実行可能知識のデザインプロセス　創造的ソフトウェア開発プロセスΛVモデル

A Creative Software Development Process as Executable Knowledge
- The Semantic Turn from V-Shaped Process to ΛV Model

大槻 繁
株式会社一(いち)

OTSUKI Shigeru
Ichi Co., Ltd.

Abstract : 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 in accordance with the process of 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 Λ V-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.

Key Word : Software Development Process, Executable Knowledge, Λ V-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．

本論文では，従来の工業的なソフトウェア開発から脱し，多様で創造的な活動を目指すデザイン論の観点を取り入れた新しいプロセス「ΛVモデル」を提案する．ソフトウェア開発におけるユーザ／ベンダ，利用者／開発者，初期構築／保守運用といった役割分担は意味を持た なくなってきており，ソフトウェアという言葉自身も「実行可能知識」とでも呼ぶべきものである．本論文は一つのパラダイム提唱であり，従来のソフトウェアエンジニアリングの知見を，造形物を生み出すデザインプロセスに適用することと，逆に，デザインプロセスの考え方をソフトウェア開発に持ち込み，創造的な活動の契機とすることを目標としている．

第1章ではΛVモデル提案の背景となる歴史的背景や近年台頭してきているアジャイルプロセスの動向を概観する．第2章ではΛVモデルを基礎づけている哲学，理論的枠組みについて解説する．第3章ではこのモデルがもたらす社会的インパクトと今後の課題について言及する．第3章ではこのモデルがもたらす社会的インパクトと今後の課題について言及する．

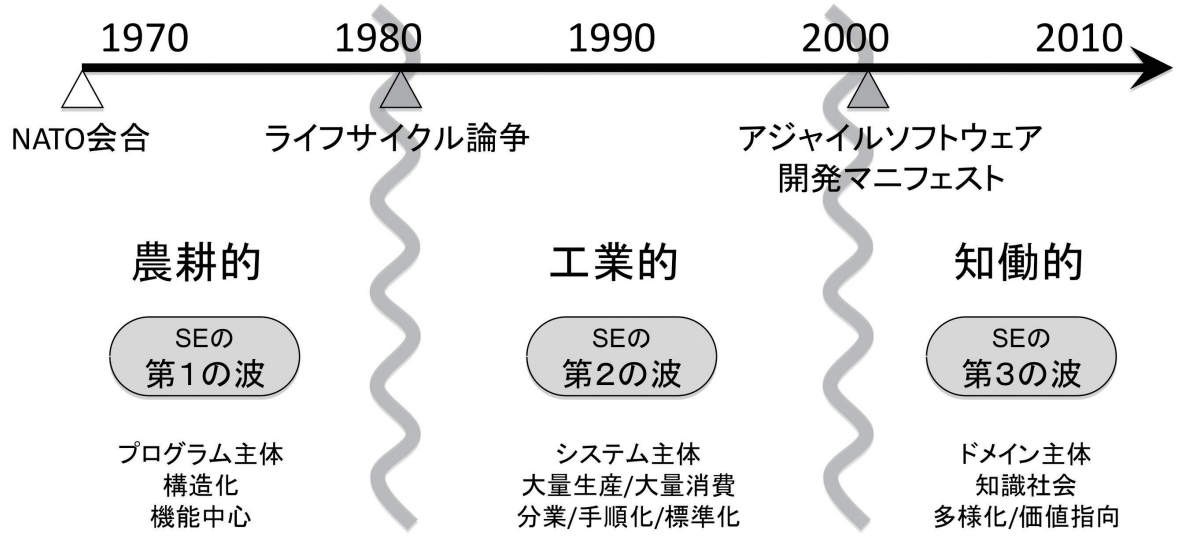


図1．ソフトウェアエンジニアリングの『第三の波』

2. ソフトウェアエンジニアリングの潮流

2.1. 第三の波

ソフトウェアエンジニアリングの発祥は，1968年にドイツのガルミッシュ・パルテンキルヒェンで開催されたソフトウェアエンジニアリングカンファレンス（通称 NATO 会合）と言われている．本格的にソフトウェアエンジニアリングの研究が行われたのは，1970年代からである．以降，さまざまな方法論や手法が提唱されている．高々半世紀程度のことであるが，その歴史は図1に示すように，アルビン・トフラーの『第三の波』[1] に擬え，3つの波，すなわち，不連続なパラダイムシフトがあったと解釈できる．

第1の波は「プログラム」中心の時代である．プログラム開発そのものを研究対象とする技術が明確に意識され，見通しのよいプログラム構造，階層化，テストなどの技法，情報隠ぺいやモジュール化といった基本原理が次々に提唱された．

第2の波はプログラムの総体としての「システム」中心の時代である．1980年代初頭のソフトウェアエンジニアリングの学会誌を中心とした「ライフサイクル論争」が起こったことが契機となっている．プログラムという実装に対して，「仕様」という概念が確立し，大規模化し，多くの人々が携わって組織的に開発することが常態化した．製造業の工場生産のように，分業，手順化，標準化によって，大量にソフトウェアを作る時代である．

第3の波は「知識」中心の時代である．1990年代後 半からインターネットが急速に普及したこともあい まって，今世紀に入り問題把握，要件定義，ビジネスプロセスと開発プロセスの統合等に関心が集まっている．2001年に発表された「アジャイルソフトウェア開発マニフェスト」[2] がその状況を的確に表している．多様化した価値観，新しい世界，新しいビジネスモデルを生み出す知的活動が中心になっている．本論文で提案するパラダイムは，第3の波（これを「知働的」な時代と呼ぶ）を中心としたものである．

2.2. 技術のもたらす価値と解

ソフトウェアエンジニアリングのさまざまな知見を整理すると，図2に示すように価値と解のマトリクス[3] のようになる．これは2008年の春に，アジャイルプロセス協議会に参加している会員にヒアリングを行ったものに基づいている．

ここで言う「価値」とは，組織・チーム・人による期待感（期待効用）であり，三種に分類して考えている．「組織（マネジメント）」の価値とは，ビジネス領域や業界で評判（reputation）を得るマネジメントができており，組織の知的資産価値を提供するベースラインとなるものである．「不確実性(変化への対応)」の価値とは，マーケットの状況，ビジネス環境の変化といった不確実性に対処できることである．「進化・適応(学習と成長)」の価値とは，組織の事業継続，システムの維持や保守，自律的な修復能力，そして，学習や成長，イノベーションを図る能力のことである．

一方，「解」は，開発方法論，ツール，プロセス等の構成要素を因数分解してみると，最終的には，「抽象化」「自動化」「モジュール化」に帰着できると考えられる．「抽象化（Abstraction）」とは，問題を解決するために不要な情報を捨象（捨て去る）ことである．プログラミング言語の高級化も，ハードウェアやオペレティングシステムの個別のことを知らなくても，計算や操作の指示ができるように抽象化を図っていると言える．「自動化（Automation）」とは，計算機や機械，さらには，人手によって網羅的・機械的に作業や業務を実行できるよ

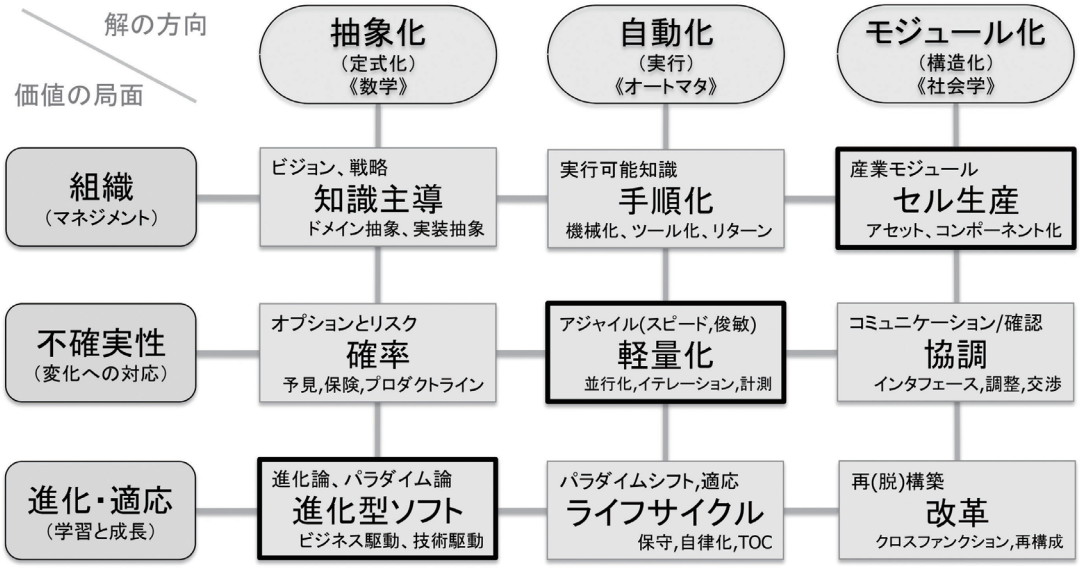


図2．ソフトウェアエンジニアリングの価値と解

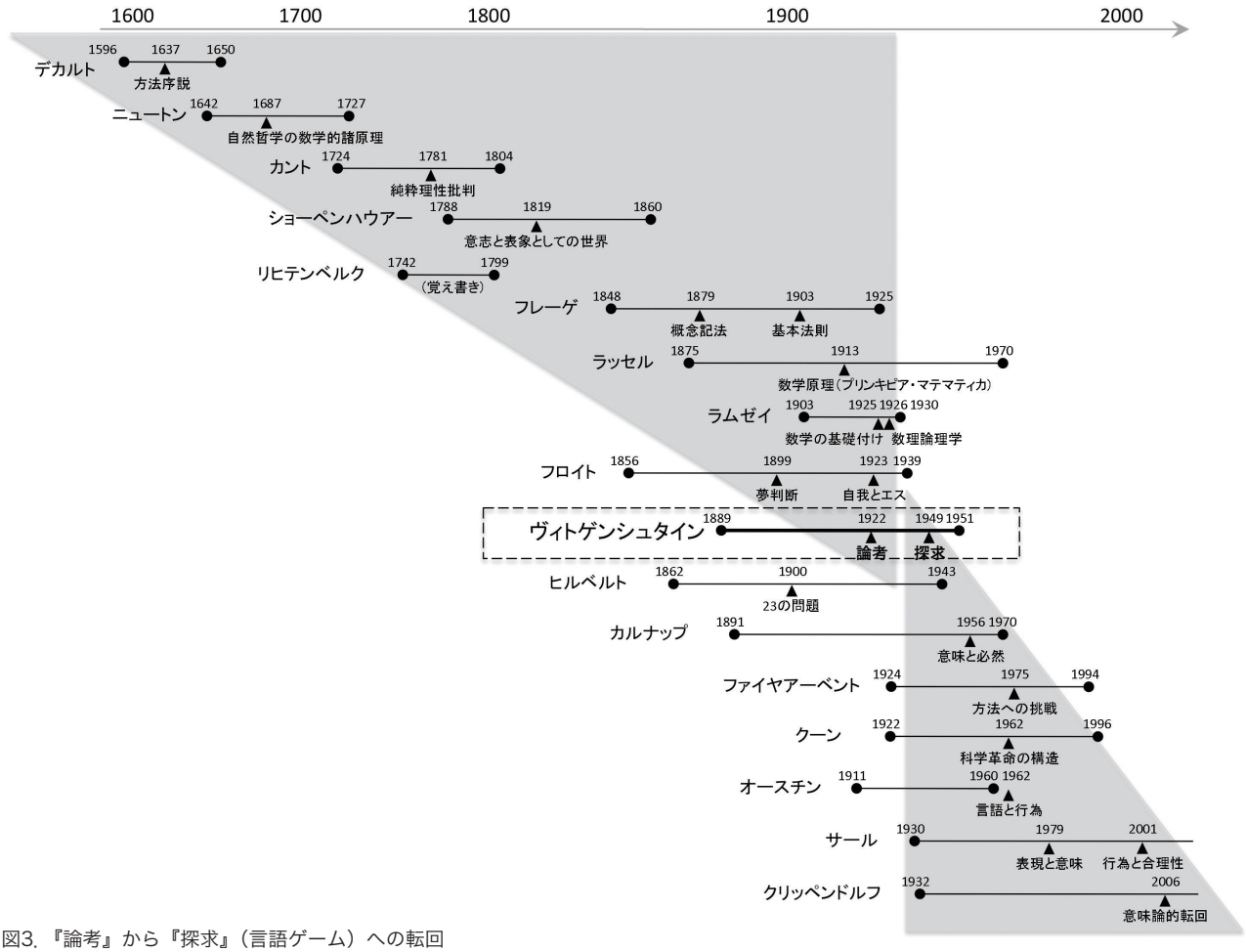


図3.『論考』から『探求』(言語ゲーム)への転回

うにすることである．コンパイラによる言語処理や変換，ツールによる処理は典型的な自動化であり，定型的な業務遂行も人手による自動化と考えられる．「モジュール化（Modularization)」とは，ソフトウェアの構成要素を分割することや，組織において分担が可能なように役割やルールを設定することである．この価値と解のマトリクスによって整理することによって，ソフトウェアエンジニアリングの技術潮流の解釈や予測ができる．2.1節で述べた第3の波は，アジャイルプロセスの台頭によって特徴付けられるが，不確実性の高い環境，それに起因した要求変化に対応するために，安定と不安定事項を分離する手法，俊敏で軽量な開発プロセス，顧客や関与者間のコミュニケーションを的確に行う方法などが提唱されていることが分かる．さらに，近年ではシステムの継続性や進化に，技術の中心がシフトしてきていると考えられる．

2.3. 新ソフトウェア宣言

我々は，第3の波の時代にいるにもかかわらず，ソフトウェアの開発方法は依然として第2の波の工業的な考え方に呪縛されている．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』[4] 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み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

ウィトゲンシュタインは「意味の使用説」として「言語ゲーム」[5] という考え方を提唱した．「物（モノ）の世界」でも「事（コト）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．無論，無意識や心理的な事項もあることは認めるが，最終的には言語として現れることになる．この哲学は，図3に示すように，デカルトやニュートン以来の世界観を決定的に覆し，以降の言語行為論やデザイン論に多大な影響を与えた．この新しい世界観をソフトウェアの世界に適用することによって，新たな地平を開くことができると考えている．

□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. ΛVモデル

3.1. アーティファクトの世界観

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あり，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．図4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」と言うと，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．

これを模式的に図式で示したものが図5である．世界はまとめて分析・検討することができる複数のドメインから構成される．コンピュータやその上のプログラム（ソフトウェア）を含めて，「機械（マシン）」と呼ぶ．機械も一つのドメインとみなされることもある．ドメイン間の接続は，両者ドメインの間の共有現象であり，これを「インタフェース」と呼ぶ．特に，機械とドメインの間の共有現象を「仕様」と呼ぶことがある[6]．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る．

3.2. ソフトウェアの本質的特性

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

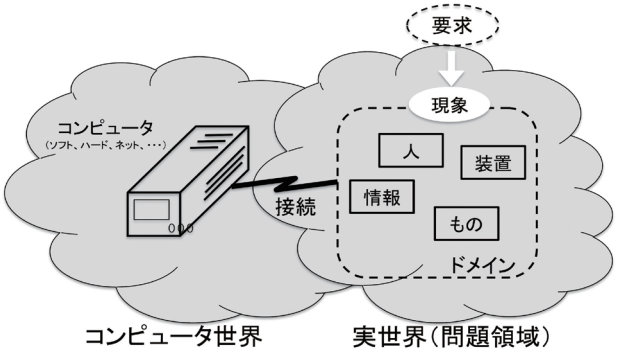


図4. 機械・実世界・要求の関係

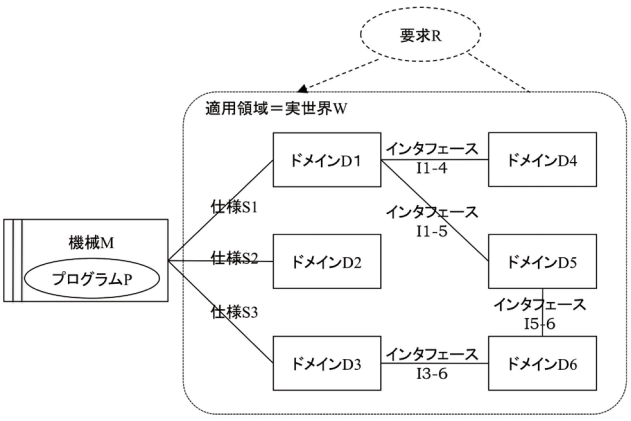


図5. ドメインとインタフェース

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』[7] 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）とは，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）とは，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至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）とは，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，新たな要求がでてくることになる．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）とは，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アーティファクトと異なる性質として，「実行可能性（executable)」であることが挙げられる．ソフト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メカニズムが必要である．

3.3. 山モデル

マニー・レーアン（M.M.Lehmann：英国インペリアルカレッジ）[8] が図6に示すように，Eタイプ（Embedded-type）と呼ばれる考え方を提唱している．“embedded” は「実世界に埋め込まれる」という意味である．「モデル化」というのは，あらかじめ決められた言語によって記述するということである．つまり，言語という色眼鏡をかけて見えたものを記述するということである．見えないところ，つまり，捨象されてしまう事柄もある．従って，モデル化には必ず抽象化が伴う．仕様が実世界の正しいモデルになっていることを，「確認（validation)」する必要がある．これと同時に，プログラムが仕様を満たすことを「検証（verification)」しなくてはならない．確認とは，記述の正しさを，実世界の状況に照らし合わせて人間が判断することである．実世界そのものは全て記述することができないために，人間が記述やプログラム動作を見て判断するしか方法がない．これに対し，検証とは，二つの記述の間の無矛盾生や完全生，充足可能性といったさまざまな性質を，数学的

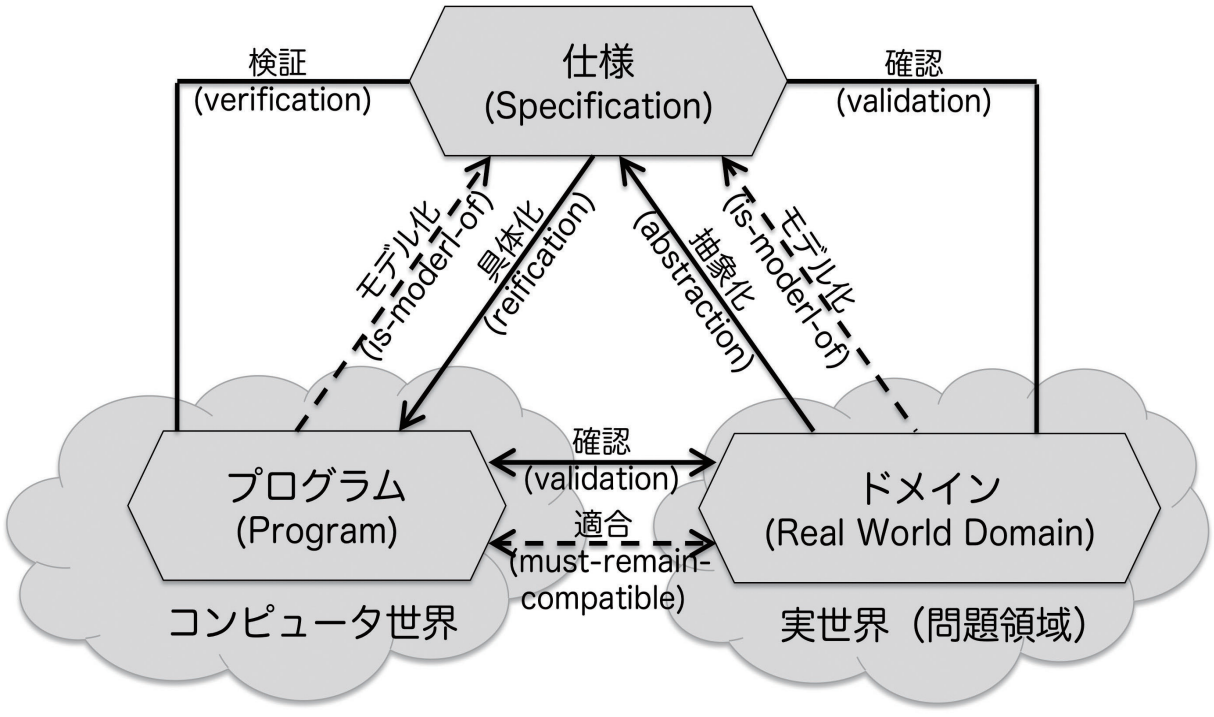


図6．Eタイプソフトウェア

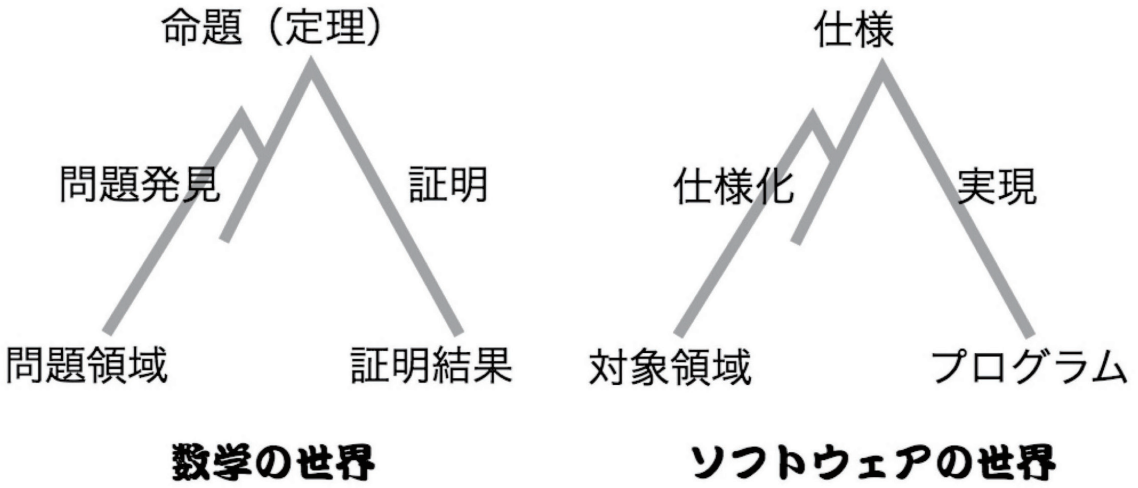


図7．抽象山モデル

な証明，テスト等によって判断することである．仕様が形式的な仕様記述言語で記載された場合には，ある限られた条件下では，プログラムが仕様を満たすことを全て証明によって判断することも可能である．

得られたプログラムと実世界との関係は，実世界でプログラムが正しく動作することを「確認」することと同時に，これが「適合」していくように，実世界とプログラムとが両立可能な変化をしていく必要がある．実世界はプログラムの実行によって何らかの影響を受けるので，実世界も変化する．無論，実世界の変化によって，プログラムが何らかの不都合を生じる場合には，それを修正していく必要がでてくる．この関係を常に保ち続けることが進化型のプログラムに要求されていることである．

Eタイプをより単純化した考えが，抽象山モデルである[9]．ソフトウェア開発というのは，図7に示すように，対象領域にある問題をプログラムによって解くことで，対象領域にある問題を抽象化して記述したものが「仕様」で，この抽象化のプロセスを「仕様化（要件定義）」と呼ぶ．また，仕様を満たすプログラムを作る具現化のプロセスを「実現（設計・プログラミング）」と呼ぶ．

ここで言う抽象化とは，「問題を解決するために不要 な情報を捨象すること」である．仕様化と実現との関係は，数学の世界に対応 させて考えると分かりやすい．問題領域（実世界）にある問題を抽象化して，方程式（命題や定理）として定式化し，方程式を立てる．この得られた方程式を解き，現実世界の解を得る．ソフトウェア作りをするということは，「抽象山」という山を登り，降りるようなものである．抽象化という観点から見れば，仕様化の方が，実現より難しい．もしツールを活用して自動化ができるとすれば，抽象山を降りる「実現」の部分に限られる．

「仕様」が，対象領域（実世界）の抽象化でもあり，かつ，プログラム（コンピュータ世界）の抽象化でもあるところが重要である．すなわち，仕様は実世界とコンピュータ世界との「インタフェース」である．インタフェースというのは，異なる世界を結ぶものである．同じインタフェースの記述が実世界から見た場合と，コンピュータ世界から見た場合とでは意味は異なる．これは天動説を信じている人と，地動説を信じている人が，同じ「太陽」を見ても，異なる意味で解釈するというのに似ている．実世界からインタフェースの記述を見ると，コンピュータの世界の実装に関わる事項は捨象される．他方で，コンピュータ世界から見ると，実世界の事項は考慮しなくてよいということになる．

3.4. 実行可能知識

実世界で作用し合うソフトウェアを対象とし，価値駆動のソフトウェア作りを目指す「知働化」パラダイムのコンセプトは，以下のように集約される[10]．

ソフトウェアに対する新しい見方

- ソフトウェアとは，実行可能な知識の集まりである
- ソフトウェアとは，実行可能な知識を糸や布のように紡いだもの（様相）である
- ソフトウェアを作る／使うとは，現実世界に関する知識を実行可能な知識の中に埋め込む／変換する過程である
- ソフトウェアを作る／使う過程では，知識の贈与と交換が行われている

- ソフトウェアを作ることと使うことの間には，本質的な違いはない
- 様相／テクスチャとは，「動く，問題と解決の記述」のことである
- 「機能」を実現することから，顧客の「知識」をコンテンツ化し，実行可能にすることへ

従来のパラダイムでは，ソフトウェアがもたらす価値，ビジネスモデル，要求の発生プロセス，ソフトウェアの実行による実世界での認識の変化といった事項に対して思考停止していると考えられる．これはパラダイムシフトであり，従来の世界観や価値観は通用しない．ソフトウェアというのは，元来，実世界の問題を解決するものである．既に解かれている同様の問題を，何回も解き続けるということなら，従来の，工業的なパラダイムで済む．実際に，画面のレイアウトや入出力の仕様が決まったら，そのコードを書くことは，手順化されているし，自動化も可能である．

実世界におけるソフトウェアのもたらす意味は，人，あるいは，組織にとって主観的であり，多様である．また，ソフトウェアは，人間が創造するアーティファクトである．アーティファクトの「デザイン」に関する基礎理論，哲学として，知働化のパラダイムに最も適合している考え方が，クラウス・クリッペンドルフ（Klaus Krippendorff）の『意味論的転回』[11]である．

クリッペンドルフの主張は，「科学」というのは，過去に起こったことを分析して何か法則を見いだしたり証明したりするのに対して，「デザイン」というのは将来について意思決定していくので，ぜんぜん違う，「人間中心」の「意味」を扱う体系でなくてはならないということである．

デザインでは，人間の理解，意味とはどういったものかを明確にしておかなくてはならない．無論，デカルト主義を排す立場では，絶対的，客観的真実というのは無い．従って，ヴィトゲンシュタインの「言語ゲーム」的に規定していくことになる．

意味とは，以下のように規定される．

- 意味は，構造化された空間，期待される感覚のネットワーク，一連の可能性である．意味は行為をガイドする．
- 意味は，いつも誰かの構築物である．
- 意味は，共有できない．
- 意味は，言語の使用の中で現れる．
- 意味は，固定されない．
- 意味は，感覚によって引き起こされる．
- 意味は，知覚されたアフォーダンスである．

図8で「外部の世界」というのが，ソフトウェアの実行による作用によって影響を受ける実世界である．「意味」そのものは，直接語ることはできない．人間が持つ意味は，実世界に対する「（言語）行為」として現れ，その行為による因果関係によってもたらされる現象が，「感覚」として帰ってくる．

人工物をデザインするには，この枠組みに従って，2次的理解を定義する．つまり，人々が「理解することを理解する」ことによって，それをデザイナーの意思決定に活かしていけるような枠組みを提示している．この枠組みは，図4のソフトウェアの機械・実世界・要求の関係や，図7の抽象山モデルとの関

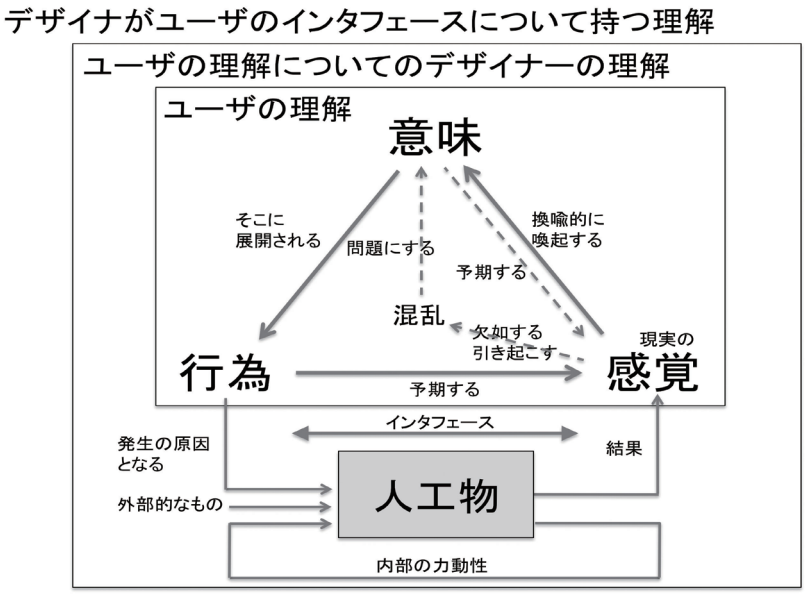
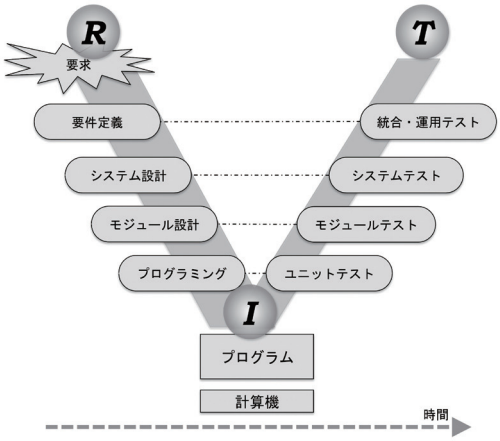


図8．人工物（アーティファクト）のデザイン

図9. V字型開発プロセスモデル



係と整合させることができ，デザイン論をソフトウェアエンジニアリングの世界に援用できることを示唆している．

3.5. ΛVモデル

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」がある．これは図9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．象徴的に要求をR（Requirements），プログラムをI（Implementation），テストをT（Test）で表すことにする．

ウォータフォール型開発プロセスでは，厳格に左から右に一つずつ進めていく．アジャイル型開発プロセスでは，V字全体を俊敏に細かく繰り返しながら進めていく．

V字モデルは，ソフトウェア開発者の世界の説明のモデルとしては優れているが，要求は外部から与えられるものであり，テストもその確認（validation）の論拠を示すことができない．また，ソフトウェアを人工物としてとらえた時に，一般のデザイン論で言う今までにない未来指向の理想追求型[12]のデザイン活動を扱うことができない．あくまでも，ソフトウェアは実世界の問題を解決するためにあり，その要求はクライアント側から提示されるものという扱いになっている．

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．クリッペンドルフが『意味論的転回』[11]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．ΛVモデル[13]の概要を図10に示す．

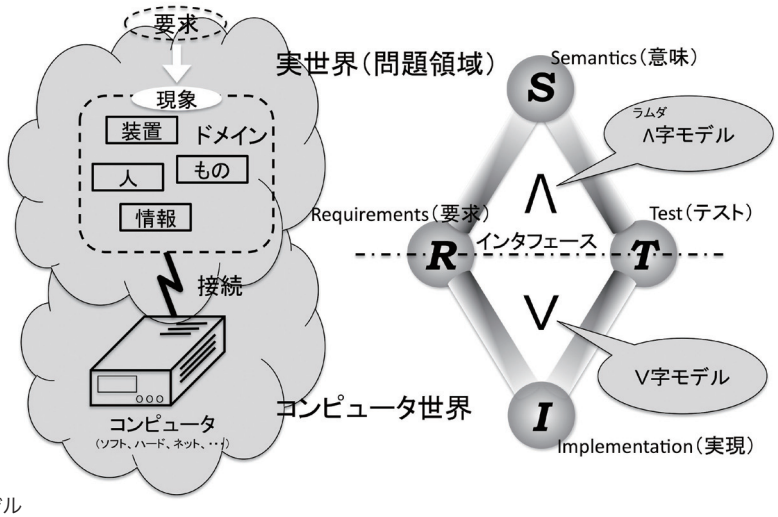


図10. V字型開発プロセスモデル

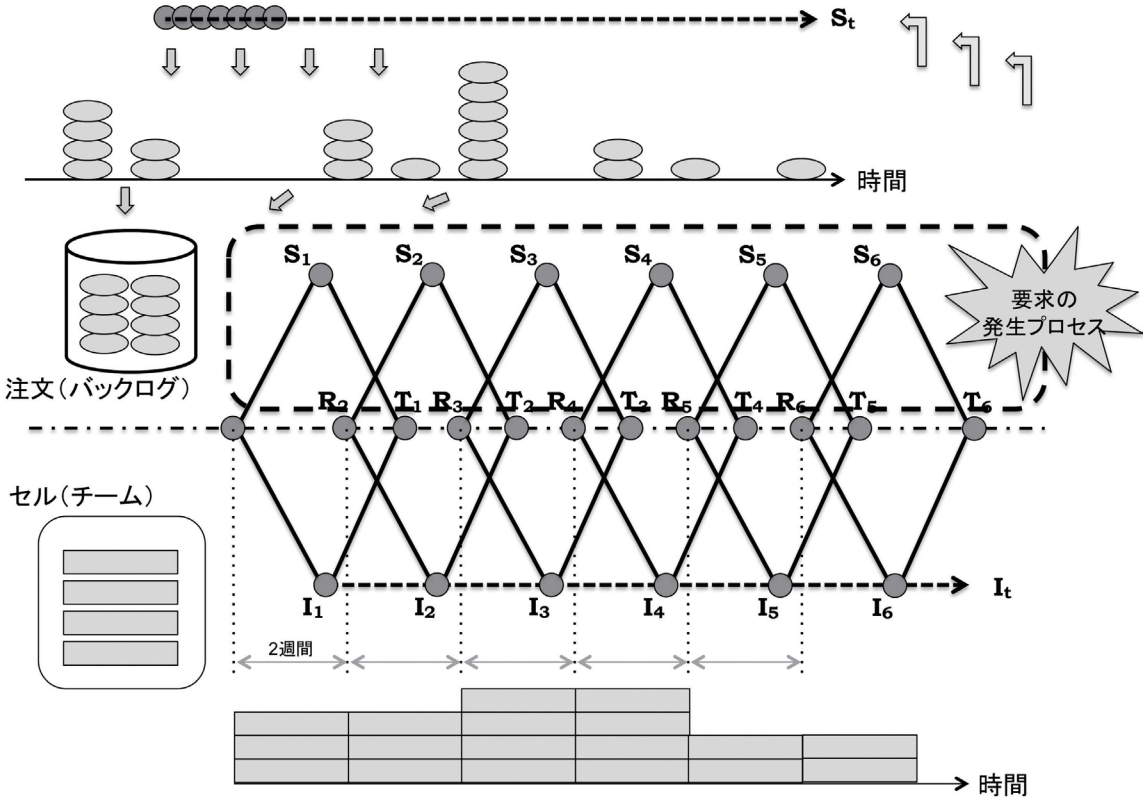


図11．タイムボックス方式（アジャイル）プロセス

実際にアジャイル型開発プロセスの一つである，ソフトウェアセル生産方式では，一定の期間に固定されたタイムボックス作業を繰り返しながら，時々刻々とユーザ側から発生する要求を，順次実装していく方法が採られる．図11に示すように，こういったプロセスをΛVモデルによって説明することができる．

4．おわりに

従来のソフトウェアでは，ユーザ，ベンダといった役割分担による仕組みで開発してきたが，ビジネス環境の変化が激しく，対応しきれなくなってきている．抜本的に諸活動を見直し，組織境界を再定義し，再構成していかなくてはならない．図12は，近年，ビジネス環境に 俊敏に対応していくために，「内製化」が進められている組織境界の変異を示している．

V字モデル内部の改善では限界がある．新しいソフトウェア作りの世界を築いていくために，従来のソフトウェア開発プロセスで扱っていなかった領域に踏み込んでいかなくてはならない．今後，このプロセスを具現化し，変化や進化のダイナミズムに耐えうるアーキテクチャやパターンの検討を進めていく予定である．

第3節で述べたΛVモデルは，第2節で述べた呪縛に対して完全な解放には至っていないが，ある程度の方向性は示している．特に，ソフトウェアへの各ステークホルダの関わり方は多様であり，全体のコンセプトを宣言[5]に従っている．諸活動を「言語ゲーム」として扱うという哲学は宣言[7]に，ソフトウェアエンジニアリングとデザイン論とを統合していく方向は宣言[5]に関係している．

創造的ソフトウェアを対象とした時にも，多くに人々が活用できる技法を構築できると考えている．3.1節の図5のドメインとインタフェースの図式の整理は，ソフトウェア構築のパターンを発見していく手助けになる．ジャクソンは『プロブレムフレーム』の中で，5つのパターンを示し，あたかも，数学の問題を解く際のガイドを示すように，ソフトウェアの世界での問題の把握とその解き方を提示している．この考え方をΛVモデルで扱っている範囲，すなわち，実世界やその認識，さらには，そこでの認識の変化にまで広げることにより，より強力な知見を発見・蓄積していくことができると考えている．

ソフトウェアが一般のデザイン対象と異なる最も重要な差異は「実行可能」なことであり，これを解明し，手法として確立していくことが，ソフトウェアの「デザインエンジニアリング」とでも言うべき新たな地平を切り開いていく方向だと確信している．

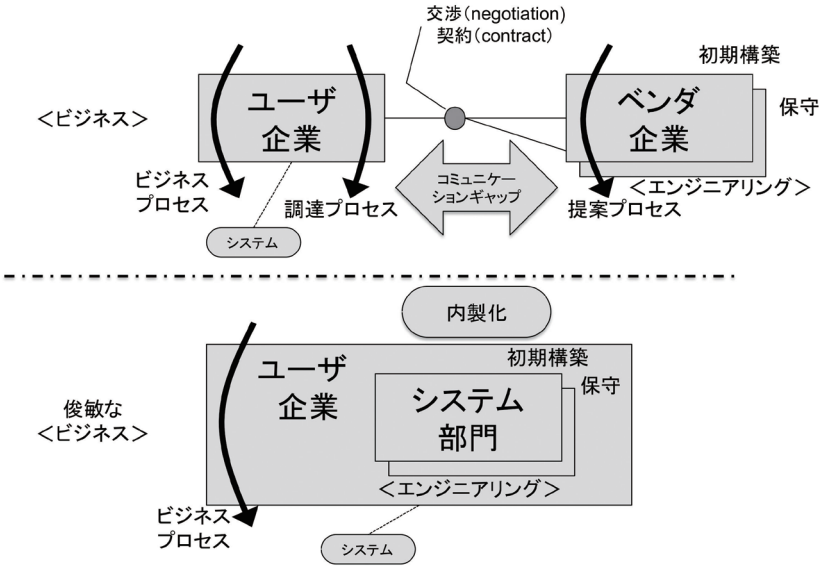


図12．タイムボックス方式（アジャイル）プロセス

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，ヒアリングや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） Alvin Toffler, The Third Wave, HarperCollins Publishers Ltd, 1980.7（アルビン・トフラー，第三の波，日本放送出版協会，1980.10）
2） アジャイルソフトウェア開発マニフェスト，http://agile-manifesto.org/，2001.2
3） 大槻繁，価値駆動アプローチのすすめ，EM-Zero Vol.1, 2008.8
4） 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, 2010.6
5） 黒崎宏，ウィトゲンシュタインの生涯と哲学，勁草書房，1980.10
6） Michael Jackson, Problem Frames: Analyzing and structuring software development problems, Addison-Wesley, 2001（マイケル・ジャクソン，プロブレムフレーム：ソフトウェア開発問題の分析と構造化，翔泳社，2006年5月）
7） Frederic P. Brooks, Jr., Mythical Man-Month, The: Essays on Software Engineering, Anniversary Edition [Special Edition], Addison-Wesley Professional, 1995.8（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，2002.11）
8） M.M.Lehmann, Software Evolution and the Real World, Lecture Notes on University Sannio, 2001 May
9） 大槻繁，ソフトウェア開発はなぜ難しいのか，技術評論社，2009.11
10） 山田正樹，大槻繁，対談：人働説から知働説へ，知働化研究会，2009.7
11） 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，2009.4.1）
12） 田浦俊春，永井由佳里，デザインの創造性と概念生成，Cognitive Studies, 17(1), 66-82, 2010.3
13） 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 Volume1，105-131，2010.11