用－新規開発における開発期間短縮へのアプローチ－ , ソフトウェア品質シンポジウム 2014, 2014
3) J. Zhang, and B. H. C. Cheng: Model-Based Development of Dynamically Adaptive Software, Proc. of ICSE '06, pp.371-380, 2006.
4) B. Schätz, et al.: Model-Based Development of Embedded Systems, Advances in Object-Oriented Information Systems, LNCS Vol. 2426, pp 298-311, 2002.

5) C. John: Evolutionary Systems Development: A Practical Guide to the Use of Prototyping within a Structured Systems Methodology, Plenum Press, 1991.
6) K. Beck: Extreme Programming Explained, Addison-Wesley, 2000.
7) R. C. Martin: Agile Software Development: Principles, Patterns, and Practices, Prentice Hall PTR, 2003.
8) K. Schwaber: Scrum Development Process, Business Object Design and Implementation, pp.117-134, Springer London, 1997.
9) H. H. Jo, H. R. Parsaei, and J. P. Wong: Concurrent Engineering: The Manufacturing Philosophy for the 90's, Computers & Industrial Engineering, Vol.21, Issues 1–4, pp.35–39, 1991.
10) D. E. Carter, and B. S. Baker: Concurrent Engineering, Addison-Wesley, 1992.
11) M. Aoyama: Beyond Software Factories: Concurrent-Development Process and an Evolution of Software Process Technology in Japan, Information and Software Technology, Vol.38, pp.133-143, 1996.
12) W. Sollecito, and J. K. Johnson: McLaughlin and Kaluzny's Continuous Quality Improvement in Health Care, Jones & Bartlett Pub, 2011.
13) E. Ries: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, 2011.
14) T. Moon, E. Kruzins, and G. Calbert: Analyzing the OODA Cycle, PHALANZ Online, Vol. 35, No. 2, pp. 9-13, 34-35, 2002.
15) N. Rozanski, and E. Woods: 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].
16) J. O. Coplien, and N. B. Harrison: Organizational Patterns of Agile Software Development, Prentice-Hall, 2004.
17) P. C. Pendharkar: A Probabilistic Model for Predicting Software Development Effort, Software Engineering, IEEE Transactions on, Vol.31, Issue 7, pp.615-624, 2005.

著者

林 健吾
はやし けんご

走行安全技術 4 部
超音波センサを利用した ECU ソフト量産開発に従事