連載　企業および社会における情報システムの意味を考える
第 10 回　開発工程の考え方

大島　正善（MBC: Method Based Consulting）

1. ウォーターフォール・モデルとアジャイル方式

先月は、保守工程を維持・改善工程と考えるべきと書きました。そう考えることによって、開発という作業のとらえ方が変わってきます。とはいっても、従来の考え方を一気に変えるのは簡単ではありません。今回は、従来の考え方に基づく開発工程モデルというハウツーの“なぜ”について書いてみます。

情報技術は総じて手段でありハウツーです。ハードウェア製品やソフトウェア製品もそれを導入することが目的ではありませんし、設計手法や分析手法など開発方法に関することもそれを使うことが目的ではありません。

世の中にはさまざまな開発方法論がありますが、それぞれ成立の背景なり目的があります。日本では、そういった背景や目的はほとんど語られることなくITベンダーや雑誌、IT動向の調査会社などにより宣伝され使われてきました。今でも、その風潮は変わりません。

手段でありハウツーであるということは、料理でいえば、たとえば包丁にあたります。IT業界のベンダーは確かに包丁作りを手掛ける企業なのですが、それを利用する企業などの組織が欲しいと思っているのは“おいしい料理”なのです。どういう料理を食べたいかを知っているのは顧客であり、その料理を作るのはコックですが、日本では、包丁作りだけでなく、顧客の役割もコックの役割もITベンダーが行っているところが多いわけです。包丁作りに顧客が食べたいと思っている料理を作れるかどうかはあやしいかもしれません。コックが考えるべきことは、顧客が欲しがる料理は何かということであり、包丁は多くのものの中から選びだすものの一つにすぎません。魚包丁作りの名人が“オレの作ったのはすばらしいぞ”と言ったからと言って、それを鵜呑みにしてケーキ作りの名人がそれを使うなどということは考えられません。しかしながら、IT業界では、そういうことがフツーに行われているようです。きわめて危険です。

IT業界では、かれこれ２０年ほど前からウォーターフォール・モデルの評判は良くありません。８０年代までは、ホスト・コンピューター中心の時代であり、ディスプレイ端末を使って操作するシステムを開発するだけで十分でした。業務の合理化が目的のコンピューター・システムの導入がほとんどであり、稼働後にバグを出さないシステムを確実に開発することが開発作業の主たる目的でした。９０年代の半ば、クライアント・サーバーの時代となり、使いやすさが強調され画面設計の変更が容易になる開発ツールが使いだされた当初から、ユーザー要件は簡単には決まらないので、スパイラル・モデルやイテレーションあるいたイテラティブな開発が望ましいということが言われ始めました。その中でScrum[*1]をはじめアジャイル方式の開発スタイルが提唱され、テスト・ファーストというようなことも語られ、ウォーターフォール・

モデルの開発では決して成功しない、とでもいうような話され方がされることもあります。超の付く一流どころのエバンジェリストが「アジャイルでやるべきだ」と語っているからアジャイルでやってみようか、というわけにはいきません。もちろん、そんな判断をされる方はいないと思いますが、自由奔放をイメージさせる「アジャイル」という言葉は、「フラッと」してしまいそうな良い響きを持っています。[*2]

しかしながら、開発方法論は包丁にすぎません。製品ソフトウェアは、まさに包丁でよいのですが、企業などの組織における情報システムの開発では、どのような料理を作るのかということが命題です。そのような違いを理解して、製品作りの名人が編み出した方法論を鵜呑みにして適用するのは、考えものです。工程モデルという包丁は、対象に応じて選択すべきものです。さらに言えば、今ある開発工程モデルも、そのうち古くなり使い物にならなくなる可能性は否定できず、採用にあたっては常に批判的精神が求められます。そして、批判的精神の根幹は、“なぜ”（目的）を考えることです。

今でも企業などの組織における情報システム開発のプロジェクトにおいて、ウォーターフォール・モデルは採用され続けています。政府調達案件では、フェーズごとに契約を分けることが推奨されているので、ウォーターフォール・モデルを採用せざるを得ないということもあります。大手のSIベンダーの提案では、多くの場合ウォーターフォール・モデルを採用しています。特に開発規模が一定以上になるとその傾向が顕著です。このことに関しては、異論はないと思います。なぜなのでしょうか?ベンダーに発注する企業からすると解せないことかもしれませんが、提案するからには理由があり、知っておいても損はありません。

国際標準であるSLCP(System Life Cycle Model[ISO/IEC 15288]または Software Life Cycle Model[ISO/IEC 12207]，あるいはＩＰＡが策定している共通フレーム2013）の作業定義（プロセス、アクティビティ、タスク）が、上流から下流に流れるように記述されているのも事実です。しかし、そういった標準工程モデルは、当学会の方々には釈迦に説法かと思いますが、時間軸とは無関係であり作業の順番を規定しているわけではありません。

時間軸とは無関係といっても、次のようなことは可能でしょうか? （以下、カテゴリーAの作業と記す）

① 運用を開始してからプログラムを開発する

② プログラムをテストしてから、設計を開始する

③ テスト・ケースを洗い出してから要件を決める

④ プログラムの設計を行ってからアーキテクチャを決定する

⑤ コーディングしてから要件を決める

さて、いかがでしょうか?いくら時間軸から独立だとはいえ、こういった順番での作業は考え

られません。やはり作業には順序があります。では、以下のようなことはどうでしょう。(以下、カテゴリーBの作業と記す)

⑥ プログラムをある程度開発してから、詳細な設計をつめ問題のある
　設計を見直す

⑦ 完成品のイメージを見せてから、設計を確定させる

⑧ プログラムが完成してから設計ドキュメントを完成させる

⑨ 運用時の姿を明確にしてから、要件の詳細を決める

このようなことは可能ですし、実際に大きな問題とはならずに行われています。いずれも、下流から上流に手戻りが発生しているように見えますが、カテゴリーAの作業とカテゴリーBの作業は何が違うのでしょうか?そのことを考えると、開発プロセスをどのように決めるべきかという本質が見えてくるように思います。

システム開発における開発工程モデルは、もの作りでの工程管理モデルの考え方を取り入れています。開発プロセスは生産工程であり、手法は一個流し、平準化、あるいは単能工方式かセル生産方式かという生産方式や、材料や部品をどういう順番で混ぜたり組み合わせたりするのかという手順の規定に該当すると考えられます。また、設計・開発やテストのツールは設備と考えることができます。

製造業でのもの作りでは、上流工程から下流工程に流すのが当然であり、不良が見つかったら作業を停止させるべきとは言われますが、アジャイルのやり方で作業を継続させるべし、ということが語られることは決してありません。IT業界ではそうではなく、作りながら見直すというやり方が推奨されていますが、何が違うのでしょうか?アジャイルで開発するというやりかたが推奨されるのは、なぜなのでしょうか?

2. 開発工程定義のための前提要素

開発工程は、多くの要素を考えて決定するものです。その要素を決め、そして要素を評価することがまさに、“なぜ”に答えることになります。ある一つの要素だけで、ウォーターフォールにするのかアジャイルにするのかを決めるということはありません。もちろんその要素の中に、“世の中での普及の程度”や“評判”などということは含まれません。考慮すべき要素は多くあり、また、それらが複雑に絡み合っているのできれいに分類・整理することが難しいのですが、重要な要素がいくつかあります。

ウォーターフォール・モデルが批難されるとき語られるのが、「要求事項を最初に確定するのは困難である」ということです。要求事項がどの程度変化するのかというのは、確かに要素の一つです。しかし、「要求事項が変わる」という時に問題としていることは、「ビジネス環境が変化して要求事項が変わる」ということよりも、「要求事項を文書にしただけではコミュニケー

ション・ギャップが生じる」ということを問題としていることの方がはるかに多いのが実態でしょう。ある事実を文字で表して伝えるよりも、画像や写真など図で表して伝える方が、はるかに正確に伝わるということはよく知られています。そのために、プロトタイプを作りながら要求事項を確定していくということを繰り返すことが行われます。しかしながら、それが可能なのは多くはユーザー・インターフェースに限った話で、ビジネス・ロジックに関しては、プロトタイプを作成して確認することは困難です。複雑で量の多い組織のルールや法律などの制約条件を、業務上の要件としてきちんと文章化（可視化）しないで、アジャイル方式で開発すればうまくいくとは考えるのは間違いです。つまり、「要求事項の確定が困難である」ということは、「ビジネス環境の変化が発生しうる程度（本当の変化の多さ）」、「UI設計の重要度と複雑さ」、「ビジネス・ルールの可視化の必要性の程度」という３つの要素に分解して評価してみるべきです。[*3]

それ以外に考慮すべき事項として、開発規模があります。規模が小さければ、誤解を恐れずに言えば、開発工程をどうすべきかということは、実際には重要な問題ではないでしょう。業務プロセスが複数の組織をまたがる大きな事業のシステム開発と、範囲が限られた業務システムの開発とでは、開発のやり方は変わります。

規模が違うということは、開発体制が違うということでもあります。人数、チーム数、企業とベンダーとの役割分担も重要な要素になります。業務要件のまとめを誰の責任で行うのか、あるいは、ベンダーが参加する場合契約方式はどうするのか、そういった要素も開発規模から影響を受けます。開発工程モデルの選択は、技術的にどちらがすぐれているのかという議論になりがちですが、技術的要素よりも、どのような体制が組めるのか、共有すべき情報が的確に伝わるのか、ということがより大きな要素です。

もう一点重要な要素があります。それはどういったツールの利用ができるのか、といったことです。手工業方式で開発を行うのか、それとも、設計・開発ツールを駆使しながら作業を行うのか、それによってやり方は異なります。ツールを使う場合、人材確保が前提となります。ウォーターフォール・モデルは、すべて人手で開発をするということを前提とした開発工程モデルだと言っても過言ではありません。一方、アジャイル方式にしろ、プロトタイプ手法にしろ、イテレーション方式にしろ、基本的に設計・開発ツールを前提にした考え方です。もちろん、ツールの機能として、設計の変更による影響分析ができる機能を持っていることが必要です。人手で変更を管理することしかできないツールでは意味がありません。

開発工程の考え方の中に、インクリメンタル方式というのがあります。いってみれば、最初は小さく作って一部の機能のみリリースし、徐々に機能を追加していくという考え方です。これは、製品系ソフトウェアや、Ｗｅｂシステムであれば実現しやすいやり方ですが、企業の基幹業務システムでは、採用は困難でしょう。なぜなら、業務機能の一部だけ切り離して先行して稼働させるというのは難しいからです。（絶対できないわけではなく、多少の手作業のムダを許容するという決定を行うことができれば可能です）

基幹系システムは、データベースの設計が鍵となるので、一部の機能のみ先行開発しても、機能を追加するとデータベース設計に大幅な変更が入る可能性があります。すると、先行リリースした機能にも手を入れなくてはならないという事態になります。そのため、先行リリースしたコードに手を入れなければならなくなることがあります。そういった手間と手戻りを許容できなければ、インクリメンタル開発を基幹系で実現するのは困難かと思います。

アジャイル方式を採用した開発をしたいのだけれど、体制はどうすべきか?という議論を聞くことがよくあります。しかし、本来は話が逆で、開発規模が決まり、開発対象システムの特徴が見えてきたときに、どのようなやり方がふさわしいのかを決め、それに合わせた体制を検討すべきです。ウォーターフォールなのかアジャイルなのかを先に決めて、組織体制を確立するわけではありません。開発工程は、高品質のソフトウェアを開発するための開発手順という手段にすぎず、その前提となる要素を明確にすることが大切です。[*4]

3. 開発工程は、“最後から考える”のが王道

開発工程を考える上で重要な点があります。それは、“後ろから考える”ということです。別の言葉でいえば、ターゲット志向であるということです。ターゲットを明確にせずに開発工程を定義しそれに従えばシステムが完成するということはありません。ある意味で、トヨタ流の「後工程はお客様」という考え方と同じです。

何年後のことかわかりませんが、人が考えたことをそのままソフトウェアに実装できる手段が発明されたとします。そのような状況では、今行っているような要件定義、設計や開発という作業は不要になっているでしょう。あるいは、ビジネス・プロセス、ワークフロー、ビジネス・ルール、情報モデル、アーキテクチャを入力したらソフトウェアの生成が可能になったとします。（いわゆるモデル・ドリブン・アプローチです）その場合も、開発のやり方は今とはずいぶん変わったものになるでしょう。

これらのことは、次のことを意味しています。作業プロセスは、アウトプットを生み出すために定義されるのであり、アウトプットを生み出すためにどのようなインプットから何をすればそのアウトプットを生み出せるのかを考えて定義したのが開発プロセスである、ということです。（余談ですが、業務プロセスも同じ考え方で設計します）人が考えた事柄を、そのまま自動生成できるのであれば、設計やプログラム開発といった作業は不要です。

現在、多くの文書化された要求事項をインプットにして、一気に（設計という作業もなく）全体のシステムを作り出すことはできません。そこで、全体システムを構築する前に、いくつかのサブシステム単位で開発を行うのがよいと考え、それらを集積するサブシステム統合という作業を最後に定義しています。そのサブシステムでも規模が大きすぎるので、サブシステムよりものさらに小さな単位に分割しておこうということで、一定の規模の単位で機能群を定義し、その単位で開発をするような工程定義を行います。同様に、プログラム開発の前には設計

が必要で、その前に要件を定義することが必要である、という発想で開発工程が定義されます。“後ろから考える”とはこのようなことであり、開発工程モデルは、基本的にこのような考え方をベースとして生み出されています。最初にプランニングを行い、次に要件定義、そして設計...というように順番に作業を行えばシステムが完成する、という考え方では決してありません。

最初に，カテゴリーＡとカテゴリーＢの作業は、何が違うのかという問いかけをしました。実は、カテゴリーＢの作業は、最終アウトプウト（ターゲット）を見据えた作業といえます。ですから工程としては逆に見えても問題ないのです。

システム開発に参加する多くの技術者にとって開発工程は与件となります。したがって、規定された順番に従って作業を行えば、きちんとシステムはできあがると考えていても不思議ではありません。ウォーターフォール・モデルを採用する場合、プロジェクト・メンバーがこのような錯覚を持つことが、実は大きな問題です。川の流れは上流から下流に流れますが、そこを流れる水滴一つずつは、上流にいたり下流にいたりします。ウォーターフォール・モデルも同じで、全体としてみれば、要件定義は最初に行われますが、個々の機能になれば、要件の定義を行っているものや、設計を行っているものがあるのは技術論としてはおかしくありません。大切なのは、最終ターゲットを意識しながら作業をするということです。ウォーターフォールで作業を進めても、問題が起きれば（つまり、最終ターゲットの達成が危ぶまれる事態になる）前に戻ることは、技術的には問題ありません。それが問題となってしまうのは、ウォーターフォールの開発モデルから逸脱するからではなくコストのコントロールが難しくなるからです。もちろん、プロジェクト・コントロールの力量も試されます。

要求事項を整理するという作業を最初に行うのですが、実はその目的は、「最終アウトプットの姿を描く」ことです。設計作業の前提を書き留めているわけではありません。ターゲット志向ができている場合とそうでない場合で、同じ“要件定義”という言葉を使っても、結果に大きな差が生じます。要件定義のアウトプットは、稼働後の業務とＩＴシステムの姿を示すものであり、それを目指して設計や開発を行うのだという意識を持つと、開発作業の意味が理解でき、問題が発生しても解決のスピードが早まるでしょう。

このような意識を持つと、要件変更は受け付けないという判断は、それが間違ったものでない限り本来ありえないことだということが理解できるでしょう。稼働後の業務とＩＴシステムの姿を検討しているのですから。

そして、要件変更があり得るという前提でシステム開発を行うことになると、変更要求を設計に反映しやすい仕組み、つまり、ツールの利用が前提（使用しなければ、成功しない　ということ）となるでしょう。また、予算の使い方のコントロールと優先順位づけが重要となり、ベンダーとの役割分担や契約方式も大きな課題となってくることは当然です。変更を受けつけたくないのは、契約金額内でコストのコントロールができないリスクがあるからであり、そのような契約方式を採用したまま、ベンダーがアジャイルで開発することは基本的に無理があるといえます。発注側企業もベンダーも損をする契約になります。この点は、それだけで大きな

課題ですので、別のテーマとしたいと思います。

［*1］ スクラムガイド（日本語版）
http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20JA.pdf#zoom=100

［*2］ アジャイル方式には多く要素があり、これがアジャイルだという定義はあいまいです。アジャイル開発宣言が有名ですが、それは理念の宣言です。ここでは、Scrum ベースのチームの共通理解と結束を重視したやり方のことをアジャイル方式ということにします。なお、アジャイル方式とウォーターフォール・モデルを並列に比べるのは無理があります。ウォーターフォール・モデルは、工程全体にわたって何(What)をするのかという規定ですが、もう一方は作業のやりかた（How）を扱っています。ウォーターフォール・モデルと並列に比べるのであればスパイラル・モデルが妥当です。別の見方をすれば、技術論だけを見れば、ウォーターフォール・モデルをベースとしたアジャイル方式は可能だということです。

[*3] ユーザー・インターフェースが機能を決定するのか、データ構造やビジネス・ロジックが機能を決定するのかという問題があります。たとえば、Office ソフトや開発ツールといった製品系ソフトや Web のユーザー・インターフェース部品のようなソフトウェアを開発する場合は、使い勝手といった見せ方が機能決定の重要な要素です。そういった要件は文書化で伝えることは明らかに困難です。よって、文書化を軽視することは悪いことではありません。一方、ビジネス・プロセスやビジネス・ルールの実装に関わるソフトウェアは、扱うデータ構造の設計とプロセスやルールの可視化・文書化が設計の前に必要です。２－３名で開発できる規模であれば、開発するだけでした文書化しなくても可能です。しかし、メンテナンスを開発担当者と違う人が行う可能性があるのであれば、業務上の要件を明確にした文書は必要です。よく、仕様は「コードを見ればよい」という優秀なプログラマーがいますが、ビジネス要件とコードは１：１ではないので（つまり抽象度が違うということ）、コードが読めてもビジネス要件がわかるということは決してありません。恐らくそういうことを言うプログラマーは製品系ソフトの開発技術者ではないかと思います。そうであれば、コードにコメントがあれば問題ないでしょう。

[*4]アジャイル　プロジェクト　マネジメント（ジム・ハイスミス著,平鍋健児他訳 ）によれば、アジャイル方式が対象とする製品開発は、次の分野としている。１）ソフトウェア製品、２）組み込みソフトウェアを必要とする工業製品、３）ある程度の機動性、探索が求められる企業内情報システム

以上