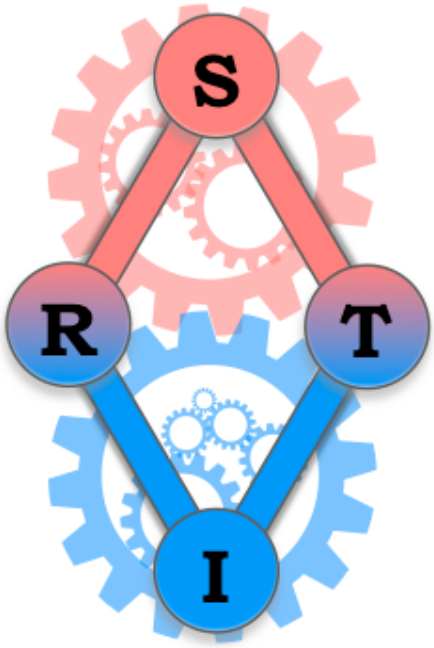
小論：知働化研究会 1/1/2010

新春特別寄稿

Λ∨モデル

∨字モデルからの意味論的転回



大槻繁（おおつきしげる）

株式会社一（いち）副社長
アジャイルプロセス協議会 フェロー
知働化研究会 運営リーダ

はじめに

本小論は、知働化のパラダイムに移行していくために、あえて、伝統的なソフトウェア開発プロセスの基底となっているV字モデルを転回して、知働化の意義を再認識してみようという試みです。

筆者は、知働化をポスト・アジャイルプロセスと位置づけており、「不確実性への対処」の次に来るものは、「進化・適応」のプロセスだと考えています。一方で、伝統的なプロセスは依然として残り続けることでしょう。そこで、これ等両者の関係を整理しておくことも、それなりに意味があると考え、「ΛVモデル」というハイブリッドなプロセスモデルを考えてみました。

表紙の絵柄の背景に配置した歯車の図は、ここのところ筆者が気に入って使っているものです。青で示したITシステム・ソフトウェア、あるいは、エンジニアリングの世界と、赤で示したビジネス、あるいは、実世界とが、連動してかみ合うべきだということを象徴的に表現したものです。青と赤の色使いは、全体を通して一貫させています。

全体構成は、4部構成で、ⅠとⅡが前置きとか背景で、中心はⅢです。最終章Ⅳでいくつかのケーススタディをモデルの観点から考察しています。

Ⅰ．V字モデルの復習

ウォータフォール型開発プロセスの原理を説明するモデルとしてV字モデルを復習します。

Ⅱ．知働化への転回

知働化のエッセンスを紹介し、開発プロセスの観点からどういった転回が進んでいるかを簡潔に示します。

Ⅲ．ΛVモデルの概要

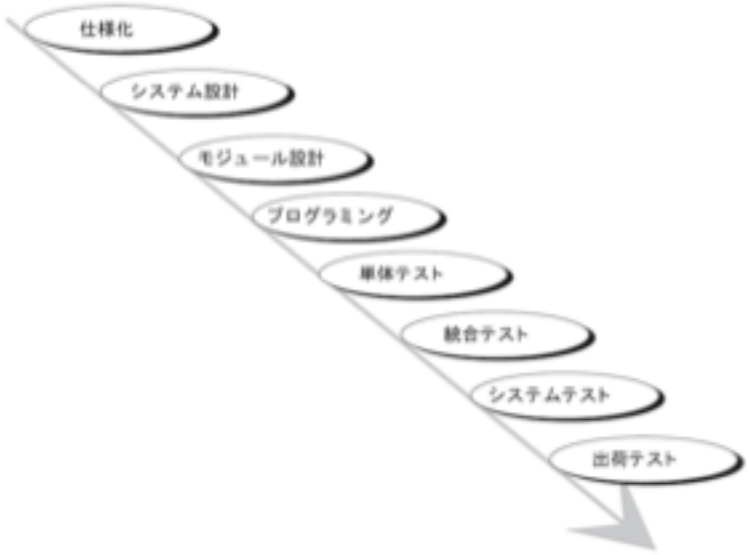
V字とそれに対峙するΛ字の部分の関係、モデルのポイントを解説します。

Ⅳ．いくつかのケーススタディ

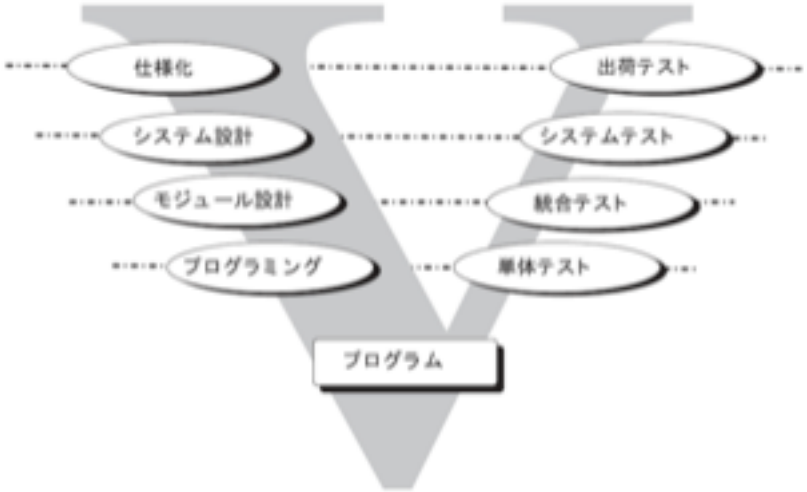
身近な事例をΛVモデルの視点から眺めてみることにします。

Ⅰ．Ｖ字モデルの復習

ウォータフォール型開発プロセスは、前工程で得られた中間成果物を確定・固定化した後に、それを前提として該当する工程の作業を行う方式です。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ります。「ウォーターフォール型」という名前の由来も、工程を進めるということを、滝を下ることに見立てているところにあります。滝を逆に登るのが大変であることも、手戻りのコストが高いということをうまく表しています。



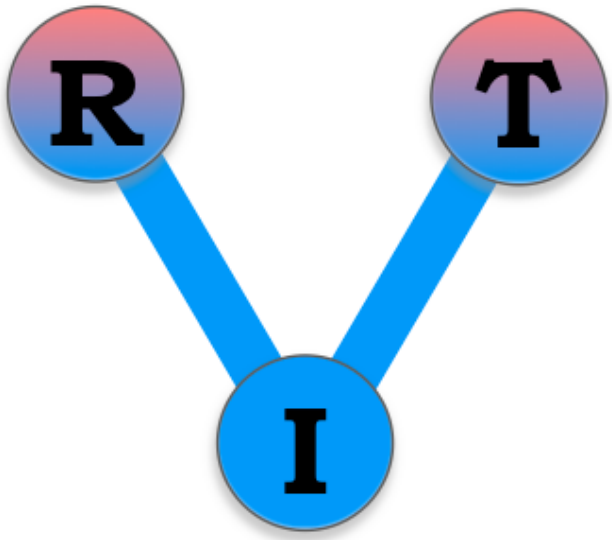
仕様化、システム設計、モジュール設計、プログラミングのそれぞれの工程の正しさを、出来上がった成果物＝プログラムに照らし合わせて確認する工程がテスト工程です。このことをよりわかりやすく表したのが、「Ｖ字モデル」です。



ウォータフォール型開発プロセスをV字モデルによって解釈することによって、旧来の開発プロセスの欠点が明らかになります。それは、最初に行った意思決定の正しさが最後にならないとわからないことです。

V字モデルは、各工程が左側から右側へ時間的な順序を表わしていると見なせばウォータフォール型開発プロセスになりますが、時間的な関係を捨象してしまえば、段階的モデル（Incremental Model）や進化型モデル（Evolutionary Model）などの説明のモデルとしても使用することができるようです。

V字モデルの工程の個数や名称にさほど意味があるわけではありません。そこで、簡潔に、象徴的に以下のように表わすことにします。

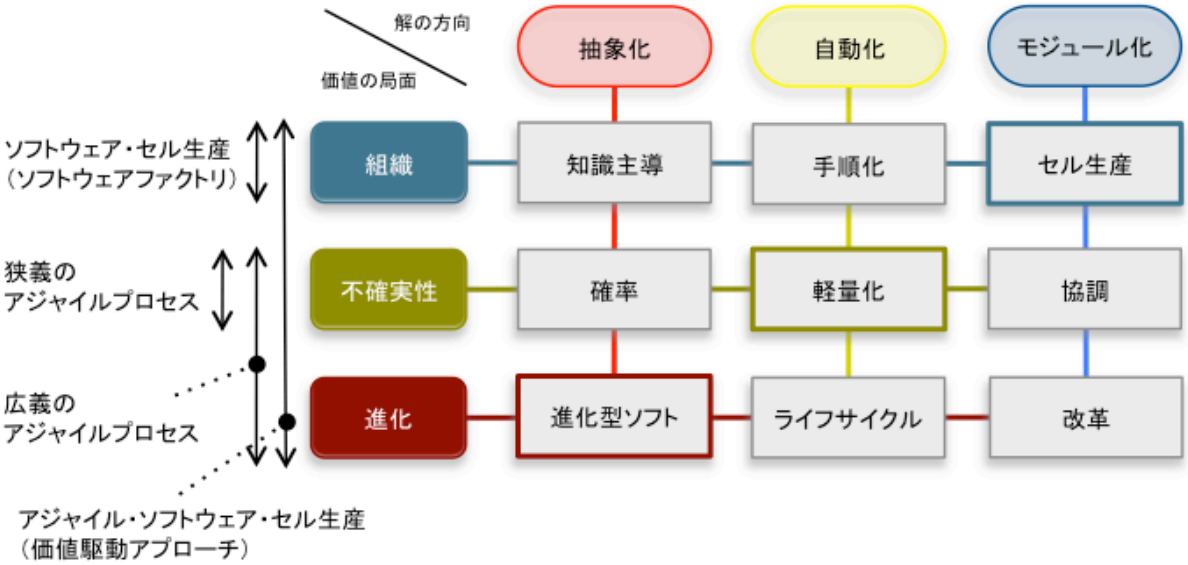


R は要求（Requirements）、I は実現（Implementation）、T はテスト（Test）を表わしています。「実現」とは実行可能なコード、および、そのコードを構築するための設計、さらに、その設計を確認するための内部テストも含みます。「要求」は実現されたコードの作用の対象となる実世界の現象に対する要望や願望です。その一部がコードに対する仕様に対応しています。「テスト」とは、「実現」が「要求」を満たしていることを、コードを実行することによって確認することです。

非常に簡単に言えば、R という期待を持ち、I を実行させ、R の期待通りかどうかを T によって確かめるということです。

Ⅱ. 知働化への転回

アジャイルプロセス協議会の活動の中で、アジャイルプロセスの意義や今後の方向性について、検討を重ねてきました。2008 年 6～7 月にかけて、見積・契約 WG のメンバの方々と議論して、最終的に以下のような方向性を得ることができました。特に、アジャイルプロセス、および、それを遂行する組織が生み出す価値が、不確実性への対応の段階から、ライフサイクルを通じた進化（と適応）に移行していくという共通認識に至りました。



2009 年夏に発足した知働化研究会では、実世界でのソフトウェアの作用に焦点をあて、価値駆動のプロセスを検討し始めています。知働化のコンセプトは、以下のように集約されます。

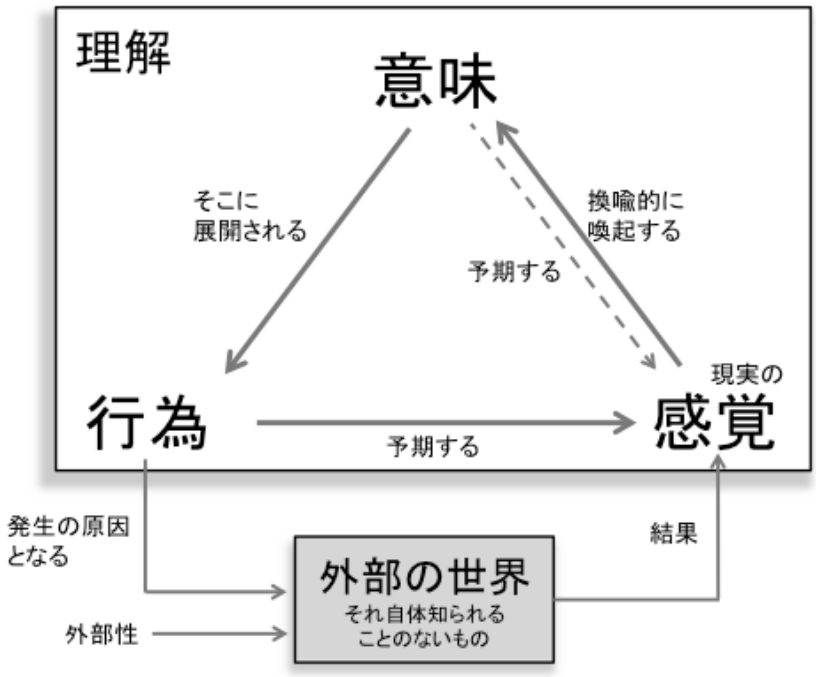
- ✓ ソフトウェアに対する新しい見方
  - ➢ソフトウェアとは，実行可能な知識の集まりである
  - ➢ソフトウェアとは，実行可能な知識を糸や布のように紡いだもの（様相）である
  - ➢ソフトウェアを作る/使うとは，現実世界に関する知識を実行可能な知識の中に埋め込む/変換する過程である
  - ➢ソフトウェアを作る/使う過程では，知識の贈与と交換が行われている
  - ➢ソフトウェアを作ることと使うことの間には，本質的な違いはない
- ✓ 実行可能知識と様相／テクスチャ
  - ➢様相／テクスチャとは、「動く、問題と解決の記述」のことである
  - ➢「機能」を実現することから、顧客の「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ます。

これはパラダイムシフトです。従来の世界観や価値観は通用しません。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す。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でしたら、従来の、工業的なパラダイムで済むでしょう。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、手順化されているでしょうし、自動化も可能でしょう。筆者は、このようなパラダイムは、たとえ大規模化や組織化が必要だとしても、本質的な課題はそこには無いと考えています。

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、多様です。また、ソフトウェアは、人間が創造する人工物（アーティファクト）です。人工物の「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』です。

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す。



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りません。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わけで、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ります。

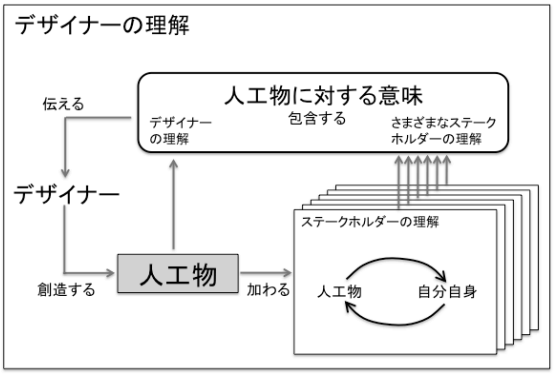
意味とは、以下のように規定されます。

- ✓ 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。

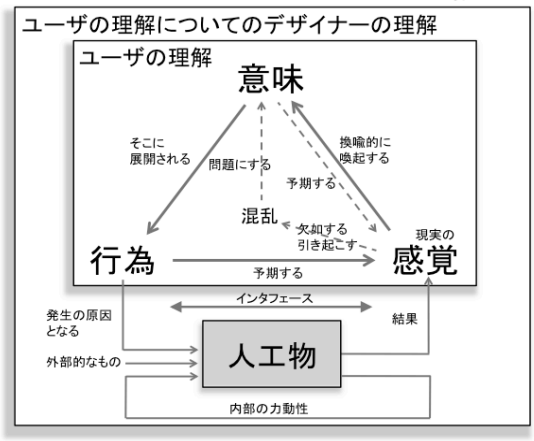
図で「外部の世界」というのが、ソフトウェアの実行による作用によって影響を受ける実世界です。「意味」そのものは、直接語ることはできません。人間が持つ意味は、実世界に対する「(言語) 行為」として現れ、その行為による因果関係によってもたらされる現象が、「感覚」として帰ってきます。

人工物をデザインするには、この枠組みに従って、２次的理解を定義します。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ます。

二次的理解としてのデザイン意味論



デザイナがユーザのインタフェースについて持つ理解

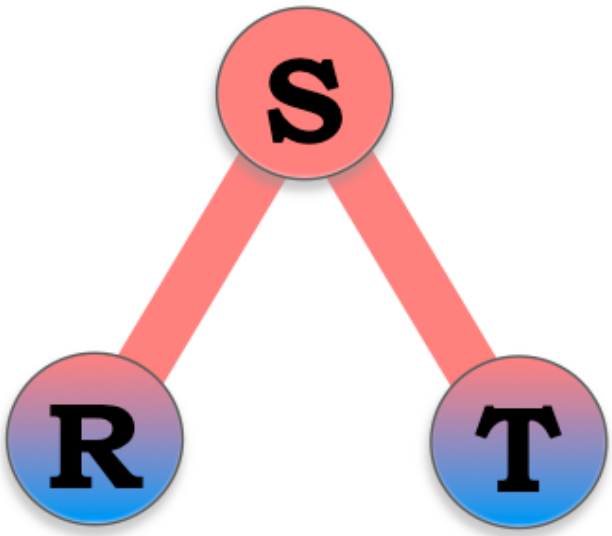


クリッペンドルフが提唱している、人工物（ソフトウェア以外も含む）に対するデザイン原則は、以下の通りです。

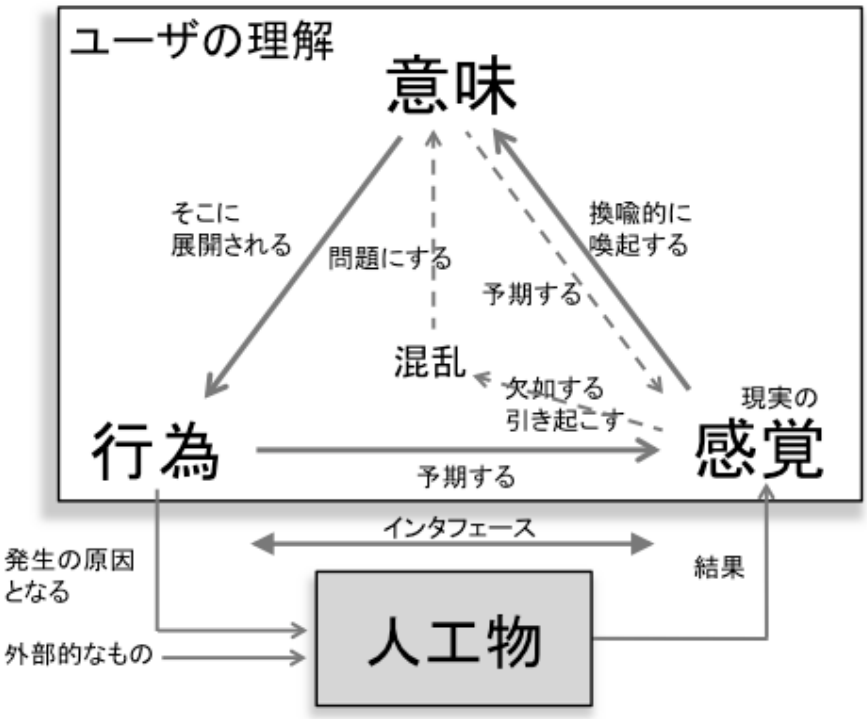
- ✓ 人間中心性：技術の概念化、理解と使用を意味の前提とする。
- ✓ 意味のあるインタフェース：認識→探求→信頼の移行を促進。
- ✓ 二次的理解：ユーザの概念モデル、シナリオを知る。
- ✓ アフォーダンス：人工物の使用における意味の基本単位。
- ✓ 制約：ユーザの注意や相互作用を方向付ける意味的な方法が望ましい。
- ✓ フィードバック：ユーザ行為の即時的、直接的フィードバックが望ましい。
- ✓ 一貫性：意味の層を定義。
- ✓ 学習可能性：ユーザが学習していくことをデザインする。
- ✓ 多感覚の冗長性：視覚、聴覚、触覚等複合的なもの。人による感覚の差。
- ✓ 変動性ー多様性：人工物の変動性は、ユーザの集団における多様性と一致。
- ✓ ロバスト性：障害や事故回避。
- ✓ デザインの委託：ユーザがデザインすること

Ⅲ. Λ∨モデルの概要

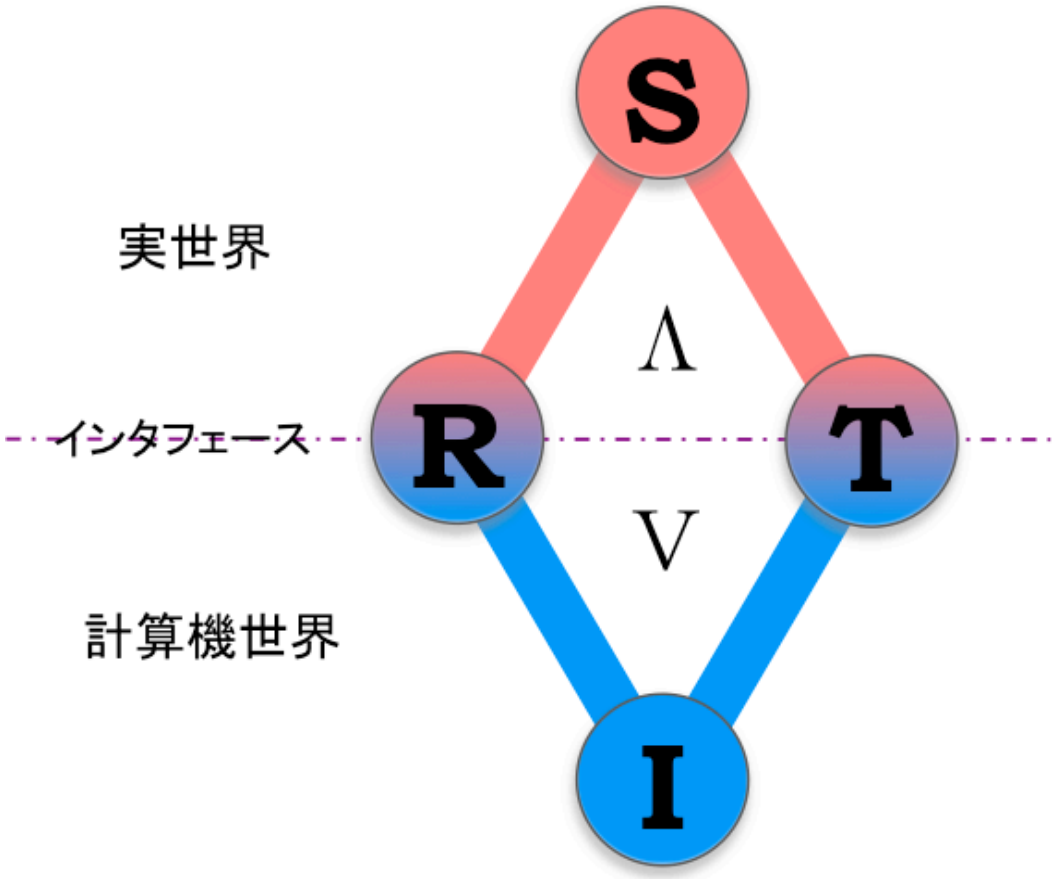
前節の意味／行為／感覚の三角形に、ソフトウェアの要求とテスト（実行の反応）を当てはめてみることにします。すなわち、S は意味（Semantics）、R は要求（Requirements）、T はテスト（Test）を表わしています。∨字に対峙して、Λ字モデルとでも言えるでしょう。



Λ字モデルでは、人工物としてのソフトウェアが実行することによる、認識の変化・修正を扱うことができます。（T は Test というより Track や Trace の T とした方がよいかもしれません。）



第Ⅰ節のV字モデルと統合すると、以下のようになります。



これがΛVモデルです。

（1） そもそもRなんて判らない（語れない）かもしれない。

（2） でもRを満たすIを作る（既にあるものかもしれない）
「実行」できるのはIだけ。

（3） だからIを実行させてTをやる。
IがRを満たしていない場合には、Iの実現方法に誤りがあることになる。
（これは、従来の伝統的な内部テストの問題）

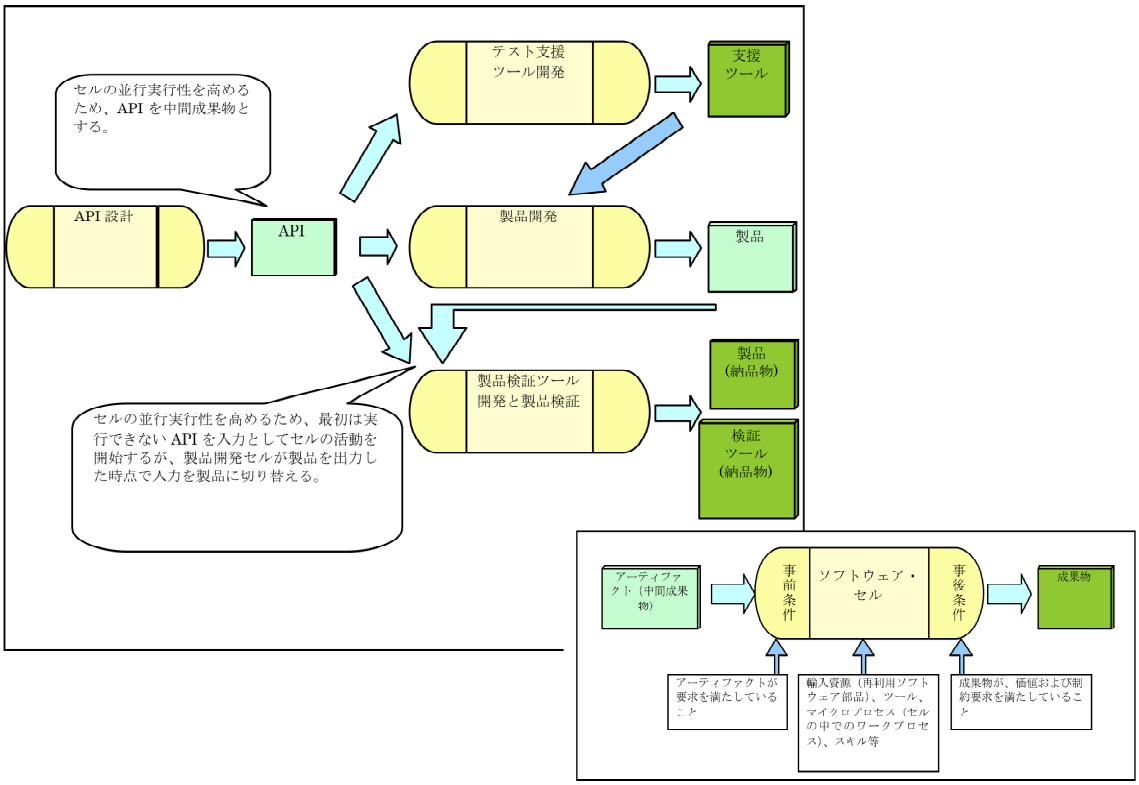
（4） Tという作用を受けて＜S 意味（認識）が変わってR（要求）が変わる＞ことがある。
「想い」を発する根元であるS（意味：これは語り得ないものです）があって、これに基づいてR（要求：これは一般的にはある程度語ることができます）が発せられ、T（テストや運用）からフィードバックを得て、S が修正され、新しいRが出るといったプロセスを繰り返す。

Ⅳ．いくつかのケーススタディ

以降、４つの事例をΛＶモデルのケーススタディとして考察してみます。

ケース１：伝統領域でのソフトウェア開発会社

- ✓ 中堅の独立系ソフトウェア開発会社の事業部
- ✓ 数十人月～百人月程度の中規模案件の請負受注が多い。
- ✓ 2005 年頃よりアジャイルプロセスの味見（なんちゃってアジャイル）
- ✓ 2007 年頃よりアジャイル・ソフトウェアセル生産方式を確立

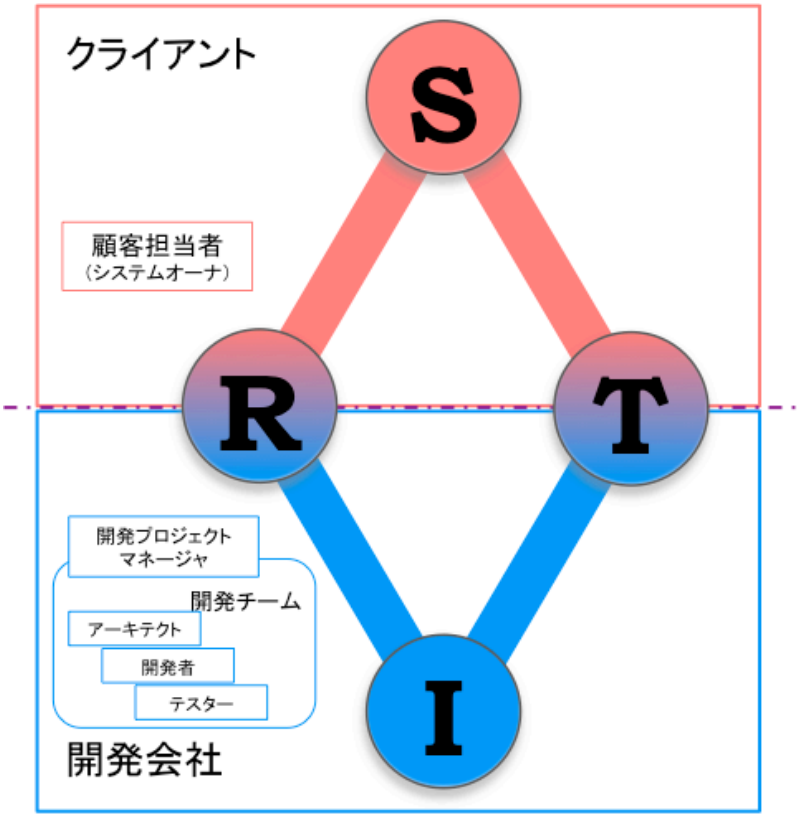


- ✓ 日常的に、タスクカード（ソフトかんばん）、バーンダウンチャート、朝会、ふりかえり等のプラクティスを行っている。
- ✓ 新人は定常的に採用しており、アジャイルプロセスの実践チームは、人材育成のために研修的に参加することもある。
- ✓ 進化型のフレームワーク（Groovy/Grails など）やプロセスの構築も継続的に行っている。

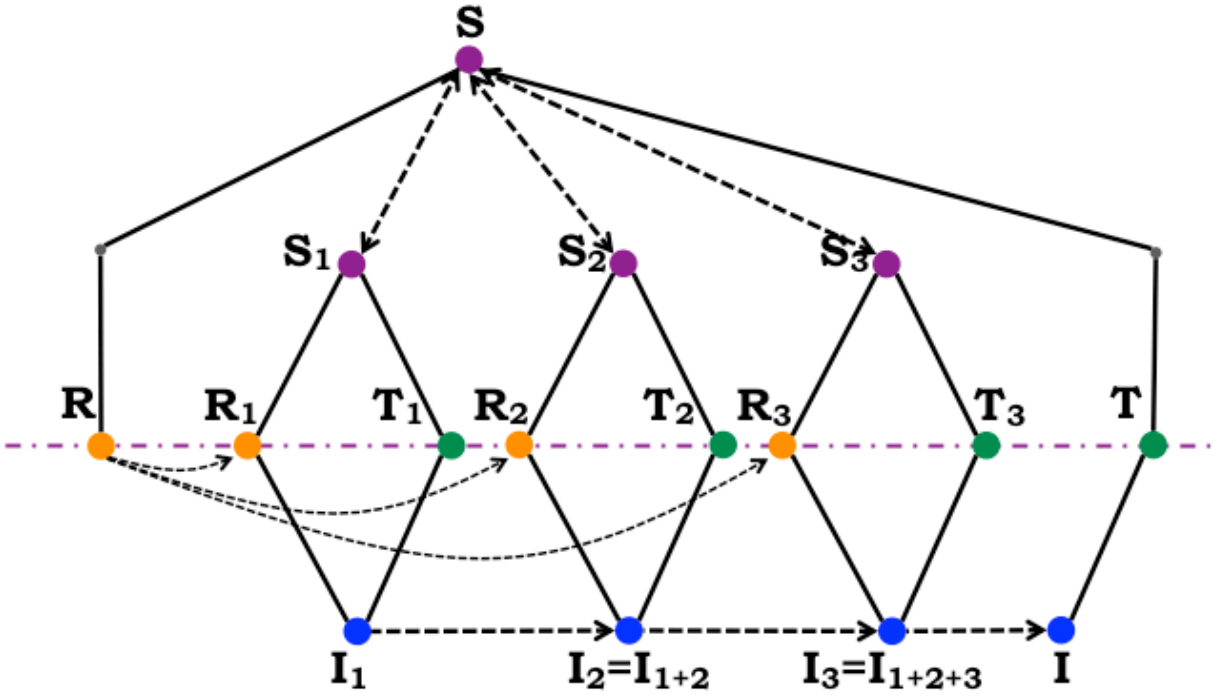
〔プロジェクト例〕

- ✓ 請負契約による約１年程度の受注
- ✓ 顧客から要件定義書が提示。要件定義の曖昧や不備などがあるが、基本的に変更はない。
- ✓ ソフトウェアは、実験などの測定データを管理する目的で使用され、顧客側のビジネスに直結するものではない。
- ✓ 開発チームは１０名程度。内５名がエキスパート、残りが新人。
- ✓ 開発プロセスは、インクリメンタル（段階的拡充）型開発プロセスとし、１～２ヶ月程度のイテレーションを繰り返すものとする。
- ✓ イテレーションごとに顧客側の確認をとり、要件の変更もある。従って、安定しているもの、重要なものから実装していくこととする。
- ✓ 実装環境は、Java, Groovy, Struts, Spring, Hibernate など。デザインパターンやアスペクト指向設計なども行う。
- ✓ テストは、イテレーションごとに回帰テストを行う。

顧客と開発企業との間の組織境界は、Λ∨モデルにおける実世界と計算機世界との境界と一致しています。従来の典型的な請負受発注と考えられます。



時系列でプロセスを俯瞰してみると、以下のようになります。



ソフトウェアの価値は、クライアントの世界や意味 S に関わることです。開発企業からは見えません。要求 R は、最初にクライアント側から文書の形で提示されます。これを受けて、開発企業では基本設計やそれに続く詳細設計が進められます。インクリメンタル型の開発プロセスをとるために、最初の段階で、全体の要求に基づいてイテレーションの分割を行い、各段階での要求を明確にします。

それぞれのイテレーションは、決められた要求 R_n（n=1～3）に対し、実現 I_n を開発します。R_n を意味付けている意味 S_n は、おそらくこのソフトウェアに関わる全体の意味 S の一部です。テスト T_n は、実現 I_n が要求 R_n を満たしていることを確認する行為です。I_n の実行による認識の変更も発生し、これは、次のイテレーションの要求 R_{n+1} に反映されます。

イテレーション n とイテレーション n+1 との関係は、前者の拡充が後者という位置づけです。実現は、I_n を拡充して I_{n+1} とします。T_{n+1} は、T_n のテスト項目を回帰的に適用することになります。

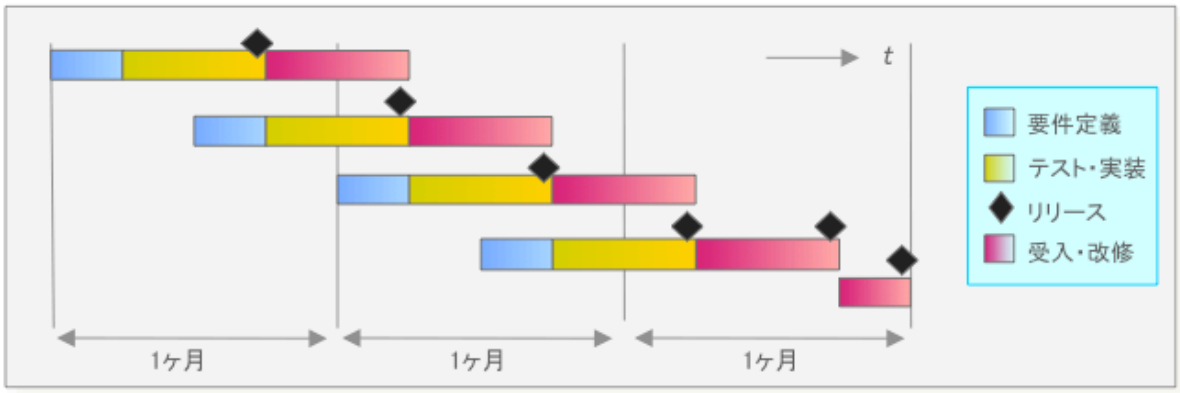
最終イテレーションの結果が、このプロジェクト全体の結果となります。

ケース２：Web サービス領域のソフトウェア開発会社

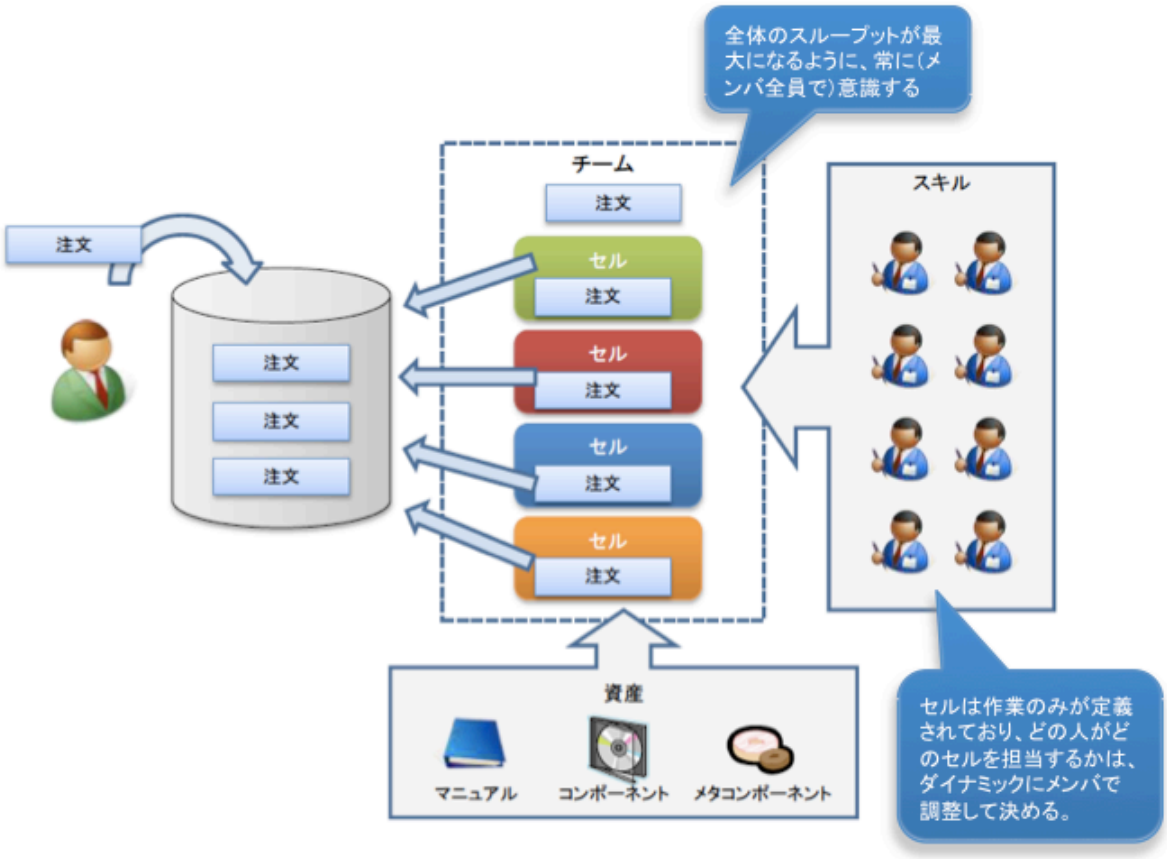
- ✓ アジャイルプロセスを積極的に取入れている小規模のソフトウェア開発会社
- ✓ Web システム開発の分野を得意としている。
- ✓ 独自のアジャイル・ソフトウェアセル生産方式を確立している。
- ✓ アジャイルプロセスを、より組織化し、より大きな案件も開発できるようにしていきたいと考えている。
- ✓ アジャイルプロセスの基本的なプラクティス：朝会、振返り、イテレーション、バーンダウンチャート、週４０時間労働などは全て導入している。

この会社のアジャイル・ソフトウェアセル生産の特徴は、タイムボックス方式と呼ばれる固定間隔（２週間）のイテレーティブな方法を取入れている点です。クライアント側から見ると、定期的にリリースを受入れ、検収を行わなくてはならないことになります。イテレーションを継続していき、先々の予想は困難ですから、契約は、本来、保守契約のようなものが適しているものと考えられます。

- ❖ 母体の開発は1ヶ月程度で完了させる
- ❖ 実装期間を2週間に固定する
- ❖ 情報システムのライフサイクルが尽きるまで継続される
- ❖ 2週間でテスト・実装期間が全て完了していなければならない
- ❖ 2週間毎に必ずユーザへリリースし受入検収を受ける
- ❖ 要件定義と受入検収とテスト・実装の期間は重なる

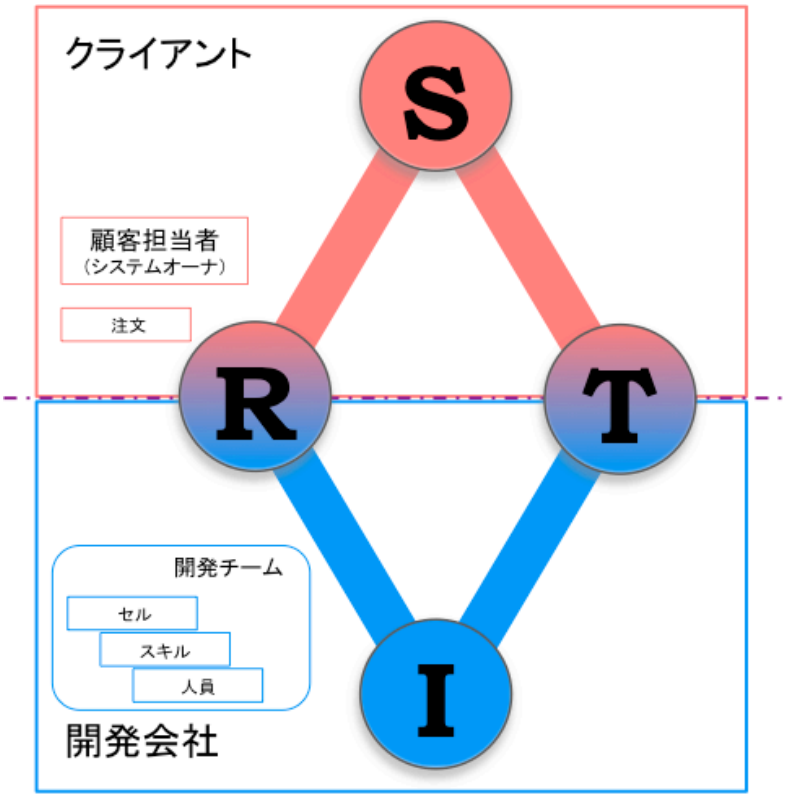


もう一つの特徴が、「セル」の導入です。これは所定の作業をこなす仮想的な人格と見なすことができます。「人月からの脱却」という観点では、仕事の工数を「セル・月」で把握するということになります。このように作業（役割）と人的リソースとを切り離すことは、マネジメント上、多くのメリットを得ることができます。

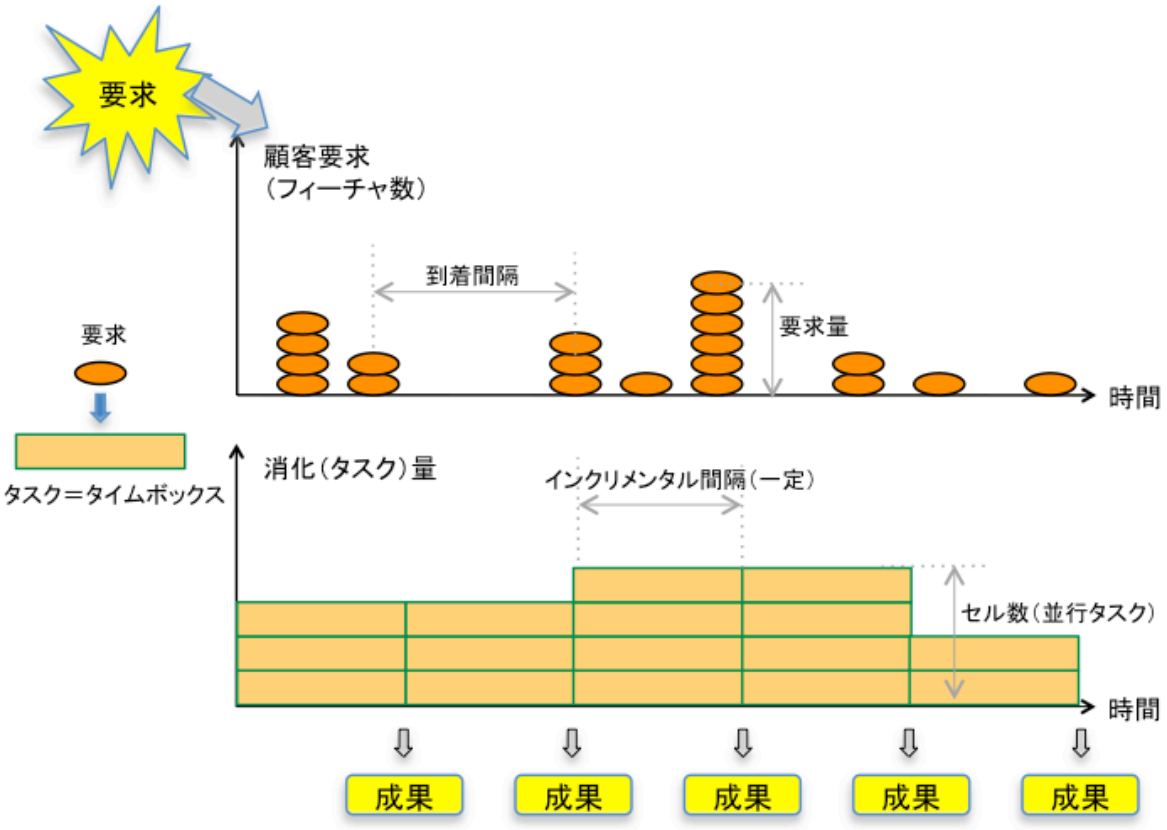


（1）クライアント側に対して人員（人月）を隠蔽することができる。
（2）セルはクライアントからの注文に直接対応しているため進捗が可視化できる。
（3）セルは並行に実行できるため段取りに気を使わなくてよい。
（4）チームメンバは、セルのスループットを向上させることのみに注力すればよく、上位のマネジメントに無駄がなく軽量化できる。
（5）セルの進捗を見てダイナミックに人の割当もチーム内で調整しながら進めることができる。

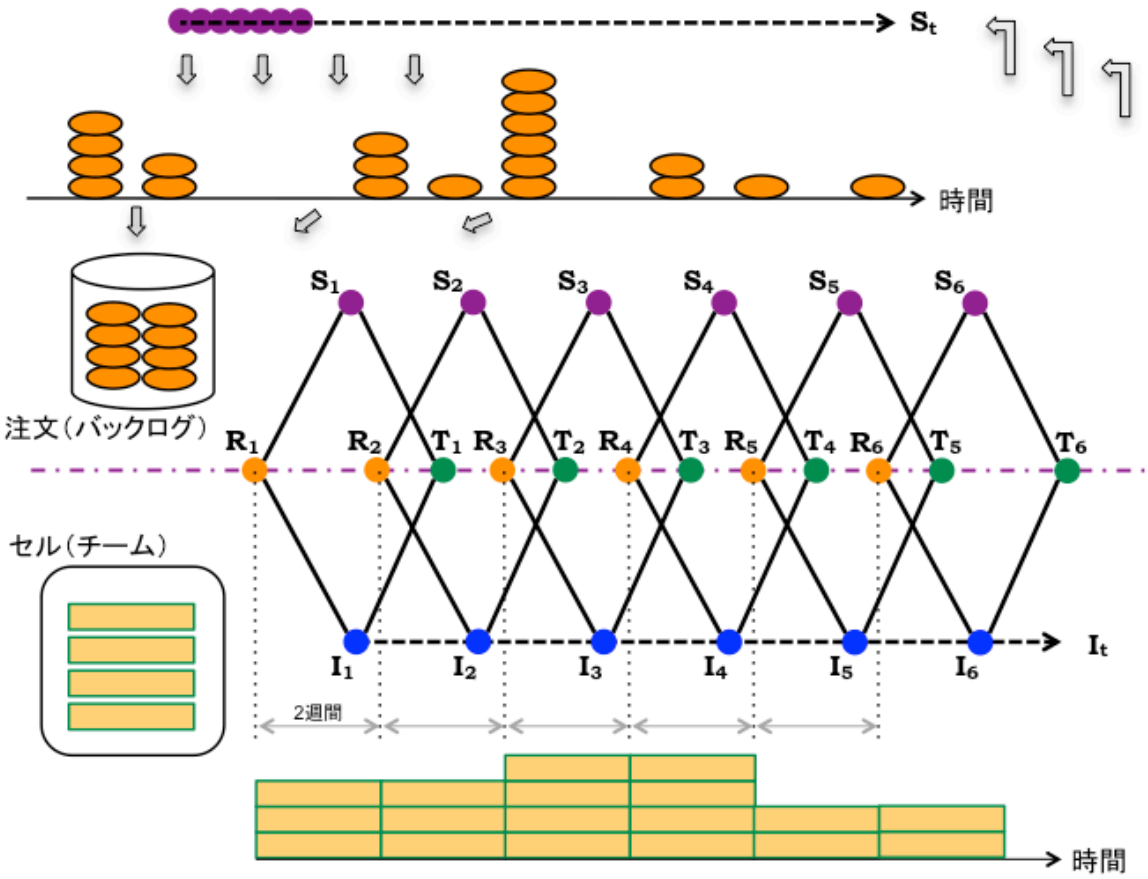
顧客と開発企業との間の組織境界は、Λ∨モデルにおける実世界と計算機世界との境界と一致しています。従来の典型的な請負受発注と同様です。



クライアントからの要求は、いつどれくらいでてくるかは不確実です。**アジャイルプロセスとは、明確になった要求（仕様）を、明確になった時点で速やかに実現するということ**です。



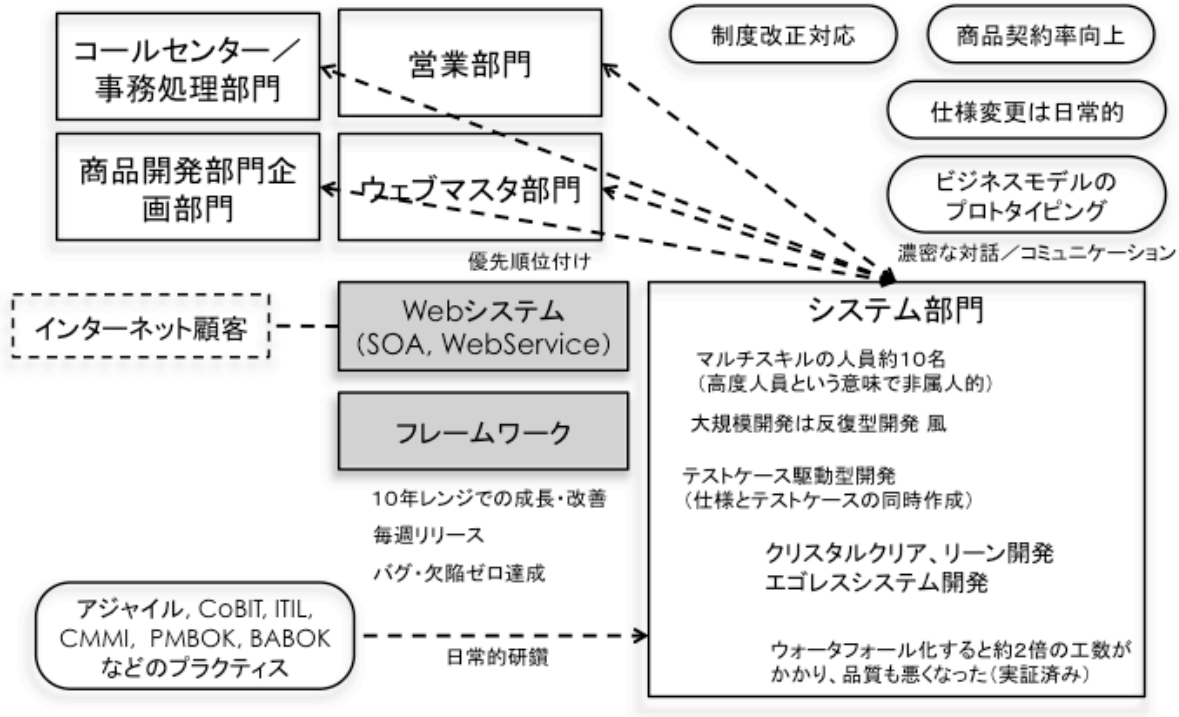
タイムボックスマネジメントによる、ダイナミックな要求とセルによるタスク消化の状況は、以下のように表わすことができます。



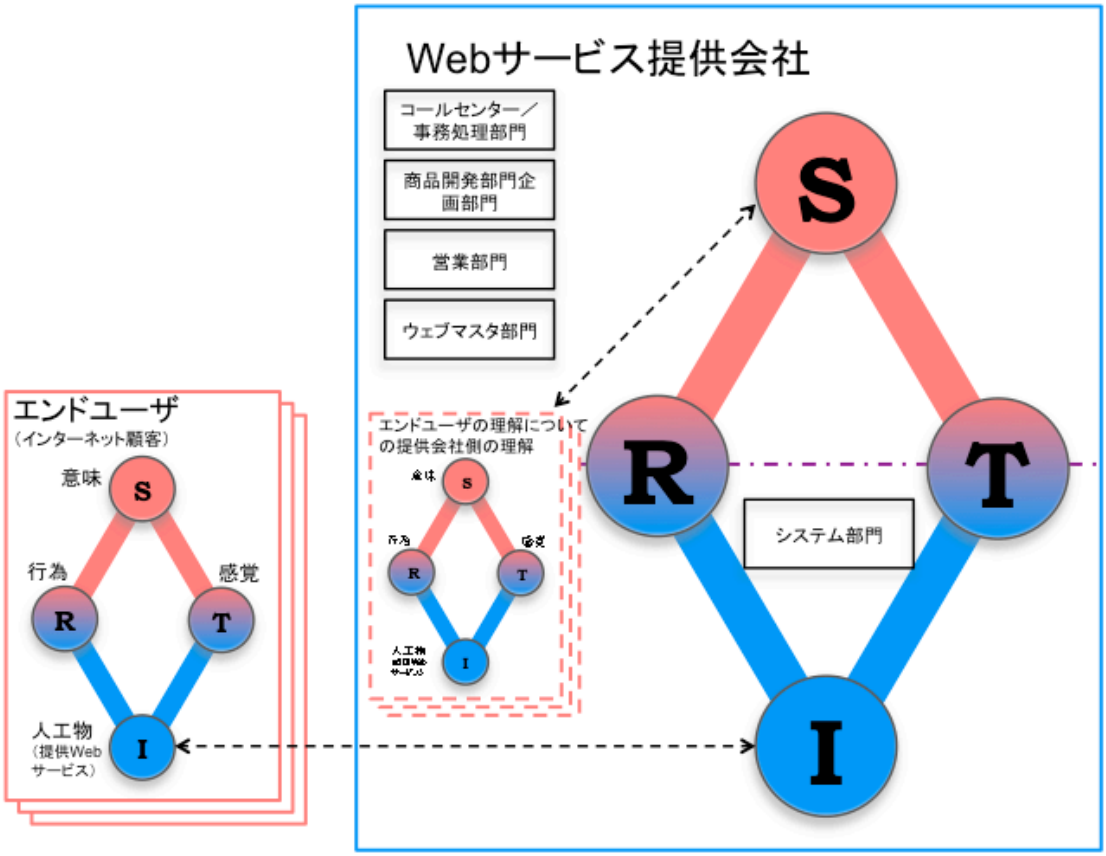
クライアント側からの要求は、時々刻々と不確実に出されます。これを注文バックログとして溜めます。これが、クライアントと開発企業との間のバッファとなります。注文は、開発チーム側でセルに割当てられ消化されていきます。注文の量は、波があると考えられ、多くなりそうな時期にはセルをたくさん用意して準備しておきます。

ケース３：ユーザ企業内製のシステム部門

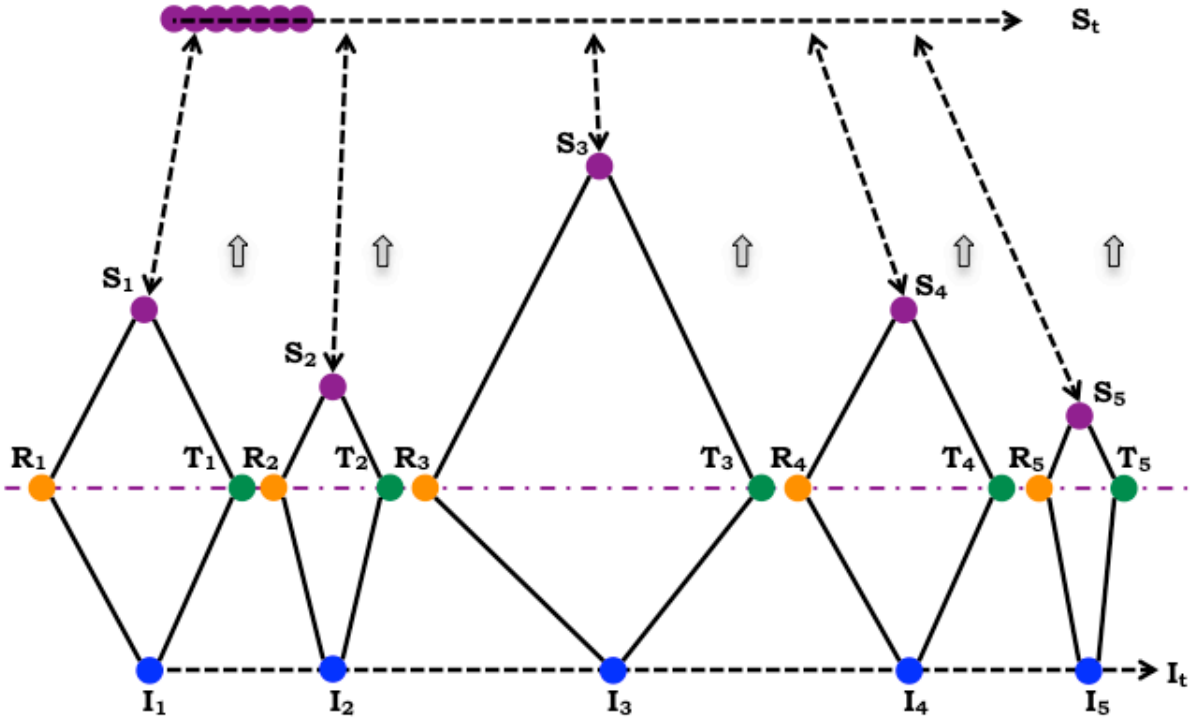
- ✓ インテーネットからアクセスする顧客に対して商品案内から契約に至る支援をする Web システムを継続的に構築・維持している。
- ✓ ユーザ企業内製型のプロセスである。
- ✓ 営業、業務部門などとのコミュニケーションを密にして、常時、新規フィーチャ、改善要求について検討している。
- ✓ プロジェクトという概念は無く、システム部門の約１０名の体制で、週１回のリリースを繰り返す方法を採っている。



- ✓ 開発チームは、長い年月をかけて成長・改善をしてきており、レベルが高い。
  - ・文献や書籍の読込みと、徹底した討論をすることによって、チームに導入すべきものを吟味している。
  - ・システム開発の負荷が少ない時に、育成対象の人材に責任を持たせて業務を遂行することによって、業務ノウハウの伝承を行っている。
- ✓ エゴレスシステム開発を実践し、チームメンバ全員がどの部分、どういった開発も可能なようにして（チーム内の）非属人化を図っている。

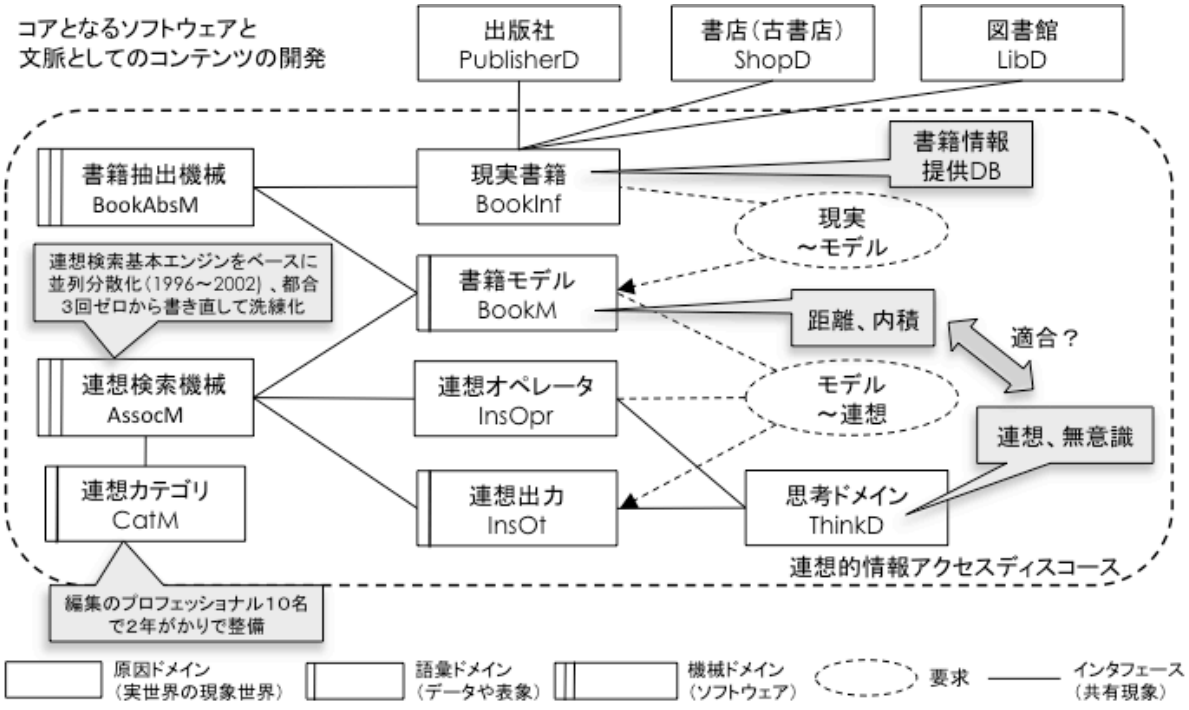


エンドユーザの提供 Web サービスに対する意味、文脈、理解（誤解）などの状況は、営業部門などとシステム部門とがコミュニケーションを密にとることによって、ニーズの把握、既存システムに対する改善項目など、常に的確に把握できるようにしています。すなわち、システムの意味が時間とともに変化していきます。



ケース４：新規領域開拓型研究機関

- ✓ 「連想検索」と呼ばれる人間の創造的活動を支援するための新しい情報アクセス方式を核とした活動
- ✓ 基礎研究によって得られた「連想検索」の方式に基づくエンジンを開発
  これは３回ゼロからコードを書き直す方法で、順次、インタフェースやアルゴリズムを洗練化（約６年程度かけている）。
- ✓ 連想検索エンジンを活用した Web サイトの構築をプロデュース。
  （書店、図書館、古書店など）
- ✓ 連想を効果的に、価値あるものにするために、書籍のカテゴリ分類を行う。
  これは出版社の編集長レベルの人に２年がかりで作成してもらった。
- ✓ 連想検索の効果を実証できたため、複数のサイトを連携して検索を行う仕組みを作り、実現した。

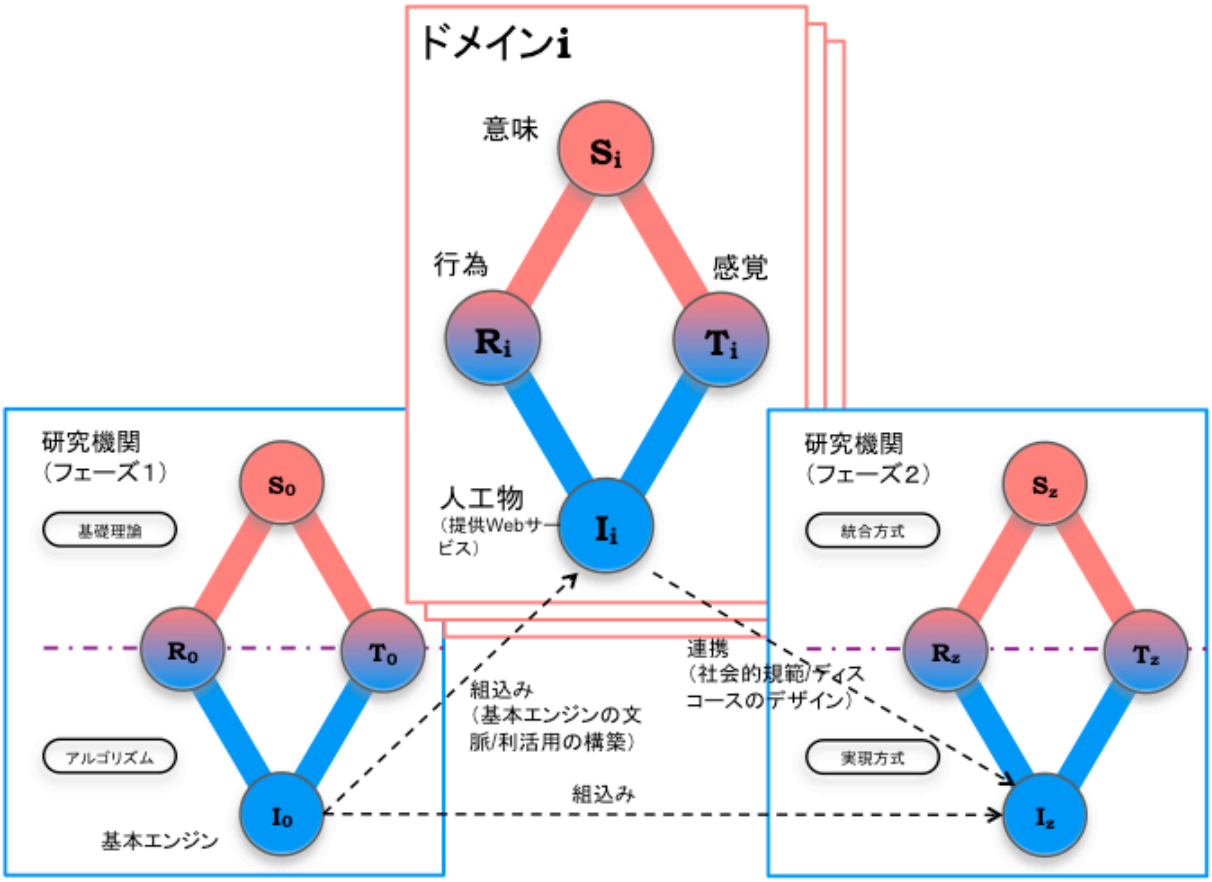


基本エンジンを開発するところは、従来の開発プロセスと考えてよいでしょう。仕様や方式をしっかりと固めてから、コードを書くというものです（次図左下部分）。次の各ドメインの Web サイトの構築は、基本エンジンを活用しながら、それぞれの分野の知識やノウハウをコンテンツとして埋込んでいく作業です。古本屋のおやじさんがどういった仕事をしているのか、図書館の司書さんの仕事、分類の仕方はどうなっているのかといったことが解明されていきます（次図中央部分）。ここで

重要なことは、システムに埋込むべき知識が、システム外から来ることです。その中でキーになるものが、出版物のカテゴリ情報です。これを、ゼロからの発想で、出版社の編集長に参加して作ったというところが、連想検索の価値を高めています。そもそも、人間が創造的活動をして検索を行う場合、何か目的があって探し物をするわけではなく（そういう場合には Google のキーワード検索を使えばよい）、何か関連しているもの、想いもよらないものなどを発想の刺激として出してほしいわけですから、出版社の編集長のような横断的に見る力というのが組込まれていることがよい結果を生みそうだと直感的に思えます。

最後のサイトを横断した連想検索を行う統合部分については、それぞれのドメインでの結果を、他の検索に活かす方式を、分散検索の方式を考え実現したものです。これは、ソフトとしては Ajax を使って、小さな規模（数ヶ月）で実装できるものです（次図右下部分）。

所謂、ソフトウェア開発という観点からは、受発注の関係や、要求をどうするかといった観点はさほど重要ではなく、人間の発想とはどういうものかとか、今までにないディスコース（言説）をデザインしていく活動で、まさに「知働的」なものと言えます。



考察とまとめ

ケーススタディのまとめ

前節のケーススタディの４つの例は、アジャイルプロセス、あるいは、知働化のいくつかの典型的なものと言えるでしょう。

ケース１：伝統領域でのソフトウェア開発会社

「狭義のアジャイルプロセス」を実践しつつ、「セル」を定義することによって、作業の前提条件、終了条件、制約、リソースなどの管理を行い、かつ、セルの並行化によって、タイムリーな開発も可能にしています。

伝統的なユーザ／ベンダの受発注、請負開発の中で、アジャイルに対応していくベースラインを確保している好例です。

ケース２：Web サービス領域のソフトウェア開発会社

「広義のアジャイルプロセス」を実践しつつ、タイムボックスマネジメントとセルの定義、さらに、セルと人的リソースの分離も行って、システムのライフサイクルを継続的に支援していく仕組みを確立しています。

Web システムや比較的小規模案件に特化しているとは言え、伝統的なプロセスへの配慮、組織化や仕組みの確立を目指しており、アジャイルプロセスによる不確実性への対応を実践している好例です。

ケース３：ユーザ企業内製のシステム部門

ユーザ企業の Web システムを構築・維持していく内製型の開発チームです。エンドユーザの利用アクセスや商品販売の誘導などの状況を、営業や業務部門との緊密な連携によって対応しています。

アジャイルプロセス、エゴレスシステム開発、さらには、ITIL や Cobit なども含め、自ら勉強し、検討し、議論し、実践する高度な研鑽によって、開発の質を維持している好例です。

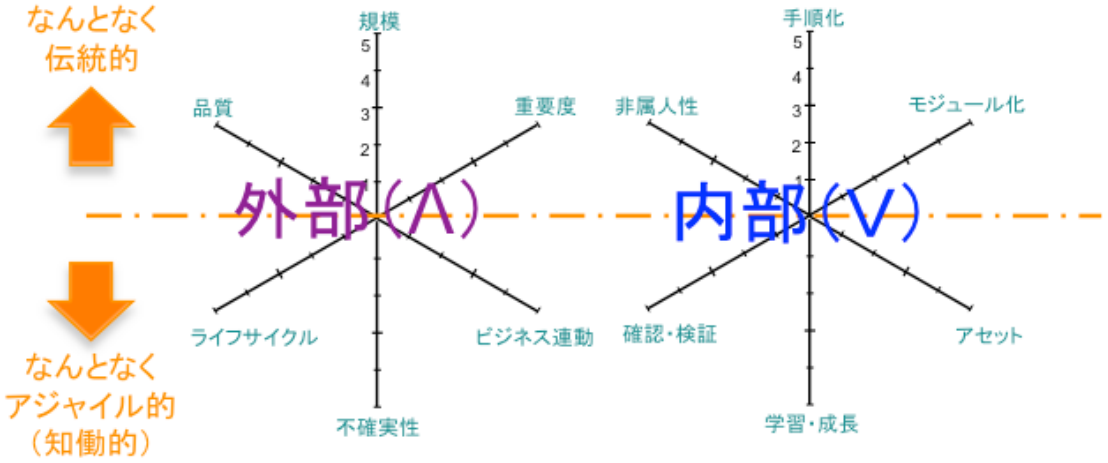
ケース４：新規領域開拓型研究機関

基礎研究から生まれた理論をソフトウェアのエンジンとして開発し、さらに、それを活用した社会的な仕組みや規範をデザインし、プロデュースも行っています。従来のソフト開発という範疇からは外れますが、知識をどのようにシス

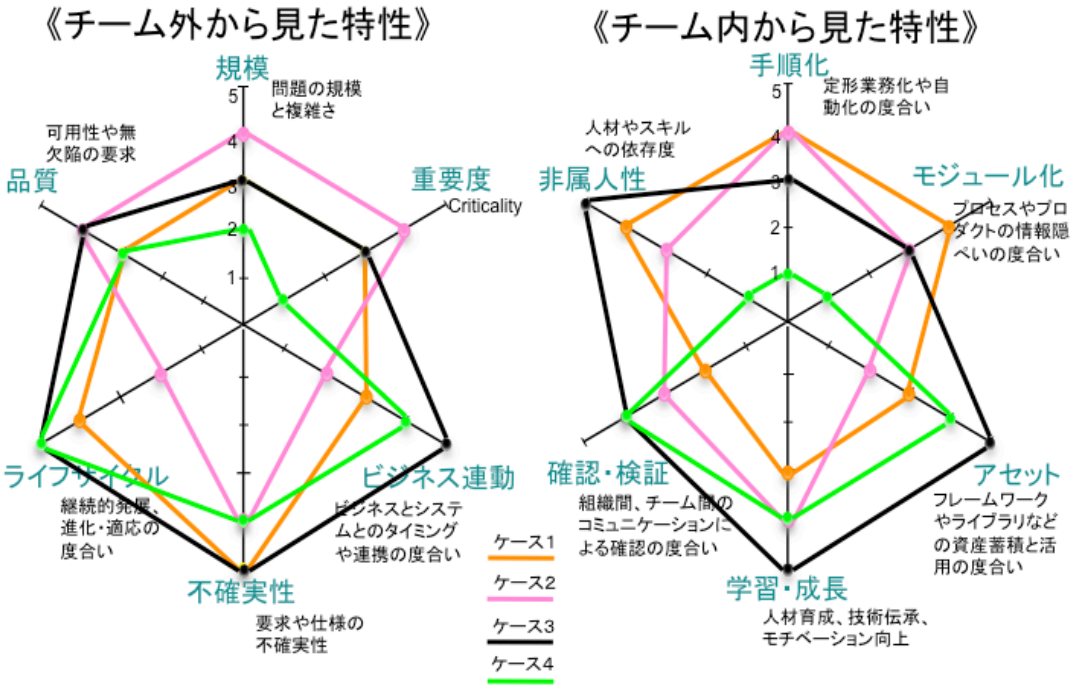
テムに組込んでディスコースをデザインしていくかという知働的な考えの好例です。

プロセスの特徴付け

これ等のケースを、強いて整理してみるとすれば、いくつかの軸によって特徴づけることができます。



左側のレーダチャートが、Λすなわち、実世界から見える特性です。もし、実世界が開発チームの外部であるということでしたら（組織境界に依存しますが）、Λ世界を「外部」と呼んでもかまわないでしょう。逆に、右側のレーダチャートはV、すなわち、開発チーム内部の特性です。



なんちゃってマニフェスト

本小論の最後に、筆者の主張を簡単なマニフェストの形で提示しておこうと思います。ここで示したものは、新しい「パラダイム」による、新しいものの見方です。伝統的な領域にとどまって、伝統的なものの見方をしていては、単なる＜改善＞に終わってしまいます。今、必要なのは＜改革＞なのです。

■工業的生産　より　実行可能な人工物のデザイン

ソフトウェアとは、実行可能な＜人工物（アーティファクト）＞です。決まりきったことを、手順化や自動化を行い、大量生産するという工業製品の＜製造＞とは異なります。ソフトウェアには製造はありません、創造的、知的活動としての＜デザイン＞が中心でなくてはなりません。デザインの対象となるソフトウェアが、一般の人工物と異なる点は、＜実行可能＞であることです。計算し、情報を処理し、実世界の現象に作用を及ぼす＜実行＞する人工物であることが本質です。

■コード　より　知識の織り込み

ソフトウェアづくりとは、知識をソフトウェアに織り込むことです。コードを書き、実行可能なコードをつくることも大切ですが、そのコードには、計算機世界の知識だけではなく、実世界や人間の知識を織り込んでやらなければ、価値はありません。織り込むべき知識がどこから来るのかということは、とても重要な観点です。クライアントからの要求や要件定義に従って、開発企業がソフトウェアを開発するという場合には、多くの知識は、クライアント側の奥深い業務領域から伝言ゲームのようにしてやってきます。多くの知識は、∧∨モデルの∧の領域に隠れていることでしょう。

■初期構築　より　ライフサイクルとフィードバック

ソフトウェアの意味は、その＜実行＞によって継続的に変化します。社会状況や世界観も変わりますし、ソフトウェアを実行させることが、利用者や関与者の認識を変えてしまうものです。従って、ソフトウェアは時間とともに変わる状況の中で生き続けなくてはなりません。伝統的な領域の概念で言えば、保守・運用をし続けるライフサイクルの観点がソフトウェアづくりの中心です。ソフトウェアは実世界に投入され、実世界の現象に影響を及ぼし、世界や人間の認識を変えてしまいます。

その変化した世界や認識を、ソフトウェアづくりにフィードバックし続ける仕組みを考えていく必要があります。

■契約　より　社会的様相

二者間対峙（ユーザ／ベンダ、企業／雇用者など）の契約の改善の余地はまだまだあるでしょう。制度の問題というのは、最終的には整備しなくてはならないものです。ソフトウェアづくりに関わる組織や企業間の関係は、もっと複合的なものになっていくでしょう。織り込むべき知識の源泉、仕様化、実現、ライフサイクル維持など多くの活動が多くの組織とともに関わってきますし、組織間の関係もダイナミックに変化していきます。こういった広い意味でのソフトウェアを取巻く社会的な＜様相（texture）＞をデザインしていくことが重要です。また、全てが金銭的取引きに帰着できるわけでもありません。コミュニティ活動や地産地消的な事柄も、社会的様相として捉えていく必要があります。

執筆後記と参考資料

この小論を執筆しようと思った理由は、昨年末（2009 年 12 月）のアジャイルプロセス協議会/知働化研究会で「知働化研究誌」の企画の話があって、春先までに何か書かなくてはならなくなったことと、SEC（Software Engineering Center）の非ウォータフォール研究会のメーリングリスト上でV字モデルに関するやりとりをした際に、少し頭の整理をするために自分なりの見解をまとめておこうと考えたことにあります。

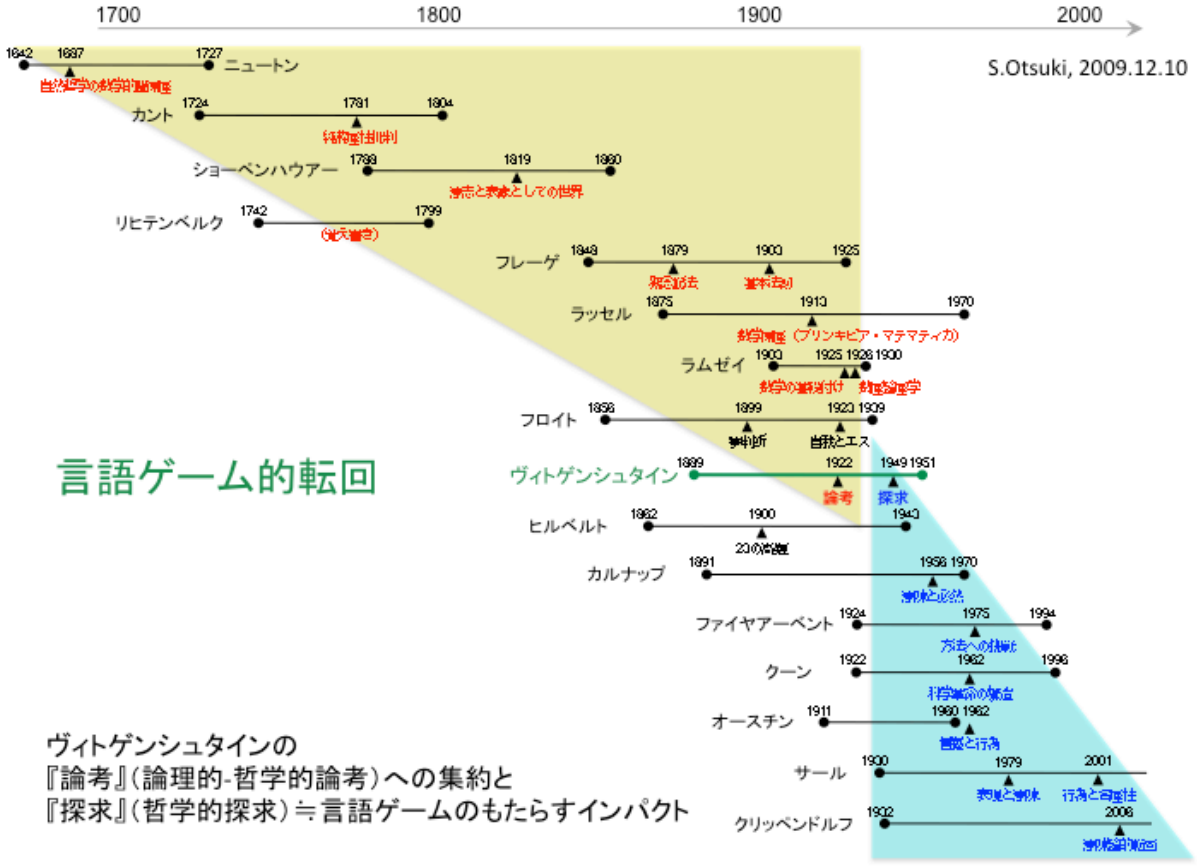
V字モデルは、1980 年代初頭のライフサイクル論争の頃に、ウォータフォール型開発プロセスの欠陥を説明するために、設計とテストとの対応を整理するために提唱されたもので、現在でも SLCP の標準の中で、さまざまな開発プロセスを説明するために使われています。このモデルの本質は、要件定義や設計の正しさを確かめるテストという活動が＜実行＞するプログラム実体を前提としているということです。それ以上でも、それ以下でもないのですが、多くの誤解が生じていることも確かです。

アジャイルプロセスや知働化の検討では、あえて、V字モデルで扱う伝統的な領域とは一線を画し、その領域の言葉も使わずに、新しいパラダイムであるという立場を貫くようにしてきましたが、本小論では伝統的領域からの視点の差異を明確にするアプローチを採ってみることにしました。ソフトウェアやシステムが置かれている＜実世界＞や＜様相＞に目を向けるというのが、新しいパラダイムの第１歩です。ですから、V字の上のΛ字を描いてみました。「超上流」（この言葉は好きになれませんが）のもっと外側の世界あるよということですし、その世界もどんどん変化していくし、かつ、人によっても認識が異なる主観的なものです。

「パラダイム」という言葉が大げさな感じもするので、本小論の副題にあるように「意味論的転回」としてみました。第Ⅱ章の「知働化への転回」では、クリッペンドルフのデザイン論を紹介していますが、そこで使われている用語です。

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、とても重要だと私は考えています。無論、無意識や心理的な事項もあることは認めますが、最終的には言語として現れることになります。こういった哲学的な基礎付けをしたのは、20 世紀初頭に活躍した奇才、ヴィトゲンシュタインです。彼の後期哲学を『探求』とか『言語ゲーム』の哲学と称しますが、筆者はこの思想は、ソフトウェアの世界にも当てはまると考えています。昨年秋にクリッペンドルフの書籍を知り

ましたが、まさに我が意を得たりという感覚でした。



一言で私の主張を集約するならば、**「ソフトウェアエンジニアリングの世界で言語ゲーム的転回を進めよう」**ということです。新しい世界を描く努力なくして、未来は切り開けません。

〔参考図書〕

・黒崎宏, 言語ゲーム一元論：後期ウィトゲンシュタインの帰結,勁草書房,1997.12
・黒崎宏, 科学と人間：ウィトゲンシュタイン的アプローチ, 勁草書房, 1977.10
・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
・山本信 黒崎宏編, ウィトゲンシュタイン小辞典, 大修館書店,1987.7
・橋爪大三郎, はじめての言語ゲーム, 講談社現代新書, 2009.7
・クラウス・クリッペンドルフ, 意味論的転回：デザインの新しい基礎理論, SIB アクセス発行, 2009.4.1

〔筆者の知働化関連の書き物〕

（http://www.exekt-lab.org/　にアップロードしてあります。）

・アジャイル・ソフトウェア・セル生産：人月から価値駆動へ（大槻繁,濱勝巳），PM Conference 2008, 2008.8.1
・人働説から知働説へ（対談：知働化研究会始動（山田正樹−大槻繁））, 2009.7.15
・知働化への接近，知働化研究会第２回会合, 2009.10.15
・クリッペンドルフの『意味論的転回』読書感想, 2009.10.15
・ビジネス駆動の先進的な IT システム・ソフトウェアの世界観：知働化：不確実性への対応から進化・適応プロセスによるソフトウェアづくり，IPA Forum 2009 / ソフトウェア・エンジニアリングセッション, 2009.10.29
・知働化の世界観：不確実性への対応から進化・適応プロセスによるソフトウェアづくり, 2009.11.3
・１０８の質問に対する解答（デカルト vs ヴィトゲンシュタイン）, 2009.11.4
・言語ゲーム的転回の年表，知働化研究会第３回会合, 2009.12.7

本小論は、「知働化研究会」の自由研究の一環として作成したものです。2010 年春に発行予定の知働化研究誌に掲載予定の原稿案も兼ねています。ケーススタディで採上げた４つのケースについて、情報提供、議論いただいた方々に心より御礼申し上げます（諸般の事情を配慮し、本小論でお名前を掲載するのは避けておきます)。皆様との交流を通じて、多くの刺激をいただき、「知働化の世界観」の確立をめざして、引き続き研究を進めようと思います。ご協力、ご支援の程、よろしくお願いいたします。

2010 年 1 月 1 日（元旦）

大槻 繁

■