車載機器開発へのインクリメンタル反復開発プロセスの適用

- 不確実性の早期低減による新規開発の品質確保 -

An Introduction of Incremental and Iterative Process to Automotive Equipment Development - Quality Assurance for New Product Development by Early Reduction of Uncertainty -

株式会社デンソー　熱エレクトロニクス開発部
DENSO CORPORATION　Thermal Systems Electronics R & D Dept.
○蛸島　昭之 1)
○Akiyuki TAKOSHIMA[1)]

Abstract This paper proposes to introduce Incremental and Iterative Development process to a brand new automotive ECU software development, aiming at reducing the amount of uncertainty hiding inside a project. First, we clarify the challenges in a new software development with conventional Waterfall Development model and examine the effectiveness of Incremental and Iterative Development model for a new software development. Then, we verified that Incremental and Iterative Development model has expected effectiveness for a new software development.

1. はじめに

車載製品向け組み込みソフトウェア開発では，既存のソフトウェアに対して改良や機能追加を行う「派生開発」が一般的である．しかし，近年急速に進んでいる自動車のエレクトロニクス化により，その状況に変化が現れ始めている．メカとエレクトロニクスを融合したメカトロニクス（機電一体）技術の発達により，従来製品からの派生開発ではない全くの「新規開発」の機会が増加すると予想される．

私が開発を行っているカーエアコンでも，コンプレッサの電動化にみられるように，従来メカ主体で行われていた制御の多くを電子制御に置き換える取り組みがすでに進んでいる．このような新規開発では，派生開発に比べて非常に多くの不確実性を抱えた状態でプロジェクトがスタートすることになる．プロジェクトマネジメントでは，この不確実性から発生する事象をリスクと定義している．すなわち，プロジェクトの初期でできるだけ多くの不確実性を取り除くことこそが，プロジェクトマネジメントにおけるリスク管理となる．しかし，従来の開発プロセスではプロジェクトの後半まで多くの不確実性が明らかにならないまま開発が進んでしまう．そこで，今回行った新規製品のソフトウェア開発では，不確実性をなるべく早期に低減するために，インクリメンタル反復開発(IID: Incremental and Iterative Development)[1]プロセスを採用した．

実際に IID プロセスを適用し開発を行い，想定通りの結果が得られることを確認したことで，今後の新規開発にも繰り返し適用可能な開発プロセスの基盤を確立することができた．そこで，本論文ではその成果を紹介する．なお，本論文では，読者に反復型開発手法の 1 つであるスクラムに関連する用語の知識があることを想定している．用語の定義については[2][3][4][5]などを参照して頂きたい．

1）株式会社デンソー　熱エレクトロニクス開発部
Thermal Systems Electronics R & D Dept. DENSO CORPORATION
〒448-8661 愛知県刈谷市昭和町 1-1　Tel: 0566-63-7674
1-1, Showa-cho, Kariya-shi, Aichi Japan　Tel: +81-566-63-7674

2.　市場変化と開発手法の陳腐化

近年，様々な要因により市場が変化し，これまでは最も多くのプロジェクトに適用されてきたウォーターフォール型の開発手法にも見直しが必要となってきている．市場変化の要因としては，自動車のエレクトロニクス化を背景とした車両メーカー要求水準の高度化，開発規模の拡大，開発期間の短期化，ユーザー嗜好の多様化による大量生産から少量多品種への転換などが挙げられる．また，市場変化のスピードそのものも年々速まっている．このように競争が激化する自動車部品市場で生き残っていくには，既存製品にはない新たな付加価値を持った製品を継続的に開発していくことが求められる．そのため，新規開発に適応した新たな開発手法を確立する必要がある．

2.1　市場の変化による不確実性の増加

新規開発と派生開発の最大の違いは，プロジェクト開始時点での不確実性の多さである．ソフトウェア開発における不確実性は，図 1が示すように開発フェーズを経るごとに減少することが知られている．[6] 新規開発プロジェクトの開始時では，見積りにして実に 16 倍もの差が発生する可能性がある．それに対して派生開発では，一旦すべての不確実性が収束した状態からの差分のみが不確実性を発生させる要因となるため不確実性の幅はずっと小さくなる．

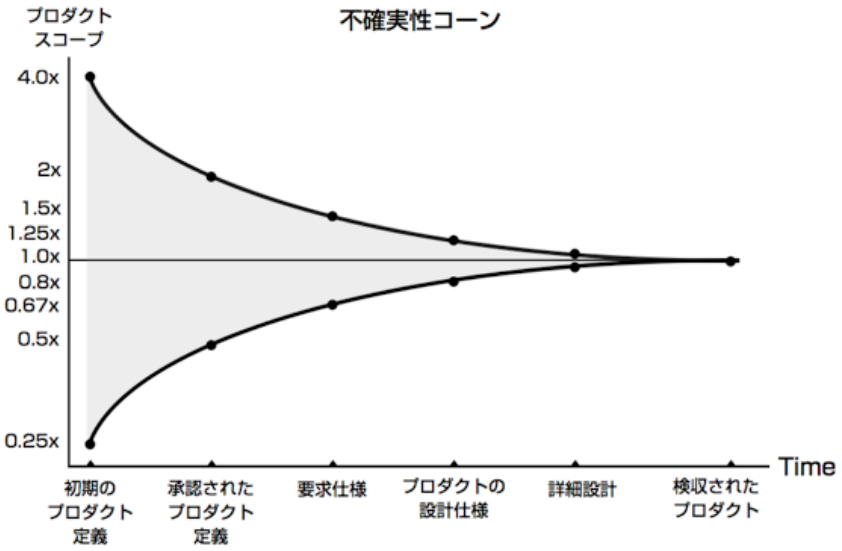

図 1 不確実性コーン

新規の組み込みソフトウェア開発でよく見られる不確実性を以下に挙げる．あらゆる新規開発がこれらすべての不確実性を抱えているわけではないし，派生開発であればまったく不確実性がないわけでもない．しかし，一般的に言って新規開発のほうが派生開発よりも多くの不確実性に対処しなければならないのは間違いない．

- 開発開始時に全ての要件が固まっていない
- ハードウェアとインテグレートして評価するまで仕様の妥当性・成立性を判断できない
- 新規のマイコンや IC が仕様通りの動作をするかわからない
- 新規のマイコンやコンパイラの性能がわからない

2.2　現行プロセスで新規開発を行うときの問題点

従来の開発プロセスでは，図 2に示すように，大きな車両評価イベント毎にV字プロセスを回すウォーターフォール型の開発を行っている．

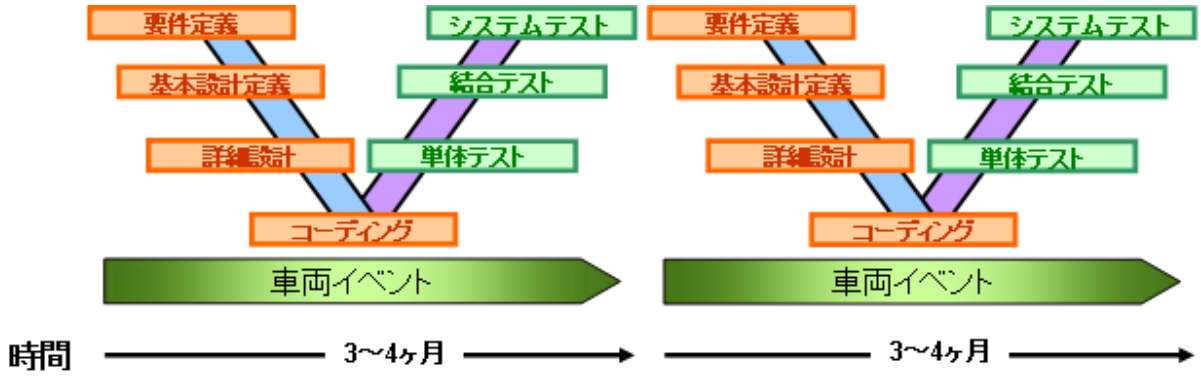

図 2 従来の開発プロセス

従来プロセスで新規開発を行う問題点として，フェーズ毎の不確実性減少カーブの特性がある．新規開発では 2.1 で挙げたようにハードウェアに起因する不確実性が多く存在する．そのような不確実性は，ターゲットボード上での実機動作確認を行って初めて解消することができる．そのため，図 3 フェーズ毎のリスク推移に示すように，V 字プロセス終盤のテストフェーズまでリスク低減できずにプロジェクトが進行してしまうことである．

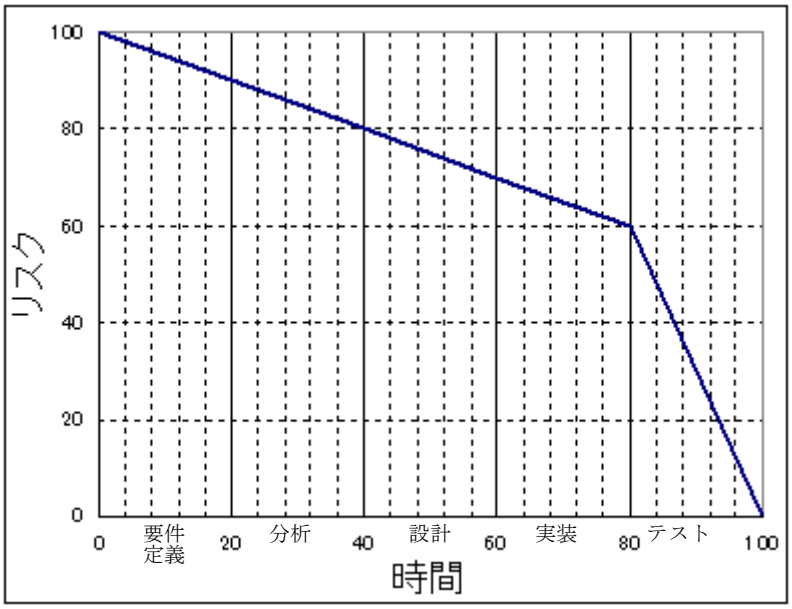

図 3 フェーズ毎のリスク推移

実機テストを実施して不確実性が解消された時点で得られる結果には２つの種類がある．一方は，想定と結果が一致し，リスクが顕在化しないケースである．もう一方は，想定と結果が一致せずにリスクが顕在化してしまうケースである．リスクが顕在化したときの影響として，それまで行ってきた分析・設計・実装・テストの再実施が必要となり大幅な手戻り作業が発生することで車両イベントにソフトウェアのリリースが間に合わなくなることが考えられる．逆に，納期を最優先して無理にリリースを行うと低品質なソフトウェアのせいで車両評価[1]に支障をきたしたり，技術的負債を抱えることで後々の開発ペース低下を招いたりといった問題を抱えることになる．さらに深刻な場合には，製品コンセプトそのものが成立せずに製品企画が中止となる可能性もある．そのような場合も，多額の投資を行う前に中止を決定するために，なるべく早期に問題が明らかになったほうが望ましい．

[1]車両に開発品を搭載して行う実車評価

2.3　課題のまとめ

以上の分析から，新規開発を行う場合の課題として以下の 3 つを挙げることができる．

課題 1　マイルストーンの開始時点ですべての要件が確定できていなくても，大きな後戻りをせずに進化する要件に追従できること（要件追従性）

課題 2　新規ハードウェアの動作を確認するために一部機能を実装した評価用ソフトウェアを段階的かつ頻繁にリリースできること（段階的リリース）

課題 3　不確実性の多くをマイルストーンの終盤まで残さないよう，なるべく早期にターゲット上でのテストを実施できること（不確実性の早期低減）

3.　開発プロセスの比較と選択

3.1　インクリメンタル反復開発プロセス

開発を進めながら要求の精度を高めていくための開発プロセスとしては反復(Iterative)型のプロセスがある．また，段階的なリリースを可能とする開発プロセスとしてはインクリメンタル(Incremental)型のプロセスがある．車載製品向け組み込みソフトウェアの新規開発では，2.3 で挙げたように「進化する要求への追従」と「段階的なリリース」の両方が必要となる．そこで，インクリメンタル型と反復型のハイブリッドであるインクリメンタル反復開発プロセスの採用を検討する．なお，以降の説明では インクリメンタル反復開発を IID(Incremental and Iterative Development)と表記する．

ウォーターフォール開発では，マイルストーンの中で要件定義，分析，設計，実装，テストの各フェーズを一度だけ行う前提で開発を進める．そのため，すべての要件定義が完了するまで分析以降のフェーズには進むことが出来ない．これに対して，IID ではマイルストーン内で要件定義，分析，実装，テストのサイクルを回しながら開発を進める．（図 4 参照） IID では，すべての要件を確定させる前からでも開発をスタートすることが可能なため課題 1（要件追従性）を解決することができる．

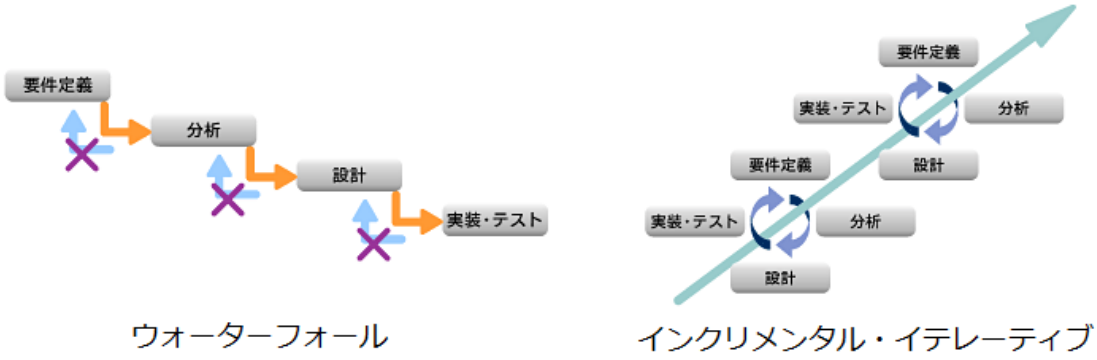

図 4 フェーズ戦略[7]

IID では，機能を 1 つずつ追加しながらインクリメンタルに開発を進めていくため，機能の優先順位を調整することができる．そのため，ハードウェアチームによる評価の開始までに評価対象の機能を実装することが可能なため課題 2（段階的リリース）を解決することができる．また，テストフェーズまで全て完了した状態で評価用ソフトをリリースするため，問題が起きた場合もハードウェアとソフトウェアの同時デバッグを行うという非効率でリスクの高い作業を避けることができる．

1 要件毎に要件定義，分析，設計，実装，テストのサイクルを回す IID では，ウォーターフォール開発のようにフェーズごとに作業を進めていくよりも早く不確実性を解消しプロジェクトのリスクを低減することができる．ウォーターフォールと IID で，5 つの要件を同じ期間をかけて開発したと仮定した場合のリスク減少カーブを図 5 に示した．このように，IID であれば課題 3（不確実性の早期低減）を解決することが可能となる．

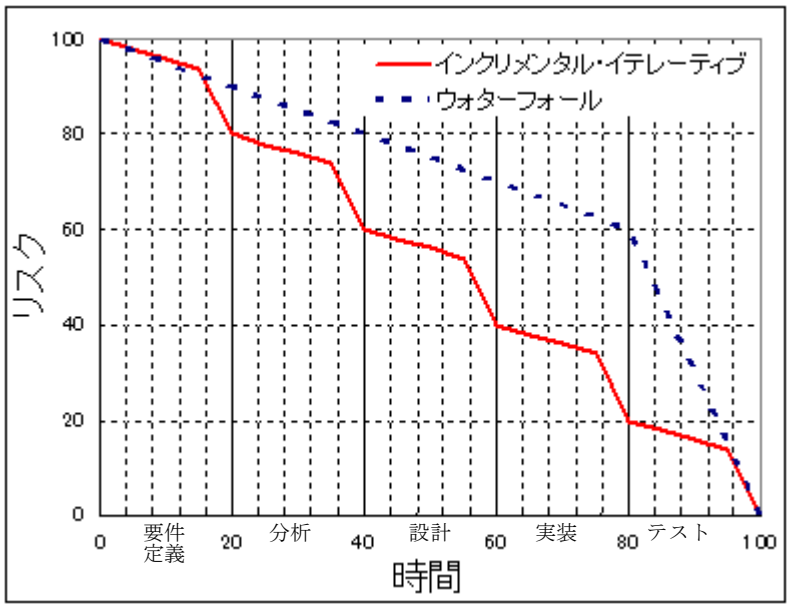

図 5 リスク減少カーブ

3.2 インクリメンタル反復開発を行う場合の課題

3.1 の考察から，IID を採用すれば従来プロセスで新規開発を行った場合に想定される 3 つの課題を解決できることがわかった．しかし，IID を採用する場合には，従来プロセスとは別の解決すべき課題がある．これらの課題をどのように解決するかについては 4 で述べる．

課題Ⅰ 機能追加を行うために既存機能のソースコードへの変更が発生するため，機能追加によって既存機能がデグレードしていないかを常時確認する必要がある（デグレード検出）

課題Ⅱ 複数の開発者が異なる機能の追加を同時に実施することになるため，従来通り 1 つのブランチ上のみで開発を行うと変更の競合が発生しビルドのブレークが多発する（ビルドブレーク防止）

4. インクリメンタル反復開発プロセスの実装

IID プロセスを実装するにあたっては，最も適用事例の多い[7]スクラムをベースにすることとした．各マイルストーン内でのアクティビティーの流れを図 6 に示す．

オリジナルのスクラムとの大きな違いは，プロダクトオーナーと呼ばれるプロダクトバックログの管理に責任を持つ専任者を置かず，ソフトウェアチーム，ハードウェアチーム，制御チームの 3 者による合議制を取ったところにある．また，オリジナルのスクラムでは，スクラムマスターと呼ばれるスクラムの理解と成立に責任を持つ専任者も置くことになっているが，この役割はソフトウェアチームのリーダーとサブリーダーがソフトウェア開発業務と兼務することとした．

(1) バックログ登録

制御チームとハードウェアチームが作成した仕様をソフトウェアチームがストーリーに分解しプロダクトバックログに登録する．

(2) リリースプラニング

ソフトウェアチーム，ハードウェアチーム，制御チームが協議して，プロダクトバックログから今回のマイルストーン（車両評価イベント）でリリースするソフトウェアに含めるアイテムのリスト(リリースバックログ)を作成する．

(3) スプリントプラニング

ソフトウェアチーム，ハードウェアチーム，制御チームが協議して，リリースバックログ

から今回の反復開発期間(スプリント)に含めるアイテムのリスト(スプリントバックログ)を作成する．リリースプラニング時点から追加，変更された要件はスプリントプラニングでリリースバックログへ反映する．

(4) スプリント

一定期間に区切られた開発サイクル．スプリントの中では，スプリントバックログに記載されたアイテムを優先度の高い順から実装していく．原則として，スプリント期間中のスプリントバックログへの作業の追加は行わない．スプリントバックログ上のアイテムが全て消化できなかった場合もスプリント期間の延長はせず，次のスプリントに持ち越す．

(5) リリース

構成管理システム上でタグ化などを行い，スプリントで機能を追加したソフトをリリース可能な状態にする．実際にリリースするかは，ハードウェアチーム，制御チームの評価日程による．

リリースされたソフトをハードウェアチームもしくは制御チームが評価した結果，所望のパフォーマンスが得られない，ハードウェアのバグが見つかるなどの要因でソフトウェアの変更が必要となった場合，仕様変更のための作業を新たなアイテムとしてプロダクトバックログに追加する．

(6) ふりかえり

スプリント期間内での活動を KPT などの手法を使ってふりかえり，次スプリント以降で生産性と品質を向上するための活動を計画する．

(7) スプリントを繰り返す

今回のスプリントがマイルストーン中の最終スプリントでなければ，(3) スプリントプラニングへ戻る．

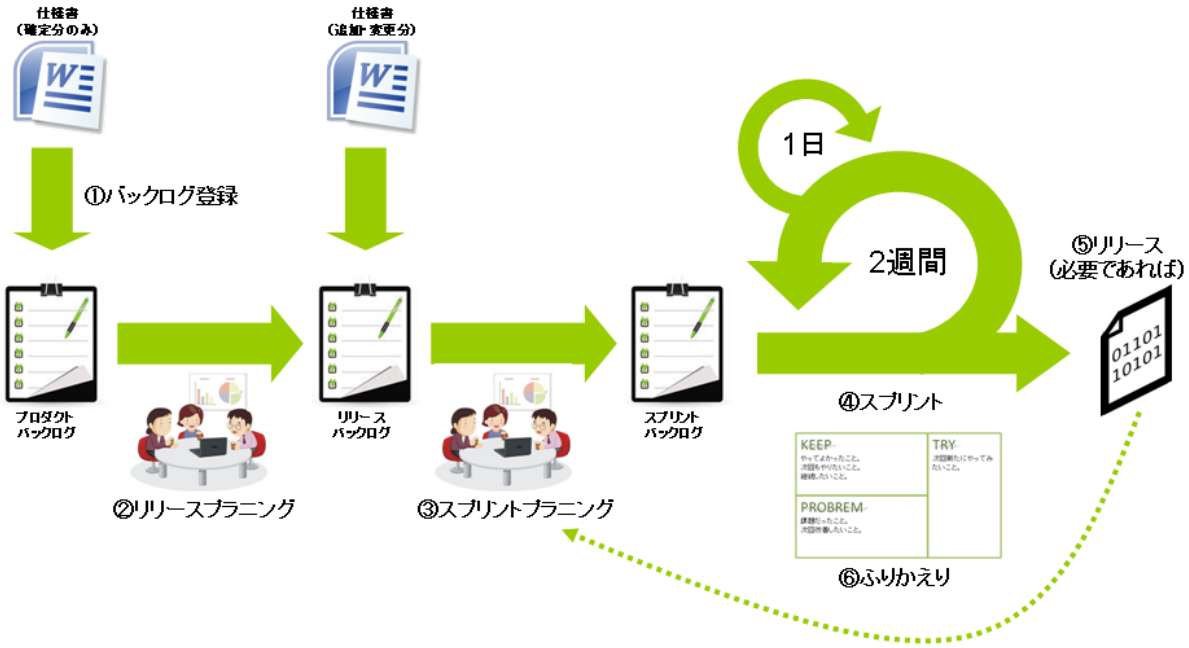

図 6 IID プロセスの実装

ここまで，どのように IID プロセスを実装するかを示した．しかし，これだけでは 3.2 で述べた 2 つの課題は解決できない．課題 I （デグレード検出）を解決するには，機能追加の度に回帰テストを実施すればよい．しかし，手動でテストを行っていては機能が追加される毎にテストにかかる工数が増大するため持続可能ではない[9]．そこで，ソースコードの実装はエクストリームプログラミング(XP ： eXtream Programming)のプラクティスの 1 つであるテスト駆動開発(TDD ：

Test-Driven Development)で行うこととした．TDD を行うことで，自動実行可能な回帰テストスイートが整備されるためテスト工数を一定以内に抑えつつ課題Ⅰ（デグレード検出）を解決することができる．TDD の詳細についてはここでは述べないが，組み込み環境における TDD については[10]が詳しい．

次に，課題Ⅱ（ビルドブレーク防止）を解決するための構成管理方針を検討する．複数の開発者が同一ブランチ上で同じファイルを同時に変更することで変更の競合が発生する．編集中はファイルにロックをかける悲観的ロック方式がもっとも確実な方法だが，複数の開発者が並行して作業をすることができなくなるため極端に生産性が低下してしまう．そこで，プロダクトバックログアイテム(追加する機能)ごとに新たなブランチを作成し，機能の実装とテストが完了した時点でメインブランチへマージする「フィーチャーブランチ戦略」を採用することとした．フィーチャーブランチ戦略を採用することで，メインブランチは常にビルド可能で，全回帰テストをパスする状態を維持することが可能となり課題Ⅱを解決することが可能となる．さらに，メインブランチを常に安定した状態に保つことで，ハードウェアチームや制御チームから要求があれば，スプリントの途中でも最新のソフトをいつでもリリースすることが可能となる．実際の開発では，マージのたびに発生する繰り返し作業の工数を低減するため，静的解析，単体･結合テスト，オブジェクトファイルのビルドなどを自動で行う継続的インテグレーション環境[11]を整えた．

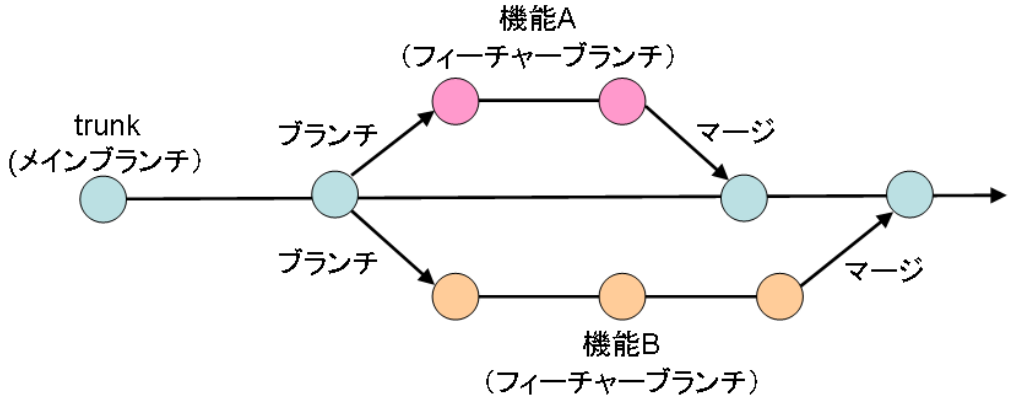

図 7 フィーチャーブランチ戦略

5. IID 試行結果の評価

開発プロセスに IID を採用したことで，各マイルストーンの初期で不確実性の多くを明らかにすることができたため，計 5 回のマイルストーンにおけるリリースを納期遅延なしで達成した．また，インクリメンタルに機能を追加することでプロジェクトに潜む不確実性を早期に明らかにできた例として以下のような成果があった．

プロジェクトに潜む不確実性を早期に明らかにできたことで

- マイルストーン最初のスプリントでリリースしたソフトウェアにより実機動作確認を行うことで，新規 IC の過電流保護に関する欠陥が見つかった
- 単機能ごとに実装を行っていくため，非機能要件(パフォーマンス，ROM/RAM 使用率)上のボトルネックが発生しても即座に検出することができた

インクリメンタルに機能を追加することで

- ソフトウェア，ハードウェア，制御仕様のコンカレント開発が可能となった
- 試作ソフトも最新の安定バージョンをベースに開発を行うことが可能となり，低品質なソフトウェアと未検証のハードウェアの組み合わせによるソフト･ハード同時デバッグを避けることができた

また，IID 採用により以下のような副次的効果を得られた．

- 毎スプリント終了のたびにふりかえりを行うことで，プロジェクト期間を通じた絶え間ない改善のフィードバックループが生まれた
- 機能の追加･変更を強く意識することで，疎結合・高凝集な設計が導かれた

6. おわりに

6.1 本研究のまとめ

従来プロセスで新規開発を行った場合に予想される問題点を挙げ，その解決策として IID プロセスを導入することを提案した．また，実際の新規開発に IID プロセスを適用し，想定通りの結果が得られることを確認したことで，今後の新規開発にも繰り返し適用可能な開発プロセスの基盤ができた．

6.2 今後の進め方

実際に IID プロセスで新規開発を行う中で以下の課題が見つかった．これらの課題に対する解決策を検討することで，開発プロセスの更なる改善を行っていく．

- プロダクトバックログアイテムが着手可能な状態になっているかの判断基準(Ready の定義)がないため，設計フェーズに入って仕様の検討不足が発覚し，次スプリント以降へ機能実装が先送りになる
- バックログアイテムの見積り精度が甘く，スプリント内で予定していた作業の全てを完了することができず，次スプリント以降へ機能実装が先送りになる
- バックログの優先順位付けを行う際に，システムの外部的な振る舞いを検証する要件を重視しすぎて，デバイスドライバ層の成熟が遅れる
- 新たな評価ソフト作成依頼による工数増加のリリーススケジュールに対する影響が可視化できていない

参考文献

[1] Craig Larman, Victor Basili, “Iterative and Incremental Development: A Brief History”, IEEE Computer, 2003

[2] Ken Schwaber, Jeff Sutherland, “The Scrum Guide“, http://www.scrumguides.org/, 2013

[3] Ken Schwaber, Mike Beedle 著, 「アジャイルソフトウェア開発スクラム」, ピアソン･エデュケーション, 2003

[4] Ken Schwaber 著, 「スクラム入門」, 日経 BP ソフトプレス, 2004

[5] 西村直人, 永瀬美穂, 吉羽龍太郎著,「SCRUM BOOT CAMP THE BOOK」, 翔泳社, 2013

[6] Steve McConnell 著, 「ラピッドデベロップメント」, アスキー, 1998

[7] 萩本順三, 初歩の UML 第 10 回 開発プロセスの上手な組み合わせ http://www.itmedia.co.jp/im/articles/0310/08/news001.html, 2003

[8] Publickey, 「過半数がスクラムを採用」アジャイル開発に関するアンケートが今年も公開 VersionOne http://www.publickey1.jp/blog/13/versionone.html, 2013

[9] James W. Grenning, “Agile Embedded Software Development”, ESC Boston, 2011

[10] James W. Grenning 著, 「テスト駆動開発による組み込みプログラミング」, オライリー･ジャパン, 2013

[11] David Farley, Jez Humble 著, 「継続的デリバリー」, アスキー・メディアワークス, 2012