15-A-5

ビジネスへの貢献が求められる時代のソフトウェア開発の考え方 [1]

～超高速開発ツールがもたらす方法論のイノベーション～

1. テーマと背景

情報システムが組織活動と一体となっている、という考えに反対意見を唱える人は、今ではほとんどいないと思われる。情報システムが単に業務の効率化の道具ではなく、新たな市場・顧客・調達先を獲得することに貢献する道具だとの考えが浸透している。企業だけでなく、あらゆる組織において、情報セキュリティの問題も含めて、“情報をどう扱うのか？”、“情報とどう向き合うのか？”ということが今の時代以上に、一般人を含めた多くの人から脚光を浴びている時代はない。

現在は、かつてのように軍事や企業の戦略立案を行う人たちだけでなく、一般の人々を含めて情報活用の術（どう扱うか、あるいは、無視して扱わないかということ）が求められる時代である。このことは、「情報システムを“構築すること”に価値があった」時代から、「情報システムを“活用して価値を生み続けること”にこそ価値がある時代」に変化したということを意味している。

ビジネス上の価値を生み出すために、人の活動を助け、あるいは、代替する役割を担う情報システム（特にアプリケーションシステム）は、適用領域をますます広げている。それでも、相変わらずバックログは減らないどころか、増加している。新しい情報機器の登場もあるし、新たなシステム化（ソフトウェア化による問題解決）のニーズは、ますます増え続けているからだ。

一方、現在の多くの情報システムは、「ビジネス環境の変化に迅速に対応できる」状態ではないことが問題となっている [2]。それは、ユーザー企業から情報システム開発のノウハウが失われてしまったことを示しており、その原因として次のようなことが挙げられる。

（1）　システム開発と保守の仕事をシステム開発の専門企業に長年委ねてきた。

（2）　別の言い方をすれば、ユーザー企業が情報システムの開発と保守作業のオーナーシ

---

[1] 事例提供: 一般社団法人 ICT 経営パートナーズ協会／MBC（Method Based Consulting）
大島 正善 氏

[2] 64%の中小企業が“業務にあったシステム機能の変更や追加”が課題と挙げている[1]。 また、“法制度などの新設、変更に伴う改修などの柔軟性、拡張性の向上”が保守・運用コストの削減、初期導入コストの削減に続いて、第三位に挙げられている[2]。

ップを失ってしまった。

(3)　企業の事業や業務が複雑になり、その全体像をつかんでいる人材がユーザー企業内にも、ほとんどいなくなってしまった。

(4)　現在の基幹システムの基本的な機能の多くは、20 年程度前に開発されたものであり、全体像をつかんでいる人がほとんどいなくなってしまった。

(5)　業務の詳細な仕組みや手続きは、ソフトウェアで実装されているにもかかわらず、そのソフトウェアの中身を深く理解している人がほとんど残っていない。

このような状態をいつまでも放っておくことが望ましいわけはない。こういった状態になっているのは、ソフトウェアシステムの構築のやり方のどこかに、大きな欠陥があると考えざるをえない。その中には、ソフトウェアの構築プロセス、成果物の管理、発注側と受注側の契約のありかたなど、さまざまな要因が考えられる。

もちろん、一旦作り上げたアプリケーションシステムに機能を追加したり修正したりすることは大変な困難を伴うのは事実である。作り上げたソフトウェアの仕様はプログラムにしか残っていないと言っても過言ではない。開発時に作成した外部仕様や内部仕様は、プログラム開発時点ではすでに古いものになってしまい（プログラムの変更と同期をとって修正することは費用が嵩んで現実的ではない）使い物にならないのである。

数年前から、ソフトウェアの設計情報や業務視点で記載された機能仕様をリポジトリ[3]というデータベースに管理できる開発ツールが市場に提供されている。リポジトリを持つ開発ツールは、従来困難とされてきた「一旦作り上げたソフトウェアに機能を追加したり、機能を変更したりすること」が、きわめて容易に実現できる（理由は 2.4 に記載）。こういったツールを活用することで、情報システムの開発と保守のやりかたが変わるだけでなく、真に“ビジネスの変化に迅速に対応できる情報システム”を構築することが可能となる。

本稿では、そういった“リポジトリを持つ開発ツール”が実現する価値を示しながら、ソフトウェアの開発と保守のやりかたにどのような変化をもたらすのかを明らかにすることを狙いとして執筆した。

2. ビジネスの変化に迅速に対応できる情報システムに求められること

2.1. ソフトウェアはどのような状態として作り上げられるべきなのか？

ソフトウェアの構築上のさまざまな課題への対策を一足飛びに考える前に、そもそも“ビジネスの変化に迅速に対応する”ためには、ソフトウェアシステムがどのような“状態になっている”必要があるのか、ーつまりゴールの姿であるー、その要件を考えてみたい。

[3] 業務の機能やソフトウェアの設計の仕様を保持しているデータベース。詳細は、「超高速開発が企業システムに革命を起こす」[3]を参照のこと。

そのことを考えるためには、現状がどうなっているのかを整理するのが分かり易い。現状のソフトウェアは、ある意味で、情報システムを“変化に迅速に対応できないような状態を持つものとして作ってしまった”ということでもある。以下のような状況にあるのが、現在のソフトウェアである。(とりあえず、分かり易い順に記載する)

(1) 設計仕様を記述したドキュメントとソースプログラムが乖離している。
(2) ソフトウェアの全体構造が可視化されていない。
(3) 事業部門・業務部門のビジネスパーソンが、情報システムの開発・保守とはかけ離れた位置にいる(物理的な距離というよりも、コミュニケーションの距離と頻度)。
(4) ビジネスの変化を生み出す対象や要素を、ソフトウェアの構成要素と結びつけて設計していない。
(5) ビジネスの要素とソフトウェアの関係付けがなされていない。

(1)から(3)は特に説明は不要であろう。(4)と(5)は“ビジネスの変化に迅速に対応できる”ようにするための本質的な要素を含んでいるので説明したい。(詳細は参考文献[3]を参照していただきたい)

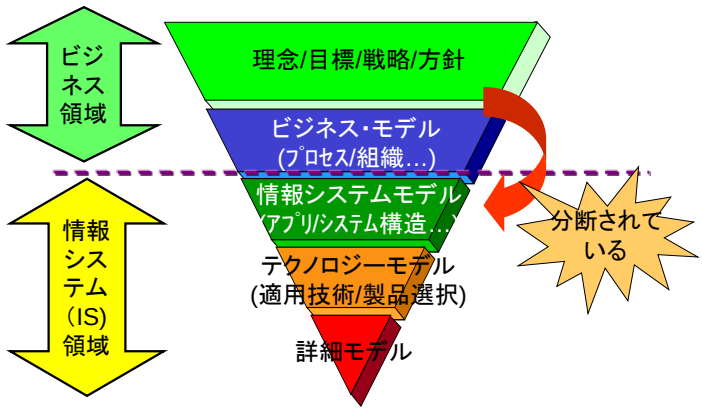

図 15-A-5-1 ビジネス領域と情報システム領域の分断

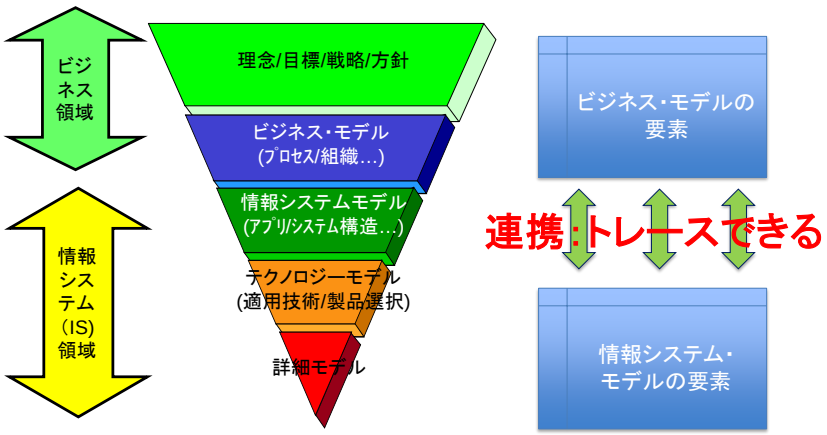

図 15-A-5-2 ビジネス領域と情報システム領域の連携

図 15-A-5-1 は、上位の二つの階層がビジネス領域を示し、下位の３つの階層が情報システムの領域を示したものである。

現行の情報システムは、“ビジネス領域の要素と情報システム領域（特にアプリケーションソフトウェア）の要素が関連付けて管理されていない”という意味を深く理解する必要がある。そのことが理解できれば、何をすべきかが見えてくる。“関連が取れている”という状態は、２層目の要素と３層目の要素の間に“トレースできる状態になっている”ということである。ビジネスモデルの構成要素と、情報システム領域（特にアプリケーションソフトウェア）の要素との関連が取れていることを示すのが図 15-A-5-2 である。

2.2. ビジネス領域の要素は多次元の関連を持つ

この点をもう少し具体的に見てみよう。“ビジネス領域の要素”には、顧客、商品・製品・サービス、組織、ビジネスプロセス、ビジネスルール、生産拠点、販売拠点、調達チャネル、販売チャネルなどがある。もちろん、企業理念や長短期のゴール、あるいは戦略・戦術という要素もある。ビジネス活動はそういったものの組み合わせで成立している。そして、それぞれの要素間には、例外を除けば、すべて多対多の関係がある。それを示したのが、表 15-A-5-1 である。〇を付与した箇所は、関連が存在する箇所である。

表 15-A-5-1　ビジネスを構成する要素間の関連

		商品・サービス					
		事業A	事業B	事業C	事業D	事業E	事業F
顧客	セグメント1	〇			〇		
	セグメント2	〇	〇	〇	〇		
	セグメント3		〇	〇	〇	〇	
	セグメント4			〇	〇	〇	〇
	セグメント5				〇	〇	〇

顧客も商品・サービスもさらに詳細化する

		商品・サービス					
		事業A	事業B	事業C	事業D	事業E	事業F
組織・体制	組織1	〇	〇	〇	〇		
	組織2					〇	〇
	組織3	〇	〇		〇	〇	
	組織4	〇				〇	
	組織5			〇			〇

組織・体制も商品・サービスもさらに詳細化する

		商品・サービス					
		事業A	事業B	事業C	事業D	事業E	事業F
調達先	調達先1	〇	〇				
	調達先2		〇	〇		〇	〇
	調達先3	〇			〇		
	調達先4	〇				〇	
	調達先5			〇			〇

調達先も商品・サービスもさらに詳細化する

		チャネル（販売、製造、調達、マーケティング…）					
		チャネルA	チャネルB	チャネルC	チャネルD	チャネルE	チャネルF
方針	戦略1	〇	〇				
	戦略2					〇	〇
	戦略3	〇		〇	〇		
	戦略4	〇					
	戦略5	〇				〇	

戦略もチャネルもさらに詳細化する

たとえば、商品と自社組織（たとえば営業組織、営業担当者、ヘルプデスクの担当者など）との関係で言えば、一つの商品が複数の組織で販売され、一つの組織が複数の商品を販売しているとすれば、商品と組織との関係は“多対多”ということになる。表 15-A-5-1 に示した以外でも、「生産拠点と製品の関係」、「目標と戦略の関係」、「自社商品と競合商品の関係」のどれをとっても、１対１（または１対多）の関係であることは例外であり、ほとんどすべてが、多対多の関係となっている。ビジネス活動というのは、きわめて複雑な構造をしているのである。

そういった、ビジネス活動の多様な要素間の関係の全体像を示したのが、図 15-A-5-3 である。

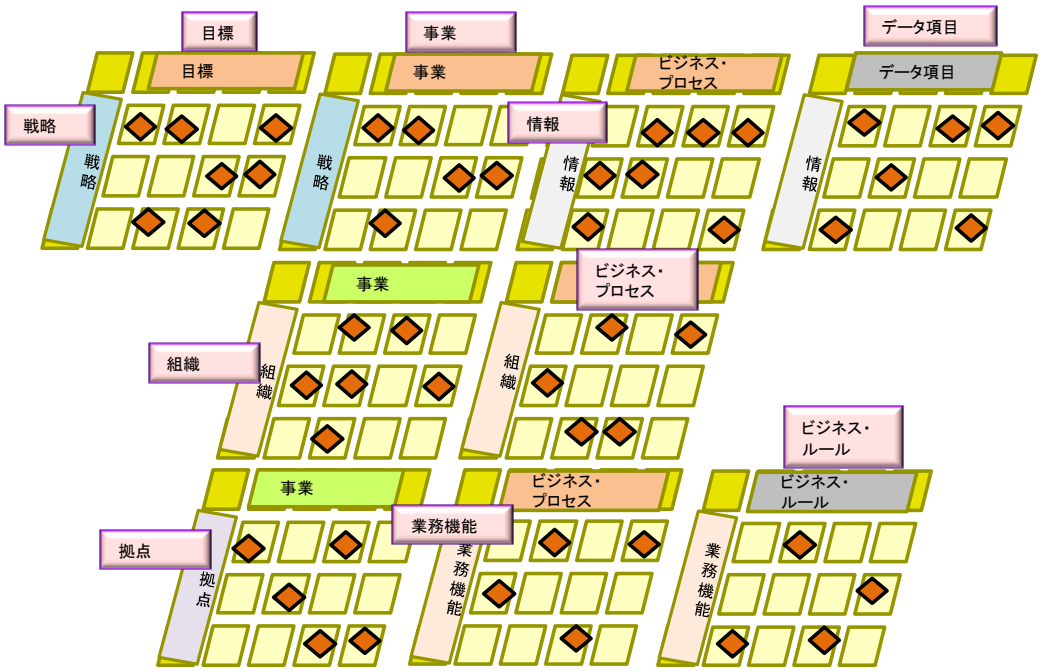

図 15-A-5-3　ビジネスを構成する要素間の全体の関連

ビジネス活動のあらゆる要素は絶えず変化している。日常活動の取引に基づく変化（売上、利益、在庫といった変化）は、情報システムが持つ「受注」や「発注」、「仕入」、「入出庫」といたトランザクション処理の結果を通じて知ることができる。その変化に基づいて人が諸々の判断をするだけで済んでいる（ただし、販売と物流と生産・調達の情報流が途中で途切れ時間差が生じることは稀ではない）。しかし、こういった取引に基づく変化ではなく、ビジネスモデルの要素間の変化に対しては、現在の情報システムは、簡単には対応できず、事業の変化を起こそうとすると障壁になりかねない。新商品の開発や、競合製品の台頭、価格戦略の見直し、災害などの突発的事象への対応、ビジネスルールの変化などへの対応がその例である。特定のビジネスプロセスやルールの前提の上に成り立っているのが情報システムなので、そういった事象が発生したときに、影響分析がどうしても必要になる。しかし、そ

れを効率よく行える手段が必ずしも存在するわけではない。なぜなら、ビジネスの要素と情報システムの要素との間にトレーサビリティが存在しないからである。

2.3. ソフトウェアも多次元の構造物である

一方、情報システム（その中でも特にソフトウェア）の構造はどうなっているか。それを概念図で示したのが図 15-A-5-4 である。この図は、ソフトウェアの全体が、ツリー構造（Structure）ではなく、多次元（Multi-Dimension）のマトリックス構造になっていることを示している。この認識は非常に重要であり、恐らく、今までソフトウェア開発に携わってきた技術者やマネジメントの完全な盲点となっているところと思われる。人は、このような多次元の構造体の姿をイメージすることはできない。宇宙の構造が 9 次元とか 10 次元とか言われても、その姿をイメージできないのと同じである。

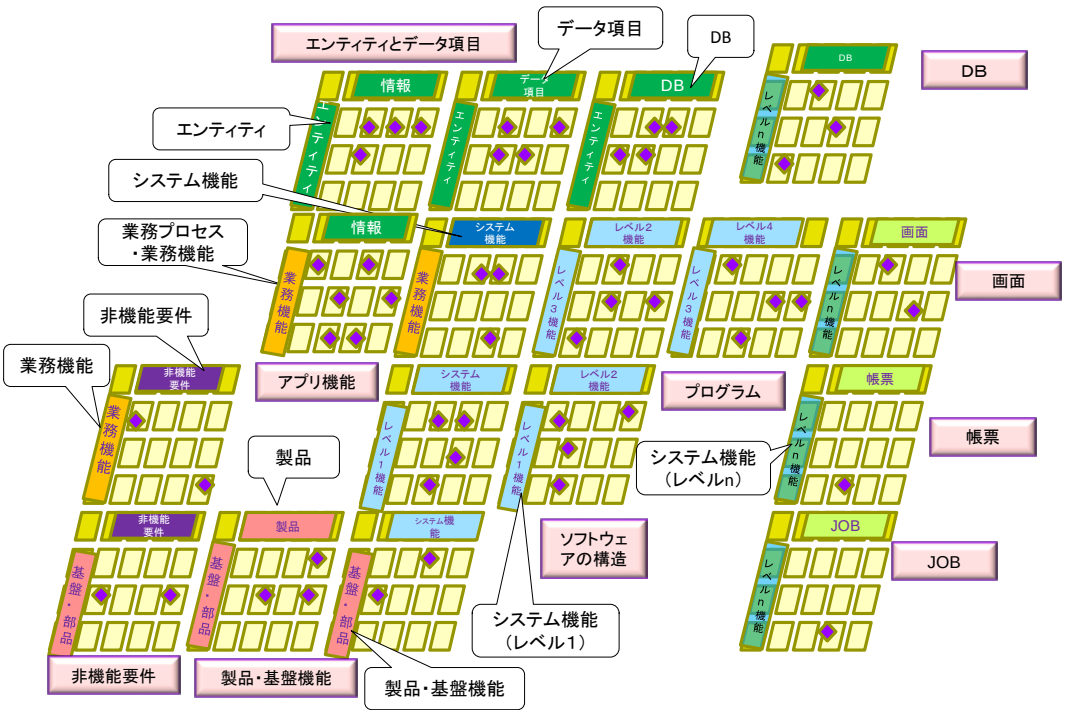

図 15-A-5-4　ソフトウェアシステムを構成する要素間の関連

たとえば、図 15-A-5-5 を見てほしい。この図は、よく見られるシステム機能構成図である。左に業務機能が示され、それがブレークダウンされてシステム機能が定義されたことを示している。これとまったく同じでなくとも、これと似た図は多くのプロジェクトの成果物の一つとなっているであろう。しかし、これはビジネス機能とシステム機能の関係も含め、すべての要素間の関係があたかも“ツリー構造”であるかのように描いている点において、基本的な間違いがある。実際には、どの関係もディメンジョン（マトリックス関係）で示すのが適切である。

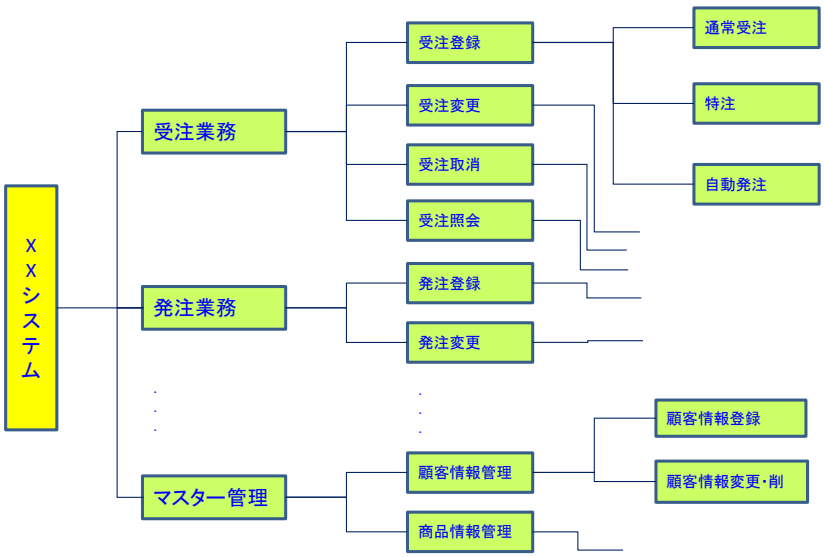

図 15-A-5-5　一般的なシステム機能階層図

要素間の関係がすべてマトリックスの関係にあり、全体の構造が多次元であるということは、システム構築プロジェクトの作業に抜本的な見直しを要求する。システムの“構造（アーキテクチャ記述や設計の要素間の関係を示す成果物すべて）”に係る成果物のみならず、開発方法論やマネジメント手法も抜本的な見直しが求められる。品質マネジメント＆コントロール、進捗マネジメント＆コントロール、コミュニケーションマネジメントなども、これにより大きな影響を受ける。そもそも、ソフトウェア構築という作業に、大量の要員を投入する労働集約的な作業のやり方が、はたして向いているのだろうか？という疑問も生じる。

プロジェクトのさまざまな役割、たとえば、ビジネスプロセスと業務機能定義、アーキテクチャ設計、アプリケーション機能設計、データベース設計、UI 設計、部品設計、システム運用設計などの役割を、縦割りの組織構造（それこそ“ツリー構造”の典型）の中に割り当て、できるだけ“疎結合”であるかのように“見せかけて”責任を分担しながら作業を進めているのが古くから存在する多くのプロジェクトの組織モデルである。実際には、役割間のインターフェースがあるので、レビューや仕様調整会議といった類の場があるのだが、それは、副次的な役割（その場で生まれる結果に担当者は戦々恐々としており、担当者から見れば“存在悪”として位置付けられている）であって、技術者個人から見れば、なければ越したことはない会議体である。しかし、多次元の構造体を構築しているという意識を持つと、それは、存在悪ではなく、むしろ存在が必然の場で“早くやるべき調整”ということになってくる。

図 15-A-5-6 は、EA[4]体系のそれぞれのアーキテクチャ間に、すべて多対多の関係が存在していることを示した図である。アプリケーションサーバとアプリケーションシステムとの関係、DB サーバと DB との関係、非機能要件と機能要件や採用する製品との関係、なども例

[4] Enterprise Architecture

外なく多対多の関係にあることはすぐ理解できるであろう[5]。

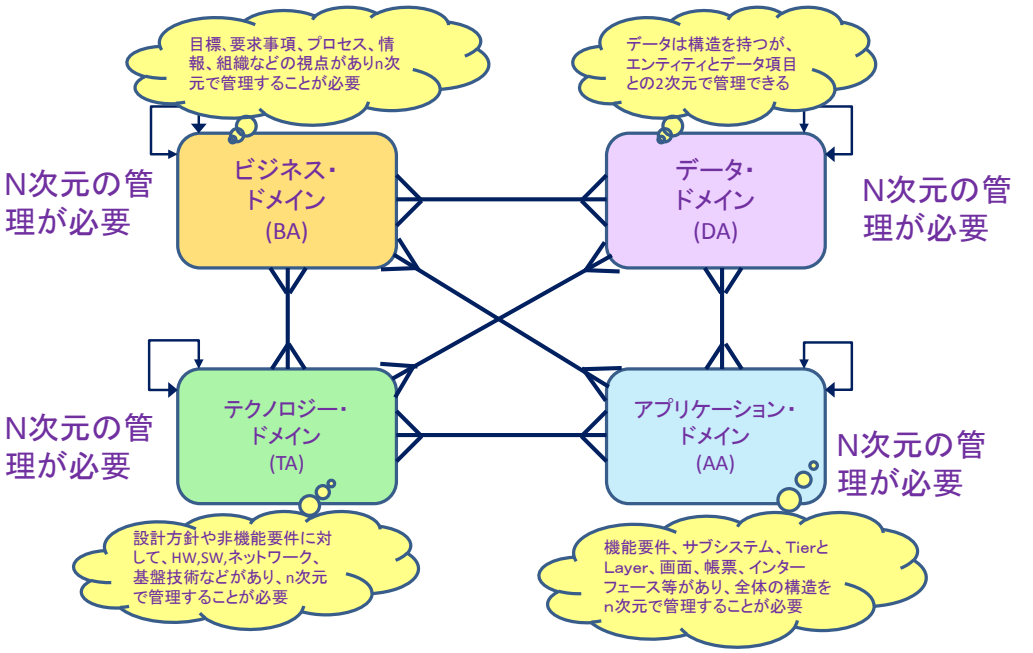

図 15-A-5-6　EA 体系におけるアーキテクチャ領域（ドメイン）間の関係

絶えず仕様の調整が必要になる多次元構造体が、たとえば 100 人以上が参加するプロジェクトの場合、果たして成功するだろうか？成功させる実践的な方法があるのだろうか？それ以上になったら？（単純な式で示せば最大のインターフェースの数は N(N-1)/2（N は人数）なので、100 人いれば 4,950 ヵ所の確認が必要になる。）現実には、そういうことにならないように階層構造の組織にし、できるだけインターフェースを少なくなるようなチーム構成にするという定石がある。とは言っても、高い品質を維持するためには多大なワークロードがかかってくる。大規模プロジェクトでは、それを十分行うことが困難であることは、世界的に見ても 60-70%のプロジェクトが失敗しているという事実が物語っている[6]。

さて、構築すべきソフトウェアがこのような構造を持っているということを明確に理解し、意識してソフトウェアを設計・開発しているプロジェクトは存在するだろうか？しかも、ビジネスモデルの要素とソフトウェアの要素とのトレーサビリティを確保しながら。ビジネスの変化に迅速に対応できる情報システムを構築し維持する仕組みとして、新たなメカニズムが必要になってくることが理解できるであろう。

---

[5] Len Bass 等が、著作[4]の中で「機能性と品質特性は直行する」と述べていることである。

[6] http://cnx.org/contents/5e9177d7-9998-43d0-9b98-91a369c6a371@7.1:7/Project_Management、http://agencyagile.com/why-agile-matters/　や http://www.ibm.com/developerworks/rational/library/dec05/grundmann/　などを参照されたい

2.4. 超高速開発ツールとは何か

現在、超高速開発ツールと言われるシステムライフサイクル全般にわたる生産性と品質向上を狙いとしたツールが多く販売されている。実は、それらツールの多くが今まで述べた多次元の構造体であるソフトウェアシステムの構成要素を“リポジトリ”と呼ばれるデータベースで管理する仕組みを持っている。リポジトリはソフトウェアシステムの構成要素間の関連を維持しているのである。そのことにより、極めて高い品質を持つソフトウェアを自動的に生成したり、あるいは、実行エンジンによりリポジトリの情報を参照しながら稼働させる機能を実現している。

ここでは､それらの個別のツールについて説明するのは主題でないので詳細は省略するが、多次元構造体という“化け物”を枠にはめて“手なずけて管理している”のが、そういったツールである、と捉えると理解しやすいであろう。そのことによって、次のことが実現できている。

ソフトウェアを短期間で開発し､“保守（維持改善)”しながら機能を追加していく

このことが実現できる主な開発・保守上のメリットは以下の通りである。

(1) 開発期間が短くなり、早期に稼働の効果を享受できる。
(2) 追加・修正案件への対応期間と工数が大幅に短縮する。
(3) ビジネスの変化への対応力が上がる。
(4) システムのライフサイクルが大幅に延びライフサイクルコストが低減する。
(5) ソフトウェア開発の品質が向上し生産性が向上する。
(6) 取り込む要件について、ユーザー企業とベンダの間で無駄な議論をしないようになる。
(7) バッグログが減る。
(8) 組織の中に継続的な変革の風土が生まれる。
(9) 業務担当と情報技術担当の会話がスムースになる。
(10) 開発とソフトウェアの維持改善担当者のモチベーションが向上する。
(11) 事業に貢献する情報システムという評価を勝ち得ることができるようになる。
(12) 業務の仕組みやルール、あるいはソフトウェアの設計情報が資産となる。

そして、これが本質的な効果であるが、このような情報システムを保有するユーザー企業は、彼らのお客様の要望に迅速に対応することができるようになり、高い顧客満足度を実現できるようになる。

2.5. ユーザー企業のオーナーシップの確立が前提になる

業務の仕組みやルール、あるいはソフトウェアの設計情報を資産として管理できる仕組みを持つということは、従来のようにソフトウェアシステムの開発と保守を外部企業に委託するやり方を大きく見直すきっかけとなる。

特に業務の仕組み（ビジネスプロセスやビジネスルール）をリポジトリで管理していく仕事を、外部企業、特に資本関係のない外部のシステム会社に任せるということは、ビジネス環境の変化に迅速に対応する上で大きな障害となるのは明らかである。製造業であれば、新製品の企画や研究開発という作業、あるいは研究テーマの管理を外部企業にまかせるようなものである。

ソフトウェアの設計が、業務の仕組みとの関係で行われる作業であることを考えると、ソフトウェアの設計と開発作業を、業務の分析と切り離して行うことは無駄が生じることになる。特に、設計情報を業務機能と関連づけて管理する場合には、業務の分析・設計とソフトウェアの設計は一連の作業になる。それら、全体の一連の作業を切り離して行うわけにはいかない。

現在、日本の企業の多くがソフトウェアの構築と保守作業を、外部のシステム開発会社に頼っている。“頼っている”というのは、作業の主体が外部の企業にあるということであり、その企業のやり方やスキル、リソースに依存するということである。依頼の主体が発注側であったとしても、発注側の意図どおりに作業がはかどらないことがあるのは致し方ないことであり、発注側が責任を持って対応できない状況となる。それが、現在の発注側と受注側の関係である。

しかし、ビジネスの変化に迅速に対応することが情報システムの使命となっている時代では、ソフトウェアシステムの構築を外部の企業に任せていては、競争力の妨げになる。それを打開するためには、ユーザー企業の強いオーナーシップの確立が必要になることが前提といってもよい。そのような考えを持った企業が一部に現れるのみならず、グローバル展開をしている大手企業にも現れ始めているのは心強い限りである。

そもそも、ビジネスの構造も多次元モデルになることはすでに述べたとおりである。現在、詳細な業務機能（ビジネスプロセスやビジネスルールなど）を記載した業務マニュアルが存在し適切に維持管理されている企業は多くはない。業務の仕組みやルールが、プログラムコードにしか残っていないということである。したがって、ビジネス機能やビジネスルールが、どのソフトウェアに記載されているかをトレースできるように管理している企業は皆無に近いのではないだろうか。このような状況を改善することが求められている。

ビジネスの変化に迅速な対応をするためには、限られたリソース（人、金、時間）の中で、優先順位を付けたうえで、どの要件に対して、どの範囲で、どれくらいの投資をして、どういった人材を投入するのかという判断が必要になる。QCD[7] をすべて達成することをユーザー企業と契約でコミットしている外部の企業にその判断ができるわけではない。変化に迅速に対応するためには、QCD と Scope を加えたプロジェクトの目標としての指標である QCDS[8] を自由にコントロールできるマネジメントがリーダーとして参加することが条件な

[7] Quality（品質）、Cost（価格）、Delivery（納期）
[8] Quality（品質）、Cost（価格）、Delivery（納期）、Scope（スコープ）

のである。それは開発のときもそうであるし、稼働後の機能追加の案件でも同じである。それができるのはユーザー企業でしかありえない。

「2.ビジネスの変化に迅速に対応できる情報システムに求められること」を整理すると、次の 2 点になる。

(1) ビジネスの変化に、多次元構造体であるソフトウェアシステムを迅速に対応させるためには、多次元構造を保持するリポジトリを持つツールの導入が必然である（大量に作成した文書を維持するのは人の管理能力を超えた作業になる）。

(2) 迅速さを求めるのであれば、多くの要件に対して QCDS の優先順位づけが必要であり、それができるのはユーザー企業であって外部の企業ではない。ユーザー企業のオーナーシップの確立が前提である。

次の「3.これからのソフトウェア構築と保守の進め方」では、ソフトウェアシステムの全体を多次元構造体ととらえた時に、ソフトウェアの構築と保守のやり方はどうあるべきか、ということを記述する。

3. これからのソフトウェア構築と保守の進め方

これまで記述してきたとおり、情報システムを構築することに価値を置く時代から、ビジネス活動を行う中で、それを運用し活用することでビジネス上の価値を高めることに重きを置く時代になっている。そういった考え方を実現するためには、システムが成長し改善し続けるものであるという認識に立ち、ビジネスとソフトウェアの部品管理を前提にした情報システムの構築・維持改善と、それを支援するソフトウェア開発の進め方が求められる。

理解を容易にするために、モノ作りの生産工程がどうなっているのかを見てみたい。非常に粗い図であるが、図 15-A-5-7 にそれを示した。通常の製造物では、製造工程と販売期間が重なり合っており、販売を通じて市場と対話しながら、製品を適宜改善しているのが実情である。

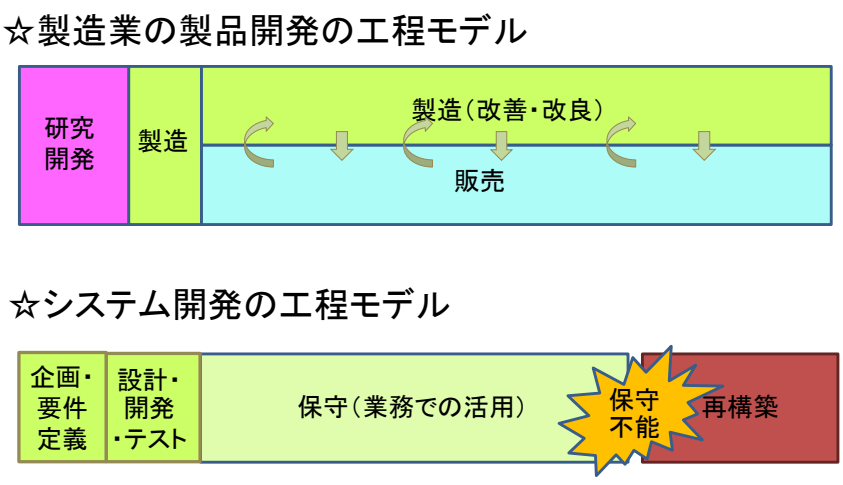

図 15-A-5-7 製造業の生産工程モデルとソフトウェアの開発工程モデル

実は、ソフトウェアのライフサイクルも製造物と似た面を持っている。保守工程が一番長く、その間は業務で使われている。しかし、使いながら利用者の声を反映し、適宜改修していくのが保守のあるべき姿であるということは、長い間あまり考えられてこなかった。ある時期になると、突然保守不能となり、廃棄に至り再構築するということを繰り返しているのが実態である。

それは、すでに述べたように、我々の頭の中に特定のライフサイクルモデルが前提のようにこびりついていて、その枠から抜け出せないからでもある。その例の一つを図 15-A-5-8 に示した。この図の意図するところは、従来のライフサイクルモデルが保守を「開発が終わったシステムを“保ち守る”工程」だと思わせていることからくる問題である。つまり、開発までに QCD に Scope も加えて 100 点をとるべきであると思い込んでいるために、ユーザー企業も、要件の変更や追加を絶え間なく提示してくるという問題である。

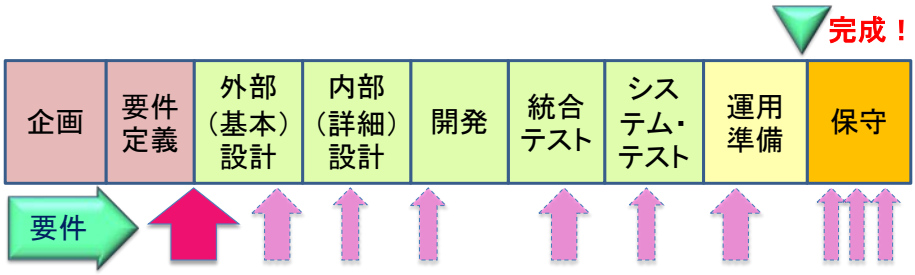

図 15-A-5-8　従来のライフサイクルモデル

ビジネスへの貢献が求められる情報システムは、自動車、電気製品、携帯端末、日用品、化粧品など一般の製品や商品が販売しながら一定サイクルでモデルチェンジや改良された新商品を発売していくのと同じように、使いながら改善していくやり方に変革していく必要がある。保守（Maintenance）ではなく改善（Improvement）が求められている。（稼働時に達成すべき最低限の品質を明らかにしておくべきことはいうまでもない）

つまり、保守工程を図 15-A-5-9 のように維持改善工程だととらえれば、話は変わってくる。定常的作業として、新たに発生する要件を取り込む仕事を通じて、システムに機能を追加させ、あるいは変更するということである。まさに DevOps[9]の考え方である。

図 15-A-5-9　保守ではなく維持改善工程と考えるライフサイクルモデル

[9] Development と Operation を組み合わせた造語

業務の仕組みやソフトウェアの設計情報を管理しているリポジトリを持った開発ツールを使えば、運用しながら改善するということが容易になる。長期にわたって運用し続けることが求められる基幹業務システムなどでは、リポジトリを持つ開発ツールを活用したシステムを保有するのが今後のあるべき姿であろう。開発のやり方も、開発ツールを前提としたやり方に変えていくことが必要になる。その、現実的なモデルは、図 15-A-5-10 のようなやり方になる。

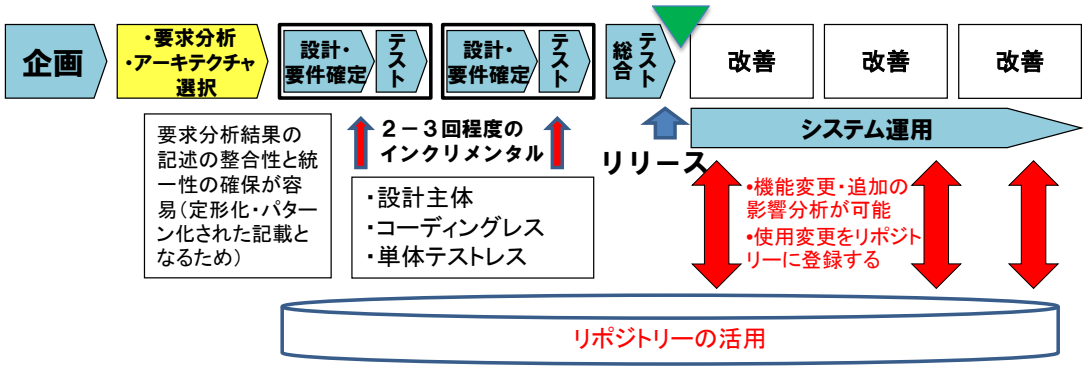

図 15-A-5-10　開発ツールを活用したソフトウェアの開発工程モデル

もう一つ重要な点がある、それは、ソフトウェアシステムが多次元の構造物であるということを考慮した仕事のやり方と管理の仕組みを導入する必要があるということである。一人が作り出す成果物が、他のメンバーの作業に影響を与えるということである。それは、進捗の把握や統合されたソフトウェアの品質評価に係る問題につながる。自分が開発している対象が多次元構造物であることを自覚すると、コミュニケーションが重要であり、チーム間のタイムラグの少ない仕事のやり方が望ましいことが理解できるであろう。

そういった観点からも、紙の成果物を主体とした労働集約的なやり方では、ある意味、30年ほど前に受注台帳や在庫台帳を紙で管理していた時代と同様の問題（情報の不一致）を引き起こすことが、容易に理解できるであろう。リポジトリを持つということは、ソフトウェア開発という業務に必要な情報の統合管理を推進することであり、開発ツールはシステム開発という業務の情報共有のツールである、という理解が必要になってくる。

こういった開発のやり方をする上での具体的な作業上のポイントは以下の点となる。

(1)　少人数で、小さな規模のシステムを少しずつ完成させながら開発を進める（大人数が同時に同じ情報を更新しようとするとパフォーマンス上の問題を引き起こすのは、業務システムの場合と同じである）。

(2)　業務モデルの構築（要求分析）の中で、プロトタイプを作りながら、完成後のイメージを利用者に確認しながら開発を進める。

(3)　アプリケーションの開発に関しては、最初から最後まで同一技術者が参加する。

(4)　工程を分断することはしない。

(5)　QCDS の優先順位を定期的に見直すための意思決定組織を常設する。
(6)　多次元構造体の小さな単位で、関係者がインターフェースの確認を行いながら仕事を進める（機能定義者、データ管理者、部品開発担当など)。
(7)　データ項目とデータ構造の定義を優先する（データ項目の登録が優先されるのは、受注登録の前に顧客情報や商品情報などのマスター情報が登録されていなければならないのと同じである)。
(8)　業務とシステム全体の鳥瞰図は企画段階で作成する（目的、狙い、事業におけるシステム化範囲の決定、ビジネスプロセス関連図の作成、上位の業務フローモデルなど)。
(9)　開発ツールのインプットとなる成果物の作成は従来のやり方を参考にできる [10]。ただし、8 割程度完成したと判断したらツールに投入し、その後の追加・修正はツール上で行う。

4. 事例と効果

ここでは、超高速開発ツールを使って「3.これからのソフトウェア構築と保守の進め方」で記述したようなやり方を採用して構築した情報システムの事例を紹介することで、その効果を記述 [11]する。

4.1. 開発生産性

表 15-A-5-2 と表 15-A-5-3 が開発局面における生産性の実績である。比較のために、表 15-A-5-2 では、労働集約的なやり方の開発提案との比較、表 15-A-5-3 では、既存システムの記述したコードの量との比較を記載している。表 15-A-5-4 は、メトリックス調査 2014[5] に記載されたものであり（xRAD [12] が超高速開発ツールのことである)、それぞれのやり方を採用した場合の開発費用、工数、工期の平均値を比較したものである。それによると、超高速開発ツールを採用した場合、３つの指標のいずれもが、ウォーターフォール（WF）型開発と比べて、およそ 1/3 程度になることを示している。

[10] 正確にいえば、開発ツールが必要な成果物を適切に生み出せるやり方を採用しなければならない。開発ツールを活用する上で、その点も重要なテーマであるが、本論からそれるので詳細には触れない。
[11] ここに記載する内容は、筆者が開発ツールのベンダやユーザー企業から集めた情報をもとにしたものであるが、筆者自身が開発に関与した事例ではないことをお断りしておく。
[12] xRAD：eXtreme Rapid Application Development

表 15-A-5-2　開発時の効果 1

開発	アプリケーション	労働集約的方法による提案	超高速開発ツール採用実績	削減率
ツールA（ソース生成型）	A社 生産管理システム	100人月	35人月	65%削減
	B社 基幹業務	300人月	167人月	45%削減
	C社 輸出入業務	185人月	45人月	75%削減
ツールB（実行エンジン型）	D社 法定帳票作成システム	160人月	60人月	62%削減
	E社 仕訳検証システム	50人月	10人月	80%削減
	F社 基幹システム	400人月	200人月	50%削減
ツールC（実行エンジン型）	G社 基幹システム	6,000万	800万	86%削減

※開発ツールの費用と社内コストは含まれていない。サンプル数 40 程度 [13]。

表 15-A-5-3　開発時の効果 2

開発	既存言語	現行規模	超高速開発ツール採用実績	削減率
ツールE（実行エンジン型）	COBOL	1,300k steps	34k steps	97%削減
	PL/I	3,700k steps	78k steps	98%削減
	RPG	100k steps	2.7k steps	97%削減
	VB(C/S)	130k steps	8.5k steps	93%削減

表 15-A-5-4　3 種類の開発方式の生産性比較 [14]

		WF	アジャイル	xRAD	アジャイル／WF	xRAD/WF
総費用/JFS	平均	112.19	135.45	40.7	1.21	0.36
	係数	28.2	57.65	6.4		
工数/JFS	平均	1.28	2.15	0.48	1.68	0.37
	係数	0.44	1.6	0.26		
工期/JFS	平均	0.31	0.24	0.1	0.77	0.32
	係数	0.13	0.04	0.03		

※　WF：ウォーターフォール、xRAD：超高速開発、JFS：Juas Function Scale

JFS=画面数＋2／3×帳票数　で求められる。係数はグラフにプロットしたときの傾きを示す。

xRAD は係数が小さいので、規模による生産性の変動が少ないことを示している。

[13] サンプル数が 40 程度であり、より正確な分析を行うことが必要である。

[14] JUAS メトリックス調査 2014 図表 6-258 3 種開発法の比較（参考値）をもとに作成。

こういった生産性の高さが示される理由として、以下のようなことが考えられる。

（1）　投入人数が従来の場合と比べて少ないため、コミュニケーションのロスが少ない。

（2）　紙で作成するドキュメントでは、チーム間の作業の進捗や打ち合わせなどにタイムラグが発生することで不整合が発生しやすいが、リポジトリに設計情報が登録されると設計情報の共有がなされるので、品質の作り込みが可能で手戻りが少なくなる。

（3）　プロトタイプを作成して要件を確認する方式を採用するので、業務要件の確認と確定がしやすく、比較的短期間で要件をまとめやすい。

ソフトウェア開発の生産性は、開発規模が大きくなるにつれて低下することが知られている。その理由は、投入人数が増えるとコミュニケーションパスが指数関数的に増加し、欠陥数もそれに応じて増加するので手戻り作業が増えるのが原因である。その結果、特定の成果物を作成するのに必要な時間が多くかかり、生産性が悪化するということになる。

例をあげると、ある程度の規模のシステムになれば、開発チームの数は必然的に多くなる。販売管理アプリケーションを考えれば、アプリケーション開発だけでも、「受注アプリチーム」、「発注アプリチーム」、「在庫管理アプリチーム」、「入出庫アプリチーム」、「債務管理アプリチーム」、「債権アプリチーム」、「マスター管理アプリチーム」、「部品開発チーム」などが作られるであろう。それ以外に、「データ管理チーム」、「アーキテクチャ設計チーム」、「UI プロト開発チーム」、「基盤設計チーム」、「インフラ検討チーム」、「運用検討チーム」、「移行検討チーム」など数多くのチームに分割して作業が進められる。

たとえば、「UI プロト開発チーム」の検討の中で、「A という画面には x という、新たなデータ項目を表示したい」という話が出たとしよう。当然、その x というデータ項目は、「データ管理チーム」やいずれかのアプリチームにも伝わらなければならない。しかし、そこにはタイムラグが発生する。きちんと伝われば幸いだが、伝わらなければレビューで指摘されるか、何も指摘されずに、テストでバグが発見されるということになる。当然、大量の人数で開発を行うと、こういった手戻り工数は全体で見れば膨大なものになる。部品開発チームとアプリケーション開発チームとの関係を考えても、同じ課題が存在することが認識できる。

このような現象が頻繁に発生しているのが、現状のシステム開発の実態である。少人数で開発できれば、生産性が高くなるのは当然の理屈である。結局のところ、特定の開発チームの設計に係る様々な情報を、他のチームにいかにスピーディに伝えることができるのか、ということが、品質と生産性に大きな影響を与えるということである。

そのような視点で見ると、リポジトリを持つ超高速開発ツールを活用すると、なぜ高い生産性が得られるのかが理解できるであろう。リアルタイムに情報が伝わることが、その理由である。最新情報をもとに作業ができるということである。

このように、超高速開発ツールを活用すると作業の質が向上することで、レビューでの欠陥の指摘が少なくなり、その結果手戻りが大幅に減少することが、工数減少と期間短縮、ひいてはコストの削減につながる。

4.2. 保守作業の生産性

表 15-A-5-5 と表 15-A-5-6 は、保守局面における生産性の比較である。表 15-A-5-5 は、SI ベンダへの支払いコストの相違の比較であり、表 15-A-5-6 は導入企業の方の評価である。

表 15-A-5-5　保守時の効果 1

保守	アプリケーション	導入前	導入後	削減率
ツールA （ソース生成型）	J社	年間外部支払料金 6,000万～8,000万	年間外部支払料金 100万	98%
	K社	4名 （10システム）	4名 （15システム以上）	同じ要員で 追加システム開発
ツールB （実行エンジン型）	D社 法定帳票作成システム	1人/100人 月	1人/50人月 （変更対応200件／年間）	50%削減
	F社 基幹システム	3,000万円 （ERP 年間ランニングコスト）	1,000万円 （年間保守料ライセンス込）	66%削減
ツールC （実行エンジン型）	L社 基幹システム	年間保守料金 2,520万円	年間保守料金 336万円	85%削減

表 15-A-5-6　保守時の効果 2

保守	企業	削減率
ツールD （EAIツール）	O社	更新負荷66%削減
	P社	運用負荷70%削減
	Q社	ECサイトへの出店を 30日から2日へ

こういった保守作業の生産性の高さが示される理由として、以下のようなことが考えられる。

(1)　リポジトリを使って影響分析ができるので、安心して変更対応ができる。
(2)　既存の仕様を変更することに、抵抗感なく取り組める。
(3)　テスト作業が大幅に軽減される。
(4)　コード・レベルでの品質検証が不要になる。
(5)　作業量が大幅に削減される（1 箇所の変更対応で済む）。

リポジトリに設計情報が保持されていると、変更の影響を確実に把握できる。それが、最大の強みであり、その価値を十分生かした作業ができることが、こういった非常に高い生産性につながっている。

4.3. 定性効果とビジネス上の効果

システム開発・保守における定性効果と事業や業務上の効果という面では、表 15-A-5-7 に示されるような効果が生まれていることが、導入企業の方の声として挙げられている[4][15]。

表 15-A-5-7　超高速開発ツールの導入効果

	システム開発・保守上の効果（例）	事業・業務上の効果(例)
通販向けコールセンター業務	素早く安価にシステムを開発できるようになった	他社との差別化が達成
	新たな要件への対応が容易になった	小規模顧客のニーズの取り込みを実現
	自社開発が可能になった	事務作業の効率化目標を達成
観劇事業社内業務と顧客管理	ユーザー部門からの情報要求に適切な対応が迅速にできるようになった	人員の最適配置の実現
	自社開発が可能になった	個人売上の向上
		業務効率向上
板金加工事業	短期開発を実現	ITカイゼン（業務のムリ、ムダ、ムラの減少）の達成
	自社開発が可能になった	顧客の要望やクレームへの責任ある対応の実現
	保守・運用コストの大幅削減	構成部品情報管理、修理品価格管理などが簡単に実現
		情報の履歴管理を実現し、顧客フレーム管理を高度化
農協における基幹業務システム	電算経費64%削減	業務の要望の早期実現
	システム開発時間の短縮(3.2倍)	システムの戦略的活用が実現
	自社要員の活用と内製化の達成	雇用の保証の実現
	システム保守経費の削減	
	若い人のモチベーションアップ	
	若い技術者への業務知識含めた技術移転を達成	
	少人数開発の実現	
インフォテイメント機器・自動車用電子機器の基幹業務システム	PM,SE,プログラマを兼務で開発ができるようになり、情報伝達ロスが削減	業務スピードの向上
	システム担当者が業務要件に精通するようになった	ユーザーニーズを高いレベルで実現
	驚異的な開発効率の達成	
	仕様変更、機能追加がスピーディに行えるようになった	
	システムライフサイクルが長い	
	JSOXなど内部監査対応が容易	
	内製化の実現	
医療機器の開発・製造・販売事業の与信管理業務	要件を固める中でプロトタイプが完成	業務効率化の達成
	業務部門主導開発の実現	資料検索作業の大幅削減
	システム開発のハードルが飛躍的に低下	本来の与信管理業務の実現
	マネジメントの参加意欲の増大	遠隔地との情報共有の実現
		ユーザー部門の意識改革の実現
		業務改革意欲が増強

これらの事例に示されていることは、超高速開発ツールを活用すると、システム開発や保守の生産性が著しく向上するというシステム作りの価値だけでなく、まさに多くの企業が求めている「ビジネス価値の向上」にも情報システムが寄与できるという事実である。「他社と

[15] https://www.x-rad.jp/usecase/　にも、超高速開発ツールを導入された企業が、ビジネスの変化やその顧客からの要望に迅速に対応できるようになったことが記載されているので参照されたい（2015.9.30 時点）

の差別化」、「個人売上の向上」、「顧客の要望やクレームへの責任ある対応の実現」、「ユーザーニーズを高いレベルで実現」、「遠隔地との情報共有の実現」などという導入企業の方々の発言は、意味深いものがある。

一見すると、超高速開発ツールとそういったビジネス価値がどのように結びつくのか分かりにくいと思われるかもしれない。しかし、超高速開発ツールのリポジトリが業務の仕組みを管理していること、そして、その仕組みが可視化されシステム担当者と業務担当者が共有できる状態となっていることを理解すれば、それらが原因と結果の関係を生むことは容易に納得できるであろう。

さらに、「若い人のモチベーションアップ」、「システム担当者が業務要件に精通するようになった」といった事実は、ある意味で企業風土の改善にも大いに寄与したことを物語っており、その意義は個々のビジネスへの貢献ということ以上に価値あることかもしれない。

なお、これらの事例にみられる成功要因がすべて超高速開発ツールにあるというつもりはない。それぞれの企業の文化・風土、マネジメントの方々も含めて関わった方々の意欲や意識が成功の重要要因であったに違いない。そして、システム開発プロジェクトでよく問題となる要件定義という作業が滞りなく行えたということは、プロジェクトの意思決定に大きな問題が生じなかったことの現れであり、プロジェクトに関わったメンバーのチームワークと目的意識の明確さをうかがわせるものである。そういったことを前提にしたとしても、超高速開発ツールの活用は十分に評価に値するといえよう。

4.4. 超高速開発ツール活用の課題

ここまで、超高速開発ツールを活用することのメリットを書いてきた。しかし、活用する上での課題がないわけではない。いくつか指摘しておきたい。

(1) リポジトリは一種のデータベースであるので、同時に大量の更新アクセスは性能劣化を招きやすい。リポジトリを実装している DBMS と性能要件を確認して必要に応じて分割運用を検討する必要がある。

(2) リポジトリを分割する場合には、リポジトリ間のデータ連携の仕組みや統合のタイミングを決めておく必要がある。

(3) リポジトリのバージョン管理も重要な要素になる。リポジトリは整合性を保証できるが、不整合を許す仕掛けを持っている（異音同義語のチェックが自動的にできるわけではないので）。したがって、定期的に整合性を確認する作業を事前に計画しておき、そのタイミングでバージョンを確定するなどのやり方を採用することが望ましい。

また、今までのやり方とは違うやり方になるので、それに合わせて体制上にも新しい役割が必要になり、作業の進め方にも工夫が必要になる。たとえば、以下のようなことを考慮する必要がでてくる。

(1) リポジトリの運用管理責任者という役割の設置

この役割は、業務システムにおけるデータベースアドミニストレーター、あるいは、データマネジネント責任者と同じである。リポジトリの情報の正確性向上と整合性の維持管理に努めるのが役割である。具体的には、不整合の発見と開発チームへの伝達、リポジトリのバージョン管理、バックアップ・リカバリーなどの作業に関する責任を持つ。

(2)　定期的な整合性確認作業の実施

リポジトリがあるので、必ず自動的に整合性がとれるかというと、そういうわけにはいかない。最近、マスターデータマネジメントの重要性が喧伝されているが、ソフトウェア開発の現場でも同じことが発生する。たとえば、異音同義語の問題がある。リポジトリに登録されたデータの品質が常に高いとは言い切れない。開発の初期段階では、作業の進行を早めることを優先して、多少の不整合を許す進め方をするほうがよい。初期段階では、検討中の作業が多いので、変更・削除は頻繁に発生する。そのような段階で整合性を優先すると作業がはかどらないからである。しかし定期的に不整合箇所を確認し、確定した内容（データ項目名、コード値、機能名など）に修正をかける必要がある。設計工程の終盤にかけては、その頻度を上げて品質優先で不整合をなくしていくことが必要である。このように、開発プロセスの大きなアクティビティとして、「整合性確認」ともいうべき作業を割り当てておくことが望ましい。

4.5. 事例と効果に関するまとめ

今まで記述してきたことを全体的にまとめると、本事例における効果は以下のようになる。

(1)　開発期間・工数・コストの削減
(2)　保守作業の期間短縮・TCO[16]の削減（外部へ支払う金額）の削減
(3)　ビジネス環境の変化への対応力の強化と経営スピードの向上
(4)　業務担当者とシステム担当者のとの会話がスムースにできるようになる
(5)　システムライフサイクルの長期化とライフサイクルコストの削減
(6)　業務の効率化、事業や業務の変化への迅速な対応の実現、ユーザー企業の顧客のニーズの迅速な取込の実現
(7)　ユーザー企業のお客様の要望への迅速な対応の実現
(8)　組織風土の改善

そのほか、超高速開発ツール、特に実行エンジンを持つタイプは、OS やミドルウェアの違いをツールが吸収するので、アプリケーションが全く影響を受けない、あるいは最少の修正で対応可能などという効果がある。

この数年、機能の追加や変更に容易に対応できる超高速開発ツールが、本格的な基幹業務システムに活用できるレベルにまで成長している。ここに示したのは、そういったツールを活用して開発・保守を行うことで、ビジネスやシステムの環境変化に対する対応力を強化さ

[16] Total Cost of Ownership

せ、市場での競争に負けない体質を作りあげてきている企業の事例である。そういった企業とそうでない企業の競争力は圧倒的な違いが生まれることに、企業経営者や行政組織の方々に気が付いていただきたいと願うものである。

5. まとめ

今まで記述してきこと整理すると次のようになる。ソフトウェアシステムの全体が多次元構造でその全体像を事前に人がイメージできない対象であるという認識を新たにし、イノベーティブなシステム開発のやり方の登場が望まれていることを認識していただければ幸いである。

(1) 目に見えないソフトウェアシステムは多次元の構造物であり、労働集約的な開発で手におえる相手ではない。それが無理であることは実績が語っている。

(2) ビジネスへの貢献が求められる情報システムは、短期間で開発してビジネスの果実を早期に収穫することが大事である 。

(3) ビジネスプロセス、ビジネスルール、業務機能、組織、調達チャネル、販売チャネル、物流、拠点などを含むビジネス活動の要素の変化がソフトウェアにどのように影響を与えるのかを即座に分析できるためには、ビジネスモデルの要素とソフトウェアの要素との関連を維持することが前提となる。

(4) そのためには、リポジトリを持つ開発ツールを活用して、ソフトェア構造物を管理することが条件となる。

(5) リポジトリに登録された業務システムの設計情報を資産として永続的に活用していくことを考えるべきである。

(6) ビジネスの変化を知るのはユーザー企業自身であり、ユーザー企業自らがリポジトリを維持し開発ツールを活用していくことが求められる。

(7) ソフトウェアの maintenance は単なる保守ではなく継続的改善であるという意識の醸成が求められる。

参考文献

[1] 公益財団法人全国中小企業取引振興協会 : 平成 25 年度中小企業の情報利活用に係る実態調査（概要版）P24 、2014

[2] 独立行政法人情報処理推進機（IPA）:「第 7 回地方自治体における情報システム基盤の現状と方向性の調査」及び「情報システム調達のための技術参照モデル（TRM）に関するアンケート調査」、2014、P48

[3] 関 隆明（監修）、一般社団法人 ICT 経営パートナーズ協会、大島正善他（著）: 超高速開発が企業システムに革命を起こす、日経 BP 社、2014

[4] Len Bass 他（著）、前田卓雄他（訳）: 実践ソフトウェアアーキテクチャ、日刊工業新聞社、2005、p95

[5] 一般社団法人 日本情報システム・ユーザー協会（JUAS）: ユーザー企業 ソフトウェアメトリックス調査 2014、2014

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（IPA/SEC）