橋本 隆成　*HASHIMOTO, Takanari*

ウォーターフォールか，反復型か？

開発プロセスの基礎知識

ソフトウェア開発にとって，なぜプロセスが重要なのでしょうか．ソフトウェア開発に限らず，ハードウェア開発やビルや橋など建築物の開発でも，プロセスはもちろん重要です．しかし，ソフトウェアの場合は“開発対象が目に見えない”上に，他の分野と比較して非常に急激に技術が進歩したため，

ソフトウェア開発を開発者が完全に制御しきれていません．そのため，「要求定義」～「テスト」工程で作業すべき項目を明確にし，慎重にレビューを行いながらソフトウェアの品質を作りこんでいくという作業が不可欠になります．これが，ソフトウェア開発においてプロセスが重要になる理由です．

また，生産性の視点からも，プロセスは重要です．上流工程での欠陥は，それが下流工程で発見されて修正される場合は，上流工程で発見されて修正される場合と比較して，コストと時間が莫大になってしまうことが定量的に示されています．つまり，「ソフトウェアの品質と生産性は，プロセスによって決まる」と言っても過言でない状況にあるのです．

特に，ソフトウェア開発では，「実装」工程前の上流工程の作業が極めて重要です．ソフトウェア開発を，「要求定義」工程から「テスト」工程に分割して，レビューや各種インスペクションを行いながら進めていくことで，要求定義書からプログラムまでの技術的な大きな隔たりを，確実に効果的に埋めていくことが可能になります．

大規模なソフトウェアシステムでは，多数のエンジニアが開発に関わり，複数のチームによる開発が行われるのが普通です．各作業工程をそれぞれ異なるエンジニアが担当することも多いため，開発プロセス次第で開発されるソフトウェアの生産性，品質が大きく変わってくることになります．

プロセスの種類と特徴

以下では，プロセスについて詳しく解説していくことにします．まずは，プロセスについて「種類」

橋本 隆成（はしもと・たかなり）VZA10247@nifty.ne.jp
科学的なソフトウェア開発アプローチを推進する部署に在籍．オブジェクト指向技術，テスト技法，CASE Tool導入の社内推進業務および，CMMによる品質改善業務を担当．

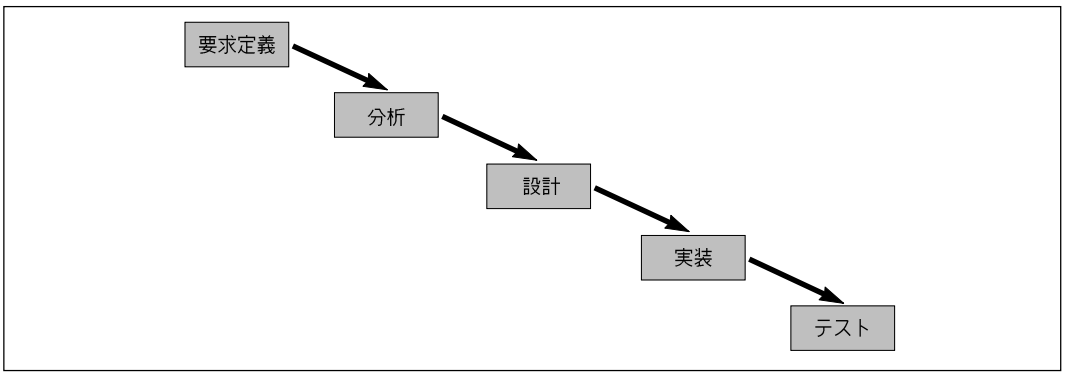

図1 ウォーターフォール型プロセス

や「特徴」などからいくつかの視点で分類・整理を試みます．

最初に，「プロセスとは何か」を確認しておきましょう．プロセスで定義される内容には，主に次のようなものがあります．

①作業担当者（ワーカー）
②作業（アクティビティ）
③作業間の明確な順序
④各作業の開始条件と終了条件
⑤開発ライフサイクル
⑥各作業の成果物
⑦規則や基準

このように，プロセスには，ソフトウェア開発・ITインテグレーション業務の上流工程から下流工程まで必要とされる作業手順や基準，および担当者が記述されます．

プロセスを利用する組織やプロジェクトの戦略にもよりますが，プロセスにはソフトウェア開発の範囲のフェーズだけでなく，「ビジネスエンジニアリング」と呼ばれるビジネス構想やビジネス企画についての作業が含まれる場合もあります．

ウォーターフォール型と反復型

ソフトウェア開発では，大きく2種類のプロセスが用いられてきています．「ウォーターフォール型」と「反復型」です．

■ウォーターフォール型プロセス

ウォーターフォール型プロセスは，「要求定義」から「テスト」までの作業を，明確なフェーズとマイルストーンを設けて実施するプロセスです（図1）．マイルストーンでは，各フェーズの作業成果をレビューで確認し，問題がなければ次のフェーズに進むことになります．

いったん次のフェーズへ進んだら，二度とそれ以前のフェーズへ戻ることがないのがこのタイプのプロセスの特徴で，上流工程から下流工程までが逆戻りすることなく一方通行で流れていくというその特徴から，「ウォーターフォール」と呼ばれています．

最近では，ソフトウェアの大規模化・複雑化にともない，新規性の高いソフトウェア開発では，ウォーターフォール型ではなく反復型の開発プロセス

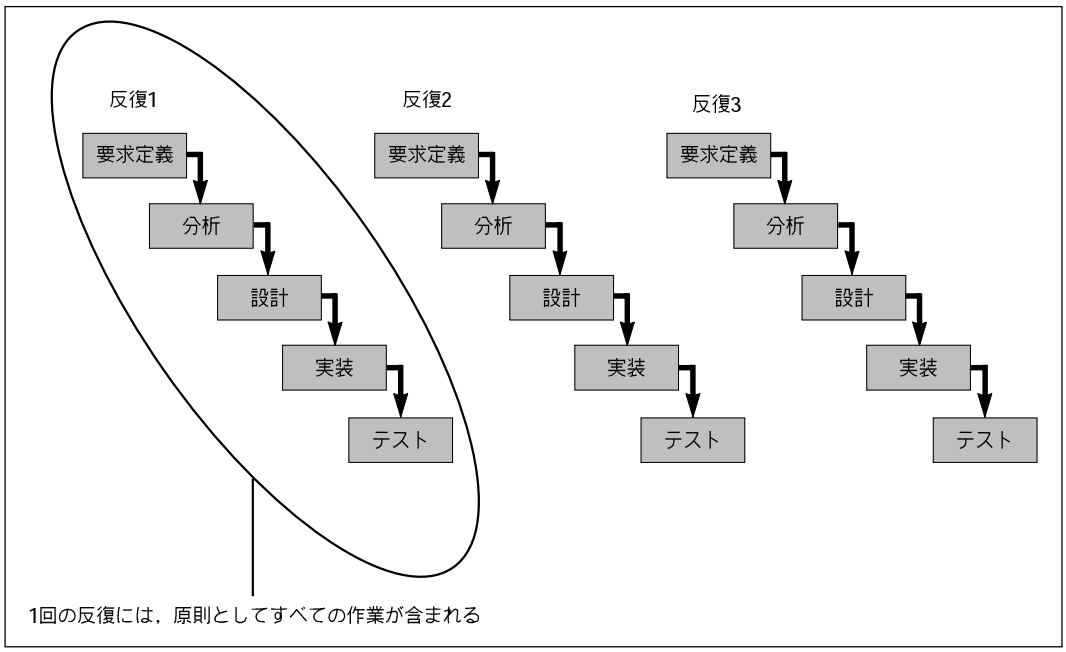

図2　反復型プロセス

が主流になってきています．

■反復型プロセス

ソフトウェアは，ハードウェアと異なり直接目に見えないために，各工程間の作業成果物のレビューで完全に見落としや間違いを発見することが困難です．また，ソフトウェアの実行速度やマルチタスクなどのタイミング，リソースの競合の検証は，机上のチェックやレビューでは限界があり，実際にソフトウェアを動作させてみないと検証が困難な検証項目も存在します．

そこで，実現する機能を少しずつ作成し，動作確認をしながら開発を進めるという「反復型開発」が不可欠になってきました．反復型開発では，実際に機能の一部分を実現したソフトウェアを開発して動作させることで，要求されている機能がユーザーの要望と合致しているか，設計上の問題がないかなどを確認しながら，機能を徐々に実現していきます．

◇　◇　◇

反復型プロセスは，開発のライフサイクルを複数回のイテレーション（反復）によって実施します(図2)．各イテレーションは，原則としてウォーターフォール型プロセスの作業を一通り実施することで完了します．また，反復型プロセスは，ウォーターフォール型プロセスが抱える課題を解決するアプローチを提案しています．

反復型プロセスは，別名「リスク削減型プロセス」あるいは「リスクドリブン型プロセス」と呼ばれます．イテレーションごとにリスクを確実に軽減していくことで，ウォーターフォール型プロセスのような問題発生を軽減するのが狙いだからです．

反復型プロセスでは，反復ごとに開発する機能を，特に技術的リスクの高いものから選びます．反復の結果，テスト済みの実行可能プログラムができるため，目標としたリスク軽減を達成できたかどうかを検証できることになります．

反復型プロセスは，ウォーターフォールアプローチに比べ，リスクを段階的に軽減させることを目標に進められます．反復ごとに開発する機能を肉付けしていく方法を用いることが多いため，典型的には図？のように開発が進められ，システムが成長していくことになります．

代表的な反復型プロセスには，UP（Unified Process）やRUP（Rational Unified Process），XP（eXtreme Programming），SCRUMなどがあります．

◇ ◇ ◇

以上，ウォーターフォール型と反復型という2種類のプロセスの概略について見てきました．以下では，開発プロセスの各工程の意味と注意点を簡単に解説していくことにします．

要求定義（要件開発）

要求定義とは，開発するソフトウェアの機能要求や制約事項など洗い出し，定義する作業を指します．「要求定義」の作業の成果物が，「要求仕様書」とか「要求定義書」と呼ばれるものであり，開発するソフトウェアの機能要求や制約事項などを記述したドキュメントです．

ソフトウェア開発において，「要求仕様書」は大変重要です．「要求仕様書」に記述された内容を基に，以後の開発作業は進められていきますから，「要求定義」作業が十分に行えず，不完全あるいは誤った「要求仕様書」が作成されれば，それを基にした分析・設計そして実装もすべて誤ったものになってしまいます．

近年，ソフトウェアが複雑になるにつれて，「要

ソフトウェア開発の歴史（1）開発方法論の登場

ソフトウェア開発の世界におけるプロセスの歴史と課題について知識を深めることは，最新のプロセスや利用状況を理解するために効果的と思います．そこで，ここではこれまでのソフトウェア開発の歴史を，プロセスを中心に紹介していきましょう．

ソフトウェア開発はハードウェア開発に比較されることが多く，そのたびに「ソフトウェア開発にはプロセスがない」とか「プロセスが未整備でカオス的作業が行われている」と揶揄されたりします．しかし，ソフトウェア開発において，決してプロセスが無視・軽視されてきたわけではありません．

ソフトウェア開発の世界でも，従来からプロセスに注意が向けられ重視されてはいましたが，それは主に分析・設計手法や方法論といった「エンジニアリング」に主眼を置くものでした．その後1980年代後半から90年代にかけて，ソフトウェアの規模が急激に大規模化され，機能も複雑化していきました．

このような状況に対してソフトウェア開発の現場は，それまでの開発方法や手法に限界を感じ始め，新しい開発手法や方法を導入・検討をしていきます．それに合わせて，1980年代後半から多くのソフトウェア開発方法論が登場し，それらの優劣の議論と共に，競って現場に導入されていきました．

代表的なソフトウェア開発方法論には，オブジェクト指向方法論の「コード・ヨードン法」，イギリスでは主流だった「ジャクソン法」，現IBM-Rationalのジェームス・ランボーたちがGEの研究所時代に開発した方法論で，日本で最も普及したオブジェクト指向方法論だった「OMT（Object Modeling Technique）」，最古のオブジェクト指向方法論で，現在のMDAのバイオニア的存在である「シュレイヤー＆メラー法」，Ada言語を対象とした設計の方法論からスタートし，各種方法論のベストプラクティスを取り入れながら，分析から実装までをカバーしたオブジェクト指向方法論となっていった「ブーチ法」などがあります．■

Process Revolution

求定義」が非常に困難になってきました．複雑かつ大規模化するソフトウェアの機能や制約事項を，矛盾や過不足なく正確に記述することは大変な作業だからです．

分析

分析工程の作業は，開発するシステムの要求機能や制約事項を整理し，ソフトウェアで実現する対象の構造や振舞いの特徴を洗い出す作業です．ソフトウェアで実現する問題対象領域（ドメイン）を，論理的に「静的な構造」と「動的な構造」の両面で形式化します．

オブジェクト指向で開発する場合，「静的な構造」を表現する中心的な成果物は，UMLでは「クラス図」や「パッケージ図（サブシステム図）」，「コンポーネント図」などになります．一方，「動的な構造」は，「状態図」や「コラボレーション図」，「タスク構成およびタスク間通信」などが中心となります．ただし，コンポーネントおよびタスク構成とタスク間通信の検討は，設計で行うのが普通です．

分析工程作業で重要なことは，開発の実行環境（OSやハードウェア環境）や開発言語などを考慮しないで，問題対象領域（ドメイン）の要求機能や制約事項を形式化することです．具体的な実行環境や開発言語への最適化は，「設計工程」で行うことになります．

この分析工程の作業が不十分なまま設計工程の作業に進んでも，優れた設計を行うことは困難となってしまいます．システムの優れた保守性，拡張性を持つアーキテクチャを設計するには，「分析」工程の作業で，しっかりと問題対象領域（ドメイン）の構造や振舞いの特徴を形式化し，把握しておかなければなりません．

設計

設計工程では，分析工程の作業結果を基に，開発の実行環境（OSやハードウェア環境）や開発言語などを考慮し，問題対象領域（ドメイン）の要求機能や制約事項を最適化します．

設計工程の作業の中心は，ソフトウェアアーキテクチャの決定と構築になります．ソフトウェアアーキテクチャの決定では，プログラム実行時のパフォーマンスやコードサイズ，利用するミドルウェア，ライブラリーなども重要な課題となります．

最近主流であるオブジェクト指向設計であれば，オブジェクト指向言語のメカニズムである「カプセ

ソフトウェア開発の歴史（2）開発環境の登場

開発方法論の登場と時を同じくして，ソフトウェア開発を効果的・効率的に進め，品質も向上させることを期待して，CASE（Computer-Aided System Engineering）ツールが大変な注目とともに開発現場に導入され始めました．筆者も，この時期にCASEツールを初めて使用し，開発の設計から実装および設計書などのドキュメントを作成したのを覚えています．

この時代のCASEツールは，主にUNIX上で動作していました．また，値段が高額だったこともあり，すべての企業で導入されるには至りませんでした．

その後CASEツールは，要求分析から実装までをカバーする本格的なCASEツールなどに発展していくものと，MicrosoftのVisual C++などに代表される開発環境の2種類の方向に枝分かれしていきました．■

ル化」や「継承」,「インタフェース」による「ポリモーフィズム」，さらにはデザインパターンやアーキテクチャパターンを利用して，保守性と拡張性をアーキテクチャ上に実現させることを検討するのもこの設計工程の作業です．

当然ながら，設計工程の作業には，OSやミドルウェア等の専門知識も必要とされます．こうした専門知識を身につけ，アーキテクチャの設計を担当するエンジニアは「アーキテクト」と呼ばれ，大変重要な責任を持つことになります．

設計工程での作業では，オブジェクト間の協調関係を詳細に定義し，クラスのメソッドの引数の数や型も検討していきます．ただし，一度の多くの技術的課題や機能を設計し，アーキテクチャを構築するのは困難です．そこで，反復型開発プロセスに従い，少しずつ実現する機能設計の妥当性を検証していくのが通常です．

実装

実装工程では，設計工程作業結果を受けて，プログラミングを行います．

オブジェクト指向開発の場合，設計工程の成果物であるクラス図や状態図が重要な入力になります．最近では，開発環境であるCASEツールの機能が充実してきており，たいていのCASEツールであれば，クラス図や状態図から，プログラム言語のスケルトンを自動生成することが可能です．

テスト

このエ程では，プログラミング言語で実装したソースコードをテストします．テストには，「単体テスト」「コンポーネントテスト」「システムテスト」などと呼ばれるテストがあります．

要求仕様書が作成されれば，並行してシステムテストの計画や作業準備を始めることが可能です．反復型プロセスのRUPでは，テストの初期からのプロセスも明確に定義されています．

テスト工程で注意すべきことは，実装工程の作業が終了してから慌ててテストの準備をするのは避けなければならないということです．ウォーターフォール型プロセスではテストは最終フェーズですので，作業進捗の遅れが生じた場合にはテスト作業期間が圧迫されるため，どうしても不十分なテストになりがちです．

ソフトウェア開発の歴史（3）開発言語の世代交代

オブジェクト指向技術の発展と同時に，プログラミング言語の世界にも，C++言語やAda言語等のオブジェクト指向言語が登場し，利用され始めました．その後，彗星の如くJava言語が登場し，瞬く間に世を席巻したのは記憶に新しいところです．C++言語やAda言語，Java言語は，オブジェクト指向またはオブジェクト指向ベースのプログラミング言語で，それまでのプログラム言語に比べて保守性や拡張性に優れた言語となっています．また，オブジェクト指向設計を言語として実装するための機能を備えているため，設計結果をプログラムにストレートに反映させることが可能になっています．

プログラムの実装作業も，近年は随分変化してきました．プログラミング言語をOからハンドコーディングすることは少なくなってきており，CASEツールによって分析・設計情報からソースコードを生成するのが通常です．CASEツールによっては，100%のソースコードを生成するものも登場しています．

前述のVisual C++や，最近ではJavaのIDEであるEclipseなどに代表される各種ツールは，人に代わって可能な限り単純作業を実行し，生産性・効率および品質の向上を実現する方向で発展してきています．■

Process Revolution

しかしながら，テストは大変重要な作業ですので，最近ではテストを最初に行う「テストファースト開発」と呼ばれる開発作業スタイルも登場しています．

ウォーターフォールの課題

ウォーターフォール型プロセスは，決して悪いプロセスではないのですが，前のフェーズに戻って検討する手順が存在しないために，昨今のように要求仕様の変更が多い場合や，新規性の高いソフトウェア開発の場合には，うまく機能しないプロセスになりつつあります．

要求仕様の抜けや設計の問題点は，開発作業後半のフェーズの実装およびテストフェーズに入ってから発覚する場合がほとんどです．したがって，最悪の場合，納入に間に合わないために開発担当者の増員や，機能の削減，納入時期の延期などで対応せざるをえない事態になってしまいます．

そのため，現在では新規性の高いソフトウェア開発では，ウォーターフォール型プロセスはメリットよりもデメリットのほうが大きいということで，採用されることが少なくなってきています．

ただし，新規性が高くない，システムのバージョンアップにともなう差分開発のような場合には，開発のリスクも低くなるために，ウォーターフォール型プロセスあるいはウォーターフォール型プロセスに近いプロセス（反復回数が少ないとか，アーキテクチャの検討が少ないプロセス）を採用することもあります．

ウォーターフォール型プロセスと反復型プロセスそれぞれの長所と注意点を表1にまとめましたので，参考にしてみてください． ■

参考文献

1.『XPエクストリーム・プログラミング検証編』ジャンカルロ・スッチ他著，小野剛他訳，ピアソン・エデュケーション．
2.「Agileなソフトウェア開発」，橋本隆成著，『Software People』Vol.1，技術評論社．
3.「実践！CMM導入徹底ガイド」，橋本隆成著，『Software People』Vol.2，技術評論社．
4.「知識創造とソフトウェア開発」，山田正樹著，『Software People』Vol.2，技術評論社．
5.『ラショナル統一プロセス入門 第2版』，フィリップ・クルーシュテン著，藤井拓監訳，日本ラショナルソフトウェア（株）訳，ピアソン・エデュケーション．
6.『UMLによる統一ソフトウェア開発プロセス オ

ソフトウェア開発の歴史（4）課題とプロセス

開発方法論，CASEツールおよびプログラミング言語は発展し続けており，うまく導入すれば効率が向上するのも事実です．しかしその一方で，現在においてなお，当初期待したほどソフトウェア開発の効率や品質が向上していないことも事実なのです．いったい何が原因なのでしょうか？そして，何が不足しているのでしょうか？

最近では，CASEツールなどの開発環境やソフトウェア開発のエンジニアリングに焦点を当てた開発方法論だけでは効果は期待できないということで，ソフトウェア開発のプロジェクトマネジメントの重要性が改めて強調されています．

また，携帯電話，自動車の車載システムなどに代表されるように，開発するシステムがソフトウェア開発技術や環境の発達以上に複雑かつ大規模化してきており，私たち開発者が対応しきれていないことが最も大きい原因になっているといえるでしょう．これまで常識と思われてきた開発技法や方法論が，さまざまな問題を抱えていることも明らかになってきています． ■

	長所	注意点
ウォーターフォール型	①上流工程から下流工程に至るまで，明確に作業の順序が定義されている ②各フェーズの仕様書，設計書など成果物が明確にされ，わかりやすい ③プロセス自体がシンプルなので，適用しやすい	❶後戻りのできないプロセスのため，上流工程で欠陥があると，下流工程がそれを負担することになる．したがって，開発後半に致命的な修正がかかる危険がある ❷開発後半になるまで実装･テストを行わないため，設計に欠陥があっても開発後半になるまでわからない ❸各作業（要求・分析・設計・実装・テスト）は1度だけしか行わないため，要求仕様の変化に対応するのが困難 ❹ソフトウェア構造に対する開発指針が弱く，ソフトウェアアーキテクチャへの考慮が不十分
反復型	①上流工程から下流工程までの作業を何度も繰り返すので，開発依頼者（ユーザー）の要求とプログラムとの間のズレを早期に確認・修正したり，要求の変更に対応したりすることが可能 ②最新の開発技術や不慣れな技術を扱う場合でも，検証しながら開発を進めるため，リスクを削減可能 ③限られた期間の中で，優先度の高い機能から動作検証しながら開発を進めることが可能 ④保守性・拡張性の高いソフトウェアアーキテクチャを，検証しながら開発することが可能	❶従来のウォーターフォール型の進捗管理とは異なる進捗管理が必要になる ❷各反復（イテレーション）ごとに戦略的かつ明確なマイルストーンを設ける必要がある

表1　ウォーターフォール型プロセスと反復型プロセスの比較

ブジェクト指向開発方法論』，イヴァー・ヤコブソン他著，藤井拓監修，日本ラショナルソフトウェア（株）訳，翔泳社．

7.『ソフトウェアプロジェクト管理』，ウォーカー・ロイス著，日本ラショナルソフトウェア（株）監訳，ピアソン・エデュケーション．

8.『XPエクストリーム・プログラミング入門　ソフトウェア開発計画の究極の手法』，ケント・ベック著，長瀬嘉秀監訳，飯塚麻理香訳，ピアソン・エデュケーション．

9.『XPエクストリーム・プログラミング実行計画編』，ケント・ベック他著，長瀬嘉秀監訳，永田渉・飯塚麻理香訳，ピアソン・エデュケーション．

10.『決定版 プロジェクト管理　成功するソフトウェア開発の最新スタイル』，橋本隆成著，技術評論社．