アジャイルプラクティス導入による組込みシステム開発のプロセス改善

Process improvement of embedded system development by introducing agile practice

アンリツ株式会社
Anritsu Corporation
三宅 康宏
Yasuhiro Miyake

Abstract The "schedule delay" and "over budget" in development were issues that we had not been able to solve for many years. Previous attempts to solve this problem by introducing an agile process failed. The reason for this was that the understanding of processes and practices was inferior and there was no consideration of the characteristics of the products to be developed (being embedded system development) and team characteristics (members' skills).

In this time, based on this reflection, I adopted the approach to customize and incorporate only the necessary practices on the basis of the current process. As a result, compared with conventional development, we were able to reduce to 1/3 bug occurrence in system test and 2 manhours for debugging.

Although agile development is not common in embedded system development, we could confirm that we can enjoy the benefits of Agile development by incorporating it according to the characteristics of development.

1 はじめに

弊社はこれまでウォーターフォールモデルにより組み込みシステムの開発をおこなってきたが、プロジェクト後半での不具合の顕在化とそれによる日程遅延／コスト増加という問題をたびたび抱えてきた。その対策として我々チームが当初行ったことは、ウォーターフォールにおけるこの種の問題への対策として一般的な「ドキュメントの厳格化」と「レビューの強化」であった。しかしながら、この方法においても上記問題を解決するには至らなかった。

次に、ウォーターフォールではその構造上これらの問題を解決することは難しい[3]と考え、アジャイルへの移行を検討した。しかし、その試みは実践自体がうまくいかずプロジェクトが混乱したため途中で中断した。

今回は、前回の反省点を踏まえ新たなプローチでプロセスの改善を目指すこととした。前回のアジャイル導入における失敗の原因は、リーダーである自分自身が実践するプロセスやそれを構成するプラクティスの真の意味や目的を理解していなかったことにあった。今回は、現行のプロセスを捨てて新しいプロセスに乗り換えるのではなく、アジャイル等のプラクティスを取り入れな

アンリツ株式会社
Anritsu Corporation.

神奈川県厚木市恩名 5-1-1 Tel: 046-296-6683 e-mail:Yasu.Miyake@anritsu.com
5-1-1, Onna, Atsugi-shi, Kanagawa Japan
アンリツ株式会社計測事業グループ計測事業本部ｻｰﾋﾞｽｲﾝﾌﾗｽﾄﾗｸﾁｬｿﾘｭｰｼｮﾝ事業部商品開発部主任
Assistant Manager, Anritsu Corporation, Product Develoment Dept. Service Infrastracture Solutions Div. Measurement Bussiness Div. Measurement Business Group

【キーワード：】プロセス、アジャイル、組込みシステム

がら現行のプロセスを再設計するというアプローチをとった。これにより自分自身でコントロール可能なプロセスにすることができると考えた。

2　プロセス導入の失敗

前回、アジャイルプロセスを実践した際は XP やスクラムの本に書いてあることをその通りに実行していた。 結果としては、以下のような状況に陥り継続が不可能となった。

- プラクティスを実践する際に、どこまでやればよいのか、また何が正解なのかが分からない。また、うまくいかなくなったときに、どうすればよいのかわからない。
- プラクティスの実行に必要なスキルが分からない為、できないものをやろうとしてしまう。
- 必要のないプラクティスを実施してしまい日程を圧迫する。

また、現状のやり方が大きく変わることによるメンバーの混乱も大きかった。

3　今回のプロセス導入方針

今回は、前回の反省を踏まえ以下のような方針を立てた。

A) プロセスおよびプラクティスを理解する

プロセスおよびプラクティスの本質を深く理解する為には、自分でプロセスを設計することが最も効果があると考えた。第３者が作ったプロセスをいくら勉強しても理解の限界があるが、自分で設計すれば当然全てを理解できる。ただし、全てを独自に考案するのは効率が悪いので、有用なプラクティスは最大限取り入れる。

B) プロセスを必要以上に変えない

現状のプロセスをベースにしながら良い部分はそのまま残し、問題のある部分だけを新しい手法に置き換える。

4　現状プロセスの問題点分析

4.1　ビジネスに与える悪影響

必要な開発プロセスを明確にするため、現状抱えている問題の分析を行った。ビジネスに悪影響を与える問題は以下の３点であった。

A) 日程遅延

実装フェーズまでは予定通り進捗するが、結合から進捗が停滞しはじめシステムテストでバグの大量発生とデバッグ効率の低下からテストフェーズが長引き日程遅延を引き起こす。問題が顕在化し始めるのがプロジェクトの最終段階であり、有効な対策を打つには遅すぎる。

B) コストオーバー

一般に修正工数は後工程になればなるほど大きくなる[4]と言われているが、前述のとおり後工程で顕在化する不具合によってその対策コストが増大する。

C) メンテナンス効率の低下

システムテストが混乱するため設計が荒れ、それにより改造時のデグレード検証コストが増加する。コストが増大するだけでなく、機能追加／改造効率が下がり市場における競争力が低下する。

4.2 問題の原因分析

次に、上記問題の原因がどの工程においてどのように発生しているかを分析した。図 1 に現行プロセスのフローを示す。ウォーターフォールそのものであるが、まずプロジェクト全体の計画を全体計画で行い、次に仕様を作成する。完成した仕様をもとに実装以降の見積を更新し、開発費に収まるように仕様を調整する。その後実装、ソフト結合、ハード結合、テストの工程を経てリリースとなる。

図 1 現状のプロセス

表 1 に各工程で露呈する主な問題とそれが埋めこまれる工程を示す。

表 1 現行プロセスの各工程で露呈する問題とそれが埋め込まれる工程

工程	露呈する問題	問題が埋めこまれる工程	番号
全体計画更新	仕様に基づいた実装見積が計画を超えた場合、仕様見直し作業が発生する。	仕様	1
実装	実装コストを無視した仕様や、仕様漏れが見つかり仕様見直し作業が発生する。	仕様	2
結合	担当者間で認識違いによるインターフェースミスや実装漏れ作業が発覚する。	実装	3
	ソフト単体の不具合によりハード／ソフトの不具合原因切り分け作業に時間が掛かる。	実装	4
	ハードウェアの仕様変更、日程遅延に対しソフトウェア開発が柔軟に対応できない。	結合	5
テスト	修正と差戻しを繰り返しバグが収束しない。実装者の仕様理解が不足している。	仕様	6
	実装時に残作業が増加しているにも関わらず見積の更新がされず、日程のプレッシャーから実装／テストの質を薄めた実装が不具合として露呈する。	実装	7
	実装フェーズで仕様の理解不足による間違った実装が不具合として露呈する。	仕様 実装	8

	使ってみたら使いにくい､イメージと違う等で仕様見直し作業が発生する。	仕様 実装	9
	特定の人にバグが集中しクリティカルパスとなる。	実装	10
メンテナンス	設計が荒れている為、改造の度にデグレ検証コストが発生	テスト	11

5 番以外は仕様および実装といった前工程で埋め込まれた問題であり、これが後工程で不具合として顕在化し日程遅延をもたらしている。
5 番は組み込みシステム特有の問題であり、ソフトウェアの開発に与えるインパクトは大きい。

5　プロセスの設計思想

次にプロセスを設計するにあたり、前述の問題を解決するために新規採用するプラクティスと現行プロセスで残すものを決定した。

5.1　新規採用するプラクティス

表 1 の問題に対する解決策とそれを実現するためのプラクティスを表 2 に示す（番号列は表 1 の番号列と対応している)。

表 2 現行プロセスの問題点を解決するためのプラクティス

番号	解決策	プラクティス
1	定期的に残作業の見積を更新する	イテレーション開発、ベロシティ計測
2	仕様と実装を同時期に行い、細かい単位で工程移行する。	イテレーション開発
3	コミュニケーションを密にする。 頻繁に結合する。	スクラム会議、継続的インテグレーション
4	動く実物で検証を行う。	イテレーション開発､テスト駆動※1
5	優先度の組み換えや機能の取捨選択を可能にする。	イテレーション開発､テスト駆動※1
	ハードウェアに依存しない実装にする。	ハードウェアシミュレータ※2
6	全員で仕様を作成し全員の仕様理解を深める。	全員参加
7	動く実物で検証を行う。	イテレーション開発､テスト駆動※1
	定期的に残作業の見積を更新する	ベロシティ計測
8	全員で仕様を作成し全員の仕様理解を深める。	全員参加
	動く実物で検証を行う。	イテレーション開発､テスト駆動※1
9	動く実物で仕様のレビューを行う。	イテレーション開発
10	担当を固定せず､誰でもどこでも作り修正できるようにする。	ソース共有
11	ソースをリファクタリングする。テストを自動化する。	テスト駆動、リファクタリング

※1 テスト駆動はイテレーション開発には、テストの重複を避けるために必須である。

※2 これはアジャイルではなくモデルベース開発を参考にしている。

5.2　現行プロセスから継続採用するプラクティス

現行のプロセスにおいて良い結果が得られているものはそのまま残すべきと考えた。

A)　仕様策定後の全体計画

仕様策定後に全体の計画を立てることでゴール明確になる。我々の開発では要求に対する変更はそれほど入らない為、通常のアジャイルのように仕様策定を全てのイテレーションに分散させるよりも、前もって確定させ方が良いと判断した。これはエンタープライズ系開発におけるハイブリッドアジャイル [1]と同様のアプローチである。

B)　見積手法

WBS での時間による見積手法は数値の根拠が具体的であり、確度の高い見積を出せていたのでこれは残すこととした。スクラムの見積ポーカーやポイント制より現行の手法の方が良いと判断した。

5.3　プラクティス採用のための工夫

新たに採用するプラクティスにおいて、そのままでは効果的に実施することができないと思われるものがあったので、以下のように対策を行った。

A)　イテレーション開発

対象とする開発は組み込みシステムであるが、ハードウェア（FPGA,プリント板など）との結合ができるようになるのはプロジェクトの後半となる。それまでは、ソフトウェアは先行して開発を進めることになる。イテレーション開発では“動くソフトウェア”でテストを行う必要がある為、ハードウェアの動作をシミュレートするものが必要となる。従来の開発においてもユニットテストの為にテストスタブを用意していたが、それでは動くソフトウェアでシステムテスト相当のテストを行うには不十分であった。そこで今回は、デジタル信号処理のツール（SCILAB）を組み込み、自由かつ簡単に本物に近いデータを生成できるようなツールを作成した。そのツールで作成したシミュレーションデータを FPGA が出力するデータフォーマットに成形し、ソフトウェアの最下層レイヤであるドライバ（正確にはユーザ空間のドライバのラッパーレイヤ）から read できるようにした。このアプローチはモデルベース開発を参考にした。

B)　テスト駆動開発

イテレーション開発においては重複した回帰テストを避けるためにテストの自動化は必須となる。テスト駆動の目的は回帰テストの自動化であるが、これはアジャイルのプラクティスの中でも特に効果的なものの１つであり、難易度も高い[1]。テスト駆動開発には先にテストコードを書くことでアーキテクチャをテストしやすい設計にすることができるという効果がある[2]。テストしやすい設計は非常に重要であり、テスト自動化の実現だけでなくメンテナンスを効率的におこなうために必要である。しかしながら、テストコードを先に書けば誰でもこのような設計ができるわけではなく、技術的な前提としてオブジェクト指向の知識と設計スキルが要求される。弊社はメ

ーカであるため組み込み出身のソフトウェア技術者が多くオブジェクト指向設計が得意とは言えず、実際に前回のアジャイルのチャレンジにおいてもこの点において失敗していた。そこで今回は xUnit を用いた一般的なテスト駆動開発を行うのではなく、テスト自動化のためのフレームワークをソフトウェアアーキテクチャに組み込むことにした。プログラマがそのフレームワークに従い実装することで、必然的にテストしやすい設計となるようなものを用意した。
このフレームワークは、GUI の操作をスクリプトとして出力し、回帰テスト時はそれを実行させる機能を持っており、テストコードを 1 から記述する必要が無いのでテスト工数短縮にもなる。Presentation Model 型 MVC で設計したアーキテクチャにおいて、View から Presentation Model へのデータの流れをキャプチャすることで本機能を実現している。本手法により効率的のテスト自動化が可能となるがカバレッジの計測ができないことが今後の課題である。

6　プロセス詳細

6.1　全体の流れ

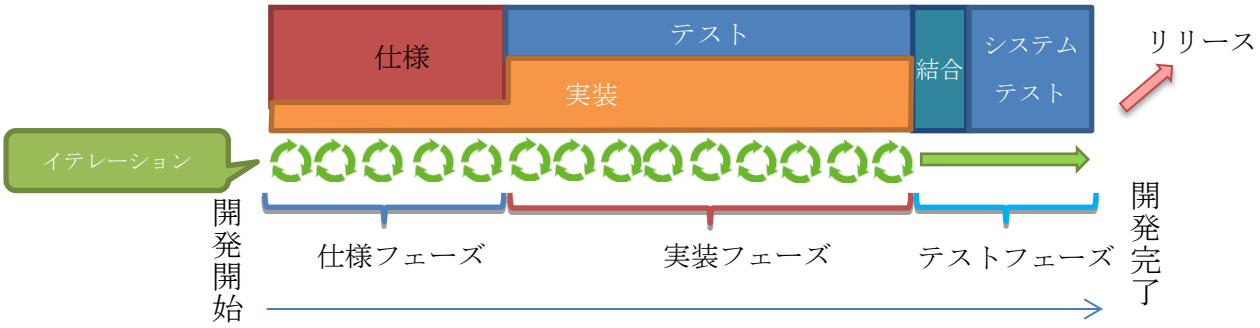

図 2 プロセスの流れ

実施したプロセスの全体の流れは大きく仕様フェーズ、実装フェーズ、テストフェーズの 3 工程に分かれる。この流れ自体はウォーターフォールと同様であり正式リリースはシステムテスト後の 1 回である。また、このプロセスはイテレーションの区切りのタイミングでメンバーの声や反応に基づき随時軌道修正を行った。

6.2　仕様フェーズ

本フェーズの目的は、仕様を確定しプロジェクトの全体像（コスト、日程）を明確にすることである。ただし、仕様策定のみを行うと、各種手戻りが生じるためこのフェーズにおいても設計・実装も同時に行い 2 週間のタイムボックスでイテレーション開発を行った。あくまで、ここでは全体像把握が目的なので設計・実装は完全には行わず、表示や動作のレビューができる程度にとどめた。
メンバー全員で手分けしてソフトを実装し、それを動かしながらレビューを行い仕様およびソースにフィードバックを掛けた。仕様書に記載する内容の粒度としては従来のウォーターフォールと同様に詳細に記述した。従来と異なる点は、仕様を書いてから実装するというよりも動作させレビューした結果をドキュメントに残すという点である。

イテレーション完了毎に作成した仕様に対する実装工数を見積り、その進捗をモニタしながら全体の見積工数が開発費を超えないよう仕様を調整した。
このフェーズの終了時に各仕様のプライオリティを決定し、実装フェーズへ移行した。

6.3 実装フェーズ

本フェーズの目的はハードウェアが上がってくるまでにソフトの実装をテストも含め完了させることである。2 週間のイテレーションで優先度の高い機能から順に設計、実装、テストを行った。
実装チームとテスト仕様担当に分かれ、実装チームは仕様書を元に実装を進め、テスト仕様担当者はテスト仕様書を作成した。従来の開発においては、テスト仕様書はシステムテスト直前に作成していたが、イテレーション開発ではイテレーション内でシステムテスト相当のテストが必要になるためこのようにした。テスト仕様作成時に仕様のミスや漏れが見つかることも多いが、それを直ぐに実装チームへフィードバックを掛けられるという効果も得られた。
テスト仕様担当者は毎朝のスクラム会議でテスト内容を実装チームへインプットし、実装チームがテスト内容を意識しながら実装できるようにした。ソースは毎日コミットし、継続的インテグレーションを行った。また、実装の担当部分は、基本的に高い技術やノウハウが必要な部分以外は固定させず、誰でもどこでも実装できるようにしたことにより、個人の遅れをチーム全体でカバーできるようになった。
イテレーションの最後には全員でテスト仕様に基づきテストを実施した。従来の単体テストと異なり第 3 者の目線で動くものを評価できるため効果的なテストを実施することができた。
イテレーション完了後は残作業の見積もりを全員で更新した。残作業が予定した見積を超えることが分かった場合は、即座に後のイテレーションで実装する機能の仕様を調整やリソースの調整を行った。

6.4 結合およびシステムテスト

これ以降は従来と同様である。ハードソフト結合は組み込みシステム開発においては最もリスクのある工程である。今回は、シミュレータ上でソフトウェアは十分テストされていたので従来と比較するとハードソフト結合はスムーズに進み、その工数は 1/2 に削減できた。
テストが自動化されている機能に関しては、本来システムテストでのテストが不要であるが、テスト自動化は初めての試みであった為、念のためシステムテストでは自動化された部分も含めハード上で再度テストを行った。当然であるが、自動テスト済みの部分に関するバグは発生しなかったので次回以降はこれらのテストは省略する予定である。

7 結果

イテレーション開発によりその時点の進捗を正確に把握でき、早めに効果的な対策を打つことができた。それにより余裕をもって日程を守ることができた。
システムテストでの発見バグ件数を 1/3 にすることができた。自動化したテストを省略すればテスト工数の削減が可能である。
手戻り工数に関しては過去のデータが無いため定量評価はできないが、今回の手戻り工数は 20H

程度であり過去の開発と比較すると確実に減っているといえる。

8　考察

プロセスを自分で設計し中身を深く理解したことにより、その時々における最適な実施方法を選択できたことが成功につながった。実際にプロジェクトを遂行する過程で想定と違うことがしばしば起こるが、その際にプロセスおよびプラクティスの目的を深く理解していれば、それらに修正を加えたり代替案で対応したりすることが可能となる。

アジャイルはユーザの要求変更に対して柔軟なプロセスであると認識されているが、ユーザの要求変更だけでなくすべての変更に対して柔軟なプロセスである。ハードウェアの仕様変更・日程遅延、既存製品の緊急対応や解決に時間のかかるバグの発生も当初計画に対する変更であり、これらに対しても有効である。これはイテレーション内で実装する部分（ストーリー）が単独で完結しており他機能への依存度が低いため、イテレーション単位またはストーリー単位で自由に優先度の組み換えや取捨選択が可能だからである。

9　結論

プラクティスを深く理解したうえでそれを採用することで確実にその効果を得ることができる。

組み込みシステム開発においてもアジャイルのプラクティスは有効である。

本論文で説明した具体的手法はあくまで一例であり、他の開発においてそのまま有効であるとは限らない。重要なのは開発対象製品の特性（新規性、コスト、納期、品質等）やチーム特性（人、文化、スキル、経験等）を考慮して、プロセスを最適化することである。またその最適化はプロセス設計時だけでなく、開発中もメンバーの声や反応を元に行う必要があるがイテレーション開発であればその区切りで軌道修正が可能である。

10　今後

今回の報告では主にアジャイルのプラクティスについて述べたが実際に運用しているプロセスでは開発対象に最適化する為に XDDP や独自のプラクティスも取り入れている。

今後、クラウド、IoT、AI などへの対応により組み込みシステム開発もますます多様化していくことが予想される。そのような状況において必要になるのは、現在の開発プロセスのような開発手順を定義したフレームワークではなく、開発対象やチームの特性に合わせアダプティブにプロセスを設計するためのフレームワークであると考えている。今後、そのような手法を考案し提案していきたい。

参考文献

[1]英繁雄, 奈加健次, 平岡嗣晃, 前川祐介, 長瀬嘉秀「ハイブリッドアジャイルの実践」,リックテレコム,2013

[2]ケント ベック, シンシア アンドレス「エクストリームプログラミング」,オーム社, 2015 年

[3]ジェフ サザーランド「スクラム」,早川書房,2015

[4] 奈良隆正「ソフトウェア品費保証の方法論、技法、その変遷」JASPIC SPIJapan2009