ソフトウェアデザインプロセスの呪縛からの解放

－人働説から知働説への転回－

大槻　繁[*1]

Liberation from the Spell of the Software Design Process
- A Turn from Man-Month Model to Knowledge Centric Model -

Shigeru OTSUKI

Type of not only waterfalling, but also agile development process, are bound to curse fix the requirements specification, then, that to do the implementation. In principle, these are so-called V-shaped Process in accordance with the process of software development. Software is executable knowledge be placed in the real world, and also must be considered to be the target of the design artifact. Use of software is a process of action and reaction by its execution. Through this, recognition and requirements are changing. In this paper, we propose a process of software construction and evolution process model called ΛV-shaped model considering the user feedback process. This model is based on the language game regarded as usage and development be the same, then we handle the process of evolution and adaptation with a focus on aspects of software.

キーワード：ソフトウェア開発プロセス，実行可能知識，ΛVモデル，言語ゲーム

Key words: Software Development Process, Executable Knowledge, ΛV-shaped model, Language Game

1. はじめに

ソフトウェアは、人間の創造的活動によって生まれる人工物（artifact）であると同時に、人間の日常生活や組織活動にとって不可欠であり、産業領域として大きな位置を占めている。系統的にソフトウェアを開発・保守する知識体系はソフトウェアエンジニアリングと呼ばれており、数多くの方法論、開発プロセス、ツールなどが提唱されている。

一方、建築物や工芸品などのモノ作りの世界では、技術・芸術を融合させ、人間の創造性に中心を置いたさまざまなデザイン活動が行われている。モノ作りのデザインの世界とソフトウェアエンジニアリングの世界とは、今までそれぞれ独自の取り組みをしてきたが、さまざまな領域にソフトウェアが浸透し、関わるようになってきたため、両者を統一的に扱うことができる手法が求められている。

本論文では、第２章でデザイン対象であるソフトウェアの、ソフトウェアであるがゆえの特性が何であるかを説明し、開発プロセスや組織上の役割分担の観点からの課題を示す。第３章で従来の工業的なソフトウェア開発パラダイムという呪縛から脱し、多様で創造的な活動を支える方法を確立していくためのアプローチを述べる。特に、今世紀に入り台頭してきているアジャイルプロセスの活動を考察し、中心課題が変貌してきている様相を示すとともに、ΛVモデルと言うデザイン論とソフトウェアエンジニアリングとを統一的に扱うプロセスを解説する。第４章で総括として、アジャイルプロセスに代表されるソフトウェアデザインプロセスが次に目指すべき事項を簡潔なマニフェストとして提示する。

*1　株式会社一（いち）　副社長, 日本デザイン学会（正会員），
ビジネス・ブレークスルー大学 経営学部　教授
Ichi Corporation, Vice President, and
Business Breakthrough University, Professor

2. ソフトウェアエンジニアリングの呪縛

2.1 ソフトウェアの本質的困難

ソフトウェアがソフトウェアであるがゆえに難しい性質というのは、「本質的困難(essential difficulty)」として知られている。ブルックス(Frederic Brooks, Jr.)の『人月の神話』[1]の中で掲げられているものは以下の４つである。

複雑性(complexity) : ソフトウェアは大きくて複雑なことそれ自身が問題なのだということである。レゴブロックを集めて組み合わせるようなわけにはいかない。他のどのような構造物よりも複雑である。ソフトウェアの構成要素間の依存関係も規模が大きくなるほど非線形に増大する。

同調性(conformity) : ソフトウェアはソフトウェア単独で存在しているわけではなく、ハードウェアやネットワーク、他のシステム、人間の行動や習慣にいたるさまざまなものと関係を持ち続けるものである。ソフトウェアの宿命は、こういった外部に常に同調（順応）し続けなくてはならないところにある。

可変性(changeability) : ソフトウェアは常に変化し続けなくてはならないということである。たとえ、当初の計画通りシステムが出来上がったとしても、それを使っているユーザはさらなる要求を思いつくし、そもそも出来上がったシステムが世界を変えてしまうため、人間の認識にも影響を及ぼし、新たな要求が出てくることになる。最近のビジネス環境の変化は激しく、技術進歩も速いため、ソフトウェアもこれに常に対応していくことが要求されている。

不可視性(invisibility) : ソフトウェアが複雑な概念の集積であることに起因して、それが目に見えないということである。開発プロセスや意思決定の経緯も見ることができず、単純な図面で全体が理解できるということもない。

上記４つに加え、ソフトウェアが一般の人工物と異なる性質として、「**実行可能性**（executable）」が挙げられる。ソフト

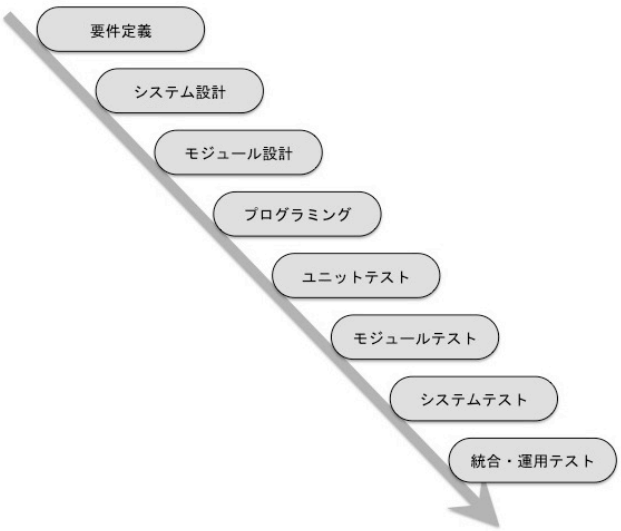

Figure 1 ウォータフォール型開発プロセス

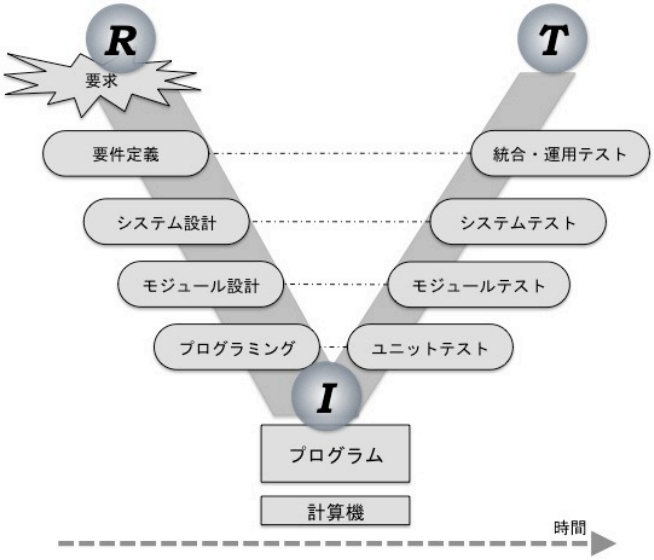

Figure 3 V字モデル

ウェアは最終的にプログラムコードによって実現される。抽象機械、計算、変換といったオートマタや実行のメカニズムが必要である。

2.2 開発プロセスの限界

1970 年代に確立され、現在でも多くのプロジェクトで採用されている開発プロセスは、ウォータフォール型を基本としたものである（Figure 1）。仕様化（要求分析・定義）を行い、そのレビューをした後、これを確定させ、以降、順次工程を進めていくプロセスである。一般に、中間成果物の固定化は、契約や会合等によって権威付けられた形で行われ、前工程での誤りが、後工程に持ち込まれると手戻り（フィードバック）を生じ、全体の工数も増加してしまうことになる。

開発活動中の実装段階に着手してからの要求変動に対処することは難しいため、初段の分析や基本設計フェーズを着実に進める方法として、バリー・ベーム（Barry Boehm）がスパイラルモデル[2]を提唱した（Figure 2）。これは、初段の業務分析、要件分析、概要設計、基本設計について、計画、意思決定、

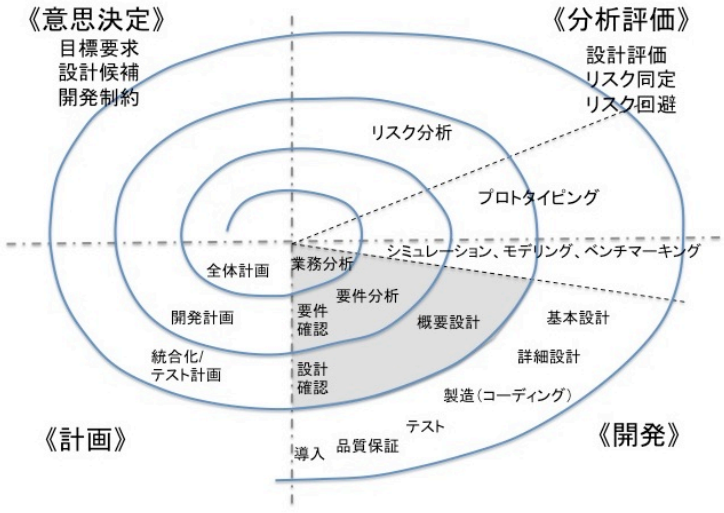

Figure 2 スパイラルモデル

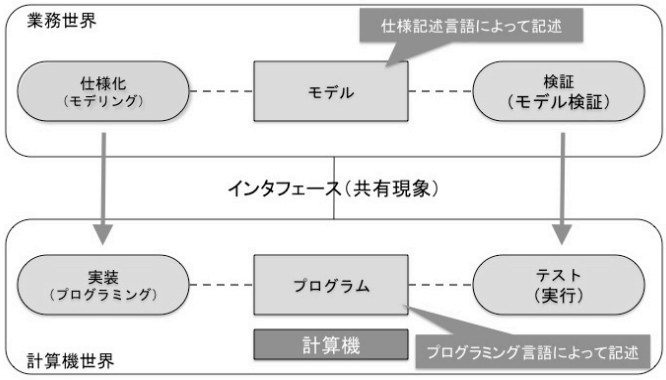

Figure 4 利用と開発の世界

分析評価を繰り返しながらプロトタイピングを交え、段階的に拡充して仕様確定をしていくという考え方である。

ソフトウェア開発プロセスの構成要素である要求定義、設計、プログラミング、テストなどの関係を原理的に分かりやすく説明するモデルとして、「V字モデル」[2]がある。これは Figure 3 に示すように、V字の左側に位置する意思決定の結果、プログラムが作成され、そのプログラムの実行によってV字の右側に位置するテストによってプログラムの正しさを検証していくことを表している。V字モデルによって、ウォータフォール型開発プロセスの原理的な欠陥、最初の意思決定の正しさが最後にならないと確認できないといった事項を明確にすることができる。

ソフトウェア開発の難しさは、開発技術そのものが技術革新によって高度化、複雑化していくことだけでなく、Figure 4 に示すように、利用と開発という全く独立した世界の両者にまたがるコミュニケーションを必要としているところにある。上部が利用の世界であり、下部が開発の世界である。

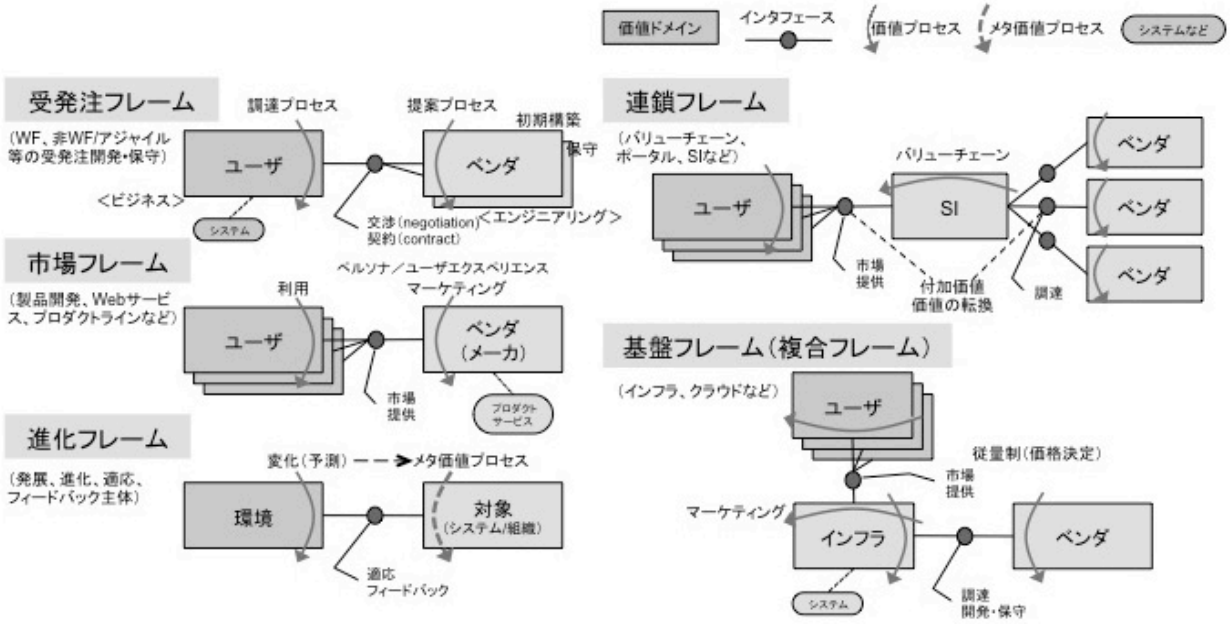

Figure 5 ビジネススキームの種類

2.3 役割分担の限界

ソフトウェアに関わる代表的なビジネススキームを分類すると、Figure 5 のようになる。「受発注フレーム」は、ユーザ企業がベンダ企業に対してシステム開発の調達を行い、これをベンダ側がシステムの初期構築や保守・運用を請け負うもので、従来からの典型的なユーザ/ベンダの取引関係である。「市場フレーム」は、エンドユーザに対して直接サービスや組み込み製品などを提供するものである。「進化フレーム」は、継続的なサービスを提供する場合に、ユーザや市場環境からのフィードバックを得てシステム側が進化・適応していく、これからの時代に要求される形態である。「連鎖フレーム」は、日本独特の SI 企業の商習慣を表したもので、ユーザとベンダとの間に元締めの役割を果たす企業が介在する形態である。「基盤フレーム」は、プラットフォームやクラウドサービスのようなインフラ基盤を提供する形態を表している。これ等を複合的に組み合わせたビジネススキームを考えることもできる。

これ等のビジネススキームのフレームを使うと、昨今の競争の激しいビジネス環境変化へ対応するための取り組みを説明することができる。例えば、多くのユーザ企業は、Figure 6 に示すように「内製化」を図って、ユーザ/ベンダのコミュニケーションギャップを埋めるような対策を採る傾向がある。ユーザ企業が自らのビジネスを遂行するために、利用しているシステムの変化要求に俊敏に対応する必要がある。その際に、従来の「受発注フレーム」に示された調達プロセスに従い、ベンダへの発注を行うと時間的に間に合わないために、エンジニアを雇い社内で直接開発を行う。ユーザ/ベンダのコミュニケーションギャップを埋める利点と、人を抱えるリスクを負う欠点とのトレードオフの問題としてとらえることができる。

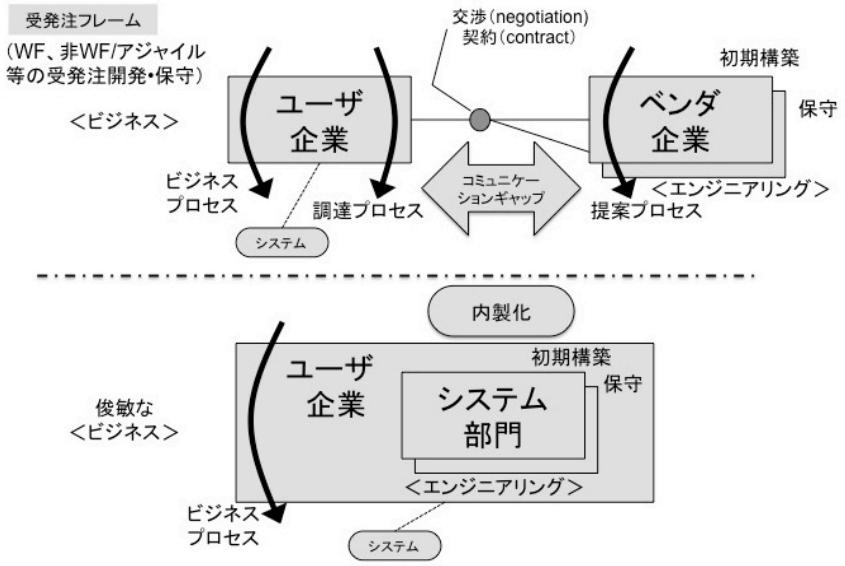

Figure 6 内製化の概念

しかし、こういった対策もソフトウェアエンジニアリングの本質を忘れた素朴な人員投入を行うだけでは、いずれ 2.1 節で述べたソフトウェアの本質的困難である「複雑性」の問題に向きあうことになるであろう。

3. 呪縛からの解放へ向けて

3.1 アジャイルプロセス

ソフトウェア開発の新しいアプローチとして、今世紀に入り、アジャイルプロセスが台頭してきている。2001 年にソフトウェア開発に関わる専門家が集まり「アジャイルソフトウェア開発マニフェスト」[3]が宣言され、これを分かりやすく図案化すると Figure 7 のようになる。以降、Figure 8 に示すような多くのアジャイル開発方法が提唱されてきた。

Figure 7 アジャイルソフトウェア開発マニフェスト

名称	提唱者	説明
エクストリーム プログラミング (XP)	Kent Beck	プログラミングを中心としたプラクティスを提示
スクラム (Scrum)	Ken Schwaber, Jeff Sutherland	マネジメントにフォーカスした方法論
フィーチャ駆動型開発 (FDD)	Jeff De Luca, Peter Coad	古典的な繰り返し型開発プロセスで、かつ、軽量
クリスタル (Crystal)	Alistair Cockburn	マネジメントにフォーカスした弱い方法論、ワイドスペクトラムな方法(小規模～大規模)
適応的ソフトウェア開発 (ASD)	Jim Highsmith	カオス適用理論(CAS)を用いたフレームワーク
リーン ソフトウェア開発 (LSD)	Mary Poppendiek	トヨタのカンバン方式(最小在庫＝ドキュメント)の原理応用
エクストリーム モデリング (XM)	OMG-MDA等	検証実行可能なモデリング(ツール)を利用

Figure 8 アジャイル開発方法の種類

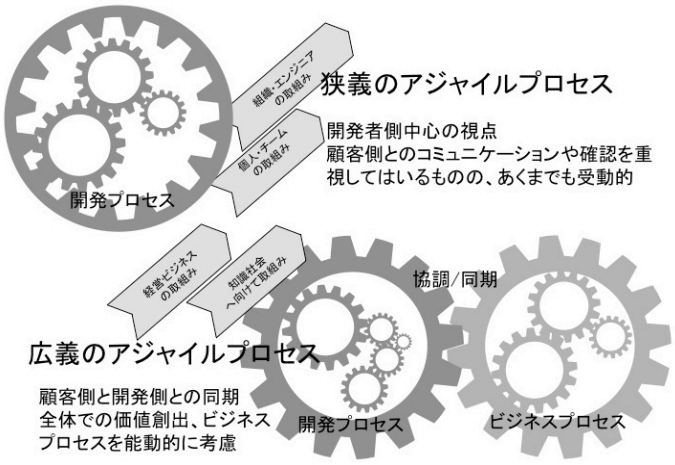

Figure 9 狭義から広義のアジャイルプロセスへ

アジャイルプロセスに共通する本質的な姿勢は以下の通りである。

ソフトウェアに対する要求（仕様）が明確になった時点で、明確になったもののみを、迅速に実現する

すなわち、全ての仕様を初段で固める必要はなく、段階的に開発を進めていく繰り返し（イテレーション）型のプロセスである。アジャイルソフトウェア開発マニフェストは、伝統的な開発プロセスが、不要なドキュメントに埋もれ、計画に工数を掛け過ぎ、契約や交渉のオーバーヘッドを生じ、本来のソフトウェア開発の姿から逸脱して硬直化してきたことへの反省・反動と見なすこともできる。

3.2 人働説から知働説へ

ソフトウェア開発における商取引の多くは、どれだけの人がどれくらいの時間を掛けるかという「人月」をベースに行われることが多く、ソフトウェアのもたらす「価値」は中心では無かった。こういった状況を改革していくための活動の一つとして、アジャイルプロセス協議会のワーキンググループとして 2009 年に「知働化研究会」が設立された。

アジャイルプロセスの分野のこの 10 年ほどの間では、Figure 9 に示すように、初期の頃は開発者中心のチームマネジメントやファシリテーションが主体の「狭義のアジャイルプロセス」であったものから、近年は、ビジネスプロセスとの連携が価値主導の見方を扱う「広義のアジャイルプロセス」に移行してきている。このような動きを受けて、知働化研究会 [4]では、ソフトウェアそのものの定義を見直し、利用の世界と同化して、価値や様相を扱う「実行可能知識（executable knowledge）」について検討している（Figure 10）。

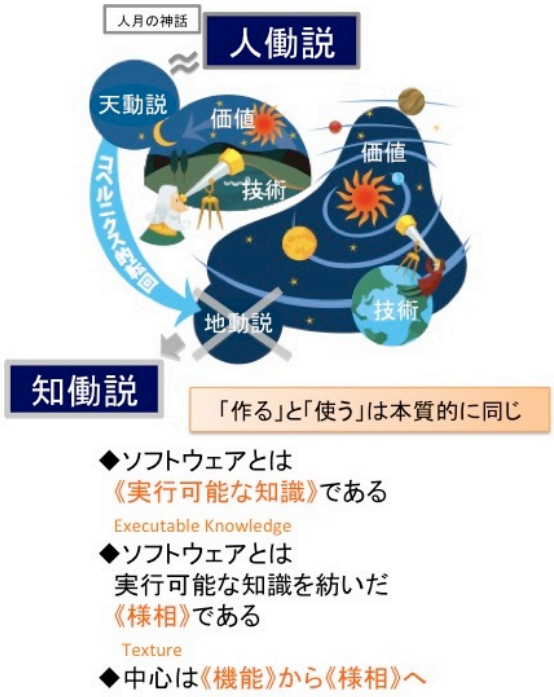

Figure 10 人働説から知働説へ

3.3　新ソフトウェア宣言

前述の「知働化」の流れの延長戦上で、新しい時代のソフトウェアはどういった考え方に基づかなくてはならないかを、2010 年 6 月に開催されたソフトウェア・シンポジウムの「ソフトウェアエンジニアリングの呪縛 WG」に各方面の専門家が集まり討議し、以下の 7 項目からなる『新ソフトウェア宣言』[5]としてまとめた。

〔1〕ソフトウェアは、数学的理論探求の上に成り立つ

プログラムの動作原理は、計算理論という数学的理論によって支えられている。ソフトウェアを作るということは、抽象機械を構築することを意味している。人間が自由に発想し、新しい概念を生み出し、定式化し、計算を実行する機械として実現する。

〔2〕ソフトウェアは、部分に還元することが不可能な全体

大きなソフトウェア、大きな問題に対処する有効な方法が「分割統治（divide and conquer）」である。モジュール化、段階的詳細化といった方法があり、これらが有効な場面も多いのは確かであるが、少なくとも限界を知る必要がある。企業やチーム、さらには、ソフトウェアそのものを生命体のような複雑系と見なすことは自然な発想である。

〔3〕ソフトウェアは、実行可能な知識である

社会の中で変化していくソフトウェアやそれへの関わり方を、探求していく必要がある。我々は、知識主導社会に生きており、「知識」はその中心である。ソフトウェアに関わる活動は、知識活動である。それは、作り方/使い方の知識であり、それを糸や布のように紡いでいくことであり、実世界の知識を実行可能な知識に埋め込み/変換していくことであり、知識の贈与と交換とが行われる世界でもある。

〔4〕ソフトウェアは、学びの副産物に過ぎない

ソフトウェアは、人が作り、人が使うものである。人、組織、社会が関わっているということは、そこには、認識の進展、知識の獲得、ノウハウの蓄積があり、これらが実は中心であり、ソフトウェアは知的活動の副産物に過ぎない。常に学び続けることが人の本性である。

〔5〕ソフトウェアは、制約条件下で創造される美しい人工物

ソフトウェアは、人間がデザインする「人工物（artifact）」であることは間違いない。しかも、概念的で目に見えないこと、最終的にはコンピュータによる実行を伴うことが特徴である。

〔6〕ソフトウェアは、富を生む経済活動の資源である

経営論の主題は、組織と組織、人と人との間の関係性である。経済活動や社会活動の目的は、富を創出すること、適切な配分を行うこと、協調して生き延びる仕組みを造ることである。人、金、もの、情報が経営資源であるが、ソフトウェアそのものも「資源」とみなすことができる。

〔7〕ソフトウェアは、言語ゲームである

ヴィトゲンシュタインは「意味の使用説」として「言語ゲーム」[6]という考え方を提唱した。「物の世界」でも「事の世界」でもない、全ては「言語的事象の世界」であるという主張である。ソフトウェアエンジニアリングというのが、プログラムコードの記述に限らず、あらゆる活動を言語活動や言語現象であるとみなすという立場は、とても有望である。

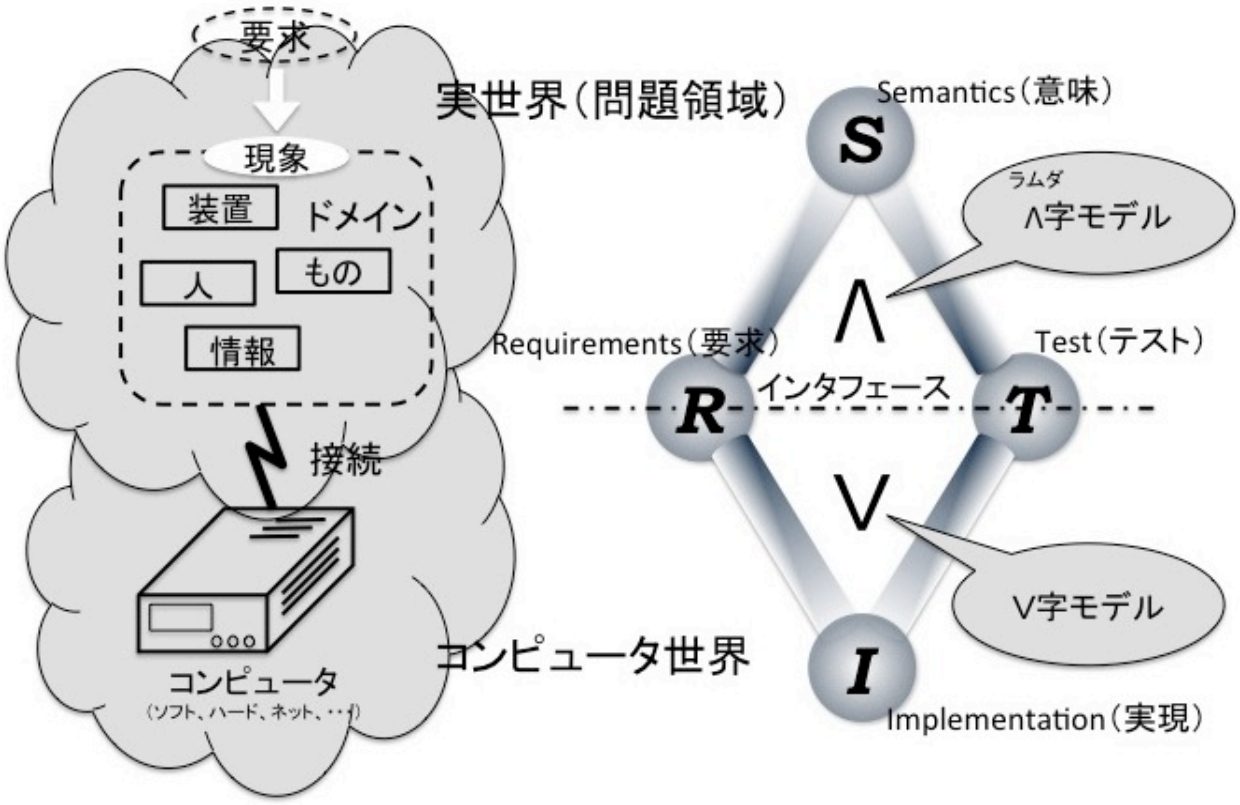

Figure 11　ΛＶモデル

今までのソフトウェアエンジニアリングの探求は、実践主体でモジュール化やプロジェクトマネジメントによる素朴な知識体系であったが、上記の７つの宣言に見られるように数学、複雑系、知識論、進化論、デザイン論、経営学、哲学といった多岐にわたる総合的なアプローチを必要としている。

3.4 ΛＶモデル

「新ソフトウェア宣言」を具体化していく第一歩として、「ΛＶ（ラムダヴィ）モデル」[7)10)]という開発プロセスを分析するための枠組みを考えた。ソフトウェアはアーティファクト（人工物）であり、そのアーティファクトをデザインすることは、そのソフトウェアが稼働するコンピュータが在り、コンピュータに接続された実世界に対する要求に従って、ソフトウェアを構築することになる。Figure 11 左側はこの関係を示している。注意してほしい事項は、「要求」は実世界の現象を対象としているということである。「ソフトウェアの要求」というと、コンピュータシステムを対象にした要求と誤解されがちであるが、本来は、コンピュータが接続された世界における現象への要求でなくてはならない。

ソフトウェア構築、あるいは、進化のプロセスを分析し、手法を構成していくためには、ソフトウェアを取り巻くコンテクストを明示的に扱うプロセスが必要になる。これをここではΛ字プロセスと呼ぶことにする。クリッペンドルフ（Klaus Krippendorff）が『意味論的転回』[8)]の中で主張しているように、「デザイン」というのは将来について意思決定していく行為であり、「人間中心」の「意味 S(Semantics)」を扱う体系でなくてはならない。これは行為・感覚・意味のプロセスに対応させることができる。ちょうどユーザの人工物（ソフトウェア）に対する「行為」、実行による「感覚」がインタフェースに相当しており、行為と感覚とから仮説推論（abduction)などによって意味の獲得、あるいは、修正が行われる。

ソフトウェアそのものも実世界に置かれて実行されるため、ソフトウェアの実行（テスト）が、ユーザの認識を変え、新たな要求を生み出すことになる。このソフトウェア開発プロセスとユーザの認識プロセスとは接合した形で一つのフィードバックサイクルを構成する（Figure 11 右側）。利用者側から見れば、要求(R)を発したことにより、ソフトウェアの実現(I)が行われ、出来上がったものを実行・テスト(T)することによってフィードバックが得られ認識や意味(S)が修正される。一方、開発者側から見れば、提供したソフトウェアの実行・テスト(T)によって、ユーザの認識を変化させ(S)、それに基づく新たな要求(R)がフィードバックとして得られ新たな実現・コード化(I)が為される。このような利用と開発との作用・反作用のダイナミックな関係性がこのモデルの本質である。

本節で述べた世界観は、ソフトウェアに限らず、一般的なデザイン対象のアーティファクトも同様にとらえることができ、デザイン論の領域の知見をソフトウェアエンジニアリングに援用することが可能になる。逆に、ソフトウェアエンジニアリングの知見をデザインの世界に応用するという全体統合の途を切り開くことができると考えている。

4. 新アジャイルプロセスマニフェスト

2001 年に提唱された「アジャイルソフトウェア開発マニフェスト」[3)]は、ソフトウェアエンジニアリングでの新しい方向性を示すことができた。筆者はこれに習い、さらなる発展に向けた新しいマニフェストを掲げることとしたい（Figure 12）。

（１） 個人能力や相互作用より、
知識主導や言語ゲームが重要である。

旧マニフェストでは、プロセスやツールにこだわるより、ソフトウェアが個人の能力や、人と人の間の相互作用やコミュニケーションが重要であるとしていた。個人の能力の中でさらに重要な事項は知識主導であることであり、また、相互作用の基盤となるのは、概念形成やコミュニケーションを通じた言語ゲームとしての活動として捉えることが要請される。特に、文化が異なる領域や組織間のコミュニケーションでは、暗黙知や態度といった語り得ない事項があり、概念や意味そのものを言語ゲームとして見る必要がでてくる。

この項目は、3.3 節で述べた新ソフトウェア宣言の〔7〕に対応した宣言文である。

（２）動くソフトウェアより、実行可能知識が重要である。

旧マニフェストでは、仕様書やマネジメントに翻弄されるより、まず動くソフトウェアが第一に大切なものとしている。ソフトウェアが実世界の中で位置づけられるということは、それを使用した人・組織や業務領域も実行されるということを意味しており、コンピュータとそれを取り巻く環境含めた「実行」が中心にならなくてはならない。

この項目は、新ソフトウェア宣言の〔3〕そのものである。デザイン対象がソフトウェアである場合には、他の一般の手法との差異が、この実行可能性によって特徴付けられることになる。

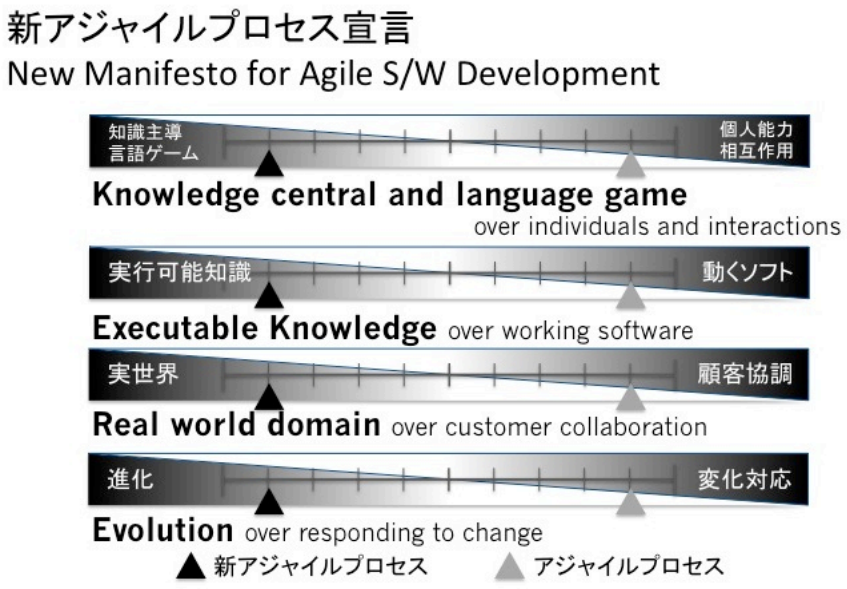

Figure 12 新アジャイルプロセスマニフェスト

（３）顧客協調より、実世界（ドメイン）が重要である。

旧マニフェストでは、契約や交渉に明け暮れるより、顧客との実質的な協調を行うべきだとしている。これは、顧客と開発者とのコミュニケーションという観点から言うと、もう一歩踏み込んで、お互いにお互いの世界を理解し合う必要性を示している。顧客はおそらく利用や実世界に通じているはずではあるが、ソフトウェアに対する要求を必ずしも知っているとは限らない。要求定義の世界で「ユーザは要求を知らない」と言われているが、このことに対応していくためには、顧客と開発者は、実世界そのもの、適用領域（ドメイン）の把握や分析に注力すべきことを示唆している。

この項目は、新ソフトウェア宣言との明確な対応は無いが、〔７〕の言語ゲーム的な世界観において、ソフトウェアが置かれる世界や状況によって、ソフトウェアの意味が規定されること、〔３〕の実行可能知識のコンセプトである「作る/使うは同じ」といったことに関連し、さらに、〔１〕による数学的な定式化や抽象機械の実現に落とし込んでいく必要がある。

（４）変化対応より、進化が重要である。

旧マニフェストの最もインパクトのある命題である。そもそもビジネスや技術革新が早くなり、より不確実性の高い状況になったために、要求や技術に対する変化対応が叫ばれたのであるが、これはより突き進めていけば、変化がなぜ起こるのかという問いをたてる必要がある。不確実性への対応には、アジャイルプロセスでは、いわば軽量マネジメントにシフトし、さまざまな作業を並行化したり、コミュニケーションや確認を迅速に行う方策であった。ソフトウェアやそれを取り巻く組織を一つの生物と見なし、進化・適応のメカニズムを追求していくことが重要である。

この項目は、新ソフトウェア宣言の〔２〕の複雑系の観点や、〔４〕の学習や継続的発展といった事項に対応している。

5. おわりに

本論文では、ソフトウェアが一般の人工物と異なる性質として、2.1 節述べた本質的困難が普遍的な前提として在り、2.2 節以降でウォータフォール型の開発プロセス、ビジネススキームなどの呪縛を示した。呪縛から解放されるための一つの契機として、3.1 節でアジャイルプロセス、3.2 節で知働化研究の活動を採り上げた。3.3 節で大局的な方向性を新ソフトウェア宣言として紹介し、開発プロセスについてはΛＶモデルを提示することによって、デザイン手法の領域との接合を図った。最後に、第 4 章でアジャイルソフトウエア開発マニフェストを発展させる形で、次のステップを踏み出すための新アジャイルプロセスマニフェスト（宣言）をまとめた。

振り返って見ると、普遍的と考えられているソフトウェアの本質的困難も、その意味が少しずつ変わってきていると思われる。例えば、複雑性は、今まではモジュール化、抽象化（高級言語の開発）、自動化といった方向で、何とか複雑性を解決しようとしてきたが、複雑系の観点が入ってくると、ありのままに複雑なものは複雑なままに扱うといった姿勢が浮かび上がってくる。同調性や可変性については、ΛＶモデルで示したような実世界とコンピュータ世界とを統一的に見るようにすると、実はそういった問題は存在しないのかもしれない。不可視性についても、何とか言語化、モデル化したり、見える化という名のもとに、さまざまな計測手段やツールが開発されてきているが、見えないものは見えないままにしておくというアプローチもあるように思える。

今後は、本講演で述べた新しいパラダイムの下での、産業構造、組織の役割分担、技法、パターン、フレーム、さらには、創造性や知的活動の源泉の探求などを進めていきたい。

謝辞

本研究を進めるにあたり、知働化研究会のコンセプトリーダである山田正樹氏、アジャイルプロセスに関して長年にわたり研究・開発を進めてきた濱勝巳氏、飯泉純子氏、そして、討論に参加していただいたアジャイルプロセス協議会のメンバの方々、さらに、デザイン論に関する研究の契機をいただき、さまざまなご指導をいただいた永井由佳里氏に、感謝の意を表します。

参考文献

1) Frederic P. Brooks, Jr.: Mythical Man-Month / The Essays on Software Engineering, Anniversary Edition [Special Edition], Addison-Wesley Professional, 8(1995) （フレデリック・ブルックス Jr, : 人月の神話, ピアソンエデュケーション, 11(2002)）
2) 大槻繁: ソフトウェア開発はなぜ難しいのか, 技術評論社, 11(2009)
3) アジャイルソフトウェア開発マニフェスト, http://agilemanifesto.org/, 2(2001)
4) 山田正樹,大槻繁, : 対談/人働説から知働説へ, 知働化研究会, 7(2009)
5) 大槻繁,濱勝巳ほか: 新ソフトウェア宣言,ソフトウェア技術者協会, ソフトウェア・シンポジウム 2010, ソフトウェアエンジニアリングの呪縛 WG, Post Proceedings WG1, 6(2010)
6) 黒崎宏：ウィトゲンシュタインの生涯と哲学, 勁草書房, 10(1980)
7) 大槻繁：ΛＶ モデル / Ｖ字モデルからの意味論的転回, 知働化研究会誌, **1**, 11(2010) , 105-131
8) Klaus Krippendorff: The Semantic Turn / a new foundation of design, Taylor&Francis Group, LLC, (2006)（意味論的転回 : デザインの新しい基礎理論, 星雲社, 4(2009)）
9) 大槻繁: 実行可能知識としてのソフトウェア構築プロセス, 日本デザイン学会/ デザイン学研究 Bulletin of JSSD 2012, 6(2012)
10) 大槻繁: 実行可能知識のデザインプロセス, デザイン学研究特集号, JSSD, **19-4 No.76**, (2012), 22-31