

## 初心者モデラーによるアジャイル開発の実践事例報告

藤井 拓<sup>†</sup>, 鶴原谷 雅幸<sup>†</sup>

本論文では、アジャイル開発手法に、開発の大局的なリスク管理を行うためライフサイクルに統一プロセスと同様の4フェーズを導入し、開発の実現性や開発作業の内容の把握のためにCRCカードとUMLを組み合わせた3種類のモデリング作業を導入することを提案している。この開発手法の特徴は、アジャイル開発のスピードを落とさず、しかもモデリングの初心者でも実践できるようにモデリング作業を取り入れた点である。また、モデリング作業の効果を測定するために既存の設計や実装の品質評価手法に基づく簡易な設計品質方法を提案している。さらに、初心者モデラーからなる開発プロジェクトに本手法及び設計品質の測定方法を適用した結果を報告する。

## A Case Study of Agile Development with Inexperienced Modelers

Taku Fujii<sup>†</sup>, Masayuki Tsuruharaya<sup>†</sup>

This paper proposes an agile development method that incorporates 4 phases in development lifecycle for risk management in long range and also incorporates 3 modeling activities based on CRC card and UML for feasibility of development and understanding of development tasks. This method is designed not to slow down development speed and to be easy for inexperienced modelers. Also, a simple measurement system based on existing measurement techniques of design and program is proposed to evaluate effectiveness of the modeling activities. A case study of this method and measurement system on a project composed with inexperienced modelers is reported.

### 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]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。

これらのアジャイルな開発手法の中で最も知名度が高いのが Beck らにより提案された XP (eXtreme Programming)[2]という手法である。XP の設計のプラクティスは、実装作業と設計作業を融合させた形で行うためのものであり、以下のような実装に先行してモデルを作るという従来の設計作業は明示的には行われない。

- A) ソフトウェアの実現方式を高いレベルで決めるために、アーキテクチャのモデルを作る
- B) 開発すべき内容の仕様を決めるために、実装前に詳細設計モデルを作る

前者は、XP のライフサイクルの最初に設定されるアーキテクチャスパイクと呼ばれる期間で検討される可能性があるが、そこでどのようなモデルが作られるかは明らかではない。また、XP では各反復で実現すべきユーザストーリー(end-to-end の機能)を割り当てられた各ペアが一つのソースコードベースを同時に拡張することが想定されている。そのような開発方式は、基本的には各開発者あるいはペアのうち最低一人が暗黙的に詳細設計を行うスキルを持っていることを仮定している。しかし、世の中の一般的な開発プロジェクトを考えた場合、設計を行うスキルを持っている開発者の方が少数派であり、XP の仮定は現実的ではないと考えられる。

本論文では、ソフトウェアの実現性を早い段階で評価したり、開発すべき内容の仕様を具体化するためにアジャイル開発手法に設計モデリングを取り入れる方式 AMRM (Agile Method Reinforced with Modeling)を

<sup>†</sup>(株)オーグス総研技術部ソフトウェア工学センター

提案し、その開発プロジェクトへの適用結果を報告する。AMRM の特徴は、以下の2点である。

- 開発ライフサイクルに4つのフェーズを取り入れ、フェーズ毎に異なるモデリング作業形態を取り入れた
- モデリングの初心者でも実践しやすい協調的なモデリング手法を用いている

以降の章では、これらのモデリングをどのように取り入れたかを説明する。ついで、結果として得られた設計モデルの評価手法について説明する。さらに、AMRM を実際のプロジェクトに適用した結果を紹介する。最後に、AMRM の有効性評価と今後改良すべき点について論じる。

## 2. モデリングの取り入れ方

### 2.1. フェーズとモデリング作業

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[3]では、開発ライフサイクルに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4つのフェーズを設定した。なお、AMRM では各反復での目標設定等の進め方等の基本的なプラクティスについては、文献[4]に記したものをを用いている。

表 1 本研究で設定した4つの開発フェーズ

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
推敲	アーキテクチャの確定
作成	機能の作りこみ
移行	ソフトウェアの評価，改良，導入成果物等の作成

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のために必要な期間実施され、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。推敲フェーズ以降では、フェーズの目的の達成を長期目標として、短期的には反復単位で追加すべき機能やその他の目標を設定し、プログラムの拡充及びテストを行う。

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。

筆者らは、このような4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のような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ブ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。

表 2 本研究で提案する3種類のモデリング作業

適用時期	モデリング作業名	モデリング作業内容	所要時間
方向づけフェーズ後半	概念設計モデリングセッション	システムの主要機能について内部機能の相互作用をUMLのシーケンス図で大雑把にモデリングする	1 主要機能につき 0.5-1 日間
推敲フェーズ終盤	詳細設計モデリングセッション	システムの主要機能について相互作用の詳細をUMLのコラボレーション図でモデリングし、クラスに送信されるメッセージをCRCカードに書き込む	1 主要機能につき 0.5-1 日間
作成フェーズ終盤	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 1-2 日間



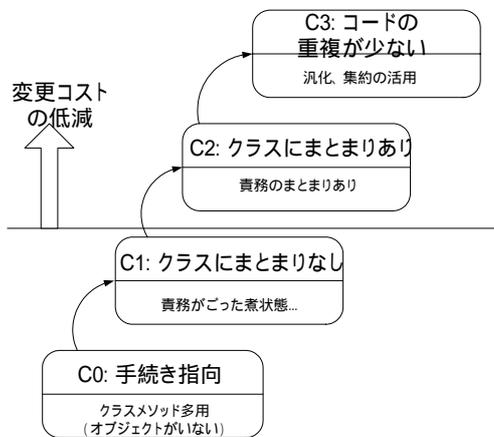


図 2 クラスの設計品質のレベル

CRCカードやUMLを用いてモデリングを行うことにより、C0やC1のメソッドやクラスの割合が減少すれば、そのモデリングの有効性が示せると考えられる。

### 3. 設計品質の測定手法

本研究では、前節で説明したC0及びC1に相当する設計部分の割合を、表3に示されるいくつかの測定量の閾値を超えるものの割合で定量化した。

C0の検出に用いたMcCabeのシクロマティック複雑度(CC)[6]は、以下の式で求まる測定量である。

$$CC = e - n + 1$$

n: メソッド内の制御フローのノード数

e: メソッド内の制御フローのエッジ数

CCは、元々は分岐複雑度を定量化する測定量であるが、オブジェクト指向設計やプログラムが手続き的である度合いを示す指標として用いられている。これは、複雑な責務を複数のより小さな責務や協調に分解せず1つのメソッドに実装すれば結果として分岐複雑度が高まるとの考えに基づいている。メソッドの命令行数(NOS)も同様の理由で手続き的なメソッドの検出に用いられる。また、クラス操作は一般的に手続きとデータをモジュール化したインスタンスを基本とするオブジェクト指向の考えから外れるものであり、クラス操作の割合の高さは手続き的であるという度合いを示していると考えた。

一方、C1の検出のためには、クラスのメソッド間凝集度の欠如(LCOM)と他のクラスに対する関係数(Ce)を用いた。LCOMは、属性とメソッドの対応関係から1つのクラス内のメソッド間の凝集度の欠如を定量化する測定量である。LCOMについては複数の測定方法が提案されているが、本研究ではHenderson-Sellersの提案した以下の計算式[6]に基づく測定方法を用いた。

$$LCOM - HS = \frac{\left( \frac{1}{a} \sum_{j=1}^a \mu(A_j) \right) - m}{1 - m}$$

m: メソッド数,  $A_j$ : j番目の属性, a: 属性数  
 $\mu(A_j)$ : j番目の属性にアクセスするメソッド数を求める関数

表 3 C0及びC1に相当する品質の設計部分の検出方法

測定対象	測定手段	閾値
手続き型のコード(C0)	メソッドの命令行数 (NOS: Number Of Statements)	20
	McCabeのシクロマティック複雑度 (CC: Cyclomatic Complexity)	4
	クラスメソッド (CM: Class Method)	
クラスのまとまりのなさ (C1)	クラスのメソッド間凝集度の欠如 (LCOM: Lack of Cohesion Of Methods)	40
	他のクラスに対する関係数 (Ce: Efferent Coupling)	25

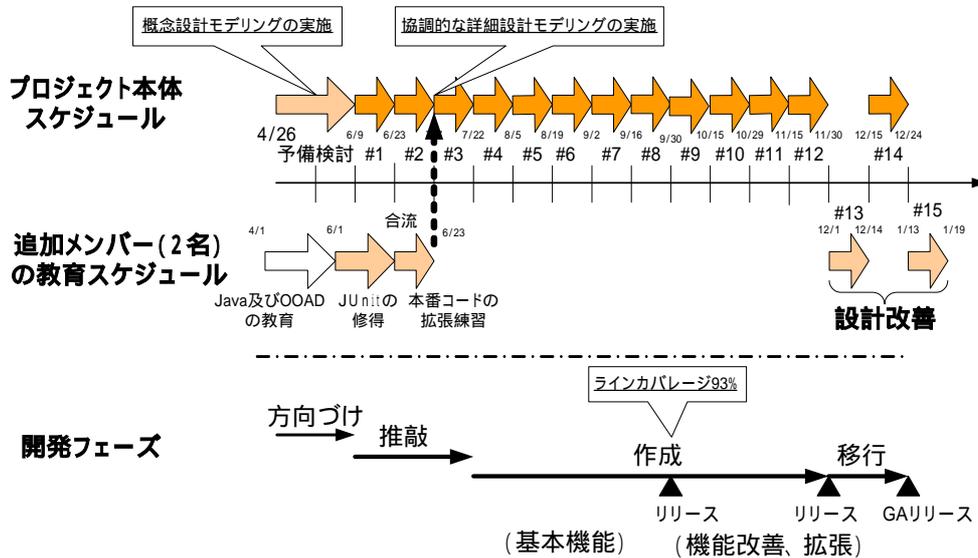


図 3 開発スケジュールとモデリング作業の適用時期

Ce は、各クラスの他のクラスへの関係による参照数を測定したものであり、協調が多すぎて蛸足状態のクラスを特定するために有効な測定量である。蛸足状態のクラスは、変更の影響を広範囲に波及させる可能性が高いため、変更コストを抑制する設計である基本的な条件だと考えられる。

本研究では、表 3 の C0 に対する各測定量において閾値が示されているものについては閾値を越えたメソッド数を計測し、それらのメソッド数のプログラムコード全体でのメソッド数に対する割合を求めることによりプログラムコードの C0 に相当する部分の割合の定量化を行った。クラスメソッド数については、直接プログラムコード全体でのメソッド数に対する割合を求めた。また、C1 の各測定量については、測定量が閾値を超えたクラス数のプログラム全体のクラス数に対する割合を求めることによりプログラムコードの C1 に相当する部分の割合の定量化を行った。

Java プログラムが測定対象の場合、表 3 に挙げられているクラスメソッド数以外の測定量は無償提供されている Java の統合開発環境である Eclipse の中で無償提供されている Eclipse Metrics Plugin[7]により求めることができる。また、表 3 の測定量に対する閾値は Eclipse Metrics Plugin において初期設定されている値を採用している。

#### 4. 適用結果

本研究で提案したモデリング手法を社内向けのツール開発において適用した。開発対象のソフトウェアは、Java 言語で実装され、Java の分散オブジェクト技術である RMI(Remote Message Invocation)や Java の XML パーサなどの技術を適用している。

図 3 は、モデリングの手法を適用したプロジェクトのスケジュールの概要とモデリングの適用時期を示したものである。開発メンバーは、推敲フェーズの反復#2 まで専任メンバー兼プロジェクトリーダー1名と要求定義者兼コーチ1名(兼務)の体制で進行し、反復#3 で専任メンバー2名を増員し、反復#3 から反復#8 まで3名の専任メンバーと要求定義者兼コーチ1名(兼務)で開発を行った。基本的な機能の実装が完了した反復#9 以降、専任メンバーが1名減少している。

また、図 3 に示されるように方向づけフェーズで概念設計モデリングを実施し、推敲フェーズの反復#3 で詳細設計モデリングセッションを実施した。また、反復#13 と反復#15 では設計改善モデリングセッションを実施し、反復#12 までに開発したコードに対する設計改善を試みた。

開発メンバーで要求定義者兼コーチ1名を除く3名は、オブジェクト指向分析設計のトレーニングは受講したものの実開発でのモデリング実践経験はほとんどない状態であった。そのため、コーチ1名が概念設計モ

デリングでモデルをレビューしたり、詳細設計モデリングセッションや設計改善モデリングセッションの実施初期段階にモデリングの実演を行う形で指導を行った。

図 5、図 6、図 4はモデリング手法を適用したプログラムコードをEclipse Metrics Pluginで測定した結果を示す。

図 5に示されるクラスメソッドの割合は、反復#2において 50%を超えていたが反復とともに単調に減少し、反復#9では 1.2%に低下した。図 6から、C0に相当する手続き指向のメソッドの割合が詳細設計モデリングセッションを実施した反復#3以降で減少したものの、反復#6以降で再び増加していることが分かる。また、図 4から反復#3ではメソッドの凝集度が低いクラスの割合は低いものの、反復#4以降で徐々に増えていることが分かる。図 4において、反復#2でのメソッドの凝集度が低いクラスの割合は一見低いように見えるが、これは LCOM-HS の測定からクラスメソッドが除外されているため見かけ上良くなっているものと考えられる。

また、反復#13及び反復#15では反復#13で行った設計改善モデリングセッションを行った結果の一部を実装したが、図 6、図 4に示されるように CC>4 のメソッドや LCOM-HS>50 のクラスの割合はほとんど減少しなかった。

これらの測定と同時に、他のクラスに対する関係数が 25 を超えるクラスの検出も試みたが、反復#2の時点で1クラス検出されたのみに留まり反復#3以降は検出されなかった。

これらの測定結果から詳細設計モデリングセッションにより、モデリング直後に全体的な設計品質は改善したものの、モデリングした時点以降で設計品質が再び低下したことが分かる。このような品質低下は、反復#3で作成された設計モデルにおいて反復#5以降で足りない機能がいくつかあるのが判明し、それらを実現するために設計部分を拡張した箇所において発生した。さらに、設計品質の低下が生じている部分を調べた結果、以下の 2 種類の形で品質が低下していることが判明した。

- 既存クラスへのまとまりのない責務の追加
  - 責務にまとまりがあった既存のクラスに、まとまりのない責務を追加した
- 責務がごった煮の大きなクラスの追加
  - #10以降に当初に予定していなかった機能を追加するためのクラスが、粒度が大きく責務がごった煮になっていた

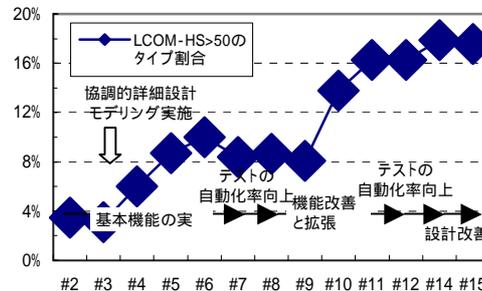


図 4 LCOM-HS>50 であるクラスの割合の推移

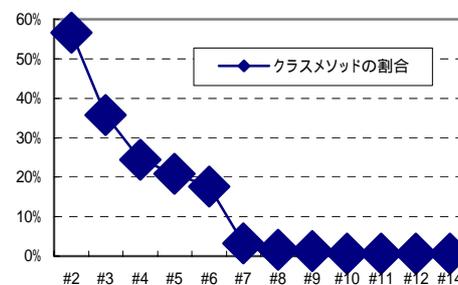


図 5 クラスメソッドの割合の推移

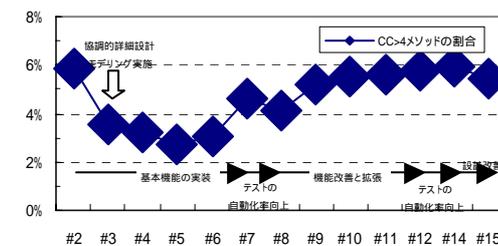


図 6 CC>4 であるメソッドの割合の推移

このような当初の設計に対する設計拡張における設計品質の低下は、以下の 3 点においてスキルが不足していたことが原因で発生したと考えられる。

- A) 責務を追加すべき場所を特定できない
  - 追加すべき機能を既存のクラスに責務を追加すべきか、新規のクラスを追加すべ

きかについて適切な判断ができない

- B) 既存のクラスのリファクタリングが判断できない
  - コラボレーションパスが増えるにつれて既存のクラスの分割や責務の再配置も考えるべきだったが、そのような可能性を考えられなかった
- C) 責務内部の処理フローが見通せない
  - ハイレベルの責務は定義されるが、ハイレベルの責務に必要な一連の低レベルの責務を担うクラスを追加されない。その代わりに、メソッドの実装の複雑度を増すことで吸収しようとする

A)については、クラスの責務を個別の責務レベルでしか捉えられず、クラス全体の目的や関心事というレベルでの整合性について考慮できなかったということを意味する。B)については、協調クラス側の責務を括り直すことにより、複数のコラボレーションに同時に対応できるようなクラスを見つけることができなかったということである。C)については、責務の実現のために必要な処理フロー(コラボレーション)を実装に先行して見通せなかったために発生したということである。

これらの3点のスキル不足を克服するためには、以下のように開発途上でモデリングの教育や支援を行う必要があるのではないかと考えられる。

- A) 責務を追加すべき箇所の特定
  - 責務だけではなくクラス全体の目的や関心事を意識するようにモデリングの進め方を改良する必要があると考えられる。すなわち、詳細設計モデリングセッションではコラボレーション図とCRCカードの責務や協力者クラスの欄だけを用いてきたが、一旦CRCカードとして見出されたクラスの目的や関心事も記入していくことが望ましいと考えられる
- B) 既存のクラスのリファクタリング[8]の判断
  - 実際のコードを題材に、協調クラス側の責務を括り直すことにより、複数のコラボレーションに同時に対応できるような実例を示す
- C) 責務内部の処理フローのモデリング
  - ソフトウェアが依存するライブラリやフレームワークのAPIレベルのクラスの責務呼び出しに至るまで処理のフローをモデ

リングすることを徹底する

これらA),C)については、既に行った詳細設計モデリングセッションの延長として理解できる作業でもある。そのため、モデリングの初心者でも複数回の指導により習得できる可能性がある。それに対して、B)についてはA),C)よりもさらに抽象化するスキルを求められる。そのため、モデリングの初心者が短期間に習得するのは困難ではないかと思われる。結果として、モデリング初心者のチームの場合にはB)を実現するためにリファクタリングのスキルを持つ技術者の支援が必要になると考えられる。

## 5. 有効性の評価と課題

これらの測定結果及びプロジェクトの進行状況から概念設計モデリングセッション及び詳細設計モデリングセッションの有効性は以下のように評価できる。

- 概念設計モデリングセッション: アーキテクチャで実現すべき機能と相互作用の大筋が明らかになったことで、後続するアーキテクチャ上のリスクを洗い出すことができた。相互作用の大筋を詳細設計モデリングセッションの出発点にすることにより、モデリング作業が効率化した。
- 詳細設計モデリングセッション: 概念設計モデリングセッション以降に実装したコードの設計品質を改善し、結果的に実装作業も効率化した点で有効だと考えられる。
- 設計改善モデリングセッション: 作成フェーズで追加された責務がごった煮状態の大きなクラスを2回の反復程度の設計改善で改善できなかった。設計改善よりもC1やC0のクラスを予防することの方が有効だと考えられる。

今回の手法では、モデリングの指導者の介在を減らすために推敲フェーズ終盤と作成フェーズ終盤に詳細設計モデリングセッションと設計改善モデリングセッションを1回ずつ実施することを提案した。しかし、結果的には詳細設計モデリングセッションの実施以降に設計品質が徐々に低下してしまった。そのような低下を防ぐためには、反復毎に半日程度の詳細モデリングセッションを設けて、前節で論じたような指導を行うとともに、拡張機能の設計への追加方法をチーム内で議論するようにすることが有効ではないかと考えられる。

## 6. まとめ

本論文では、アジャイルなソフトウェア開発に統一プロセスと同様の4フェーズ(方向づけ, 推敲, 作成, 移行)を導入することを提案している。さらに, 方向づけ, 推敲, 作成フェーズで開発に伴うリスクを低減したり, 設計品質を向上させることを狙った3種類のモデリング作業を導入することを提案した。すなわち, 方向づけフェーズの後半で要求内容を検証するための”概念設計モデリングセッション“を実施し, 推敲フェーズ終盤で責務の分割の点で良好な品質な設計を作るための”詳細設計モデリングセッション“を実施し, 作成フェーズ終盤で開発途上において劣化した設計品質を改善するための”設計改善モデリングセッション“を実施する。これらのモデリング作業は, 概ね半日から数日間で実施することができ, モデリング経験がほとんどない開発者でも少ない指導で実践できるように考案された。

本研究で提案した3種類のモデリング作業をモデリングの初心者から成る社内向けソフトウェア開発プロジェクトに適用した。その結果, 詳細設計モデリングセッションを実施することにより, 手続き的なコードを減少させたり, クラスに対する責務の割り当てを向上するなど設計品質の改善が達成された。その一方, 詳細設計モデリングセッションで得られた設計を拡張した部分において設計品質が低下する傾向が見られた。この設計品質の低下は, ”既存のクラスへのまとまりのない責務の追加“や”責務がごった煮状態の大きなクラスの追加“により発生した。開発終盤で”設計改善モデリングセッション“を実施することで, 低下した設計品質を回復しようと試みたが, ほとんど回復できなかった。

これらの結果より, 開発の実現性を確認したり, 設計品質を向上させるうえで, 概念設計モデリングセッション及び詳細設計モデリングセッションの実施が有効であることが確認できた。但し, 詳細設計モデリングセッション及びモデリングの指導を推敲フェーズの終盤だけに実施したことが, 推敲フェーズ以降の設計品質低下を招いたと考えられる。また, 今回の事例からクラスの責務のまとまりがある程度確保されない設計品質では設計改善モデリングセッションは有効ではないことが分かった。

これらの問題点を解決するためには, 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の実施において以下のような改良を行う必要があると考えられる。

A) 推敲フェーズ以降, 反復毎に半日程度の時間

で小まめに実施する

- B) 責務やコラボレーションだけに注目するのではなく, クラスの目的, 関心事も明らかにする
- C) ハイレベルなクラスや責務の抽出に留まらず, APIなどの低レベルのクラスとのコラボレーションまで明示的にモデリングする
- D) 既存のクラスのリファクタリングの判断は, モデリングの指導者が支援する

今後, これらの改良点を実際に適用して有効性を確認したいと考えている。

謝辞

本研究で提案しているモデリング作業の実開発への適用においてご協力頂いた同僚の矢田貴也さん及び舩屋康典さんにこの場を借りてお礼をさせていただきます。

## 参考文献

- [1] 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
- [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] 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
- [4] 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
- [5] Rebecca Wirfs-Brock and Alan McKean, *Object Design – Roles, Responsibilities, and Collaborations*, Addison-Wesley, 2003
- [6] Brian Henderson-Sellers, *Object-Oriented Metrics – Measures of Complexity*, Upper Saddle River, 1996
- [7] <http://www.teaminabox.co.uk/downloads/metrics/index.html>
- [8] マーチン・ファウラー, *リファクタリング: プログラミングの体質改善テクニック*, ピアソン・エデュケーション, 2000