Enterprise Agile Processing
－Scrum+XPの中規模開発への適用とその考察－

第１ソリューション本部
ユビキタス・PCMソリューション部
第１ソリューション室
シニアプロジェクトスペシャリスト

七野　修一

Shuichi　Shichino
shuichi-shichino@exa-corp.co.jp

第１ソリューション本部
ユビキタス・PCMソリューション部
第１ソリューション室
ITエンジニア

大橋　絢子

Ayako　Ohashi
ayako-ohashi@exa-corp.co.jp

第１ソリューション本部
ユビキタス・PCMソリューション部
第１ソリューション室
ITスペシャリスト

村木　信也

Shinya　Muraki
shinya-muraki@exa-corp.co.jp

第１ソリューション本部
ユビキタス・PCMソリューション部
第１ソリューション室
ITスペシャリスト

和田　宗靖

Muneyasu　Wada
muneyasu-wada@exa-corp.co.jp

第１ソリューション本部
ユビキタス・PCMソリューション部
第１ソリューション室
シニアITスペシャリスト

工藤　大輔

Daisuke　Kudo
daisuke-kudo@exa-corp.co.jp

近年、要求の変化に強い開発手法としてAgile開発プロセスの適用事例が増えつつある。一方、Agile開発プロセスは、要件の決定と開発を同時に進行させることによる要求コントロールの失敗、開発アーキテクチャの破綻、開発者間のコンフリクトを引き起こしやすい。これらは、プロジェクトに関わる人数が多い、開発規模が大きなプロジェクトほど起こりやすい。筆者らは、比較的規模が大きなプロジェクトに対し、開発管理プロセスにScrum®、開発プロセスにXPと呼ばれるAgile開発プロセスを適用し、高いお客様満足度を得た。本稿では、この適用事例を紹介する。

1. はじめに

現在日本国内で主に実施されている「ウォーターフォール開発」を利用したプロジェクトの失敗原因として、「仕様決定が遅い」、「仕様変更が多い」、「要件定義があいまいなまま次フェーズを実施」といった要件に関する事由が常に上位に挙がっている。これはウォーターフォール開発が、要件変更時には前工程の作業のやり直しが必要であるという本質的な性質を持っているためである。しかしながら、要件変更がなくなるケースは稀である。

一方、要求をコントロールしつつ、発注者の要求変化に逐次対応する「Agile（アジャイル）開発」を利用したプロジェクトが増えつつあるが、次の３つの問題でプロジェクトを破綻させることが多い。

１つ目の問題は、要求コントロールの失敗である。その結果、不要機能の開発や大規模な仕様変更を招くケースが多い。また、Agile開発では要求コントロールと仕様確認をリアルタイムに行うために、発注側と開発側が一体の組織となって開発することを前提としているが、発注側は実業務が多忙なため、そうならないケースが多い。そのため、要求コントロールが難しく、スコープクリープが発生しやすくなっている。

２つ目の問題は、開発アーキテクチャの破綻である。Agile開発は仕様変更受入を前提としているため、アーキテクチャがお客様の要求の追加、変更に追随できず、リファクタリングと呼べない書き直し作業が増大することがある。その結果、変更の影響判断ミスや修正漏れが発生しデスマーチにつながりやすい。

３つ目の問題は、開発チームにコンフリクトが発生しやすいことである。設計書や開発標準のない多数の要員を抱える開発プロジェクトにおいて、ミーティング主体の情報伝達は、情報の到達時間のずれや誤認識による問題が発生しやすい。その結果、このコンフリクトによる生産性の低下や、さらなるコンフリクトを招きやすい。

これら３つの問題により、一般的なAgile開発の成功事例としては、開発要員が４,５人で、発注者と開発者が密接する小規模なプロジェクトや要求コントロールが容易な社内開発の事例が多い。

そこで筆者らは、開発管理プロセスにScrum®[4)8)]、開発プロセスにXP（Extreme Programming）[3)]を組み合わせ（Scrum+XP）[18)]、実プロジェクトにカスタマイズしつつ適用することで、比較的規模が大きく、お客様が開発サイトにいない状態においても、この３つの問題に対処可能な開発アーキテクチャ、開発体制、開発プロセスの実践方法などについて様々な知見を得た。

本論文では、このAgile開発手法を「Enterprise Agile Processing（以下、EAP）」と呼び、このEAPの適用事例について紹介する。また、この知見に基づく３つの問題の対策を「プロアクティブな要求コントロール」、「EAP開発アーキテクチャ」、「自律したチーム」と呼び、それぞれについて説明する。

2. 一般的なAgile開発の概要

本章では、一般的なAgile開発[1)2)5)6)7)10)11)12)]とAgile開発手法であるScrum、XPについて述べる。本章以降、開発管理プロセスの総称をScrumと呼ぶ。

2.1. Agile開発

一般的にAgile開発とは、迅速かつ適応的にソフトウェア開発を行う軽量プロセスの総称のことである。要求の追加、変更といったような変化を受け入れることを１つの目標としており、変化に対し機敏に適応することで創造性が高まり、より迅速に顧客に価値を提供することができるというコンセプトを持っている。この変化に機敏に対応するために、「短いサイクルでの頻繁な納品」や必要なものだけを実装するという考えに基づく「シンプルな設計」をコンセプトとしている。

また、この軽量プロセスの成功事例の共通点をまとめたものとして「Manifesto for Agile Software Development」[1)]が広く知られている。

2.2. Scrum開発管理プロセス

ScrumはAgileのプロジェクト管理プロセスの１つであり、その名前はラグビーのスクラムに由来している。すなわち、文字通り、発注者と開発チームがスクラムのように一体となり徐々に製品を完成に近づける作業を行うことを意味している。

Scrumの開発管理は、一般的に以下の手順で行う。

(1)　製品に対する要求事項(フィーチャ、ストーリー）を記述したプロダクトバックログを作成する。

(2)　スプリント（イテレーション）の開発期間（タイム

ボックス）を決定する。期間は２～４週間の間で固定とする。

(3)　スプリントバックログを作成し、開発チームに割り当てる。開発チームは６～８人程度がよいとされる。

(4)　スプリント（イテレーション）を実施し、製品を作成する。

(5)　製品をプロダクトオーナーが受領し、レビューする。

2.2.1. Scrum内の体制

Scrumでは、以下の体制を構築しプロジェクトを進行する。

(1) プロダクトオーナー

プロダクトバックログに優先順位を付けることができる唯一の役割であり、お客様がこの役割を果たす。一般的なScrumでは、プロダクトオーナーには、製品ビジョンの伝達能力や、プロジェクトの制約から発生した課題、問題に対する意思決定能力が求められる。理想的には、お客様内における意思決定権限のあるキーマンであることが望ましい。

(2) スクラムマスター

スクラムチーム内のスクラムプロセスの実装と管理、およびスクラムチーム外部との作業（要件定義など）を主に行う。スクラムマスターにはファシリテーション能力や、プロセス改善能力が求められる。理想的には、リーダー経験豊富な人物が望ましい。

(3) スクラムチーム

スクラムマスターを含むスクラムを実践し、すべての開発を行う少人数のチームである。

本プロジェクトではこれらの体制に加え、要求コントロールを専門に行うスクラムマネジャーを新設した。スクラムマネジャーの詳細は４章で述べる。

2.2.2. プロジェクトの運営

Scrumは、２つのバックログと３つのミーティングを中心にプロジェクトを運営する。各工程の計画書、管理プロセスの規定等の開発標準を多数用いるウォーターフォール開発に比べ、管理手法が軽量となっている。

(1) バックログ

① プロダクトバックログ

② スプリントバックログ

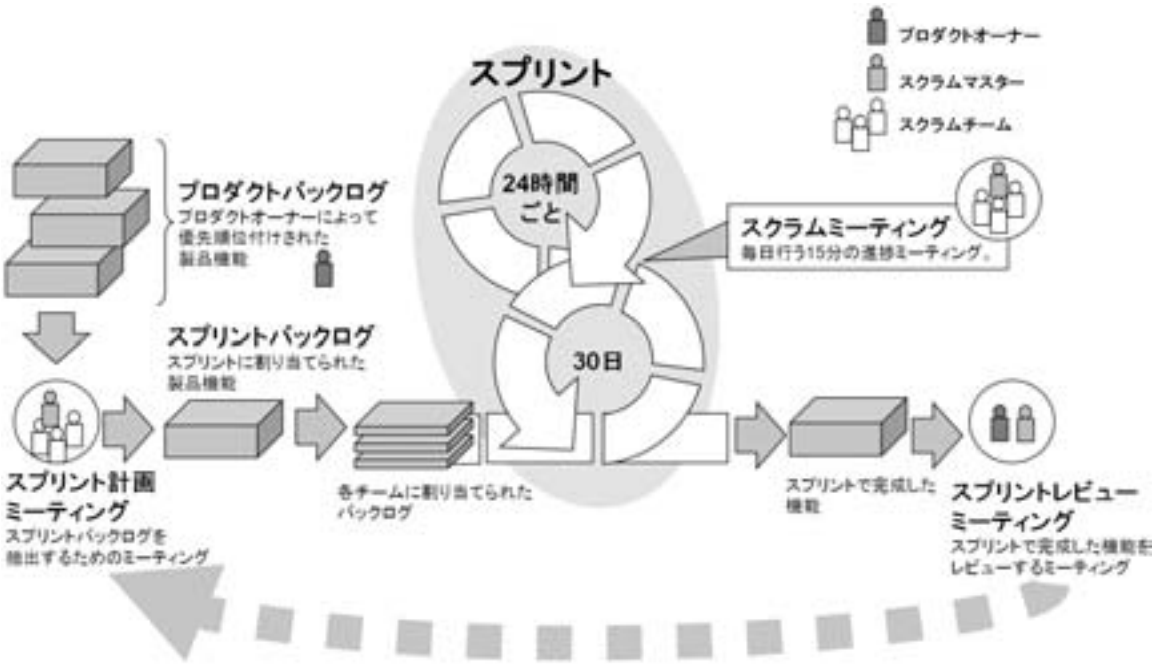

図１　Scrumの概要

(2) ミーティング
① スプリント計画ミーティング
② スクラムミーティング
③ スプリントレビューミーティング

プロダクトバックログは、プロダクトオーナーによって優先順位付けされた製品に対する要求事項を記述したものであり、要求コントロールのベースラインとなる。

プロダクトバックログが完成後、スプリント計画ミーティングを実施する。「スプリント計画ミーティング」は、スクラムマスターとスクラムチームが次スプリントで開発する要件をプロダクトバックログから抽出するためのミーティングである。このミーティングで抽出されたバックログを「スプリントバックログ」と呼び、スプリントのスコープとなる。これを基に開発側は「スプリント」と呼ばれる製品開発プロセスを行う。

スプリントでは、スプリントバックログにある機能を開発する。Scrumの特徴として、スプリント中は毎日スクラムミーティングを実施する。「スクラムミーティング」とは、スクラムチームによって毎日15分程度実施される進捗報告ミーティングであり、進捗と問題点を確認する。

スプリント終了後、スプリントレビューミーティングを実施する。「スプリントレビューミーティング」は、スプリントで開発した製品に対し追加の要求があるか、あるいは変更するかをプロダクトオーナーとスクラムマスターが議論するミーティングである。このミーティングの内容を基に次のスプリント計画ミーティングが実施され、次スプリントバックログを決定する。

このように、Scrumは①スプリント計画ミーティング、②スプリント、③スプリントレビューミーティングの３つのプロセスを１つのサイクルとして繰り返すことで、製品を開発する。

2.3. XP（Extreme Programming）開発プロセス

XPは４つの価値と５つの原則、そして12のプラクティスをベースとしたソフトウェア開発プロセスである[3)]。

以下にXPの価値を記述する。
(1) 価値（XPの基礎となる考え方）
① コミュニケーション(Communication)
② シンプルさ(Simplicity)
③ フィードバック(Feedback)
④ 勇気(Courage)

以下にXPの原則を記述する。
(2) 原則（XPの基本原則）
① 瞬時のフィードバック
② シンプルの採用
③ インクリメンタルな変更
④ 変化を取り入れる
⑤ 質の高い作業

以下にXPの12のプラクティスを記述する。
(3) プラクティス（XPの基礎となる考え方）
① 計画ゲーム(The Planning Game)
ビジネス優先度と技術的見積もりにより次回リリースの範囲を早急に決める。現実が計画と変わったら、計画を更新する。
② 小規模リリース(Small Releases)
シンプルなシステムを早急に生産する、その後、非常に短いサイクルでバージョンアップを行う。
③ 比喩(Metaphor)
どのように全体のシステムが機能するかを示すシンプルな喩え話(メタファー)をメンバーが共有することですべての開発を導く。
④シンプルデザイン(Simple Design)
システムはできる限りシンプルに設計する。また、現在必要なことだけを行う単純なコードを書くことに徹する。
⑤テスティング(Testing)
プログラマは継続的にユニットテストを書き、動作しつづけるよう保守を行う。顧客は、機能ごとに受け入れテストを実施する。
⑥リファクタリング(Refactoring)
システムの挙動を変えることなく、重複コードを共通化し、煩雑な記述を除去あるいは単純化し、将来の仕様変更に対し柔軟に対応できるよう再構成する。
⑦ペアプログラミング(Pair Programming)
すべてのコードを２人のプログラマが１台のマシンを使用し作成する。
⑧共同所有権(Collective Ownership)

誰もが、すべてのコードを修正できるようにする。

⑨継続的インテグレーション（Continuous Integration）

１日に数回、実装者が作成したコードを結合し、成果物の作成とテストを繰り返す。また、テストは100%成功するようテストケースとビルドを維持する。

⑩週40時間（40-Hour Week）

週40時間以上仕事をしてはいけない。

⑪オンサイト顧客（On-Site Customer）

実際のユーザをチームに加え、常に質問に答えられるようにする。

⑫コーディング標準（Coding Standards）

プログラマは、コーディング標準に従う。

それぞれ個々の詳細な説明は省くが、筆者らは、XP開発プロセスをすべて採用することはやめ、プロジェクトの実情に合わせカスタマイズし使用することとした。なお本論文では、XP第１版をベースに説明したが、現在第２版XP[12)20)]がリリースされている。

3. プロジェクトの概要とAgile開発適用時の課題

本章ではプロジェクトの概要と、このプロジェクトにAgile開発を適用した場合の課題について述べる。

3.1. プロジェクトの概要

本論文で紹介するプロジェクトは、Webを利用した新規事業を立ち上げるプロジェクトであった。開発当初は業務要件も明確でなく、Web分野に関するお客様の知見も少なかったため、ウォーターフォール開発では要件定義とその後の要求コントロールも困難であることが予想された。そのためお客様と合意の上、Agile開発を採用した。

プロジェクト概要を表１に、プロジェクトのスケジュールを図２に示す。

表１　プロジェクトの概要

プロジェクト内容	Webアプリケーション開発
エクサ担当範囲	プロダクトバックログ作成、設計、開発、テスト、リリース
体制	お客様、コンサルティング会社、コンテンツ企画会社、エクサの計4社
スプリント実施時の人数	14～23名

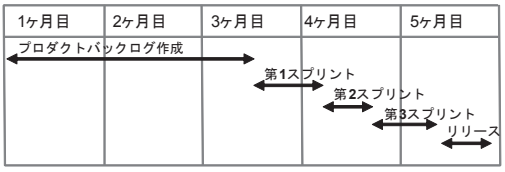

図２　プロジェクトのスケジュール

3.2. Agile開発適用時の課題

Agile開発を適用するにあたり、本プロジェクトにおいて、「要求コントロール」、「開発アーキテクチャ」、「チームビルディングと開発プロセスのコントロール」に以下の課題があった。

(1) 要求コントロール

本プロジェクトは４社が関わるプロジェクトであった。４社の作業拠点は互いに離れており、各社間の情報の共有が困難であり、ステークホルダー間の合意に時間がかかることが予想された。

(2) 開発アーキテクチャ

開発期間が短く設定されたため、開発初期から複数の開発者による開発となった。そのため、生産性と品質を高めるために、並列で効率的に開発可能かつ開発者間のコンフリクトが起こりにくいアーキテクチャが必要であった。

(3) チームビルディングと開発プロセスのコントロール

開発期間が短く設定されたこともあり、開発メンバーが14～23人と多くなった。そのため、開発メンバーへの情報発信方法や進捗のコントロールとメンバー間のコンフリクト解消を効率的に行う必要があった。

以降の章では、３つの課題の対策として、「プロアクティブな要求コントロール」、「EAP開発アーキテクチャ」、「自律したチーム」について述べる。

4. プロアクティブな要求コントロール

Agile開発では、小さい作業範囲（スコープ）の製品要求事項を複数回実装することで最終製品を生み出す。スコープを小さくすることで、プロダクトオーナーの早期決断を促すことが可能となり、早期に開発着手が可能となる利点がある。ただし、本プロジェクトにおいては、小さな

スコープを平均２週間間隔の短期間に決定していくプロセス（要求コントロール）上に、以下の課題が存在した。

(1) お客様と開発者が同一拠点で作業できない

本プロジェクトも含め、日本に多く見られる開発体制ではシステム開発時にお客様と開発者が同一拠点で開発することは少なく、要求事項を詳細化するための情報を得るのに時間がかかるケースが多い。

(2) 要求コントロール用成果物とプロセスの定義

要求事項の詳細化が十分でない上、ウォーターフォール開発に比べ、簡単に要求事項を変更することが可能なため、無制限に要求が増加しないような歯止めが必要であった。

(3) ステークホルダーの要求コントロール

４社が合意できるよう要求をコントロールする必要があった。

本章では、これら要求コントロールに関する課題の対策と実プロジェクトでのScrumを用いた対策を述べる。

4.1. Scrum内の体制の改良

Scrumの一般的な体制については2.2章で述べたが、実際のプロジェクトでは以下の２つの問題があることが多い。

(1) プロダクトオーナーとなり得る人物は、他業務を兼務するが故に、プロダクトオーナーとしての作業に十分な時間を割けないことが多い。

(2) スクラムマスターは、優秀な人物であるが故に、チームの管理や各種ミーティングの準備に加えてチームメンバーとして開発に携わることも多く、仕事の負荷が集中する可能性が高い。

本プロジェクトでも、お客様側の中心人物が多忙であり、理想的なプロダクトオーナーの役割を担うのは現実的に難しかった。また、ステークホルダー間の物理拠点が離れていることによる距離（移動時間）の制約、および開発チームの短期立ち上げが必要というスケジュール上の制約から、スクラムマスターは開発リソースの管理に専念する必要があった。そこで、本プロジェクトでは新たに「スクラムマネジャー」を定義し、本来プロダクトオーナーやスクラムマスターが担うべき要求コントロールに関するタスクを専門的に代行／支援させることとした。スクラムマネジャーの役割を図３にまとめる。

凡例　×➡○：×の担当作業を○へ委譲　　○➡△：○の担当作業の一部を△側で支援

	作業	役割 プロダクトオーナー	スクラムマネージャ	スクラムマスター
プロダクトバックログの作成	プロダクトバックログのメンテナンス	× ➡	○	－
	開発項目の優先順位の決定	○ ➡	△	－
スプリント計画ミーティングの開催	日時調整と進行	－	○ ⬅	×
	次スプリントでの開発項目の提案	－	○ ⬅	×
スプリントの実施	チームを外部妨害から守る	－	○ ⬅	×
	スプリントの遂行	－	△ ⬅	○
スプリントレビューミーティングの開催	日時調整と進行	－	○ ⬅	×
	スプリントで終了した開発項目の説明	－	○ ⬅	×

図３　スクラムマネジャーの役割

4.2. 要求コントロール用成果物とプロセスの定義

要求コントロールはScrumのフレームワークに準じて実施した。[8)]

4.2.1. 要求コントロール用成果物

Scrumでは要求コントロール用成果物としてプロダクトバックログを使用する。本プロジェクトでは、プロダクトバックログとして、「機能一覧」と「画面仕様書」を定義した。「機能一覧」は、粗い粒度の機能の一覧で、個別の機能の優先順位、複雑度、処理の概要、相対値による見積もりを記載している。「画面仕様書」は、機能一覧の機能を画面単位に分割した資料で、より細かな機能を、テキストインプットやサブミットボタンなどの画面要素として可視化している。一般的にプロダクトバックログはテキストベースのユーザ要求の一覧であるが、新規事業ということもあり、お客様がサービスをイメージできる画面仕様書をベースとした。この資料はコンサルティング会社とコンテンツ企画会社が作成し、お客様に提示する前に、実装面で破綻しないことをエクサが確認することで、要求および実現性と開発コストをコントロールした。

4.2.2. 要求コントロールプロセス

Scrumでは、プロダクトバックログを用いて要件をコントロールする。本項では、要求コントロールプロセスと先に挙げた問題に対処するため新設したスクラムマネジャーの作業について説明する。[9)]

(1) スプリント期間内の初期要件の決定

Scrumでは、スプリント開始前に「スプリント計画ミーティング」を実施する。スプリント計画ミーティングは、スプリント内に製品に実装する開発項目を決めるためのミーティングである。このミーティングで要求事項に対する要件をドキュメントにまとめ、プロダクトオーナーに説明し、スプリント内の開発項目と要件を明確化した。ただし、この時点では画面デザインのみが明確になっていた。

(2) スプリント期間内の要求の詳細化とコントロール

本来は、各スプリントの開発項目の詳細な仕様の決定権は、スクラムマスターを含むスクラムチームにある。また、仕様に対するプロダクトオーナーとの認識のずれは、スプリントレビューミーティングで得られるフィードバックコメントとして吸収し、これらをプロダクトバックログに新たに追加する。しかしながら、本プロジェクトでは、イテレーション回数が３回と少ないため、フィードバック回数も少なくなる。そのため、お客様の要求を最大限に取り入れつつ、以下の課題を克服する必要があった。

① 開発期間が短期間（約２ヶ月）であるが、リリース後、すぐにサービスインを行うため高い品質が要求される
② 遠隔作業となるため仕様確認が遅れ、詳細化に時間が費やされる
③ デザイン会社の作業という外部依存タスクが存在し、随時仕様調整が必要
④ 開発作業中の項目にもかかわらず、仕様変更要求が発生
⑤ サービス内容が固まっていない開発項目の検討をスプリントと並行して実施することが必要

これらの作業をプロダクトオーナーとスクラムマスターが行うと、双方の作業負荷が高すぎてプロジェクトの遅延が発生する。そのため、以下の調整作業の大部分をスクラムマネジャーが専門的に行った。

① サービスインまでに開発可能な開発項目を、相対見積もり[19)21)]を用いて提示
② 仕様の詳細化が不十分な項目に対し、開発が破綻しないよう要求をコントロールしながらお客様と調整を実施
③ あいまいな仕様に関しては、チームの混乱を避けるべく実装できる範囲までを伝え、最終的な詳細仕様が決定した時点で簡潔な形でメンバーに伝達
④ デザイン会社の作業が破綻しないように仕様調整を実施すると同時に、開発タスクの遅延が発生しないようにデザイン会社からの受け入れを調整
⑤ 画面設計書で決定した開発項目の入れ替えを、次のスプリントで実施することとし、現スプリントでの要求受け入れをブロック。お客様の仕様変更要求を評価、作業見積もりを実施、さらに、次のスプリントの開発項目と開発範囲を検討

スクラムマネジャーがこれらの作業を代行することで、スクラムマスターはチームの管理や作業の効率化、自律化に専念、そしてスクラムチームは開発に専念でき、開発効率が向上した。また、あいまいな仕様によるスクラムチームの混乱、要求事項の評価不足による作業規模拡大、および仕様調整遅延による開発期間の逼迫などのスコープクリープとなる原因を効果的に取り除いた。

(3) 要件のフィードバック（仕様変更作業）

Scrumでは、スプリント終了後に「スプリントレビューミーティング」を実施する。スプリントレビューミーティングは、スプリントで開発した項目をデモンストレーション形式で説明し、お客様が要望した製品であるかどうかのフィードバックを受けるためのミーティングである。

スクラムマネジャーは、スプリントレビューミーティングで得られたフィードバックコメントやお客様の要望を判断材料として、プロダクトバックログをメンテナンスし、優先順位を付ける。その上でスプリント計画ミーティングに臨み、要求変更項目も含んだ形で次のスプリントに開発する機能を提案する。この提案は、プロダクトオーナーにそのまま承認されることもあれば、開発リソースの上限に達しているためすべての要求を満たすことができず調整が入る場合がある。その場合は、全要求を満足する見積もりを提示した上で、次のスプリントの開発予定機能および仕様変更項目の優先度を比較し、優先度によって対応範囲を

調整した。

一般的なScrumでは、プロダクトバックログに新たな要件を追加し、優先順位を決定する作業は、本来プロダクトオーナーの役割であり、継続的に内容を更新するのもプロダクトオーナーの役割である。しかし、本プロジェクトでは、プロダクトオーナーに代わり、プロダクトバックログの内容を更新する作業はスクラムマネジャーの役割とした。加えてスクラムマネジャーは、開発項目の優先順位を提案し、プロダクトオーナーの意思決定をサポートする役割とした。その結果、プロダクトオーナーはその提案内容を勘案し、最終的な開発項目の優先度を迅速に決定することが可能となった。

4.3. スクラムマネジャーの利点

スクラムマネジャーの役割を設けることにより、プロダクトオーナーは最も重要な作業である開発項目の優先順位付けに専念できるようになる一方、スクラムマスターもチームの管理や作業の効率化、自律化に専念することが可能となった。

もしも、本来の方式に固執し、スクラムマスターがこれらの作業を担当していたとすると、お客様からの追加開発要求の検討時間が増え、チームのコントロールも含む開発作業に集中できなくなり、スプリントの開発は困難を極めたであろう。中規模以上のAgile開発を成功させるためには、一般的なScrumの体制に囚われず、プロジェクトの性格や特性、想定される作業とその負荷に応じて役割分担を明確にし、柔軟に体制を組むことが重要である。

5. EAP開発アーキテクチャの設計

開発の生産性および品質を上げるためにXPを採用したが、「多人数での開発」かつ「インクリメンタルな仕様追加」に対応するには、耐性のある開発アーキテクチャが必要となる。また、前章で述べたように本プロジェクトでは「機能一覧」と「画面仕様書」で要求コントロールしており、要件（特に画面仕様書やHTMLのモック）からコード製造につながる仕組み、ツール、および技術を採用することで開発効率を向上させた。本章では、このプロジェクトで採用した開発アーキテクチャについて述べる。

5.1. EAP開発アーキテクチャ設計方針

EAP開発アーキテクチャを設計する上で、本プロジェクトの特性を加味し、XPの５つの基本原則中、次の３つを採用した。

(1) シンプルさの採用
　システムをできる限りシンプルに保つ

(2) インクリメンタルな変更
　現在必要なことだけを行う単純なコードを書き、将来の変更はそのときになってから行う

(3)　変化を取り入れる
　お客様の要求が開発当初と異なった場合に素早く変更する

EAPの開発アーキテクチャの要件として、多人数で生産性を向上させるメカニズムと、変更に柔軟で、シンプルかつメンテナンス性の高い構成が必要となる。そのため、この基本原則から、アーキテクチャの設計方針を以下の６つに詳細化した。

(1) レイヤ設計の明確化
(2) DI（Dependency Injection：依存性の注入）の利用
(3) アスペクト指向による共通機能の分離
(4) 画面デザインをプログラムへ透過的に反映する仕組み
(5) データベーススキーマの変更に対する透過的なコードの変更
(6) DTO（Data Transfer Object）優先

以下、アーキテクチャ設計の６つの方針について説明する。

5.1.1. レイヤ設計の明確化

システムをできる限りシンプルに保つために、MVCモデルのオープンソースフレームワークを活用し、以下の３層構造を基本構成とし、３層それぞれが担う責務を以下のようにオブジェクトのグループとして細分化し、それぞれに対してデザインパターンおよびアーキテクチャパターンを適用した。

(1)　プレゼンテーション層

①　Page

画面に表示するデータ構造を持つ

②　Action

業務ロジックの呼び出しと画面遷移を制御する

(2)　ロジック層

①　Service

一連の業務ロジックの処理を担当し、トランザクションの境界として振る舞う

②　Logic

再利用可能な単一の業務ロジック

(3)　永続化層

①　DAO（Data Access Object）

SQLの発行と、ORマッピング機能によるデータベースのレコードとエンティティオブジェクトとのマッピングを担当する

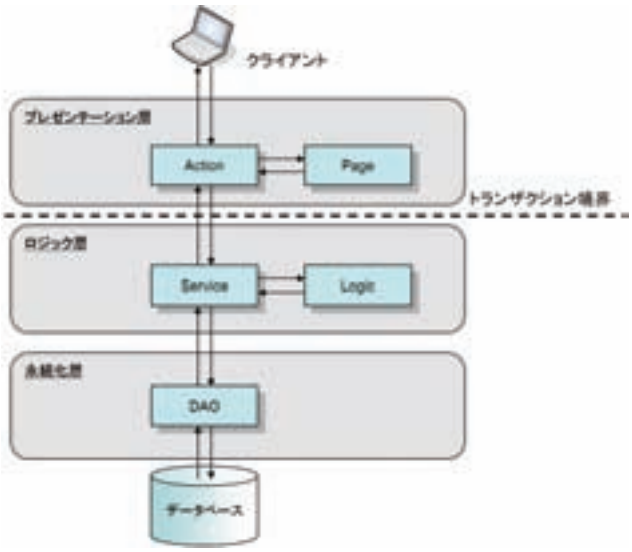

図４　３層構造のレイヤ設計

図４に３層構造のレイヤ設計を示す。層（レイヤ）による基本構成を開発者間で共有することにより各レイヤが担う責務や呼び出しのルールが明確になり、システムの構成をシンプルに保つことにつながる。また、各レイヤにデザインパターンおよびアーキテクチャパターンを適用することにより、記述するコードに対する規約ベースの標準を与えた。これにより作業の分担と作業コンフリクトの最小化を図った。

5.1.2.　DIの利用

DIにより「設定と利用を分離する」ことが可能となる。例えばプレゼンテーション層から呼び出すロジック層の機能が未実装であった場合でも、プレゼンテーション層のコードを変更せずにロジック層の機能を実装したモックを適用し、ユニットテストを実施することができる。

図５にDIによる設定と利用の分離について示す。

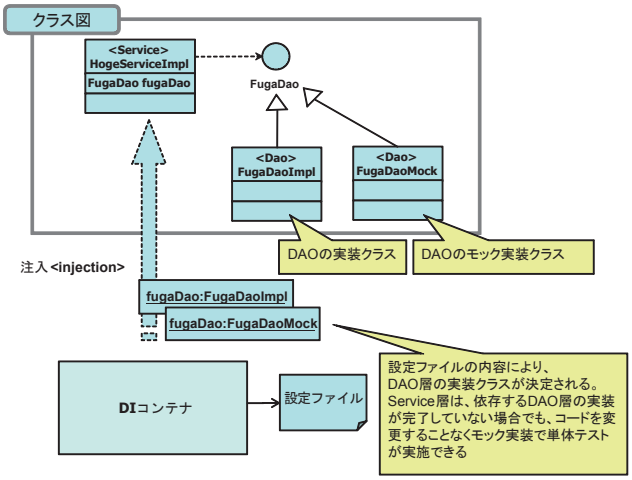

図５　DIによる設定と利用の分離

5.1.3.　アスペクト指向（AOP）による共通機能の分離

アスペクト指向フレームワークを利用し、以下の共通機能を各コードから分離した。

(1)　認証

(2)　アクセス制御

(3)　トランザクション

(4)　トレースログ

これらの共通機能は横断的関心事と呼ばれ、従来の実装方法では個々に実装する必要があり、変更に際しては対象箇所が複数に及ぶため、メンテナンス性の低下につながっていた。これらの問題に対し、AOPを採用することで、開

発者は各レイヤの責務に則したコードの実装に集中することができ、共通機能の変更箇所も限定的になるためメンテナンス性が向上した。

図６に従来の実装方法とAOPによる実装の比較を示す。

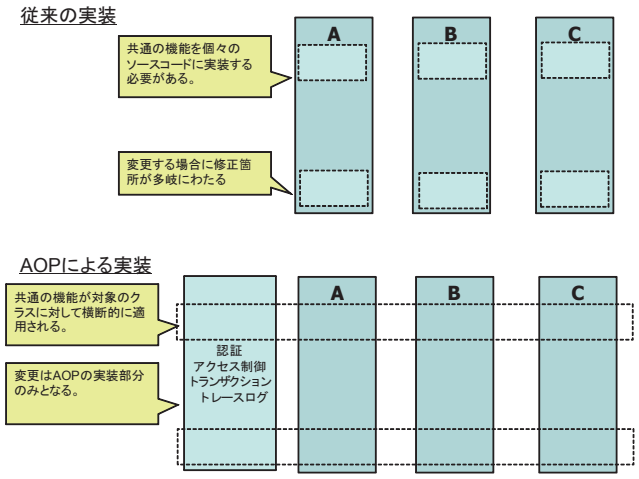

図６　従来の実装方法とAOPの比較

5.1.4. 画面デザインをプログラムへ透過的に反映する仕組み

多くのMVCフレームワークでは、画面のテンプレートにJSP®（Java Server Page）を採用しており、画面デザイン担当者から提供されるHTMLをJSPに反映する作業が大きな障害となっていた。

そのため、画面のテンプレートをXHTMLとし、画面デザイン担当の成果物をそのままプログラムに取り入れることで変更に素早く対応することを可能とした。また、XHTMLの制作にあたり、画面デザイン担当者に対してフレームワークの特性を説明し、開発側で作業が発生しないように本プロジェクト用のXHTMLコーディング規約を事前に作成した。

5.1.5. データベーススキーマの変更に対する透過的なコードの変更

データベーススキーマの変更が発生した場合、変更を必要とするソースコードが多岐にわたることが問題となる。そのため、テーブル設計書および実データベースから、ソースコードを自動生成するツールを採用し、アーキテクチャの一部として適用した。このツールにより、スキーマの変更とソースコードの変更を同期させることが可能となった。

図７に透過的なコード変更について示す。

また、表２にツールによって自動生成するソースコードの概要について示す。

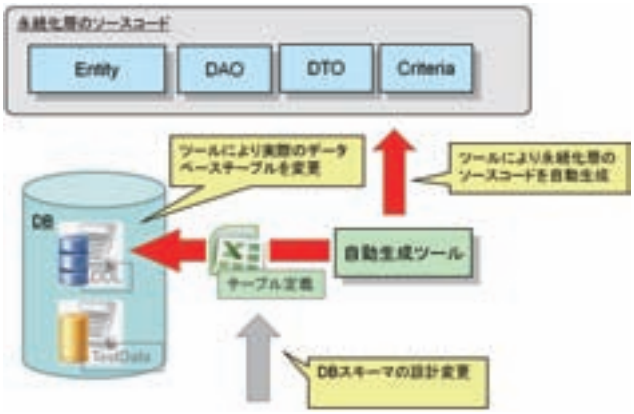

図７　永続化層における透過的なコード変更

表２　永続化層自動生成コード一覧

自動生成コード	自動生成コードの役割
エンティティ（Entity）	テーブルと1対1でマッピングされるオブジェクト
DAO	SQLの発行と、ORマッピング機能によるレコードとエンティティとのマッピングを担当するオブジェクト
DTO	エンティティを単純なデータモデルとして扱うためのオブジェクト
Criteria	DAOが発行するSQL文を、テーブル名、カラム名などの物理名をマッピングするためのオブジェクト

5.1.6. DTO優先

「インクリメンタルな変更」の基本原則に従い、レイヤ間のデータ受け渡しについてはDTOを利用することとし、ドメインオブジェクトを事前に設計・実装することを禁止した。これは、画面ベースで要件を決めており、開発初期段階では全体像が明確ではなく、業務ロジックも変更の可能性があるので、業務ロジックを推測してドメインオブジェクトを設計することにリソースを使うのは非効率と判断したためである。

業務ロジックが明確になり、ドメインオブジェクトによる明確なデータ構造およびロジックの統合が必要となった

段階で、ドメインオブジェクトの設計と実装を実施するというルールを明確にした。

5.2. EAP開発アーキテクチャの構成

ここでは、EAP開発アーキテクチャを実現するための構成について説明する。EAP開発アーキテクチャは、オープンソースフレームワークを組み合わせて構成されており、６項目の設計方針に対してオープンソースの標準機能でまかなえない部分は独自に拡張した。図８にEAPに適用したアーキテクチャとフレームワークの構成を示す。

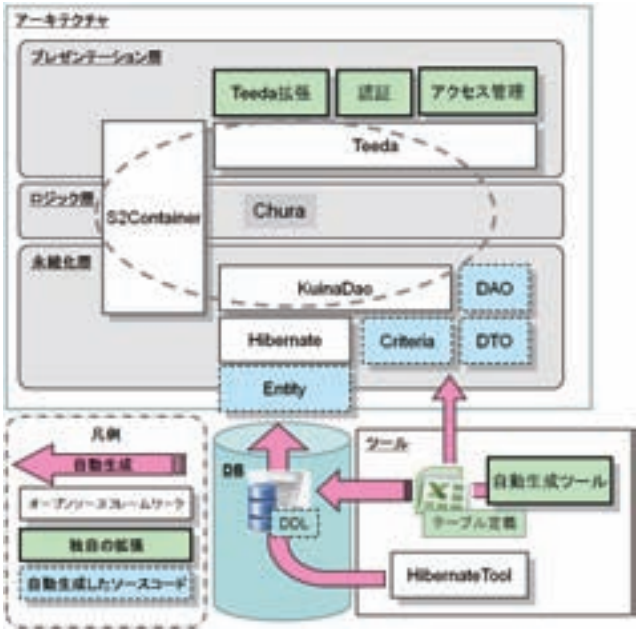

図８　EAP開発アーキテクチャとフレームワークの構成

5.2.1. オープンソースフレームワークの利用

オープンソースフレームワークの機能を採用することで、以下の４つの設計方針について開発工数を削減し生産性を向上させた。

(1) レイヤ設計の明確化
(2) DI（Dependency Injection：依存性の注入）の利用
(3) アスペクト指向による共通機能の分離
(4) 画面デザインをプログラムへ透過的に反映する仕組み

オープンソースフレームワークとして、オープンソースコミュニティ Seasar Project[13] のフレームワークスタック「Chura」が必要な機能をスタックとして統合化していたため採用することとした。「Chura」では、「Easy Enterprise」と呼ばれる概念で、エンタープライズ向けのWebアプリケーション開発に適したフレームワーク製品群のスタックおよび開発ツールを提供しており、MVCフレームワーク、３層レイヤの構成と規約、DIコンテナ、AOPと必要な機能を包含していることが採用の大きな要因であった。オープンソースのフレームワークスタック「Chura」の製品構成および設計方針との対応を表３に示す。

表３　「Chura」のフレームワーク製品群

分類	フレームワーク/ツール名	概要	メリット	設計方針との対応
MVCフレームワーク	Teeda	XHTMLをテンプレートとして利用可能なJSF拡張フレームワーク	規約重視による生産性の向上および、画面デザインをプログラムに対し透過的に適用できる	(1)、(4)
DIコンテナ	S2Container	DI機能およびAOP機能を提供するコンテナ	単体テストを容易にし、修正時の変更箇所を限定できる。また、AOPにより、横断的関心事に対する変更箇所を限定できる	(2)、(3)
DAOフレームワーク	Kuina-Dao	AOP機能を利用し、規約とアノテーション（注釈）から、SQLを自動発行するフレームワーク	SQLの記述を必要最低限に抑えられるため生産性が向上する	(1)、(5)
ORマッピングフレームワーク	Hibernate™	データベースとオブジェクトのマッピング機能を提供するフレームワーク	データベースとアプリケーションの結合度を低く抑えられ、変更に柔軟に対応できる	(5)
	Hibernate-Tools	データベース上のテーブル定義からEntityのソースコードを自動生成するためのツール	テーブル設計の変更を即座に、ソースコードに反映できる	(5)

5.2.2. 独自の拡張

独自拡張部分について、表４に実装したフレームワーク、ツールの説明および設計方針との対応を示す。特に永続化層については、以下の方針に従い、自動生成ツールを実装し、コーディングルールを策定することで、永続化層への仕様変更に対しても、柔軟にコード変更可能な対策をとった。

(5) データベーススキーマの変更に対する透過的なコードの変更
(6) DTO優先

6. 自律したチームの構築と管理

Agile開発では迅速な情報共有とコンフリクト解消および自発的なプロジェクト進行が必須条件となる。そのため、最小限のコスト（時間）で情報共有とコンフリクト解消ができる場が必要となる。この場を有効に運営することで、チーム内の関係や開発プロセスをイテレーションごとに改善し、状況変化に柔軟に適応できるチームに成長させる。本章では、自律したチームのチームビルディングと省力化した開発プロセス管理について説明する。

6.1. チームビルディング

自律したチームを構成するためには、リスクを早期に検知し、コンフリクトを随時解消するサイクリックなプロセス改善活動が必要となる。それを効率よく行うためのプラクティスとしてスクラムミーティングと振り返りを、実例に基づき説明する。

6.1.1. スクラムミーティング

本プロジェクトではスクラムミーティングは朝10時の定時に開催する朝ミーティング[14]とした。スクラムミーティングでは、スクラムチームのメンバー全員が「前日に実施したこと」「本日実施すること」「問題点あるいは課題」を報告し、問題点あるいは課題については、関係者だけが集まった別のミーティングで収束させることとした。課題や問題の議論をしないようにすることで議論に直接関係しないメンバーの時間の浪費を抑え、作業時間を長くとれるようにした。当初は１時間を越えることもあったが、プロジェクト途中からは15分程度（一人あたり１分）で終了できるようになった。

スクラムミーティングの目的は以下の３点に集約される。

(1)　メンバー全員が担当タスクを理解していることの確認とタスク実施のコミットメントを促す
(2)　メンバー全員がチームの進捗状況を把握する。また、スキルにバラツキがある場合は、早期に問題解決でき

表４　独自の拡張

分類	フレームワーク/ツール名	概要	メリット	設計方針との対応
プレゼンテーション層	Teeda拡張	汎用的なカスタムバリデータ/コンバータなどを提供するTeedaの仕様に則した拡張ライブラリ	再利用性が高い	(1)、(4)
	認証	ログイン/ログアウト機能を提供するフレームワーク	AOPにより、ログイン/ログアウトの処理を、コントローラの処理およびロジックから分離できる	(3)
	アクセス管理	ログインユーザの権限に対してページごとのアクセス管理機能を提供するフレームワーク	アクセス管理の処理を、コントローラの処理およびロジックから分離できる。また、共通の抽象親クラスを提供することで実装方式を統一できる	(3)
永続化層	自動生成ツール	設計書からDDLファイルおよび永続化層のソースコードを自動生成するツール	テーブル名、列名の変更を自動的にソースコードに反映され、リファクタリングすることなくテーブル設計の変更をソースコードと同期できる	(5)、(6)

るよう助言を促す
(3)　問題点（リスク、コンフリクト）を日々確認し、早期に対策を打つことでリスクとコンフリクトを最小化する

これらを実践することで早期にリスクとメンバー間のコンフリクトを解消した。ミーティング開催時刻に関しては、朝と夕方のどちらがよいかという議論があるが、朝の方が個々人の問題が整理されており、作業にはいりやすいとの意見が多かった。また、参加メンバーの人数が多くなると効率が悪くなることもあるが、なるべく全員が集合し、１回で済むように実施するのが望ましい。

図９　KPTTの例

6.1.2. 振り返り

本プロジェクトでは、スプリントの完了ごとにレトロスペクティブ[15)16)17)]と呼ばれる「振り返りミーティング」を実施した。このミーティングはスプリントメンバー全員が集まり、スクラムの進め方や課題などをチェックし改善していくためのものである。レトロスペクティブの進め方は、KPTTという手法を利用した。KPTTの各文字の意味は以下のとおりである。

(1)　K（Keep）…プロジェクトで今後も続けていきたいこと（Try結果が良好だったものの集合）
(2)　P（Problem）…問題があったことであり今後改善が必要なこと
(3)　T（Try）…プロジェクトをより良くしていくために今後チャレンジしたいこと（Problemの対策）
(4)　T（Thanks）…感謝したいこと

今回は、ホワイトボードにKPTTの４つの枠を作成し、各メンバーが付箋に自分の気づいた点を自由に記述し張っていく方法をとった。全員の意見がそろったところで、類似項目を集約し、共通の課題やタスクを抽出して、今後の進め方について意見を出し合うようにした。時間は、各自の意見を出すのが20分、整理および意見交換が30分、まとめを10分程度とし、計１時間を目安に終了するようにした。図９にKPTTの実施例を示す。

このレトロスペクティブを行って、スプリントごとに問題点を明確にし、次のスプリントに反映することで、作業効率の改善、コンフリクトの解消を行った。そして、Thanksという項目で、明確に感謝を述べることで、お互いの助け合いを助長し、チームワークの向上につながった。また付箋を使うことで、ミーティング形式ではあまり主張しないメンバーからの意見も幅広く収集することができ、この点もコンフリクト解消につながっている。

6.2. 開発プロセスの管理

本プロジェクトでは、自発的な情報発信の促進とリアルタイムに進捗を管理するためにチケットシステムを導入した。チケットシステムとしてはTracを採用し、進捗管理用のバーンダウンチャートプラグイン機能を追加し、6.2.1項で述べるタスクの消化フローに合うよう、チケットの状態をカスタマイズした。

6.2.1. チケットシステムによる開発プロセスの管理

タスクは、スプリントバックログから以下の観点で抽出し、日々の進捗が確認できるように0.5人日～2人日程度になるように分割した。

(1)　スプリントバックログで必要となる機能の設計
(2)　スプリントバックログを実現するための個々のPage、Action、Service、Logic、DAOクラスの実装および単体テスト
(3)　スプリントバックログ単位の結合テストの作成、実施

実際の開発では、開発環境（Eclipse™）にEclipseとTrac、Subversion®を連携するためのMylyn™プラグインをインストールした。このプラグインにより、開発者はTracで管理されるタスクをEclipse上で確認することができ、そのタスクと紐付けて製造したコードをSubversionにコミットすることで、スクラムマスターはタスクあるいは障害ごとに、新規製造あるいは変更したファイルの一覧と差分情報を取得できるようになった。これにより、スクラムマスターのタスクの完了と障害改修の受け入れ作業が容易になった。

次にタスクの消化フローについて説明する。第１スプリントでは、以下の手順でチケットを管理した。

(1)　スクラムマスターがスクラムメンバーにチケットをアサインする
(2)　スクラムメンバーは、チケットの内容を実施し、完了したらスクラムマスターにレビュー依頼をする
(3)　スクラムマスターはチケットの内容が正しく実施されているかをレビューし、クローズまたは再アサインする

しかし、この手順では、スクラムマスター不在時に、タスクが開始できなくなる。この課題を改善するため、第２、第３スプリントにおいてはスクラムメンバーが自分でタスクを取得できるようタスクの依存関係を記録し、各自でチケットを取得して実施する手順とした。これにより、スクラムメンバーのタスク割り当て待ち時間を減少した。さらに、スクラムメンバーが自分の裁量でチケットを選択できることでモチベーションが向上し、効率よくタスク消化できるという副次的効果も見受けられた。

6.2.2. 進捗の確認

進捗は、チケットに作業の見積時間と実績時間を入力し、Tracのバーンダウンチャートプラグインによりグラフ化し管理した。バーンダウンチャートとは、残作業時間をグラフで管理する方法である。バーンダウンチャートでは、縦軸に残作業時間、横軸に経過日数を指定し、残チケットの見積時間の合計をプロットする。日々のタスクを消化していくにつれ、右下がりのグラフになり残作業時間が０になった時点でスプリント完了となる。この、バーンダウンチャートのプロットの傾きから、スプリント完了日までにすべてのタスクが終了できるかを見通すことができた。また、本プロジェクトでは、課題、障害も同一のチケットシステム上で区別なく管理しているため、PMあるいはスクラムマスターが、１つのビューで開発の進捗を把握できることがメリットとなる。バーンダウンチャートの例を図10バーンダウンチャートに示す。

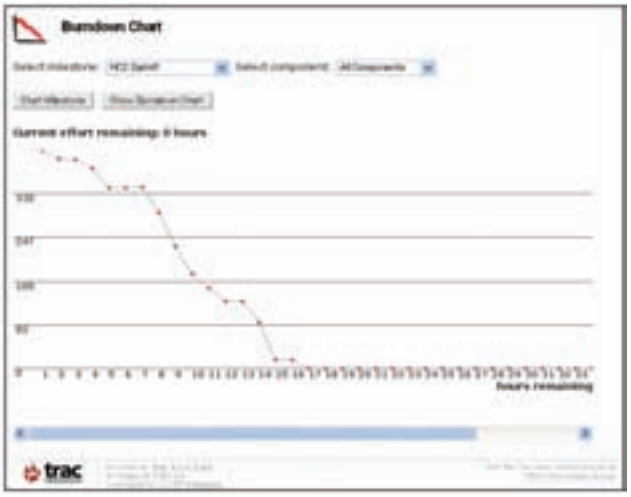

図10　バーンダウンチャート

6.3. 効率の良い開発プロセスの運営

開発プロセスを実際に運営する上で、以下の２点の改善が効果的であった。

(1) 開発効率の良いタスクの割り当て

第１スプリントでは、スプリントチームを画面チーム、ロジックチーム、DAOチームのようにレイヤごとに分けて開発を進めた。この場合、各チームで実装・単体テスト後、スプリント終盤に結合試験という手順となり、各チーム間の認識の齟齬により正常に動作しないということが発生した。そこで第２スプリント以降は、スプリントチームを実現すべき機能ごとに分け、その機能を実現するための画面、ロジック、DAOのすべてを実装するようにした。またCIシステム（CruiseControl）を導入し、常に製品をビルドできる環境を構築した。これにより、チーム内で機能ごとに実装・単体試験終了後にすぐに結合して動作させることができ、開発効率が改善した。

(2)　座席のならび

第１スプリントでは画面チームが２チームあったが、座席の空きの関係で１チームだけ少し離れた場所で作業をすることになった。Agile開発では、コミュニケーションを重視する必要があるが、席が離れたチームはコミュニケーションが希薄になる傾向があった。座席に関しては、スクラムマスターを中央に集め、スクラムチームのメンバーをそれぞれのスクラムマスターの周囲に配置しスター状の形をとるのが最良である。

7. まとめ

今回Scrum+XPの各プロセスをプロジェクト固有の状況に合わせカスタマイズした「EAP」を適用することで、Agile開発の３つの問題に適切に対応し、お客様の要望どおりの製品をリリースすることに成功した。これらの経験から、我々はプロジェクトにAgile開発を適用することで、次の３つのメリットを得ることができると考える。

(1)要件の決定と詳細化が遅れがちなプロジェクトにおいては、ウォーターフォール開発に比べ、Agile開発の方が、要件コントロールに関するリスクが少ない。すなわち、利用して初めてわかる問題点や、文書では伝わらない要件の意識違いを早期に抽出できる

(2)開発初期から優先順位の高い機能を盛り込んだ製品をお客様と開発側双方で確認できるため、いつでもリリースできる(常に動作する)という安心感を得られる

(3)アジャイル開発（Scrum）では開発初期からリリースまでメンバーを固定するため、振り返りや普段のミーティングを通じ、継続的にチームの生産性、品質、モチベーションを向上しやすい

一方、一般的にAgile開発では、開発規模が大きくなるにつれお客様の要求コントロールの失敗や、そこから派生する開発アーキテクチャの破綻からプロジェクトの失敗につながりやすい。そのため、Agile開発について必要十分な方法論を基に、プロジェクトごとに実プロセスと開発アーキテクチャを構築することが必須と考える。今後、この適切に管理できた事例を元にEAPの三要素、すなわち①プロアクティブな要求コントロール、②EAP開発アーキテクチャ、③自律したチームの考え方を汎化し改善して、他案件への横展開も図りたい。

参考文献

1) Kent Beck他,「Manifesto for Agile Software Development」, http://www.agilemanifesto.org
2) バリー・ベーム,リチャード・ターナー.「アジャイルと規律~ソフトウェア開発を成功させる2つの鍵のバランス」, 日経BP社,2004
3) Kent Beck,「XP エクストリーム・プログラミング入門 - ソフトウェア開発の究極の手法」,ピアソンエデュケーション,2000
4) クレーグ・ラーマン.「初めてのアジャイル開発　～スクラム、XP、UP、Evoで学ぶ反復型開発の進め方～」, 日経BP社,2004
5) ロバート・C・マーチン,「アジャイルソフトウェア開発の奥義」,ソフトバンク　パブリッシング,2005
6) アリスター・コーバーン,「アジャイルプロジェクト管理」, ピアソンエデュケーション,2002
7) アリスター・コーバーン,「アジャイルソフトウェア開発」, ピアソンエデュケーション,2002
8) ケン シュエイバー,マイク ビードル.「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
9) 岡島　幸男,「プロジェクトを成功させる現場リーダーの技術」, ソフトバンククリエイティブ,2006
10) スコット・W・アンブラー,「アジャイルモデリング 」,翔泳社,2003
11) スコット・ローゼンバーグ.「プログラマのジレンマ 夢と現実の狭間」, 日経BP社,2009
12) 江渡 浩一郎.「パターン、Wiki、XP ～時を超えた創造の原則」, 技術評論社,2009
13) ひが やすを.「Seasar2入門」, ソフトバンククリエイティブ,2009
14) 平鍋健児,「プロジェクトファシリテーション　実践編　朝会ガイド」,永和マネージメント（ObjectClub.jp）,2005 http://www.objectclub.jp/download/files/pf/MorningMeetingGuide.pdf
15) 天野勝,「プロジェクトファシリテーション　実践編　ふりかえりガイド」,永和マネージメント（ObjectClub.jp）, 2007 http://www.objectclub.jp/download/files/pf/RetrospectiveMeetingGuide.pdf

16） 土屋正人,「前進するためにふりかえる」,SRA（sra.co.jp）,2008
http://www.sra.co.jp/public/sra/event_seminar/event2008/080620-h/kouen-2.pdf
17） Esther Derby, Diana Larsen.「アジャイルレトロスペクティブズ　強いチームを育てる「ふりかえり」の手引き」,オーム社,2007
18） Henrik Kniberg.　「Scrum and XP from the trenches」
http://www.infoq.com/minibooks/scrum-xp-from-the-trenches
19） Kent Beck,　Martin Fowler.「 XP エクストリーム・プログラミング実行計画」, ピアソンエデュケーション,2001
20） Don Wells.「Extreme Programming:　A gentle introduction」
http://www.extremeprogramming.org/
21） Mike Cohn.「Agile Estimating and Planning」.
Prentice Hall,2005
