第 13 章　ソフトウェア開発の手順

ソフトウェア開発の作業

第 12 章では、ソフトウェアを開発し、活用するために、全体としてどのような作業が必要かについて、「ISO／IEC 12207：2008（ソフトウェアライフサイクルプロセス）（日本では JIS X 0160：2012 ）[JIS12a]」、あるいはそれを基に日本の状況に合わせるべく手を入れた「共通フレーム 2013 [IPA13a]」に基づいて、大枠を述べた。

そこで述べたのは文字通り、ソフトウェアだけでなくハードウェアも含むシステム全体の、開発作業を支援するものなどまで含めた全てのライフサイクルにおける作業であって、狭い意味でのソフトウェアの開発に関わるものだけではなかった。

この中からソフトウェア開発に関わる作業を抽出すると、以下のようになる[1]。

1. ソフトウェア実装プロセス開始の準備
2. ソフトウェア要件定義
3. ソフトウェア方式設計
4. ソフトウェア詳細設計
5. ソフトウェア構築
6. ソフトウェア結合
7. ソフトウェア適格性確認テスト
8. ソフトウェア導入
9. ソフトウェア受入れ支援

この章の以下の議論では、純粋なソフトウェアの開発だけを対象にして、上記 9 個の作業で議論を進める。

ソフトウェア開発の手順

ソフトウェア開発の手順とは、この 9 個の作業を、どのような順序で行うかを示すものといえる。その主なものとして、次のものを挙げることができる。

- ウォータフォール型開発手順
- インクリメンタル型開発手順
- 修正ウォータフォール型開発手順
- スパイラル型開発手順
- 進化型開発手順
- ラショナル統一プロセス（RUP）
- イタレイティブ型開発手順
- U 字型開発手順

以下でこれらの 8 つの開発手順について、それぞれ考えてみたい。また最後に、モックアップとプロタイプ作成について簡単に触れる。

いずれのソフトウェア開発手順も、起点には要件定義書であり、その終点で得るものは要件定義書に記載されたとおりのソフトウェアである。

---

[1] ここでは「共通フレーム 2013」のソフトウェア実装プロセスから、下位のプロセスを抽出して示した。別のいい方をすると、アクティビティとそれ以下のタスクはここには示していない。

ウォータフォール型開発手順

ウォータフォール型開発手順による開発とは、ISO／IEC 12207 : 2008 や共通フレーム 2013 で表現されている通りの手順で作業を進めることを意味する。この概略の図を、図表 13-1 に示す[2]。

ウォータフォール型開発手順は、1970 年にウィンストン・ロイス（Winston W. Royce）が命名したことになっている[3][ROY70]。私はそれ以前の 1962 年に大学を卒業して、ソフトウェア開発の世界に入った。この時の開発手順はこのような立派な名前こそ付いていなかったが、まさにこのウォータフォール型だった。考えてみれば、これは当然である。ソフトウェアに限らず何か物を作ろうとする場合、まずどういう物を作るのかを考える。ソフトウェアの場合、この作業を「要件定義」と「システムの設計」と呼んでいる。その後実際の物作りが始まる。作った物が当初想定したとおりにできているかのチェック、つまりテストは、物が完成した後でなければ行うことができない。必然的にこの作業の順序は、ウォータフォール型になる。

「ウォータフォール」とは、「水の流れ」を意味する[4]。山に降った雨が川を流れ降りて、最後は海に注ぐ。この水の流れの大きな特徴の 1 つは、決して後戻りしないことである。「ウォータフォール」という名前は、ここから来ている。

つまりウォータフォール型開発手順では、ソフトウェア開発の作業は決して後戻りしない。「システムの企画」[5]の作業の結果を基に、ある情報システムの開発が決裁されると、まずそのソフトウェアについての要件定義、つまりそのソフトウェアで実現したいこと（機能とそれに関連した機能以外の事項）は何かが明確にされる。この作業をここでは「要件定義」[6]と呼んでおく。ここで明確にされたことを、次に実際のコンピュータ（ハードウェア）やネットワークの上で、オペレーティング・システム（OS）やデータベース管理システムなどの既成のソフトウェアの下で、どう実現するのかを考えて、明確にする。この作業を「システムの設計」[7]と呼ぶ。この過程で、どういうプログラムを作るのかも明らかになる。この結果に基づいて「プログラミング」[8]を行う。さらに最初はその個々のプログラムが、最後はソフトウェア全体が、当初考えたとおりのものになっているかどうかの確認（「テスト」[9]）を行い、そのテストに合格すると、そのソフトウェアを使って我々が当初想定した作業を行う。これを、「情報システムの

---

[2] ソフトウェアの開発手順を構成する作業は、9 個あると述べた。私の図ではこの 9 個の作業を 4 つに集約し、それに企画と保守の作業を合わせて 6 個で表すことにする。

[3] ウィンストン・ロイスは、彼のペーパでウォータフォール型開発手順についての素晴らしい図を書いた。しかし彼のこのペーパには「ウォータフォール」という言葉は出てこない上、彼はこのペーパでこの開発手順を否定している[ROY70]。

[4] 英和辞典で Water Fall を引くと、「滝」と出てくる。だからウォータフォール型開発手順を「滝型」と訳す人がいる。しかし私が大学で教えていた頃、試験の答案にこれを「滝型」と書いた学生がいると、それだけでこの科目の単位を与えないことにしていた。私が知っている限り、情報システムの開発に最も長い期間がかかったケースは 8 年である。8 年間もかかって落下する水の滝は、この世界にはない。

[5] 「システムの企画」の作業は、第 19 章で述べる。

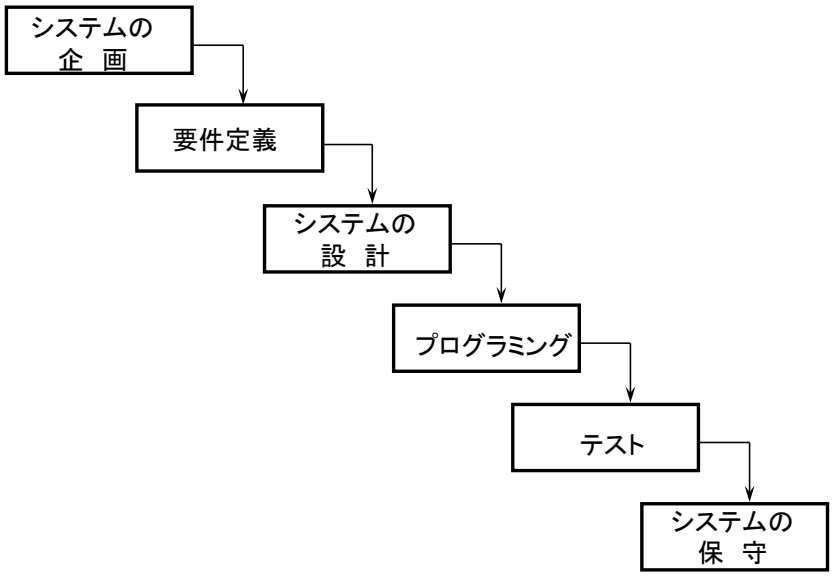
[6] 「要件定義」の作業は、第 21 章で述べる。

[7] 「システムの設計」の作業は、第 23 章から第 26 章で述べる。

[8] 「プログラミング」の作業は、第 27 章で述べる。

[9] 「テスト」の作業は、第 30 章と第 31 章で述べる。

運用」と呼ぶ。システムの開発者たちはこの段階で、「システムの保守」[10]の作業を担当する。



図表 13-1　ウォータフォール型開発手順

だから仮にこの当初の考え通りに開発を行うことができれば、開発プロジェクトの管理は比較的容易で、ワークロードも予算も決めやすい。

このウォータフォール型開発が成り立つための要件が、2 つある。

- 1 つ目は、開発の最初の時点でこの情報システムへの要求を明確に決めることができること
- 2 つ目は、この決めた情報システムへの要求が最後まで変わらないこと

である。

しかし残念ながらこの 2 つの要件は、一般には成り立たない。つまり開発の最初の段階で、情報システムの要件がなかなか決まらない。要件を決めるべき立場にあるユーザは優柔不断で、適切に「決める」ことができない。そしていったん決めたこの要求が、開発の期間中変わり続ける。具体的には、世の中が変わる。それに応じてビジネスのやり方も変わる。技術も進歩する。当然情報システムへの要求も変わる、というわけである。ユーザの気が変わることも、もちろんあり得る。ケーパース・ジョーンズ（Capers Jones）は、「当初の要求は平均して、毎月その 1%が変わり続ける」といっている[JON93]。

さらにこの開発手順の欠陥として、情報システムからのアウトプットをユーザが実際に見ることができるのは開発のかなり遅い時期になってからである。この段階でユーザがその出力に問題を発見し、変更しようとすると、スケジュール上、あるいは開発予算上、大きな問題が出る。つまり、ユーザからのフィードバックがたいへん効きにくい開発手順であるということになる。

このためこのウォータフォール型開発手順を採用したプロジェクトには、失敗が多いという報告がある[LAM04]。アメリカの国防総省も、発注したソフトウェアをウォータフォール型手順で開発するように 1980 年代に定めた規格（MIL-STD-2157）を 1994 年 12 月に取り下げて、

10 「システムの保守」の作業は、第 33 章で述べる。

進化型開発を採用するよう別の規格（MIL-STD-418）を定めた[LAM04]。

ウォータフォールは水の流れだといったが、そこから上流、下流という言葉が出てくる。つまり作業の最初の方にある「システムの設計」までの作業を「上流工程」、「プログラミング」以降の作業を「下流工程」と呼ぶ。更に企画や要件定義の作業を、「超上流」と呼ぶこともある。

インクリメンタル型開発手順

ウォータフォール型開発手順は、その対象とする情報システムの規模に関係なく一気に全体を開発する。しかし情報システムの規模が大きくなるとこれが必ずしも現実的ではなく、いくつかのサブシステムに分けて開発し、順次稼働させる方が好ましいことがある。この開発方法を、「インクリメンタル型開発手順」と呼ぶ。その作業の流れを、図表 13-2 に示す。

インクリメンタル型開発手順を採用する理由は、2 つある。その最初のものは、少なくとも一回目のカットオーバーは、ウォータフォール型開発手順で全体を一挙に開発する場合と比較して早いことである。つまりユーザは、一部の機能について早い時期から使うことができるようになる。

2 つ目は、情報システム開発の難しさを軽減することができるかもしれないということである。「情報システム開発の難しさは、規模の二乗に比例する」などといわれることがある。よく調べてみると、難しさの増加の割合は 2 乗ではなく、1.3 乗程度の数である [11]。具体的には二乗の場合、規模が倍になると難しさは 4 倍になることになる。現実は、そこまでは行かない。しかし規模の増加以上に、難しさが増えることは事実である。

別のいい方をすると、規模を半分にすると、難しさは半分以下になる。だから当初の規模を 2 つに分けて別々に開発し、それらを単純に合わせることで当初の情報システムが実現できるのなら、開発の難しさは当初のものを一挙に開発するより少なくなる。つまり、この開発手法を積極的に採用するべきであるということになる。

現実の話は、そううまくはゆかない。まず、2 つに分ける分け方が難しい。下手に分けると、二回目の開発結果を既に稼働している一回目のものと合わせるとき、既に稼働している部分に大幅な手直しが入ることになる。これが開発の難しさを一挙に増加させ、システムを不安定にすることもある。

さらに一般に、二つ目のプロジェクトの管理がたいへんに難しい。関係者は全員、最初のプロジェクトの進捗を気にしており、そちらに力が入る。その影響を受けて、二つ目のプロジェクトには充分な人員や予算が割り当てられないことが多い。最初のものが仮にうまくいっても、二回目はスケジュール遅延や予算超過、あるいは品質不良などのソフトウェア危機の症状を起こすことが多い。最初のものと二つ目の開発が並行に行われると、これが一層顕著になる。

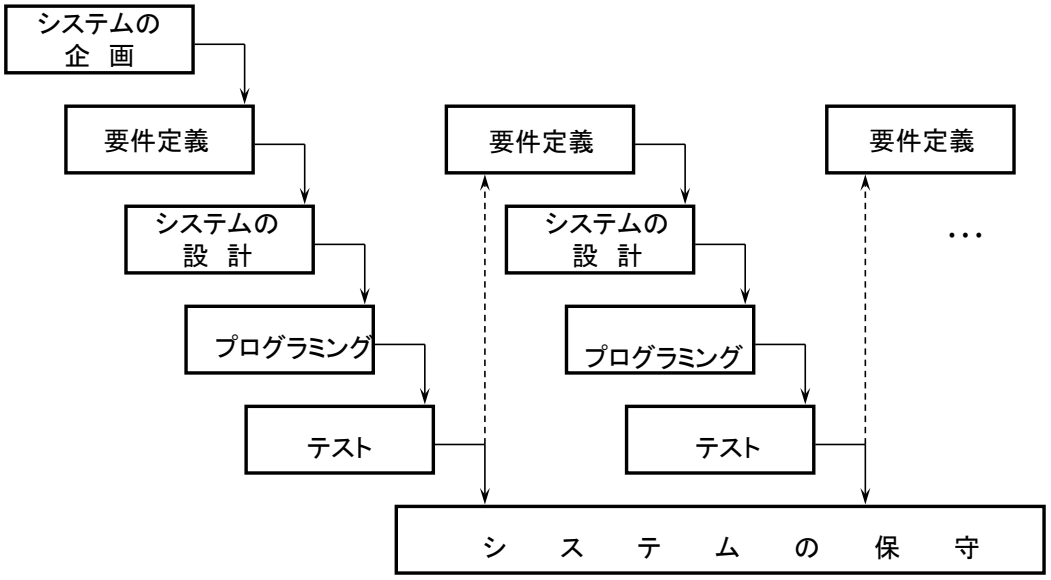
これまでの説明は全体を 2 つに分けての開発の話で進めてきたが、何も 2 つにこだわることはない。3 つ以上に分けても構わない。話のポイントも変わらない。

修正ウォータフォール型開発手順

ウォータフォール型開発手順に伴う問題点は、既に述べた。建前としてウォータフォールでは作業の後戻りはないので、仕様変更などへの対応は形式的に不可能である。そうはいっても現実問題として、ソフトウェア開発には多くの仕様変更がある。したがって、仕組みとしてそ

[11] バリー・ベーム氏が 1980 年に書いた[BOE80]にある COCOMO モデルで、この数字が使われている。

れを入れない開発手順には無理がある。そこで進め方の基本はウォータフォール型だが、途中で作業を止めて仕様変更に対応するようにしようという開発手順が提案された。それを、修正ウォータフォール型開発手順と呼んでおく。この方式はジェラード・ワインバーグ（Gerald M. Weinberg）が提案した[WEI92]もので、その進め方を図表 13-3 に示す。



図表 13-2　インクリメンタル型開発手順

修正ウォータフォール型開発手順でも、前述のように基本的にはウォータフォール型開発の手順で作業を進める。その途中でどんどん入ってくる仕様変更などにはすぐには対応せず、受け取って棚の上に積み上げておく。開発作業が進んで一区切り付いたところ、例えば「システムの設計」が終わった時点で、棚の上の仕様変更を取り出してきて、それぞれについて受けるか拒否するかを判断する。もちろんこの判断のベースには、変更による効果と変更のためのワークロードの対比が使われることがあるので、判断の前に変更による影響の範囲やワークロードの見積もりなどの作業が必要になる。

その変更を受け入れるとすれば、「要件定義」の作業に遡って、既に完成している要件定義書などの成果物に必要な修正／追加を施し、レビューして整合性／一貫性が保たれていることを確認した後、「システムの設計」の作業の一部をやり直す。その後で「プログラミング」の作業を始める。

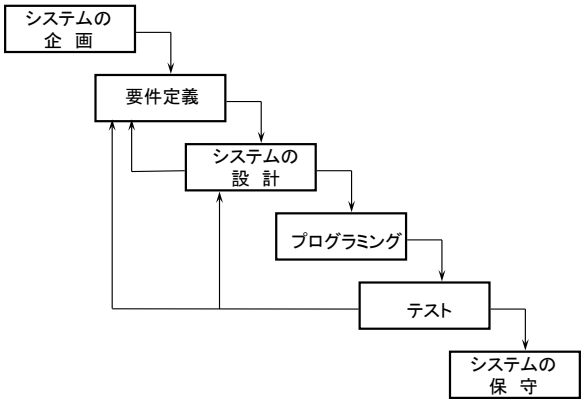
その変更要求を受け入れない場合は、提案者に理由付きでその旨を通告して、その変更要求への対応は終わりとなる。

このように修正ウォータフォール型開発手順は、それぞれの要所でそれまで貯まっている仕様変更の対応をまとめて行うことで、基本的にはウォータフォール開発手順に従いながら、仕様変更への対応をうまく図ろうとする方式である。これは仕様変更をソフトウェアの構成管理[12]の下で行う場合の作業手順と、基本的に同じになる。

この方式の問題は、開発予算やカットオーバーの時期を当初段階で決めることが難しいということである。しかし仕様変更がある場合には純粋なウォータフォール型でも開発予算やカッ

12 ソフトウェアの構成管理については、第 8 章で述べた。

トオーバーの時期を決めるのが難しくなるので、特別大きな問題とはならないかもしれない。



図表 13-3　修正ウォータフォール型開発手順

スパイラル型開発手順

1988 年にバリー・ベーム（Barry w. Boehm）は、それまでのウォータフォール型開発手順とはかなり異質の、スパイラル型開発手順を発表した[BOE88]。そのスパイラル型開発手順の作業の方式を、図表 13-4 に示す。

スパイラルとは渦巻きのことで、ここではその渦巻きの一回転がウォータフォールの全工程を意味している。あるいはスパイラル型開発手順とは、インクリメンタル型開発手順をもっと小刻みに、何度も実施するという捉え方ができるのかもしれない。

このスパイラルの始め方に、次の二通りの方法が提案されている。リスク駆動型とクライアント駆動型である[LAM04]。

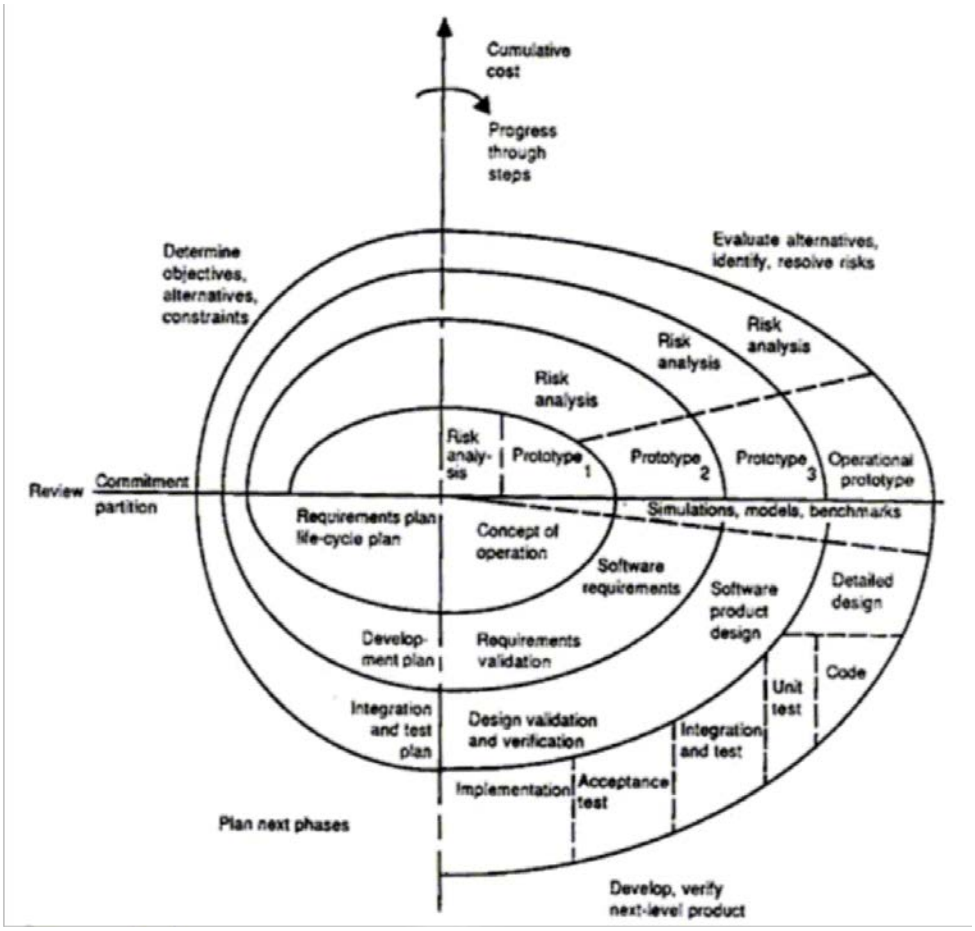
リスク駆動型開発は、今開発部隊が持っているリスクの高いものをまず選んでスパイラルを回し、大きなリスクをまず解消しようとする開発方法である。この方法を繰り返すことでその開発部隊は開発に関わる大きなリスクを持たないようになり、開発成功の可能性が高くなる。

もう一つのクライアント駆動型開発では、それぞれのスパイラルをこれから回そうとする前にソフトウェアの開発部隊はユーザと会合を持ち、これまでに実現された機能を確認して、その上で次のスパイラルで何を開発するかを取り決める。このことによってスパイラル型開発手順では、ウォータフォール型開発手順での 2 つの問題点、つまり開発の当初段階で仕様がなかなか決まらないことと、開発過程で仕様が変更され続けることから生じる問題の両方を、一挙にクリアすることができる。

いずれの場合も一回りのスパイラルの後で、そこで開発したソフトウェアをユーザに提供することを原則とする。

しかし逆に、何回スパイラルを回せば開発が終わるのか、その結果開発コストやスケジュールがどうなるのかについての問題の解決は、これまで述べてきた開発手順の中で一番難しいか

のもしれない。



図表 13-4　スパイラル型開発手順（[BOE88]より）

先にも述べたようにベーム氏の論文発表は 1988 年だったが、実際はアメリカなどではその前からこの方式が広く行われていた。そして今では、この方式が既に主流になっている。後で述べるアジャイル・ソフトウェア開発[13]は、この開発手順が前提になっている。またマイクロソフト社などが採用している「ディリービルド」[14]の方式などは、この開発手順があってはじめて可能になるものである。

進化型開発手順

進化型開発手順は、スパイラル開発によく似ている。

つまり最初は、これから開発しようとするソフトウェアに不可欠な、基本的な機能を開発する。その後で、プロジェクト・チームとして、ユーザや顧客にとって必要と考える機能、あれ

[13] アジャイル・ソフトウェア開発については、第 14 章で述べる。

[14] ビルドとはそれまでに完成しているプログラム群を統合してテストの目的で動かしてみることをいい、ディリービルドとはそれを毎日実施することを意味する。マイクロソフトの開発の進め方の、1 つの特徴である。

ば便利な機能、などを順次それまでに開発した「核」に付け加えて行く。

Cem Kaner などは、テストの進め方の解説をこの開発手順で行っている[KAN99]。それによると、この開発に成功するためには、当初の「核」に当たる部分をよく考えて、柔軟に開発しておかなければならないとしている。当然の指摘というべきだろう。

ラショナル統一プロセス（RUP）

スパイラル型開発手順では、毎回のスパイラル（螺旋）に特徴があるわけではない。前述のようにスパイラルを回す前にユーザを交えて毎回テーマを決めて、開発に取り組むことなどを行う。したがって毎回開発のテーマは異なるけれど、基本的には同じスタンスでユーザが希望する機能などを用意することになる。もちろん最初のスパイラルではアプリケーション部分だけでなく、それを稼働させるためのアーキテクチャの部分が含まれることになる。しかしそれは、積極的に意図したものではない。

規模の小さな情報システムの場合は、この方式で充分に対応できるだろう。しかし規模の大きな情報システムの場合、例えば設計作業の中でアーキテクチャの設計[15]と呼ばれるものに充分なワークロードをかけ、さらにそれを具体的なプログラムにしてテストをするのにもっと大きなワークロードを必要とするような情報システムは、この通常のスパイラル方式での開発は難しくなる。

この問題を解決するためにラショナル社は、ラショナル統一プロセス（RUP：Rational Unified Process）を発表した[KRU03]。RUP の手順としては、スパイラル開発手順のように何度もスパイラルを回すけれど、スパイラルを回すときのポイントをウォータフォール開発手順のように要件定義からアーキテクチャ設計、詳細設計、プログラム開発、テストなど順次と移動させてゆく方式である。この手順を図表 13-5 に示す。

ラショナル社は IBM に吸収されて、日本ラショナル社も今は日本 IBM の一部になっている。ラショナル社は、オブジェクト指向技法の開発や普及で著名な企業だった。当然このラショナル統一プロセスも、オブジェクト指向技法を使った開発との折り合いが良いとされている。

この RUP については、IBM（元のラショナル社）からライセンスの供与を受けて、そのガイドブックや成果物のひな形などを手に入れ、それを自社流に修整して使用することができる。また RUP の簡易版として、オープンな UP（Unified Process）を入手することも可能である[LAM04]。

これまでオブジェクト指向技法を使って開発されたソフトウェアは規模の小さなものが多かったので、開発手順としては通常のスパイラルで良かったのかもしれない。しかしそのソフトウェアの規模が大きくなると前述の通り単純なスパイラル型開発手順には無理があり、それと共にこの方式が主流になって、徐々に使われるようになってゆくものと思われる。

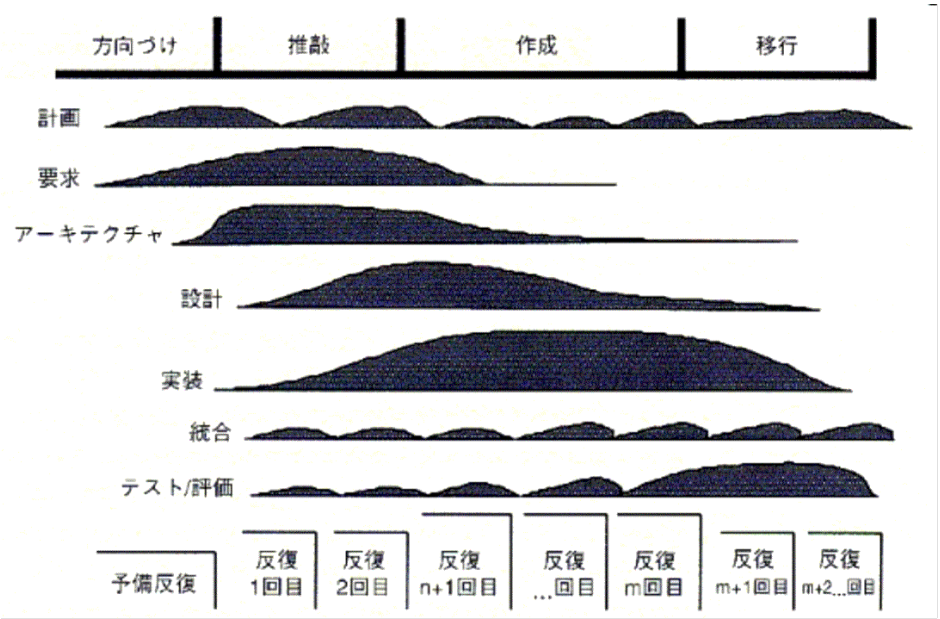
イタレイティブ型開発手順

これまでの開発手順は基本的に、ウォータフォール型開発手順で示された順序に踏襲して開発を進めるものであった。違いは後戻りなく進めるのか／後戻りを許すのか、一回で完遂するのか／何度もそれを繰り返すのか、というようなところにあった。

しかし次に示す手順は、それとも無関係に開発者の気の向くままに進めようというものである。それをここではイタレイティブ開発手順と呼んでおく。その作業の進め方を図表 13-6 に示

[15] アーキテクチャの設計については、第 23 章で述べる。

す。



図表 13-5　ラショナル統一プロセス（[KRU03]より）

いくら作業の順序を無視するといっても、やはり最初は「システムの企画」があり、最後は「システムの保守」になる。「システムの保守」、つまり「システムの運用」は、システムの「テスト」が終わってから開始される。このように、絶対的に無視できない順序がある。

しかしそれ以外の作業の間に、基本的な順序を設けない。前述の通り開発担当者の気の向くままに進めてみようとする方式である。この方式はアジャイル・ソフトウェア開発、とりわけその中でも著名なエクストリーム・プログラミング（XP）[16]と折り合いがよい開発手順である。

U 字型開発手順

日本情報システム・ユーザー協会（JUAS）は、ウォータフォール型開発手順の変形である U 字型開発手順を発表している。この開発手順の概要を、図表 13-7 に示す。

U字型開発手順の特徴は、以下のところにある[JUAS05]。

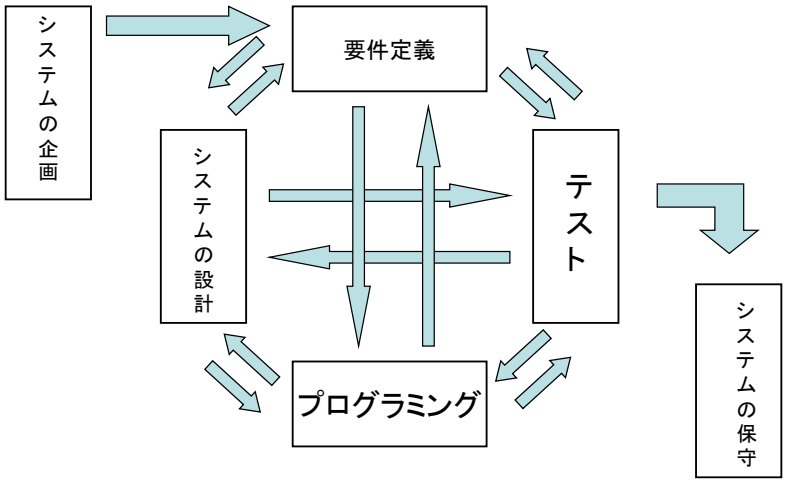
- プログラマによる単体テストが終わった直後に、「単体機能テスト」と呼ぶ機能確認のためのテストを実施する。
- 単体テスト段階でデータ変換のための移行システムを開発し、前述の単体機能テストからそのデータの使用を始める。

つまりテストも移行システムも、通常のウォータフォール型システムよりずっと前倒しで行うところに、U字型開発手順の特徴がある。

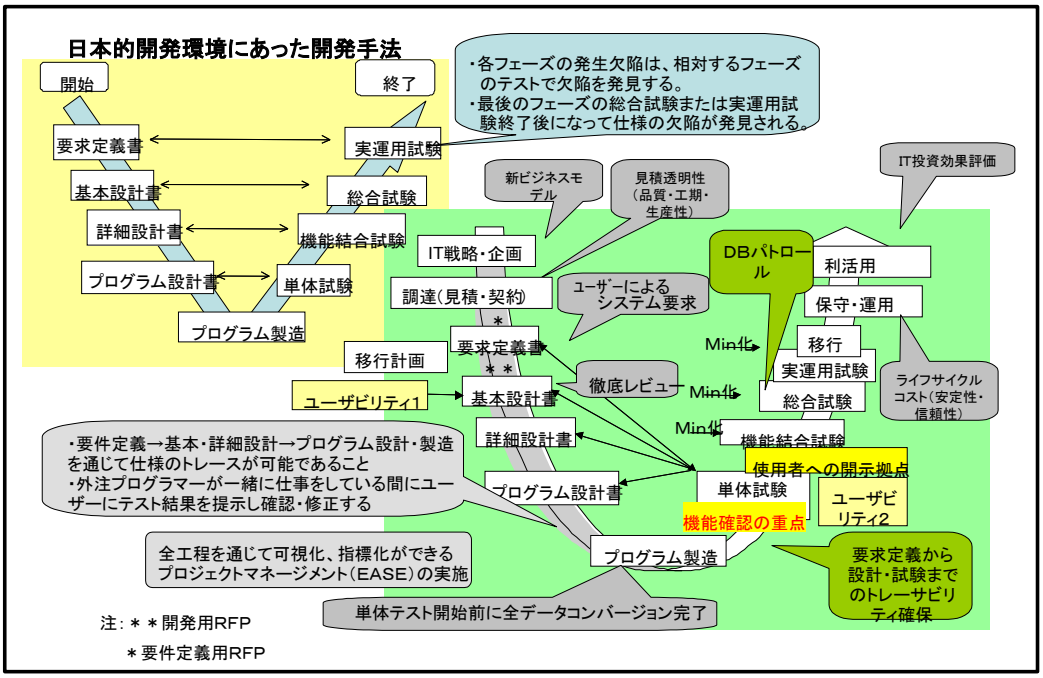
欠陥の発見は、開発工程の前の方で行うほど効果が大きいことは後述する。そのためにテストよりレビューが重要とされているわけだが、単体機能テストはテスト段階での欠陥の発見を極力前倒しで行うところに意味がある。これはソフトウェア工学上の理論的な議論であるが、

[16] アジャルトソフトウェア開発とエクストリームプログラミング（XP）については、この次の第 14 章で述べる。

実質的な効果も大きい。



図表 13−6　イタレイティブ開発手順



1

図表 13-7　U 字型開発手順（[JUAS05]より）

日本で今普通に行われている開発では、実際に機能確認を行うシステムテスト段階では、プログラムを作成した人たちはすでにその開発現場を去ってしまっており、そこで欠陥が発見されると、そのプログラムを開発した人たちではないソフトウェア技術者がバグを見つけ出し、プログラムを修正しなければならない。これは、必ずしも効率的ではない。単体機能テスト段階ではプログラマがまだその開発現場に残っており、バグの修正はそのプログラムを作成した人たちによって実施される。ここに、実質的な効果がある。

移行システムを早期に開発して単体機能テストで実データから変換したデータをテストで使用することも、単体機能テストのテストデータの準備と移行システムの早期での検証の両面で効果がある。

既成の考え方に囚われないユニークな発想は、良い効果をもたらすことがあるという好事例である。

モックアップとプロトタイプの作成

モックアップ作成とプロトタイピングは、これまで述べてきたようなプログラム開発全般に関係するような開発手順ではない。要件定義や設計の一部の位置づけされ、ある種のユーザの要求を明確にする際に使用される作業である。だからこのモックアップ作成やプロトタイピングは、ウォータフォール型開発手順などと組み合わせることもできるし、スパイラル型開発手順、あるいはイタレイティブ型開発手順と組み合わせてもよい。

「モックアップ」とは、一種の模型である。例えば、あるコンピュータからの出力についてのユーザが要望を知りたいとき、見かけだけその出力と同じ物を作ってユーザに提示して意見を求める。そういうものが何もない状態で自分の要望を理路整然と述べることができるユーザは極めて少ないが、何かを提示されると自分が望んでいるものと提示されたものとの違いを指摘できる人は多い。モックアップは、このために使われるものである。

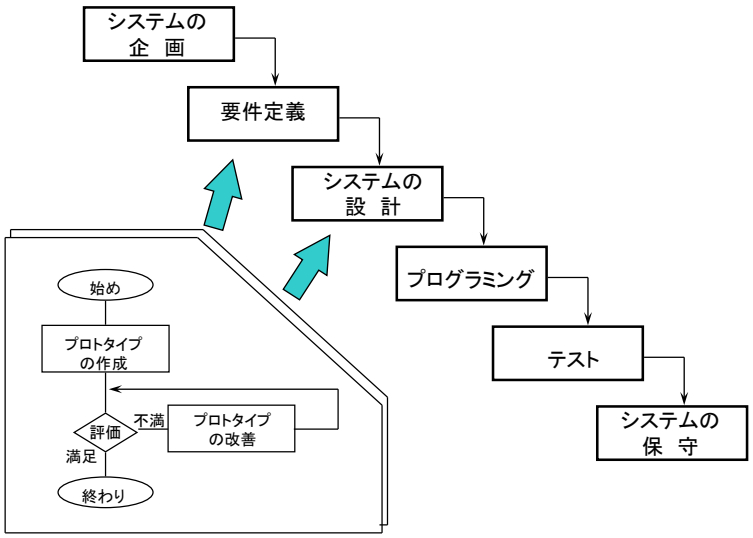
もちろんモックアップを一回提示し、意見を聞いてそれで終わりにするものではない。ユーザが「これでよい」というまで、何度も指摘された違いをモックアップに反映して提示し直すというプロセスを繰り返す必要がある。ウォータフォール型開発手順と組み合わせたモックアップ作成の進め方を、図表 13－8 に示す。

モックアップはユーザの要求を確認するのが目的であり、その目的を果たせば、それを作成したプログラムを破棄しなければならない。つまりモックアップを作成するために、あるプログラム言語を使用してプログラムを作成する。しかしこのモックアップのためのプログラムは、最終的に残るプログラムに含めてはならない。

最終的に残るプログラムはその情報システムのアーキテクチャにマッチし、品質面の配慮も充分になされたものでなければならない。逆にモックアップは、迅速にユーザの要求を把握するためにある。だから素早く目に見える物を作ることが必要である。この両方の要求を同時に満たそうとすることは、たいへんに難しい。モックアップはこの難しいものにはあえて挑戦せずに、簡便に対応しようとする考え方である。

一方「プロトタイプ」はモックアップと同じ目的で作成し、使用するけれど、そのプログラムを最終の製品の一部として取り込むところに違いがある。そのためにプロトタイプを作成するに当たっては、アーキテクチャへのマッチや品質面での配慮がたいへん重要な要件になる[17]。

[17] ここで記述した「モックアップ」と「プロトタイプ」の言葉の使い方は、今の我々の使い



図表 13－8　モックアップの利用

キィワード

ウォータフォール型開発手順、上流工程、下流工程、COCOMO、インクリメンタル型開発手順、修正ウォータフォール型開発手順、スパイラル型開発手順、リスク駆動型開発、クライアント駆動型開発、ラショナル統一プロセス、RUP、UP、イタレイティブ開発手順、アジャイル・ソフトウェア開発、エクストリーム・プログラミング、U 字型開発手順、モックアップ、プロトタイプ

略語

RUP：Rational Unified Process
UP：Unified Process

人名

ウィンストン・ロイス（Winston W. Royce）、ケーパース・ジョーンズ（Capers Jones）、ジェラード・ワインバーグ（Gerald M. Weinberg）、バリー・ベーム（Barry w. Boehm）

規格

ISO／IEC 12207：2008、JIS X 0160：2012、共通フレーム 2013、MIL-STD-418

方とは若干かけ離れているかもしれない。この考え方は、ISO／IEC TR 14759：1999 で明確に述べられているものであった[ISO99]。しかしこの規格は、今では廃止されている。

参考文献とリンク先

[BOE81] Barry w. Boehm, “Software Engineering Economics,” Prentice-Hall, 1981.

[BOE88] Barry w. Boehm, “A Spiral Model of Software Development and Enhancement,’ Computer, Vol 21, No. 5, IEEE, May, 1988.

[IPA13a] 情報処理推進機構ソフトウェア・エンジニアリング・センター編、「共通フレーム 2013 経営者、業務部門が参画するシステム開発及び取引のために ソフトウェアライフサイクルプロセス 共通フレーム 2013」、オーム社、平成 25 年.

[ISO99] ISO／IEC, “Software Engineering – Mock up and prototype – A categorization of software mock up and prototype models and their use ISO／IEC TR 14759：1999,” ISO／IEC, 1999.

[JIS12a] 日本工業標準調査会審議、「ソフトウェアライフサイクルプロセス JIS X 0160-2012（ISO／IEC 12207：2008）」、日本規格協会、平成 24 年.

[JON93] Capers Jones 著、島崎恭一他監訳、「ソフトウェアの病理学 システム開発・保守の手引き」、構造計画研究所、1995 年.

この本の原書は、以下のものである。

Capers Jones, “Assessment and Control of Software Risks,” Prentice Hall, 1993.

[JUAS05] 日本情報システム・ユーザー協会編、「システム・リファレンス・マニュアル（SRM）」、日本情報システム・ユーザー協会、2005 年.

[KAN99] Cem Kaner、Jack Falk、Quoc Nguyen 著、テスト技術者交流会訳、「基本から学ぶソフトウェアテスト」、日経 BP 社、2001 年.

この本の原書は、以下のものである。

Cem Kaner, Jack Falk, Hung Quoc Nguyen, “Testing Computer Software second edition,” John Wiley and Sons, 1999.

[KRU03] フィリップ・クルーシュテン著、藤井拓監訳、「ラショナル統一プロセス 第 3 版」、アスキー、2004 年.

この本の原書は、以下のものである。

Philippe Kruchten, ‘The Rational Unified Process - An Introduction, Third Edition,” Addison Wesley Professional, 2004.

[LAM04] クレーグ・ラーマン著、児高愼治郎他監訳、越智典子訳、「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 年.

この本の原書は、以下のものである。

Graig Larmann, “Agile and Iterative Development A manager’s Guide,” Peason Education, 2004.

[ROY70] Royce, Winston W. (1970): “Managing the Development of Large Software Systems: Concepts and Techniques. In: Technical Papers of Western Electronic Show and Convention (WesCon),”. August 25-28, 1970, Los Angeles, USA.

この論文は今、次の URL からダウンロードすることができる。

http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

[WEI92] G. M. ワインバーグ著、大野とし郎監訳、「ソフトウェア文化を創る 1 ワインバーグのシステム思考法」、共立出版、1994 年.

この本の原書は、次のものである。

Gerald M. Weinberg, “Quality Software Management: Volume 1 System Thinking,” Dorset House Publishing, 1992.

（2006 年（平成 18 年）8 月 14 日　初版作成）
（2007 年（平成 19 年）8 月 29 日　一部修正）
（2008 年（平成 20 年）8 月 12 日　一部修正）
（2010 年（平成 22 年）7 月 16 日　一部修正）
（2014 年（平成 26 年）3 月 19 日　一部修正）
（2017 年（平成 29 年）1 月 10 日　一部修正）