解説

アジャイルなソフトウェア開発におけるモデリング

Modeling Approaches
in Agile Software Development

藤井　拓
（株）オージス総研ソフトウェア工学センター
taku@otc.ogis-ri.co.jp

従来のソフトウェア開発方式とモデル

近年，開発依頼者との間や開発チームのメンバ間の協調関係を重視し，顧客のニーズに即したソフトウェアを開発することを提案するアジャイルなソフトウェア開発が注目されている．アジャイル（agile）とは，「機敏な」という意味であり，アジャイルなソフトウェア開発とは，顧客の要求の変化に「機敏に」対応する開発方式を提案するものである．本稿では，従来の開発方式におけるモデリング作業の位置付けや問題点を説明し，モデリングにおいてアジャイルなソフトウェア開発が提案する解決策を説明する．そのために，まず従来の開発方式におけるモデリング作業の位置付けを説明する．

中規模以上の商業的な開発プロジェクトでは，以下のような作業を行うことで従来開発を進めてきた．

- **要求**：開発依頼者がソフトウェアに行って欲しいことを定義する
- **分析**：要求を開発者が整理する
- **設計**：要求の実現方法を定義する
- **プログラミング**：設計内容をプログラミングする
- **テスト**：作成されたソフトウェアが要求と合致するか確認する

さらに，これらのプロジェクトの多くでは，**図-1** に示されるようにこれらの作業を順次 1 回ずつ実行する「順次的な開発の進め方」を用いてきた．

このような開発の進め方の形式とその中で実行される作業内容の組合せを，本稿では開発方式と呼ぶ．

このような順次的な開発の進め方に基づく従来の開発方式では，要求，分析，設計などの作業の作業結果を以下のような形式で表現し，次の作業に伝達する．

- テキスト
- 図

要求，分析，設計の内容を図で表現する記法の例としては，1997 年に OMG（Object Management Group）という標準化団体により策定された UML（Unified Modeling Language）という仕様が有名である．UML は，

中規模以上の商業的なソフトウェア開発プロジェクトにおいて近年でかなり普及してきた．しかし，ソフトウェア開発プロジェクト全体を考えた場合には UML を使うプロジェクトはまだ少数派にとどまっており，大多数のプロジェクトではテキストを要求，分析，設計の情報伝達の主な手段として用いている．本稿では，これら図やテキストで表現した要求，分析，設計の情報をソフトウェア開発のための「モデル」と呼ぶ．

順次的な開発の進め方で分析や設計などのモデルを作る目的は，要求内容の不明確な点や設計上の問題点をプログラミング以前に取り除くことである．結果として，多くの労力を必要とするプログラミングにおける試行錯誤や修正を減らせれば，全体的な開発効率を向上させることができる．そのようなモデルを用いた問題発見を助けるため，モデルは必要な情報を網羅し，一定の期間を

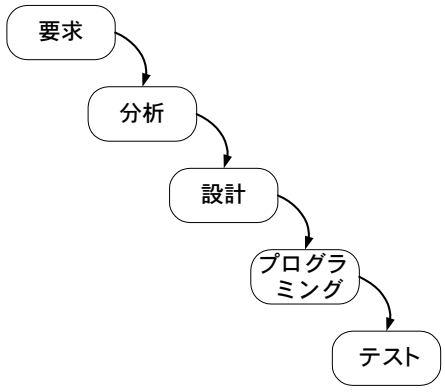

図-1　順次的な開発の進め方

かけて整合性よく作ることが良いとされてきた.

一方，プログラミングに先駆けて要求や設計をモデリングするためには，前の作業で集められた情報だけを頼りに概念やアイディアをモデリングする必要がある．しかし，そのような概念やアイディアのモデリングはプログラミングのように具体的な作業より難しく，分析者やシステムエンジニアと呼ばれるような専門家が必要だと考えられてきた.

このような従来のソフトウェア開発におけるモデルの考え方をまとめると，以下のようになる.

- テキストや図に基づくモデルを使う（テキストが多い）
- 開発作業はモデルを受け渡して進行する
- モデルを作るのは専門家である

従来のソフトウェア開発の問題点

前章の説明で，ソフトウェア開発においてモデルを作成することが開発生産性の向上につながると思われたかもしれない．しかし，現実はそう単純ではない．モデルの作成が開発生産性の向上につながるためには，以下のようにいくつか前提となる事柄がある.

(A) 要求を確定した後に，開発依頼者の要求が大きく変わらないこと
(B) プログラミング以前に，妥当な設計を考えられること

(A) は，開発依頼者の要求は開発初期に確定し，開発中変わらないという仮定である．しかし，現実には開発依頼者が作成されたソフトウェアの動作を見ることにより，ソフトウェアに求めるべきことが明らかになり，作成されたソフトウェアに対する大幅な修正や作り直しが発生することも多い．大幅な修正や作り直しは顧客満足度の低下を招き，開発依頼者と開発者の信頼関係も損なうため，ビジネスの上では致命的な問題となる．このような作成されたソフトウェアに対する大幅な修正や作り直しは，開発依頼者の要求の変化を表すものと考えることができる．また，開発中のビジネス状況の変化や開発技術の変化によって要求する内容が変わることもある．結局，順次的な開発の進め方は要求が開発初期に確定されることを前提としており，このような要求の変化への対応は困難である.

(B) は，プログラミング作業の内容を先行して設計で考えられるという仮定である．たとえば，現在開発で盛んに用いられている Java や C++ などのオブジェクト指向プログラミング言語を用いる場合，複数の手続きが集まったクラスを単位にして設計を行う．そのような設計においてクラスの定義の妥当性を確認するためには，クラスが他のクラスからどのように呼び出されるか，すなわちクラス間の相互作用を想定して考える必要がある．しかし，このような相互作用をモデル上で全部記述し尽くすことはきわめて困難である．そのため，現実的には代表的な相互作用だけを考えてクラスの定義を決め，プログラミングを開始することになる．相互作用の抜けや変更によりプログラミングの段階で設計を大きく変更せざるを得なくなったら，設計作業は必ずしも開発生産性を向上させるためには有効ではないことになる.

また，商業的な開発プロジェクトでは，近年プロジェクトの小規模化や短期間化が進行し，10 名以下の小規模チームで 6 カ月未満の期間で開発を行わなければならない場合が大勢を占めてきている．そのような小規模開発プロジェクトでは，要求，分析，設計の専門的スキルを持つ開発者を確保することは困難である.

一方で，行き当たりばったりの開発方式では高品質で顧客満足度の高いソフトウェアを開発することはできないことも確かである．そのような状況の中で，ソフトウェア開発においてモデルを作ることの意義や作り方が改めて問われている.

アジャイルなソフトウェア開発

前章で述べたように，順次的な開発の進め方は要求の変化に対応するのが困難だという点が大きな短所である．この短所を克服する方法として期待されているのが**図-2** に示す反復的な開発の進め方である．反復的な開発の進め方は，開発依頼者から要求を聞き，その要求内容を一定の期間で要求，分析，設計，プログラミング，テストを実行し，結果として作成されたソフトウェアを開発依頼者に示すことを 1 つの「反復」とし，これを繰り返すことにより進行する．このような反復的な開発の進め方を用いることにより，反復を単位にして開発依頼

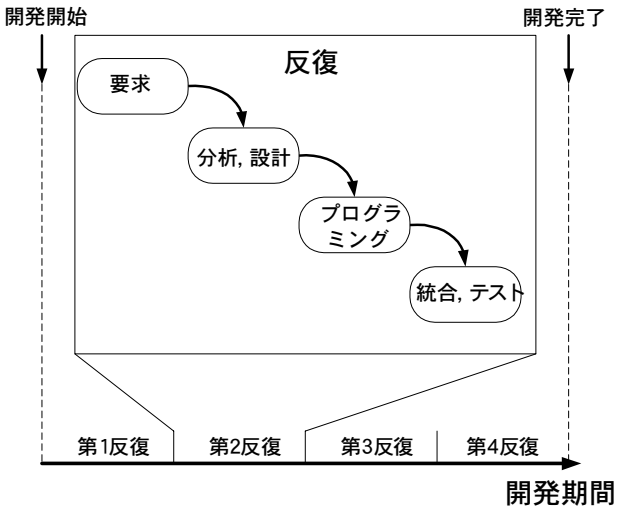

図 -2　反復的な開発の進め方

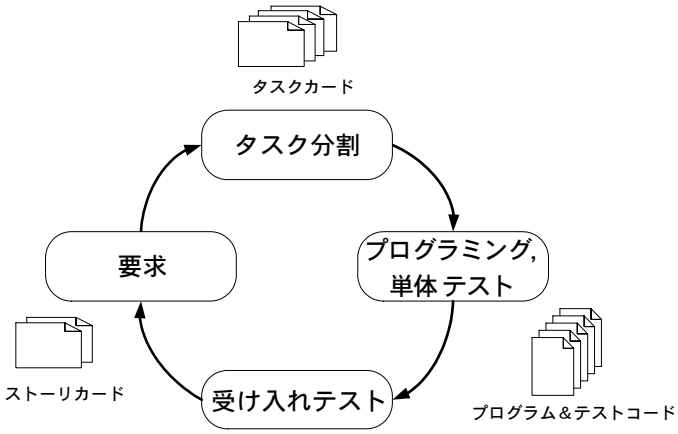

図 -3　XP の開発サイクル

者のフィードバックや要求の変化をソフトウェアに反映することができる.

しかしながら，順次的な開発の進め方を前提とした開発組織が反復的な開発の進め方に移行する場合，以下のような点が問題になる.

(A) 開発要員の稼働率の低下
(B) モデルを更新する手間の増加

(A) は，要求，分析，設計などの専門的スキルが必要とされる作業が繰り返し現れ，プロジェクトの大半を占めるプログラミングの作業要員が待ち状態になってしまうということである. (B) は，要求の変化に対応して要求，分析，設計などのモデルを更新しなければならないオーバヘッドが生じるということである.

アジャイルなソフトウェア開発とは，要求の変化に対応した開発を実現するために「反復的な開発の進め方に基づき，自律的に判断する開発者が良好なチームワークで開発を行うこと」を提案するものである. 自律的な判断を重視するため，アジャイルなソフトウェア開発では開発の進め方を事細かにマニュアル化するのではなく，価値，原則，プラクティスなどの作業指針により導こうとしている. すなわち，開発チームが共有すべき価値を定め，そのような価値を尊重した場合に作業の実践において守るべき指針や心得を原則やプラクティスとして定めている. このようなアジャイルなソフトウェア開発を実現する開発方式としては複数のものが提案されており，それらの開発方式の提案者が共通に合意する価値，原則などを Agile Manifesto[1] という宣言として発表している.

以降，アジャイルなソフトウェア開発を実現する開発方式で最も有名な XP（eXtreme Programming）[2] とアジャイルなソフトウェア開発におけるモデリングの作業指針を提案する AM（Agile Modeling）[3] の両者のモデリングに対する考え方を紹介する.

XP におけるモデリング

XP は，1999 年に Beck らが文献 2）にて発表したアジャイルなソフトウェア開発を実現する開発方式である. XP は，開発者 2 人のペアで開発を進めるペアプログラミングを推奨するなど多くの特色を有するが，大雑把には**図 -3** に示されるように 4 つの作業で進行する. 各作業の内容は，以下のとおりである.

(1) **要求**：開発依頼者の要求をその優先度とともにストーリカードというインデックスカードに順次記し，カードごとに見積もりを行い，今回の反復の開発範囲を決める
(2) **タスク分割**：開発者が集まって各ストーリカードを実現するために必要な開発作業をタスクカードというインデックスカードに書き出し，各開発者ペアが担当する作業範囲を決める
(3) **プログラミングと単体テスト**：各開発者ペアがタスクカードに対応するプログラミングを行い，作業範囲が完了したことを単体テストで確認する
(4) **受け入れテスト**：作成されたプログラムにおいてストーリカードで合意した開発範囲が実現しているか確認する

ここで，インデックスカードとは図書目録などを作るために使われる紙のカードのことである. これら (1) から (4) のサイクルを 1 回実行することを XP ではイテレーション（反復）と呼ぶ. XP では，1 回のイテレーションを 2 ～ 3 週間の短期間で実行する. このよう

な短期間でイテレーションを行うことにより，要求の変化にきめ細かく対応し，開発依頼者の要求に即したソフトウェアの開発が行えるとされている．

XP の反復的な開発の進め方では，モデリングを以下のように行っている．

(A) 要求，プログラミング，テストを中心として反復的に開発を進める
(B) 開発途上でモデルをできるだけ作らない
(C) 要求やタスクの分割においてインデックスカードを積極的に利用している

XP では，分析，設計の作業を行わないことにより，プログラミング作業者が待ち状態になることを防いでいる．また，開発途上で極力説明やモデルを作らないことで，それらを維持するオーバヘッドをなくしている．結果として，プログラミングを行う途上で要求内容の矛盾，不整合，不明点が発見されるが，それらは開発依頼者に適宜質問して解決する．

また，XP の考え方は，UML のようなある程度の専門的なスキルを求める記法を用いて分析や設計を行うことに否定的である．その一方で，XP の作業の要求，タスク分割ではインデックスカードが開発依頼者と開発者の間，あるいは開発者間で協調的に要求や作業範囲を決めるのに用いられている．これらのカードは，要求やプロジェクト管理のための一種のモデルと考えることができる．このようなカードを用いたモデリングは以下のような長所がある．

- 短い文書とインデックスカードだけに基づくので，容易に習得できる
- グループワークに適している
- プログラミングとモデリング間の作業の重複が少ない

一般的に UML のような記法は教育してから，自らモデルを書けるようになるまでに数週間から数カ月間の期間を要する．しかし，数週間から数カ月間の期間を教育に投資できる商業的な開発プロジェクトは限られている．それに対して，このようなカードを用いたモデリングは 5 分間程度の説明で実践できるという大きな利点がある．さらに，カードを用いたモデリングでは，カードを参加者に分担させることが可能であり，参加者全員を巻き込んで協調的にモデリングを行うことを可能にする．従来のモデリング方法では，まずモデルを考えてから，議論することが一般的であったが，カードを用いたモデリングであれば準備なしに即興的にモデリングを行うことが可能になる．また，モデリングをプログラミングを補完する作業と位置付け，両者の重複を防ぐことにより，無駄な作業が発生するのを防止している．

このようなカードを用いたモデリングは，アジャイルなソフトウェア開発でのモデリングの 1 つのかたちを示すものだと考えられる．その一方で，カードを用いたモデリングは，UML のようなより形式的なモデリング方法と比べると以下のような限界がある．

(A) 矛盾や不整合を除くのが困難
(B) 維持や更新するのは困難
(C) 全体像の把握には難しい

XP では，カード形式のモデルはあくまで過渡的なものだと割り切ることにより (A)，(B) の限界を避けている．最終的に矛盾や不整合を除いたり，維持する対象はプログラムコードであり，プログラミングに入ればカードは破棄される．また，XP では開発チーム内でソフトウェアのアーキテクチャに対するメタファ（喩え）を用いることを推奨している．たとえば，電子商取引で用いられている買い物かごはこのようなメタファの 1 例だと言える．このようなメタファは，開発チームのメンバが開発すべきものに対する共通の理解を持つために役立つと考えられる．

まとめると，XP が提案するアジャイルなモデリングとは以下のようなものだと考えられる．

- インデックスカードやメタファに基づくモデルを使う
- モデルはプログラミング段階で破棄する
- モデル作りにおいて専門的なスキルを求めない

Agile Modeling

AM（Agile Modeling）は，Ambler により提案されたアジャイルなソフトウェア開発においてモデリング作業を効果的に行うための価値，原則，プラクティスである．AM の原則やプラクティスにおいて，特徴的な点は以下の 3 点である．

(A) すばやいフィードバックを重視する
(B) モデリングを理解したり，話し合う手段として考える
(C) 多様なモデリングテクニックを広く，浅く使う

(A) には，2 種類の意味がある．第 1 に，要求を理解する段階でモデリングを活用することにより，開発依頼者にすばやいフィードバックを提供するということである．この点では，従来の開発の要求段階の進め方に近く，開発依頼者が要求を具体化するためにモデルを積極的に用いるということである．そのために，AM では要求を表現するために後述するように多様なモデリングテ

作業種別	モデリングテクニック	形態	即興性	大局的表現	UML で規定
要求	ストーリカード	カード	○		
	ユースケース図	図	○	○	○
	本質 UI プロトタイプ	付箋紙	○		
	ビジネスルール	カード	○		
分析	ロバストネス図	図	○	○	△*
	CRC カード	カード	○		
	クラス図	図			○
設計	データモデル	図		○	△**
	クラス図	図			○
	相互作用図	図			○
	ステートチャート図	図			○
	コンポーネント図	図	○	○	○
	配置図	図	○	○	○
	CRC カード	カード	○		

*：UML のプロファイル（拡張記法）として，サポートされている

**: UML のプロファイルとして，提案されている

表 -1　アジャイルモデリングで推奨される主なモデリングテクニック

クニックを推奨している．第 2 に，モデルに対するフィードバックをなるべく早く得ることを促すものである．従来の開発では，分析，設計などのモデリング作業では一定の期間にひたすらモデリングを行うが，AM ではモデルをある程度作成したら，そのモデルの妥当性をプログラムによって確かめることを推奨する．この点では，AM でも XP のように短期のイテレーションを推奨する立場である．

（B）は，モデリングを，作業間での情報伝達の手段ではなく，理解したり，話し合うための手段だと位置付けるということである．後者を実践するために，XP で使っているようなカード型のモデルを作成したり，白板に UML のモデルを書いたり，付箋紙を貼り付けながら，協調的にモデリングを行うことを推奨している．

（C）は，UML を含む多様なモデリングテクニックを広く，浅く用いるということである．AM で推奨されているモデリングテクニックは，全部で 32 ものテクニックに及ぶ．AM で推奨されているモデリングテクニックの中で代表的なものを抜粋したものを示したのが**表 -1** である．この表では，各種のモデリングテクニックを作業種別，形態および以下のような 3 つの観点で分類している．

- **即興性**：議論しながら，モデリングを行うのに適したテクニックであるか
- **大局的表現**：ソフトウェアの全体像の表現に適したテクニックであるか
- **UML で規定**：UML で規定された記法であるか否か

表 -1 で挙げられている本質 UI（User Interface）プロトタイプとは，Usage Centered Design で用いられているモデリングテクニックで付箋紙を用いて画面や帳票のモデルを作る方法である．また，CRC カードとは，インデックスカードを用いる分析，設計テクニックである．さらに，ロバストネス図は Objectory という方法論で提案した Interface, Control, Entity という 3 種類の類型的なオブジェクトを用いるモデリングテクニックである．また，データモデリングとしては ER（Entity Relationship）手法のような関係データベースを対象としたモデリングテクニックを指している．

表 -1 に示されるように，AM では XP で使われているカードを用いたモデリングのような即興的なテクニックを取り入れ，さらに図形式のモデリングテクニック，UML のモデルやデータモデルなども併用するのが特色になっている．これらのモデリングテクニックはそれぞれ細かい記法まで含めると習得に相当な時間を要するが，AM ではこれらのモデリングテクニックのごく基本的な記法だけを広く，浅く使うことを推奨している．そのために，5 分間程度の説明で誰もが使えるようになるための記法の簡単な見本を準備することを提案している．

AM はモデリングに特化しているため，XP のような開発方式と併用することにより，それらの開発方式のモデリングに関する部分を強化することができる．たとえば，AM を XP に併用した場合には，XP の弱点であった以下のような点を補うことができる．

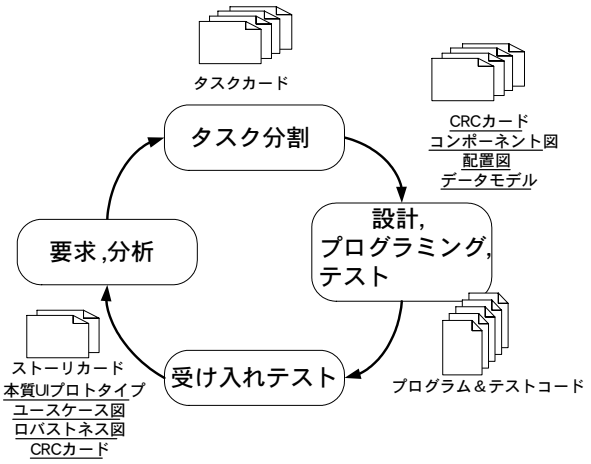

図 -4　AM を併用した XP の開発サイクルの例

- 要求に対するフィードバックがイテレーション単位と遅いこと
- 開発内容の全体像に対する共通の理解を形成しにくいこと

AM を XP と併用して，J2EE（Java 2 Platform, Enterprise Edition）で実装されるような Web ベースのデータベースシステムの開発に適用する場合，**図 -4** のような開発サイクルを考えることができる．

- **要求**：ストーリカードに本質 UI プロトタイプやユースケース図を併用することにより，要求内容に対する素早いフィードバックを行う
- **分析**：要求内容に基づいて，グループワークでロバストネス図を作成したり，CRC カードでモデリングを行う
- **設計**：分析結果に基づいて，グループワークで CRC カード，UML のコンポーネント図，配置図，データモデルなどを用いて全体的な設計を行い，各自の詳細設計とプログラミング作業に入る．また，詳細設計とプログラミングの作業が進行している期間にも，日次あるいは随時設計上の問題点や変更点をモデルにより協議する．

この開発サイクルの例を考えるにあたって，開発メンバの大半がプログラミングに先行してクラス図等の UML のモデルが書けないことを前提にした．そのため，UML の図は記法が比較的容易に習得できるコンポーネント図や配置図にとどめてある．もし，開発メンバの UML の習熟度が高ければ，クラス図や相互作用図などの UML の図を使用することも可能である．

AM では，XP と同様に，第 1 義的に維持するのはプログラムコードだと位置付けているが， 維持する労力を上回る価値があるモデルは，維持することを推奨している．たとえば，分析ステップで作成した CRC カードのモデルは破棄する．その一方で，コンポーネント図や配置図のように大局的な設計に関する共通認識を持つために有効なモデルは維持することになる．

今まで説明した AM のモデリングに対する考え方をまとめると，以下のようになる．

- インデックスカード，付箋紙，図によるモデルを推奨する
- 維持する労力を上回る価値があるモデルは維持し，そうでないものは破棄する
- モデル作りには少しだけ専門的なスキルが必要

まとめ

本稿では，開発依頼者の要求の変化に対応する開発方式として注目を浴びているアジャイルなソフトウェア開発におけるモデリングに対する 2 つの考え方について紹介した．これら 2 つの考え方で共通するのは，以下の点である．

- すぐに実践できるモデリングテクニックを用いている
- モデルは，プログラムコードを補完する過渡的なものである

今後，これらの考え方が多くの開発プロジェクトで実際に受け入れられるかどうかが大きな焦点となる．

参考文献

1）http://www.agilemanifesto.org/
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）スコット・アンブラー：アジャイルモデリング― XP と統一プロセスを補完するプラクティス，翔泳社（2003）．

（平成 15 年 6 月 30 日受付）