2006－SE－151（12）
2006／3／23

アジャイル開発における設計とテスト作業の具体化とその適用結果

藤井　拓[†], 鶴原谷　雅幸[†]

本論文では, 5つのフェーズで開発ライフサイクルを構成するアジャイル開発手法AMRMの設計作業とシステムテスト作業の具体的な実施方法を考案し，開発プロジェクトに適用した結果を報告する．考案した設計作業とシステムテスト作業の具体的な実施方法の有効性を確かめるため，これらの具体的な実施方法を2つのプロジェクトに適用し，開発途上での設計品質，開発生産性，累積不具合検出数を測定した．測定結果から，設計作業を推敲フェーズ以降に適宜実施する「小まめにモデリングアプローチ」が設計品質を高めるために有効であること及びソフトウェアの不具合の発見には，インフォーマルなシステムテストを早期に実施することが有効であることが分かった．

Implementation of Design and Testing Activities in Agile Development and Its Application Results

Taku Fujii[†], Masayuki Tsuruharaya[†]

This paper reports implementation and application results of design and system testing activities in an agile method, AMRM, based on 5 phases lifecycle model. AMRM was applied to 2 projects and design qualities, productivity, cumulative defects detected by testing, were measured in those 2 projects to examine the effectiveness of the implementation. A design approach to have multiple detail design and refactoring sessions after the elaboration phase is proven to be effective in giving better design qualities. Also, early application of informal system testing activities is proven to be effective to detect defects in the program codes.

1. はじめに

近年，開発途上での顧客ニーズや技術の変化に対応しながら開発を行うことを目指すアジャイル開発手法[1], [2]と呼ばれるソフトウェア開発手法が提案されている．アジャイル開発手法は，開発方法論の観点では要求，設計，実装，テストのサイクルを重ねながら開発を進める反復型開発アプローチの1種であるが，価値や原則やプラクティスで開発手法を定義している点で通常の反復型開発プロセスと異なる．本論文では，要求，設計，実装，テストを通じて動作するソフトウェアを作り上げる1サイクルを反復（iteration）と呼ぶ．

アジャイル開発では，一般的に1回の反復の期間は1週間から1ヶ月程度に設定されており，各反復の完了時にその反復で作成されたソフトウェアを開発依頼者にデモで見せ，提供し，開発依頼者からのフィードバックを得る．さらに，開発依頼者からのフィードバックは優先度を設定し，次回以降の反復の開発目標へと組み入れていく．

アジャイル開発の大きな特徴は，開発途上での顧客ニーズや技術の変化に対応しながら開発を行うことで開発依頼者にとって有用なシステムを効率的しかもタイムリーに開発することである．しかしながら，実際に開発途上に発生した顧客ニーズや技術の変化に対応するためには，それらの変化に対応するためのプログラムコードの変更を効率的にしかも品質低下を招くことなく行うことが必要になる．すなわち，アジャイル開発の特徴を発揮するためには，以下の2点が課題になる．

- プログラムコードの変更を効率的に行う
- プログラムコードの変更による品質低下を防ぐ

また，アジャイル開発では，一般的に変化への対応を重視するものの最終的にどのような機能が実現するかという見通しを開発初期や開発途上で開発依頼者と話し合うことを積極的に推奨していない．しかしながら，商業的な開発を想定した場合には，最終的にどのような機能が実現するかという見通しがないと開発が承認されることは困難である．すなわち，このような見通しを提供することがアジャイル開発の3番目の課題となる．

これら3つの課題を解決するために，筆者らはアジャイル開発手法に以下の 3 つのプラクティスを導入したAMRM（Agile Method Reinforced with Modeling)[3]という開発手法を提案した．

- ライフサイクル目標
- 協調的な設計モデリング
- テストの自動化

以降の章では，まず AMRM の概要を説明する．次に AMRM の設計作業を具体化する際に考案された2

† (株)オージス総研　技術部ソフトウェア工学センター

つの実施方式とシステムテストの実施方法について説明する．さらに，これらの 2 つのモデリング実施方式を各々適用した 2 つのプロジェクトにおいて測定された設計品質，開発生産性，累積不具合数の推移を紹介し，それらの測定結果に基づいてモデリング実施形式の得失やシステムテストの実施方法の有効性について論じる．

2. AMRM の概要

アジャイル開発手法と同様に反復型開発アプローチに基づく統一プロセス[4]では，開発ライフサイクルを 4 つのフェーズに分け，フェーズ毎のライフサイクル目標を設定している．このようなフェーズを導入することにより，プロジェクトのハイレベルの目標設定を設定したり，プロジェクトを継続するか否かの判断を下すポイントを設けることができる．著者らは，アジャイルな開発においても同様なフェーズ及びライフサイクル目標を導入することで同様の効果が期待できるのではないかと考えた．そこで，AMRM では統一プロセスのフェーズを参考にして表 1に示されるような 5 つのフェーズを設定した．AMRM では，αリリースを用いた広範なユーザからフィードバックを得るために，統一プロセスの作成フェーズを基本作成フェーズと発展フェーズに分割している．

以降，AMRM の各フェーズで実施する作業や目標の概要を説明する．

方向づけフェーズでは，初期要求の策定と検証，アーキテクチャに使われる技術候補の選定を行い，開発対象のソフトウェアの大雑把な機能及びその実現に使う技術を確定させることを目指す．また，推敲フェーズ以降の大雑把な開発計画が立案される．

推敲フェーズでは，実現すべき最低限の要求事項(最低限要求)とそれ以外の要求事項(残要求事項)を確定する．さらに，部分的な実装を行うことでアーキテクチャを検証し，確定する．さらに，推敲フェーズでの実装作業の実績に基づいて基本作成フェーズ以降の開発計画を改訂する．

基本作成フェーズでは，最低限の機能を実現し，αリリースを作成する．αリリースを用いてデモンストレーションを行い，広範な利害関係者からのフィードバックを得る．得られたフィードバック(追加要求事項)と推敲フェーズで実現を保留した残要求事項に対して優先度を設定し，発展フェーズで実現する範囲を確定する．

発展フェーズでは，基本作成フェーズで設定された範囲の要求を実装する．また，ソフトウェアに対するシステムテストを実行し，ソフトウェアが受け入れテストを開始できるレベルの品質に達していることを確認する．

表 1 AMRM の開発フェーズ

開発フェーズ名	開発フェーズの目標
方向づけ	初期要求の策定と検証， 技術調査 初期開発計画の立案
推敲	アーキテクチャの確定 最低限要求の確定 開発計画の改訂
基本作成	最低限要求の実現 αリリース
発展	最低限機能の改良と追加機能の作りこみ βリリース
移行	不具合への対応，開発ドキュメント及び導入成果物等の作成 納品リリース

要員計画の点では，統一プロセスと同様に開発に伴うリスクが高い方向づけフェーズではなるべく少人数でスタートし，以降のフェーズでリスクを軽減しながら順次開発者を増員していくことにより開発に伴うリスクと投入する費用とのバランスを取ることができる．

AMRM では 2 週間単位の反復を設定しているが，各反復で実行する設計以外の開発作業に対するプラクティスについては，文献[5]に記したものを用いている．

筆者らは，このような 5 フェーズに基づくアジャイル開発方式において開発のリスクの早期軽減と設計品質の改善を行うために表 2に示す 3 種類のモデリング作業を考案した．これら 3 種類のモデリング作業は，いずれも協調的にグループワーク形式で行うため，モデリングセッションと呼んでいる．これらのモデリング作業の考案にあたり，重視したのは以下の 3 点である．

- 軽量さ
  - 2 週間というアジャイルな開発の短期の反復サイクルに収まること
- 容易性
  - 最低限の指導でモデリング経験がほとんどない開発メンバーが自ら実践できること
- 設計改善
  - 実装前の設計で多くの時間悩むより，と

表 2 AMRM の設計モデリング作業

モデリング作業名	モデリング作業内容	所要時間
概念設計モデリングセッション	システムの主要機能を実現する論理的な内部機能の相互作用をモデリングする	1 主要機能につき 0.5-1 日間
詳細設計モデリングセッション	システムの主要機能を実現する相互作用の詳細をモデリングする	1 主要機能につき 0.5-1 日間
設計改善モデリングセッション	当初の設計に対して適用された拡張を UML のクラス図でレビューし，クラスの分割の見直し等を行う	1 反復につき 1 日間

りあえず実装し得られた設計を改良すること

これらの AMRM の設計モデリング作業は，CRC（Class Responsibility Collaborator）カードを使った責務駆動設計[6]の方法を簡略化したものである．責務駆動設計に基づいたモデリング作業を行うことにより，以下のような点で設計品質が向上すると期待できる．

- 巨大で複雑なクラスの予防
- クラスの責務のまとまりの向上

このような設計品質の向上により，要求の変化に対応してプログラムコードを変更する場合にも，変更の波及範囲が小さくなると期待できる．

最後に，AMRM ではテストの自動化のプラクティスとして，以下の 3 点を実践することを求めている．

- JUnit 等のテストティングフレームを用いて，反復内で追加されたコードのテストを自動化する
- 自動化されたテストの実行時にテストカバレージを測定し，可能な限り命令行網羅率（C0）と分岐網羅性（C1）を可能な限り高める
- テストコードはアプリケーションプログラムのクラス単位で作成するが，厳密な意味での単体テストを行うことに拘らない

3番目のプラクティスで述べている「厳密な意味での単体テスト」とは，テスト対象のクラスが他のクラスとコラボレーションする場合に，コラボレーション先のクラスに対するスタブを作成して単体テストを実施しないということを意味する．すなわち，自動化されるテストは単体テストと統合テストが混合したものになる．

このようにテストを自動化することにより，既開発コードを変更した際の回帰エラーの発生を抑制できると期待できる．

3. 設計作業とテスト作業の具体化

前章で説明した AMRM の設計モデリング作業やテスト作業に関するプラクティスは比較的抽象的なものに留まっている．そこで，これらのプラクティスのより具体的な実践方法を決めるため，具体的な実践方法を考案し，実際の開発に適用し，それらの開発を測定することで，実施方法をさらに改良していこうと考えた．

筆者らが，AMRM の具体的な実施方法を考案するうえでまず注目したのは，以下の 2 点だった．

- 3 種類の設計モデリング作業をどのようなタイミングでどの程度の労力を投ずればよいか
- システムテストにはどのようなタイミングでどの程度の労力を投ずればよいか

前者は，設計品質に影響を与え，その結果としてアジャイル開発が約束する変化への対応の容易性を左右する可能性が高いのではないかと考えて注目した．後者は，以下の 2 点を達成するために自動化された単体テストや統合テストだけではなく，システムテストの実施方法が大きく影響するのではないかと考えて注目した．

- 納品時のプログラムコードの不具合を実稼動に耐えられるレベルまで減少させる
- 納品する機能の実装完了から実稼動に至るまでの期間を短縮する

これら設計作業とシステムテスト作業の実施方法を考えた結果，3 種類の設計モデリング作業の実施方法については 2 つの方式を考案し，システムテスト作業の実施方法は一つの方式を考案した．以降，これらの方式について説明する．

3 種類の設計モデリング作業の実施方式としては，表

表 3 設計モデリング作業の2つの実施方式

方式名	概念設計モデリングセッション	詳細設計モデリングセッション	設計改善モデリングセッション
まとめてモデリングアプローチ	● 方向づけフェーズで1回実施 ● UML のシーケンス図を使用	● 推敲フェーズの終わりに1回実施 ● CRC カードと UML のシーケンス図を使用	● 発展フェーズの終わりに1回実施 ● UML のクラス図を使用
こまめにモデリングアプローチ	● 方向づけフェーズで1回実施 ● CRC カードを使用	● 推敲フェーズ以降，必要に応じて反復の最初に実施 ● CRC カード及び UML のクラス図を使用	

3 に示す2つの方式を考案した.これら2つの実施方式を比較した場合,「まとめてモデリングアプローチ」には,「小まめにモデリングアプローチ」に対して，以下のような潜在的なメリットとデメリットがあると考えられる.

- 設計品質: まとめてモデリングした以降，変更や変更や拡張があった場合，その変更や拡張を設計モデリングしないことによる品質低下が発生する（デメリット）
- 設計モデリング時の指導: 設計モデリング作業を集中的に実施するため，指導者の確保がしやすい（メリット）
- 開発生産性: 設計モデリング作業を集中的に実施するため，プログラミング作業を継続的に行うことが可能になり，開発効率が上がる（メリット）

これら2つの方式の有効性を評価するためには, 2つの方式を実施し，これら3点の潜在的なメリットやデメリットが実際の開発にどのような影響を及ぼすか明らかにする必要がある.

一方，システムテスト作業については表 4に示す実施方法を考案した. この方式は，以下の 2 点の考え方に基づいて考案した.

- 開発の早い段階からインフォーマルなシステムテストを実施して不具合を顕在化させ，可能であればその不具合が修正されたことを確認するテストを自動化する
- 労力をある程度必要とする系統的なシステムテストを3回に集中させ，段階的に発展させることで人手を要するテストの負荷（コスト）を軽減する

ここで，インフォーマルなシステムテストは以下の2つの形態で実施する.

- 一般的な利用手順と想定される手順とテストデータを用いてシステムを操作し，期待される結果が得られるか確認する
- 任意の手順やテストデータでシステムを操作し，期待される結果が得られるか確認する

一方，系統的なシステムテストは以下の形態で実施する.

表 4 システムテストの実施方法

フェーズ名	実施内容
基本作成	● システムとしてある程度まとまった動作をし始めた反復からインフォーマルに実施 ● αリリース前の反復から系統的に実施
発展	● 通常の反復はインフォーマルに実施 ● βリリース前の反復は系統的に実施
移行	● 発展フェーズまでのテストケースをさらに拡充し，このフェーズを通じて系統的に実施

表 6 具体化された作業の有効性の評価方法

作業の評価項目	具体的な測定量
設計品質	● サイクロマティック複雑度(CC)が 4 より大きいメソッドの全メソッドに対する割合 ● メソッド凝集度の欠如(LCOM-HS)が 50 より大きいクラスの全クラスに対する割合
開発生産性	● 開発されたアプリコードの量をその時点までの累積開発労力で割った値
システムテストの有効性	● システムテストによる累積不具合検出数

- システムの利用パスのバリエーションや発生しえるエラーとシステムに期待される動作を一覧表にして，それらのケースを実際に実行し，システムが期待通りに動作するか確認する

以上述べた 2 方式の設計作業とシステムテスト作業の有効性を評価するために，表に示す3種類の測定を用いた． 開発途上でのこれら 3 種類の測定値の時間変化を求め，設計作業やシステムテスト作業の実施時期を対比することにより，実施した作業の効果を定量化できる可能性がある．

サイクロマティック複雑度は，McCabe のサイクロマティック複雑度 (Cyclomatic Complexity: CC)[7]を意味する．CC は，元々プログラムコードの分岐複雑度を定量化する測定であるが，ここではメソッド単位の分岐複雑度の測定値を用いている．分岐複雑度の高いメソッドは，クラスやメソッドの分割が適切ではないことに起因している可能性がある．

メソッド凝集度の欠如は Henderson-Sellers のメソッド凝集度の欠如 (Lack Of Cohesion Of Methods: LCOM) [7]を意味する．LCOM-HS は，1 つのクラス内のメソッドの凝集度の低さを百分率で表現する測定である．

開発生産性は，特定の時点までに開発されたプログラム全体のメソッドコード行数 (LOCm) をその時点までに投じたすべての種類の作業に渡る開発労力(人日)で割った値として求める．開発生産性を求めることで，モデリング作業による開発生産性の低下が顕著に発生するかどうか確かめることができる．

システムテストの有効性の評価については，システムテストによる累積不具合検出数を用いる．開発途上でのシステムテストによる累積不具合検出数を調べることで，各種のシステムテストの実施形態が不具合の検出に有効であるかを調べることができる．さらに以下の2つのテストについても同様の累積不具合検出数を測定することで，これらのテストと不具合検出率を比較することでシステムテストの相対的な有効性を示すことができる．

- 単体テスト及び統合テスト
- 受け入れテスト

4. 適用結果

前章で述べた AMRM の 2 つの設計実施方式を 2 つのプロジェクトに適用した． これらの 2 つのプロジェクトでは，前章で述べたシステムテストの実施方法が適用された．

表 5は, AMRM を適用した 2 つのプロジェクトで適用した設計作業の実施方式と開発の各フェーズで実行された反復の番号を示している．なお，本表において A プロジェクトの方向づけフェーズでは1回の反復が実行されたと記されているが，この反復は 1 ヶ月強の期間と例外的に長期間だった．それ以外の反復の期間はすべて 2 週間であった．また，使用する技術が少し異なるものの 両プロジェクトはともに社内利用のための Java アプリケーションを開発するプロジェクトであった．

表 5 AMRM を適用した 2 つのプロジェクトの概要

	A プロジェクト	B プロジェクト
設計作業の実施方式	まとめてモデリングアプローチ	こまめにモデリングアプローチ
方向づけ	#1	#1-#3
推敲	#2-#3	#4-#5
基本作成	#4-#8	#6-#10
発展	#9-#12	#11
移行	#14	#12

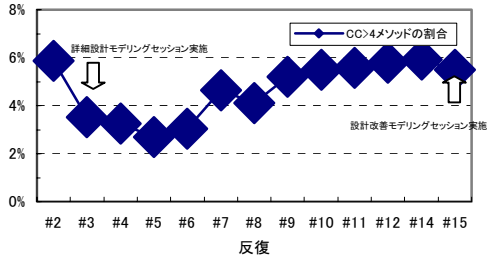

図 1 CC>4 であるメソッドの割合の推移 (A プロジェクト)

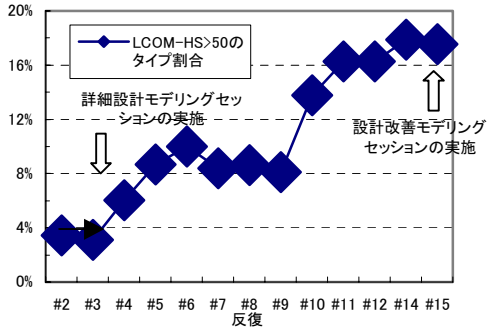

図 2 LCOM-HS>50 であるクラスの割合の推移 (A プロジェクト)

4.1. 設計作業

2 つのプロジェクトの設計品質は, Java のプログラムコードを Eclipse Metrics Plugin[8]で一次測定し, その測定値をデータベースと表計算ソフトにより集計や演算を行って求めた.

A プロジェクトでは, #3 反復で詳細設計モデリングセッションを集中的に実施した. さらに, #13 反復で設計改善モデリングセッションを実施したがそのモデリング結果は開発完了後の#15 反復で実装された. 詳細設計モデリングセッションは,モデリング指導者が3名の開発者とともに1機能を半日弱の時間でモデリングし, 残る 3 機能を 3 名の開発者が 3 日程度の時間でモデリングを行う形で実施された. また, 設計改善モデリングセッションは, モデリング指導者が既存の設計の問題点を指摘し, その改善案を 3 名の開発者が考案する形で行われた.

図 1 と 図 2 は, A プロジェクトにおける CC>4 のメソッド割合と LCOM-HS>50 であるクラスの割合の推移を示している. A プロジェクトの CC>4 のメソッド割合と LCOM-HS>50 であるクラスの割合は, 詳細設計モデリングセッションを実施して以降, 単調に悪化していることが分かる. また, #13 反復で設計改善モデリングセッションを実施した結果を実装した#15 反復において, 悪化した CC>4 のメソッド割合と LCOM-HS>50 であるクラスの割合が回復していないことが分かる.

#13 反復で設計改善モデリングセッションを行う際に, CC が 4 を大きく上回るメソッド及び LCOM-HS が 50 を大きく上回るクラスを調べたところ, 複数の関心事を扱っているような大きなクラスや当初の設計の凝集度を低下させるように拡張した部分が顕著に見られた. このような設計品質の低下は, 設計改善モデリングセッションでモデリングされた範囲から外れた#9 反復以降に発生したと考えられる. また, #15 反復で設計改善の効果が明確に現れないのは, 複数の関心事を扱っているような大きなクラスや当初の設計の凝集度を低下させるような拡張の改善コストが非常に大きく困難であることを表していると考えられる.

B プロジェクトでは, #4 反復で詳細設計モデリングセッションを実施し, 以降設計を拡張しなくてはならなくなった#7, #9 反復に詳細設計モデリングセッションを追加実施した. また, #7 反復以降のほぼすべての反復の開始時に CC が 4 を大きく上回るメソッドと LCOM-HS が 50 を大きく上回るクラスを 3-4 箇所選んで設計改善モデリングセッションを実施し, 改善案をその反復内で実装した. 設計改善の対象となったのは, CC が 4 を大きく上回るメソッドが大半であった. B プロジェクトでは, 詳細設計モデリングセッションは設計者 1 名が原案を作り, その原案を指導者がレビューする形で実施された. また, 設計改善モデリングセッションは, モデリング指導者が既存の設計の問題点を指摘し, 問題箇所の担当者が設計改善案を考えるという形で行われた. なお, B プロジェクトでは, #5 反復まで実装者(兼設計者)は 1 名であったが, #6 反復の途中から実装者が 1 名増えている.

図 3 と 図 4 は, B プロジェクトにおける CC>4 のメソッド割合と LCOM-HS>50 であるクラスの割合の推移を示している. B プロジェクトにおいても, CC>4 のメソッド割合は#7 反復以降悪化し, #9 反復で 8%程度に落ち着いている. 一方, LCOM-HS>50 であるクラスの割合は#10 反復に至るまで概ね 4%程度に抑えられている.

B プロジェクトで LCOM-HS>50 であるクラスの割合が低いレベルに留まっているのは, 詳細設計モデリングセッション及び設計改善モデリングセッションを適宜実

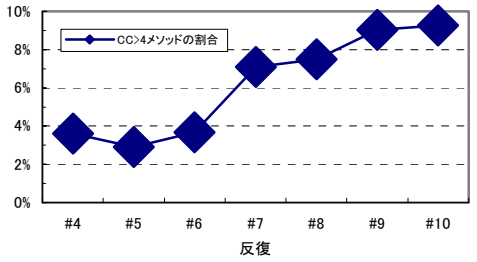

図 3 CC>4 であるメソッドの割合の推移
(B プロジェクト)

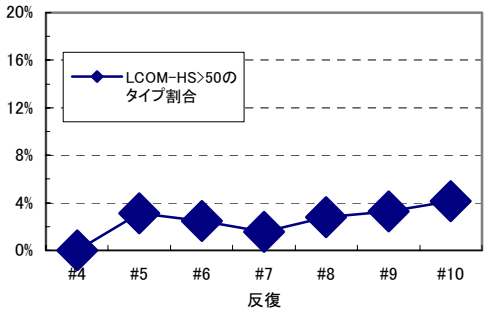

図 4 LCOM-HS>50 であるクラスの割合の推移
(B プロジェクト)

施したことの効果を示していると考えられる.

A, B プロジェクトにおいて, 各々#14 反復, #10 反復の時点までの開発生産性を測定した結果は以下のとおりであった.

- A プロジェクト: 35.2 LOCm/人日
- B プロジェクト: 36.0 LOCm/人日

B プロジェクトでは, #11 反復以降で系統的なシステムテストを実施したため, 開発完了時の開発生産性はこの測定値より若干(最大で 1 割低下)低下する可能性がある.

この開発生産性の測定結果から設計作業の実施方式による開発生産性の違いはほとんどないことが分かる.

A,B プロジェクトにおいて推敲フェーズ以降でモデリング指導者が費やした指導時間の合計は両プロジェクトともに概ね 2 日間程度だと考えられる.

これまで説明した A,B プロジェクトの測定結果から「まとめてモデリングアプローチ」は,「こまめにモデリングアプローチ」との比較で以下のように評価できる.

- メソッド凝集度の点でより良い設計品質が達成された
- 開発生産性やモデリング指導者の指導時間の点では,「まとめてモデリングアプローチ」とほぼ同等であった

4.2. システムテスト

A, B プロジェクトでは, システムテストの実施方法は同じだったが, テストカバレージ(C0 及び C1)を測定してテストの網羅性を可能な限り高めることを開始した時期が以下のように少し異なる.

- A プロジェクト: #7 反復
- B プロジェクト: #4 反復

また, 開発内容の点では以下のような違いがある.

- A プロジェクト: 入出力するファイルの形式が一定で不整合性が発生しにくい
- B プロジェクト: 入出力するファイルの形式の自由度が大きく, 不整合が発生しやすい

図 5 と 図 6 は, A, B プロジェクトにおいて3種類のテストで発見された累積不具合数の推移を示したものである. ここで受け入れテストされた不具合とは, 専任の開発者以外が発見した不具合もしくは専任の開発者がデモで様々な操作を行っている際に発見された不具合を意味する.

これらの不具合の推移は, 以下のような傾向を示していると考えられる.

- 単体テスト: テストの網羅性を可能な限り高めることを開始した時点あたりで不具合数が増え, その後飽和する
- システムテスト及び受け入れテスト: 基本作成の後半から不具合数が増加し, その後飽和する

これらの結果から以下の 2 点が分かる.

- 不具合の検出数は, システムテストが最も高く, 単体テストが2番, 受け入れテストが最も低かった

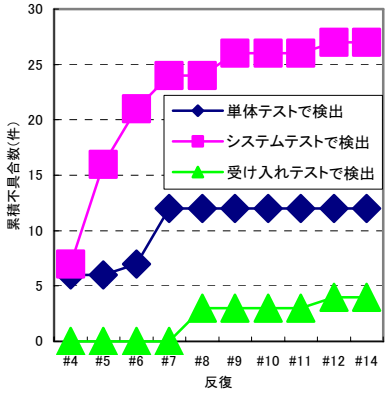

図 5 累積不具合数の推移
(A プロジェクト)

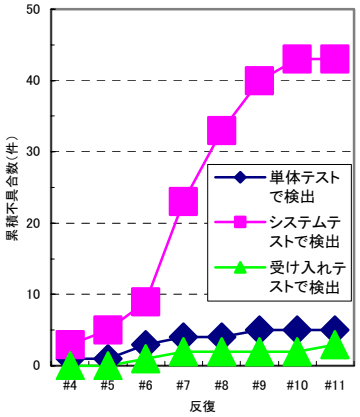

図 6 累積不具合数の推移
(B プロジェクト)

- システムテストによる不具合の大半は，インフォーマルなシステムテストによって検出された

すなわち，ソフトウェアの品質を開発の早い段階で高めるためには AMRM で行っているようなインフォーマルなシステムテストを早期に実施することが有効であるということが分かる．また，ソフトウェアの品質を高めるためには単体テストや統合テストの自動化とシステムテストを効果的に組み合わせることが大事であることが分かった．

5. まとめ

本論文では，5 つのフェーズで開発ライフサイクルを構成するアジャイル開発手法 AMRM の設計作業とシステムテスト作業の具体的な実施方法を考案し，開発プロジェクトに適用した結果を報告した．

適用結果から，設計作業を推敲フェーズ以降に適宜実施する「小まめにモデリングアプローチ」が設計品質を高めるために有効であることが示された．また，ソフトウェアの不具合の発見には，インフォーマルなシステムテストを早期に実施することが有効であることが示された．

参考文献

[1] アリスター・コーバーン，*アジャイルソフトフェア開発*，ピアソン・エデュケーション，2002

[2] ケント・ベック，*XP エクストリーム・プログラミング*，ピアソン・エデュケーション，2000

[3] 藤井他，*初心者モデラーによるアジャイル開発の実践事例報告*，情報処理学会研究報告 2005-SE-149, pp.89-96, 2005

[4] フィリップ・クルーシュテン，*ラショナル統一プロセス入門*，ピアソン・エデュケーション，2001

[5] 藤井他，*普通のプロジェクトへの適用を目指したアジャイルな開発手法の構築と適用結果*，情報処理学会研究報告 2004-SE-145, pp.15-21, 2004

[6] Rebecca Wirfs-Brock and Alan McKean, *Object Design – Roles, Responsibilities, and Collaborations*, Addison-Wesley, 2003

[7] Brian Henderson-Sellers, *Object-Oriented Metrics – Measures of Complexity*, Upper Saddle River, 1996

[8] http://www.teaminabox.co.uk/downloads/metrics/index.html