非ウォーターフォール型開発 WG
活動報告書

平成 23 年 3 月 31 日

独立行政法人 情報処理推進機構
ソフトウェア・エンジニアリング・センター

目次

第二部　アジャイル開発のモデル

第三部　非ウォーターフォール型開発における技術及びスキル

第四部　非ウォーターフォール型開発にふさわしい契約

第五部　アジャイルにまつわる開発手法体系の歴史と動向

はじめに

システム構築やソフトウェア開発においては、対象のシステムやソフトウェア、及びそれらが使用される社会やビジネス環境の特徴（ここでは、これらを“コンテキスト”と称する）に応じて最もふさわしい形態で行うことが望ましい。独立行政法人情報処理推進機構(IPA)/ソフトウェア・エンジニアリング・センター(SEC)では、コンテキストに適したシステム構築・ソフトウェア開発形態を選択するための考え方を整理しようとする取組みを行っている。

数年前、わが国において、ソフトウェアの不具合を主要因とする重要インフラ情報システム等の障害が頻発し、国民生活や社会経済活動に大きな影響を及ぼしたことは、いまだ記憶に新しい。その後、それらの経験に基づく教訓をもとに、上流工程における品質確保や、ソフトウェアライフサイクルプロセスに基づく開発管理の徹底等の施策が進められた。これらは、いわゆる「ウォーターフォール型」のソフトウェア開発である。その結果が功を奏し、最近ではシステム障害の報道件数が減少傾向にある。

一方、昨今、ビジネスの進展スピードの高速化は著しく、ビジネス環境の変化が激しくなっている。そのため、ビジネスに戦略的に活用されている情報システムに対しても、ビジネス環境の変化に迅速に対応することが一層求められるようになっている。このようなコンテキストにおいては、従来のウォーターフォール型のソフトウェア開発形態では十分に対応できないことが多い。こうした中で、アジャイル開発等、「非ウォーターフォール型」の開発形態が成功を収めている事例も、少なからず報告されている。

ウォーターフォール型開発形態でソフトウェア開発を経験してきた人たちにおいては、非ウォーターフォール型開発形態について十分に理解されているとは言えない。また、わが国のソフトウェア産業の仕組みに照らした非ウォーターフォール型開発の課題もいくつか指摘されている。本書は、ウォーターフォール型開発形態に馴れ親しんだそのような人たちを対象に、非ウォーターフォール型開発形態についての整理された情報を提供することを主目的とする。また、非ウォーターフォール型開発形態に馴染んでいる人たちに対しても、対象ソフトウェアのコンテキストに適しているかを今一度検証してもらうことに使用して頂きたいと考えている。本書が、これらの人たちにとって、対象ソフトウェアのコンテキストに応じた最適な開発形態を選択するためのヒントとなれば幸いである。

本報告書は 5 部から構成される。まず、第一部で、非ウォーターフォール型開発検討におけるアジャイル開発の位置づけを明らかにする。次に、第二部において、本検討の前提とするアジャイル開発のモデルを設定する。その後、非ウォーターフォール型開発における技術及びスキル、非ウォーターフォール型開発にふさわしい契約について、第三部、第四部で、それぞれ検討する。最後に、第五部で、アジャイルにまつわる開発手法体系の歴史と動向について紹介する。

第一部　非ウォーターフォール型開発検討における
アジャイル開発の位置づけ

１　背景と目的

１・１　非ウォーターフォール型開発に向けた検討の背景

ウォーターフォール型でないソフトウェア開発手法、すなわちアジャイル開発など「非ウォーターフォール型」の開発手法は、日本国内のソフトウェア開発においても、Web アプリケーションや Web サービス開発などを中心に広がり、競争力のある製品およびサービス開発、顧客ニーズへの迅速な対応、開発者、技術者のモチベーション向上等に成果を上げている。IPA/SEC では、「非ウォーターフォール型」開発手法の成果の源を分析し、その適用領域や適用方法について整理するための検討に取り組んでいる。
また、この検討の結果として、日本のソフトウェア産業全体が同様の成果を享受できるようになることを期待している。

この非ウォーターフォール型開発の検討は、国内におけるソフトウェア開発の状況として、以下の３つを背景としている。

①ビジネスニーズへの適切な対応
②ウォーターフォール型開発における問題
③ソフトウェア産業構造上の課題

（１）ビジネスニーズへの適切な対応

ビジネスの世界では、他社に先駆けた製品やサービスの市場投入が必須で、それにより徐々に明確となる顧客ニーズを迅速に反映し改善していくことが必要な分野が出現している。また、顧客ニーズは最初に全ては把握できず、把握できたニーズもビジネス環境の激しい変化に伴い変化するが、その状況に迅速な対応が必要となっている。

その結果、情報システム及びそれが実現するサービスに対し、早期サービス提供と効果確認、ニーズ変化への俊敏な対応が求められている。

これらに対応するために、ソフトウェア開発においてもビジネスのスピードに追いつくための開発方式が必要となっている。

（２）　ウォーターフォール型開発における問題

ウォーターフォール型開発では、ソフトウェア開発の初期段階で全ての要求内容を確定することを前提とする。そのため、誤要求や要求の誤解が総合テストの段階で判明すると、手戻りやスケジュール遅延等、多大な影響を引き起こし易い。また、開発期間が長期になる大規模システムでは、外部環境変化の影響で顧客の要求そのものが変化することもあり、ウォーターフォール型開発での対応が難しくなっている。

これらのリスクを軽減するための一手法として、要求確定部分からの順次開発開始と妥当性の早期確認が、昨今、提案されている。

また、ウォーターフォール型開発で発生しやすい手戻りの防止やプロジェクトマネジメント・リスクの早期低減、顧客側と開発側のギャップ解消も、ウォーターフォール型開発の課題として指摘されている。

（3） ソフトウェア産業構造上の課題

日本のソフトウェア産業において、若年世代を中心としたソフトウェア開発技術者の元気がなく、プロジェクトへの参画意識や達成感が低いということが問題として指摘されている[1]。

また、ソフトウェア産業は多重下請構造に支えられており、その産業構造は大きな問題と指摘されている[2]。

これらの問題を解決するためには、開発の過程と各開発技術者の役割や成果を可視化する工夫が必要であり、非ウォーターフォール型開発を取り入れることにより、創造的な開発スタイルや開発者技術者のモチベーション向上、多重下請構造への対処が実現できるのではないかと期待されている。

１．２　非ウォーターフォール型開発に向けた検討の目的

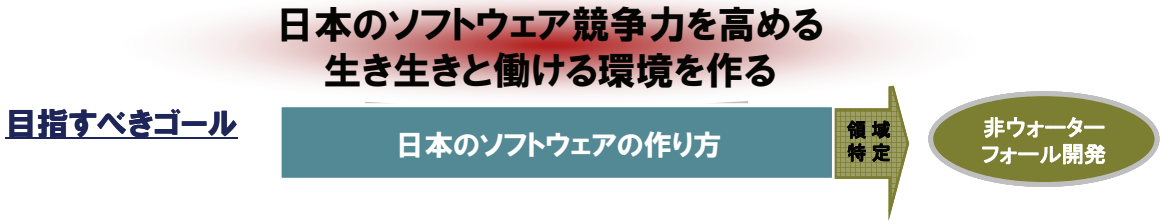
情報システムの開発を大別すると、要求が変化しないことを前提としたウォーターフォール型開発と、要求が日々変化することを前提とした非ウォーターフォール型開発の二つタイプがあると考えられる。

ウォーターフォール型開発は、高信頼性が求められる基幹システム等、過去のほとんどの分野で実績がある。これに対し、非ウォーターフォール型開発は、情報システムを市場へいち早く提供していくことに価値があると考えられる分野に向いている。特に、開発形態が多様化している後者において、非ウォーターフォール型開発の適用に適した領域を見定め、その活用を促進していくことを本検討の目的とする。

また、現在、非ウォーターフォール型開発があまり適用されていない領域においても、非ウォーターフォール型開発の特質を明らかにすることにより、非ウォーターフォール型開発の今後の適用を検討していく。

非ウォーターフォール型開発に向けた検討の結果、考えられるメリット例は以下の通り。

①日本のソフトウェア産業の実態に適しており、かつ世の中のパラダイム転換に対応することのできるソフトウェアの作り方の提案になる。
②グローバルな視点から見たわが国のソフトウェア産業の競争力を強化することにつながる。
③優先度の高い機能から順次提供され、提供された機能を検証、投入することで、変化が激しく、優先度も変化するビジネス環境に対応できるサービスやシステムを手に入れられる[3]。
④エンジニアが自分自身の成長を実感でき、開発したシステムが利用者の役に立っていると実感できることで、エンジニア一人ひとりが生き生きと働くことのできる環境の整備につながる。



図表 1-1　非ウォーターフォール型開発の目指すべきゴール

また、アジャイル開発の一手法であるスクラムは、一橋大学の野中郁次郎・竹内弘高氏共著による論文 「The New New Product Development Game」（Harvard Business Review 1986 年 1～2 月号）にその名前の由来[1] をもっており、さらに、その起源において、リーンに大きく影響されている。

リーン生産方式は、1980 年代にトヨタ生産方式を中心とした、日本の自動車産業の強さの秘訣を体系化したものである。リーンの考え方（リーン・シンキングの応用）は、製造業から始まり建築、医療、サービス産業へと波及し、製品開発の分野でも最近、リーン・プロダクトデベロプメント、リーン・エンジニアリング、として浸透してきた。

米国で 2003 年に発表されたリーンソフトウェア開発では、ソフトウェア開発にリーン生産方式を適用する考え方が提唱された。リーンソフトウェア開発においては、「ムダの徹底的な排除」や「品質の作り込み」、「全体を最適化する」等の原則を元に、ソフトウェア製品を素早く提供することの重要性が提唱され、これらの原則に加え、現場で働く人々が考える環境の醸成、人間性尊重の概念も重要な基盤としている。

さらに、現在、米国のINCOSE[2] においても、リーンをシステムズエンジニアリング分野に適用するためのLEfSE (Lean Enablers for Systems Engineering)が議論されている。

このように、日本に起源を持つリーンが世界のさまざまな産業を変えさらにソフトウェア開発にも浸透してきていることを誇らしく思うと同時に、現在の日本の産業構造に適したソフトウェア開発のあり方、開発者のワークスタイルのあり方、グローバルビジネス環境の中で国際競争力のあるソフトウェア製品やサービスについて、私たちは再度自分たちで考える契機と捉えている。

---

[1] この論文中で、富士ゼロックスの FX-3500 の製品開発においては、開発フェーズが重なりあっている（刺身システム）ことが成功要因となっていると指摘されている。また、ホンダのシティ 1200CC の開発では、実作業を担うメンバ個々人が責任と権限を持ち、メンバ全員が一丸となって開発に取り組んだ（ラグビーシステム）ことが成功要因となっていると指摘されている。

[2] INCOSE(International Council on Systems Engineering) .－ Systems Engineering Handbook v.3.2－

２　アジャイルとは何か

ここでは、再度、この活動で使われる「非ウォーターフォール」および「アジャイル」という言葉を定義する。

ウォーターフォール型のソフトウェア開発では、品質の高いソフトウェアを生産性高く開発するために、開発初期に要求の固定をはかり、ドキュメントの形で仕様を形式化してソフトウェア・エンジニアリング的な開発モデルに乗せようと努力してきた。

しかし、そもそも要求が刻々と変化している場面では、要求を固定すること自体が製品やサービスの販売リスクを拡大してしまう場合が多い。また、開発の中には技術リスクが大きく、実際に作ってみないとそのリスクを解消できない場合がある。このような状況においては、従来のウォーターフォール型ではない、別のソフトウェア開発モデルが必要とされてきている。

ここでは、「非ウォーターフォール」という言葉を使って、広くこの文脈を表現している。

２．１　非ウォーターフォール型開発の定義

非ウォーターフォール型開発とは、仕様を開発前に固定し、それを分析、設計、テスト等のフェーズを順次踏んでいくという 1970 年の Winston W. Royce の論文「Managing the Development of Large Software Systems」でのウォーターフォール型開発[3]以外の開発モデルの総称である。

非ウォーターフォール型開発の例として、

- **プロトタイプ**　(Frederick P.Brooks, Jr.－1975 年「人月の神話」)
- **スパイラル**　(Barry w. Boehm－1988 年
  「A Spiral Model of Software Development and Enhancement」)[4]
- **RAD**　(James Martin－1991 年 「ラピッドアプリケーションデベロップメント」)
- **RUP**　(Philippe Kruchten－2000 年「ラショナル統一プロセス入門」)
- **アジャイル**
  - **Evo**　(T.Gilb－1981 年「Evolutionary Development」)
  - **Scrum** (Ken Schwaber－1993 年「アジャイルソフトウェア開発スクラム」)
  - **DSDM**　(1995 年「DSDM　ver1」)
  - **XP**　　(Kent Beck－1996 年「XP エクストリーム・プログラミング入門 」)

[3] この論文では、ウォーターフォールの言葉は使われておらず、下流工程までを予備的に行って上流工程へ戻る「フィードバックループ」が推奨されている。
[4] スパイラル開発は、ウォーターフォール型大規模開発でも使われており、プロトタイプ開発は、ウォーターフォール型開発の要件定義工程でも使用されている。

FDD－Feature-Driven Development
（Peter Coad－1997 年「Java エンタープライズ・コンポーネント」）
Lean Software Development
（Mary Poppendieck, Tom Poppendieck－2002 年「リーンソフトウェア開発」）
Crystal Clear （Alistair Cockburn－2004 年「アジャイルソフトウェア開発」）
EssUp－Essential UP
（Ivar H. Jacobson－2005 年「Rational Software Development Conference」）
Kanban（David Anderson－2010 年「Kanban」）
を含む Agile Manifesto(後述)に則る手法がある。

これらの非ウォーターフォール型開発の代表例が、1990 年代後半に現れたアジャイル開発である。本資料では、アジャイル開発を非ウォーターフォール型開発の代表例として論じている[5]。

[5] アジャイル以前の繰り返し型開発を、IID(Incremental and Iterative Development)と総称することもある[4]。

２．２　アジャイル開発の定義

アジャイル宣言[6]の４つの価値と１２の原則を満たすソフトウェア開発手法をアジャイル開発とする。

要約すると、

アジャイル開発とは、不確実なビジネス環境の中で変化するニーズへの迅速な対応を目的としたソフトウェア開発手法である。

この目的を達成するために、アジャイル開発は、徐々に明確となる顧客ニーズや要件をシステムへ反映し、プロジェクトマネジメント・リスクの早期低減、顧客側と開発側のギャップを解消する。アジャイル開発の主要な特徴は以下のとおり。

アジャイル開発は、 ● 「顧客の参画の度合いが強い」 ● 「動くソフトウェアを成長させながら作る」 ● 「反復・漸進型である」 ● 「人と人のコミュニケーション、コラボレーションを重視する」 ● 「開発前の、要求の固定を前提としない」 という特徴をもつ。

以下に、アジャイル宣言を引用する。

・アジャイル宣言（Agile Manifesto）における４つの価値

私たちは、ソフトウェア開発の実践を手助けする活動を通じて、よりよい開発方法を見つけだそうとしている。
この活動を通して、私たちは以下のことを重視する。
①プロセスやツールよりも個人と対話を、
②包括的なドキュメントよりも動くソフトウェアを、
③契約交渉よりも顧客との協調を、
④計画に従うことよりも変化への対応を、
すなわち、①～④の各文の前者（「よりも」の前の言葉）に価値があることを認めながらも、私たちは後者（「よりも」の後の言葉）の事柄により価値をおく。

[6] アジャイル宣言（Agile Manifesto）は、アジャイルな開発手法の提唱者 17 名が集まって、2001 年に発表された。
http://agilemanifesto.org/iso/ja/manifesto.html

・アジャイル宣言（Agile Manifesto）の背後にある 12 の原則

私たちは以下の原則に従う。

①顧客満足を最優先し、価値のあるソフトウェアを早く継続的に提供する。
②要求の変更はたとえ開発の後期であっても歓迎する。
　変化を味方につけることによって、顧客の競争力を引き上げる。
③動くソフトウェアを、２−３週間から２−３ヶ月というできるだけ短い時間間隔でリリースする。
④ビジネス側の人と開発者は、プロジェクトを通して日々一緒に働く。
⑤意欲に満ちた人々を集めてプロジェクトを構成する。
　環境と支援を与え仕事が無事終わるまで彼らを信頼する。
⑥情報を伝えるもっとも効率的で効果的な方法は、フェイス・トゥ・フェイスで話をすることである。
⑦動くソフトウェアこそが進捗の最も重要な尺度である。
⑧アジャイル･プロセスは持続可能な開発を促進する。
　一定のペースを継続的に維持できるようにしなければならない。
⑨技術的卓越性と優れた設計に対する不断の注意が機敏さを高める。
⑩シンプルさ（ムダなく作れる量を最大限にすること）が本質である。
⑪最良のアーキテクチャ・要求・設計は、自己組織的なチームから生み出される。
⑫チームがもっと効率を高めることができるかを定期的に振り返り、それに基づいて自分たちのやり方を最適に調整する。

３　アジャイル開発に適した領域・試行領域

本書においては、すべてのソフトウェア開発にアジャイル開発を適用できる、あるいはすべきだ、という立場ではない。ビジネスや市場、その他の開発の文脈によって、ウォーターフォール型の開発が適している場面もあれば、アジャイル型の開発が適している場面もある。ここでは、現時点で、アジャイル開発が適用されてきている領域と未だ適用されておらず、適用するにしてもアジャイルをそのままでは適用できず、さらなる拡張や工夫が必要とされる領域を特定したい。

大まかには、開発当初に要求を確定せず、ビジネス環境の変化に伴った市場や顧客ニーズの変化への対応が最優先される分野が、アジャイル開発が最も得意とする第一適用領域である。他方、基幹システム等で開発当初に要求をあるレベルで確定可能（あるいは確定すべき）な領域のシステムの開発においては、現在のアジャイル開発は試行領域となっている。

３．１　アジャイル開発に適した領域

アジャイル開発は、「顧客の参画の度合いが強い」「動くソフトウェアを成長させながら作る」「反復・漸進型である」「人と人のコミュニケーション、コラボレーション重視」「開発前の、仕様の固定を前提としない」等を特徴とするため、以下の領域での開発を得意とする。

①ビジネス要求が変化する領域
- 要求の変化が激しく、あらかじめ要求が固定できない領域。

②リスクの高い領域
- 不確実な市場を対象としたビジネス領域（市場リスク）
- 技術的な難易度が高い開発領域（技術リスク）

③市場競争領域
- 他社に先駆けた製品・サービス市場投入が命題で、TTM（Time to Market）の短縮が優先となる領域（Web のサービス、パッケージ開発、新製品開発）。

３．２　アジャイル開発の試行領域

アジャイル開発による経験が十分には蓄積されておらず、現在、チャレンジと創意工夫が求められているのは、以下の領域である。

①大規模開発

アジャイル開発はコミュニケーションを重視するために、開発者 10 人程度の規模が最もうまく機能する。これを超えると、システム分割、チーム分割が必要となる。この場合、サブシステムやチームの分割方法、および、そのチーム間のコミュニケーションについては課題であり、さまざまな方法が提案されている。

②分散拠点（オフショア含む）開発

アジャイル開発はコミュニケーションを重視するために、コロケーション（１つの場所に集まったチーム）が原則である。開発拠点が分散し、さらに時差によって分断される場合のコミュニケーション手法、また、それをサポートするツールの必要性が指摘されている。

③組織（会社）間をまたぐ開発チームによる開発

アジャイル開発は、共通のビジネスゴールをもったチーム、を前提とする。会社や組織をまたいだ構成のチームでは、このような体制を組むことが難しい。

特に、日本では複数の会社が契約をまたいでチームを構成することが多いため、これが大きな課題になる可能性がある。

④組込みシステム開発

ハードウェアの開発と並行してソフトウェアが開発され、高信頼性が求められる組込みシステムでは、リリース後のソフトウェアの修正が極めて困難であることからも、アジャイルの特性の一部が適用しにくく、採用には工夫を要する。

大規模、分散については、現在多くのシステム開発がこれを避けられなくなってきていることもあり、かなりの事例が出ている。さらに、開発組織全体をアジャイルに転換する企業も増えており、組織改革と同時に企業戦略としてアジャイルが採用されている例も多い。

以下、前頁のケースの現在を列挙する。

salesforce.com Co., Ltd.が全社を挙げて Web のサービス開発にアジャイルを採用した事例。
http://www.slideshare.net/sgreene/salesforcecom-agile-transformation-agile-2007-confrence
http://www.slideshare.net/cfry/salesforce-agile-rollout-2007-presentation

IBM が社内の大規模パッケージ開発にアジャイルを適用した事例。
http://agile.dzone.com/videos/sue-mckinney-agile-2009

モトローラの事例を含む、大規模アジャイル開発への XP を適用
Large-Scale Agile Software Development.
http://www.amazon.com/Large-Scale-Agile-Software-Development-Crocker/dp/0321166450

British Telecom のアジャイル組織適用事例
http://www.methodsandtools.com/archive/archive.php?id=43

その他、アジャイルの大規模適用に関する書籍
http://www.amazon.co.jp/dp/4798120405/

さらに、組込みについても、品質を早期に高める手法、テスト中心の手法として、逆にアジャイルのよい特性を利用する事例が出ている。

組込みでのアジャイルのレポート。豊富なメトリクスがある。
「Embedded Agile Project by the Numbers with Newbies」[Van Schooenderwoert]
http://www.leanagilepartners.com/publications.html#newbies

組込みへのアジャイル適用
（Embedded Systems ConferenceBoston October 2008 資料より）
http://www.renaissancesoftware.net/files/articles/ESC-206Slides_Grenning-v1r1.pdf

その他、組込みでのアジャイルの事例のリストがここから参照可能。
http://www.infoq.com/jp/news/2009/01/levison-embedded-agile

③は、日本では産業構造から、契約の課題が避けられないケースが多い。その中でも、挑戦的な例として、リクルートが契約をまたぐプロジェクトチームでアジャイルを適用した事例がある（アジャイルジャパン 2009 より）。
http://www.agilejapan.org/session/CT_Recruit.pdf

４　アジャイルのメリットと今後の課題

アジャイル開発を様々なコンテキストの開発で利用するにあたり、いろいろな立場や背景の方にそのメリットを理解して頂く必要がある。ここでは、顧客[7]の視点とシステムの開発者の視点、それぞれから見たアジャイル開発のメリットとソフトウェア・エンジニアリング上の主な課題をまとめた。

①顧客から見たメリット

・動くソフトウェアで、気になる点が早く確認できる。
・最初からすべての要求が出揃わなくても開発を始めることができる。
・要求の優先順位や、要求そのものの変更に柔軟に対応できる。
・リスクを早期に軽減できる。
・無駄なものを作りださない。

顧客から見た場合、開発の早期に実際に動くものが見える、という安心感がある。最終のテストまでリスクが除去できないウォーターフォール型開発とはこの点が大きく異なる。また､全要求をあらかじめ揃える必要がなく､要求の優先順位を変えることができるので、ビジネスの状況をみながら最終的に最も ROI の高い成果に近づけることが可能となる。最初に要求を固めないので、要求が開発中に劣化してしまうリスクも回避していることになる。

②システムの開発者から見たメリット

・システム利用者の価値と対応させながら仕事を進められる。
・短いサイクルで経験を積み上げ、チームとして成長できる。
・達成感によるモチベーションを維持できる。

これらによって、開発者は責任感とプロ意識を持って生き生きと仕事ができる。

開発者から見た場合、「指示された要求を満たす」だけではなく、利用者の立場でシステムの価値と自身の仕事の価値を結びつけるよう、考えることができる。また、繰り返し開発の中で、技術的な学習、業務の学習をしながら成長し、リリースの都度、顧客とともに達成感を得ることができる。プロフェッショナルとしての仕事のやりがいを、アジャイル開発によって取り戻すことができる。

ただし、上記のようなアジャイル開発のメリットを享受するためには、以下のような意

---

7 「顧客」という言葉は広い意味を持っているが、開発プロジェクトごとに異なる意味で使われている。本書で使用されている「顧客」は､「顧客企業」および「エンドユーザ」という開発者（作る人）以外の関係者（ステークホルダー）として使用されている。
組込みソフトウェアや製品開発、Web サービス開発などの領域も含めて､「顧客」や「ステークホルダー」の意味をコンテキストごとに捉え直すことや、それらに対しての価値を示すことがアジャイル開発ではより重要になる。

識と行動が前提となっている。

・開発者が顧客のビジネスを理解している。
・顧客がチームの一員として参画し、タイムリーな意思決定を行う。
・顧客および開発者が、スコープの変更を許容する。

このように、顧客も開発者も、アジャイル開発においてはゴールを１つにした意識が必要である。このため、日本では契約のあり方が１つの課題となる。

コラム　システムインテグレーションとアジャイル開発（1/3）

アジャイル開発の適用例・成功例のリストを眺めていると、アプリケーションの自社開発、開発ツールや業務パッケージ自体の開発などが多いことに気がつく。一方で、欧米においてはパッケージ導入と、アプリケーションの自社開発の比率が高く、日本においてはユーザ企業とシステムインテグレーターとの受委託によるアプリケーションのスクラッチ開発の比率が高い点は良く指摘される。したがって、受委託によるスクラッチ開発が多いことは、日本においてのアジャイル開発の適用比率が低い原因の一つではないかと思われる。

また、日本においては、ネットあるいは携帯電話などを対象とした比較的新しいビジネス領域においてのアプリケーションの自社開発の比率が高いようであり、アジャイル開発の適用例も他のビジネス領域に比べると多く見られる。このような領域においては他社との競合も激しく、サービスインまでの時間との勝負になること、またビジネスの変化への迅速な対応が必要とされるために、アジャイル開発のメリットを享受できる可能性が高いことが理由の一つであろう。

もう一つの理由はビジネスの成長期においては積極的に雇用増が図れるからではないだろうか。企業の成長期においては、システム開発に必要な良質な人的リソースをいかに調達するかが課題である。成長期においては、積極的に雇用を増やすことができるので、雇用が流動的である欧米の国々と同様に自社開発の形態が取りやすい状況と考えられる。一方で、その対象となる産業領域が成熟・安定期に入ると、雇用の流動性が低い場合には情報システムの拡大につれて増強してきた人的リソースをどう有効活用するかを考えねばならない。

つまり、アジャイル開発の適用、自社開発の比率、雇用の流動性の３つは互いに大きく関連していて、アジャイル開発の適不適には、対象となるビジネス領域が直接的に影響を与えているのではなく、そのビジネス領域が成長期にあるかどうかがポイントではないかと考えられる。その辺りについて製鉄業を題材に以下に論じてみたい。

コラム　システムインテグレーションとアジャイル開発（2/3）

製鉄会社は比較的早期にコンピュータを生産現場に導入したことで知られている。たとえば新日鉄の八幡製鉄所には 1961 年に IBM 7070&1401 が導入された。当時アプリケーション開発はユーザ企業側が担当し、OS、コンパイラ、TP モニターなどのシステムソフトウェアの供給を汎用機ベンダが担当していたようである。実際、国鉄の MARS の開発などもそのような役割分担であったようで、当時には珍しくなかったようである[5]。また、アプリケーションの領域においては現在のアジャイル開発に近い形態でのユーザ企業自身による開発も多かったようで、保守・運用はユーザ企業が自前で実施していた。

コンピュータの生産現場への導入が早かった一方で、80 年代後半の円高不況によって本業すなわち製鉄業の成長に大きく期待できない状況が比較的早期に到来した。そこで、多角化経営、すなわち様々な業種にビジネスを広げることに注力し始めた。これには、他にも様々な目的があったと思われるものの、先に挙げた人的リソースの有効活用も主要な動機の一つであったであろう。リソースの有効活用の観点からは、情報システム部門をアウトソーシングするやり方もあるが、たとえば新日鉄の場合はシステムインテグレーターとして新日鉄以外の顧客に対するビジネスを狙うことになった。当時としては、システムエンジニアなどのコンピュータ関連技術者が著しく不足するであろうとの予測が立てられていたことも、システムインテグレーターとしてのビジネスに乗り出すことを後押ししたと思われる。

このようにユーザ系と呼ばれるシステムインテグレーターには、企業の成長期から安定期に入るなどによる情報システムの開発リソースの所要量の変化に対応するために生まれたものも多くあったであろう。これには雇用の流動性が大きく関わっている。雇用の流動性が低く、開発の所要量の変化が大きい、もしくは見通しが不明確であると、本来不必要な固定費用が増大する恐れがあるために、「保有よりも利用」の方向性が高まることになる。実際、個別の業種あるいは企業を見ると、ビジネスが拡大するところ、停滞するところ、縮小するところ、と様々な局面にあるだろう。よって、社会全体としては、縮小あるいは停滞するところから拡大するところに IT 人材のリソースを回せるような流動性を確保することが期待される。その中で、ユーザ系企業が自社開発のリソースを活用してシステムインテグレーターとしての子会社を作ることは、雇用の流動性の低さを補ってリソースの最適活用を図る一つの有力な方法であったはずである。

コラム　システムインテグレーションとアジャイル開発（3/3）

クラウド時代に入って、ビジネスからの情報システムに対するアジリティやフレキシビリティへの要求が、一層強まってきている。新しいビジネス領域のみならず、既存のビジネス領域においてもクラウドのリソースを活用してビジネス上の価値を創造するタイプのシステムが今後重要となる。そのようなシステム開発においては、スコープや仕様などをあらかじめ確定させるのは困難であるので、アジャイル開発が活用できるだろう。一方で、企業として何を保有し、何を利用すべきかを見直す機運も高まりつつある。大部分の産業においては成長の度合いが不透明な時代となってきているので、自社開発に必要なリソースを保有するリスクも相対的に高くなっている。

そこで、雇用の流動性が現状と変わらない（もしくは固定化が進む）ことを前提とするのであれば、アジャイル開発の促進のためには、受委託あるいはそれに準じる形をベースにしつつも、アジャイル開発に適した役割モデル、コストとリスクの負担の構造、それらに応じたモデル契約などを明確化・整備することによって、自社開発に近い状況を作り出す方向性があり得る。自社開発においては、開発者と利用者（提供者）が一体化している、もしくは開発者と利用者（提供者）に共通的な意志決定者（責任者）がいる、といった特徴がある。これらによって、受委託によるものに比べて、開発スコープのコントロールや、何か問題が起きたときの対処が容易になると思われる。これらと同様な状況が契約や第三者機関などの仕組みによって作り出すことができれば、受委託においてもアジャイル開発の適用が進むことが期待される。

アジャイル開発を実現するための人的リソースに目を移してみると、SE、すなわち IT 技術者の不足が叫ばれた当時と異なって、現在は頭数としてはオフショア開発などの拡大もあって余剰感が感じられるが、一方でハイスキルな技術者は不足していると言われる。アジャイル開発を実行できるリソースとしては、現状ではおそらくハイスキルな技術者が必要であろう。したがってアジャイル開発に耐えうる開発リソースが不足する可能性がある。この課題に対しては、ハイスキルな技術者の流動性を高める、アジャイル開発が可能であるような技術者を育成する手段を整備する、それほどスキルを必要としない開発形態を確立する、などの方策が必要である。システムインテグレーターとしても、現有の開発リソースをアジャイル開発でいかに活用できるようにするかを考えねばならない。

（非ウォーターフォール開発 WG　南　悦郎　記）

５　経営層にとっての価値

非ウォーターフォール型開発は、ソフトウェア開発手法ではあるが、従来のように、プロジェクトチームだけが関与すべきものではない。とりわけ利用企業の経営者は、開発手法の採用決定から開発成果の確認にいたるまで、関与することが不可欠である。しかしながら、この点に関する理解は必ずしも高くない。技術的な問題であり経営上の問題であるとは認識されていないからである。

不確実な経営環境であればあるほど、グローバルなオペレーションが進めば進むほど、さらに、企業同士の吸収合併、分離など、変化が激しいほど、従来型の開発手法は適合しない。そのため、非ウォーターフォール型開発の必要性が増大し、早急な取組みが、ユーザ企業のみならず、IT ベンダを含む IT 産業における大きな課題となっている。

５．１　経営層にとっての情報システム開発のパラダイムシフトの意義

企業や組織における情報システムの位置付けの大きな変化を述べ、それに対応するための新しい情報システムの構築プロセスを提案し、発注側の経営者（CEO)、現業部門の要求責任者（COO)、情報システム役員（CIO）の主体的で濃密な関与が必須であることを述べる。

従来のソフトウェア開発において、CEO、COO、CIO など企業の経営層は、開発費用と効果目標を設定した後は、情報システムの実現をプロジェクトチームに、いわば、丸投げ、してきたケースが多い。つまり開発プロセスに経営層が自ら関与する必要がないと考えていたのである。しかし、日々発生する経営環境の変化、戦略の変更、業務改革の進展などの不確実な事態によって、経営層は情報システムのあり方と無関係ではいられなくなっている。

企業ガバナンスの重要性を提起したOECDコーポレートガバナンス原則によれば、企業会計基準、ITガバナンス、IT統制、ITプロジェクト統制などの遵守、とりわけ、企業ガバナンスとITガバナンスが価値連鎖によって結ばれているとして、まさしくITの諸課題は、経営層自身の問題として取り組むことを経営者に求めている。また、日本においても経済産業省が主導して、ITガバナンスに関する具体的なガイダンスが公表されている[8]。

実際に、大きな情報サービスの品質トラブルの背景には、経営者の不作為を問われかねない事例[9]も少なくない。経営者がシステム開発にどのようにかかわるべきか、どのような要求を実現し、どのような開発手法を採用すべきか、などは、開発チームの問題だけではなく経営者の問題でもある。非ウォーターフォール型開発の啓発は、まず、経営者にとっての意義と価値を理解することから取り掛からねばならない。

本報告書で検討される非ウォーターフォール開発とは、このような企業ガバナンスとITガバナンス、いいかえると企業統制と内部統制に、少なからぬ影響を与えるものである。また、非ウォーターフォール型開発が、エンドユーザコンピューティング、ITリスク評価、開発・保守に関する手続き策定と保守、システムテスト、システム運用・管理、プロジェ

---

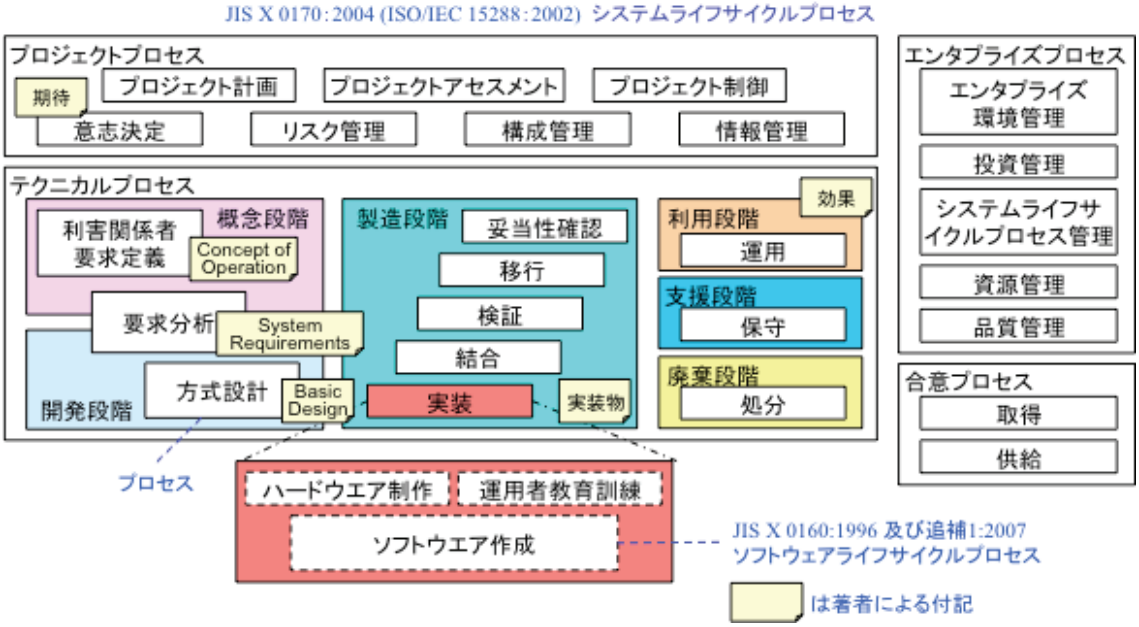
[8] 「IT 経営ポータル : IT ガバナンス」、「IT 経営憲章」、「企業の IT ガバナンス向上に向けて」など、いずれも経済産業省のサイト(http://www.meti.go.jp/)から入手可能

[9] 近年多発する金融機関におけるシステム障害も経営者の責任が大きいといわれている。

クト管理、外部委託管理[10]など に深く関係することはいうまでもない。

（1）情報システムとは

ISOとIECは 2002 年に「システムライフサイクルプロセス」（ISO/IEC15288）という国際標準を発行した。これは 2004 年にX 0170 として日本工業規格（JIS）になっている。これによるとソフトウェアライフサイクルプロセス（図表 1-2： ISO/IEC 12207、JIS X 0160）は、システムライフサイクルプロセスの実装部分に当たるとされる[11]。ここでいうシステムとソフトウェアの違いとは何か。



図表 1-2　システムライフサイクルプロセス

正面から「システムとは何か」を問うことはあまりなく、一般には、システムとはソフトウェアの集まりであるくらいにしか考えていないかもしれない。システム論は、1950 年代中頃に起こった認識論の一つで、さまざまな分野を超えて見られる現象を、「システム」と呼ぶ共通な概念で解釈しようとするものである。システム論は本質的に全体論（Holonism：ホロニズム）を基礎とし[12]、システムを構成する要素の相互作用によって、要素の総和以上のパフォーマンス（創発特性）を示し、要素間の通信を効率化するための自己組織化やフラクタル性などの特性をもつ。また、システム論では、問題の原因を特定

[10] 経済産業省「システム管理基準追補版「財務報告に係る IT 統制ガイダンス（平成 19 年 3 月 30 日）」」に記載

[11] その後 2008 年に、ISO 15288 と 12207 は改訂されて、ソフトウェアライフサイルプロセスは、ソフトウェアを対象とするシステムライフサイクルプロセスであるとされた。この説明は、システムの特性であるフラクタル性（自己相似性、つまり、ソフトウェア自体もシステムと見ることができる）を理解していないと非常に奇妙なものと感じる。

[12] [6][7][8]を参照

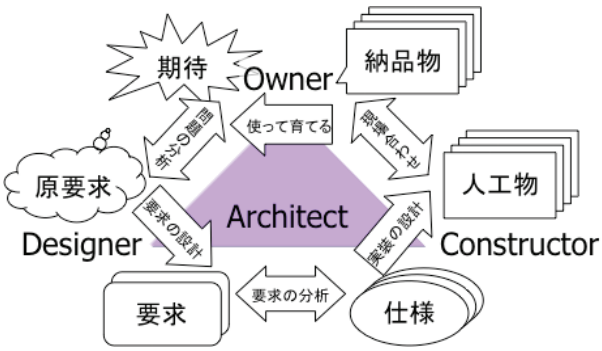
のシステム要素に還元するのではなく、発現している問題自体をもシステムが創発した結果と見る。したがって問題解決のためには、システムそのものを変える必要があると考えている。

システムはその性質から、ハードシステムとソフトシステムに大別される[8]。ハードシステムはその要素が複雑に相互作用するものの、理論的にその振る舞いが予測されるものである。たとえば、太陽系（the solar system）では、いつ地球で日食が起きるかを予測できる。一方、ソフトシステムは、人が要素に加わっており、合目的的であるが、その振る舞いの予測は困難である。たとえば、政治システムがその典型で、その施策の正しさは理論的に証明されない。情報システムもソフトシステムの一つと考えるべきで、一群のソフトウェアと、それを所有する人、それを使って仕事をする人、ソフトウェアを作る人などから成る。それは情報を扱い、それによって何らかの目的を達成しようとするシステムである。

あえてこのようなシステム論の議論を紹介したのは、ここで問題にしようとしている良い情報システムを作るためのプロセスとその評価基準をあらためて、問い直したいためである。

（2）情報システムサイクル

上で述べたような情報システムを構築するにあたって、標準の「システムライフサイクルプロセス」には明記されていない重要な前提がある。それは、このシステムライフサイクルを回していく主体は誰かという問いである。その主体は、「利用段階」において情報システムが効果を生み出すよう促し、さらに効果を高めるために、情報システムそのものを変えていく責任を持つ。この主体も含め、情報システムの構築に関し、「情報システムサイクル」（図表 1-3）が提案され、その中で、情報システムの Owner の役割と責任について詳述された。ここで、Owner とは発注側の経営者（CEO）、現業部門の要求責任者（COO）を指す。



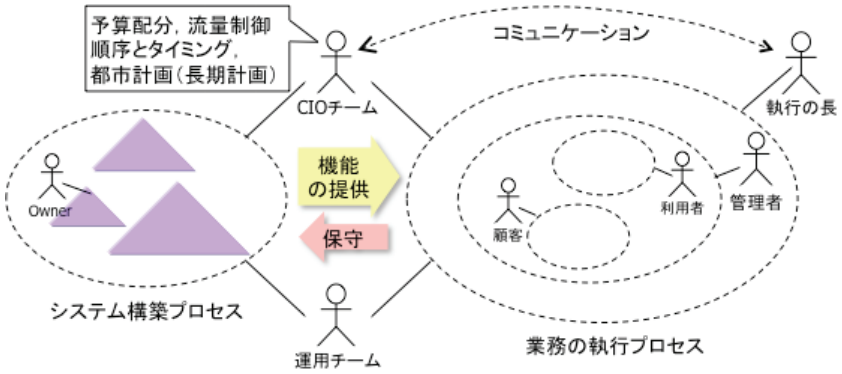
図表 1-3　情報システムサイクル [13]

[13] 児玉[9]は、Owner、Designer、Constructor を、それぞれ「施主」、「設計者」、「施工者」と訳している。Enterprise Architecture の基礎になったと言われる Bachman[15]も、情報システム開発と建築との類似点に基づいて論じているが、日本では建築のメタファー（比喩）に少なからず抵抗感もあるとの配慮から、本報告では大文字で始まる英単語とした。表記上は単数だが、概念上はロールであり、それぞれ 1 人以上の要員からなる。

情報システムの Owner は出来上がったソフトウェアを利用者に使わせることで、企業にとっての効果を生み出すように努めなければならず、そのために小さな保守（現場合わせ）を繰り返し、問題を再認識することで情報システムを作り替えるということを長期にわたって続けなければならない。こうして情報システムは、長い時間をかけて変化し価値を生み続けるか、価値を生まなくなることで廃棄される。長い時間生き延びた情報システムは、それ自体その企業の文化を反映していると同時に、その企業の仕事の仕方を規定していることが多い。

さて、企業情報システムは個々の情報システムから構成される大きなシステムである。これは、歴史的に個々のシステムが少しずつ作られ、それらが相互に連携するように調整され、さらにまた新しいシステムと連携するというように徐々に統合され、拡大してきたという経緯にある[11]。この過程は都市が形成される過程になぞらえることができる[12]。すなわち、ある程度の大きさの企業情報システムができてしまえば、勝手に個々の情報システムができてしまわないように、そして企業情報システム全体がより最適な状態に近づくようにコントロールする行政機構が必要となり、その長である情報担当役員（CIO）が行政機構を所掌する必要が生ずる。企業情報システムのあり方を（都市計画のように）中長期的に設定し、それに向けてインフラを整備する。この中長期計画のための枠組みこそが Enterprise Architecture（EA）であり、CIO の重要な武器である。

CIO が率いる組織は、一般に「情報システム部門」と呼ばれる。情報システムの構築に関わる資源の配置とプロジェクト遂行の支援を行うとともに、開発が完了するとシステムと要員を現場に配備して業務の執行に使用する。これが効果的に稼働しているかをモニターして、業務執行の長と常に対話することで情報システムサイクルをコントロールする。このイメージを図表 1-4 に示す。



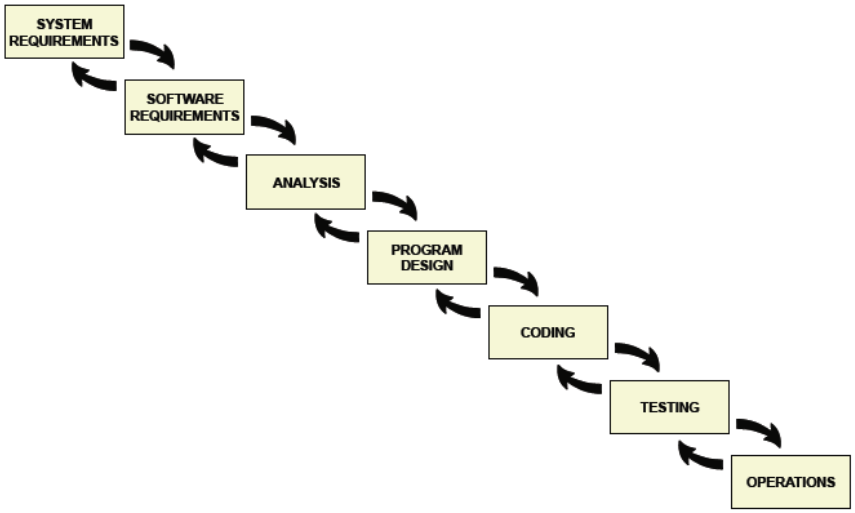
図表 1-4　企業情報システムの構築プロセスと業務執行プロセス

このことは、企業のビジネスにとって、情報システムが必要不可欠な道具となっていることを示唆する。ただし、ビジネスは止まらないので、その道具は、例えば航海しながら船を増築しているような危うい状況にある。これが現代の企業情報システムを巡る問題状況であると考える。

（3）情報システムの開発プロセス

このような現代的な情報システム観に立って、これまでの開発プロセスを再評価してみたい。

まず 1970 年に発表された Royce[13]が提起したプロセスを見てみよう。図表 1-5 はその一例である。これは後にウォーターフォールモデルの提案と誤読された図である。Royce はこの論文の中で、一つずつ前のステップに逆戻りすることを認めているし、二度作れ（Build Twice!）とさえ述べている。「ウォーターフォール」という言葉さえ使っていない。



図表 1-5　Royce のソフトウェアプロセス[13]

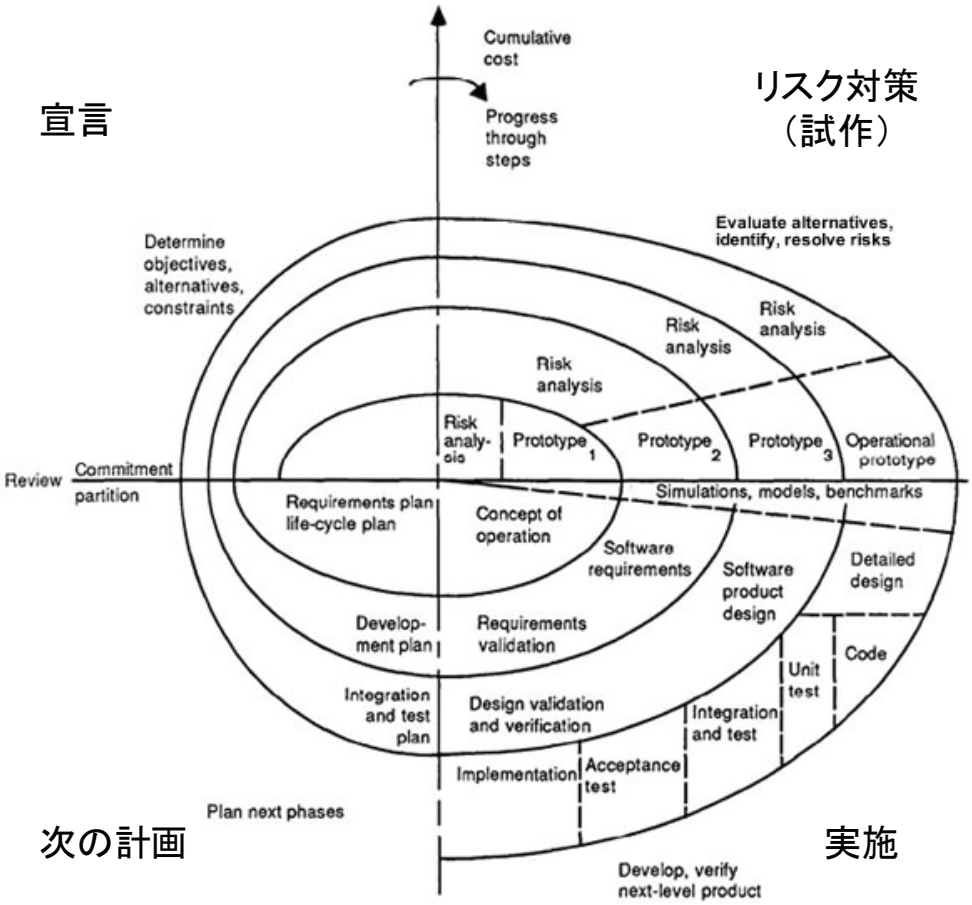
Royceの主張は、現在広く知られている「ウォーターフォール」と呼ばれる手戻りなしの一度きりのプロセスモデルではない。決められた作業を順に行うことで目標の製品が出来上がるという、工場の生産ラインのようなプロセスは単純でわかりやすく、理想的に見えるが、情報システムは量産品ではない。その都度、個別に異なる問題状況を分析して構想・設計・試作を繰り返す新製品の開発プロセスに近い。だから現実の情報システム開発では、当然手戻りが発生する（もし発生していないとすれば、それは問題を後工程に押しつけているだけか、量産品を作っているかのどちらかである）。いつどこで、このような「ウォーターフォール」の誤読が始まったのだろうか[14]。

さらに、Boehmのスパイラルモデル[14]を見てみよう。図表 1-6 に示すように、宣言、リスク対策、実施、次の計画を行う四つの象限があり、プロセスはこの中を右回りにらせんを描くように進む。最初の 1 周目は業務構想書（Concept of Operation）を作成する目的で回す。業務構想書が承認されると、ソフトウェア要求定義[15] のサイクルに進むかどう

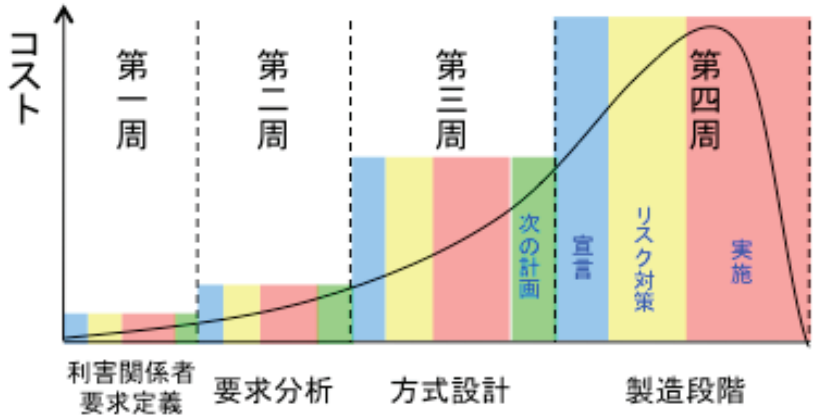
[14] 米国国防総省のソフトウェア開発及び調達に関する標準 DOD-STD-2167 で用いられたと言われる。これにより失敗プロジェクトが頻繁に発生し、改訂文書が出されたものの、徹底されなかった模様である[16]。

[15] システムライフサイクルプロセスが定義されている現代では、このソフトウェア要求定義はシステム要求定義と読み替えるべきだろう。

かを判断する。進むとなれば、まずその旨を宣言し、要求定義のリスクを減少させるためのプロトタイピングを行った上で要求定義書を作成する。同様に、次のサイクルでソフトウェア製品設計書を作成し、次のサイクルで一気に詳細設計からコーディング、単体テスト、統合テスト、受入れテスト、配備へと進む。つまり 1 周ごとに次の周回へ進むかどうかを判定し、4 周で情報システムを開発し終わるようになっている。注目すべきは、作業の実施前に毎回プロトタイピングを行う点である。



図表 1-6　Boehm のスパイラルモデル



図表 1-7　スパイラルモデルとシステムライフサイクルプロセスとの対応

このスパイラルを、横軸に時間を割り当てて書き直したのが図表 1-7 である。ここから Rational Unified Process（RUP）への影響や、その後の反復型プロセスや、システムライフサイクルプロセスへの影響が感じられないだろうか。この論文が発表されたのは、実に 1989 年のことなのであり、Royce や Boehm などの先人の成果が正しく理解されていないのではないだろうか。

（4）情報システム開発の環境の変化

このスパイラルモデルと Royce のモデルとの大きな違いは、「上流」工程での頻繁なプロトタイピングの実施にある。このことは、Royce の頃（1970 年代前半）に比べて、Boehm の頃（1980 年代後半）には、情報システムの開発により大きなリスクを伴うようになったことを意味する。

実際、経験的にも 1970 年代のソフトウェアは、帳票作成が中心[16] だったことは明らかである。この頃は、ソフトウェアそのものがほぼ情報システムと同意であった。それが 1980 年代の後半からは、ソフトウェアの適用業務領域が拡大し、それに連れてOwnerの要求は複雑となり、前もって明確に記述できないものとなった。言い換えると、情報システムはソフトウェアで構成されるとは言えなくなったのである。その大きな原因は、システムが対顧客業務の中で使われる（オンサイト）ものとなり、マーケットがシステムの構成要素になったことである。

このような変化は、ハードウェアや OS、DBMS、ネットワークなどのプラットフォームでも見られ、メインフレーム中心からネットワークで有機的に接続されたサーバ群が主流となり、“クラウド”と一括りにされる多様なコンピュータ資源の活用技術が現れた。開発手法もオブジェクト指向言語、モデリング言語、開発ツールや開発方法論が急激に整備されてきた。ここ 10 年で、情報システムの開発に対する要請が大きく変わったと感じる。

（5）情報システムの評価基準

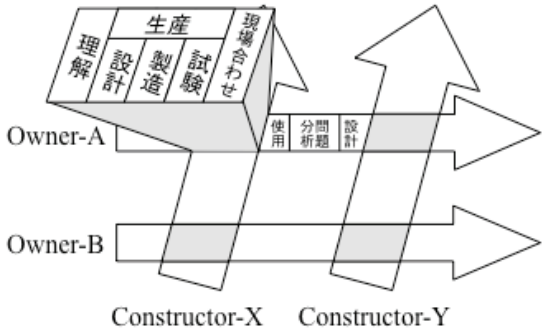
さて、以上の議論を基に、もう一つのテーマである情報システムの良さの基準をどこに置くかを考えてみよう。われわれは、既にソフトウェアと情報システムは別の見方であることを理解した。情報システムをソフトウェア、つまり“もの”として見るとき、その開発努力は QCD、つまり、品質、コスト、納期を目標どおりに達成するために費やされる。これは「ものづくり」では当たり前のことである。ただし、ここでいう品質は仕様どおりできていること、バグが少ないこと程度を指す。そして、これはあくまでも Constructor 側の論理である。

情報システムをシステム、つまり“こと”として見るとき、その開発努力は EEE を高めるために費やされる。EEE とは Efficacy, Efficiency, Effectiveness で、それぞれ、正しい答えが出せること、効率がいいこと、意図どおりに動くことなどと訳せるが、これらの厳密な違いよりも、システムが目的に対して効果的であることの重要性を説いている。EEE の指標は経営指標でもあり、ビジネスの性格によって個別に選択される。例えばサービスの平均応答時間や在庫回転率などである。これは、Constructor の価値観とは大きく異なる。

図表 1-8 は、価値の基準が Owner と Constructor とで異なることのイメージを示している。両者の論理が重なるのは「プロジェクト」と呼ばれる実装段階だけで、それが終わ

16 当時のコンピュータはまるで印刷機だったといわれている。

るとそれぞれ異なる道を行く。



図表 1-8　Owner と Constructor の価値観の違いのイメージ

このように、良い情報システムを作るためには、Constructor の論理だけではなく、Owner の価値観を明示的に導入する必要がある。

（6）発注者との濃密なコミュニケーション

本節の冒頭に述べたように、情報システム開発に対する要請が変わったにもかかわらず、発注側の経営者たちは相変わらず、情報システムの実現をプロジェクトチームに、そしてプロジェクトチームは Constructor に、いわば丸投げしてきたケースが多い。ウォーターフォール型プロセスモデルは、それを可能とするプロセスになっているからである。このプロセスは、前工程の成果が常に正しいという幻想を前提にしており、明示的、暗黙的にかかわらず問題を後工程に順送りにして、テスト工程でカバーするという事実上不可能な仕組みになっている。そのため、建前上は手戻りがないことにして、後工程で前工程をやり直すことも多い。実際、そのような問題指摘は、以前からあり、Brooks[16]は明確に「ウォーターフォールモデルは間違いだ」とさえ述べる。ここに日本的な下請け構造が重なることで、様々な不合理な問題に対する異議申し立ては封じ込められてきた。

一方、Constructor 側もなぜ「ウォーターフォール」を採用し続けるのか。その最大の理由は、現役でなくなった、かつての開発者が、管理者になって効率的に「管理」するために、プロジェクトに一律で横並び的な「工程」が必要だったからであろう。

ウォーターフォールの問題を否定しきれなくなってきた現在では、非ウォーターフォール型プロセスが強制するソフトウェアプロセスへのOwnerの濃密な参加という過大な負担を問題視することで、新しいプロセスの採用を拒否しようとしてきた[17]。実際、アジャイルプロセスでは、それが大規模になるほどOwnerの負担は大きくなる。それでも、効果的な情報システムを作るには、これらの負担は必要不可欠であると考えられる。

このような発注者との濃密なコミュニケーションのあり方について、近年の建築の世界でのパラダイムシフトのいくつかの取組み（コラム「建築の知見の情報システム学への適用」を参照）が参考になるだろう。

[17] 笑えない事実であるが、インタビューで、「やったことがないから」という理由を告げられたこともあった。

（7）非ウォーターフォールモデルの課題

しかし、実際に大規模な基幹システムの再構築をアジャイル（例えばスクラム）で実施してみると、Ownerとの濃密なコミュニケーションは、大規模になるほど難しいと感じる。Constructor側は複数のチームを作ってOwner側の担当者の参加を要請するが、チーム体制に見合った人数の担当者を参加させられないし、担当者間での意思統一も難しい。単に、複数のスクラムチームをまとめるチーム[18]を用意すればいいというものでもない。

ある例では、このような課題が予想されたため、あらかじめ基本的な情報システムアーキテクチャを開発して提供してきた。対象業務の予想以上の複雑さに、アーキテクチャ改訂が追いつかず、Constructor 側の要員も機能要求を十分に理解できないままスプリントを繰り返し、顧客価値が得られないという課題も噴出した。大規模アジャイルでの開発は、小規模プロジェクトの単純な延長や拡大ではない。しかし、ではウォーターフォール型開発が適しているということでは必ずしもない。ウォーターフォールで実施したとすれば、このような課題は、さらに後の工程で噴出し、取り戻すことが難しい状況になってしまうだろう。問題の早期可視化こそが大きな価値なのである。

経験的にいえば、大規模システム開発のリファレンスモデルとして、Boehm のスパイラルモデルの採用が妥当と思われる。そして、各周回のリスク減少と実施フェーズではタイムボックスを設け、アジャイルに行う。例えば、システム設計プロセス、すなわちスパイラルの第２周目では、開発規模を Owner 側の要員が Designer と濃密にコミュニケーションがとれるサイズに切り出して、模型を何度も作り直してアーキテクチャを安定させながら、機能要求と方式設計をまとめ上げる。まさしく、これがウォーターフォール（工学主義）とアジャイル（反工学主義）とを止揚した新しいパラダイム[21]（批判的工学主義）といえるであろう。

本節では、アジャイル開発が、日本のソフトウェア競争力を高めるための重要な開発手法として位置づけられるとともに、現代の不確実性の高いビジネス環境において、日々その役割が増大していると考えてきた。そのためにも、経営層への理解浸透が、次の一歩のために極めて重要であり、大きな課題であると認識し、企業情報システム開発における考え方のパラダイムシフトの意義を検討した。今後、非ウォーターフォール開発の導入に際しての企業ガバナンスと IT ガバナンスに対するさまざまな影響について、今後、具体的な実証作業を行う必要があるだろう。

[18] 大規模開発では、スクラムチームを複数設定しそれぞれに開発を進めるのと並行して、代表者からなる“Scrum of Scrums”チームを設定することがある[22]。さらに Owner グループが要求定義レベルで Scrum を行う“メタ Scrum”チームを設けることがある。

コラム

建築の知見の情報システム学への適用

建築の知見を情報システム学も手本とすべきであろう。建築の世界では、現在、注目すべきパラダイムシフトが起こっている。1970 年代半ば、日本では「組織・ゼネコン」建築家と「アトリエ」建築家の対立があった。

前者は、高層ビルや巨大建築を量産するための技術を優先させ、「工学主義」[19] と呼ばれた。後者は、組織に従属するか、都市建築から撤退せざるを得なかった。これはちょうど、米国で、より人間的な都市建設をめざして、Alexanderが「パタン・ランゲージ」[17]を提案した頃である。そうして、都市には表層的に、短工期、建築費の削減、人間工学的なユーザ快適性が得られたが、深層では都市全体の景観を損ね、運用維持費の増大、環境への悪影響という問題を抱えるに至った。真鶴町が、1993 年にパタン・ランゲージに基づく景観保護に関する条例[18]を制定したのは、この対立問題が表面化しやすい、都市と地方の境目にあったからだろう。

2000 年代に入って、両者の対決を止揚しようとする動きが出てきた。「批判的工学主義」を掲げる藤村龍至らの実践がそれだ。一般に、建築の設計では施主の要求を探索するために、要所ごとに複数の案を作って一つを選択させて枝分かれさせ、徐々に精緻化していく。この手順は後戻りすることもあるので、設計作業の軌跡は、ループのあるツリー状になる。工学主義ではこのような手間を嫌って、周辺環境との調和を半ば無視して既成の設計を押しつけてしまう。

これを避けるために、批判的工学主義では、スピードを重視しつつ、建築に対する施主の与条件を深読みして、社会の要求に応えようとする。藤村が提唱する「超線形プロセス」[19]では、設計プロセスにおいて、「ジャンプしない」、「枝分かれしない」、「後戻りしない」という三原則を貫く。それゆえ、設計作業の軌跡は線形となる。彼は設計検討のたびに模型を作って保存する。設計を改善するときは改善点を一つに絞る。このようないわば「ログ」があるからこそ、上の三原則が可能となり、一定のスピードが得られるという。

このようなプロセスは、施主との濃密なコミュニケーションを必要とする。このコミュニケーションの存在ゆえに、一見ウォーターフォール型に見える線形プロセスも、顧客価値を生むのである。濱野[20]は、この設計プロセスと情報システムのアジャイルプロセスとの類似を指摘している。筆者も、この模型を媒介にした Owner との濃密なコミュニケーションの存在こそ、非ウォーターフォールプロセスが成立する要件と考える。

（非ウォーターフォール開発 WG 児玉　公信　記）

---

[19] アトリエ建築家は工学的でないという意味ではない。工学主義は、社会工学や人間工学を無批判に取り入れ、建築を工業化し、Alexander のいう Quality without a name を忘れてしまったという意味だろう。

５．２　経営層にとっての情報システム開発手法の選択と価値

情報システム開発のパラダイムシフトの意義に賛同し、発注者との濃密なコミュニケーションを含むアジャイル開発へチャレンジしていく企業が増えることを期待する。

一方、アジャイル開発を採用することによって、開発上の課題が全て自然に解決するわけではない。ウォーターフォール型開発ではなかった課題が発生することもあるし、どちらにも共通する課題もある。よって、生半可な覚悟でアジャイル開発に取り組むと、それらの課題につきあたり、「そんなはずではなかった」となり、元のウォーターフォール型開発へ戻ることにもなりかねない。

ウォーターフォール型開発へ戻らない覚悟を持つためには、アジャイル開発でしかビジネス上の成功を得ることができない、ウォーターフォール型開発でも開発は完了するだろうけどビジネス上の成功を得ることができない、という裏づけが一番強いと考える。逆に言うと、アジャイル開発でしかビジネス上の成功を得ることができない、というプロジェクトへアジャイル開発を導入していく。

本節では、そういう観点から、アジャイル開発、ウォーターフォール型開発の採用の考え方についての一例を示す。あくまでも一例であり、大切なのは、アジャイル開発でしかビジネス上の成功を得ることができないかどうかを自らのプロジェクトに問いかけることである。

なお、本節では、情報システム開発における役割として、下記を定義する。

①顧客（利用部門）

情報システム開発・運用など情報システムサービスにかかわる費用を負担し、そのサービスを利用することにより、ビジネスを効率的に推進する部門。

②IT 部門

顧客に対して、情報システムの開発によりサービスを提供する部門。顧客と同一企業の部門であるが、自社内のみでは閉じず多くの外部リソースを活用しているのが一般的である。中小規模の企業では、「部門」というほどの組織はなく、担当者と外部アウトソースの場合もある。

（1）顧客（利用部門）視点の情報システム

顧客からみた情報システム価値の分類の一つとして、図表 1-9 を示す。

情報システム価値分類	説明
高信頼性、高スループット、整合性に価値をおく情報システム[20]（信頼性重視システム）	✓ システム要求が長期固定的であり、寿命も長いシステムである。 ✓ あらかじめ設計されたビジネスプロセスにしたがって、高信頼で高スループットを実現し、そのデータ整合性を常に保つことに最大の価値がある。 ✓ ユーザの固定度は高く、システムの使い勝手によりユーザ数が大きく変動してしまうということはないので、一般的には高度な使い勝手までは求められない。 ✓ 典型的には、財務・調達などバックオフィスシステム、銀行オンラインシステムがこれにあてはまる。
要求へのスピーディな対応、使い勝手に価値をおく情報システム[21]（俊敏性重視システム）	✓ システム要求が次々に変更・追加され、寿命の短いシステムである。 ✓ ビジネス上の理由[22]により、あらかじめ情報システムへの要求を出し尽くすというアプローチをとるよりも、開発した情報システムを評価しながら要求を追加・変更していくというアプローチをとった方が合理的な情報システムである。 ✓ 多くは、新たなユーザ獲得を目指すビジネスで求められ、高度な使い勝手が求められる。 ✓ 典型的には、新たなインターネットサービス、グローバル市場向け新ソフトウェア製品がこれにあてはまる。

図表 1-9　情報システム価値からみた分類

「信頼性重視システム」は、ERP など業務パッケージ活用、SaaS による業務サービス提供により、開発量は減少方向であり、また、企業価値向上への寄与度は小さくなる傾向と考えられる。

「俊敏性重視システム」は、インターネット、モバイルデバイスの普及により、新ビジネス開拓には必須となりつつある。

---

[20] 基幹系システムなどのように、最初から十分な完成度を要求されるシステムを意味する。

[21] Web ビジネス、インターネット系システムなどのように、高信頼性、高スループット、整合性を網羅した十分なシステムの完成度よりも、要求へのスピーディな対応や他社に先駆けた製品・サービスへの対応、使い勝手が要求されるシステムを意味する。

[22]「ビジネス上の理由」というのは大切なキーワードである。「あらかじめ情報システムへの要求を出し尽くせない」ということが、否定的なことではなく、ビジネス上のチャレンジから必然的にそうなる、という肯定的な文脈であることが重要である。

（2）IT 部門にとっての情報システム開発

システム開発手法の採否には、IT 部門の文化が深くかかわっている。まず、図表 1-10 のように２種類に分けて考えてみよう。おそらく実際には、 ２種類の間のどこかに位置付けられることが多いと思われる。

IT 部門の文化	説明
ウォーターフォール型開発への親和性が高い	✓かなり以前から(多くはメインフレーム時代から) システム開発を行っており、ウォーターフォール型開発を基にする独自開発方法が根付いている。プロジェクト管理体制・ノウハウ、人材確保方法などが、その開発方法に最適化されており、その IT 部門の開発文化となっている。 ✓長期間、開発を継続していること自体が、ある程度の成功を物語っており、ウォーターフォール型開発を基にする独自開発方法を肯定的にとらえている。
ウォーターフォール型開発への親和性が低い	✓インターネット文化が根づいた頃からビジネスを開始した企業の IT 部門。その企業では、インターネットに関わるサービスビジネスを自ら行っている場合もある。 ✓ユーザの反応をみてシステムを漸次改善していく、という「永遠のβ版」という考え方に慣れ親しんでいる。

図表 1-10　IT 部門の文化

アジャイル開発には多様な技法・プロセスがあり、その中から自プロジェクトにあったものを採用することが特徴であるが、短期(２W～４W)の反復(イテレーション)開発は共通の特徴といえる。よって、反復開発期間毎に、動作しているソフトウェアを確認することができる。これにより、機能内容・使い勝手などを検証し、早くフィードバックすることができる。また、情報システムへの要求が不十分な段階から開発に入ることができ、できあがってくる機能を動作させながら、要求追加・変更を繰り返していくことができる。

その他にも、開発スピード、生産効率、品質、人材スキルが向上すると言われている。しかし、それは、どのようなプロジェクトにおいても達成できるというものではなく、プロジェクトの特徴をみて、その効果をめざしてアジャイル開発を導入するかどうか個別に判断すべきものと考える。

以上で述べてきた情報システム価値とそれを開発する IT 部門の文化によるウォーターフォール型開発、アジャイル開発の推奨パターンを図表 1-11 に示す。

文化 情報システム価値	ウォーターフォール型開発への親和性が高い文化	ウォーターフォール型開発への親和性が低い文化
信頼性重視システム	①ウォーターフォール型開発を推奨	②推奨なし
俊敏性重視システム	③弱いアジャイル開発を推奨	④アジャイル開発を推奨

図表 1-11　ウォーターフォール型開発、アジャイル開発の推奨パターン

①信頼性重視システム/ウォーターフォール型開発への親和性が高い IT 部門にはウォーターフォール型開発が推奨される。情報システムへの要求の多くは、論理的には開発当初の要求定義フェーズで確定すべきものであり、ウォーターフォール型開発への親和性が高い IT 部門では、過去、そのために多くの改善投資をしてきており、一定の効果がでている。短期でのシステム動作と要件へのフィードバックは、このパターンでは必須とはならない。よって、文化として定着している、ウォーターフォール型開発を基にする独自開発方式を継続して採用することが妥当と考える。

②信頼性重視システム/ウォーターフォール型開発への親和性が低い IT 部門には特に推奨するものがない。情報システムへの要求の多くは、論理的には開発当初の要求定義フェーズで確定すべきものであるが、ウォーターフォール型開発への親和性が低い IT 部門では、そのための体制・ノウハウが不十分な場合が多い。その場合、改善投資によりウォーターフォール型開発へ持っていく、または、アジャイル開発を取り入れてそれを補う、という選択肢がある。その選択は、個々のプロジェクト状況によって、または、IT 部門内文化のあり様によって、選択していくべきことと考える。

③俊敏性重視システム/ウォーターフォール型開発への親和性が高い IT 部門には、アジャイル開発を弱く推奨する。本情報システムの特徴としては、アジャイル開発のメリットと合致するものである。本情報システムを、フェーズ分けなどウォーターフォール型開発で開発できるかどうか検討した上で、ウォーターフォール型開発では無理と判断した場合に、アジャイル開発を採用すべきである。ウォーターフォール型開発に慣れ親しんでいる IT 部門では、アジャイル開発に「変更」すること自体、いろいろなデメリットが生じる。それを、ウォーターフォール型開発では本情報システム開発はそもそも無理で、アジャイル開発(短期の反復開発)が必須である、という意識で乗り越えることが重要となる。

④俊敏性重視システム/ウォーターフォール型開発親への親和性が低い IT 部門には、アジャイル開発を推奨する。本情報システムの特徴としては、アジャイル開発のメリットと合致するものであり、ウォーターフォール型開発のノウハウが薄い IT 部門では、わざわざウォーターフォール型開発を採用する理由はない。

ウォーターフォールに基づく開発を採用するか、アジャイル開発を採用するかは、ビジネス上の成功の観点で考えるべきことである。ウォーターフォールに基づく開発を採用し

ていて、ビジネス上成功を獲得していて、将来もそれが続きそうであれば、アジャイル開発へ変更する必要はないかもしれない。

しかし、俊敏性重視システム開発において初期段階で要求定義が固まらないことによるコストオーバランが多くなっていたり、信頼性重視システム開発案件が減少し俊敏性重視システム開発案件が増加していたりする場合には、上記の推奨パターンを考慮して、経営的観点でアジャイル開発導入を検討すべきと、考える。

５．３　経営層にとっての懸念事項

ウォーターフォール型開発の諸問題とそれに対する非ウォーターフォール型開発の利点や優位性について、どれだけ説明しても、利用企業の経営層を十分に納得できるとは限らない。いくつかの懸念が解消しないからである。経営者から、信頼できる見積り手法、品質保証、進捗管理の見える化などが要求される。

従来のウォーターフォール型開発では、全体を見通し、あいまいであっても要求仕様を確定することで、工数と費用が見積もられ、経営者にとっては、どの程度の資源が必要となるかが比較的、理解しやすかった。それに対して非ウォーターフォール型開発は、要求仕様があいまいなままプロジェクトを進めるため、経営者は、必要資源についての情報が提供されないことに対して、フラストレーションを感じるのは当然ともいえる。

また、非ウォーターフォール型開発では、あえて先の設計はせずに、直近のイテレーションサイクルのみの見積りを行うのが普通である。したがって、見積りが常に概算となり、ウォーターフォール型開発と比べ、不正確であるとの懸念が生じるのもやむを得ない。

しかし、従来のウォーターフォール型開発でさえ、プロジェクトの失敗要因の多くが要求仕様確定後の変更の多さや要求仕様のあいまいさ、意味の共有不足にあることはよく知られている。したがって、要求仕様で見積もっても不正確であって、非ウォーターフォール型開発で要求仕様があいまいであるから見積もれないとはいえない。つまり、ウォーターフォール型での要求仕様確定から見積りという流れ自体が、もはや適合しないことを示している。

見積りは当初は不確実性が高く、プロジェクトが進捗するにつれて正確さが増大することは知られている。その際に、２つの点を考慮したい。まず、当初のあいまいな時期に要求仕様を確定することが困難であるにも関わらず、工数や費用を確定することによって、この見積りに制約され、プロジェクトの失敗を助長してしまう可能性がある。要求仕様に基づいて正確に見積もることに精力を費やすよりも、現実的には、過去のデータから、どの程度の工数だったのか、あるいは、期待効果から、投入可能な予算を推定する方が現実的であろう。これは製品の価格を、原価を積み上げるのではなく、市場で受け入れ可能な価格を参照して設定することと同じ方法である。そしてプロジェクトの進捗に応じて仕様が明確になるにつれて実現可能性を考慮しながら、工数、費用を見直すのが、まさしく非ウォーターフォール型開発にふさわしい見積り方法といえるだろう。

避けなければならないのは、従来のように要求仕様があいまいかつ変更の可能性があるにもかかわらず、IT ベンダに見積り依頼を行い、契約を締結してしまい、その費用に拘束されてしまうことである。もし、あいまいな時期に契約をする必要があるならば、それは確定一括発注方式をとるべきではなく、変更可能な出来高方式、単価方式など、状況に応じた柔軟な契約方式をとるべきである。この点は、調達部門の理解が不可欠であり、部門

横断的であるがゆえに、経営者の理解が必要となる。

品質についても、非ウォーターフォール型開発によって向上するのかどうか、検討してみよう。基本的には、２週間から１ヶ月の期間を開発単位として、要件定義、設計、実装、テスト、リリースを繰り返し、システム構築するのが典型的なアジャイルソフトウェア開発のプロセスである。ここでは、全期間にわたるユーザの関与によって品質の向上が期待される。プログラムは、ユーザが指定したテストに合格するように作成され、ユーザは、短期間で結果のフィードバックが得られ、再度、要望を挙げることができる。結局、品質はユーザが決め、ユーザが納得しなければならないため、このプロセスを経てユーザの納得感にかなった品質を保証しようとする。

さらに、コードレビュー、単体テストなど実装時にバグを混入させないことが期待される。実装時に行うコードレビューが、バグの低減に有効であることはよく知られているが、これまで、開発期間が長くなるとの理由からあまり実施されることはなかった。さらに、コードの書き換えのたびに、何回も回帰テストを行っていては、ユーザの要望に迅速に対応できないとも考えられていた。しかし、ペアプログラミングを行う過程で、コードレビューを組み込むことで生産的なコードレビューが可能となる。また、テスト駆動開発やテストの自動化によって、システム変更後の品質保証が容易になるとともに、わかりやすいプログラムコードへとリファクタリングすることによって、バクが発生しにくい構造へと改善することができる。これは品質向上に寄与するであろう。

ウォーターフォール型開発が、工程の最後に大規模にテストを行い、統計的に品質を確保しようとするのに対し、アジャイルソフトウェア開発は「工程で品質を作りこむ」ことを基本としている。これは、トヨタ生産方式をソフトウェア開発の領域へと応用することを意味する。現在のソフトウェア品質の課題は、開発から稼働までの間でいかに作り込むかだけではなく、本番稼働後の経営環境や戦略の変更、ユーザの期待品質の変化などに対応して、いかに保守しやすく品質向上させやすくするかの方が大きな問題である。その点でも、非ウォーターフォール型開発の利便性は極めて高いばかりでなく、改善可能性こそ経営管理面での大きな利点である。

進捗管理についても、疑問が生じるかもしれない。どこまで進捗したかがわかりにくいと述べるかもしれない。しかし、パッケージソフトやミドルウェアの採用、開発済みモジュールの再利用など、従来のウォーターフォール型開発型であっても、もはや、現在の開発現場での現状ではコード数だけで管理することは困難となっている。さらに、ファンクションポイントを、進捗を管理する指標とすることは困難である。むしろ、WBS で工程を追いかけるのが通常となっている。その意味で、両者の相違はないといえる。むしろ数字だけで進捗を把握するのではなく、テストを頻繁に実施することで、経営層も進捗を理解することは容易である。

また、近年のツールの発達は、非ウォーターフォール型開発で有効活用されている。たとえば、品質プロセス（統合支援）を管理するツールとしての HP Quality Center Software、開発ツールとしての Eclipse、ユニットテストツールの JUnit、XP 管理ツールの XPlanner、XpTrackerPlugin などは生産性向上のみならず、進捗の可視化に有用である。プロジェクト管理ツールとしての Redmine は情報共有のためにも活用できる。テスティングフレームワーク、デプロイツール、バージョン管理ツールや、Vim や Emacs などの高機能にカスタマイズできる簡易なエディタ、インスタントメッセンジャーなどの汎用的

ツールも活用され、ログデータの活用などによって、進捗管理上も有用である。

これらのツールは、本質的に、アジャイルかどうかではなく、開発者としては、常に、自分の仕事の生産性向上、仕事の品質向上に向け、情報収集し、試行すべきものかもしれない。ウォーターフォール型開発よりも非ウォーターフォール型開発の技術者にそういう姿勢が備わっているということかもしれない。

しかし、日頃から、利用企業も IT 企業でも、「開発の連中の言葉は、さっぱり分からない」と述べる経営者が多いため、いかに論理的に説明しても理解を得るのは容易ではない。機能単位で見積もったとしても、結局、人月や費用という経営陣のわかる言葉に変換する方が効果的であり、現場にとっても生産的である。したがって、ツールを活用することで、経営層にもわかるグラフや数字に置き換えて表示することは効果的である。これらは単にツールを導入すれば効果的な進捗管理ができるというよりは、可視化を図ることで管理しやすくなっているからこそツールが有効であることを示している。

経営層の懸念は、非ウォーターフォール型開発に特有というよりも、現在のウォーターフォール型開発においても、すでに発生している問題である。それを解決するためにこそ、非ウォーターフォール型開発が有効であることについて、経営層の理解を得るべきであろう。アジャイルの価値を一言で述べるとすれば、利用者に実物を見せて進捗を報告できるからからこそ、“ごまかしがきかない”、ということに違いない。経営層にはその価値、つまり見える化の価値を十分にアピールする必要があるだろう。

コラム　顧客・経営層に対する進捗の見える化（1/3）

システム構築のマネジメント上、顧客や経営層から進捗状況に関する報告を求められることが多い。これに適切に応えることにより、顧客・経営層と開発プロジェクトチームとの間のコミュニケーションが円滑化され、プロジェクトそしてビジネスの成功に結び付く。

このような QCD の「見える化」は、ウォーターフォール型開発においては、概ね確立されたプロジェクトマネジメントの技法に基づいて行われる。すなわち、組織やプロジェクトによりあらかじめ定められたメトリクスと手順にしたがって、開発データが随時収集され、分析される。そして、開発のどの時点においても、顧客・経営層からの求めに応じて、あるいは、開発プロジェクトが必要と判断した場合に、（コスト等への）所定の換算を行った後、進捗状況を適切に報告することができる。

これに対して、アジャイル開発においては、QCD の見える化が一般に確立しているとは言い難い状況にある。たとえば、テスト駆動型開発やペアプログラミング、リファクタリング等、採用するプラクティスによっては、ウォーターフォール型開発において用いられるメトリクスでは進捗状況を適切に評価できないケースがある。また、進捗等の管理のためのデータ収集・分析作業そのものが、アジャイル開発のメリットを大きく損なうオーバーヘッドになるという考え方もある。したがって、反復やリリースの途中では、顧客・経営層に対して適切な進捗状況を報告できないことがある。このような状況は、顧客・経営層がアジャイル開発の採用を躊躇する理由の一つともなっている。

ただし、アジャイル開発では、もともと顧客の参画を前提とするため、顧客に対しては、開発進捗状況の「見える化」についての特段の対応は不要とも考えられる。しかし、考え方としてはその通りであるが、現実には、顧客の参画の内容と度合いによっては、全く報告が不要ということにはならないケースも少なくないと思われる。

一方、マクロな視点で見ると、ウォーターフォール型開発では、システムに要求される全ての機能が完成して初めて、顧客・経営層が知る（安心する）ことができることが基本であるのに対し、アジャイル開発では、たとえば、機能単位にリリースすることにより、顧客・経営層は、開発途中での完成状況を知ることができる。これにより、顧客・経営層は、ビジネス上のアクションを素早く採ることも可能となる。

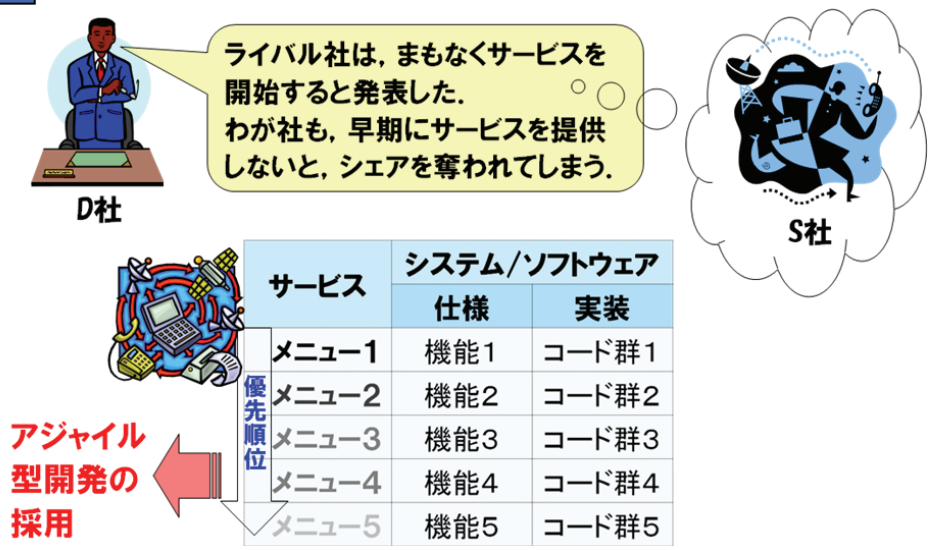
このことについて、図表　C-1 に示す簡単な例を用いて説明する。

同図は、D 社が、ビジネス上の必要から 5 個のメニュー項目が想定されるサービスの早期提供を迫られている状況を表す。同社は、要求の確定したメニュー項目の機能から順次開発・サービス提供を行うため、アジャイル開発の適用を決断したものとする。

図表　C-2 は、この開発プロジェクトの進捗状況のイメージを表す。左側がアジャイル開発、右側がウォーターフォール型開発の場合であり、下段が開発プロジェクトにおける実際の進捗度を、上段が顧客・経営層に見えるソフトウェアの完成度を、それぞれ表す。アジャイル開発においては、サービスメニューの完成ごとに、順次リリースするものとする。

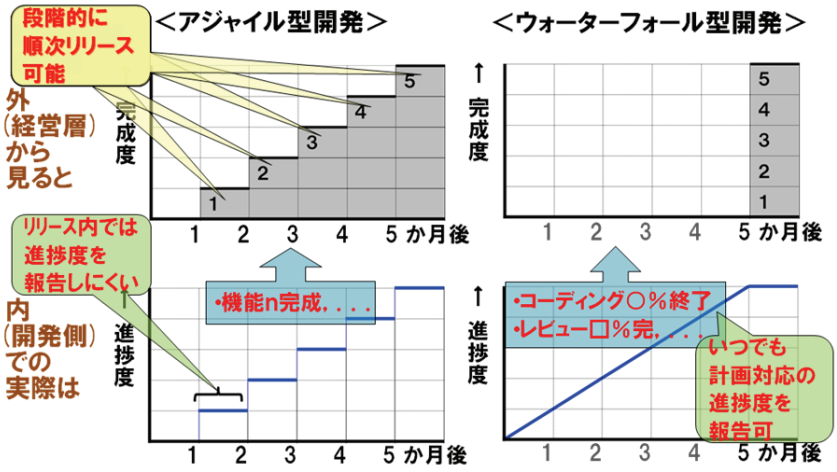
コラム

顧客・経営層に対する進捗の見える化（2/3）



図表 C-1　顧客・経営層によるアジャイル開発手法の採用判断の例

図表 C-2 に示すように、ウォーターフォール型開発のプロジェクトでは、求めに応じていつでもその時点での進捗度を明確に報告することができる。しかし、顧客・経営層がプロダクトの完成を実際に目にするのは、全ての機能の開発が完了した後となる。これに対して、アジャイル開発のプロジェクトでは、一般に、反復/リリース内では、明確な進捗度を報告しにくいことが多い。しかし、顧客・経営層は、一つの機能が完成する度に、それを実際に確認することができる。



図表 C-2　アジャイル型とウォーターフォール型における開発進捗度の見え方の特徴

コラム

顧客・経営層に対する進捗の見える化（3/3）

このように、アジャイル開発には開発進捗状況の見える化に関する得失がある。よって、重要なことは、開発形態の特徴に応じた見える化ということになる。

アジャイル開発の最大の特徴の一つは、ビジネス環境変化への俊敏な対応であるから、それが確実に実現できているか、またはできそうであるかの確認という観点での、開発進捗状況の見える化が求められる。たとえば、反復の単位で予定した機能の開発が完了せず、次の反復に持ち越しとなるケースがあり得るが、このようなケースの連続する回数や累積数は、ビジネス戦略上の予定時期までに所定の機能がリリースできるように開発が順調に進んでいるか否かを判断する材料の一つとなる。

また、アジャイル開発においては、一般に、反復の都度、コードを書き変えていくスタイルが採られるが、コードの整備のためにリファクタリングを繰り返す。このような状況が続くことにより、プロダクトの品質に重大な悪影響が及んでいるか、または及ぶ可能性があるかの早期見極めという観点での、プロダクト品質の見える化が求められる。たとえば、コードのエントロピー（乱雑さ、無秩序さ）増大状況は、プロダクト品質を見通す材料の一つとなる。

以上のことから、顧客・経営層は、アジャイル開発の採用を決断した時点で、開発プロジェクトの進捗状況の把握に関する十分な理解と覚悟が必要となる。また、アジャイル開発の特徴に応じた「見える化」項目を用いて開発プロジェクトとの円滑なコミュニケーションを図り、アジャイル開発採用の本来の目的が損なわれないように努める必要がある。

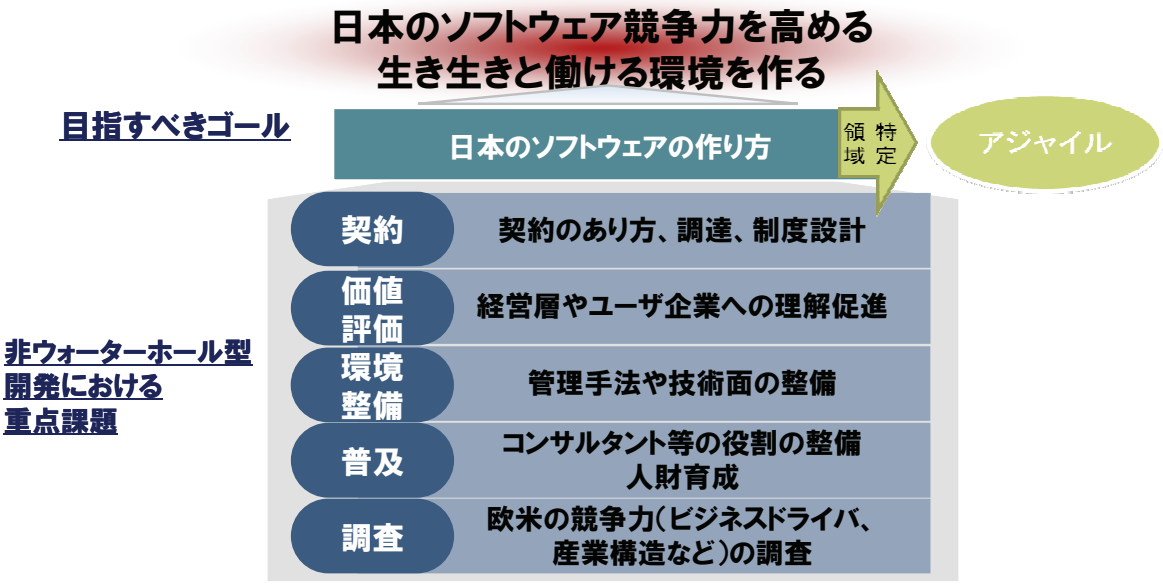
（IPA/SEC 山下　博之　記）

６　平成 22 年度の検討課題

IPA/SEC で平成 21 年度に実施した「非ウォーターフォール型開発の調査研究」では、国内における非ウォーターフォール型開発の採用に際して、次の５つの重点課題を指摘した。（図表 1-12 参照）

①契約：　契約のあり方、調達の制度設計
②価値評価：　経営層やユーザ企業へのアピール
③環境整備：　管理手法や技術面の整備
④普及：　コンサルタント等の役割の整備、人材育成
⑤調査：　欧米の競争力（ビジネスドライバ、産業構造）などの調査

今年度（平成 22 年度）の活動では、特に、①、②、③、④について取り組んだ。
昨年度（平成 21 年度）の事例収集をもとに、非ウォーターフォール型開発のモデル化（プロセスモデル、ビジネス構造モデル）を最初に行い、それを基に、①の契約、②の経営層へのアピール、③のスキル定義、④の人材育成につての論議を行い、本報告書をまとめた。



図表 1-12　非ウォーターフォール型開発の課題と目指すべきゴール

今年度はこれらの課題について検討するため、非ウォーターフォール型開発WG[23]で前年度の課題全体について討議を行った後、開発モデルPT[24]、技術スキルPT、契約問題PTの三つのPTを設置した。

（1）開発モデル PT の目的

①この活動におけるアジャイルの定義を行う。

②開発モデル（プロセスモデル、ビジネス構造モデル）を作成し、契約問題 PT、技術スキル PT への入力とする。

（2）技術スキル PT の目的

①課題－顧客・経営層へのアピールへの対応

・顧客・経営層が考慮すべき点とその検討。

②課題－非ウォーターフォール型開発に必要な技術の明確化－への対応

・エンジニアリング手法の検討

顧客側と開発側に必要な包括的エンジニアリング技術・プロジェクト運営技術・スキルの明確化。

③人材スキル・人材育成方法の明確化

・非ウォーターフォール型開発に必要な技術の明確化を受けての人材育成方法の検討

必要な技術・スキルの獲得方法の検討（人材像・スキル体系・人材育成方法）の検討。

・UISS、ITSS、ETSS への対応は、来年度以降、検討の予定とする。

（3） 契約問題 PT の目的

①非ウォーターフォール型開発に適した契約モデルの検討

・（日本における）非ウォーターフォール型開発に適した契約モデルの検討。

②非ウォーターフォール型開発に適した契約のひな型の検討

・非ウォーターフォール型開発に適した契約モデルに沿った契約のひな型の作成。

アジャイル開発は変化への対応を重視するため、開発開始時点では仕様を固定せず、ユーザとのコミュニケーションにより状況に応じて開発途中で仕様を決めていく。その結果、契約時点で成果物が不明確なため、「仕事の完成」に対して報酬を支払う請負

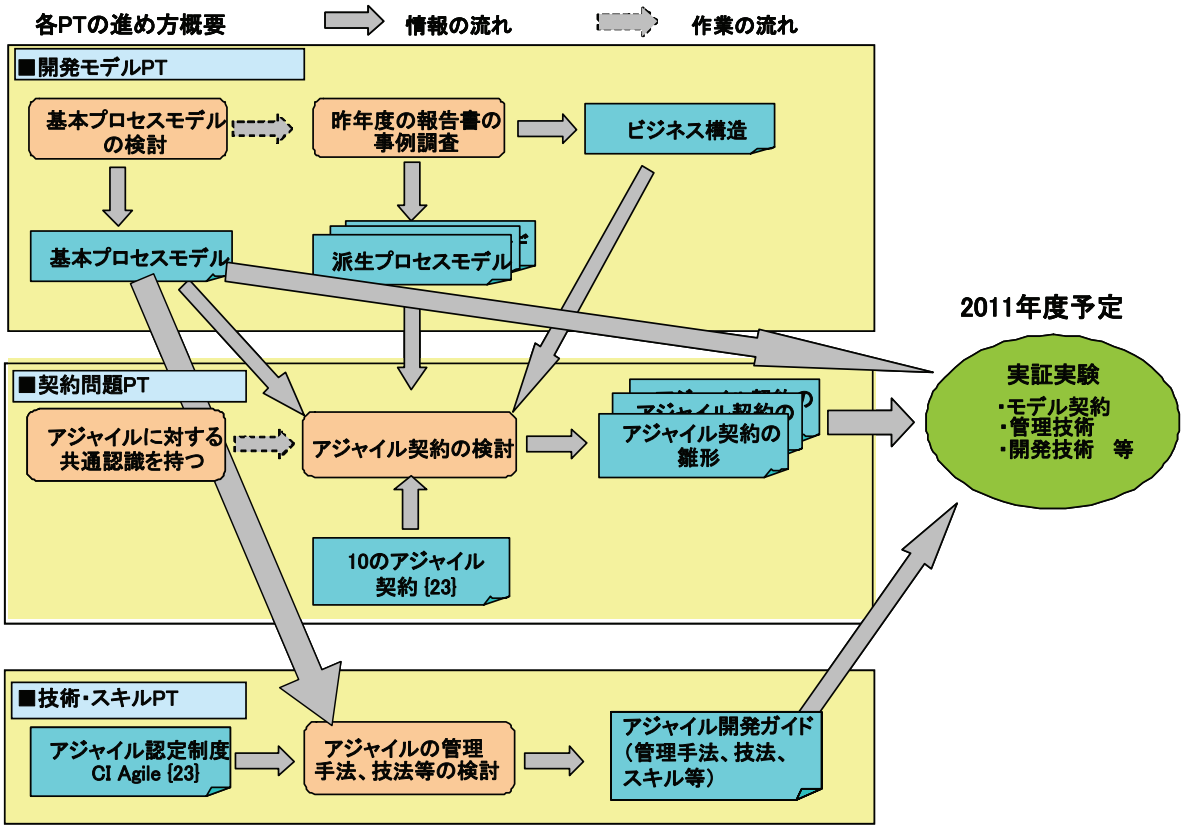
---

[23] WG（Working Group）
SEC（ソフトウェア・エンジニアリング・センター）では、WG をソフトウェア問題や課題が発生した際に、その対応のため特別かつ一時的に組織された作業グループと定義している。

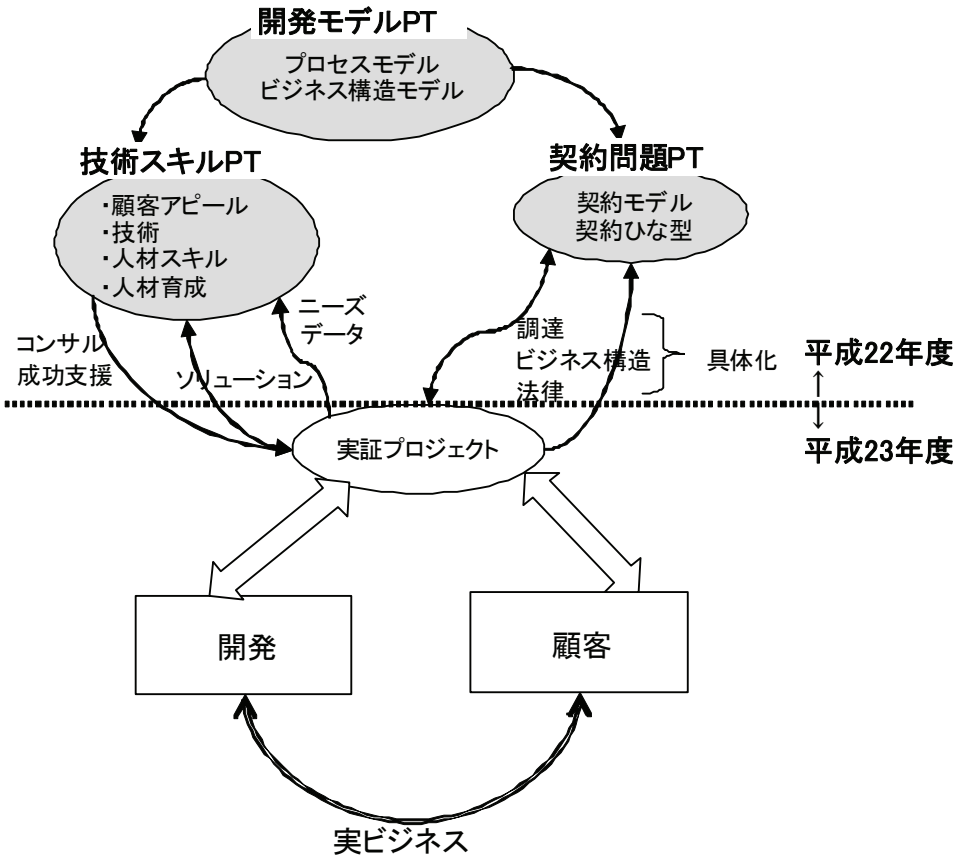
[24] PT（Project Team）
SEC（ソフトウェア・エンジニアリング・センター）では、PT を特定テーマの目的遂行のために、特別かつ一時的に成された WG 配下の組織と定義している。

契約はなじみにくい。

開発モデル PT、技術スキル PT、契約問題 PT 間の関係と各 PT の対象領域について、それぞれ図表 1-13、図表 1-14 に概要を示す。



図表 1-13　３つのＰＴの関係



図表 1-14　３つのＰＴの対象領域

７　まとめ

第一部では、本WG発足の背景、目的を述べ、そのコンテキストの中で「非ウォーターフォール型開発」および「アジャイル開発」を定義した。この活動の中では、現在の日本の産業構造の問題点を認識した上で、日本のソフトウェア競争力を高めるために、および、エンジニアがいきいきと働ける環境を作るために、これらの新しい開発手法を位置づけたいと考えている。

また、本検討ではアジャイル開発はすべての領域で最も優れた手法である、という立場をとっていないことに注意して欲しい。アジャイル開発には得意領域があり、その領域が現代の不確実性の高いビジネス環境の中で日々に拡大しているという認識である。

さらに、経営層への理解浸透が、次の一歩への大きな要素であるという認識から、企業情報システム開発における考え方のパラダイムシフトの意義を考察し、アジャイルへの懸念事項とともにまとめた。

参考文献

[1] IPA/SEC,「IT 人材白書 2010」, IPA/SEC, pp.143-154, 2010
[2] 経済産業省産業構造審議会情報経済分科会情報サービス・ソフトウェア小委員会,「情報サービス・ソフトウェア産業維新　～魅力ある情報サービス・ソフトウェア産業の実現に向けて～」, 経済産業省, 2006
[3] Mary Poppendieck& Tom Poppendieck, "Lean Software Development", Addison-Wesley, 2003, 訳: 平鍋健児/高島裕子/佐野建樹,「リーンソフトウェア開発~アジャイル開発を実践する 22 の方法」, 日経 BP 社, 2004
[4] Craig Larman, Agile and Iterative Development A Manager's Guide, Addison-Wesley Professional, 2003, 訳: 児高慎治郎ら,「初めてのアジャイル開発 ~スクラム、XP、UP、Evo で学ぶ反復型開発の進め方~」, 日経 BP 社, 2004
[5]「情報技術の革新とシステムインテグレーション事業の変容」, 産業研究（高崎経済大学附属研究所紀要）,第 44 巻第 1 号
http://www1.tcue.ac.jp/home1/sanken/pdf/44-1/44-1ishikawasekikawa.pdf
[6] Bertalanffy, L. Von, "General Systems Theory," George Braziller, 1968. 長野敬・太田邦昌（訳）「一般システム理論」, みすず書房, 1973.
[7] Weinberg, G. M., "An Introduction to General Systems Thinking," John Wiley & Sons, 1975. 松田武彦（監訳）「一般システム思考入門」, 紀伊國屋書店, 1979.
[8] Checkland, P. B., "Systems Thinking, Systems Practice," John Wiley & Sons, 1981. 高原康彦ほか（監訳）「新しいシステムアプローチ」, オーム社, 1985.
[9] 児玉公信,「情報システムサイクルと原要求の記述」, 日本情報経営学会誌, Vol. 28(2), pp. 77-87, 2007.
[10] Zachman, J. A., "A framework for information systems architecture", IBM Systems Journal, Vol. 26(3), pp. 276-292, 1987.
[11] 経営情報学会システム統合特設研究部会（編）「成功に導くシステム統合の論点–ビジネスシステムと整合した情報システムが成否の鍵を握る」, 日科技連出版社, 2005.
[12] 南波幸雄,「企業情報システムアーキテクチャ」, 翔泳社, 2009.
[13] Royce, W. W., "Managing the Development of Large Systems," Proceedings of IEEE WESCON, pp. 1-9, 1970.
[14] Boehm, B. W., "A Spiral Model of Software Development and Enhancement," ACM SIGSOFT Software Engineering Notes, Vol.11 (4), pp.14-24, 1989.
[15] Larman, C., "Agile and Iterative Development: A Manager's Guide," Reading, Addison-Wesley, 2003. ウルシステムズ（訳）「初めてのアジャイル開発–スクラム, XP, UP, Evo で学ぶ反復型開発の進め方」, 日経 BP, 2004.
[16] Brooks Jr., F. P., "The Mythical man-month: essays on software engineering," Anniversary edition, Addison-Wesley, 1995. 滝沢徹ほか（訳）「人月の神話–狼人間を打つ銀の弾はない」原著発行 20 周年記念増訂版, アジソン・ウエスレイ・パブリッシャーズ・ジャパン, pp. 256-258, 1996.
[17] Alexander, C., et al. "A Pattern Language," Oxford Univ. Press, 1977. 平田翰那（訳）「パタン・ランゲージ」, 鹿島出版, 1984.
[18] 真鶴町,「美の基準—デザインコード」, 真鶴町まちづくり条例, 1993.
[19] 藤村龍至,「グーグル的建築家像をめざして–［批判的工学主義］の可能性」, in 東浩紀・北田暁大（編）「特集アーキテクチャ」, 思想地図, Vol. 3, 日本放送出版協会, 2009.
[20] 濱野智史,「藤村龍至の『超線形設計プロセス』の限界とその突破」, Art and Architecture Review, Feb. 2010.
[21] Kuhn, T. S.,"The Structure of Scientific Revolutions," The University of

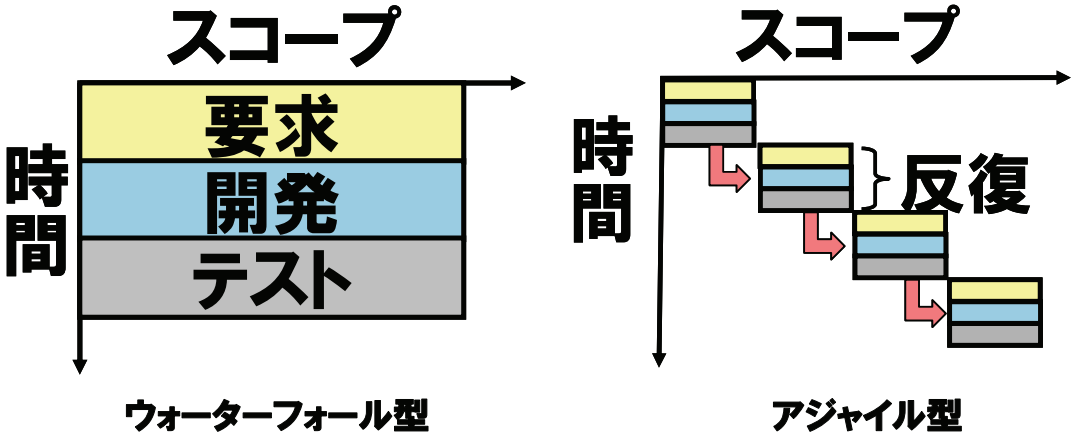
Chicago Press, 1962. 中山茂（訳）「科学革命の構造」，みすず書房，1971.
[22] Schiel, J., “Enterprise-Scale Agile Software Development” CRC Press, 2009.
[23] 「10 のアジャイル契約」
Peter Stevens，10 Contracts for your next Agile Software Project，2009
http://agilesoftwaredevelopment.com/blog/peterstev/10-agile-contracts
「アジャイル認定制度」（CI Agile）
http://www.icagile.com

第二部　アジャイル開発のモデル

１　アジャイル開発の標準モデル

アジャイル開発は、顧客の要求にしたがって優先度の高い機能を、要求・開発・テスト・リリースを[25]短い期間で繰り返しながらシステム全体を構築していくことが標準となっている。

原則として、事前に開発の詳細な計画は作らず、１週間から４週間といった一定の短い周期で要求・開発・テストを繰り返しながら、動作可能なソフトを作り上げる。開発の繰り返し単位を「反復」と呼ぶ。



図表 2-1　アジャイル開発を単純化した標準モデル

[25] ここでの「要求、開発、テスト」等の用語は、イメージを持って頂くことを主目的に曖昧なまま使用している。SLCP やアジャイルの用語との対応については後述。

２　アジャイル開発の開発モデル

非ウォーターフォール型開発に適した契約モデル等を検討するにあたって、今後の課題を明確にすることを目的に、IPA/SEC で昨年度（平成 21 年度）実施した「非ウォーターフォール型開発の調査研究」において調査した非ウォーターフォール型開発の 17 社 22 事例をベースに、アジャイル開発の実績を調査しモデル化した。

このモデルは、以下の２つからなる。

①プロセスモデル

②ビジネス構造モデル

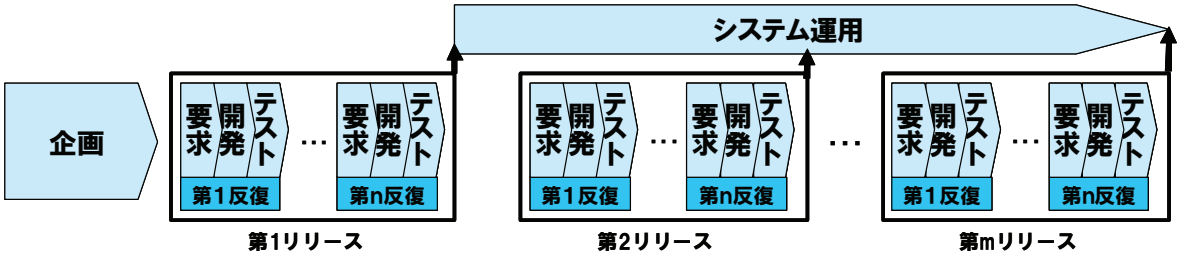
２．１　アジャイル開発のプロセスモデル

非ウォーターフォール型開発に適した契約モデルの検討にあたっては、開発プロセスをモデル化し、それに沿って契約形態を考える必要がある。

IPA/SEC で昨年度（平成 21 年度）に実施した「非ウォーターフォール型開発の調査研究」で調査した非ウォーターフォール型開発の 17 社 22 事例をもとに以下に示す３つのプロセスモデルにまとめた。

モデル１を基本プロセスモデルとし、モデル２、モデル３はその派生プロセスモデルとした。

（１）　プロセスモデル１（図表 2-2）

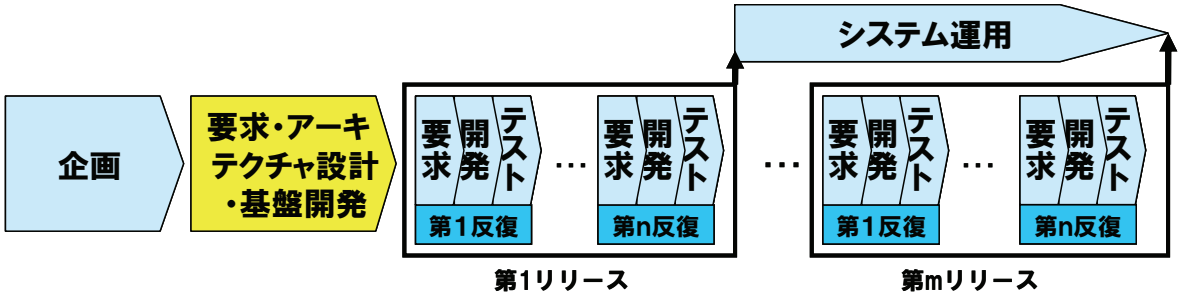


図表 2-2　アジャイル開発のプロセスモデル１

システム全体の要求・アーキテクチャ設計・基盤開発を行わずに、直ぐに第１リリースの開発に取りかかるモデル。

調査事例では、11 事例がこのプロセスモデルであった。

（2） プロセスモデル２（図表 2-3）



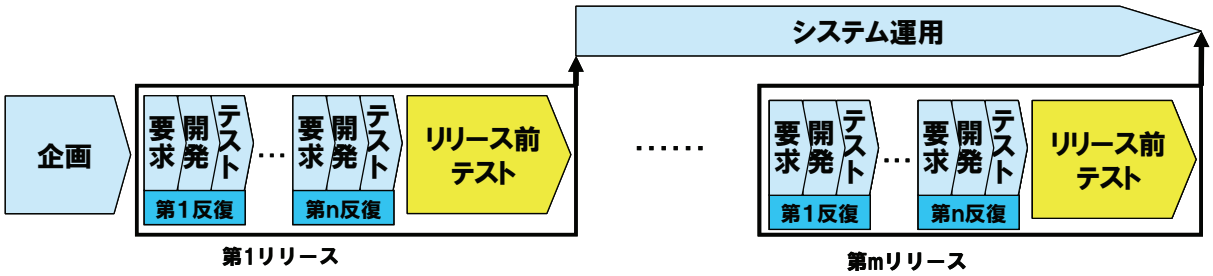
- 比較的大規模システム／新規開発で全体のシステム構造が不明確なケースなど

図表 2-3 アジャイル開発のプロセスモデル２

第１反復/第１リリースの前に、システム全体の大まかな要求定義・アーキテクチャ設計・基盤開発を行った後に、反復開発に着手するモデル。

調査事例では、3 事例がこのプロセスモデルであった。

（3） プロセスモデル３（図表 2-4）



- アジャイル開発では反復ごとにリリースできる品質までテストを行うことが原則だが、各リリース工程前に行う 重点的なテストを実施することがある。
- リリースは複数回繰り返される

図表 2-4 アジャイル開発のプロセスモデル３

アジャイル開発では反復ごとにリリースできる品質までテストを行うことが原則だが、各リリース前に、品質を担保するために、重点的なテストを実施するモデル。

調査事例では、8 事例がこのプロセスモデルであった。

以上の 3 プロセスモデルは、現実のプロジェクトを抽象化したものである。実際の各事例とプロセスモデルとの対応を付録１に示す。

（4） アジャイル開発のプロセスモデルで使用されている用語の説明

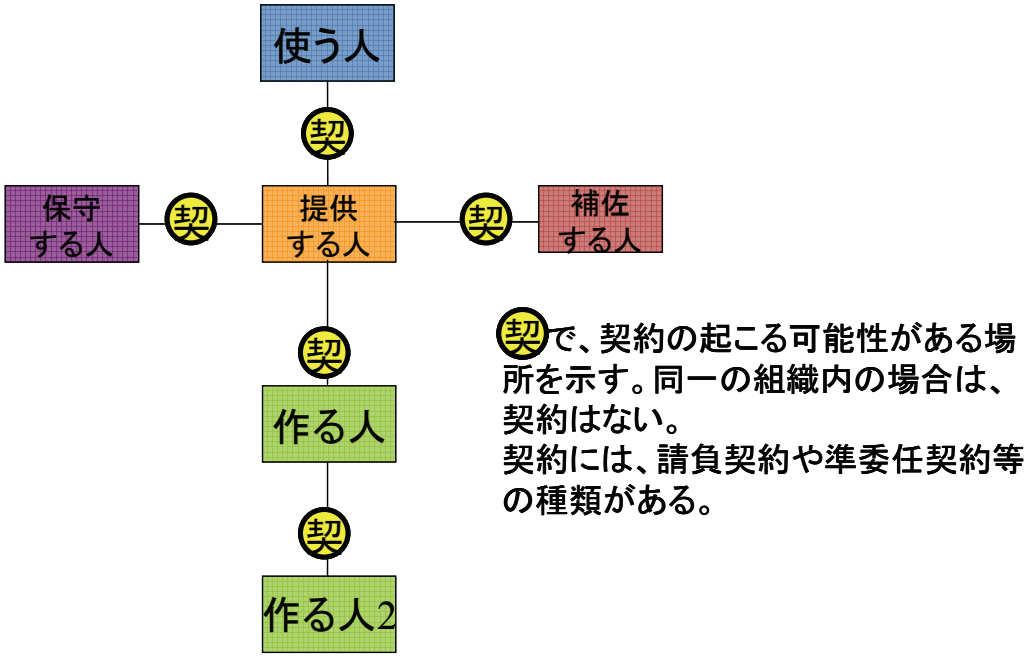
アジャイル開発のプロセスモデルで使用されている用語の説明を図表 2-5 に示す。

Agile開発の用語	用 語 の 解 説
企画	ウォーターフォール型開発の「企画プロセス」と「要求定義プロセス」と同様の内容。 ビジネスゴールをもとに、プロダクト／プロジェクトのコンセプト、システム化の目的や目標となるゴール、ビジネス要求に紐付けされたシステム要求と主要な機能について立案し、ステークホルダー間で合意を形成する。
反復	1～4週間の連続した期間で行う要求・開発・テストの繰り返しの単位を反復と呼ぶ。この要求・開発・テストを繰り返す固定された時間枠を「タイム・ボックス」（時間枠）と呼ぶ。作業が時間的に間に合わない場合でもタイム・ボックス自体の延長はしない。 反復を繰り返すことで、部分的に稼働する最終システムが作り出され、連続する反復は直前の反復の作業に基づいて最終製品が完成するまでシステムを進化／洗練させていく。 Scrumでは「スプリント」、 XPでは、「イテレーション」と呼ぶ。
リリース	動作するソフトウェアを顧客（理想的にはシステムの利用者）に提供することをリースと呼ぶ。 顧客は、リリースされたソフトウェアを検証することで、開発の実際の進捗度を確認することができ、要求仕様が正確に成果物に反映していることを確認でき、実際に機能の一部を使ってみることで、要求仕様そのものの検証が行える。
要求	各反復の最初に、顧客（理想的にはシステムの利用者）と開発者は、対話・協調しながら今回の反復で何を開発し、何を開発しないのかについての意思決定を行う。そこで決定された今回の反復で開発する内容を指す。 顧客と開発者は要求の中から、今回の反復で、開発・実装する機能を選択する。 通常、顧客は、最も必要な優先度の高い機能から順に選択し、開発者は、選択された機能を優先度順に実装していく。
開発	要求で決定された機能の設計、プログラミング（実装）、単体テスト、レビュー、ふりかえり等の一連の作業を指す。アジャイル開発では、不要なドキュメントは作らず、動作するソフトウェアを重視する。 レビューには、コードレビューやテストレビュー、顧客（理想的にはシステムの利用者）も参加したプロダクトのレビューなどがあり、設計や実装に対するフィードバックを目的に行う。 ふりかえりとは、反復の最後にチームメンバーが集まり、チームのやり方やチームワークを点検し、改善する特別なミーティングのことで、次の反復に役立たせて成果へとつなげる。 XPにおける主なプラクティスとして、「シンプルデザイン」、「テスト駆動開発」、「頻繁なリファクタリング」、「ペアプログラミング」、「共同所有権」、「継続的インテグレーション」、「コーディング規約」等が上げられる。 Scrumにおける主なプラクティスとして、「スプリント計画」、「スプリントバックログ グラフの作成」、「自律的な組織化チーム」、「スクラムミーティング」、「日次ビルド」、「スプリントレビュー」等が上げられる。
テスト	反復で作成されたプログラムによって要求で合意した機能が実現しているか、確認するための受入れテストを指す。受入れテストの内容は、顧客（理想的にはシステムの利用者）が定義する。 顧客はこのテストにより、成果物の確認と進捗の確認を行う。 受入れテストでは、単体テスト後に検証される機能より、粒度は大きくなる。
アーキテクチャ設計・基盤開発	開発を始めるまえに、最低限のシステム全体の大まかな要求定義・アーキテクチャ設計・基盤開発を行う作業を指す。 （全体の基盤設計・開発を第一反復／リリースとして行う場合もある。） 反復に入る前に、変更のコストが大きい意志決定（アーキテクチャ設計など）を行うことがある。
リリース前テスト	アジャイル開発では反復ごとにリリースできる品質までテストを行うことが原則だが、各リリース工程前に重点的なテストを実施することがある。 納品や次のリリースの製造に向け、各反復の最後に行うテストの最後に追加して、品質保証の観点でテストを行う。
システム運用	システムを実際に運用する

図表 2-5 アジャイル開発のプロセスモデルで使用されている用語の説明

２．２　アジャイル開発のビジネス構造モデル

平成 21 年度の「非ウォーターフォール型開発に関する調査」において実施した 17 社、22 事例の調査結果から導きだした、アジャイル開発のビジネス構造モデルの基本パターン（ソフトウェア開発に関わる役割と契約の起こる位置）を図表 2-6 に示す。



図表 2-6　ビジネス構造モデルの基本パターン例

・ 使う人　　⇒　システムを実際に使用する人（システムの利用者、エンドユーザ）。
・提供する人　⇒　開発されたシステムやサービスを、使う人に提供する人。
・作る人　　⇒　システムの開発者
・作る人２　　⇒　作る人の会社と契約している、作る人と一緒に開発する人。
（作る人の会社とは別会社が多い）
・保守する人　⇒　リリースされたシステムを保守する人。
（作る人と重なることが多い）
・補佐する人　⇒　アジャイル開発の進め方等で、開発プロジェクトを支援する人。
（コンサルタントやファシリテーターが多い）

図表 2-6 から、アジャイル開発を行う際に契約の起こり得る箇所を抽出できる。上記の四角で囲まれた役割間で契約が起こり得るが、役割が同じ会社内の場合には契約は起こらない。また、現時点で契約形態としてよく知られているのは、「請負契約」や「準委任契約」である。

上記基本パターンをもとに、各事例を分析した結果を付録２に示す。

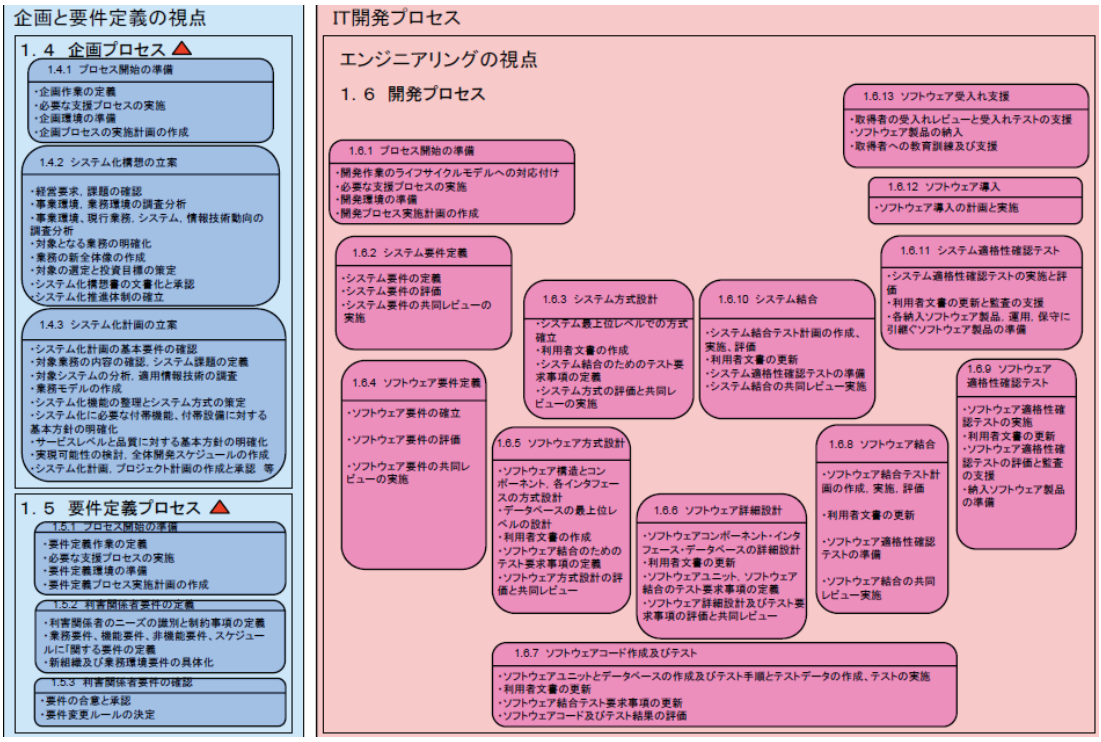
３　アジャイル開発プロセスの SLCP へのマッピング

本来、アジャイル開発は、システム開発に関する既存の価値観、原則を大きく置き換える考え方である。よって、既存のフレームワークと言葉をマッピングすることは、誤解を招きかねない作業である。しかしながら、既存の用語に馴染んだ方にもアジャイル開発を理解して頂けるためのガイドとして、この章を設けた。

ウォーターフォール型開発プロセスの視点からアジャイル開発プロセスを俯瞰するために、共通フレーム 2007 をベースとし、SLCP（software life cycle process）に沿ったシステム開発に関連する作業とアジャイル開発プロセスとのマッピングを行った。

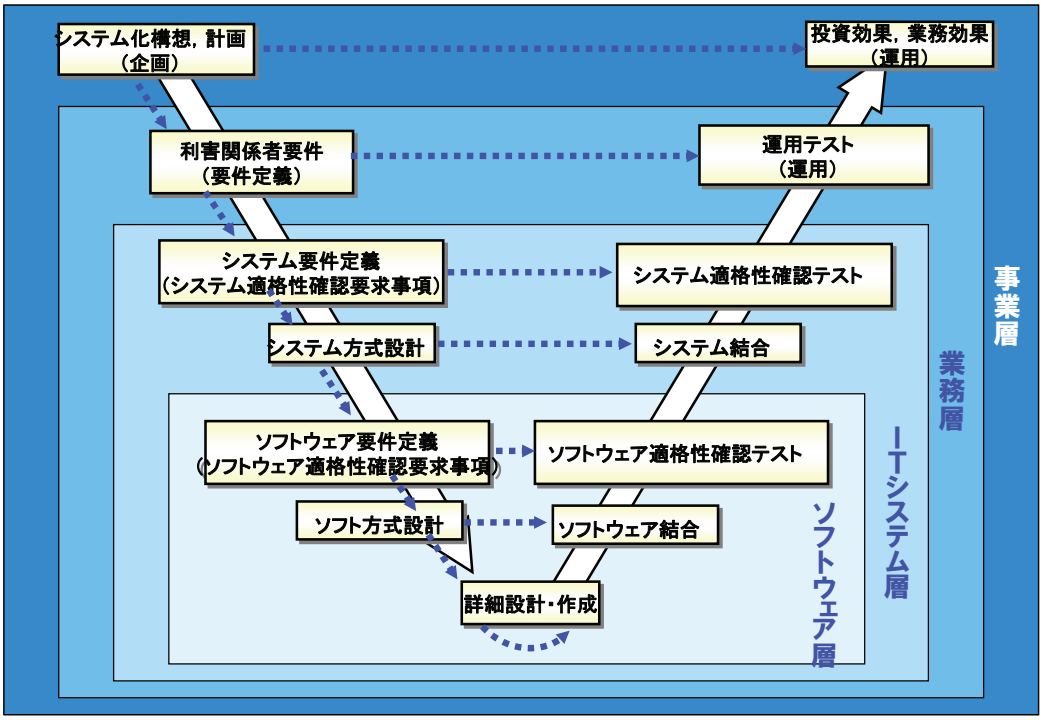
３．１　共通フレーム 2007 体系　抜粋

SLCP を表す主要な図として、「共通フレーム 2007 体系　抜粋」を図表 2-7 に、「適格性確認要求事項と適格性確認テストとの関係」を図表 2-8 に示す。



出典：「共通フレーム 2007　第 2 版」（IPA/SEC, 2009）

図表 2-7　　共通フレーム 2007 体系　抜粋



出典：「共通フレーム 2007 第 2 版」（IPA/SEC, 2009）

図表 2-8　適格性確認要求事項と適格性確認テストとの関係

３．２　SLCP とデータ白書 2009 で使用されている開発工程のマッピング

データ白書 2009 で使用されている開発工程の名称と、SLCP-共通フレーム 2007 との対応関係を図表 2-9 に示す。

「工程」列には、データ白書 2009 で使用している工程名称を示している。「SLCP プロセス/ アクティビティ」と「SLCP の定義」列で SLCP との対応で工程の定義を示している。

工程	SLCP プロセス/ アクティビティ	SLCP の定義
システム化計画	システム化計画の立案	企画者は、システム計画の基本要件の確認を行い、対象業務の確認、システムが実現している機能等の確認と整理により、システム課題を定義、業務機能をモデル化する。モデルからシステム化機能の整理、システム方式とシステム選定方針の策定、付帯機能・設備やサービスレベル及び品質の基本方針の明確化、プロジェクトの目標設定、実現可能性の検討、全体開発スケジュールの作成、費用とシステム投資効果の予測を行い、具体化したシステムに対する前提条件を整理し、システム化計画として文書化し承認する。複数プロジェクトがある場合はプロジェクト計画の作成と承認を得る。
要件定義	システム要件定義 ソフトウェア要件定義	開発者は、システム及びソフトウェアに関する要件について技術的に実現可能であるかを検証し、システム設計が可能な技術要件に変換し、システム要求仕様と確立したソフトウェア要件を文書化する。また、設定した基準を考慮して、システム要件、ソフトウェア要件を評価し文書化する。さらに、共同レビューを行う。
基本設計	システム方式設計 ソフトウェア方式設計	開発者は、ハードウェア構成品目、ソフトウェア構成品目及び手作業を明確にし、システム方式及び各品目に割り振ったシステム要件を文書化する。また、ソフトウェア品目に対する要件をソフトウェア方式に変換する。最上位レベルの構造とソフトウェアコンポーネントを明らかにし、データベースの最上位レベルでの設計、利用者文書の暫定版の作成、ソフトウェア結合のための暫定的なテスト要求事項及び予定等を明らかにする。方式設計の評価と共同レビューを実施する。
詳細設計	ソフトウェア詳細設計	開発者は、ソフトウェア品目の各ソフトウェアコンポーネントに対して詳細設計を行う。ソフトウェアコンポーネントは、コーディング、コンパイル及びテストを実施するユニットレベルに詳細化する。また、ソフトウェアインターフェイス、データベースの詳細設計、必要に応じて利用者文書を更新し、ユニットテスト、結合テストのためのテスト要求事項及び予定を定義する。評価及び共同レビューを実施する。
製作	ソフトウェアコード作成及びテスト	開発者は、ソフトウェアユニット及びデータベースを開発する。また、それらのためのテスト手順及びデータを設定する。さらに、テストを実施し、要求事項を満足することを確認する。これらに基づいて、必要に応じて利用者文書等の更新を行う。また、ソフトウェアコード及びテスト結果を評価する。

出典：「データ白書 2009－工程の呼称と SLCP マッピング－」（IPA/SEC, 2009）

図表 2-9　SLCP とデータ白書 2009 で使用されている開発工程のマッピング(1/2)

工程	SLCP プロセス/ アクティビティ	SLCP の定義
結合テスト	ソフトウェア結合 システム結合	開発者は、ソフトウェアユニット及びソフトウェアコンポーネントを結合して、ソフトウェア品目にするための計画を作成し、ソフトウェア品目を完成させる。また、結合及びテストを行う。完成したソフトウェア品目と合わせてハードウェア品目、手作業や他システム等とあわせてシステムに結合、要件を満たしているかをテスト、システム適格性確認テスト実施可能状態であることを確認する。必要に応じて利用者文書等の更新を行う。テストの評価と共同レビューを実施する。
総合テスト （ベンダ確認）	ソフトウェア適格性確認テスト システム適格性確認テスト	開発者は、ソフトウェア品目の適格性確認要求事項及びシステムに関して指定された適格性確認要求事項に従って、適格性確認テスト及び評価を行う。必要に応じて利用者文書等の更新を行う。また、監査の実施と支援をする。
総合テスト （ユーザ確認）	ソフトウェア導入支援 ソフトウェア受け入れ支援	開発者は、契約の中で指定された実環境にソフトウェア製品を導入するための計画を作成し、導入する。 開発者は、取得者によるソフトウェア製品の受け入れレビュー及びテストを支援する。また、契約で指定するとおりに、取得者に対し初期の継続的な教育訓練及び支援を提供する。
フォロー （運用）	運用プロセス	ソフトウェア製品の運用及び利用者に対する運用支援を行う。運用者は、このプロセスを管理するために具体化した管理プロセスに従って、運用プロセスの基盤となる環境を確立する、など。

出典：「データ白書 2009－工程の呼称と SLCP マッピング－」（IPA/SEC, 2009）

図表 2-9　SLCP とデータ白書 2009 で使用されている開発工程のマッピング(2/2)

３．３　SLCP とアジャイル開発プラクティスとのマッピング

アジャイル開発を理解しやすくするためのガイドとして、敢えて、SLCP[26]のプロセス、アクティビティとアジャイル開発のプロセス、プラクティスのマッピングを示すと、図表 2-10 のようになる。

<table>
<tr><th colspan="2">SLCP</th><th>データ白書2009</th><th colspan="2">アジャイル開発</th></tr>
<tr><th>プロセス</th><th>アクティビティ</th><th>開発工程</th><th colspan="2">プロセス</th><th>プラクティス</th></tr>
<tr><td>企画プロセス</td><td>・プロセス開始の準備
・システム化構想の立案
・システム化計画の立案</td><td>システム化計画</td><td colspan="2">企画</td><td>Scrum　・ゲーム前計画</td></tr>
<tr><td>要件定義プロセス</td><td>・プロセス開始の準備
・利害関係者要件の定義
・利害関係者要件の確認</td><td rowspan="6">要件定義</td><td rowspan="8">反復</td><td rowspan="2">要求</td><td rowspan="2">XP　・リリース計画ゲーム
・イテレーション計画ゲーム

Scrum　・スプリント計画</td></tr>
<tr><td rowspan="7">開発プロセス</td><td rowspan="7">・プロセス開始の準備
・システム要件定義
・ソフトウェア要件定義

・システム方式設計
・ソフトウェア方式設計

・ソフトウェア詳細設計

・ソフトウェアコード作成及びテスト

・ソフトウェア結合（ソフトウェア）
・システム結合（ソフトウェア）
・ソフトウェア結合（テスト）
・システム結合（テスト）

・ソフトウェア適格性確認テスト
・システム適格性確認テスト

・ソフトウェア受入れ支援

・ソフトウェア導入</td></tr>
<tr><td>基本設計</td><td rowspan="6">開発/テスト</td><td rowspan="6">XP　・システムのメタファ
・シンプルデザイン
・テスト駆動開発
・頻繁なリファクタリング
・ペアプログラミング
・共同所有権
・コーディング規約
・受け入れテスト

Scrum　・スプリントバックログ
グラフの作成
・自律的な組織化チーム
・スクラムミーティング
・1日以内の障害除去
・共通の部屋
・日次ビルド
・スプリントレビュー</td></tr>
<tr><td>詳細設計</td></tr>
<tr><td>製作</td></tr>
<tr><td>結合テスト</td></tr>
<tr><td>総合テスト
（ベンダ確認）</td></tr>
<tr><td>総合テスト
（ユーザ確認）</td></tr>
<tr><td>運用プロセス</td><td></td><td>フォロー（運用）</td><td colspan="2">運用</td><td></td></tr>
</table>

出典：・SLCP　　「共通フレーム 2007 第 2 版」（IPA/SEC, 2009）
・開発工程　「データ白書 2009－工程の呼称と SLCP マッピングー」
（IPA/SEC, 2009）
・アジャイル開発プラクティス
「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC, 2010）

図表 2-10　SLCP とアジャイル開発プラクティスとのマッピング

[26] SLCP におけるアクティビティ、さらにそれを細分化した「タスク」は何をすべきか（What to do）を表す。これに対して、アジャイル開発におけるプラクティスは主に、どのようにするか（How to do）を表す。このように、両者は全く異なる考え方に基づいており、ここで示すマッピングは参考でしかない。

４　まとめ

第一部では、国内の事例を基に、アジャイル開発の「プロセスモデル」および「ビジネス構造モデル」を作成した。さらに、その際に理解の助けとなるよう、ウォーターフォール型開発で使われている言葉の定義とのマッピングを行った。

また、アジャイル開発に馴染みのない方にもイメージを持ってもらえ、契約問題や技術スキルの検討の前提となる共通コンセプトが示せるように留意したつもりである。

前年度の収集事例について、プロセスモデルおよびビジネス構造モデルの観点による分析を、付録１、付録２に添付する。

付録１　調査事例とプロセスモデルの対応一覧

調査事例とプロセスモデルの対応一覧[27]を付図表 1-1 に示す。

モデル名	調査事例
モデル1	小売業・ユニケージ、社内版SNS、OSS版SNS、サプラチェーンマネジメントシステム、携帯ソーシャルゲーム、携帯向けブログ、パッケージ、教育機関向け統合業務PKG、株取引Webアプリ、共通EDI、開発案件管理、製造業向けプロトタイプシステム
モデル2	研修運営システム、プロジェクト管理システム、 アプリPF開発
モデル3	共通認証システム、教務Webシステム、 ミドルウェア開発、生産管理システム、Webメディア開発、 アジャイル開発支援、検索エンジン開発、 プラント監視制御計算機システム

出典：「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC, 2010）」

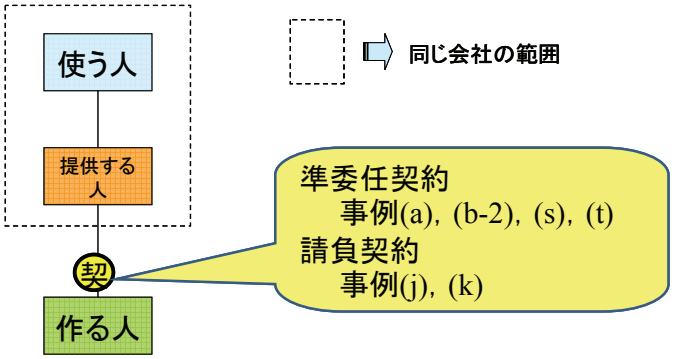
付図表 1-1　事例とモデルの対応一覧

[27] 調査したのは 17 社 22 事例であるが、モデル 1 の SNS 事例が 2 つに分かれているためここでは 17 社 23 事例となっている。

付録２　ビジネス構造モデルによるアジャイル開発事例の分析例

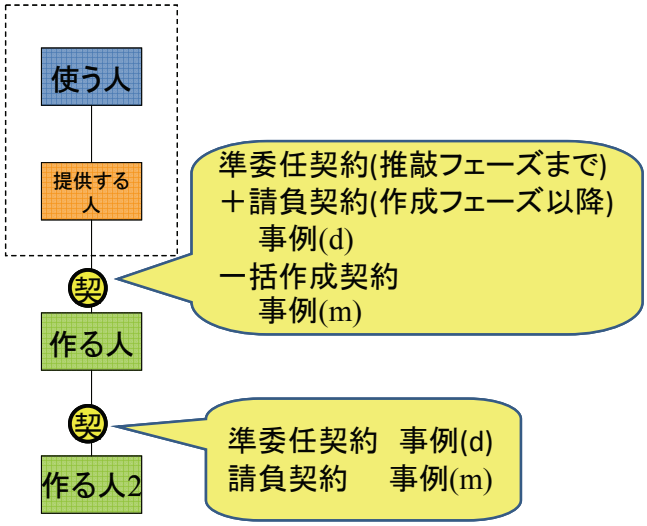
平成 21 年度の「非ウォーターフォール型開発に関する調査」において実施した 17 社、22 事例の調査結果に基づくアジャイル開発のビジネス構造を分析した結果、10 パターンに分類された。

（1）ビジネス構造モデル パターン１　（６事例）



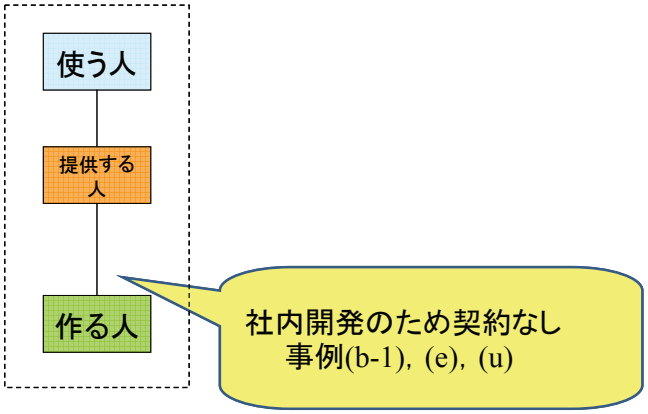
付図表 2-1　ビジネス構造モデル パターン１

（2）ビジネス構造モデル パターン２　（２事例）



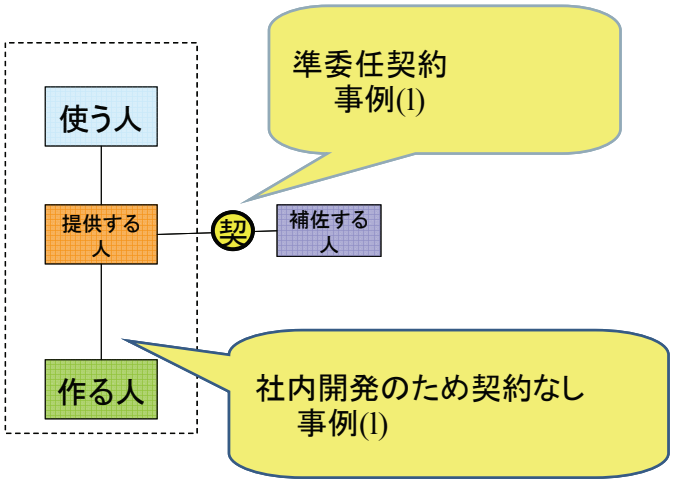
付図表 2-2　ビジネス構造モデル パターン２

（3）ビジネス構造モデル パターン３　（３事例）



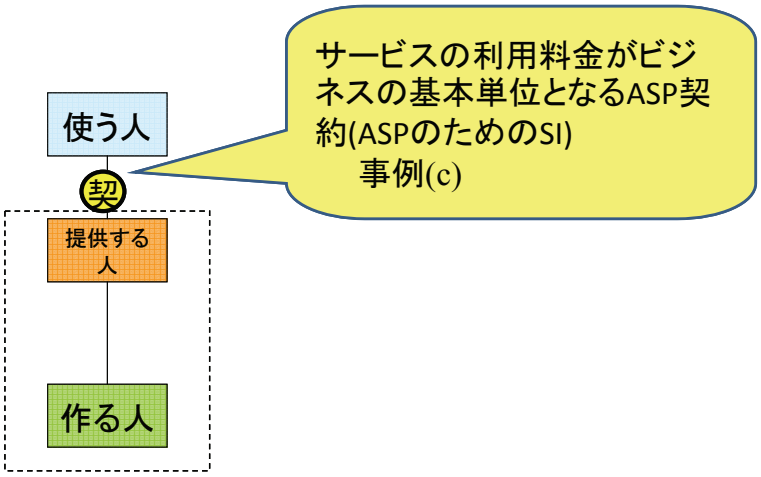
付図表 2-3　ビジネス構造モデル パターン３

（4）ビジネス構造モデル パターン４　（１事例）



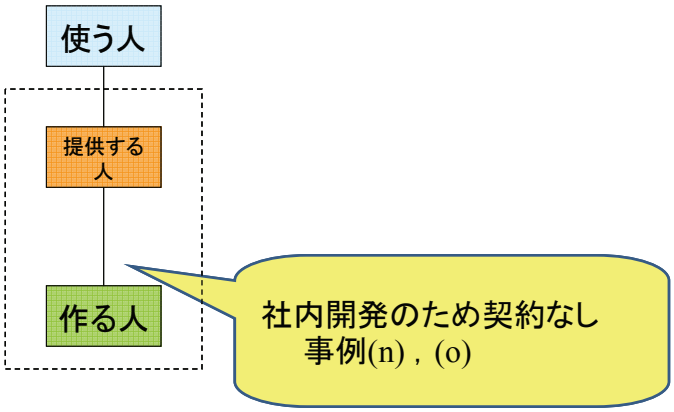
付図表 2-4　ビジネス構造モデル パターン４

（５）　ビジネス構造モデル パターン５　（１事例）



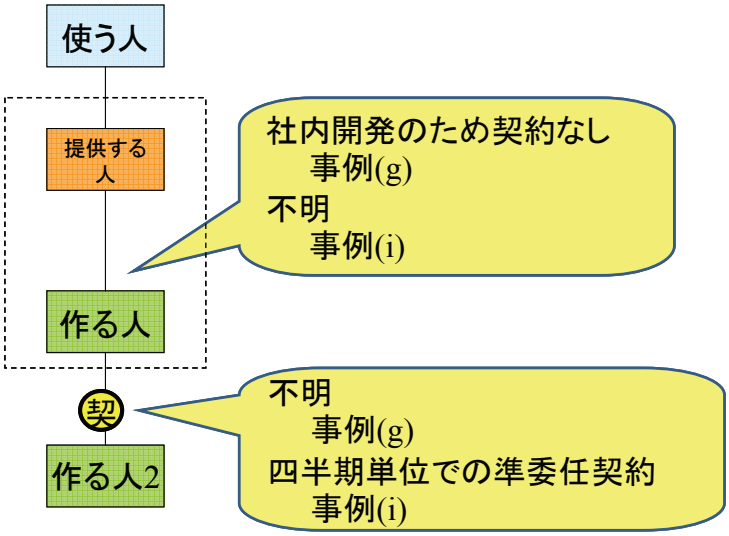
付図表 2-5　ビジネス構造モデル パターン５

（６）ビジネス構造モデル パターン６　（２事例）



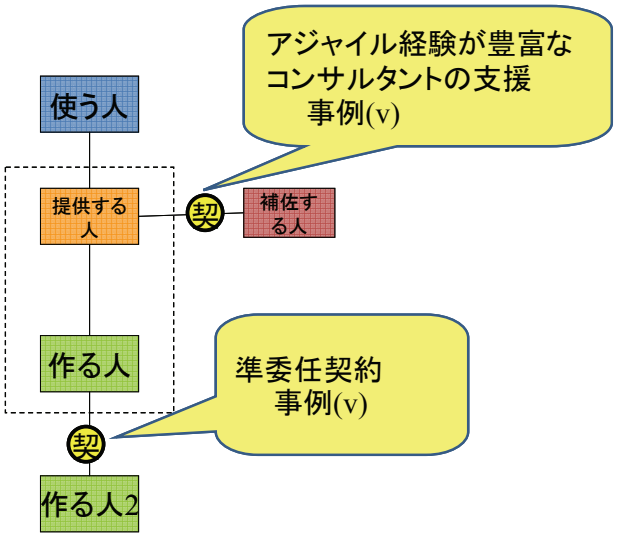
付図表 2-6　ビジネス構造モデル パターン６

（7）ビジネス構造モデル パターン７　（２事例）



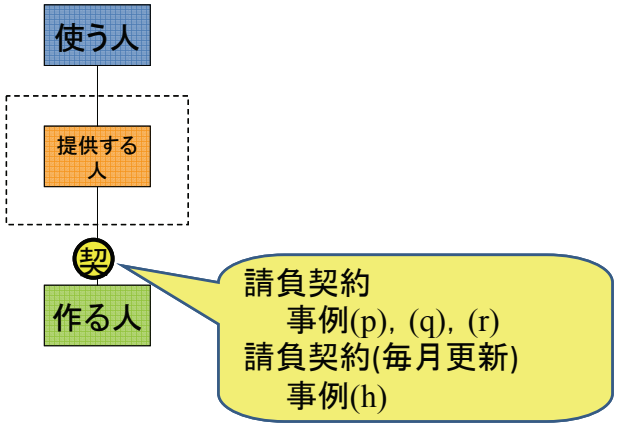
付図表 2-7　ビジネス構造モデル パターン７

（8）ビジネス構造モデル パターン８　（１事例）



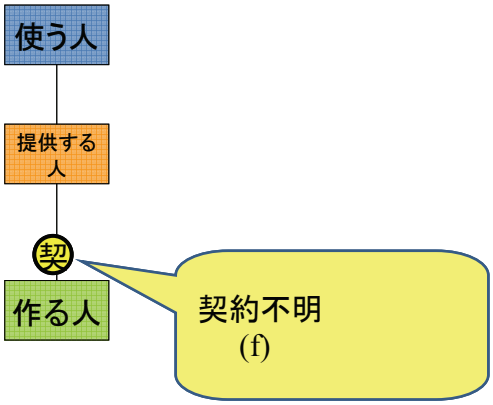
付図表 2-8　ビジネス構造モデル パターン８

（9）ビジネス構造モデル パターン 9 （4 事例）



付図表 2-9 ビジネス構造モデル パターン 9

（10）ビジネス構造モデル パターン 10 （1 事例）



付図表 2-10 ビジネス構造モデル パターン 10

基本パターンによるアジャイル開発事例の分析における、調査事例と契約の対応一覧[28]を付図表 2-11 に示す。

契約方式	調査事例
請負契約	(h)携帯向けブログシステム※毎月更新、(j)共通認証システム、(k)プロジェクト管理システム、(m)教務Webシステム※パートナー契約有り、(p)ミドルウェア開発、(q)株取引Webアプリ、(r)プラント監視制御計算機システム
準委任契約	(i)パッケージ※四半期単位、(l)アプリPF開発、(s)生産管理システム、(a)小売業・ユニケージ、(b-1)社内版SNS、(b-2)OSS版SNS、(t)Webメディア開発、(v)共通EDI
準委任契約(推敲フェーズまで)＋請負契約(作成フェーズ以降)	(d)研修運営システム
派遣契約	(o)検索エンジン開発
サービスの利用料金がビジネスの基本単位となる、ASP契約	(c)サプラチェーンマネジメントシステム
契約なし	(g)携帯ソーシャルゲーム※パートナー契約有り、(n)教育機関向け統合業務PKG、(e)開発案件管理、(u)アジャイル開発支援
不明	(f)製造業向けプロトタイプシステム

出典：「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC, 2010）」

付図表 2-11　事例と契約の対応一覧

分類された 10 パターンは、平成 21 年度に実施した「非ウォーターフォール型開発に関する調査」から分析したものであり、一般的なアジャイル開発を網羅している訳ではない。

[28]　調査したのは 17 社 22 事例であるが、準委任契約の SNS 事例が 2 つに分かれているため、ここでは 17 社 23 事例となっている。

第三部　非ウォーターフォール型開発における技術及びスキル

１　非ウォーターフォール型開発における技術スキル検討の趣旨

１．１　検討の背景

IPA/SEC では、昨年度（平成 21 年度）に実施した「非ウォーターフォール型開発の調査研究」に引き続き、本年度は、国内における非ウォーターフォール型開発の採用に際しての課題として、次の５つの重点テーマについて検討してきた。

①契約：契約のあり方、調達の制度設計
②価値評価：経営層やユーザ企業へのアピール
③環境整備：管理手法や技術面の整備
④普及：コンサルタント等の役割の整備、人材育成
⑤調査：欧米の競争力（ビジネスドライバ、産業構造）などの調査

第三部では、これらの重点課題の中から、本年度（平成 22 年度）技術スキル PT が検討を行ってきた、②価値評価：経営層やユーザ企業へのアピール、③環境整備：管理手法や技術面の整備、④普及：人材育成について記述する。

１・２　非ウォーターフォール型開発における技術スキル検討の目的

本 PT では、次の３つのサブテーマの検討内容を次のように設定した。

（１）経営層にとっての価値

従来のソフトウェア開発においては、多くの企業の経営者は開発費用と効果目標を設定した後は、プロジェクトチームに任せっきりであった。そして、そのような企業においては、作業の多くは発注担当者に丸投げされ、いわば経営者自身の関与は極めて少なかったといえる。しかし、経営環境の激変、戦略変更、業務改革などといった経営の不確実性は一段とに増大しており、もはや、経営層としては放置できなくなっている。

実際にソフトウェアにおける大きな品質トラブルの背景には、経営層の意思決定の遅れや、理解不足、さらに関係部門との経営上の調整不足などが見られ、経営者自体の能力不足の結果として責任を問われかねないケースも少なくない。経営層がどのようにシステム開発にかかわっていくべきかの議論なくして、今後の情報システム開発はあり得ない。こうした中で、どのような開発手法を採用するかも、開発チームだけではなく、経営層の問題となってきた。したがって、非ウォーターフォール型開発についても十分に理解を深める必要がある。とりわけ、経営層にとっての意義と価値を理解することが不可欠であろう。このような状況を踏まえ、新しいマネジメント手法、適切な開発形態の選択、見積り、品質評価、進捗管理、顧客側と開発側のロール（役割）の明確化、及びこれらの顧客・経営層への可視化などについて検討を行い、第一部５章にまとめた。

（２）非ウォーターフォール型開発に必要な技術の明確化

新しい開発手法についてはもはや議論すべき時期は終わった、いかに実践し実務に取り入れていくのかという現実的なテーマに取り込むべきであるともいわれる。そのためには、顧客側と開発側に必要な包括的な技術、プロジェクト運営技術、スキルを明確にしておか

ねばならないだろう。かつては、ソフトウェア開発技術者には適性が必要であるとされ、適性試験が実施された。では、今、非ウォーターフォール型開発に求められているのも適性なのか、あるいはもっと個人的な素質なのか、そうではなく、育成可能なスキルなのか、大きな問題として、十分、検討されなければならない。

（3）人材スキル・人材育成方法の明確化

本 PT では、育成可能性のある、“スキル”を議論する。それは講義形式の教育やカリキュラムで十分とは誰も考えないであろう。非ウォーターフォール型開発に必要なスキルを身につけるための人材育成方法、さらに、人材像、スキル体系について、実際に非ウォーターフォール型開発でビジネス展開している企業の人材育成方法を事例として取り上げ、検討する。また、現在の開発手法の課題を解決するために、企業内における改善活動の役割について議論する。

２　非ウォーターフォール型開発に必要な開発技術・スキルの明確化

特にビジネス環境の変化が激しい領域のビジネスに戦略的に活用されている情報システムにおいて、ビジネス環境の変化に迅速に対応することが一層求められるようになっている。このようなコンテキストにおいては、従来のウォーターフォール型のソフトウェア開発形態では十分に対応できないことが多い。それは、数多くのプロジェクトの失敗、動かないシステムの存在、IT業界がもつ３Kイメージ[1] などの様々な事象が発生していることからも明らかである。

今は、もはや、非ウォーターフォール型開発を、いかに実践し実務に取り入れていくのか、という現実的なテーマに取り組むべきである。したがって、顧客側と開発側（事業側・業務側と開発側）に必要な包括的な技術、プロジェクト運営技術、スキルを明確にする必要がある。

なお、非ウォーターフォール型開発に関する技術（例： ペアプログラミング、カンバン等）やスキルの詳細については本報告書の対象外とする。

また、検討の方法として、ウォーターフォール型開発と、非ウォーターフォール型開発の代表格であるアジャイル開発の違いに着目することとし、そこで重視される技術、スキルの違いを浮かび上がらせるという方法を採ることにする。

２．１ アジャイル開発における“スキル”の基本的考察

（1）スキルと価値観の違い

様々な定義、表現方法があると思われるが、本報告では、以下の様にスキルと価値観とを区別し定義する。また、本章ではスキルをクローズアップするため、価値観について、今回は考察しないものとする。

① スキル

プロジェクトに参加するメンバの役割に応じた、もしくは他の役割を支援するための能力。「～～が出来る、～～が作れる」と表現し、一般的な教育、育成によって成長が可能。プロジェクトのアウトプットに直接的に作用しやすい。

② 価値観

非ウォーターフォール型開発に求められる大事なものを理解し、それに沿って活動する心構え。「～～を優先する、～～を大事にする」と表現し、教育、育成による成長というよりも、気付き、自立心により磨かれていくものである。プロジェクトのアウトプットには間接的に作用し、チームビルド、コミュニケーション、自己実現といった要素には大きな影響をもたらす。

---

[1] IT 業界がもつ３K イメージと実態とは合致していない。「給与」「労働時間の長さ」「職場の雰囲気」が要因となる就業満足度への影響は影を潜めつつある。
（「IT 人材白書 2010」(IPA　IT 人材育成本部,pp.142,2010)
http://www.ipa.go.jp/jinzai/jigyou/about.html

（2）アジャイル開発における中核的な価値と基本原則

アジャイル開発で言われる中核的な価値の一部を紹介する。ただし、これは価値であり、それを理解し実現するのは適性と定義する。したがって、この価値を実現出来る能力をスキルとは呼ばないことにする。また、これらの価値は、ウォーターフォール型開発においても必要とされる。

①コミュニケーション

ペアプログラミング、コーディング標準、リファクタリング、顧客との近しい関係などの主要なコミュニケーションに焦点をあてる。

②シンプルさ

必要なことだけを行い、それ以上は何も行わない、最もシンプルな設計が実装され、複雑な実装はリファクタリングされる。

③フィードバック

振る舞いを理解し、コーディングし、テスト（顧客の受入れテストも含む）する活動を時間・日単位で実行する。すばやいフィードバックによってチームと顧客の理解とビジネスの目的を継続的に一致させる。

④勇気

コードよりも先にテストを書く勇気、シンプルな方法を選択する勇気、早い時期に納品しフィードバックを素早く受ける勇気、リファクタリングする勇気がまさしく求められている。

⑤尊重

チームメンバの相互の尊重の上に成り立つ。仲間を勇気づけ、くじかせない。

さらに、アジャイル開発における基本原則を紹介したい。これらは原則であり、それを理解し実現するものを適性と定義する。つまり、技術、スキルを挙げているのではない。これらはウォーターフォール型開発においても同様に必要とされる[1]。

①人間性

ソフトウェアは人によって人のために書かれる。人に焦点をあて、人に力を与え、利益を与える、エンパワーメント(自己の権利や自尊心を取り戻すとともに、自主的な意思決定や行動を促すとともに支援を行う)など。

②考察

なぜ、どのように行っているかを考える、チーム自身による調整、プロセスにおける反省、結果に対する反省、人に対する反省、チームをよりよく動かすための反省。

③質

生まれつき備わっているもの、品質・スケジュール（コスト）は固定され、スコープが可変であることを大原則とする。

④赤ちゃんの歩幅で前進

インクリメンタル思想、小さなサイズの機能性を実装した新しいインクリメントを反復する、「与えられた時間内でAとBを改善するために私たちに何ができるか、を重視する、すなわち、予算が決まっているときは、優先度を付けて、出来るものから少しずつ開発を積み重ねていく。

２．２ 多元的観点による“スキル”の明確化

（１） 役割によるスキルの違い

検討の前提に従い、ウォーターフォール型開発と、非ウォーターフォール型開発の代表格であるアジャイル開発において、プロジェクトに参加するメンバの期待役割を提示し、両者の違いによって表現した非ウォーターフォール型開発に必要とされる作業例を図表 3-1 に示す。

アジャイル型開発	ウォーターフォール型開発
プロダクトオーナー システムのための要件を定義し伝達する役割、優先順位を設定	ユーザー 事業、顧客側
スクラムマスター ファシリテーティング ※調整役であって意思決定をする役割ではない アジャイルプロセスのルールを施行する	プロジェクトマネージャ プロジェクトリーダー、SE 人モノ金の判断をする 仕様、スケジュール、要員の管理 報告、変更の管理
開発者メンバー 要求定義を行う コードを書きながらプロダクトオーナーやテスターと協力してコードが開発されていることを確認する コードのための単体テストを書く 受け入れテストをサポートするテストを書き、テストの自動化を行う 毎日コードを共有リポジトリにチェックインする	開発者、SE、PG 要求定義を行う コードを書く 単体テストを書く テストの自動化を行う コードを管理する
開発者メンバー（テスターを含む） 機能の理解を確認し、要望されている機能と紐付けられているか確認する。 コードが書かれている間に、受け入れテストのテストケースを書く。 受け入れテストの際にコードをテストする 毎日、共有リポジトリにテストケースをチェックインする 受け入れテストやコンポーネントテストを、継続的なテスト環境に統合するためにテストの自動化を開発する	テスター 機能理解、要望機能との紐付け確認 テストケースを書く テストをする テストコードを管理する

図表　3-1　アジャイル型/ウォーターフォール型開発に必要とされる作業例（1/2）

アジャイル型開発	ウォーターフォール型開発
プロダクトオーナー システムのための要件を定義し伝達する役割、優先順位を設定	ユーザー 事業、顧客側
アーキテクト（プロジェクト外） 多くのアジャイルチームはアーキテクトという言葉が役職に就く人々を含んでいない。 アーキテクチャはアジャイルチームにとって非常に重要である。 チームの活動を通して「アーキテクチャは出現する」と言われているが、システムレベルでは、アーキテクチャ はシステムの全体の構造を決定する責任を持つシステムアーキテクト、ビジネスアナリストによって調整されるものである。 アジャイルチームはチームの外側に存在する複数のアーキテクトと話す窓口を持つ	アーキテクト 設計初期に登場 システム全体の構造を決定する責任あり プロジェクトチームは、複数のアーキテクトと話す窓口を持つことになる。
インフラ（プロジェクト外） マシン、ネットワーク環境を提供する	インフラ マシン、ネットワーク環境を提供する
QA・品質保証 （プロジェクト外） 多くの場合、品質保証のための主な責任は、開発者やテスターに移行されることになる。 QA担当者は全体の品質を監督するという、本来意図されていた役割（レビューと品質の監視）に戻る。 QA部門にいたリソースの多くをプロジェクトチームと一緒に仕事できるように派遣してリアルタイムに品質を確認させることになる。 システムレベルのテスト開発（負荷試験等）に関わるようになる。	QA・品質保証 品質保証部門として社内検査を行う 出荷時の最終チェック的な役割 非機能試験は行わない、形式チェック

※上記の “プロジェクト外” とは、個々のプロジェクトに終始配属されるのではなく、専門の部隊として存在し、適宜プロジェクトをサポートするという意味。

図表　3-1　アジャイル型/ウォーターフォール型開発に必要とされる作業例（2/2）

ウォーターフォール型開発とアジャイル開発の大きな相違点として、
①ファシリテーターが存在する。
②毎日コードをチェックインしている。
③品質検証、保証の機能は、プロジェクトの普段の活動に組み込まれていく。
という差分がある。②③は役割の違いによる差であり、スキルの差ではない。①に関しても役割の差とも言え、アジャイル開発に求められる新たな役割もあり、それ自体がスキルの相違と言える可能性がある。

（2）進め方（プロセス）によるスキルの違い

同様に、プロジェクトの進め方（プロセス）における作業例を提示し、両者の違いから非ウォーターフォール型開発に必要とされるスキルを検討する（図表 3-2）。

	アジャイル型開発	ウォーターフォール型開発
	仕様検討 大まかなやりたい事のヒアリング	仕様検討 契約の元となる確定条件の整理
	イテレーション計画策定 やりたい事の優先度を決め、どれくらいの期間で、 どの程度の機能（シナリオ）を作るかを決める	プロジェクト計画策定 ウォーターフォール型のストレートな計画策定 基本、やりたい事は全部やる、全部設計してみる
反復	イテレーション（設計、開発、テスト） 小さな単位にしてドキュメント、コードをセットで作る テストコードも作る、毎日テストをする	基本設計 ドキュメントベースの設計 この時点で再度レビューをして再見積もりもあり 契約によっては、ここから先が別会社ということもあり
反復	イテレーション（提供・デモ、評価） 2週間程度の周期で、動くコードを見せる 実際に触ってもらう その場でフィードバックをもらう	詳細設計、開発、テスト ドキュメント、もしくはビルダーベースの設計 開発 開発が終わってから、まとめてテスト
反復	納品 ある程度の単位、決まった期日になったら、 本番環境にデプロイする	品質監査、納品 プロジェクトとは別部門による品質監査、 形式チェック 納品

図表 3-2　プロジェクトの進め方（プロセス）において必要とされる作業例

上記により、ウォーターフォール型開発と違いアジャイル開発には、

①全部やろうとしない

②ドキュメントだけで設計はしない

③２週間程度で実際に動くものを見せる

④繰り返し型のプロセスがある

という違いがある。従来のウォーターフォール型開発に必要なものとは大きく異なっている。ただし、①②の違いはあるものの、技術、スキルの相違は生じないと考えられる。①はスコープ管理、契約処理に依存する。③④を実施するためには、個々のスキルの違いというよりも、IT スキルのみでなくドメイン領域の知識、幅広い言語知識等のように、知識、スキルの広さを求められる可能性は高い。

ウォーターフォール型開発においては、特に設計とコーディングを別のメンバが担当することがよくあり、場合によっては担当する企業、組織が違う場合もある。それに比べて、アジャイル開発では、繰り返し型のプロセスであるため設計、コーディング、テストコード作成を含むテストが、同じメンバで一貫して実施される。アジャイル開発のメンバは、設計、コーディング、テストを一貫して実施出来るスキルがあると言える。これは、大きな差分になり得る。

（3）ユーザ側に必要とされるスキル

プロダクトオーナー、ユーザと言われる側のスキルについても考察してみたい。ウォーターフォール型開発におけるユーザは、開発の方法について言及する必要はなく、実現するシステム、アプリケーションの業務仕様についての責任を負うことになる。ただし、仕様の決定が、開発工程の先頭にあり、かつ反復型のプロセスを採用していないため、仕様が間違った、決めきれなかった状態を作り出す恐れがあり、かつ、それはほとんどの開発プロジェクトで起きている。さらに、必要になる能力は、決めた機能、スケジュール通りにプロジェクトが進行しているか、進行していない場合、その原因は何か、その対応策は何か、対応策を取る際に発生する差分（追加投資、追加発注等）を管理する能力になる。

アジャイル開発においては、明確な仕様を決めなくても良いとはいうものの、定期的なサイクルで実物を見てフィードバックのポイントを増やすことにより、実際のシステムを目で確認しながら、積み上げる様に仕様を決定していくことになる。ゆえに、全ての機能の仕様を洗い出す能力よりも、コアとなる機能を見定め、優先度を図りながら開発プロジェクトの運営を指揮していく能力が問われることになる。

２．３　本章のまとめ

これまで、役割と進め方（プロセス）という観点からの相違によって、アジャイル開発に必要なスキルの明確化に関し検討してきた。一般的に、従来のウォーターフォール型開発よりも高スキルが求められるという通説があるが、開発における難易度の高い作業が出来るスキルというよりも、以下の３点が、非ウォーターフォール型開発にとって重要なスキルと考えられる。

①プロジェクトのアウトプットに関わる判断ではなく、アジャイル開発の進め方を踏襲させるためのファシリテーションスキル

②反復活動の中で、実際に動くものを作りながら、小規模に、かつトータルにプロジェクトのアウトプットを積み重ねていくスキル

③設計、コーディング、テストを一貫して実施出来るスキル

その他のスキルについては、従来のウォーターフォール型開発でも求められるスキルであり、必ずしも非ウォーターフォール型開発だから必要とされるというものではない。

むしろ、プロジェクトの運営、採用、普段からの育成、契約といった他の影響により、本来、蓄積、発揮されるべきであるスキルが、十分に育成されていないという問題があるのではないだろうか。

追記事項：

アジャイルに必要なスキルとして、リファクタリングスキル、テスト駆動開発スキル等が挙げられるが、これらについては今後の課題としたい。

コラム

欧米のプロダクト開発における開発チーム・技術者のあり方 (1/2)

プロダクト開発において、欧米ではかなりアジャイル開発が浸透しているようである。それを参考にする際、そもそもの開発チームのあり方、技術者のあり方が日本とは異なり、アジャイル開発スタイルに影響を及ぼしていることを留意しておく必要がある。下記に、筆者がこれまでいろいろな機会で得た欧米での開発チームのあり方、技術者のあり方について記述する。

【開発チームのあり方】

基本的には、開発者とテスタの２つの役割がある。開発者は、通常、要求仕様作成からユニットテスト/スモークテスト(いくつかの正常ケースのプログラムテスト)までを担当する。テスタは、テスト計画、テストプログラム作成、プログラム/システムテスト実行を担当する。

大切なのはその役割の中では、技術者は対等であるということである。一つの機能やユーザーストーリーの開発、テストを、一人で、または複数人で担う。日本でみられるように、リーダーが設計して、それに従ってメンバがプログラミング、テストを実施する、というような階層構造は通常存在しない。

また、マーケティングサイドにプロダクトマネージャが存在し、プロダクトのあり方、開発機能セットとその内容などの最終責任者となる。具体的には、プロダクトマネージャと開発チームでレビューを繰り返しながら開発機能セットとその内容を決めていくが、最終決定者はプロダクトマネージャ個人となる。その代わり、そのプロダクトの損益責任も負う。

コラム

欧米のプロダクト開発における開発チーム・技術者のあり方(2/2)

【技術者のあり方】

上記のように、開発者は、要求仕様作成からユニットテスト/スモークテストまでを担当することになるため、広範囲のスキルが求められる。開発者は、ユーザ要求を推測して外部仕様を作成し、定められたアーキテクチャに基づく設計を行い、コードを作成し、ユニットテスト、スモークテストを実施できなければならない。もちろん、経験や能力によってパフォーマンスに違いが出てくると思われるが、その一部のみ(例えばユーザ要求仕様から設計までのみ)を担うというあり方は、通常ない。

また、技術者は、基本的には社員(オフショア拠点の社員も含む)である。付加価値の小さなコンポーネントを外部にアウトソースする場合はあるが、プロダクトの基本部分は社員である技術者で開発する。技術者は、要求されるスキル定義に基づいて社内外から採用され、初心者を職場で育成するという文化はない。

日本では、プログラマから始めて設計者になり、プロジェクトマネージャを経て管理者になるのが成功パターンとみなされ、優秀な技術者であればあるほどプログラミングから早く遠ざかってしまう傾向がある。欧米では、ハイレベルのアーキテクチャ設計を担う立場になっても、その設計結果を設計文書よりもプログラムコードで表す技術者も多いと聞く。

【考察と提言】

まとめると、欧米ではプロダクトマネージャ、開発者、テスタなどの役割が明確であり、その役割の中では技術者はほぼ対等である。開発者は日本に比べると広範囲な開発フェーズを担い、日本のように優秀な技術者ほど早くプログラミングから遠ざかるというようなことはない。

アジャイル開発の具体論は、このような欧米の開発スタイルを暗黙的に前提としている面が多い。よって、日本でアジャイル開発を導入する際、プラクティスを機械的に導入してもうまく実行できないという危険性がある。単にプラクティスを導入するだけではなく、開発チームや技術者のあり方、文化のあり方をどう変えていく、または変えていかない、ということを考えながら、具体的なアジャイル・プラクティスの導入とその成功見通しを検討する必要がある。

本コラムの筆者は、本来のアジャイル開発の効果をだしていくためには、日本の開発スタイルを欧米スタイルに近づけることが必須と考えている。これは、日本の IT 企業の多重請負構造、日本の雇用のあり方などとも関連があり、一筋縄ではいかない。今後の検討会には、これを乗り越えていくための知恵と工夫の検討を期待したい。

(非ウォーターフォール開発 WG 馬嶋　宏　記)

３　人材育成方法の明確化

“スキル”は育成可能性があるからこそ議論できるはずである。素質や適性、と考えれば、育成が困難であるとして、育成を放棄することにつながりやすい。しかし、スキルが育成可能であるとして、講義形式の教育やカリキュラムの議論だけで十分とは誰も考えないであろう。非ウォーターフォール型開発に必要なスキルを身につけるためには、知識だけではなく、人材の育成が不可欠である。さらにいえば、システム開発を実施する人の人材像、そしてその人が持つべきスキルの体系について、個別に明らかにする必要があるだろう。

実際に非ウォーターフォール型開発でビジネス展開している企業の人材育成方法を事例として取り上げ、さまざまな観点から分析したい。その過程で、非ウォーターフォール型開発で不可欠な積極的に改善活動に関われる人材の意義について議論する。

３．１ 人材スキル・育成方法の基本的課題

これまで、アジャイル開発を実践する上での必要な開発技術・スキルについて検討してきた。では、そのスキルを習得するのにはどうしたらよいか、さらに、技術者を育成するためにはどうすればよいのか。そして、修得すれば誰でもアジャイル開発が実践出来るのだろうか。誰がどのようにアジャイル開発に必要なスキルを習得すれば良いのだろうか、について検討し、効果的な育成方法について考察を深めたい。

（１）育成方法の課題

まず、アジャイル開発を実践するためには、様々な方法論・数あるプラクティスから、プロジェクトや組織に適したものを取捨選択し、カスタマイズすることが必要である。例えば、現在の日本の商慣習では、一括請負契約がほとんどであり、この場合は IT ベンダへのリスクが大きい。ウォーターフォール型開発では、事前に全ての要件を洗い出し、確定させることで、なるべく要件変更を抑え、リスクを軽減させてきた。しかしながら、一括請負契約でアジャイル開発を実践した場合、要件が変更される度に、IT ベンダには負担がかかり、上手くマネジメントできなければ、たちまち赤字プロジェクトとなってしまう。さらに、イテレーション計画は当然のこと、顧客と頻繁にコミュニケーション出来るためのプラクティス、環境が非常に重要となってくる。また、不具合の瑕疵担保責任を全て開発側が負うため、テストやシステムの保守容易性に関するプラクティスを整備しておくべきである。

しかしながら、今までアジャイル開発を実践した経験がない開発チームや技術者にとってプラクティスのカスタマイズは容易ではない。何故ならば、あらゆるプラクティスは他のプラクティスと相互作用するからである[2]。アジャイルプロセスの価値・原則も理解しなければならない。アジャイル開発を初めて実施するチームであれば、まずは全てのプラクティスを適用し、模範に従って忠実に実践することが大事である。相互作用を十分に理解した上で、次に、プラクティスをカスタマイズし、自分のプロジェクトで上手くいくかどうかを検証する。そこまで実践できれば、他のカスタマイズが必要なプロジェクトでも、影響が予測でき、新しいプラクティスを創造して対処出来るようになる。まさしく武道に

おける守・破・離[2]のプロセスが必要なのである。

どのようにして全てのプラクティスを適用して実践したらよいのだろうか。二つの方法が考えられる。効果的な方法の第一は、社外的に影響しない自社用のシステム開発プロジェクトにアジャイル開発を適用することである。企業によっては、新人研修のために利用される社内プロジェクトではあるが、現役エンジニア、マネージャ、顧客役を演じる人も参画することで、実際のプロジェクトを試みている企業もある。第二は、社内プロジェクトへの適用が困難な場合、アジャイル開発研修を行うことである。これは約１ヶ月から３ヶ月程度の仮想プロジェクトとしてアジャイル実践経験者を講師に迎えることで、効率良く習得出来る。

このような研修は、教育予算が確保できる大企業向けの方法と言える。では、教育に時間や費用が掛けられない中小企業ではどうしたらよいだろうか。時間や費用が多少でも掛けられるのであれば、２、３日の短期間の研修でも効果は得られるだろう。この場合、チームプラクティスをメインとした内容が望ましい。一人で出来るプラクティスは研修後、各自で習得することが出来る（図表 3-3）。

チームプラクティス	個人プラクティス
チームビルディング	テスト駆動開発
計画ゲーム	リファクタリング
ふりかえり	タスクかんばん
コードの共同所有	継続的インテグレーション
ペアプログラミング	最適なペース

図表 3-3　チームと個人のプラクティス

また近年はアジャイルプロセスに関連する勉強会やイベントが増加している。無料で参加でき、アジャイル実践経験者の生の声が聞ける場所でもあるため、積極的な参加が効果的である。

ある会社では、資金面から OJT として一人で出来るプラクティスを実際の業務に適用し、自信をつけてからリーダー役が可能なプロジェクトでエクストリーム・プログラミングを適用し、トップダウンアプローチで実践するようにしている。

[2] 剣道居合で修行する上での、心・技・気の進むべき各段階を示した３つの教えとされる。「守」とは、師に教えられたことを正しく守りつつ修行し、それをしっかりと 身につけることをいい、「破」とは、師に教えられしっかり身につけたことを自らの特性に合うように修行し、自らの境地を見つけることをいう。また、「離」とは、それらの段階を通過し、何物にもとらわれない境地をいう。

（2）学習カリキュラムの課題

では、何を学ぶことが効果的なのか。昨今、様々なアジャイルプロセス方法論が提唱されているが、アジャイルソフトウェア開発宣言とアジャイルソフトウェアの 12 の原則[3]、そして、図表 3-4 に示すXPの価値とプラクティスを最も基本とすべきであろう[3]。

5 つの価値
コミュニケーション
シンプル
フィードバック
勇気
尊重

13 のプラクティス
全員同席
計画ゲーム
短期リリース
受入れテスト
シンプルデザイン
ペアプログラミング
テスト駆動開発
リファクタリング
継続的インテグレーション
コードの共同所有
コーディング規約
メタファ
最適なペース

図表 3-4　XP の価値とプラクティス

この XP の 5 つの価値は特に重要であり、これに理解があり、身に付いている技術者であれば、全てのプラクティスを理解し、既存のプラクティスで対応出来ない課題に対しても新しいプラクティスを創造して対処出来るであろう。しかし、この 5 つの価値を身に付けるのは必ずしも容易でない。これらの一部は、人間の生まれ持った性質に関わる部分であり、身に付けるというより備わっているかどうかが問題だからである。アジャイル開発において、コミュニケーション能力が必須であるとしばしば言われるが、それは事実なのだろうか。プログラミングが得意な人は、集中力はあるけれども、パソコンに没頭するあまり、人との関わりが苦手でコニュニケーション能力が不足するといわれる。アジャイルは人を大事にするという思想が根底にあり、個々人の特性を尊重するのがまさしくアジャ

[3] 「第一部２．２　アジャイル開発の定義」参照。

イルなのである。したがって、コミュニケーションをサポートするコーチが必要である。

5 つの価値全てが身に付いている技術者がいれば理想だが、チームの中で能力が補え合えれば良い。ただし、役割によって必要な適性も異なる。チームリーダーはコミュニケーション能力や勇気が備わっている必要があるし、技術への関心が乏しいプログラマは、動きさえすれば良いと考え、オブジェクト指向プログラミングを面倒に感じてしまかもしれない。

コーチやスクラムマスターは外部から調達することも可能であるが、チームメンバこそ価値とプラクティスについて理解しなければならない。価値とプラクティスの関係、プラクティスの相互作用を理解していないまま中途半端に実践すると、途中で変化を吸収出来なくなり、失敗してしまう恐れがあるからだ。 価値においては、1 日で理解出来るようなことではなく、プロジェクトを通じて徐々に理解していく必要があるだろう。

役割と適性[4]/スキルの有無、理解すべき項目について図表 3-5 に示す。

役割 / 適性/スキル	メンバー	チーム リーダー	コーチ	スクラム マスター
コミュニケーション	○	★	★	★
シンプル	○	○	○	○
フィードバック	○	○	○	○
勇気	○	★	★	★
尊重	★	★	★	★
技術への関心	★	○	★	○
技術的プラクティス	○	○	○	－

★ ・・・ 適性としてプロジェクト開始時に備わっている必要がある
○ ・・・ トレーニングにより理解する必要がある

図表 3-5 役割と適性/スキルの対応

[4] 適性とは、価値を理解し実現するための適応力を指す。
「第三部 ２．１ アジャイル開発における“スキル”の基本的考察」参照。

（3）課題

アジャイル開発を学ぶことで、今まで正しいと教えてきた企業文化に対し、180 度異なる課題も出てくる。一例を以下に示す。

① 今まではいかに先を見越して設計し、要件を抽出出来るかが大事とされてきたが、アジャイル開発では、シシンプルであることに価値があり、将来必要になるかもしれないような機能を実装しようとしてはならない。
② 顧客が無駄に汎用的な要件をシステムに求めてきても、開発会社は、本当に必要なものは何かを顧客に問いかける必要がある。
③ 他の人の仕事を邪魔しないために、メールでのコミュニケーションを進めてきたが、Face to Face を重視し、できる限り顔を合わせてコミュニケーションを取ることが重要となる。
④ 異なる部門同士が同じフロアで仕事することで、ペアプログラミングでのコミュニケーションが活発になるが、うるさいとの苦情も多いので、チーム毎に会議室などの部屋を設置する等の対策を立てる。
⑤ 上下関係が重要視されている組織では、コミュニケーションに支障をきたすことがある。

また、人事評価基準についても、今までと同じやり方では上手くいかないケースが出てくる。 チームと個人の組み合わせで評価出来ると良い。

① チームリーダーを最も評価するだけでなく、プログラマの評価を上げる
② 個人評価を強めると、競争原理が働いてしまい、オープンなコミュニケーションや他人の不具合の修正に消極的になる
③ 組織評価を強めると、80 対 20 の法則が働き、アジャイル開発で優れた人の負担が増える

そして、顧客がアジャイル開発に対し協力的かどうかは極めて重要になる。したがって、プロジェクトを始めるに当たって以下のことを理解する必要がある。

① 顧客の責務・役割
② アジャイル開発の価値・原則の理解
③ コミュニケーション手段・方法
④ Face to Face の価値
⑤ リリース計画、イテレーション計画
⑥ シンプルな要件定義
⑦ 優先順位の付け方
⑧ 受入れテスト

アジャイル開発では、プログラマが主体となる。今まで業務系 SE を主に務めてきた技術者にも、プログラマへの転換を促すことになるが、抵抗があれば、次のような別の役割として活動出来るようにする。

① スクラムマスター、トラッカー[5]
② 顧客プロキシ（プログラマと顧客の架け橋的役割）
③ 業務コンサルタント
④ オブジェクト指向設計支援
⑤ テスタ（受入れテスト作成・実施）

以上の課題はあらゆる組織に存在するし、顧客の理解も困難な場合も多い。解決に至らなくても、これらの課題を理解しているかどうかで、状況は改善するはずである。

[5] タスクやストーリーの進捗状況、ソフトウェアのバグ状況を定期的に収集する人。

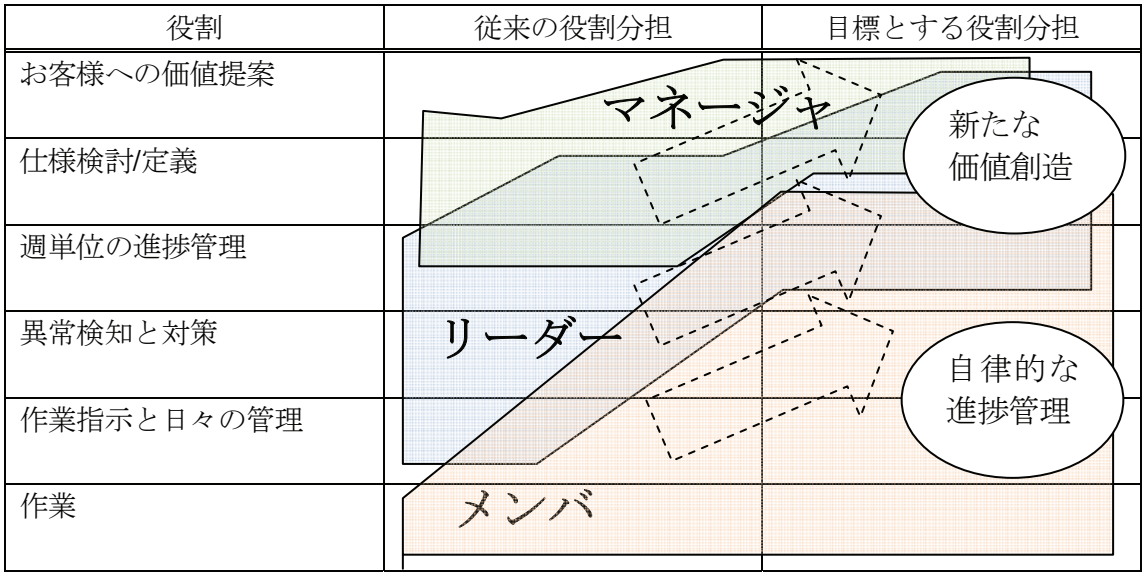
３．２ 人材育成方法のあり方

アジャイル開発では、管理面での人材育成が重要であることが、あまり知られていない。アジャイル開発を担う人材は、常に「本質」を考え、今、どうすべきかを判断できる能力が不可欠である。そのために、常に「なぜ」を問い、「本質」を考える習慣を心がけることが重要である。このような人材の育成検討が、今、求められているのである。

（1）人材育成の狙い

従来のプロジェクトでは、経営環境や仕様変更などによってプロジェクトの状況は大きく変化し、そのためにマネージャやリーダーが週単位や日々の進捗管理に追われてしまっているのが現状である。そのため、本来の仕事である「新たな顧客価値創造」を行う余裕がなくなっている。この状況が慢性化すれば、マネージャやリーダー自身が「進捗管理」を自分の仕事と思い込み、真剣に「進捗管理」に取り組み、メンバは言われたことを行うだけの「指示待ち」状態になってしまうのである。

マネージャ、リーダー、メンバなど、プロジェクトに関わる人たちを、それぞれの管理役割のレベルアップし、全員が、顧客の新たな価値創造に向かえるように意識転換を図る必要がある（図表 3-6）。



図表 3-6　管理レベルを底上げし価値創造に向かう

そこにアジャイル開発の考え方が有用である。とりわけ、「見える化」は、「今」の状況をメンバが把握し、進捗状況に関する問題に対して素早く対策をうつこと、すなわち「自律的な進捗管理」の能力を高めることを支援する。自律的な進捗管理が定着することによって、自然に、プロジェクトに不足しているスキルが明確になり、技術教育を促進する動機となるだろう。

（2）人材育成の課題と活動方針

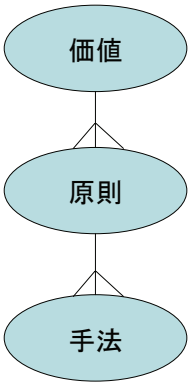
アジャイル開発の考え方を現場に理解/定着させ、管理レベルを底上げするためには様々な課題を解決する必要がある。

ソフトウェア開発の現場にアジャイル開発を定着させたいというニーズは多いが、実践が形骸化し成果につながらず失敗するケースが多い。主な原因として、現場の実践が「手法（プラクティス）」中心となってしまい、背景にある「本質」を理解できずやらされ感が生まれ、形骸化してしまう。アジャイル開発では、書籍や他の事例から得た知識を実践し、その実践から学んだ「実践知」を人に伝え続けることで「本質」への理解が深まる。アジャイル開発の指導者は、現場が形骸化せず実践知を大切にするよう指導し、成果に結びつける必要がある。ここでいう指導者とは、現場で指導の役割を担う人（例：スクラムマスター）や、現場を定期的に訪問する推進組織の担当者を指す。まず、アジャイル開発の「本質」について説明し、その後に指導者がアジャイル開発を指導する際の課題と解決策を以下に列挙する。

ⅰ）課題の解決方針

課題の解決方針を検討してみよう。アジャイルソフトウェア開発宣言（アジャイルマニフェスト）に掲げられた「価値」と「原則」が現場に伝えるべき「本質」であり、この本質を現場に伝えるための道具として「手法（プラクティス）」がある。指導ではそれぞれを次のように取り扱う。

また、「価値」は、アジャイル開発の中心となる考え方であり、変えない。「原則」は、価値に照らし合わせて現場で洗練する。「手法」は、原則に照らし合わせ現場の状況に合わせて常に変化させる。



図表 3-7　価値、原則、手法の関係

ⅱ）課題とその解決策

図表 3-7 のような関係のもと、次の３つの課題とその解決策を提示する。

①形骸化させないため、現場の様々な状況（開発方法、チーム体制、問題点）などを考慮してアジャイル開発の指導を行う必要がある。そのためには、施策として、原則と照らし合わせて現状を把握し自責表現を通して現場に考えさせる。問題を他責と捉え

るのではなく、自責と捉えることで、効果的、かつ現実的な解決方法を導くことができる。

②指導のＰＤＣＡサイクル、すなわち今回の指導内容とその結果から次回に向けての指導内容を洗練していく必要がある。施策として、実践結果を原則に照らし合わせて原則を洗練する能力を向上させる。

③指導内容がどの程度相手に伝わっているかを効果的に評価する必要がある。しかし、ある状況で指導した内容は状況が変化すれば異なった形での相手の言動として返ってくるため、指導に失敗したのか、指導は成功したが状況が変わって言動が変化したのかがわからず、評価が不明確になってしまう。 施策として、言動を常に価値観と比較して評価する、言い換えると、価値観と比較する能力を育成する。

ⅲ）施策

次のような３つの施策を提案する。第１に、本質と照らし合わせて現状を把握し自責表現を通して現場に考えさせる。

自責表現とは、例えば「チームメンバがなかなか発言してくれない」（他責表現）と嘆いているリーダーに対して、「あなた（リーダー）が普段からメンバの発言を妨げる言動をしてしまっているのではないですか？」（自責表現）という問いかけをし、新たな観点で考えてみるきっかけを与えることである。

手法は状況に応じて変化させるものであるが、書籍や事例の通りでは、現場で解決すべき課題の的を射ることができなかったり、チームが持つ課題解決能力を十分に活用できていなかったりすることが多い。

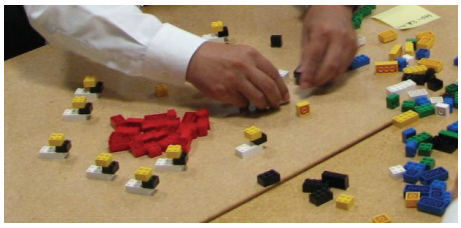
現場の状況に合わせ、現場が原則に沿って手法を常に変化させることができるよう指導を行う。通常は手法をそのまま使用した方が効率的と考えがちだが、アジャイル開発を現場で実践するにあたって大切なことは「手法を変化させる能力」であると考え、現場が常に手法を変化させることにチャレンジするよう指導する。具体的には、現場に考えるきっかけを与え、次の行動を見つけ出す手助けをする。現場が訓練を積み、未経験の状況に遭遇した時でもその場に合わせて的確に手法を選択したり、従来の手法を変化させたりすることができることを目指す。

指導内容としては、第１に、現場が意識していない観点からの質問を行うことで、現場が原則に立ち戻って考えるきっかけを作りだし、結果として原則に沿った次の手法の変化を生み出す。例えば、アジャイル開発でよく言われている「実装されていない設計書、テストされていないコードはムダである（在庫のムダ）」という指摘は、知識としては知っていても他人事で実感が湧いていないことが多い。現地で現物を指差して指摘することで問題が他責から自責に変わり、原則に立ち戻って考えることを始めるきっかけとなる。また、普段の会話の中からでも現場に考えるきっかけを与えることができる。

第２に、実践結果からのフィードバックを原則に反映させる。実践した手法の成功/失敗を分析し、価値に沿って原則を追加/修正する訓練を行わせる。通常は手法へフィードバックする方が効率的と考えがちだが、現場では常に状況が変化することから、常に瞬時に実践手法を導き出すことが求められる。そのため、現場にとって大切なことは、手法は今、その場でしか役立つものでしかないことを理解し、大切なのは「原則を洗練させる能力」であると捉え、結果を常に原則にフィードバックすることにある。原則が洗練すれば、状

況に応じて更に多くの手法を導き出すことができるようになる。

【レゴブロックを使用したふりかえり体感ワーク】
- ゲームを通して数チームで競いながらアジャイルの良さを学ぶ
- 1名が伝達役、残り3名が組立役となり、5分で組立てられる数を競う
- ゲーム後にうまくいった行動と改善点をチームで話し合い（ふりかえり）、次のゲームですぐに試す
- 繰り返す毎に得点が伸び、ふりかえりによりチームへ知恵が蓄積する様子を実感できる
- 短時間（1時間程度）でアジャイルの良さを体感できる（「知る」から「わかる」へ）

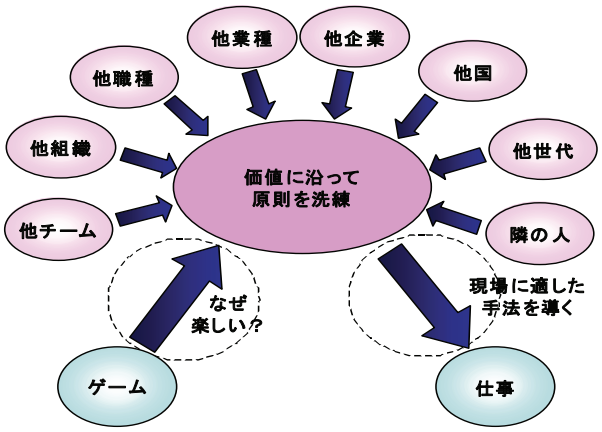


「ふりかえり」の効果による上達を実感、アジャイルを体感

レゴでのものづくりを題材としたゲームのワークショップは大変盛り上がる。現場のリーダーやメンバは子供に戻ったように熱くなる。ゲームが終わり、ゲームが自分を熱くさせてくれた要因を抽出してもらうと、次のようなことが挙がる。隣のチームと競い合える、得点がリアルタイムにわかる、すぐ達成感が味わえる、チームで喜べる、途中の状況がすぐにわかるので何をすればいいかが判断できる、手や体を動かす、など。ここで指導者から「この感動を仕事に活かすにはどうするか」と問いかけると、とたんに、皆、暗い顔になる。

図表 3-8　レゴワーク実習事例

図表 3-8 に示すようなレゴブロックを使用したふりかえり体験ワークのようなゲームから気付いた要因をノウハウとして仕事に取り込むことができれば、仕事をゲームのように楽しむことができる。異分野から積極的に知恵を取り込もうと意識することができれば、ゲーム以外からも多くのことを取り込むことができ、仕事のやり方を良くし続けることができる。例えば、自分と異なる職種（営業職、ＳＥ職、スタッフ職）、異なる業種（工場などの製造業）、他企業などからもノウハウを取り込むことができる。取り込むきっかけは「なぜ」と自問自答することである。なぜ楽しそうなのだろう、なぜ儲かっているのだろう、なぜあのようなことができるのだろう、と考える。積極的に取り込もうとする意欲を高めることで、仕事のやり方を変え続けることができる。異なる分野から学ぶものはないと思った瞬間に個人/チームの成長は止まる。広い意味でプログラミング作業は、このようなゲームのような楽しさを伴う仕事であることを気付く場となることが期待される(図表 3-9)。



図表 3-9　学ぼうとする意欲が大切

第３に、日々の言動が常に価値に沿っているかを比較して自己評価する。相手の言動を状況に応じて分析し、常に価値と比較することを訓練する。本質は抽象的概念であるため、価値や、それに従った原則を言葉だけで伝えても相手に本質が伝わらず、結果として相手の言動に表れない。指導する際に価値と原則に基づいた手法を選択し、手法を通して本質を伝える。本質が相手に伝わったかどうかは、相手の言動が本質につながっているかどうかで評価する。

訓練を積むと様々な場面での評価が可能となる。相手に伝えたことを相手がそのままの形で実践していなかったとしても、状況の変化と照らし合わせて言動を本質と比較し育成状況を評価することができる。例えば、上記で挙げた「在庫のムダ」以外のムダを現場が自発的に見つけ始めた時は、本質を現場が理解し、実践に反映し始めた時である。指導者はこのタイミングを逃さず、褒めるなどの意思表示をする必要がある。

アジャイル開発においては、管理面における人材育成が非常に重要である。プロジェクトが顧客への新たな価値創造に向け成功し始めると、不足がちなスキルが明らかにされ、技術を学ぶ動機につながる。

アジャイル開発を担う人材には、常に「本質」を考え、今どうすべきかを判断できる能力が重要である。そのためには、指導者は常に「なぜ」を問い、「本質」を考える習慣を心がけることが重要である。

３．３ 人材育成カリキュラムの実例

ここでは、あるプロジェクトでアジャイル開発を導入した際に開発したカリキュラム事例を紹介する。

（1）育成カリキュラムの概要

育成カリキュラムのステージを説明しよう。これからアジャイル開発を適用しようと思い立って、実際にプロジェクトに適用するまでの育成カリキュラムは、大きく 3 つのステージに分けて検討する。

① 導入前
② 立上時
③ OJT

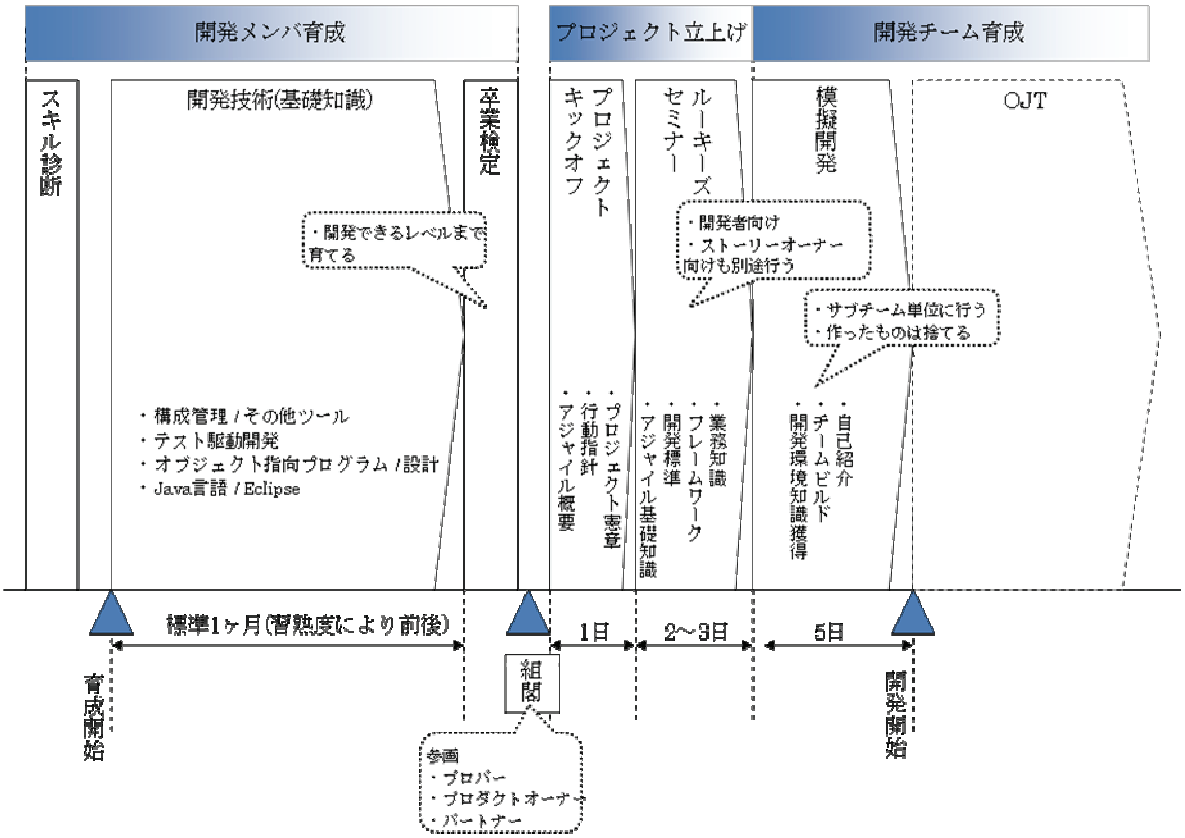
アジャイル開発の導入時に、多くの場合、導入の責任を持った個人、もしくはチームが結成されることが多い。カリキュラムの作成は、このような先行的なチームと相談を行いながら進める。この時に、先行チームにアジャイル開発の概要的な研修を行うなどして、共通の用語で話せるようになっていることが必要である。

また、事前にプロジェクトに参画するメンバが決まっている場合は、その開発スキルをチェックし、必要であればプロジェクトの開始に先立ち、必要な開発スキルを身につけるための教育を実施する。

次のプロジェクト立上時に、アジャイル開発を行うために必要な基礎教育を重点的に行う。プロジェクトには、より多くの人が関わるので、会話をスムーズにするためにも用語を統一する必要がある。事前に、基礎教育を行うことが望ましいが、通常は他の業務を行っているので、立上時にまとめて実施するのが効率的である。

しかし、座学で学習するには限界があるので、基礎教育では詰め込みすぎずに、実践的な内容などについては、OJT を活かして地道に育成を行う。

さらに、OJT として、立上時の基礎教育では説明しきれない知識、技能や、実践知が伴わないと理解できないような内容について、プロジェクトを進めていく中で育成を行う。



図表 3-10　育成スケジュールの概要

上記の育成カリキュラムのスケジュールを図表 3-10 として図示する。プロジェクトの開始に先立ち、開発メンバ候補のスキル診断を行う、もし、開発スキルが低い場合は、必要なスキルを身につけるための育成を行う。プロジェクト立上時には、プロジェクトキックオフを行い、プロジェクトのゴールを関係者で共有する。また、２～３日間のルーキーズセミナーと称した、プロジェクト固有のルールや知識(開発標準や、業務知識)などを身につける教育を行う。

その後は、サブチームごとに模擬開発を行い、開発に必要な知識を身につける。この時に、作成した成果物は、プロジェクトに即したものではあるが、実際のプロジェクトには流用せずに、捨てることにする。開発環境などに慣れていない中で作成したものはどうしてもこなれておらず、その成果物をうまく活かそうとするとどうしてもその後の開発がしにくくなることが多いので、ここでは捨てることにしている。

そして、実際のプロジェクトを進めながら、その時の経験や、必要な知識を考慮して教育を OJT として実施する。

（2）育成の対象とカリキュラム

育成の対象は主に開発チームメンバではあるが、実際のプロジェクトを成功させるためには、その他のプロジェクト関係者も適切な知識を身につける必要がある。

図表 3-11 は、アジャイル開発を組織的に導入するにあたり、外部の育成機関の援助を受けて組織内の対象者別に育成を行った際のカリキュラムの例である。「＊」と記された領域については、カリキュラムの開発を個別に検討すべきである。例えば、業務知識を身につける責務は開発チームや先行チームであるが、どのような内容をカリキュラムとして用意するかは顧客やプロダクトオーナーの検討事項である。

	開発チーム	スクラムマスター	顧客／プロダクトオーナー	先行チーム	リーダー	PM	経営者層／購買担当など
アジャイル概要	○	○	○	○	○	○	○
アジャイル基礎知識	○	○	○	○	○	○	
アジャイル擬似体験	○	○	○				
業務知識	○		＊	○			
開発環境	○			＊			
基本アーキテクチャ	○			＊			
業務分析／モデリング	△		△				
開発技術	△			＊			
ファシリテーション概要	○	○	○	○	○	○	
ファシリテーション演習		○			○	○	

○：立上げ前後の必須教育の領域

△：事前に準備が困難で OJT が必要な領域

＊：内容を組織内で個別に検討する必要がある領域

図表 3-11　対象別育成カリキュラム

各カリキュラムの概要を図表 3-12 に示す。

名称	概要
アジャイル概要	アジャイル開発に携わる方向けの基礎知識
アジャイル基礎知識	一般的なプラクティスについての紹介
アジャイル擬似体験	アジャイル開発のプロセスを体験を通して理解する チームビルディング的な狙いもある
業務知識	開発対象の業務を理解する(内容は先行チームと検討)
基本アーキテクチャ	開発対象のシステム構成や、利用するフレームワークなどを理解する(内容は先行チームと検討)
業務分析／モデリング	業務を整理し、開発側に伝えるための手法を理解する
開発技術	開発に必要な技術を身につける（必要に応じて）
ファシリテーション概要	ファシリテーションに関する知識を理解する
ファシリテーション演習	ファシリテーションに関する知識を体験を通して理解する

図表 3-12　各カリキュラムの概要

（3）教育スケジュール例（3 カ月のパイロットプロジェクト）

アジャイル開発の効果を得るべく、パイロットプロジェクトを設定し、そこにアジャイル開発を導入した。図表 3-13 に初期教育スケジュールの例を示す。

<table>
<tr><th></th><th>1日目(火)</th><th>2日目(水)</th><th>3日目(木)</th><th>4日目(金)</th><th>5日目(月)</th></tr>
<tr><td>8:45</td><td rowspan="2"></td><td colspan="4">朝会</td></tr>
<tr><td>9:00</td><td rowspan="3">Java言語教育
・Hello World
・コンパイルエラーの読み方
・変数と型
・演算子</td><td rowspan="3">Java言語教育
・参照型変数
・可視性(private、public)
・カプセル化</td><td rowspan="3">Java言語教育
・クラススコープとインスタンススコープ
・継承
・抽象クラス
・インタフェース</td><td rowspan="3">UML基礎教育
・クラス図
・オブジェクト図</td></tr>
<tr><td>9:30</td><td rowspan="2">キックオフミーティング</td></tr>
<tr><td>10:00</td></tr>
<tr><td>10:30</td><td rowspan="2">ふりかえり体験</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11:00</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11:30</td><td colspan="5">昼休憩</td></tr>
<tr><td>12:00</td><td colspan="5"></td></tr>
<tr><td>12:30</td><td rowspan="2">タスク進行シミュレーション</td><td rowspan="2">Java言語教育
・配列
・制御構文(if、for、while)</td><td rowspan="2">Java言語教育
・APIドキュメントの読み方
・JavaDoc
・配列とArrayList</td><td rowspan="2">Java言語教育
・アクセス制御詳細
・パッケージ</td><td rowspan="2">ふりかえり</td></tr>
<tr><td>13:00</td></tr>
<tr><td>13:30</td><td rowspan="3">アジャイル開発の基礎知識</td><td></td><td></td><td></td><td rowspan="3">プランニング</td></tr>
<tr><td>14:00</td><td rowspan="2">Java言語教育
・クラス
・コンストラクタ
・メソッド</td><td rowspan="2">Java言語教育
・HashMap
・TreeMap
・ポリモフィズム</td><td rowspan="2">Java言語教育
・例外処理</td></tr>
<tr><td>14:30</td></tr>
<tr><td>15:00</td><td rowspan="2">開発環境設定</td><td rowspan="2">夕会</td><td rowspan="2"></td><td rowspan="2">作戦会議</td></tr>
<tr><td>15:30</td></tr>
<tr><td>16:00</td><td colspan="5"></td></tr>
<tr><td>16:30</td><td colspan="5"></td></tr>
<tr><td>17:00</td><td colspan="5"></td></tr>
<tr><td>17:30</td><td colspan="5"></td></tr>
<tr><td>18:00</td><td colspan="5"></td></tr>
</table>

図表 3-13　初期教育スケジュールの例

開発メンバは、新人研修以来、ほとんど Java による開発経験がなかったため、スキルの高い育成担当をプロジェクトの支援メンバに任命した。その中で、3 日間の研修期間を設けて研修を進めていったが、3 日間では開発に十分なレベルには達しなかったため、さらに 4 日間の Java 言語研修を追加した。図表 3-14 に、OJT のスケジュールの例を示す。

場面	期間	育成テーマ	コンテンツ／指導ポイント
初期教育	1～2週目	開発の基礎知識を身に付ける	アジャイル開発の基礎
			Java言語の基礎
			UMLの基礎
			開発環境の使い方
週のふりかえり	毎週月曜日	チームの改善	付箋を使ったふりかえりの進め方
			Tryの実施と、その効果の確認
			Problemを挙げて失敗に気づかせる
月のふりかえり	毎最終月曜日	アジャイルなチーム	アジャイル度評価
スプリント#1	3～4週目	ストーリを実装する	計画ゲームの進行
			顧客要求の管理
			・ストーリによる管理
			・相対規模見積もり／ストーリポイント
			・見積もりポーカー
			作業の計画と管理
			・タスクボード
			・週のリズム設計／時間割
			バーンダウンチャート
			アプリケーションフレームワークの使い方
			・ドメインのインスタンスの生成
			・テストデータの投入方法
スプリント#2	5～6週目	チーム運営をチームに任せる	リファクタリング
			・Viewを中心に実施
			デザインパターン
			・Factory Methodパターン
			タスク管理について
			・日を跨ぐタスクの扱い方
			コーディング標準
			・クラスやメソッドの命名規則
スプリント#3	7～8週目	保守しやすいプログラム	マルチスキルについて
			ファシリテーションについて
			デザインパターン
			・Null Objectパターン
			オブジェクト指向の設計原則
			・単一責任の原則
			プロダクトバックログ
			追加の要求の受け方
			Subversionの使い方
			・コンフリクトの解消の仕方
スプリント#4	9～10週目	設計力向上	オブジェクト指向の設計原則
			・開放閉鎖の原則
			・依存関係逆転の原則
			リファクタリング
			・不吉な匂い
			Eclipse
			・リファクタリング機能
チャレンジスプリント	11週目	特に設定しない	
クロージング	12週目	育成効果のふりかえり	学んだスキルのたな卸し

状況に応じて	実施時期未定		アジャイル開発の特徴(スクラム、XP以外)
			・リーンソフトウェア開発
			・FDD
			オブジェクト指向の設計原則(紹介していないもの)
			デザインパターン(紹介していないもの)
			GRASPパターン
			ソフトウェア品質の保守性の考え方

図表 3-14　OJT スケジュールの例

スプリントごとにテーマを設定し、それに沿う内容を育成担当から適宜指導を行った。前半では、アジャイル開発のリズムを作るためのスキルの向上に時間を割いている。

（4）育成カリキュラムの例（50 名規模のアジャイル開発）

約 50 名が開発メンバとして参加するようなやや大規模のプロジェクトにおいて、アジャイル開発を行う際のカリキュラムを検討してみよう。このプロジェクトでは、プロジェクト運営にスクラムをベースとした手法を用いている。図表 3-15 に開発メンバの役割とその概要例を示す。

役割	説明
プロジェクトマネージャ	複数チームを束ねる。
スクラムマスター	アジャイルでの開発が滞りなく進むように 必要な準備や調整を行う。 中立かつ第３者的に開発側/ストーリーオーナー側を調整する。 開発チーム毎に専属のスクラムマスターがいるのが望ましい。
テクニカルコーチ	技術的な支援を行う。例えば、開発手法や、ツールの指導を行う。 １つのチームに属するのではなく、複数のチームを担当する。
アジャイルコーチ	技術的な支援を行う。例えば、アジャイル・プラクティスの導入・指導を行う。スクラムマスターに対して指導を行う。 １つのチームに属するのではなく、複数のチームを担当する。
プロダクトオーナー	プロダクトバックログの管理について責任を持つ。原則、１人である。
ストーリーオーナー	ビジネス側の要求をストーリーとして、開発側(主にアプリケーション開発チーム)に伝える。開発側の成果物を確認する。
ビジネスアーキテクト	対象とするビジネスの全体像を把握し、必要なレベルに抽象化する。 ドメイン開発チームのストーリーオーナー的な位置付けである。
開発チーム代表	開発チームの代表となって、他チームと連携する。他のチームと仕様の調整などを行うので、それにふさわしい技術レベルを持つ。プロパーだから、代表になるということは考えない。パートナーでも、代表になる。チーム内で役割を持ち回りしても良い。 リーダーではないので、決定権を持って、チームを牽引する役割ではない。
開発メンバ	主体的に開発を行う。ストーリーオーナーと仕様を確認し、それに適合した開発を行う。

図表 3-15　役割の一覧例

各サブチームには「リーダー」ではなく、「開発チーム代表」を設けている。リーダーがいると、リーダーが様々な決定を行ってしまい、メンバが思考停止になり、その結果、メンバのスキル向上が阻害される危険性が高い。それを防ぐために、このような役割を設定した。

このようなプロジェクトにおける人材育成のために、プロジェクトでの進め方を見据え

必要な役割やスキルを洗い出す。図表 3-16 にその例を示す。

役割																		
プロジェクトマネージャ	スクラムマスタ	テクニカルコーチ	アジャイルコーチ	プロダクトオーナ	ストーリーオーナー	ビジネスアーキテクト	開発チーム代表	開発メンバ（右欄参照）				開発メンバ						
										F/Wチーム		■	■	■	■	■	■	
										ドメインチーム				■	■			
										アプリチーム		■	■	■				
										QAチーム		■	■	■	■	■	■	
												UI層	AP層	SV層	DM層	PS層	運用基盤層	
◎	◎	◎	◎	◎	○	○	◎		プロジェクト運営			○						
◎	◎	◎	◎	◎	◎	○	◎		運営	チーム	スクラム一般	◎	◎	◎	○	△	△	
○	◎	◎	◎	◎	◎	○	◎		運営	チーム	スクラム PJ固有	◎	◎	◎	○	△	△	
△	－	－	－	○	◎	○	○		業務知識		実務	◎	◎	○	○	△	?	
△	－	－	－	○	○	◎	◎		業務知識		ドメインモデル	△	○	◎	◎	△	?	
－	－	○	◎	△	○	◎	○		業務知識		モデリング	－	○	◎	◎	○	?	
－	－	◎	◎	－	－	－	○		技術		Java	◎	◎	◎	◎	◎	?	
－	－	◎	◎	－	－	－	○		技術		OOプログラミング	○	◎	◎	◎	◎	?	
－	－	◎	◎	－	－	－	○		技術		OO設計	○	○	◎	◎	◎	?	
－	－	◎	◎	－	－	◎	○		技術		UML	○	◎	◎	◎	◎	?	
－	－	◎	◎	－	－	－	○		技術		リファクタリング	○	◎	◎	◎	◎	?	
－	－	◎	◎	－	－	－	○		技術		TDD	○	◎	◎	◎	◎	?	
－	－	◎	◎	－	－	－	○		技術		単体・結合テスト	○	◎	◎	◎	◎	?	
－	－	◎	△	－	－	－	○		技術		システムテスト	◎	◎	◎	◎	◎	?	
－	－	◎	△	－	○	－	○		技術		HTML/CSS	◎	○	－	－	－	?	
－	－	◎	△	－	△	－	○		技術		JavaScript	◎	－	－	－	－	?	
－	－	◎	△	－	○	－	○		技術		WebUIテスト	◎	○	－	－	－	?	
－	－	◎	△	－	－	－	○		技術		SQL	－	○	○	○	◎	?	
－	－	◎	△	－	－	－	○		技術		DB物理設計	－	－	－	－	◎	?	
－	－	◎	△	－	－	－	○		技術		DBチューニング	－	－	－	－	◎	?	
－	－	◎	◎	－	－	－	○		開発環境		Eclipse	◎						
－	－	◎	◎	－	－	－	○		開発環境		Subversion	◎						
－	－	◎	◎	－	－	－	○		開発環境		Trac	◎						
－	－	◎	◎	－	－	－	○		開発環境		Hudson	△	◎	◎	◎	◎	?	
－	－	◎	△	－	－	－	○		開発環境		フレームワーク	◎						
－	－	◎	△	－	－	－	○		開発環境		ステージング環境	◎						

図表 3-16　役割とスキルの対応例

アジャイル開発ではクロスファンクショナルにチームを分けるのが基本である。しかし、チーム数が増えると、基盤に近い部分の整合性を取るのがオーバーヘッドとなってくる。そこで、事前にどのようにサブチームを分けるかを検討し、それに必要なスキルと役割を決める。図表 3-16 では、チームをフレームワーク（F/W）開発チーム、ドメイン開発チーム、アプリケーション開発チーム、品質保証（QA）チームの 4 つのタイプに分けている。この中で、アプリチームは複数のチームに分かれ、他のチームは 1 つである。

４　まとめ

非ウォーターフォール型開発手法は、単に開発手法の問題だけではない。本報告書では、その取組みは、日本における IT 化への取組みが先進諸外国のみならず韓国はじめ新興の IT 先進国に対してのキャッチアップのために不可欠であり、さらに日本における IT 産業の後進性を克服するための重要な施策につながるものであると考えている。

しかしながら、その認識は、政策当局はおろか IT 産業、広く IT を道具として活用してきた多くの企業においてさえ決して高くない。したがって、本報告書では、昨年度に取りまとめたシステム開発の現状とあり方から進んで、実際にどう取り組むか、とりわけ企業やＩＴベンダの経営者にとっての価値に始まり、具体的に技術者をどのように育成するのかについて、基本構想からカリキュラムの考察まで広範な検討を行ってきた。今後の更なる議論の発展と取組みが期待される。

参考文献

[1] ディーン・レフィングウェル,アジャイル開発の本質とスケールアップ,株式会社翔泳社,2010

[2] 川端光義・阪井誠・小林修「効果的な XP の導入を目的としたプラクティス間の相互作用の分析」、ソフトウェアシンポジウム 2004 論文集、2005.

[3] Kent, B.”eXtreme Programming eXplained Embrace Change” Addison-Wesley Professional, 1999. 長瀬嘉秀ほか(訳)「XP エクストリーム・プログラミング入門―変化を受け入れる」, ピアソンエデュケーション, 2005.

第四部　非ウォーターフォール型開発に
ふさわしい契約

１　　非ウォーターフォール型開発にふさわしい契約検討の背景と目的

１．１　非ウォーターフォール型開発にふさわしい契約検討の背景

IPA/SEC で平成 21 年度に実施した「非ウォーターフォール型開発の調査研究」では、国内における非ウォーターフォール型開発の採用に際して、次の５つの重点課題を選定した。

①契約：　契約のあり方、調達の制度設計
②価値評価：　経営層や顧客[1]側企業へのアピール
③環境整備：　管理手法や技術面の整備
④普及：　コンサルタント等の役割の整備、人材育成
⑤調査：　欧米の競争力（ビジネスドライバ、産業構造）などの調査

第四部では、この５つの重点課題のうち、「①契約：契約のあり方、調達の制度設計」についての検討結果をまとめたものである。

１・２　非ウォーターフォール型開発にふさわしい契約検討の目的

アジャイル開発は変化への対応を重視するため、開発開始時点では仕様を固定せず、顧客側とのコミュニケーションにより状況に応じて開発途中で仕様を決めていく。そのため、アジャイル開発は、開発開始前の契約時点においては成果物の詳細を確定することができず、「仕事の完成」に対して報酬を支払う請負契約はなじまないと考えられる。そこで、次の二つのテーマに取り組んだ。

（1）日本における非ウォーターフォール型開発に適した契約モデルの検討

（2）非ウォーターフォール型開発に適した契約のひな型の検討と作成

１・３　非ウォーターフォール型開発にふさわしい契約検討の方法

非ウォーターフォール型開発に適した契約を探るため、二つのアプローチを取った。

一つ目の方法が、平成 21 年度に実施された「非ウォーターフォール型開発に関する調査」における 17 社、22 事例における契約形態を詳細に分析することである。実際に非ウォーターフォール型開発で利用されている契約であるため、応用可能性が高いと考えられる。

なお、「労働者派遣契約」は、要員派遣を目的とした契約であり、当該要員への指揮命令権は発注者側（委託側企業）が持っており、仕様は自由に変更できる。したがって、発注者側（委託側企業）がシステム開発のイニシアティブを持っている場合には、アジャイル開発に適していることは明らかなので、ここではあらためて検討の対象とはしていない。

二つ目の方法が、文献調査によって非ウォーターフォール型開発に適した契約を探るこ

[1] 「顧客」という言葉は広い意味を持っているが、開発プロジェクトごとに異なる意味で使われている。本書で使用されている「顧客」は、「顧客企業」および「エンドユーザ」の開発者（作る人）以外の関係者（ステークホルダー）として使用されている。

とである。日本に比べて海外の方が非ウォーターフォール型開発は普及していることから、すでに非ウォーターフォール型開発に適した契約モデルが作られており、文献になっていると考えられるからである。

２　非ウォーターフォール型開発における契約の調査事例と契約の問題点

２．１　契約方式調査（平成 21 年度実施）による契約事例と問題点

（１）契約方式調査（平成 21 年度実施）による契約事例

平成 21 年度に実施された「非ウォーターフォール型開発に関する調査」における 17 社、22 事例の契約事例[2]と契約方式[3]の一覧を示す。

No.	事例	主な契約方式
1	携帯向けブログシステム	請負契約（毎月更新）
2	共通認証システム	請負契約
3	プロジェクト管理システム	請負契約
4	教務Webシステム	請負契約
5	システム管理ミドルウェア開発	請負契約
6	株式取引のためのWebアプリケーション	請負契約
7	プラント監視制御用計算機システム	請負契約
8	パッケージソフトウェア	準委任契約（四半期単位）
9	アプリケーションプラットフォーム	準委任契約
10	生産管理システム	準委任契約
11	小売業における業務システム	準委任契約
12	社内版SNSシステム	準委任契約
13	OSS版SNSシステム	準委任契約
14	Webメディア開発	準委任契約
15	共通EDI開発	準委任契約
16	研修運営システム	準委任契約（推敲フェーズまで） +請負契約（作成フェーズ以降）
17	検索エンジン開発	派遣契約
18	サプライチェーンマネジメントシステム	サービスの利用料金がビジネスの基本単位となる、ASP契約
19	携帯ソーシャルゲーム	社内開発のため契約なし
20	教育機関向け統合業務パッケージ	社内開発のため契約なし
21	開発案件管理Webアプリケーション	社内開発のため契約なし
22	アジャイル型開発の支援環境開発	社内開発のため契約なし
23	製造業向けプロトタイプシステム	不明

出典：「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC,2010）

図表 4-1　事例と契約の対応一覧

---

[2] 詳細は、「第一部　非ウォーターフォール型開発検討におけるアジャイル開発の位置づけ 付録１　ビジネス構造モデルによるアジャイル開発事例の分析例」を参照。
調査したのは 17 社 22 事例であるが、準委任契約の SNS 事例が No.12 と No13 の２つに分かれているためここでは 17 社 23 事例となっている。
この調査では、アジャイル開発に関する契約にフォーカスしているため、同じプロジェクトの他の工程で、別の契約形態も採用されている可能性がある。

[3] 請負契約とは、開発側企業側が成果物の完成を請負、顧客側企業側が成果物に対する報酬の支払いを約束する契約形態である。準委任契約とは、業務を委託する契約であり、開発側企業側の責任は、業務を実施することにあり、成果物に対する完成責任を負わない。派遣契約（労働者派遣契約）は要員派遣を目的としたものであり、当該要員への指揮命令権は派遣先が持っている。（請負契約および準委任契約の場合、委託側企業内で作業を実施していたとしても、委託側企業に指揮命令権はない。）

（2）契約方式調査による契約事例から導かれた契約の問題点（リスク）

平成21年度に実施された「非ウォーターフォール型開発に関する調査」における17社、22事例の契約事例より導かれた、非ウォーターフォール型開発に、現行の請負契約および準委任契約を適用しようとするときに発生する種々の問題点（リスク）を図表4-2および図表4-3に、それぞれまとめた。なお、図表中、未記入の箇所でも問題がない訳ではない。

請負契約適用時の問題点（リスク）

問題点(リスク)	発注側	受注側
要件が未確定	ー	成果物の要件が決まっていないため、契約金額の範囲内で業務が完了できない場合がある。
成果物が不明確	ー	契約時に成果物を規定できないため、納入すべき成果物としての妥当性を主張できない。
性能と品質が不明確	工期延長に伴い、システムの稼動時期が遅れるリスクがある。	契約時に合意した内容がないため、発注側の要請により、工期が際限なく延び、契約金額の範囲内で業務が完了できない場合がある。
開発工数の見積りが難しい	過大見積りにより、発生工数に比べて過大な契約額となる場合がある。	過小見積りにより、トラブルが発生しなかったとしても適正な対価が得られない場合がある。
工期が不明確	・契約期間が決まらないため、発注側メンバのアサインができない。 ・システム稼動時期が想定できないリスクがある。	契約期間が決まらないため、開発メンバのアサインができない。
ユーザとベンダの連携が必要	・発注側と受注側の責任分解点を明確にできないため、開発したシステムに不具合が発生したときの責任の所在が不明確になるリスクがある。 ・発注側メンバから受注側メンバへの指示が職業安定法や労働者派遣法に抵触する可能性がある。	発注側と受注側の責任分解点を明確にできないため、開発したシステムに不具合が発生したときの責任の所在が不明確になるリスクがある。
ユーザとベンダのコミュニケーションが必要	発注側メンバから受注側メンバへの指示が職業安定法や労働者派遣法に抵触する可能性がある。	ー

出典：「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC,2010）

図表4-2　非ウォーターフォール型開発における請負契約適用時の問題点

準委任契約適用時の問題点（リスク）

問題点(リスク)	発注側	受注側
要件が未確定	・要件の確定に時間がかかると、その分委託費用が増加する場合がある。 ・成果物に不具合が発生した場合に、製造上の瑕疵に相当する内容であっても、受注者側に責任を取らせることができない。	―
成果物が不明確	・要件の確定に時間がかかると、その分委託費用が増加する場合がある。 ・成果物に不具合が発生した場合に、製造上の瑕疵に相当する内容であっても、受注者側に責任を取らせることができない。	―
性能と品質が不明確	工期の延長は、委託費用の増加につながるため、予算超過となる場合がある。	―
開発工数の見積りが難しい	―	―
工期が不明確	委託費の額が想定できないため、予算超過になる場合がある。	―
ユーザとベンダの連携が必要	発注側メンバから受注側メンバへの指示が職業安定法や労働者派遣法に抵触する可能性がある。	―
ユーザとベンダのコミュニケーションが必要	発注側メンバから受注側メンバへの指示が職業安定法や労働者派遣法に抵触する可能性がある。	―

出典：「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC,2010）

図表 4-3　非ウォーターフォール型開発における準委任契約適用時の問題点

２．２　追加調査（平成 22 年度実施）による契約事例

平成 21 年度に実施した「非ウォーターフォール型開発に関する調査」に基づく契約事例以外の契約事例も発見されたため、平成 22 年度に追加調査を行った結果を紹介する。

（1）最小限のスコープを約束する請負契約

この契約は、全体のスコープを契約段階では約束せずに、契約時点で明確になっている最小限のスコープのみを約束（最小限必須機能のみを契約書に記載）し、最小限の必須機能が完成後、以降は、顧客と追加機能を相談しながら開発していく請負契約である。

全体像はぼんやりしているが､最低限作りたいものははっきりしている場合に有効であるが、 受注者が発注者からいかに信頼されているかという点がポイントになる。

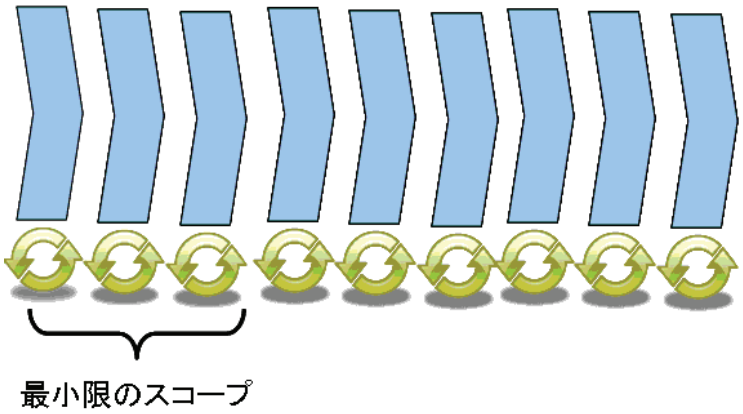
契約事例は少数に留まっている。

特徴として、

・半年～１年間の契約が多い。
・年度初めにおける発注者の予算取りのために結ぶことが多い。予算はきっちり、内容は柔軟にという姿勢で、その期間の顧客の予算を最初に押さえてもらうことが重要。
・契約時点で明確になっている機能(必須機能)を契約書に記載する。
・全体の機能でわかっている項目は全て記述するが、必須ではない。
・　契約時点で明確になっている機能が完成したら、以降は、顧客と追加機能を相談しながら見積もる。
・顧客との信頼関係の上で契約が成立している。
（契約時点で明確になっている機能以外は作成しなくても契約上は問題ないが､契約期間内では、追加機能の開発にもベストを尽くすという信頼感がベースとなっている。）
・契約期間や金額に変更がある場合のみ、変更契約を結ぶ。
・スコープを変更しても、変更契約を結ぶことはない。

が挙げられる。

図表 4-4 に「最小限のスコープを約束する請負契約」を示す。



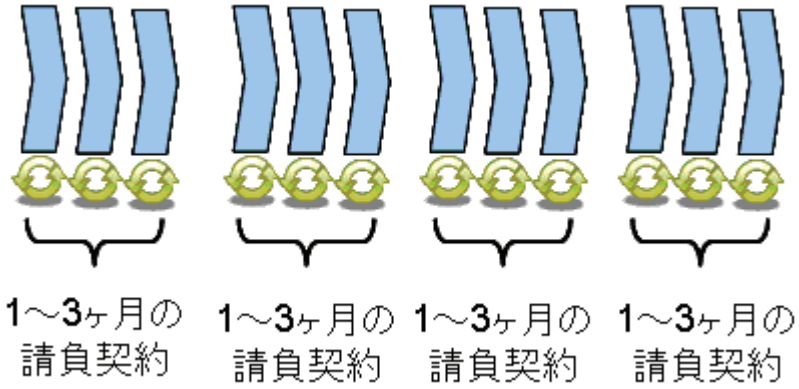
図表 4-4　最小限のスコープを約束する請負契約

（２）短期の請負契約

この契約は、全体像はぼんやりしているが、短期的に作りたいものははっきりしている場合に有効で、１～３ヶ月単位で成果物を規定する。

個々の契約は、基本的にウォーターフォール型開発での請負契約と変わらず、瑕疵担保責任は、通常すべての開発が完了してから１年間である。

図表 4-5 に「短期の請負契約」を示す。



図表 4-5　短期の請負契約

（３）機能毎の請負契約

この契約は、システム全体を 0.5～3 人月くらいの機能に分割し、優先度の高いものから、順次、五月雨式に契約し、開発していく請負契約である。

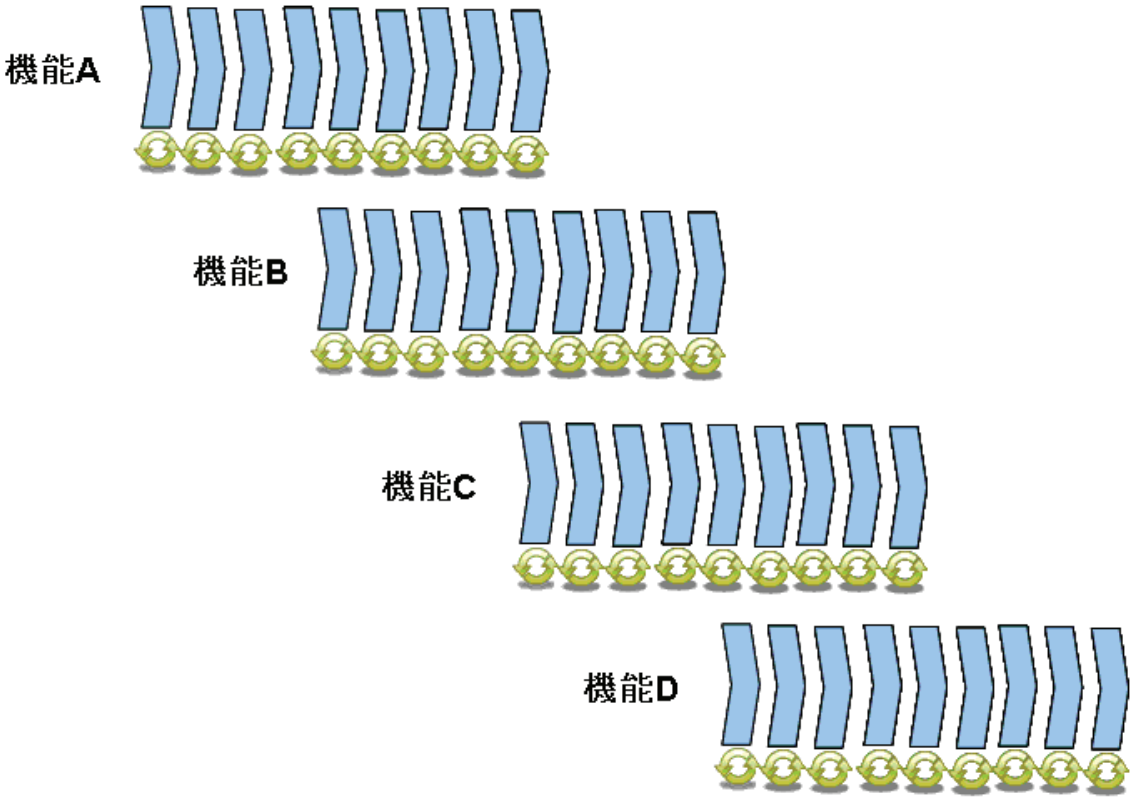
五月雨式に着手していくため、複数の契約が同時に走ることもある。

特徴として、

・ウォーターフォール型開発での請負契約と変わらない。
・１契約で複数回リリースすることもある。最終リリースを持って納品とする。
・契約時にリリーススケジュールも盛り込む。
・スコープが揺らぐことはほとんどない。
・検収もタイミングがバラバラなので、発注側の負荷が少ない。
・契約事務で揉めることはほとんどない。契約を何度も結ぶので、顧客も慣れてくる。
・機能毎に細かく契約した方がリスクヘッジになるので、顧客も嫌がらない。
・発注者が機能要件を決めきれない場合、機能毎の請負契約の前に、要件定義（２週間～１ヶ月間）の請負契約を結ぶこともある。
(成果物はその後の機能の請負契約の要件定義書、仕様書、見積書)

が挙げられる。

図表 4-6 に「機能毎の請負契約」を示す。



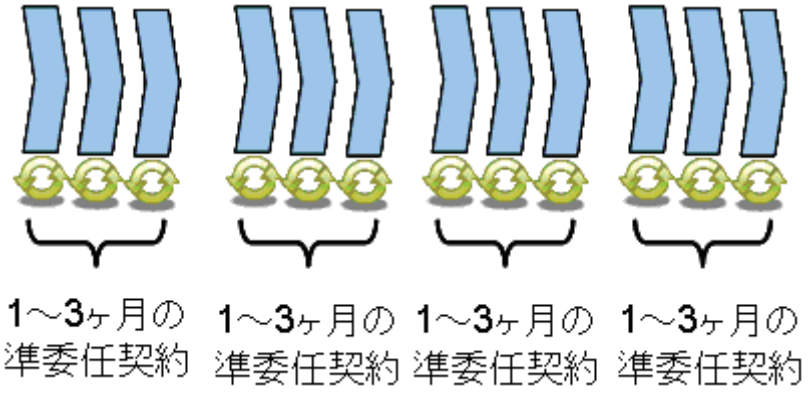
図表 4-6　機能毎の請負契約

（4）短期の準委任契約

この契約は、初期開発がひととおり落ち着きメンテナンスフェーズに入ったプロジェクトで多く採用され、たとえ１ヶ月でも成果物を明確に規定することが難しいプロジェクトに有効である。

また、 随時上がってくる顧客からの要望に応えていくことが可能だが、発注者と受注者の信頼関係ができていることが必要である。

図表 4-7 に「短期の準委任契約」を示す。



図表 4-7　短期の準委任契約

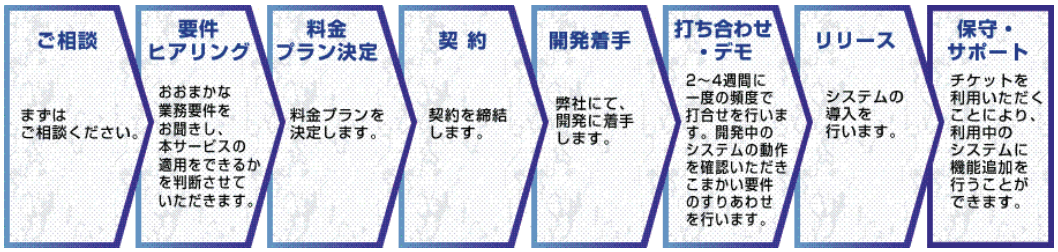
２．３　アジャイル受託開発サービスの新しい契約形態

アジャイル開発は顧客の組織やビジネスの変化に素早く対応することが可能な開発手法だが、ソフトウェア業界での受託型の請負契約は要件定義が完了してから開発見積り・契約するというやり方が当たり前となっており、顧客にアジャイル開発のメリットを実感してもらうのが難しいという課題があった。この課題を解決するために、アジャイル開発受託開発サービスの新しい契約を始めた企業がある[4]。

（１）契約形態

これまでの受託開発における一括請負型の契約では納品時に費用を全額支払うというビジネスモデルであった。新しいサービスではこのビジネスモデルから脱却し、開発したシステムを初期費用０円で提供し、その後、顧客がサービス利用料という形で月々システムサービス料を支払う契約となっている。

（２）契約の流れ



図表 4-8　契約の流れ

（３）顧客のメリット

①初期投資が不要なため、まとまった資金を調達する必要がない。

②一定量の追加開発についてはサービス利用料の範囲で対応できるため、追加開発の度に社内決済を通した後に、契約するという面倒な手続きを踏む必要がない。

③継続してメンテナンス（必要な機能の追加開発や不要な機能の削除）をし続けるので、短期的にリプレイスを繰り返すことなく、システムを長く使える。

④月額費用の中に保守・サポートも含まれる。

⑤毎月、費用対効果を測定し、効果がなければすぐに止められる。

（４）その他

①このサービスではシステムをレンタルするという形態をとるため、開発したソフトウェアの著作権はサービス会社に帰属する。

②提供するのはあくまでもシステムを使ったサービスであり、システムそのものではない。

③いつでも手数料なしで解約でき、データベースに蓄積された顧客のデータについては、顧客の手元に残る。

[4] http://www.esm.co.jp/trial/new-agile-contracts-service.html

３　アジャイル開発における契約案

３．１　アジャイル開発における契約案検討の経緯

平成21年度に実施された契約方式調査による契約事例から導かれた契約の問題点では、「アジャイル開発は変化への対応を重視し、契約時点で成果物が不明なため、「仕事の完成」に対して報酬を支払う請負契約はなじまない。」と指摘された。

しかしながら、平成21年度に調査を行ったアジャイル開発の契約の実態は、23事例の内7事例が請負契約であった[1]。

この結果を参考に、アジャイル開発における契約案を現実的な「基本契約/個別契約モデル」とあるべき姿の「組合モデル」の２つにまとめた。

①基本契約/個別契約モデル

１つの基本契約と複数の個別契約（準委任契約/請負契約）を結ぶモデル。個別契約では、プロジェクトのフェーズや状況に応じて、準委任契約/請負契約を柔軟に使い分ける。

プロジェクト全体に共通する事項を定めた基本契約を締結した上、個別の機能開発の内容が確定した部分から順次、個別契約を締結する。

プロジェクトの初期フェーズでの個別契約は準委任契約とし、その後、ユーザとベンダ[5]が相互の能力を把握し、相互コミュニケーションが安定した段階での個別契約は、請負契約とするケースが考えられる。

②組合モデル

システム開発プロジェクトにおいて、ユーザは費用を、ベンダ（複数社も可）は開発要員を出し、一つのシステムをユーザとベンダの企業が共同で企画、製作するための組織－全体プロジェクト推進共同企業体－を作り、開発を行うモデルである。スキームとしては，民法上の組合（任意組合）を用いている。[6]

土木建築業界において、一つの工事を施工する際に複数の企業が共同で工事を受注し施工するための組織－共同企業体（JV－Joint Venture）と似ており、アジャイル開発＝ユーザとベンダの共同事業と捉え、共同事業（共同作業）契約を締結し、アジャイル開発の特徴である、ユーザとベンダの緊密な協力体制をそのまま契約に反映する。

組合モデルでは、システム開発プロジェクトのコーディネートとプロジェクトマネジメントのみをJV（組合）が担当し、実際の開発は、JVがベンダに委託して行うというモデルを想定している。

---

[5] モデル契約では、「ユーザ」、「ベンダ」の言葉を使用しているため、4章と付録1～6では、「顧客」を「ユーザ」、「開発側」を「ベンダ」として使用する。

[6] 本契約モデルのスキームとして民法上の組合（任意組合）を用いているのは、①共同体の組成が容易であること（会社設立と異なり、設立事務、資本金の払込み、登記等が不要）、②ベンダによる労務出資が可能であること（株式会社、合同会社、有限責任事業組合は労務出資ができない）などを理由としている。

上記２つの契約モデル以外に、「基本契約/個別契約モデル」を検討する前提として、プロジェクトのフェーズに応じて準委任契約と請負契約を使い分ける「準委任/請負契約モデル」の検討を行った。しかし、アジャイル開発の重要な要素である「ユーザとベンダの相互協力」を盛り込むことが難しかったため、今回のモデル契約の対象とはしなかった。

以降では、上記２モデルについて、詳しく説明する。

３．２　アジャイル開発における基本契約/個別契約モデル契約案

（１）契約の前提条件

①ユーザとベンダ、双方の契約当事者に対等に交渉力がありかつ技術力を有する
②開発モデルは、アジャイルソフトウェア開発とする
③開発対象は、業務システムやパッケージソフトウェア等
③開発プロセスにおいて、共通フレーム 2007（SLCP-JCF2007）の適用が可能であること
④契約スタイルは、多段階契約、請負型、準委任型等の契約があり、契約の特徴として変更管理手続きが挙げられる。

（２）契約で想定している内容

この契約で想定しているケースは、ユーザによってプロジェクト全体についての企画[7]が終わっており、プロジェクトの全体概要及びその中で開発する機能について、ある程度イメージを持っていることを前提としている。

ユーザは、チームの一員として参画し、タイムリーな意思決定を行い、ベンダは顧客のビジネスを理解し、ユーザとベンダが共にスコープの変更を許容し、ゴールを１つにした意識をもつことが肝要となる。

この契約は、基本契約と個別契約に分かれており、基本契約では、プロジェクトの概要、ユーザとベンダの協議の場である連絡協議会、合意事項に変更が生じたときの変更管理手続といった、プロジェクト全体に関わる事項を規定している。他方、個別契約では、具体的な作業内容を法的拘束力のある形で定めることとしている。そして、個別契約は、請負型、準委任型のいずれにも対応できるものとしている[8]。

（３）想定しているアジャイル開発プロセスモデル

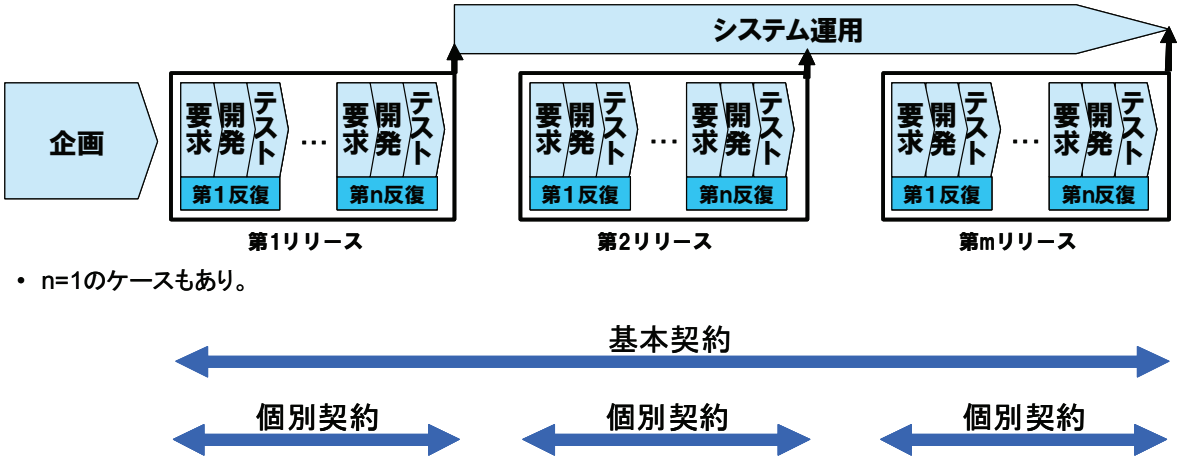
契約モデルの検討にあたっては、「第二部アジャイル開発のモデル」で想定しているアジャイル開発プロセスモデルの基本的なモデルである「システム全体の要求・アーキテクチャ設計・基盤開発を行わずに、直ぐに第１リリースの開発に取りかかるモデル」を基に、契約形態を検討した。

図表 4-9 に、「基本契約/個別契約モデル契約のアジャイル開発プロセスモデル」を示す。

---

[7] ビジネスゴールをもとに、プロダクト/プロジェクトのコンセプト、システム化の目的や目標となるゴール、ビジネス要求に紐付けされたシステム要求と主要な機能について立案し、ステークホルダー間で合意を形成する。

[8] なお、開発対象とする機能の概要がスムーズに確定しない等の理由から、ある個別契約と次の個別契約との間に、時間的間隔が空くようなケースでは、ベンダが余剰人員を別のプロジェクトに移してしまい、その結果プロジェクト進行に支障をきたす恐れがあるといった、運用上の課題があり得る。本契約案の検討においては、こうした運用上の課題を全て検討したわけではない。



図表 4-9　基本契約/個別契約モデル契約のアジャイル開発プロセスモデル

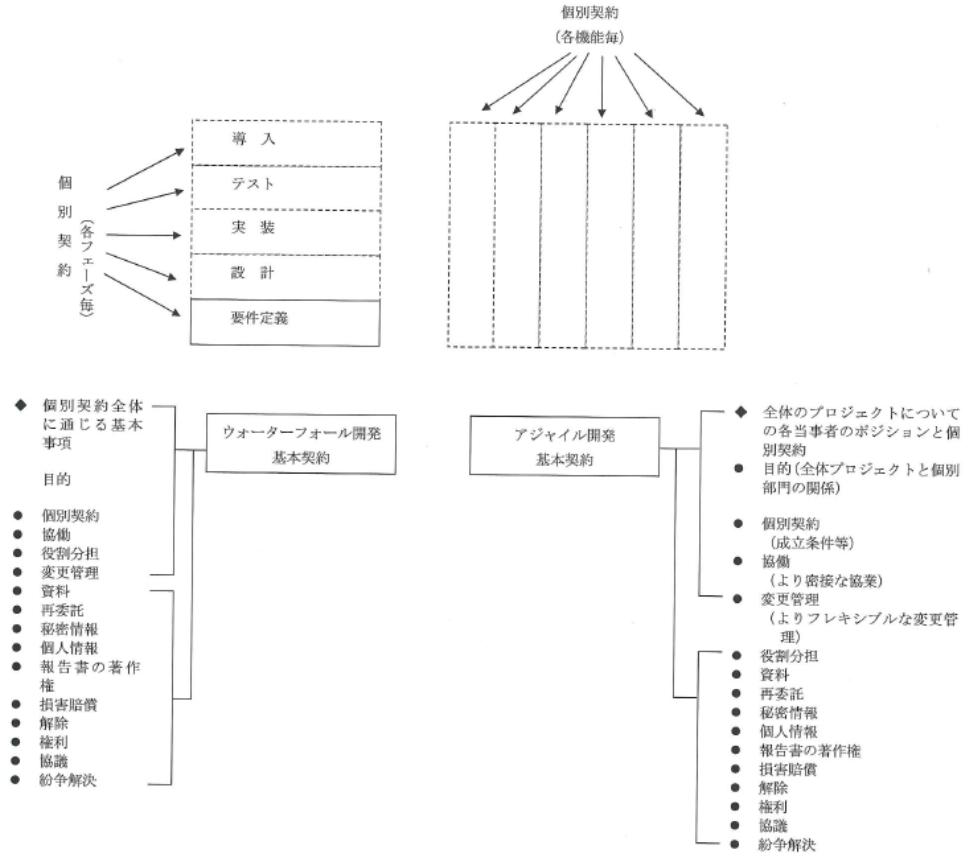
図表 4-9 で使われている用語の説明は、「第二部　アジャイル開発のモデル　２．１　アジャイル開発のプロセスモデル　（４） アジャイル開発のプロセスモデルで使用されている用語の説明」を参照。

（４）契約の概要

「３．２　リーンソフトウェア開発における 6 つの契約」で紹介されている「多段階契約（multistage contract）」をベースに、プロジェクトのフェーズに応じて、準委任契約/請負契約を使い分ける契約である。

①基本契約と個別契約（準委任契約/請負契約）を結ぶ契約であるが、プロジェクトのフェーズや状況に応じて、準委任契約/請負契約を柔軟に使い分ける、
②プロジェクト全体に共通する事項を定めた基本契約を締結した上、個別の機能開発の内容が確定したものから順次、個別契約を締結する。
③基本契約では、プロジェクト全体におけるユーザとベンダの協力体制を強調することで、両者のポジションを明確にする。
④基本契約において、ユーザとベンダが共同して、システム開発のためのプロジェクトを進行させることを規定し、ユーザとベンダは、単なる発注者と受託者ではなく、相互に協力すべき立場にあることを明確にしておく。
⑤請負契約におけるベンダの完成義務とユーザの協力義務、準委任契約におけるベンダの義務（広い意味での支援義務）及びユーザの責任を明確にしておく。

（5）ウォーターフォール型開発契約書との対比



図表 4-10　ウォーターフォール型開発契約書との対比

（6）基本契約/個別契約モデルの基本契約案

アジャイル開発における基本契約/個別契約モデルの基本契約案に解説を付加したものを、付録１に添付する。

（7）基本契約/個別契約モデルの個別契約（請負型）案

個別契約（請負型）は、全体プロジェクトにおいて開発することが決定した一定の機能群（例えばリリース単位で一括りにした範囲）につき、ユーザがベンダにその開発を委託するものである。そのため、請負型の個別契約を行う場合は、委託対象となる機能群の概要が確定している必要がある。

個別契約（請負型）請負型では、個別契約（準委任型）とは異なり、ベンダは当該機能群の完成義務を負い、また瑕疵担保責任を負うことになる。

アジャイル開発における基本契約/個別契約モデルの個別契約（請負型）案に解説を付加したものを、付録２に添付する。

（8）基本契約/個別契約モデルの個別契約（準委任型）契約案

個別契約（準委任型）は、ユーザがベンダに対し、ベンダが有する情報処理技術に関する業界の一般的な専門知識及びノウハウに基づく業務遂行を、一定の期間、委託するものである。

個別契約（準委任型）では、個別契約（請負型）とは異なり、ベンダは当該機能群の完成義務及び瑕疵担保責任を負わない。もっとも、ベンダは業務遂行にあたり、善管注意義務（善良な管理者の注意義務）を負担し、ベンダの行った業務が業界の一般的な水準を下回っていた場合には、ベンダは債務不履行責任を負う恐れがある。

アジャイル開発における基本契約/個別契約モデルの個別契約（準委任型）案に解説を付加したものを、付録 3 に添付する。

（9）基本契約/個別契約モデルの契約で使用する「連絡協議会議事録（サンプル）」

アジャイル開発における基本契約/個別契約モデルの契約で使用する「連絡協議会議事録（サンプル）」を、付録 4 に添付する。

３．３　アジャイル開発における組合モデル契約案

（１）契約の前提条件

①ユーザとベンダ、双方の契約当事者に対等に交渉力がありかつ技術力を有する

②開発モデルは、アジャイルソフトウェア開発とする

③開発対象は、業務システムやパッケージソフトウェア等

③開発プロセスにおいて、共通フレーム 2007（SLCP-JCF2007）の適用が可能であること

④契約スタイルは、民法上の組合（任意組合）型を用いているが、システム開発プロジェクトのコーディネートとプロジェクトマネジメントのみをJV（組合）が行い、実際の開発は、JVがベンダに委託して行うというモデルを想定している[9]。

（２）契約で想定している内容

この契約で想定しているケースは、ユーザとベンダが、パッケージソフトやクラウドシステムを共同で企画、製作し、完成した成果物を運用して収益を得て、それを分配するような場合である。長期間の継続を予定しているため、ユーザとベンダとの間に既に信頼関係があり、かつ、出資額を決めるため契約締結段階でプロジェクトの全体概要、概算予算が決定している必要がある。

この契約は、ユーザ、ベンダ及びプロジェクト運営には関与しない投資家が民法上の組合を構成して、各自が出資（ユーザ及び投資家は金銭、ベンダは労務）を行い、得られた収益を出資割合に応じて分配するものである（ベンダはプロジェクトマネジメント業務に係る労務提供を出資として行う。）。

プロジェクトの具体的内容については、ユーザとベンダが、組合の業務執行権限を有する業務執行組合員として、連絡協議会を開催し、その中で決定、遂行していくことを予定している。そして、ユーザとベンダの協議の結果、プロジェクトにおいて開発が決定した機能群については、組合からベンダに対して、リリース単位で請負契約に基づく開発委託を行うこととしている。

なお、本契約はあくまで試案であり、改善の余地が大きい。また、ビジネスモデルによって組合運営のための組織体制、資金の出資と分配、解散時の処理などは大きく異なることから、本試案ではこれらを未検討事項とした。例えば、組合のまま事業化することはないであろうことから、事業化に際しての株式会社への移行の取決めなどは考慮していない。さらには、組合における構成員のパススルー課税（法人税、消費税）、資産計上等の処理方法、組合として収益が上がった際の権能なき社団としての地方税の処理、などの規定も未検討である。

---

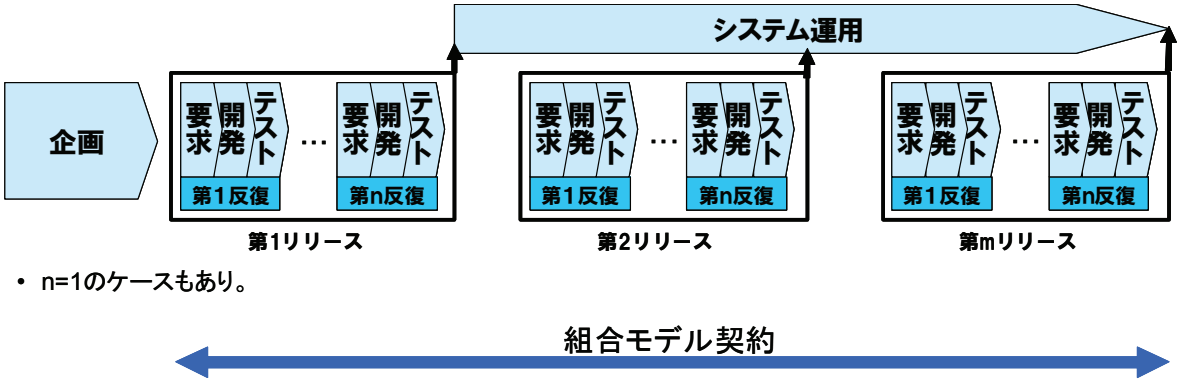
[9] 本契約モデルのスキームとして民法上の組合（任意組合）を用いているのは、①共同体の組成が容易であること（会社設立と異なり、設立事務、資本金の払込み、登記等が不要）、②ベンダによる労務出資が可能であること（株式会社、合同会社、有限責任事業組合は労務出資ができない）などを理由としている。

個々のモデルに応じて弁護士、会計士の助言をもとに、改良の上、利用していただきたい。

（3）想定しているアジャイル開発プロセスモデル

契約モデルの検討にあたっては、「第二部アジャイル開発のモデル」で想定しているアジャイル開発プロセスモデルの基本的なモデルである「システム全体の要求・アーキテクチャ設計・基盤開発を行わずに、直ぐに第１リリースの開発に取りかかるモデル」を基に、契約形態を検討した。

図表 4-11 に、「組合モデル契約のアジャイル開発プロセスモデル」を示す。

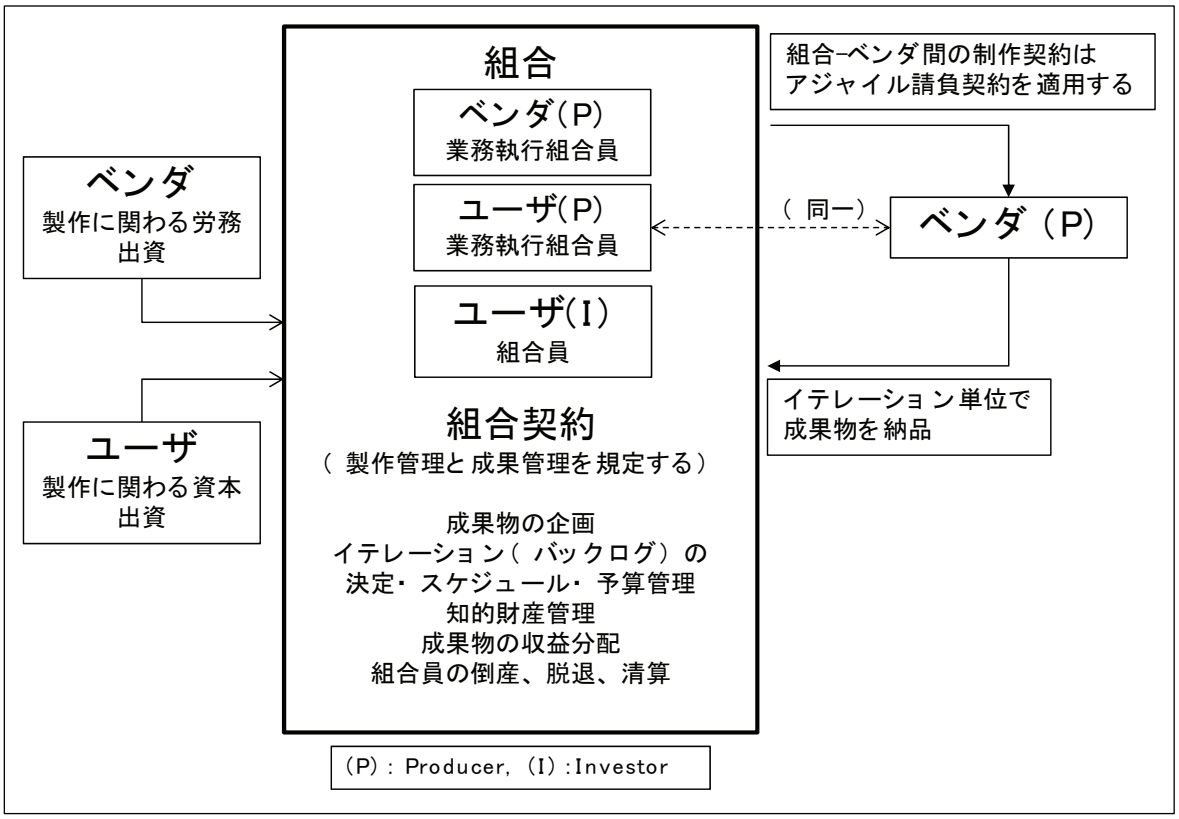


図表 4-11　組合モデル契約のアジャイル開発プロセスモデル

（4）契約の概要

アジャイル開発＝ユーザとベンダの共同事業と捉え、共同事業（共同作業）契約を締結し、アジャイル開発の特徴である、開発側と顧客の緊密な協力体制をそのまま契約に反映する。

図表 4-12 に、「組合モデル契約書（共同事業モデル）案のイメージ」を示す。



共同事業において生じたアウトプットについては、出資割合に応じて分配される。

図表 4-12　組合モデル契約書（共同事業モデル）案のイメージ

（5）アジャイル開発における組合モデルの契約案

アジャイル開発における組合モデル契約案に解説を付加したものを、付録 5 に示す。

４　海外で提唱されているアジャイル開発に適した契約形態

日本におけるソフトウェア開発では、請負契約、準委任契約、システム・エンジニアリング・サービス（SES）契約の 3 種類の契約が一般的に行われている。
海外で提唱されているアジャイル開発に適した契約形態にはどのようなものであるか、文献調査を行った。その概要を以下に述べる。

４．１　次のアジャイルソフトウェアプロジェクトのための 10 の契約[10]

米国のスクラムトレーナの Peter Stevens は、スクラムとアジャイル開発プロジェクトで利用することを前提とした 10 種類の契約形態を比較している[2]。
Peter Stevens が比較を行った契約形態は、次の 10 種類である。

①スプリント契約（Sprint Contract）
②定額請負型（Fixed Price & Fixed Scope）
③実費精算契約（Time and Materials）
④固定仕様、シーリングありの準委任契約
（Time and Materials with Fixed Scope and Cost Ceiling ）
⑤シーリングありの準委任契約
（Time and Materials with Variable Scope and Cost Ceiling）
⑥段階的開発契約（Phased Development）
⑦ボーナス/ペナルティ条項（Bonus/Penalty Clauses）
⑧固定利益（Fixed Profit）
⑨Money for Nothing, Changes for Free（早期中止、変更無料）
⑩ジョイントベンチャー（Joint Venture）

この「10 種類の契約」は、米国における従来型の契約モデルも含まれており、すべてがアジャイル開発に適したものとは言えないが、どのような契約がアジャイル開発に適しているのかを考える出発点になると思われるので、その概要[3]を以下に示す。

[10] 10 種類の契約について解説したアジャイルソフトウェア開発に適した契約」を付録 6 に添付する。

（1）スプリント契約（Sprint Contract）

・スプリント契約は、実際の商取引で利用される契約ではなく、スクラムにおいて、プロダクトオーナーと開発チームの間でスプリントごとに交わされる合意（スプリント計画）のことである。
・開発チームは、その反復の終了までに合意した機能について納品し、プロダクトオーナーは、反復が終了するまでスコープを変更しないことが前提となる。
・プロジェクトの最初に基本契約を締結し、その後、スプリント（通常 30 日）毎に個別契約を結ぶ。
・本契約は、アジャイル開発には適していると思われるが、スプリント（通常 30 日）毎の契約手続きは、煩雑になると思われる。

（2）定額請負型（Fixed Price & Fixed Scope）

・この契約は、価格とスコープを最初に固定するので、日本における一般的な請負契約に相当する。
・合意した納品物について、納品を行う。
・定額契約であるため、作業量が増加すると開発側の利益は減少する。
・仮にプロジェクトが遅延して、コストが予定以上になった場合でも決まった金額しか支払わなくてよい。
・原則としてスコープの変更は行わない。作業量や開発期間に影響を及ぼすようなスコープ変更を行う場合には契約変更が必要である。
・この契約は、スコープを最初に固定してしまうのでアジャイル開発には適していないと思われる。

（3）実費精算契約（Time and Materials）

・実費精算契約（Time and Materials）は、日本における準委任契約に相当する。プロジェクトに投入した工数（労働量）や材料費に応じて支払い額が決まる。
・開発側は作業量に応じた請求を行い、変更に関するリスクは、顧客が負っている
・この契約方式においては、開発側には完成責任がなく、支払いの上限金額が決まっていないため、プロジェクトのリスクはすべて顧客が負い、顧客はある時点で、プロジェクトの終了を検討するようになる。
・スコープをいつでも変更できる点で、アジャイル開発に適しているが、顧客側がプロジェクトのリスクを負う覚悟があり、かつ、適切なプロジェクト管理を行える場合に限られると思われる。

（4）固定仕様、シーリングありの準委任契約
（Time and Materials with Fixed Scope and Cost Ceiling ）

・スコープ（仕様）が固定され、かつ支払い金額に上限がある準委任契約であり、上限に達するまでは、実費精算契約と同様に、開発側は作業量に応じた請求を行うが、コストが上限に達すると支払いが受けられなくなる。
・要求仕様を満たすものが完成した時点でプロジェクトは終了する。
・顧客からすると、開発が早期に終了すれば、固定価格/固定スコープ型よりもコストを抑えることができる。
・この契約は、スコープを最初に固定してしまうのでアジャイル開発には適していないと思われる。

（5）シーリングありの準委任契約
（Time and Materials with Variable Scope and Cost Ceiling）

・支払い金額に上限がある準委任契約であるが、スコープが固定されていないという点で「固定仕様、シーリングありの準委任契約」とは異なる。
・実費精算契約と同じだが、コストの上限を設けておくことにより、顧客のリスクが軽減される。
・予算の制限とスコープの変更を組み合わせることにより、予算の範囲内で望ましい結果を得ようと、顧客と開発側の双方が同じ視点を持つ。
・スコープが固定されていないので、アジャイル開発に適しているが、実費精算契約と同様、顧客がプロジェクトのリスクを負う覚悟があり、かつ、適切なプロジェクト管理を行えるかどうかが重要になる。

（6）段階的開発契約（Phased Development）

・一定期間（たとえば四半期）でプロジェクト期間を分割し、その期間ごとに達成すべきスコープを定めて準委任契約を結ぶという方法。その期間で開発した機能のリリースに成功すれば、次の期間の契約が結ばれる。
・開発側にはその期間の開発を成功させようというインセンティブが働くと考えられ、顧客には、できるだけ早く高品質なシステムを完成させたいというニーズがあるので、顧客と開発チームとの間によい協力関係が生まれる可能性が高い。
・期間ごとのスコープは固定しても、全体としてのスコープは固定しなくてよいので、アジャイル開発に適している。

（7）ボーナス/ペナルティ条項（Bonus/Penalty Clauses）

・開発が予定より早く完了すればボーナスを受け取り、遅延すればペナルティ（罰金）が科せられるという契約である。
・スコープ（仕様）が固定されていることが前提となる。
・顧客に開発を早期に終わらせるインセンティブが働かないと、時間をかけたにもかかわらず成果を望めない可能性がある。
・スコープが固定で、両者に協力関係が生まれる可能性が小さいことを考えると、アジャイル開発には適していないと思われる。

（8）固定利益（Fixed Profit）

・目標納期前にプロジェクトが完了した場合は、それまでの発生経費に比例する形で利益を上乗せした支払いを行い、目標納期後に完了した場合は、発生経費に定額の利益を上乗せした支払いを行う契約。
・実費精算契約（Time and Materials）に近いが、予定期日を超えた場合には、超過分についてはコスト分しか支払われない。つまり開発側の利益は、予定を超過しても、予定通りに開発を終了した場合と同じになる。
・早期にプロジェクトが終了すると、顧客は経費が抑えられ、開発側は利益率が高くなるため、双方に、早期終了のインセンティブが働く
・売上高を重視する日本の企業を考えると、開発側に早期完了のインセンティブがあるとは思えず、スコープが固定であることを考えると、この契約はアジャイル開発には適していないと思われる。

（9）Money for Nothing, Changes for Free

・基本は、スコープの変更が可能で、予算上限のある準委任（実費清算型）契約。
・早期完了（あるいはプロジェクト中止）になった場合には、開発側は、キャンセル料として残余期間で開発側が得るべき利益分を受け取る権利があり、どの時点でプロジェクト完了、あるいは中止になったとしても開発側が得られる利益は一定となる。
・スコープの変更は自由に行うことができ、かつどの時点で開発を終了（中止）させるかについては顧客の自由である。早期に終了（中止）すれば支払い金額が少なくなるので、顧客には早期完了のインセンティブがある。一方開発側からみれば、早期完了すれば、売上は少なくなるものの利益は変わらず、利益率を考えるとメリットは大きい。開発側が売上高ではなく利益率重視であれば、早期完了のインセンティブがあることになる。
・スコープが固定でなく、両者に早期完了のインセンティブがあることを考えると、この契約は、アジャイル開発には適している。

（10）ジョイントベンチャー（Joint Venture）

・顧客、開発側双方が、共同出資の形でプロダクトに投資する契約。
・開発フェーズに両者の間で金銭の移動が生じるのではなく、そのソフトウェアを利用することで得られる利益を共有する。
・顧客と開発側でつくるチームを仮想的な企業と考えれば、社内開発と同じであり、アジャイル開発には適している。しかし、スコープ（仕様）や納期などについて両者の意見、優先度が一致しない可能性がある。場合によっては両者が対立し、意思決定に時間を要する可能性がある。

４．２　リーンソフトウェア開発における６つの契約

「リーンソフトウェア開発」[4]で取り上げられている非ウォーターフォール型開発に適した契約は、以下の６種類であり、それぞれの概要を示す。

①定額契約
②時間資源契約
③多段階契約
④目標コスト契約
⑤目標スケジュール契約
⑥利益共有契約

（１）定額契約（fixed-price contract）

日本における一般的な一括請負契約（請負契約）に相当する。

・本来顧客が対処すべきリスクを開発側に移し、リスクを開発側に持たせることで、顧客を守るために設計されている。
・作業前にコストを見積もるという点で、大きなリスクが存在する。
・入札による調達の場合、開発側は、最低額で入札し、後で発生する「変更」によって利益を上げるというやり方が助長される。
・顧客は契約時点の仕様変更を強く避ける傾向になり、開発側は顧客の真の要求を満たさなくてもリリースしてしまおうという気になりがちで、顧客満足は低くなる。

（２）時間資源契約（time-and-materials contract）

日本における一般的な作業請負契約（時間精算の準委任契約）に相当する。

・リスクは顧客が持つため、開発側は、契約が続くかぎり有利な取引だと考え、生産効率を上げようという気にならない。
・時間資源契約では、「取引コスト」（契約要件の管理、許される費用の整理等の管理コスト）が無視できないほどかさむ。
・時間資源契約を行う場合、シーケンシャル開発より、コンカレント開発（同時並行プロセスによる開発）の方が安全である。
・開発側と顧客の職場にいる実作業者間の継続的な協力が必要となる。

（３）多段階契約（multistage contract）

多段階契約（multistage contract）には、以下の２種類がある。

①大規模な定額に先立って結ばれる多段階契約

・１、２回短期契約を結び、システム全体に応札できる程度に問題について学習した後に、定額契約を結ぶ。

②全開発期間にわたり結ばれる多段階契約

・アジャイル開発に向いている。
・アジャイル開発では、初期段階で取り決めたマスター契約によって管理され、各イテレーション（反復）ではそのマスター契約にしたがって作業発注が実行される。
・顧客、開発側双方に契約を破棄する機会が頻繁にあることが最大のリスクである。
・リスクを緩和するには、信頼関係を築き上げることが重要である。

（４）目標コスト契約（target-cost contract）

目標コスト契約（target-cost contract）とは、変更分を含む全体コストが顧客と開発側の双方の責任となる契約のことである。

・目標コストを上回った場合、双方の支出が増え、全コストが目標を下回った場合、双方が利益を共有する。
・長く働いても開発側に追加の利益は入らないが、目標のコストや期間を下回れば利益を受け取れる。
・相互の仕事がなされるまで詳細は不明であることを前提に目標コスト、目標スケジュールを合意する。
・実コストが目標コストと同じである必要はない。目標コストを上回るコストについては公平に負担し、目標コストを下回った場合の利益についても公平に分配する。
・目標コスト契約を結ぶことで、顧客は機能要求を超えないようにしよう、という気になり、開発側は作業を目標コスト内で完成させようという気になる。

（5）目標スケジュール契約（target-schedule contract）

目標コスト契約（target-cost contract）がコスト優先するのに対して、目標スケジュール契約（target-schedule contract）は、コストよりスケジュールが重要なときに結ぶ契約のことである。

・スケジュールをコストより優先させ、必要に応じてリソースを追加したり、部品のライセンスを取得したりする。
・時間がなくなった時点で契約は終了し、優先度の低い機能は手を付けられず残るが、リリースを製品マーケティング全体の意図に合わせることができる。
・目標コスト契約において、資源とスケジュールを固定することで、目標スケジュール契約として実行できる。

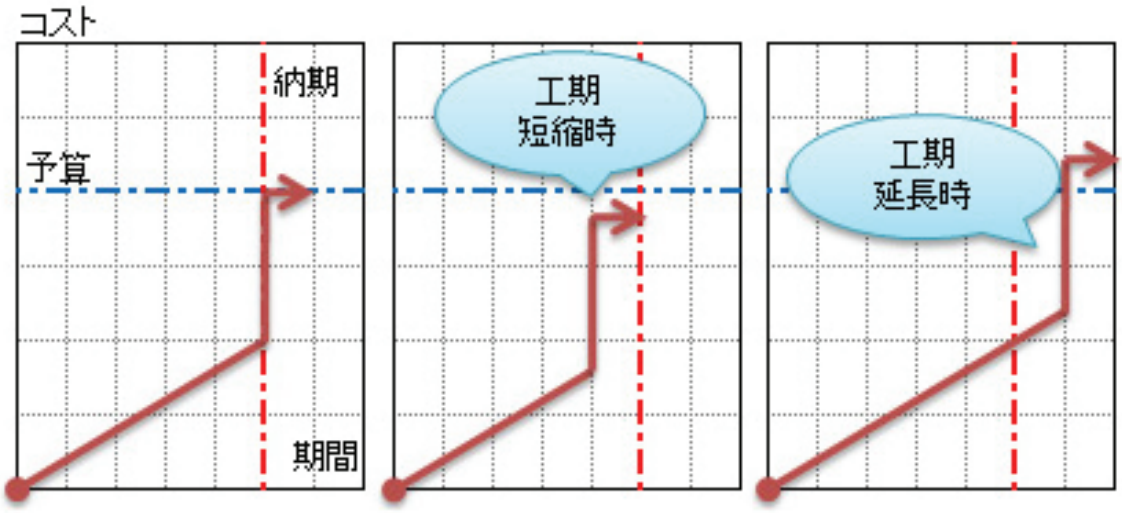
（6）利益共有契約（shared-benefit contract）

利益共有契約（shared-benefit contract）とは、顧客と開発側双方が作業のリスクと報酬を共有する契約のことである。

・顧客と開発側双方が提携し、リスクを共有しながら、相互の協力で生み出した利益をあらかじめ決めておいた配分率で分け合う成果報酬型契約。
・契約の文言そのものではなく、システムが進化することを許す方が、最初に仕様を決めてしまうよりも会社の利益が多いという認識に立つ。
・顧客と開発側相互の利益を達成するために、時とともに何を実施するかを双方が修正することを想定している。

４．３　デンマークの事例による第３の契約

デンマークのコンサルティング企業である BestBrains 社では、非ウォーターフォール型の開発プロジェクト分野を対象とした新しい契約方式[5]を試行している。



出典：「アジャイルが大企業に採用されるために解決すべきたったの２つの課題」[11]

図表 4-13　デンマークの事例による第３の契約

この契約方式は、以下の特徴を持つ[6]。

・予算の半分を工数に応じた支払いにあてる（変動費）。

・変動費を支払った残り半分を納品が完了した時点で支払う（固定費）。

その結果、発注側は開発工数変動による費用変動幅が少なくなり、開発側は、開発期間中に最低限の開発費用が回収できるメリットがある。

システムが早く完成すると、発注側は工数に応じて支払う変動費部分が減り、開発側も拘束期間が短くなる上に成功報酬は満額もらえる。そのため、発注側、開発側双方とも、早く完成させるためのモチベーションが高くなる。

逆に、完成が遅れた場合は、発注側は延長期間分の変動コストを支払わなければならなくなり、開発側も完成時の報酬が先延ばしになってしまう。そのため、発注側、開発側双方とも、遅れを取り戻して早く完成させようというインセンティブが働く。しかも、一括請負契約や準委任契約のようにどちらか片方だけがリスクを負うような事態は避けることができる。

つまり、発注側と開発側がプロジェクトの成否による利害を共有しており、成功するとどちらも得をするし、失敗するとお互いに損をするという契約モデルになっている。

---

[11] アジャイルが大企業に採用されるために解決すべきたったの 2 つの課題」より引用 http://enterprisezine.jp/article/detail/2391

４．４　変則的な準委任契約[3][12]

投入された労働量に比例してコストは増大する。そのコストに上乗せされる利益額を、労働投入量に逆比例して少しずつ減少するように設定すれば、投入労働量に比例して支払い金額（＝報酬総額＝コスト＋利益）は増加するが、利益は減少する契約が実現できる。条件としては、コストの増加幅より利益の減少幅が小さいことである。

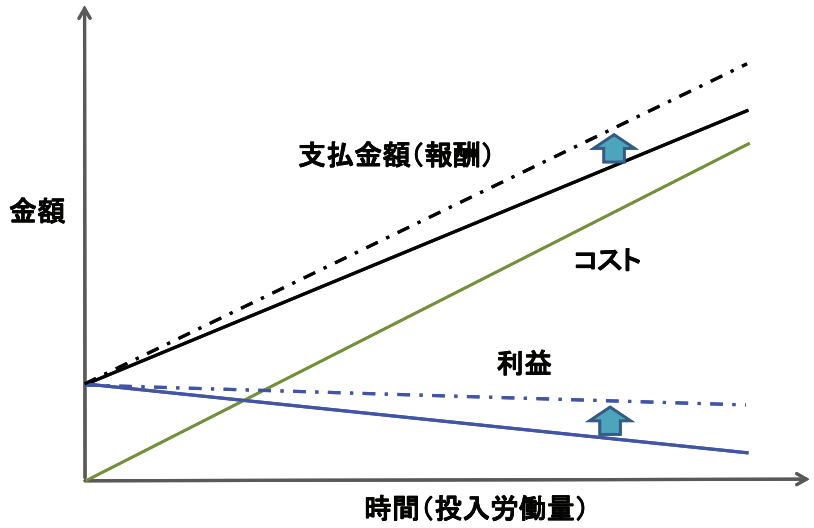


図 4-14　変則的な準委任契約

この変則的な準委任契約では、開発期間が伸びれば（投入労働量が増えれば）、発注側が支払う金額は増大するが、開発側の利益額は減少する。つまり、発注側から見れば早期に終了した方が支払い金額は少なくて済むので、早期終了に対するインセンティブが生まれる。一方、開発側も早期に終了した方が利益額を大きくすることができるので、早期終了のインセンティブが生まれる。

この契約方式が成立する前提条件は、コストをある程度正確に計算でき、双方がそれに合意できることである。

さらにその上で、開発側と発注側の早期終了に対するインセンティブの大きさがほぼ等しくなるように利益と支払い金額の傾きを決められるかどうかが問題になる。

ちなみに、この図における利益の直線を水平にすると、ピーター・スティーブンスの「あなたの次のアジャイルソフトウェアプロジェクトのための 10 の契約」における「Money for Nothing, Change for Free」と同じになる。

また、「４．３　デンマークの事例による第３の契約」も、この契約方式の一つのパターンとして位置付けられる。

12　「変則的な準委任契約」は、国内で提唱されている。

５　まとめ

「２　非ウォーターフォール型開発における契約の調査事例と契約の問題点」で見たように、23 事例の内 7 事例が請負契約であった。この事実からみれば、非ウォーターフォール型開発であっても請負契約が適用できるようにみえる。しかし、これは開発側と顧客側との間に強い信頼関係があるために多少の問題が発生しても訴訟にいたるような大問題にならず、うまく適用できているように見えている可能性が高い。もし、両者の間で成果物に対する評価や見解の相違などが生まれ、トラブルに発展した場合、非ウォーターフォール型開発という開発プロセスと契約方式の齟齬が明らかになる可能性が高い。

一方、23 事例の内、8 事例で用いられていた準委任契約は、スコープの変更が自由であり、非ウォーターフォール型開発に適した契約であると言える。しかし、準委任契約の場合、開発に関するリスクのほとんどは顧客側が負うことになる。したがって、準委任契約によって情報システムを開発するユーザ企業は、開発ベンダと同等の技術力と対等な交渉能力を有していることが望まれる。（なお、「１・３　非ウォーターフォール型開発にふさわしい契約検討の方法」で述べたように、システム開発のイニシアティブを顧客側（発注者側）が持ち、顧客側が開発ベンダの従業員に対して直接指揮命令を行う場合や、顧客側が開発ベンダに対して設備・資材を無償提供する場合など、開発ベンダが顧客から業務上独立した立場にあるとはいえないケースでは、労働者派遣契約を用いるのが適切である。）

そこで、「本 PT」では２つの契約モデルを提案する。

第１の契約モデル（基本契約/個別契約モデル）は、「多段階契約（multistage contract）」をベースに、プロジェクトのフェーズに応じて、準委任契約/請負契約を使い分ける契約である。この契約は、基本契約と個別契約に分かれており、基本契約では、プロジェクトの概要、ユーザとベンダの協議の場である連絡協議会、合意事項に変更が生じたときの変更管理手続といった、プロジェクト全体に関わる事項を規定し、個別契約では、具体的な作業内容を法的拘束力のある形で定める。個別契約は、請負型、準委任型のいずれにも対応できるものとし、契約の頻度を考え、可能なかぎり簡易なものとしている。

第２の契約モデル（組合モデル）は、非ウォーターフォール型開発を、開発側と顧客側の共同事業と捉え、開発側と顧客側が民法上の組合を組成する形の契約である（なお、組合には、プロジェクト運営には関与せず、資金提供だけを行う投資家が加わることも想定している。）。この共同事業において生じたアウトプットについては、ユーザ、ベンダ（及び投資家）の出資割合に応じて配分がなされる。

「本 PT」では、この２つの契約モデルに基づき、それぞれ契約書のひな形を作成した。可能であれば次年度以降に、非ウォーターフォール型開発モデルを用いた開発プロジェクトに対して、この２つの契約書のひな形を用い、その有効性と課題を明らかにすることが望まれる。

付録１　アジャイル開発における基本契約/個別契約モデルの基本契約案

本契約の前提条件：
契約当事者：対等に交渉力がありかつ技術力を有するユーザとベンダ
開発モデル：アジャイルソフトウェア開発 スクラム
対象システム：業務システム、パッケージソフトウェア等の開発
プロセス：共通フレーム 2007（SLCP-JCF2007）の適用が可能
特徴：多段階契約、変更管理手続き、請負型[13]、準委任型

本契約が想定するケース：
本契約の想定するケースは、ユーザによってプロジェクト全体についての企画が終わっており、プロジェクトの全体概要及びその中で開発する機能について、ある程度イメージを持っていること、としている。
本契約は、基本契約と個別契約に分かれており、基本契約では、プロジェクトの概要、ユーザとベンダの協議の場である連絡協議会、合意事項に変更が生じたときの変更管理手続といった、プロジェクト全体に関わる事項を規定している。他方、個別契約では、具体的な作業内容を法的拘束力のある形で定めることとしている。そして、個別契約は、請負型、準委任型のいずれにも対応できるものとしている。

本契約の構成概要：
(1) 本契約は、基本契約と個別契約（準委任契約/請負契約）を結ぶ構成であるが、このうち基本契約では、プロジェクト全体に関する共通事項を規定している。
(2) 基本契約ではプロジェクトの全体に関する内容が別紙に記載されるが、当該内容は法的拘束力を持たず、プロジェクトの過程で変更されることを予定している（第 2 条、第 3 条第 3 項）。
(3) ユーザとベンダは連絡協議会を構成し（第 6 条）、開発対象とする機能の内容を検討・決定する。開発対象機能が決定すれば、順次個別契約（準委任契約/請負契約）が締結され（第 3 条第 1 項）、個別契約に従って、具体的な開発が進められる。
(4) 個別契約に基づく開発の進捗管理、リスク管理等についても、連絡協議会で協議が行われる（第 6 条）。
(5) ユーザとベンダは、連絡協議会で決定された事項には（契約内容に反しない限り）従わなければならないが、一旦合意した事項について、当事者の一方から変更要請が出た場合には、連絡協議会において、変更の協議がなされる（第 4 条）。なお、変更協議を行っても協議が調わないまま、一定期間を経過した場合は、両当事者は、変更協議の原因となっている個別契約を清算できる。
(6) その他、ベンダが開発を第三者に再委託するためには、ユーザの事前の承諾を得なければならない（第 8 条）。また、損害賠償については、個別契約において、その上限額、請求可能期間を定めることとしている（第 12 条）。

[13] なお、基本契約書及び請負型個別契約書には、印紙税が課せられる。

アジャイル開発基本契約書

ユーザ甲（以下、「ユーザ」という。））とベンダ乙（以下、「ベンダ」という。）は、別紙に定める全体プロジェクト（以下、「全体プロジェクト」という。）に関して、〇〇〇〇年〇月〇日に、本アジャイル開発基本契約（以下「基本契約」という。）を締結する。

本契約書では、別紙において、基本契約締結時点において可能な範囲で、全体プロジェクトの内容を記載することとしている。
具体的には、
・全体プロジェクトの目的・ビジョン
・全体予算（概算）
・予定期間
・開発を予定しているシステムの機能
・プロジェクト遂行のための人員体制
・プロジェクト実施場所・設備
などである。
別紙に記載された事項は、あくまで契約締結時点におけるユーザの希望を示すものであり、プロジェクトを開始するにあたって、ユーザ、ベンダの両当事者が、同じ目的を共有するための青写真を提供するものである。別紙記載の内容は、プロジェクトの進行に伴って変化することが想定されているため、後述のとおり、本契約においては、全体プロジェクトの内容として別紙に記載される内容は法的拘束力を持たないこととした。

第１条（目的）
基本契約は全体プロジェクトの実現のためのユーザとベンダの権利・義務関係を定めるものである。

ユーザとベンダの目的は、全体プロジェクト実現にあること、本契約はそのための権利義務を定めるものであることを宣言している。

第２条（全体プロジェクト）
ユーザとベンダは全体プロジェクトを遂行するために相互に協力するが、全体プロジェクトを完了する法的義務を負うものではない。

アジャイル開発においては、ユーザとベンダの協力が重要であることから、ユーザからベンダへの一方的な発注関係ではなく、相互協力関係にあることを宣言している。ところで、全体プロジェクトの遂行過程では、ユーザが望む機能が変化してプロジェクトを大幅に変更したり、途中終了したりする場合があるウォーターフォールモデルの問題点の一つとして、このような契約後のユーザ要望の変化を許さない非寛容さが指摘されているが、アジャイル開発においては、このような事態に対応するため、ベンダは全体プロジェクトを完

了する法的義務を負わないこととした。

第 3 条（個別契約）

1. ユーザとベンダは協議を通じて全体プロジェクトのうちの一定の機能群又は一定の期間ごとに、個別契約（以下「個別契約」という。）を締結する。各個別契約は、基本契約と一体となって、ひとつの契約（以下「本契約」という。）を構成する。
2. 個別契約はユーザとベンダが個別契約書の各項目について合意し、同書に記名押印したときに成立する。3. ユーザとベンダは全体プロジェクトで開発が予定される全ての機能について個別契約を締結する法的義務を負うものではない。

ユーザとベンダは協議（第 6 条に定める連絡協議会に限られるものではない。）を行い、ユーザが開発を望む一定の機能群のスコープが確定した時点、又はユーザがある一定期間ベンダに開発支援を委託することが確定した時点に個別契約を締結する。この個別契約は、基本契約と一体となって一つの契約を構成するものとされている。なお、全体プロジェクトには法的拘束力をもたせないようにするため、ユーザとベンダは、全体プロジェクトで開発を予定した機能について、個別契約を締結する義務を負うものではないことを、第 3 項で規定している。

第 4 条（変更管理）

1. ユーザとベンダは、アジャイル開発においては、一旦合意した内容の変更が必ず発生することに鑑み、一方当事者より個別契約の内容又は連絡協議会で合意した事項について変更の協議の要請があったときは、速やかに協議に応じなければならない。
2. 前項の変更協議は、連絡協議会において行う。
3. 変更協議においては、変更の対象、変更の可否、変更による代金・納期に対する影響等を検討し、変更を行うかについて両当事者とも誠実に協議する。
4. 変更協議の結果、個別契約の内容を変更することが合意された場合には、ユーザとベンダは、変更内容が記載された変更版個別契約書に記名押印しなければ、当該変更は有効とならない。
5. 変更協議を行っても協議が調わないまま、最初の変更協議から○日間が経過した場合は、ユーザ又はベンダは、書面によって相手方に通知することにより、当該変更協議の原因となった個別契約を、それぞれが支出した費用に従って清算することができる。
6. 個別契約の清算は基本契約の効力に影響するものではなく、また基本契約の解除原因となるものではない。

アジャイル開発では、一旦合意した事項についても、状況の変化に応じて柔軟に変更できるようにする必要がある。そのため、ユーザ及びベンダは、合意事項について変更の要請があった場合には、速やかに協議を開き、変更の可否を協議することとしている。なお、連絡協議会で合意した事項を変更することが決定した場合は、議事録にその旨を記録しておけばよいが、個別契約で合意した内容を変更する場合には、両当事者の記名押印のある変更契約書を締結しなければ、変更は有効にならないものとしている。これは、個別契約

の内容が、予算、期間、成果物等、アジャイル開発であっても安易に変更されるべきではないと考えられるためである。
なお、変更協議を行っても協議が調わない場合には、一定期間経過後、両当事者は変更協議の原因となった個別契約を清算することができるものとしている。協議がまとまらないまま、プロジェクト全体がこう着状態となる事態を回避する趣旨である。

第 5 条（協働と役割分担）

1. ユーザ及びベンダは、個別契約の履行においてはお互いに協力しなければならないこと、かかる義務は法的な義務であることを認める。
2. ユーザとベンダ双方による共同作業及び各自の分担作業は、各個別契約においてその詳細を定める。
3. ユーザ及びベンダは、共同作業及び各自の実施すべき分担作業を遅延し又は実施しない場合若しくは不完全な実施であった場合、それにより相手方に生じた損害の賠償も含め、かかる遅延又は不実施若しくは不完全な実施について相手方に対して責任を負う。

システム開発、特にアジャイル開発においては、ユーザとベンダの密接なコミュニケーションと協働関係が必須となるため、個別契約の履行における相互協力義務を、法的義務とすることを規定している。そして、ユーザ、ベンダ双方とも、個別契約で定める自らの担当作業を怠った場合には、相手方に対して責任を負うこととしている。

第 6 条（連絡協議会の設置）

1. ユーザ及びベンダは、本件事業が終了するまでの間、開発する機能の内容決定、全体プロジェクト及び機能開発の進捗状況、リスクの管理及び報告、ユーザ及びベンダ双方による共同作業並びに各自の分担作業の実施状況、問題点の協議及び解決、その他全体プロジェクトが円滑に遂行できるよう必要な事項を協議するため、連絡協議会を開催する。
2. 連絡協議会は、原則として、個別契約書で定める頻度で定期的に開催するものとし、それに加えて、ユーザ又はベンダのいずれかが必要と認める場合に随時開催する。
3. ユーザ及びベンダは、必要があれば連絡協議会を迅速に開催できるよう、体制を整えなければならない。
4. 連絡協議会には、ユーザ及びベンダ双方の責任者、主任担当者及び責任者が適当と認める者が出席する。また、ユーザ及びベンダは、連絡協議会における協議に必要となる者の出席を相手方に求めることができ、相手方は合理的な理由がある場合を除き、これに応じるものとする。
5. ベンダは、連絡協議会において、開発すべき機能の内容を決定し、全体プロジェクト及び機能開発の進捗状況を確認するとともに、遅延事項の有無、遅延事項があるときはその理由と対応策、推進体制の変更（人員の交代、増減、再委託先の変更など）の要否、セキュリティ対策の履行状況、個別契約の変更を必要とする事由の有無、個別

契約の変更を必要とする事由があるときはその内容などの事項を必要に応じて協議し、決定された事項、継続検討とされた事項並びに継続検討事項がある場合は検討スケジュール及び検討を行う当事者等を確認するものとする。

6. ユーザ及びベンダは、本件事業の遂行に関し連絡協議会で決定された事項について、本契約に反しない限り、これに従わなければならない。
7. ベンダは、連絡協議会の議事内容及び結果について、議事録を作成し、これをユーザに提出する。ユーザは、これを受領した日から〇日以内にその点検を行うこととし、当該期間内に書面又は電子メールにより具体的な理由を明示して異議を述べない場合には、ベンダが作成した議事録を承認したものとみなすものとする。
8. 前項の議事録は、少なくとも当該連絡協議会において決定された事項、継続検討とされた事項及び継続検討事項がある場合は、検討スケジュール及び検討を行う当事者の記載を含むものとする。
9. ユーザ及びベンダは、連絡協議会を開催していない場合であっても、相手方から全体プロジェクトに関する問い合わせを受けた場合には、速やかに応答するものとする。

連絡協議会においては、個別契約を締結する前の段階では、いかなる機能を開発するかを協議し、個別契約締結後は、機能完成に向けた進捗報告、リスク管理・報告等を行うものとしている。連絡協議会で合意した事項については、基本契約又は個別契約に違反しない限り、ユーザ及びベンダはこれに従わなければならない。そのため、合意の有無を証明する議事録が重要となるが、アジャイル開発においては、連絡協議会は頻繁に開催されるため、毎回の議事録を書面で取り交わし、記名・押印をするのは煩雑である。そこで、ユーザの点検期間を規定し、その期間内にユーザから異議がなければ、議事録は承認されたものとみなすこととした。
このようにみなし承認を原則とすると、議事録にかかる手間は減るが、記名押印がないために、紛争時には議事録の真正（偽造、変造がないこと）や、ベンダからユーザへの議事録提出の事実自体が問題となる恐れがある。そのため、ベンダは、議事録をユーザに提示したことを後日証明できるよう、議事録を送付した電子メールに対して受領の返信をもらう、書面で渡した場合は受領のサインをもらうといった工夫を行う必要があろう。

第 7 条（ユーザがベンダに提供する資料等及びその返還）

1. ユーザは、ベンダに対し、本件業務に必要な資料、機器、設備等（以下「資料等」という。）の開示、貸与等を行うものとする。
2. ユーザが前項に基づきベンダに提供した資料等の内容に誤りがあった場合又はユーザが提供すべき資料等の提供を遅延した場合、これらの誤り又は遅延によって生じた費用の増大、完成時期の遅延、瑕疵などの結果について、ベンダは責任を負わない。
3. ベンダは、ユーザから提供を受けた資料等を善良なる管理者の注意義務をもって管理し、双方が合意した返還日又はユーザから請求があったときに、これらを返還する。
4. 資料等の提供及び返還にかかる費用は、ユーザが負担する。

システム開発においては、ユーザからベンダへの資料、機器、設備等の提供が必要となる場合があるが、本条項はその扱いについて定めたものである。
なお、本条項以下の規定内容は、ウォーターフォールモデルに関する経済産業省「モデル取引・契約書」と共通しているため、そちらの解説も参照されたい。

第 8 条（再委託）

1. ベンダは、事前にユーザの承諾を書面で得た場合又はユーザが指定した再委託先に再委託する場合、各個別業務の一部を第三者に再委託することができるものとする。なお、ユーザが上記の承諾を拒否するには、合理的な理由を要するものとする。
2. ベンダが、前項の承諾に関して、ユーザに対して再委託開始時期の〇日前までに当該再委託先の名称及び住所等を記載した書面による再委託承諾申請を通知し、ユーザから当該通知受領後〇日以内に具体的理由を明記した書面による承諾拒否の通知がない場合、ユーザは当該再委託を承諾したものとみなす。
3. ユーザの承諾拒否により、ベンダが他の再委託先を選定することが必要になった場合は、作業期間若しくは納期又は委託料等の個別契約の内容の変更について、第 4 条（変更管理）によるものとする。
4. ベンダは当該再委託先との間で、再委託に係る業務を遂行させることについて、本契約に基づいてベンダがユーザに対して負担するのと同様の義務を、再委託先に負わせる契約を締結するものとする。
5. ベンダは、再委託先の履行についてユーザに帰責事由がある場合を除き、自ら業務を遂行した場合と同様の責任を負うものとする。但し、ユーザの指定した再委託先の履行については、ユーザに故意又は重過失がある場合を除き、責任を負わない。

本条は、再委託に関する規定であるが、アジャイルシステム開発の取引実態に適合するものとして、ウォーターフォールモデルに関する経済産業省「モデル取引・契約書＜第一版＞」第 7 条[14]の【A案】を採用している。
再委託の可否については、①再委託先の技術力についての保証がなく、また機密保持の観点からも原則禁止とし委託者の承諾を要するとすべき（原則禁止【A案】）との考えと、②再委託を原則禁止としてしまうことによって業務の遂行における柔軟性が失われ結局提供される技術の質も効率も損なわれてしまうので原則自由とすべき（原則自由【B案】）との考えの対立があり、モデル取引・契約書第一版においても、両論が併記されている。
本契約書が対象とするアジャイル開発では、ユーザとベンダの信頼関係が極めて重要であるが、この観点からすると、そもそもベンダの技術力、プロジェクトマネジメント能力に対するユーザの強い信頼が開発の核であり、再委託を自由とすることは信頼関係を崩す結果につながりかねない。ベンダがユーザとの信頼関係を重視しつつ、再委託先とともにプロジェクトを遂行するためには、ベンダが再委託先の能力を慎重に検討した上で、ユーザ

14 モデル取引・契約書第一版 59～60p。A 案：再委託先におけるユーザの事前承諾を設ける場合、B 案：再委託先の選定について原則としてベンダの裁量（但し、ユーザの中止請求が可能）とする場合。

による評価・承認を得て、再委託を行うことすべきである。

第 9 条（秘密情報の取扱い）

1. ユーザ及びベンダは、全体プロジェクト遂行のため、相手方より提供を受けた技術上又は営業上その他業務上の情報のうち、相手方が書面により秘密である旨指定して開示した情報、又は口頭により秘密である旨を示して開示した情報で開示後○日以内に書面により内容を特定した情報（以下あわせて「秘密情報」という。）を第三者に漏洩してはならない。但し、次の各号のいずれか一つに該当する情報についてはこの限りではない。また、ユーザ及びベンダは秘密情報のうち法令の定めに基づき開示すべき情報を、当該法令の定めに基づく開示先に対し開示することができるものとする。

   ①秘密保持義務を負うことなくすでに保有している情報
   ②秘密保持義務を負うことなく第三者から正当に入手した情報
   ③相手方から提供を受けた情報によらず、独自に開発した情報
   ④本契約に違反することなく、かつ、受領の前後を問わず公知となった情報
2. 秘密情報の提供を受けた当事者は、当該秘密情報の管理に必要な措置を講ずるものとする。
3. ユーザ及びベンダは、秘密情報について、本契約の目的の範囲内でのみ使用し、本契約の目的の範囲を超える複製、改変が必要なときは、事前に相手方から書面による承諾を受けるものとする。
4. ユーザ及びベンダは、秘密情報を、本契約の目的のために知る必要のある各自（本契約に基づきベンダが再委託する場合の再委託先を含む。）の役員及び従業員に限り開示するものとし、本契約に基づきユーザ及びベンダが負担する秘密保持義務と同等の義務を、秘密情報の開示を受けた当該役員及び従業員に退職後も含め課すものとする。
5. 秘密情報の提供及び返還等については、第 7 条（ユーザがベンダに提供する資料等及びその返還）に準じる。
6. 秘密情報のうち、個人情報に該当する情報については、第 10 条が本条の規定に優先して適用されるものとする。
7. 本条の規定は、本契約終了後、○年間存続する。

ソフトウェア開発においては、ユーザ、ベンダが互いに相手方の秘密情報に接することが想定されることから、本条では、それぞれの秘密保持義務を定める。

第 1 項では、秘密保持義務の対象となる情報を特定している。本項では、対象となる情報を明確にするため、相手方が書面により秘密である旨指定して開示した情報であるか、または口頭により秘密である旨通知して開示した情報は、開示後○日以内に書面により内容を特定することを必要としている。第 1 号から第 4 号は、秘密情報の例外規定である。
第 2 項は、秘密情報の提供を受けた当事者は、秘密情報の管理に必要な措置を講ずることとしている。秘密情報の秘密管理性及び非公知性を維持するためには、提供を受けた当事

者に秘密情報を適正に保護する体制の構築を義務づけておく必要がある。秘密情報の管理については、物理的、技術的、人的、組織的管理措置を実効的に構築しなければならない。
第 3 項は、秘密情報の目的外使用を禁止し、複製、改変については相手方の承諾を要件としている。
第 4 項は、システム基本契約書及び個別契約に基づき乙が再委託する場合の再委託先も含め、秘密情報の開示を受けた役員、従業員、退職者へも秘密保持義務を負わせるよう求めている。開示を受けた者が退職してしまった場合に、第三者に秘密情報が出て行くことのないよう退職者についても秘密保持義務を課すことを義務づけている。秘密情報の開示を受ける担当者等に秘密保持の誓約書を義務づけるなど、より具体的な方策を定めておくことも考えられる。退職者に対して秘密保持義務を課す場合には、一般的に秘密保持契約を締結する必要がある。特に、現職の従業者等及び退職者と秘密保持契約を締結する際には、秘密保持義務が必要性や合理性の点で公序良俗違反（民法第 90 条）とならないよう、その立場の違いに配慮しながら、両者がコンセンサスを形成できるようにすることが重要である（「営業秘密管理指針」（平成 15 年 1 月 30 日、平成 22 年 4 月 9 日改訂、経済産業省）参照）。
本条で定める秘密情報と次条で定める個人情報は、公知情報でない個人情報について適用が重複する場合もありうるので、第 6 項でその優先関係について取り決めている。
第 7 項は、秘密保持義務は通常契約期間より長期の存続が必要であるため、本契約終了後一定期間（秘密情報の性質から鑑みて合理的な期間）、存続させるものとしている。

第 10 条（個人情報の取り扱い）

1. ベンダは、個人情報の保護に関する法律（本条において、以下「法」という。）に定める個人情報のうち、全体プロジェクト遂行に際してユーザより取扱いを委託された個人データ（法第 2 条第 4 項に規定する個人データをいう。以下同じ。）及び全体プロジェクト遂行のため、ユーザ･ベンダ間で個人データと同等の安全管理措置（法第 20 条に規定する安全管理措置をいう。）を講ずることについて、別途合意した個人情報（以下あわせて「個人情報」という。）を第三者に漏洩してはならない。なお、ユーザは、個人情報をベンダに提示する際にはその旨明示するものとする。また、ユーザは、ユーザの有する個人情報をベンダに提供する場合には、個人が特定できないよう加工した上で、ベンダに提供するよう努めるものとする。
2. ベンダは、個人情報の管理に必要な措置を講ずるものとする。
3. ベンダは、個人情報について、本契約の目的の範囲内でのみ使用し、本契約の目的の範囲を超える複製、改変が必要なときは、事前にユーザから書面による承諾を受けるものとする。
4. 個人情報の提供及び返還等については、第 7 条（資料等の提供及び返還）を準用する。
5. 第 8 条第 1 項の規定にかかわらず、ベンダはユーザより委託を受けた個人情報の取扱いを再委託してはならない。但し、当該再委託につき、ユーザの事前の承諾を受けた場合はこの限りではない。

個人情報保護法第 22 条に基づいて委託者は、委託先に対する監督の責任を負うことから、ソフトウェア開発委託契約においても、委託先の監督について取り決めておく必要がある（個人データの取扱いを委託する場合に契約に盛り込むことが望まれる事項については、「個人情報の保護に関する法律についての経済産業分野を対象とするガイドライン[15]」（以下、「個人情報ガイドライン」という。）等を参照）。また、個人情報は、秘密保持義務の対象となる秘密情報とは対象、契約で定めることが望まれる事項が異なるので、個人情報保護に関する条項を秘密保持とは別途規定してある。

第 1 項は、ベンダに個人情報保護を義務づける。ユーザ保有の個人情報については、当該個人に対し責任を持っているユーザ自身がより安全な取扱いにつき配慮すべきである。例えば、テスト時に使用するデータをユーザ側がダミー化する等してベンダに渡す等の配慮を行う必要がある。
第 2 項は、ベンダに必要な安全管理措置を義務づける。
第 3 項は、ベンダに個人情報の目的外の使用を禁止し、複製、改変についてはユーザの承諾を要件としている。
第 4 項は、個人データの提供、返還・消去・廃棄に関する事項については、第 5 条（資料等の提供及び返還）を準用する。
第 5 項は、再委託がベンダの裁量で可能な場合にも、個人情報の取扱いの再委託についてはユーザの事前承諾を要するものとしている。個人情報ガイドラインに、再委託を行う際に委託者への文書による報告を契約上規定すべきとされている趣旨に対応する[16]。

個人情報をどのように取り扱うのかについては、ユーザの事業分野に関するガイドライン等を踏まえた上で、事前に具体的内容について十分協議して、委託者と受託者の責任分担を明確にしておく必要がある。

15 個人データの取扱いを委託する場合に契約書への記載が望まれる事項について、「個人情報の保護に関する法律についての経済産業分野を対象とするガイドライン」（平成 21 年 10 月、経済産業省）（以下、「個人情報ガイドライン」という。）において、委託者及び受託者の責任の明確化、個人データの安全管理に関する事項、再委託に関する事項、個人データの取扱状況に関する委託者への報告の内容及び頻度、契約内容が遵守されていることの確認、契約内容が遵守されなかった場合の措置、セキュリティ事件・事故が発生した場合の報告・連絡に関する事項が挙げられている。

16 委託者が受託者について「必要かつ適切な監督」を行っていない場合で、受託者が再委託した際に、再委託先が適切とはいえない取扱いを行ったことにより、何らかの問題が生じた場合は、元の委託者がその責めを負うことがあり得るので、再委託する場合は注意を要する。（「個人情報ガイドライン」参照）

第 11 条（報告書の著作権）

1. ベンダがユーザに対して提出する報告書に関する著作権（著作権法第 27 条及び第 28 条の権利を含む。）は、ユーザ又は第三者が従前から保有していた著作物の著作権を除き、ベンダに帰属するものとする。
2. ユーザは、前項の報告書又はその複製物を、本件システムを利用するために必要な範囲で、複製、翻案することができるものとする。

本条は、件業務に共通して問題となりうる報告書に関する著作権の帰属について規定する。ユーザがビジネスモデルの秘密の保護やシステムのノウハウ優位性を維持するために、あらゆる著作権をユーザ帰属にするというケースが見受けられるが、本来、営業上の秘密は秘密保持条項で保護するべきものであり、著作権の帰属で保護するものではないという考え方によっている。成果物の著作権については個別契約で定める。

第 12 条（損害賠償）

1. ユーザ及びベンダは、本契約の履行に関し、相手方の責めに帰すべき事由により損害を被った場合、相手方に対して、法令に基づく損害賠償を請求することができる。但し、個別契約に請求期間が定められている場合は、法令に基づく請求期間にかかわらず、個別契約に定める期間の経過後は請求を行うことができない。
2. 前項の損害賠償の累計総額は、債務不履行、法律上の瑕疵担保責任、不当利得、不法行為その他請求原因の如何にかかわらず、帰責事由の原因となった業務に係る個別契約書に定める損害賠償限度額を限度とする。
3. 前項は、損害が損害賠償義務者の故意又は重大な過失に基づくものである場合には適用しないものとする。

本条は、瑕疵担保責任、債務不履行責任、不法行為責任等に基づく損害賠償責任の制限について規定する。情報システム開発の特殊性を考慮して、損害賠償責任の範囲・金額・請求期間について、これらを制限する規定をおくべきかどうか、またその内容をどのようにすべきかについては、ユーザ・ベンダ間で対立するところであるが、本契約書では、具体的な損害賠償の上限額、損害の範囲・請求期間の制限については、個々の情報システムの特性等に応じて、個別に決定できるように記述している。

第 1 項では、損害賠償責任の成立を、帰責事由のある場合に限定している。本項は瑕疵担保責任としての損害賠償請求についても適用されるが、ソフトウェア開発に関連して生じる損害額は多額に上る恐れがあるので、無過失責任とすることはベンダに過重な負担を

課するとの考え方による[17]。なお、損害の範囲について制限を設ける場合には、通常損害のみについて責任を負い、特別事情による損害、逸失利益についての損害や間接損害を負わないとする趣旨から、直接の結果として現実に被った通常の損害に限定して損害賠償を負う旨規定することが考えられる。
また、本項では損害賠償請求を行う場合一般について請求期間を個別契約で定めることができると定めている。当該期間をどのように設定するかは、個別具体的な事情を勘案して定められるべきである。

第 2 項は、損害賠償の累積総額の上限額を設定する規定で、請求原因の構成如何に関わらず上限が設定されている。なお、解除に伴う原状回復としての委託料の返還は、損害賠償とは異なることに注意が必要である。例えばベンダ側に重大な債務不履行があり、ユーザから本件業務にかかる契約を解除され、原状回復として委託料全額を返還したとしても、委託料の返還は損害賠償の支払いではないので、損害賠償の上限を決める累計総額には加算されないことになる。すなわち、委託料 250 万円が上限となる規定があり、委託料が支払い済みである場合でベンダの債務不履行で解除となったとき、ベンダは 250 万円の委託料の返還に加え、250 万円を上限とする損害賠償を請求される可能性が出てくることになる。

第 3 項は、第 2 項の免責は、損害賠償義務者に故意重過失ある場合には適用されないことを明記する場合の規定である。損害発生の原因が故意による場合には、判例では免責・責任制限に関する条項は無効となるものと考えられているし、重過失の場合にも同様に無効とするのが、支配的な考え方になっていることから設けられた規定である。

なお、遅延損害金について本契約書では定めをおいていない。商事法定利率である年 6 分を超える割合の遅延損害金を定める場合は、個別契約書に特約として記載されたい。

第 13 条（解除）

1. ユーザ又はベンダは、相手方に次の各号のいずれかに該当する事由が生じた場合には、何らの催告なしに直ちに本契約の全部又は一部を解除することができる。

   ①重大な過失又は背信行為があった場合
   ②支払いの停止があった場合、又は仮差押、差押、競売、破産手続開始、民事再生手続開始、会社更生手続開始、特別清算開始の申立があった場合
   ③手形交換所の取引停止処分を受けた場合

[17] たとえば、EC サイトの開発でベンダ提案が原因で、システムが長期間に渡って停止を余儀なくされ、その停止期間に本来あるはずだった売上も損害賠償の対象となるなどが想定される。ベンダの無過失責任を認めると、ベンダはこうしたリスクを受託費用に載せざるを得なくなり、結果として、システム構築費用が極めて高額になる恐れがあること、アジャイル開発の成果は、ユーザとベンダの協働関係、緊密なコミュニケーションによるもの、などがあげられる。

④公租公課の滞納処分を受けた場合
⑤その他前各号に準ずるような本契約を継続し難い重大な事由が発生した場合

2. ユーザ又はベンダは、相手方が本契約のいずれかの条項に違反し、相当期間を定めてなした催告後も、相手方の債務不履行が是正されない場合は、本契約の全部又は一部を解除することができる。

3. ユーザ又はベンダは、第 1 項各号のいずれかに該当する場合又は前項に定める解除がなされた場合、相手方に対し負担する一切の金銭債務につき相手方から通知催告がなくとも当然に期限の利益を喪失し、直ちに弁済しなければならない。

本条は、本契約の解除に関する条項である。本契約は、アジャイル開発基本契約書と個別契約書から構成されるが、本条はその全部又は一部について解除する場合の要件を定めている。

第 1 項は、取引上の重大な事由について、無催告解除事由として規定する。
第 2 項は、個別の契約違反の催告解除について定める。
第 3 項は、期限の利益喪失に関する特約である。民法にも期限の利益の喪失事由（民法第 137 条）が規定されているが、その他の信用不安事由等も加えたものである。事由の軽重により、当然に期限の利益を喪失する第 1 項所定の場合と、解除により期限の利益を喪失する第 2 項の場合とに分けた。

第 14 条（権利義務譲渡の禁止）
ユーザ及びベンダは、互いに相手方の事前の書面による同意なくして、本契約上の地位を第三者に承継させ、又は本契約から生じる権利義務の全部若しくは一部を第三者に譲渡し、引き受けさせ若しくは担保に供してはならない。

本条は、契約上の地位の移転、債権譲渡、担保化の禁止に関する規定である。

第 15 条（協議）
本契約に定めのない事項又は疑義が生じた事項については、信義誠実の原則に従いユーザ及びベンダが協議し、円満な解決を図る努力をするものとする。

本条は、一般の取引基本契約に定められているのと同様の協議解決条項である。

第 16 条（和解による紛争解決・合意管轄）
1. 本契約に関し、ユーザ及びベンダに紛争が生じた場合、ユーザ及びベンダは、次項の手続をとる前に、紛争解決のため第 6 条に定める連絡協議会を開催し協議を充分に行うとともに、次項及び 3 項に定める措置をとらなければならない。

2. 前項所定の連絡協議会における協議でユーザ・ベンダ間の紛争を解決することができない場合、本条第 4 項に定める紛争解決手続をとろうとする当事者は、相手方に対し

紛争解決のための権限を有する代表者又は代理権を有する役員その他の者との間の協議を申し入れ、相手方が当該通知を受領してから○日以内に（都市名）において、本条第 4 項に定める紛争解決手続以外の裁判外紛争解決手続（以下「ADR」という。）などの利用も含め誠実に協議を行うことにより紛争解決を図るものとする。

3. 前項による協議又は ADR によって和解が成立する見込みがないことを理由に当該協議又は ADR が終了した場合、ユーザ及びベンダは、法的救済手段を講じることができる。

4. 本契約に関し、訴訟の必要が生じた場合には、○○地方裁判所を第一審の専属的合意管轄裁判所とする。

本条第 1 項、第 2 項は、本契約に関し、紛争が生じた場合、法的救済手段を講じる前段階として、当事者間でまず十分協議し、解決に尽力すべきことを規定している。

第 3 項は、当事者間による解決が不可能な場合、当事者は、法的救済手段（仲裁又は訴訟）による解決を求めることができることを規定している。

第 4 項は、裁判所に訴訟提起する場合を前提に、専属的な合意管轄（民事訴訟法第 11 条）について規定する。なお、特許権、実用新案権、回路配置利用権又はプログラムの著作物についての著作者の権利に関する訴えについては、東京高等裁判所、名古屋高等裁判所、仙台高等裁判所又は札幌高等裁判所の管轄区域内に所在する地方裁判所については東京地方裁判所の管轄、大阪高等裁判所、広島高等裁判所、福岡高等裁判所又は高松高等裁判所の管轄区域内に所在する地方裁判所については大阪地方裁判所の管轄とされる（民事訴訟法第 6 条第 1 項）が、合意管轄も認められている（民事訴訟法第 13 条第 2 項）ので、本条の適用範囲に含まれる。

ユーザ
○○株式会社

ベンダ
○○株式会社

別紙　（全体プロジェクト）

・全体プロジェクトの目的・ビジョン
・全体予算（概算）
・予定期間
・開発を予定しているシステムの機能
・プロジェクト遂行のための人員体制
・プロジェクト実施場所・設備

・具体的なシステム全体又は個別機能のイメージ（図を含む）

付録２　アジャイル開発における基本契約/個別契約モデルの個別契約（請負型）案

本契約の構成概要：
(1) 本契約は、請負型の個別契約である。
(2) 請負型は、連絡協議会での開発することが決定した一定の機能群（例えばリリース単位で一括りにした範囲）につき、ユーザがベンダにその開発を委託するものである。そのため、請負型の個別契約を行う場合は、委託対象となる機能群の概要が確定している必要がある（第 1 条、別紙の開発対象機能、具体的作業内容）。
(3) 請負型では、準委任型とは異なり、ベンダは当該機能群の完成義務を負い（第 2 条）、また瑕疵担保責任を負うことになる（第 6 条）。他方で、ユーザも自らの担当作業を遅滞なく行うとともに、ベンダの業務が円滑に行われるよう協力する義務を負っている（第 3 条）。
(4) 具体的作業内容、成果物、納期、対価、支払い等、個別契約ごとに変動する事項については、別紙で定めることとして、チェックを容易にしている（第 4 条）。
(5) 開発の過程で生じた知的財産権の帰属については、特許権は発明した側が取得することとしている（第 8 条）。他方、著作権に関しては、3 つの案を提示し、当事者が状況に応じて選択することとした（第 9 条）。

1. 個別契約の成立
ユーザは、ベンダに対し、別紙の具体的作業内容に記載された業務（但し、ユーザ担当業務を除く。）の提供を依頼し、ベンダはこれを引き受ける。なお、本個別契約は、基本契約及び他の個別契約と一体となって、ひとつの契約を構成する。

第 1 条は、請負契約成立に関する規定である。個別契約は単独で機能するものではなく、基本契約と一体となって、ひとつの契約を構成することを確認している。

2. ベンダの義務
ベンダは、別紙で定めた自らの担当作業を遅滞なく行い、別紙で定めた納期までに、同書で定めた成果物（以下「本件成果物」という。）を納入する義務を負う。

第 2 条は、ベンダの義務に関する規定である。請負型の場合、ベンダは成果物を完成させ、納期までにユーザに納入する義務を負担する。

3. ユーザの義務
ユーザは、別紙で定めた担当作業を遅滞なく行うとともに、ベンダの業務が円滑に行われるよう協力する義務を負う。

第 3 条は、ユーザの義務に関する規定である。基本契約同様、ユーザもベンダに対して、自らの担当作業を遅滞なく行う義務、ベンダに協力する義務を負担していることを確認している。これらユーザの義務は、法的義務である。

4. 成果物、納期、対価及び支払方法

成果物、納期、対価及び支払方法は、別紙において定める。

第 4 条は、成果物、納期、対価等に関する規定である。この契約モデルでは、複数の個別契約が締結されることが予定されているが、個別契約の内容確認を容易にするために、成果物、納期、対価等、各個別契約に特有の事項は、別紙に記載することとし、契約書の本文については、個別契約ごとに変更を行わない構成としている。

5. 検収基準

1）ユーザは、納品された成果物につき、別紙において定める検査期間（以下「検査期間」という。）内に検査を行う。

2）ユーザは、本件成果物が検査に合格した場合、検収書に記名押印の上、ベンダに交付する。

3）ユーザは、本件成果物が検査に合格しなかった場合には、ベンダに対し不合格とする具体的な理由を明示した書面を速やかに交付し、修正又は追完を求めることができる。不合格理由が認められるときには、ベンダは、協議の上定めた期間内に無償で修正してユーザに納品し、ユーザは速やかに再度の検査を行う。

4）ユーザが検収書を交付しない場合であっても、検査期間内にユーザが書面で具体的な理由を明示して異議を述べない場合は、成果物は、検査期間満了日に検査に合格したものとみなす。

第 5 条は、ベンダが制作した成果物に関する検収の手続を定めるものである。 第 1 項は、成果物についてユーザが検査期間内に検査し、別途合意した内容（当該成果物の仕様等）と成果物の内容との合致を点検することを規定する。
第 2 項は、検査合格をもって当該成果物の検収完了とすることを明記する。
第 3 項は、成果物が合意した内容に適合しないことが判明した場合、ベンダはこれを修正して修正版をユーザに納入することを義務付けている。
第 4 項は、みなし検査合格に関する規定を定めることにより、ユーザの都合により検収が引き延ばされることを防ぐものである。

6. 瑕疵担保責任

1）検収された成果物につき、ベンダとユーザで合意した仕様との不一致（バグを含む。以下本条において「瑕疵」という。）が発見された場合、ユーザは、ベンダに対して当該瑕疵の修正を請求することができ、ベンダは、速やかに当該瑕疵を修正する。但し、ベンダがかかる修正責任を負うのは、別紙記載の瑕疵担保期間内にユーザから請求された場合に限る。

2）前項にかかわらず、瑕疵が軽微であって、成果物の修正に過分の費用を要する場合、ベンダは前項所定の修正責任を負わない。

3）第１項の規定は、瑕疵がユーザの提供した資料等又はユーザの与えた指示によって

生じたときは適用しない。但し、ベンダがその資料等又は指示が不適当であることを知りながら告げなかったときはこの限りではない。

第 6 条は、成果物の瑕疵に関する規定である。
第 1 項では、ベンダとユーザで合意した仕様との不一致を「瑕疵」と定義している。
第 2 項では、瑕疵が軽微であっても、納入物の修正に過分の費用を要する場合に無償での修正をベンダに求めるのは酷であるので、民法第 634 条第 1 項但書に準じた規定を設けている。
第 3 項は、民法第 634 条第 1 項但書に準じ、瑕疵がユーザの指示や提供した資料等に起因する場合にはベンダは担保責任を負わないが、ベンダがかかる資料等又はユーザの指示が不適当であることを知って指摘しない場合には担保責任を免れないとする規定である。

7. 成果物の所有権移転時期

ベンダが個別契約に従いユーザに納入する成果物の所有権は、検収完了時をもって、ベンダからユーザに移転する。

第 7 条は、成果物に対する物理的な所有権の移転時期を規定している。なお、知的財産権については、別途次条で規定している。

8. 特許権の帰属

1）本件業務遂行の過程で新たに生じた発明その他の知的財産又はノウハウ等（以下、あわせて「発明等」という。）に係る特許権その他の知的財産権（特許その他の知的財産権を受ける権利を含む。但し、著作権は除く。）、ノウハウ等に関する権利（以下、特許権その他の知的財産権、ノウハウ等に関する権利を総称して「特許権等」という。）は、当該発明等を行った者が属する当事者に帰属する。

2）ベンダは、第 1 項に基づき特許権等を保有することとなる場合、ユーザに対し、ユーザが本件成果物を本件システム（全体プロジェクトにおいて開発されるシステムをいう。）において利用するのに必要な範囲について、当該特許権等の通常実施権を許諾するものとする。

第 8 条は、特許権等の帰属に関する規定である。
第 1 項は、特許法の原則である発明者主義に従い、当事者のいずれか一方の発明者が単独で発明考案した場合には、特許権等は当該当事者に帰属するものとしている。
第 2 項は、ベンダが特許権等を保有する場合においても、ユーザがシステムを使用するのに必要な範囲では、特許権等を使用する必要があるため、通常実施権を許諾するものとしている。

9. 著作権の帰属【A 案　ベンダ帰属案】

1）本件業務遂行の過程で新たに生じた成果物に係る著作権（著作権法第 27 条及び第 28 条の権利を含む。以下同じ。）は、ユーザ又は第三者が従前から保有していた著作物の著作権を除き、ベンダに帰属する。

2）ベンダは、成果物に係る著作物のうち自己が著作権を持つもの及び前項に従って自己に帰属するものについて、ユーザに対し、ユーザが本件成果物を本件システムにおいて利用できるように利用許諾し、これについて著作者人格権を行使しない。

9. 著作権の帰属【B 案　ユーザ帰属案】

1）本件業務遂行の過程で新たに生じた成果物に係る著作権（著作権法第 27 条及び第 28 条の権利を含む。 以下同じ。 ）は、ユーザ又は第三者が従前から保有していた著作物の著作権及び汎用的な利用が可能なプログラムの著作権を除き、成果物の検収完了時をもって、ベンダからユーザへ移転する。なお、かかるベンダからユーザへの著作権移転の対価は、委託料に含まれるものとする

2）ベンダは、成果物に係る著作物のうち自己が著作権を持つもの及び前項に従って自己に帰属するものについて、ユーザに対し、ユーザが本件成果物を本件システムにおいて利用できるように利用許諾し、これについて著作者人格権を行使しない。

9. 著作権の帰属【C 案　共有案】

1）本件業務遂行の過程で新たに生じた成果物に係る著作権（著作権法第 27 条及び第 28 条の権利を含む。以下同じ。）は、汎用的な利用が可能なプログラムの著作権を除き、成果物の検収完了時をもって、ユーザ及びベンダの共有（持分均等）とし、いずれの当事者も相手方への支払いの義務を負うことなく、第三者への利用許諾を含め、かかる共有著作権を行使することができるものとする。なお、ベンダからユーザへの著作権移転の対価は、委託料に含まれているものとする。また、ベンダは、ユーザのかかる利用について著作者人格権を行使しないものとする。

2）ユーザ及びベンダは、前項の共有にかかる著作権の行使について法律上必要とされる共有者の合意を、あらかじめこの契約により与えられるものとする。ただし、相手方の同意を得なければ、前項規定の著作権の共有持分を処分することはできないものとする。

第 9 条は、成果物の著作権の権利帰属及び利用について規定する。作成された成果物の権利をベンダとユーザ、いずれに帰属させるかは、当事者の合意次第であるが、本契約では、ウォーターフォール型のモデル取引・契約書第一版と同様[18]、社会的な生産効率の向上の観点などから、汎用性のあるプログラムについてはベンダに帰属させるとともに、その余の著作権に関して【ベンダ帰属案（A案）】 、【ユーザ帰属案（B案）】 、【共有案（C案）】を用意している。

【A 案】では、ソフトウェアの再利用を促進するため、原則としてベンダに著作権

[18] モデル取引・契約書第一版 95～97p 参照。

を帰属させることとしている。ベンダに著作権を帰属させたとしても、秘密保持義務を課すことで、ユーザのノウハウ流出防止を図ることが可能である。
【B 案】では、ベンダ単独で作成した著作物の著作権についてユーザに譲渡することとし、原則としてユーザに権利を帰属させる。但し、ベンダが将来のソフトウェア開発に再利用できるように、同種のプログラムに共通に利用することが可能であるプログラムに関する権利 （ベンダが従前より権利を有していたもの及び本件業務により新たに取得したものを含む。 ）及びベンダが従前から保有していたプログラムに関する権利は、ベンダに留保されるものとする。ベンダは、本契約の秘密保持義務に反しない限り、 他のソフトウェア開発においても汎用プログラム等を利用することが可能となる。なお、著作権法第 27 条（翻訳権、翻案権等）及び第 28 条(二次的著作物の利用に関する原著作者の権利)の権利については、特掲されていなければ譲渡した者に留保したものと推定される(著作権法第 61 条第 2 項)ので、これらの権利も譲渡されることを明記している。著作権の移転については、著作権の価値に見合った対価が支払われる必要がある。開発に関する委託料とは別に対価を定める方法もあり得るが、 本契約書では開発に関する委託料に含める方法をとった。
【C 案】は、 納入物のうち本件業務によって新たに生じたプログラムに関する著作物の著作権について汎用的な利用が可能なプログラムを除き、ユーザ・ベンダ間でプログラムの著作権を持分均等（２分の１ずつ）で共有する場合の規定である。本件業務遂行の過程で新たに生じた成果物についての規定であるため、ユーザ、ベンダ、第三者により従前から保有されている著作権（乙が従来から保有する汎用プログラムに関する著作権も含む。 ）については対象外となる。従って、B 案同様ベンダは、本契約の秘密保持義務に反しない限り、他のソフトウェア開発においても汎用プログラムを利用することが可能となる。
第 2 項では、共有著作権の行使は全員の合意が必要である（著作権法第 65 条第 2 項）ので、あらかじめこれに合意するものとしている。 第 2 項但書では、共有著作権については、その持分を譲渡又は質権の目的とするためには他の共有者の同意を得なければならない（著作権法第 65 条第 1 項）ので、これを確認している。
ベンダとユーザは、著作権の有効活用とユーザの競争力の保持を考慮した上で、上記のうち最も適切な規定を選択する必要がある。

アジャイル開発契約個別契約書（請負型）別紙[19]

1.　開発対象機能

【※ユーザが開発を委託する機能群のリスト、及び、各機能の具体的な内容を、個別契約締結時点で決定している範囲で記載する。】

2.　作業体制　（例）

(1)　ベンダの作業体制
・責任者氏名（スクラムマスター20）:
・スクラムマスターは以下の責任を負う。
①プロジェクト及びスプリント（イテレーション）のビジョン、目標の把握と補強
②プラクティスの保護
③プロジェクトの進捗管理、障害の排除
日次スクラムの主導とスプリントレビューの主導
・メンバ
メンバは以下の責任を負う。
①スプリント（イテレーション）バックログに従い作業を行う。
【※組織図/氏名/役割を記載。】

(2)　ユーザの作業体制
・責任者氏名（プロダクトオーナー21）:
・プロダクトオーナーは以下の責任を負う。
①プロダクトバックログの作成と優先順位付け
②プロダクトバックログから次回スプリント（イテレーション）の目標を選択し決定
③スプリントの最後に利害関係者とシステムのレビューを実施
・メンバ
メンバは以下の責任を負う。
①スプリント（イテレーション）バックログに従い作業を行う。
【※組織図/氏名/役割を記載。】

(3)　プロジェクト実施場所

(4)　連絡協議会の開催予定日

[19] 本契約書は、スクラム開発を想定して作成している。
[20] スクラムマスターはベンダとは限らない点に留意されたい。
[21] プロダクトオーナーはユーザに限られる。

3.　具体的作業内容　（例）
【具体的作業内容は計画フェーズ、準備フェーズ、開発フェーズ、リリースフェーズごとに共同担当作業、ベンダの担当作業、ユーザの担当作業として記述する。】
(1)　共同担当作業の記述例：要件定義作業、テスト
(2)　ベンダの担当作業の記述例：開発対象機能の設計、開発対象機能の開発、進捗管理
(3)　ユーザの担当作業の記述例：必要情報の提供、ベンダの求めに応じた意志決定、プロジェクト実施場所の設置及び管理、納品成果物の検査

①計画フェーズ（ビジョンの確立、期待する事象の設定、予算の確保）
具体的作業項目の例：
ビジョン、予算案、初期のプロダクトバックログ、見積、調査のための設計とプロトタイプの作成、品質保証
(1)　共同担当作業
(2)　ベンダの担当作業
(3)　ユーザの担当作業

② ②準備フェーズ（要求の洗い出し、最初のスプリント（イテレーション）に必要な優先順位付け）
具体的作業項目（例）：
計画の策定、調査のための設計とプロトタイプの作成、品質保証
(1)　共同担当作業
(2)　ベンダの担当作業
(3)　ユーザの担当作業

③開発フェーズ（30 日のスプリントを繰り返してリリース可能なシステムを実装する）
具体的作業項目（例）：
スプリント計画ミーティング、スプリントバックログの定義と見積、日次スクラムミーティングの実施、スプリントレビューの実施、品質保証
(1)　共同担当作業
(2)　ベンダの担当作業
(3)　ユーザの担当作業

④リリースフェーズ（運用環境への導入）
具体的作業項目の例：
品質保証、ドキュメントの作成、導入・運用等の教育
(1)　共同担当作業
(2)　ベンダの担当作業
(3)　ユーザの担当作業

4.　予定作業期間

〇年〇月〇日から〇年〇月〇日

5.　成果物及び納期

成果物		納期及び検査期間
機能 A	要件定義書	〇年〇月〇日（検査期間：〇日間）
	仕様書	
	テスト報告書	
	開発プログラム一式	
	・・・	
機能 B	仕様書	〇年〇月〇日（検査期間：〇日間）
	開発プログラム一式	
・・・		

6.　瑕疵担保期間

各成果物の検収完了日から起算して〇日

（※成果物ごとに瑕疵担保期間）

7.　受託費用の支払い

受託金額（税抜）：　〇〇円

支払期限：〇年〇月〇日（支払条件：開発対象機能の検収が完了したこと）

支払方法：現金・銀行口座振込み

8.　損害賠償限度額

付録３　アジャイル開発における基本契約/個別契約モデルの個別契約（準委任型）案

本契約の構成概要：
(1) 本契約は、準委任型の個別契約である。
(2) 準委任型は、ユーザがベンダに対し、ベンダが有する情報処理技術に関する業界の一般的な専門知識及びノウハウに基づく業務遂行を、一定の期間、委託するものである。
(3) 準委任型では、請負型とは異なり、ベンダは当該機能群の完成義務及び瑕疵担保責任を負わない（第 2 条）。もっとも、ベンダは業務遂行にあたり、善管注意義務（善良な管理者の注意義務）を負担し（第 2 条）、ベンダの行った業務が業界の一般的な水準を下回っていた場合には、ベンダは債務不履行責任を負う恐れがある。なお、ユーザは自らの担当作業を遅滞なく行う義務及びベンダへの協力義務を負う（第 3 条）。
(4) 委託業務の内容、対価、支払い等、個別契約ごとに変動する事項については、別紙で定めることとして、チェックを容易にしている（第 4 条）。

1. 個別契約の成立

ユーザは、ベンダに対し、別紙の具体的作業内容に記載された業務（但し、ユーザ担当作業を除く。以下「本件業務」という。）の提供を依頼し、ベンダはこれを引き受ける。なお、本個別契約は、基本契約及び他の個別契約と一体となって、ひとつの契約を構成する。

第 1 条は、準委任契約成立に関する規定である。個別契約は単独で機能するものではなく、基本契約と一体となって、ひとつの契約を構成することを確認している。

2. ベンダの義務

ベンダは、情報処理技術に関する業界の一般的な専門知識及びノウハウに基づき、善良な管理者の注意をもって、本件業務を行う義務を負う。なお、別紙に記載された開発対象機能又は予定成果物については、ユーザが開発主体となるものであり、ベンダが完成義務を負うものではない。

第 2 条は、ベンダの義務に関する規定である。準委任契約においては、ベンダは情報処理技術に関する業界の一般的な専門知識及びノウハウに基づき、善良な管理者の注意をもって、別紙で定めた自らの担当業務を行う義務（善管注意義務）を負うことを確認している。
なお、別紙には、開発対象機能や予定成果物が記載されるが、これらの開発主体、作成主体はあくまでユーザであり、ベンダはユーザによる開発を支援するという立場にあること、請負と異なりベンダが成果物を完成させる義務を負うものではないことを明確にしている。

3. ユーザの義務

ユーザは、別紙で定めた担当業務を遅滞なく行うとともに、ベンダの業務が円滑に行われるよう協力する義務を負う。

第 3 条は、ユーザの義務に関する規定である。基本契約同様、ユーザもベンダに対して、自らの担当作業を遅滞なく行う義務、ベンダに協力する義務を負担していることを確認している。これらユーザの義務は、法的義務である。

4. 対価及び支払方法

対価及び支払方法は、別紙において定める。

第 4 条は、成果物、納期、対価等に関する規定である。この契約モデルでは、複数の個別契約が締結されることが予定されているが、個別契約の内容確認を容易にするために、成果物、納期、対価等、各個別契約に特有の事項は、別紙に記載することとし、契約書の本文については、個別契約ごとに変更を行わない構成としている。

5. 業務終了の確認

1）ベンダは、別紙記載の期限までに、業務完了報告書を作成し、ユーザに提出する。

2）ユーザは、別紙記載の期間（以下「点検期間」という。）内に、前項の業務完了報告書の点検を行う。

3）ユーザは、第１項の業務報告書の内容に異議がない場合には、業務完了確認書に記名押印してベンダに交付することで、本件業務の終了を確認する。

4）ユーザが、業務完了確認書に記名押印をしない場合であっても、点検期間内に書面で具体的な理由を明示して異議を述べないときは、点検期間の満了をもって本件業務の終了を確認したものとみなす。

第 5 条は業務終了の確認についての規定である。 本条では、準委任としてベンダが善管注意義務に基づき業務を適切に行ったかどうかの確認を行う手続を定める。
第 1 項は、ベンダはユーザに対し、業務終了後所定の期間内に業務完了報告書を提出することとする。
第 2 項は、点検期間を明確にした上で、ユーザが業務完了報告書の確認を行うことを定める。
第 4 項は、ユーザが業務完了確認を怠った場合のみなし確認を定める。

アジャイル開発契約個別契約書（準委任型）別紙[22]

1.　開発対象機能

【※ユーザが開発を委託する機能群のリスト、及び、各機能の具体的な内容を、個別契約締結時点で決定している範囲で記載する。】

2.　作業体制　（例）

(1)　ベンダの作業体制

・責任者氏名（スクラムマスター等）:

・スクラムマスターは以下の責任を負う。

プロジェクト及びスプリント（イテレーション）のビジョン、目標の把握と補強

プラクティスの保護

プロジェクトの進捗管理、障害の排除

日次スクラムの主導とスプリントレビューの主導

・メンバ

メンバは以下の責任を負う。

スプリント（イテレーション）バックログに従い作業を行う。

【※組織図/氏名/役割を記載。】

(2)　ユーザの作業体制

・責任者氏名（プロダクトオーナー等）:

・プロダクトオーナーは以下の責任を負う。

プロダクトバックログの作成と優先順位付け

プロダクトバックログから次回スプリント（イテレーション）の目標を選択し決定

スプリントの最後に利害関係者とシステムのレビューを実施

・メンバ

メンバは以下の責任を負う。

スプリント（イテレーション）バックログに従い作業を行う。

【※組織図/氏名/役割を記載。】

(3)　プロジェクト実施場所

(4)　連絡協議会の開催予定日

[22] 本契約書は、スクラム開発を想定して作成している。

3.　具体的作業内容　（例）

【具体的作業内容は計画フェーズ、準備フェーズ、開発フェーズ、リリースフェーズごとに共同担当作業、ベンダの担当作業、ユーザの担当作業として記述する。】

(1)　共同担当作業の記述例：要件定義作業、テスト

(2)　ベンダの担当作業の記述例：開発対象機能の設計、開発対象機能の開発、進捗管理

(3)　ユーザの担当作業の記述例：必要情報の提供、ベンダの求めに応じた意志決定、プロジェクト実施場所の設置及び管理、・納品成果物の検査

①計画フェーズ（ビジョンの確立、期待する事象の設定、予算の確保）

具体的作業項目（例）:

ビジョン、予算案、初期のプロダクトバックログ、見積、調査のための設計とプロトタイプの作成、品質保証

(1)　共同担当作業

(2)　ベンダの担当作業

(3)　ユーザの担当作業

②準備フェーズ

（要求の洗い出し、最初のスプリント（イテレーション）に必要な優先順位付け）

具体的作業項目（例）:

計画の策定、調査のための設計とプロトタイプの作成、品質保証

(1)　共同担当作業

(2)　ベンダの担当作業

(3)　ユーザの担当作業

③開発フェーズ（30 日のスプリントを繰り返してリリース可能なシステムを実装する）

具体的作業項目（例）:

スプリント計画ミーティング、スプリントバックログの定義と見積、日次スクラムミーティングの実施、スプリントレビューの実施、品質保証

(1)　共同担当作業

(2)　ベンダの担当

(3)　ユーザの担当作業

④リリースフェーズ（運用環境への導入）

具体的作業項目（例）:

品質保証、ドキュメントの作成、導入・運用等の教育

(1)　共同担当作業

(2)　ベンダの担当作業

(3)　ユーザの担当作業

4. 作業期間

〇年〇月〇日から〇年〇月〇日

5. 完成に向けて支援を行うユーザ成果物の一覧

	予定するユーザの成果物	予定納期
機能 A	要件定義書	〇年〇月〇日
	仕様書	
	テスト報告書	
	開発プログラム一式	
	・・・	
機能 B	仕様書	〇年〇月〇日
	開発プログラム一式	
・・・		

6. 業務の完了

(1) ベンダからの業務完了報告書提出期限：〇年〇月〇日

(2) ユーザによる点検期間：業務完了報告書提出日から〇日間

7. 受託費用の支払い

受託金額（税抜）： 〇〇円

支払期限：〇年〇月〇日（支払条件：ユーザによる業務完了の確認がなされたこと）

支払方法：現金・銀行口座振込み

8. 損害賠償限度額

付録４　基本契約/個別契約モデルの契約で使用する「連絡協議会議事録（サンプル）」

提出日：〇〇〇〇年〇〇月〇〇日

本紙を含む全〇枚

第〇回　〇〇〇〇 プロジェクト連絡協議会議事録（記入例）

（委託者）株式会社〇〇商事 御中

（受託者）△△システム株式会社
〇〇事業部
作成者 〇〇〇〇 印

アジャイル開発基本契約書第 6 条 7 項に基づき、議事録を提出いたします。ご精査の上、ご承認をお願い申し上げます。

開催日時：	〇〇〇〇年〇〇月〇〇日　〇〇時〇〇分～〇〇時〇〇分
開催場所：	株式会社〇〇〇〇〇〇〇 本社会議室
出席者：	株式会社〇〇〇〇〇〇〇
	責任者〇〇、〇〇、〇〇
	〇〇〇〇〇株式会社
	責任者〇〇、〇〇、〇〇

議事：（契約書第 6 条 8 項により、決定事項、継続検討事項がある場合は検討スケジュール及び検討当事者を記載の事。）

以上、議事の内容に相違ないことを確認し、本議事録を承認いたします。

（委託者）株式会社〇〇商事　　責任者　〇〇〇〇　印

（受託者）△△システム株式会社　　責任者　〇〇〇〇　印

付録５　アジャイル開発における組合モデル契約案

本契約の前提条件： 契約当事者：対等に交渉力がありかつ技術力を有するユーザとベンダ 開発モデル：アジャイルソフトウェア開発 スクラム 対象システム：業務システム、パッケージソフトウェア等の開発 プロセス：共通フレーム 2007（SLCP-JCF2007）の適用が可能 特徴：組合型、変更管理手続き
本契約が想定するケース： この契約で想定しているケースは、ユーザとベンダとの間に既に信頼関係があり、かつ、出資額を決めるため契約締結段階でプロジェクトの全体概要、概算予算が決定している必要がある。 本契約の想定するケースは、ユーザとベンダが、パッケージソフトやクラウドシステムを共同で企画、製作し、完成した成果物を運用して収益を得て、それを分配するような場合である（このようなケースであれば、特に規模は問わない。）。長期間の継続を予定しているため、ユーザとベンダとの間に既に高度な信頼関係があり、かつ、プロジェクトの全体概要、概算予算が決定している必要がある。 本契約は、ユーザとベンダが民法上の組合[23]を構成して、各自が出資（ユーザは金銭、ベンダは労務）を行い、組合活動から生じた成果物（知的財産権を除く）をユーザに帰属させることとしている。 プロジェクトの具体的内容については、契約書に記載するのではなく、ユーザとベンダの双方による連絡協議会で決定、遂行していくことを予定している。そのため、法的拘束力は弱まるが、迅速に開発を進める点ではメリットがある。 ※ なお、本契約はあくまで試案であり、組合運営のための組織体制、資金の出資と分配、解散時の処理など、まだまだ不十分な点があり、改善の余地が大きい。また、組合における税務上、会計上の問題も別途検討する必要があることに留意されたい。

[23] 民法組合は、2 人以上で設立、法人格を有しない、組合の債務は無限責任となる、組合契約で配当や損益分配が自由に定められる、収益についてはパススルー課税（組合に課税されるのではなく、分配された損益を各組合員の所得に合算して課税）という特徴を有する。民法 667 条～民法 688 条を参考にされたい。
（組合契約）民法 第 667 条　組合契約は、各当事者が出資をして共同の事業を営むことを約することによって、その効力を生ずる。 2 出資は、労務をその目的とすることができる。
（組合財産の共有）民法第 668 条　各組合員の出資その他の組合財産は、総組合員の共有に属する。

本契約の構成概要：

(1) 本契約は、ユーザとベンダ（及び投資家）が、民法上の組合を構成して（第 1 条第 1 項）、成果物を企画・製作するプロジェクトを進めるためのものである。

(2) プロジェクトの全体に関する内容が別紙に記載されるが、当該内容は法的拘束力を持たず、プロジェクトの過程で変更されることを予定している（第 1 条第 4 項）。

(3) ユーザとベンダ（及び投資家）は、組合位に対して出資を行うが、ユーザと投資家は金銭を出資し、ベンダはプロジェクトマネジメント業務という労務を出資することとしている（第 4 条）。出資割合及び具体的な出資金額等については、第 4 条第 1 項で定めることとされている。

(4) 成果物から得られた収益については、取り決めた出資割合に従って配分されることになる（第 7 条）。

(5) ユーザとベンダは連絡協議会を構成し（第 10 条）、開発対象とするシステムの内容を検討・決定する。連絡協議会においては、各当事者が選定した運営責任者の合意により決定がなされる（第 9 条、第 10 条第 5 項）。開発対象となるシステムの内容が決定すれば、組合はベンダに対し、開発作業を委託する（第 5 条第 1 項）。

(6) ベンダによる開発の進捗管理、リスク管理等についても、連絡協議会で協議が行われる（第 10 条）。

(7) ユーザとベンダは、連絡協議会で決定された事項には（契約内容に反しない限り）従わなければならないが、一旦合意した事項について、当事者の一方から変更要請が出た場合には、連絡協議会において、変更の協議がなされる（第 11 条第 1 項、第 2 項）。変更協議が調わない場合は、紛争解決協議を行うことになる（第 11 条第 4 項）。

(8) プロジェクトの終期は、第 2 条第 1 項において定めるが、組合員全員の合意があれば、延長されるものとしている（第 2 条第 2 項）。

組合モデル契約書

【ユーザ】（以下「ユーザ」という。）、【ベンダ】（以下「ベンダ」という。）及び【投資家】（以下「投資家」という。）は、別紙に定める全体プロジェクト（以下「全体プロジェクト」という。）に関し、〇〇〇〇年〇月〇日に、本組合モデル契約（本契約）を締結する。

本契約書では、ユーザとベンダの共同により、パッケージソフトやクラウドシステム（成果物）を企画・製作し、そこから得られる収益を分配するモデルを想定している。この契約モデルにおいては、実際にプロデューサーとしてプロジェクトを進めていくユーザ及びベンダ以外にも、成果物から得られる収益を目的としてリスクマネーを投じる投資家も参加できることとしている。
具体的なプロジェクトの内容については、別紙において、以下のような事項を記載することとしている。
・全体プロジェクトの目的・ビジョン
・プロジェクト予算計画
・予定スケジュール
・開発を予定しているシステムの機能
・プロジェクト遂行のための人員体制
・プロジェクト実施場所・設備
・システム開発後の運用計画及び収益予測
別紙に記載されたこれらの事項は、契約締結時点における想定を示すものであり、プロジェクトを開始するにあたって、契約当事者が、同じ認識を共有するための青写真を提供するものである。

第 1 条（企業体）

1. ユーザ、ベンダ及び投資家（以下、総称して「組合員」という。）は、民法上の組合である全体プロジェクト推進共同企業体（以下「本企業体」という。）を組成する。
2. 本企業体は、本契約に従って、全体プロジェクトを共同で遂行することに合意し、全体プロジェクトから生じる各組合員の利益を最大化し、これを第 4 条に定める出資比率に応じて適正に分配することを目的とする。
3. 本企業体の業務執行組合員は、ユーザ及びベンダとする。
4. 各組合員は、別紙に記載された全体プロジェクトの内容が確定的なものではなく、プロジェクト進行に伴って変更される場合があることについて同意する。

本契約書では、システム開発が、ユーザとベンダの共同事業であることを、そのまま法形式に反映させ、ユーザとベンダが双方出資をして共同の事業を営むための民法上の組合（民法第 667 条以下）[24]を組成して、プロジェクトを運営するものとしている。本条第 1

[24] 民法上の組合のほか、組合員の責任が限定される有限責任事業組合（LLP）というスキームも考えられるが、本契約ではベンダが労務出資を行うことを想定しているところ、有限責任事業組合では労務出資が認められないため、ここでは民法上の組合としている。

項は、それを宣言するものである。
第 2 項は、組合の目的を定めるものである。全体プロジェクトによって得られる利益を最大化し、各組合員の出資割合に従って適正に分配することを目的として規定している。
第 3 項は、実際にプロジェクトを推進する役割を担うユーザ及びベンダを業務執行組合員とする規定である。組合契約では、組合員の業務の執行は、組合員の過半数で決するのが原則とされているが（民法第 670 条第 1 項）、業務執行組合員（民法第 670 条第 2 項にいう「業務執行者」がこれにあたる）を定めた場合には、その者だけが業務執行権限を有し、それ以外の者（本契約では投資家）は業務執行権限を持たないことになる。そして、本契約のように、業務執行組合員が複数選任された場合には、その過半数で業務執行が決定されることになる。
本契約では、ユーザ及びベンダは実際にプロジェクト運営にプロデューサーとして携わる一方、投資家は出資だけを行うことを想定しているため、業務執行権限の点でこれらを区別し、ユーザ及びベンダを業務執行組合員としている。
第 4 項は、別紙に記載された全体プロジェクトの内容が、プロジェクトの進行に伴い、変更されうるものであることを確認するものである。

第 2 条（プロジェクト期間）
1. 本契約及び本企業体の存続期間は、本契約締結日から〇年〇月〇日までとする（以下「プロジェクト期間」という。）。
2. 前項のプロジェクト期間は、本契約の組合員全員の合意により、延長されるものとする。

本条は、本契約及び組合の存続期間を定めるものである。民法第 678 条によれば、組合契約で組合の存続期間を定めなかったときは、各組合員はいつでも脱退できることとされているが、自由な脱退によりプロジェクトが継続不能となることを防ぐために存続期間を設けている。
第 2 項では、第 1 項で定めた存続期間を、当事者全員の合意をもって延長できるものとしている。

第 3 条（プロジェクト予算）
1. 全体プロジェクトの遂行の予算（以下「プロジェクト予算」という。）は、金〇円とする。
2. 全体プロジェクトを遂行するために必要な資金が、前項に定める当初のプロジェクト予算を超過することが見込まれた場合には、超過分資金の負担につき、組合員全員による協議を行う。

本条は、全体プロジェクトの予算を定めるものである。プロジェクト予算が不足する事態が生じた際には、組合員全員により超過分資金負担について協議することとしている。

第 4 条（出資）

1. 各組合員の出資の割合は、それぞれ以下のとおりとする。なお、ユーザ及び投資家は、プロジェクト予算に相当する金銭を出資し、ベンダは、全体プロジェクトにおけるプロジェクトマネジメント業務（情報処理技術に関する業界の一般的な専門知識及びノウハウに基づき、プロジェクトが円滑かつ適切に行われるよう、善良な管理者の注意をもって、プロジェクトにおける目標、時間、コスト、品質、組織、コミュニケーション、リスク、調達を管理し、最適化する業務）に必要な労務を出資する。

   ユーザ： ○%（出資の対象： 金○万円）
   ベンダ： ○%（出資の対象： プロジェクトマネジメント業務に必要な労務）
   投資家： ○%（出資の対象： 金○万円）

2. ユーザ及び投資家は、前項に規定した出資金額を、下記のスケジュールに従い、下記銀行口座（以下「本企業体口座」という。）に振り込む方法により支払うものとする

   ユーザ： ①○年○月○日までに○万円
   ②○年○月○日までに○万円
   【必要に応じて変更】
   投資家： ①○年○月○日まで
   ②○年○月○日までに○万円
   【必要に応じて変更】
   本企業体口座
   銀行名： ○○銀行○○支店
   口座種別： 普通預金口座
   口座番号： ○○○
   口座名義： 【業務執行組合員の肩書き付きの、ユーザ（またはベンダ）名義口座】

本条は、組合への出資について定めるものである。
第 1 項では、各組合員が行う出資の割合を定めるものである。出資の内容としては、ユーザ及び投資家は金銭、ベンダは善管注意義務を伴う労務（プロジェクト・マネジメント業務）を出資することとしている。出資割合を決するにあたっては、ベンダが出資する労務の価値を、プロジェクトの内容に合わせて評価することになる。
なお、民法第 674 条第によれば、組合の損益分配の割合（利益のみならず損失も含む。）は、特約なき限り、各組合員の出資割合に応じてなされることになる。必要があれば、特約を設けて、損益分配の割合を修正することができる。
第 2 項では、金銭を出資するユーザ及び投資家の、支払期限及び支払い方法を定めるものである。組合には法人格がないため、組合名義での口座開設は認められない。そのため、業務執行組合員の肩書きをつけたユーザ（またはベンダ）名義口座を開設することになろう（組合の業務執行組合員は、ユーザ及びベンダであるから、そのいずれかが口座名義人となるのが適当であるが、ベンダは組合から請負契約を受注して支払いを受ける立場にあるため、口座についてはユーザを名義人とするのが望ましいと思われる。）。

第 5 条（システム開発作業の委託）

1. 全体プロジェクトの過程で必要となるシステム開発作業については、本企業体は、第 10 条に定める連絡協議会において開発対象と決定した機能群【(リリース単位)】ごとに、ベンダに対して当該作業を委託するものとする。
2. 対価を含めた前項の委託の内容については、連絡協議会で協議する。

本条は、全体プロジェクト遂行の過程で必要となるシステム開発作業については、本企業体とベンダが請負契約を締結する旨を規定する。ベンダには、成果物の運用段階で高い収益を得ることを目標として、限られた予算で高品質な成果物を作るインセンティブが生じることが期待されている（もっとも、開発を受注するベンダは、発注側となる組合の業務執行組合員でもあるため、利益相反、モラルハザードの危険もあることに留意する必要がある。）。
なお、ベンダとの間の委託契約については、基本契約/個別契約モデルの「アジャイル開発契約個別契約書（請負型）」を修正して使用することができる。請負の対象とする開発対象機能の範囲は、連絡協議会で自由に決定しうるが、ここでは開発モデルに応じて「リリース単位」としている。

第 6 条（権利の帰属）

1. 本企業体による全体プロジェクト遂行の過程において、本企業体の組合財産となった権利（所有権並びに特許権、著作権及びその他の知的財産権、その他一切の権利を含む。）は、組合員の共有とし、各組合員の持分は出資比率に応じるものとする。
2. 各組合員は、各自の持分につき、他の組合員全員による同意なく、譲渡、担保設定その他一切の処分を行ってはならない。

本条は、組合が取得した権利の帰属について定めるものである。民法第 668 条によれば、組合の財産は、組合員全員の共有に属するものとされているが、組合における「共有」は、通常の共有と異なり、各組合員による持分処分や分割請求が制限されるなど（民法第 676 条）、団体的拘束を受ける「合有」と解されている。
第 2 項は、各組合員の持分につき、他の組合員全員による同意がなければ処分できないものとしている。これは上述の民法第 676 条に基づく制限を確認したものである。
なお、本企業体がベンダに開発を委託する際の契約において、ベンダに帰属することとされた知的財産権は、組合財産とならないため、共有にはならない。

第 7 条（損益分配）

1. 全体プロジェクトにおいて生成された成果物（以下「プロジェクト成果物」という。）により本企業体が得た収入から、本企業体が認めた経費を控除した残額を、本企業体収入とする。
2. 本企業体口座の名義人となっている業務執行組合員は、本企業体収入を、暦年四半期毎に計算し、各四半期末日（毎年 3 月、6 月、9 月及び 12 月の末日）から○日以内に、

各組合員に対し、出資比率に応じて分配する。

3. 全体プロジェクトにおいて生成された成果物の具体的運用については、連絡協議会において協議する。

本条は、組合からの損益分配について定める。本契約では、プロジェクトの過程で生成された成果物を運用して収益を得ることが想定されているが、本企業体が得た収益全体から、必要な経費を控除した残額が各組合員に対し分配されることになる（赤字の場合も同様。）。
なお、第 3 項では、具体的なプロジェクト成果物の運用については、連絡協議会で定めることとしている。

第 8 条（会計規則、会計帳簿）

1. 組合員は本企業体の会計規則を別途合意の上定め、本企業体の会計処理に適用するものとする。
2. ユーザ【(又はベンダ)】は、本企業体に関連する帳簿、会計記録、本企業体に関連する契約書などの文書の原本又は写しを、ユーザ【(又はベンダ)】の主たる営業所に保存し据え置くものとする。
3. ユーザ【(又はベンダ)】以外の各組合員は、前項の帳簿、会計記録、その他の書面・文書等を、閲覧、謄写その他監査する権利を有する。

本条は、組合に関する会計規則と会計帳簿その他の資料の保存及びその監査について規定するものである。組合の会計処理は、ユーザもしくはベンダのいずれかの業務執行組合員に委ねることから、別途、組合会計規則を定めておき、その規則に基づき処理をする必要がある。
また、組合の業務執行組合員は、ユーザ及びベンダであるから、そのいずれかが会計帳簿類を保管するのが適当であるが、ベンダは組合から請負契約を受注して支払いを受ける立場にあるため、会計帳簿類についてはユーザが管理をするのが望ましいと思われる。
会計帳簿類については、投資家も含め、各組合員が内容を監査する権利を有する。

第 9 条（責任者の選定）

1. ユーザとベンダは、それぞれ本企業体の運営に関する責任者（以下「運営責任者」という。）を 1 名選定する。
2. ユーザとベンダは、前項により選定した運営責任者に事故があった場合に、当該運営責任者を代行する者を選定することができる。

本条は、ユーザ、ベンダのそれぞれの責任者の選定について定める。選定された運営責任者は、連絡協議会において、各当事者を代表して組合の意思決定にかかわることになる。

第 10 条（連絡協議会の設置）

1. ユーザとベンダは、プロジェクト期間中、全体プロジェクトの内容決定及び変更、プロジェクト予算の使途に関する事項、開発する機能の内容決定、全体プロジェクト及び個別機能開発の進捗状況、リスクの管理及び報告、ユーザ及びベンダ双方による共同作業並びに各自の分担作業の実施状況、問題点の協議及び解決、プロジェクト成果物の運用に関する事項その他全体プロジェクトが円滑に遂行できるよう必要な事項並びに本企業体の運営に関する事項を協議するため、連絡協議会を開催する。
2. 連絡協議会は、原則として、別途定める頻度で定期的に開催するものとし、それに加えて、ユーザ又はベンダのいずれかが必要と認める場合に随時開催する。
3. ユーザ及びベンダは、必要があれば連絡協議会を迅速に開催できるよう、体制を整えなければならない。
4. 連絡協議会には、ユーザ及びベンダ双方の運営責任者、主任担当者及び運営責任者が適当と認める者が出席する。また、ユーザ及びベンダは、連絡協議会における協議に必要となる者の出席を相手方に求めることができ、相手方は合理的な理由がある場合を除き、これに応じるものとする。
5. 連絡協議会においては、ユーザとベンダ双方の運営責任者の合意をもって、協議内容を決定する。
6. ベンダは、連絡協議会において、全体プロジェクト及び個別機能開発の進捗状況を確認するとともに、遅延事項の有無、遅延事項があるときはその理由と対応策、推進体制の変更（人員の交代、増減、再委託先の変更など）の要否、セキュリティ対策の履行状況などの事項を必要に応じて協議し、決定された事項、継続検討とされた事項並びに継続検討事項がある場合は検討スケジュール及び検討を行う当事者等を確認するものとする。
7. 組合員は、全体プロジェクトの遂行に関し連絡協議会で決定された事項について、本契約に反しない限り、これに従わなければならない。
8. ベンダは、連絡協議会の議事内容及び結果について、議事録を作成し、これをユーザに提出する。ユーザは、これを受領した日から〇日以内にその点検を行うこととし、当該期間内に書面又は電子メールにより具体的な理由を明示して異議を述べない場合には、ベンダが作成した議事録を承認したものとみなすものとする。
9. 前項の議事録は、少なくとも当該連絡協議会において決定された事項、継続検討とされた事項及び継続検討事項がある場合は、検討スケジュール及び検討を行う当事者の記載を含むものとする。
10. ユーザ及びベンダは、連絡協議会を開催していない場合であっても、他の組合員から全体プロジェクトに関する問い合わせを受けた場合には、速やかに応答するものとする。

本条に定める連絡協議会は、組合の意思決定機関である。連絡業議会においては、全体プロジェクトの内容決定及び変更、プロジェクト予算に関する事項から、開発する機能の内容決定、全体プロジェクト及び個別機能開発の進捗状況、リスクの管理及び報告、ユーザ

及びベンダ双方による共同作業並びに各自の分担作業の実施状況、問題点の協議及び解決、プロジェクト成果物の運用に関する事項まで、プロジェクトに関する事項の決定・変更と進捗管理について、幅広く協議することとしている。また、組合の運営に関する事項についても協議の対象となっている。
連絡協議会における決定は、ユーザ、ベンダ双方の運営責任者が合意したときになされるものとして、運営責任者が各当事者を代表して意思表示を行うこととしている。
連絡協議会で合意した事項については、本契約に違反しない限り、ユーザ及びベンダはこれに従わなければならない。そのため、合意の有無を証明する議事録が重要となるが、アジャイル開発においては、連絡協議会は頻繁に開催されるため、毎回の議事録を書面で取り交わし、記名・押印をするのは煩雑である。そこで、ユーザの点検期間を規定し、その期間内にユーザから異議がなければ、議事録は承認されたものとみなすこととした。
このようにみなし承認を原則とすると、議事録にかかる手間は減るが、記名押印がないために、紛争時には議事録の真正（偽造、変造がないこと）や、ベンダからユーザへの議事録提出の事実自体が問題となる恐れがある。そのため、ベンダは、議事録をユーザに提示したことを後日証明できるよう、議事録を送付した電子メールに対して受領の返信をもらう、書面で渡した場合は受領のサインをもらうといった工夫を行う必要があろう。

第 11 条（変更管理）

1. ユーザ及びベンダは、アジャイル開発においては、一旦合意した内容の変更が必ず発生することに鑑み、一方当事者より連絡協議会で合意した事項について変更の協議の要請があったときは、速やかに協議（以下「変更協議」という。）に応じなければならない。
2. 変更協議は、連絡協議会において行う。
3. 変更協議においては、変更の対象、変更の可否、変更によるプロジェクト予算及びスケジュールに対する影響等を検討し、変更を行うかについて両当事者とも誠実に協議する。
4. 変更協議を行っても協議が調わない場合、ユーザ又はベンダは、相手方に対し紛争解決のための権限を有する代表者又は代理権を有する役員その他の者との間の協議を申し入れ、相手方が当該通知を受領してから〇日以内に（都市名）において、誠実に協議（紛争解決協議）を行うことにより紛争解決を図るものとする。
5. 前項の紛争解決協議を行っても協議が調わない場合は、ユーザ又はベンダは、本企業体を解散させることができる。

アジャイル開発では、一旦合意した事項についても、状況の変化に応じて柔軟に変更できるようにする必要がある。そのため、ユーザ及びベンダは、合意事項について変更の要請があった場合には、速やかに協議を開き、変更の可否を協議することとしている。
変更協議を行っても協議が調わない場合には、紛争解決権限を有する役員等との間での紛争解決協議を行い、それでも解決ができない場合は、ユーザ又はベンダは、本企業対を清算することができるものとしている。協議がまとまらない場合に、プロジェクト全体がこう着状態となる事態を回避する趣旨である。

第 12 条（ユーザがベンダに提供する資料等及びその返還）

1. ユーザは、ベンダに対し、全体プロジェクトの遂行に必要な資料、機器、設備等（以下「資料等」という。）の開示、貸与等を行うものとする。
2. ユーザが前項に基づきベンダに提供した資料等の内容に誤りがあった場合又はユーザが提供すべき資料等の提供を遅延した場合、これらの誤り又は遅延によって生じた費用の増大、完成時期の遅延、瑕疵などの結果について、ベンダは責任を負わない。
3. ベンダは、ユーザから提供を受けた資料等を善良なる管理者の注意義務をもって管理し、双方が合意した返還日又はユーザから請求があったときに、これらを返還する。
4. 資料等の提供及び返還にかかる費用は、ユーザが負担する。

システム開発においては、ユーザからベンダへの資料、機器、設備等の提供が必要となる場合があるが、本条項はその扱いについて定めたものである。
なお、本条項以下の規定内容及び解説は、ウォーターフォールモデルに関する経済産業省「モデル取引・契約書」をベースとしている。

第 13 条（秘密情報の取扱い）

1. 各組合員は、全体プロジェクト遂行のため、他の組合員（以下「情報提供者」という。）より提供を受けた技術上又は営業上その他業務上の情報のうち、情報提供者が書面により秘密である旨指定して開示した情報、又は口頭により秘密である旨を示して開示した情報で開示後〇日以内に書面により内容を特定した情報（以下あわせて「秘密情報」という。）を第三者に漏洩してはならない。但し、次の各号のいずれか一つに該当する情報についてはこの限りではない。また、各組合員は秘密情報のうち法令の定めに基づき開示すべき情報を、当該法令の定めに基づく開示先に対し開示することができるものとする。
   ①秘密保持義務を負うことなくすでに保有している情報
   ②秘密保持義務を負うことなく第三者から正当に入手した情報
   ③情報提供者から提供を受けた情報によらず、独自に開発した情報
   ④本契約に違反することなく、かつ、受領の前後を問わず公知となった情報
2. 秘密情報の提供を受けた当事者は、当該秘密情報の管理に必要な措置を講ずるものとする。
3. 各組合員は、秘密情報について、本契約の目的の範囲内でのみ使用し、本契約の目的の範囲を超える複製、改変が必要なときは、事前に情報提供者から書面による承諾を受けるものとする。
4. 各組合員は、秘密情報を、本契約の目的のために知る必要のある各自の役員及び従業員に限り開示するものとし、本契約に基づき各組合員が負担する秘密保持義務と同等の義務を、秘密情報の開示を受けた当該役員及び従業員に退職後も含め課すものとする。
5. 秘密情報の提供及び返還等については、第 12 条（ユーザがベンダに提供する資料等及びその返還）に準じる。
6. 秘密情報のうち、個人情報に該当する情報については、第 14 条が本条の規定に優先

して適用されるものとする。

7. 本条の規定は、本契約終了後、〇年間存続する。

本条は、各組合員の秘密保持義務を定める。

第 1 項では、秘密保持義務の対象となる情報を特定している。本項では、対象となる情報を明確にするため、相手方が書面により秘密である旨指定して開示した情報であるか、または口頭により秘密である旨通知して開示した情報は、開示後〇日以内に書面により内容を特定することを必要としている。第 1 号から第 4 号は、秘密情報の例外規定である。

第 2 項は、秘密情報の提供を受けた当事者は、秘密情報の管理に必要な措置を講ずることとしている。秘密情報の秘密管理性及び非公知性を維持するためには、提供を受けた当事者に秘密情報を適正に保護する体制の構築を義務づけておく必要がある。秘密情報の管理については、物理的、技術的、人的、組織的管理措置を実効的に構築しなければならない。

第 3 項は、秘密情報の目的外使用を禁止し、複製、改変については相手方の承諾を要件としている。

第 4 項は、秘密情報の開示を受けた役員、従業員、退職者へも秘密保持義務を負わせるよう求めている。開示を受けた者が退職してしまった場合に、第三者に秘密情報が出て行くことのないよう退職者についても秘密保持義務を課すことを義務づけている。秘密情報の開示を受ける担当者等に秘密保持の誓約書を義務づけるなど、より具体的な方策を定めておくことも考えられる。退職者に対して秘密保持義務を課す場合には、一般的に秘密保持契約を締結する必要がある。特に、現職の従業者等及び退職者と秘密保持契約を締結する際には、秘密保持義務が必要性や合理性の点で公序良俗違反（民法第 90 条）とならないよう、その立場の違いに配慮しながら、両者がコンセンサスを形成できるようにすることが重要である（「営業秘密管理指針」（平成 15 年 1 月 30 日、平成 22 年 4 月 9 日改訂、経済産業省）参照）。

本条で定める秘密情報と次条で定める個人情報は、公知情報でない個人情報について適用が重複する場合もありうるので、第 6 項でその優先関係について取り決めている。

第 7 項は、秘密保持義務は通常契約期間より長期の存続が必要であるため、本契約終了後一定期間（秘密情報の性質から鑑みて合理的な期間）、存続させるものとしている。

第 14 条（個人情報）

1. ベンダは、個人情報の保護に関する法律（本条において、以下「法」という。）に定める個人情報のうち、全体プロジェクト遂行に際してユーザより取扱いを委託された個人データ（法第 2 条第 4 項に規定する個人データをいう。以下同じ。）及び本件プロジェクト遂行のため、ユーザ･ベンダ間で個人データと同等の安全管理措置（法第 20 条に規定する安全管理措置をいう。）を講ずることについて別途合意した個人情報（以下あわせて「個人情報」という。）を第三者に漏洩してはならない。なお、ユーザは、個人情報をベンダに提示する際にはその旨明示するものとする。また、ユーザは、ユーザの有する個人情報をベンダに提供する場合には、個人が特定できないよう加工した上で、ベンダに提供するよう努めるものとする。

2. ベンダは、個人情報の管理に必要な措置を講ずるものとする。

3. ベンダは、個人情報について、本契約の目的の範囲内でのみ使用し、本契約の目的の範囲を超える複製、改変が必要なときは、事前にユーザから書面による承諾を受けるものとする。

4. 個人情報の提供及び返還等については、第 13 条（資料等の提供及び返還）を準用する。

本条は、ベンダの個人情報保護義務について規定するものである。プロジェクト遂行の過程で、ユーザからベンダに対して、個人情報が提供される場合がありうるが、個人情報は、秘密保持義務の対象となる秘密情報とは対象、契約で定めることが望まれる事項が異なるので、個人情報保護に関する条項を秘密保持とは別途規定してある。個人データの取扱いを委託する場合に契約に盛り込むことが望まれる事項については、「個人情報の保護に関する法律についての経済産業分野を対象とするガイドライン[25]」（以下、「個人情報ガイドライン」という。）等を参照。

第 1 項は、ベンダに個人情報保護を義務づける。ユーザ保有の個人情報については、当該個人に対し責任を持っているユーザ自身がより安全な取扱いにつき配慮すべきである。例えば、テスト時に使用するデータをユーザ側がダミー化する等してベンダに渡す等の配慮を行う必要がある。
第 2 項は、ベンダに必要な安全管理措置を義務づける。
第 3 項は、ベンダに個人情報の目的外の使用を禁止し、複製、改変についてはユーザの承諾を要件としている。
第 4 項は、個人データの提供、返還・消去・廃棄に関する事項については、第 5 条（資料等の提供及び返還）を準用する。

第 15 条（損害賠償）

1. 各組合員は、本契約の履行に関し、他の組合員の責めに帰すべき事由により損害を被った場合、損害を与えた組合員に対して、法令に基づく損害賠償を請求することができる。

2. 前項の損害賠償の累計総額は、債務不履行、法律上の瑕疵担保責任、不当利得、不法行為その他請求原因の如何にかかわらず、損害賠償を請求する組合員が本企業体に対して出資した金額（労務出資の場合は、出資割合をもとに金額換算する）を限度とする。

[25] 個人データの取扱いを委託する場合に契約書への記載が望まれる事項について、「個人情報の保護に関する法律についての経済産業分野を対象とするガイドライン」（平成 21 年 10 月、経済産業省）（以下、「個人情報ガイドライン」という。）において、委託者及び受託者の責任の明確化、個人データの安全管理に関する事項、再委託に関する事項、個人データの取扱状況に関する委託者への報告の内容及び頻度、契約内容が遵守されていることの確認、契約内容が遵守されなかった場合の措置、セキュリティ事件・事故が発生した場合の報告・連絡に関する事項が挙げられている。

3. 前項は、損害が損害賠償義務者の故意又は重大な過失に基づくものである場合には適用しないものとする。

本条は、瑕疵担保責任、債務不履行責任、不法行為責任等に基づく損害賠償責任の制限について規定する。

第 1 項では、損害賠償責任の成立を、帰責事由のある場合に限定している。本項は瑕疵担保責任としての損害賠償請求についても適用されるが、ソフトウェア開発に関連して生じる損害額は多額に上る恐れがあるので、無過失責任とすることはベンダに過重な負担を課するとの考え方による[26]。

第 2 項は、損害賠償の累積総額の上限額を設定する規定で、請求原因の構成如何に関わらず上限が設定されている。第 3 項は、第 2 項の免責は、損害賠償義務者に故意重過失ある場合には適用されないことを明記する場合の規定である。損害発生の原因が故意による場合には、判例では免責・責任制限に関する条項は無効となるものと考えられているし、重過失の場合にも同様に無効とするのが、支配的な考え方になっていることから設けられた規定である。

なお、遅延損害金について本契約書では定めをおいていない。商事法定利率である年 6 分を超える割合の遅延損害金を定める場合は、契約書に特約として記載されたい。

[26] たとえば、EC サイトの開発でベンダ提案が原因で、システムが長期間に渡って停止を余儀なくされ、その停止期間に本来あるはずだった売上も損害賠償の対象となるなどが想定される。ベンダの無過失責任を認めると、ベンダはこうしたリスクを受託費用に載せざるを得なくなり、結果として、システム構築費用が極めて高額になる恐れがあること、アジャイル開発の成果は、ユーザとベンダの協働関係、緊密なコミュニケーションによるもの、などがあげられる。

第 16 条（除名による脱退）

1. 組合員のうちいずれかが、次の各号の一に該当した場合には、該当者以外の組合員は、何らの催告なしに該当者を直ちに本企業体から脱退させることができる。
   ①重大な過失又は背信行為があった場合
   ②支払いの停止があった場合、又は仮差押、差押、競売、破産手続開始、民事再生手続開始、会社更生手続開始、特別清算開始の申立があった場合
   ③手形交換所の取引停止処分を受けた場合
   ④公租公課の滞納処分を受けた場合
   ⑤その他前各号に準ずるような本契約を継続し難い重大な事由が発生した場合
2. 各組合員は、組合員のうちいずれかが本契約のいずれかの条項に違反し、相当期間を定めてなした催告後も、債務不履行が是正されない場合にも、当該組合員を本企業体から脱退させることができる。
3. 本企業体は、本条第 1 項及び第 2 項に基づき本企業体から脱退した者に対して、持分の払い戻しは行わないものとする。
4. 本条第 1 項及び第 2 項に基づき、本企業体から脱退した者は、他の組合員に対して負担する一切の金銭債務につき当然に期限の利益を喪失し、相殺を主張することなく、直ちに弁済しなければならない。
5. 本条第 1 項及び第 2 項に基づき、本企業体から組合員が脱退した場合には、本契約に基づき当該脱退者に帰属していた全ての権利（著作権法第 27 条及び第 28 条の規定を含む。）は、その他の組合員に、脱退者の出資割合を控除して再計算した各自の出資割合に従って、当然に無償で移転するものとする。

本条は、問題のある組合員を脱退させることができる場合を規定したものである。民法上、特定の組合員の組合員たる資格を奪うことを「除名」と呼び、民法第 680 条によれば、組合員の除名は、正当な事由がある場合に限り、他の組合員の一致によってすることができるとされている。しかし、この条文は特約で変更が可能な規定（任意規定）と解されており、本条はその特約を定めるものである。

第 1 項では、重大な過失又は背信行為や、支払い停止となった組合員については、何ら催告を行うことなく、一人の組合員の意思で、脱退させることができるものとしている。

第 2 項は、本契約に違反し、相当期間内に違反状態を是正するように求める通知（催告）を受け取っても、是正されない場合には、やはり他の（一人の）組合員の意思で脱退させることができるものとしている。

第 3 項は、前 2 項に基づき脱退させられた者に対しては、持分の払い戻しは行わないものと規定している。民法第 681 条によれば、脱退した組合員については、持分の払い戻しがなされるのが原則であるが、払い戻しによって組合が存続できなくなることを防ぐため、払い戻しは行わないこととしている。

第 4 項は、期限の利益喪失に関する特約であり、他の組合員保護のため、脱退した組合員は、支払期限を待たずに直ちに債務を弁済しなければなわないものとしている。

第 5 項は、脱退した組合員の有していた権利は、他の組合員に対し、その出資割合に従って当然に移転するものとしている。

第 17 条（権利義務譲渡の禁止）
組合員は、他の組合員全員の事前の書面による同意なくして、本契約上の地位を第三者に承継させ、又は本契約から生じる権利義務の全部若しくは一部を第三者に譲渡し、引き受けさせ若しくは担保に供してはならない。

本条は、契約上の地位の移転、債権譲渡、担保化の禁止に関する規定である。

第 18 条（協議）
本契約に定めのない事項又は疑義が生じた事項については、信義誠実の原則に従い、各組合員が協議し、円満な解決を図る努力をするものとする。

本条は、一般の取引基本契約に定められているのと同様の協議解決条項である。

第 19 条（和解による紛争解決・合意管轄）
1. 本契約に関し、組合員間に紛争が生じた場合、紛争当事者となった組合員は、次項の手続をとる前に、紛争解決のため連絡協議会（ただし、投資家が紛争当事者である場合は、投資家も責任者を決定した上、連絡協議会に参加する。）を開催し協議を充分に行うとともに、次項及び 3 項に定める措置をとらなければならない。
2. 前項所定の協議会における協議で組合員間の紛争を解決することができない場合、本条第 4 項に定める紛争解決手続をとろうとする当事者は、相手方に対し紛争解決のための権限を有する代表者又は代理権を有する役員その他の者との間の協議を申し入れ、相手方が当該通知を受領してから〇日以内に（都市名）において、本条第 4 項に定める紛争解決手続以外の裁判外紛争解決手続（以下「ADR」という。）などの利用も含め誠実に協議を行うことにより紛争解決を図るものとする。
3. 前項による協議又は ADR によって和解が成立する見込みがないことを理由に当該協議又は ADR が終了した場合、紛争当事者となった組合員は、法的救済手段を講じることができる。
4. 本契約に関し、訴訟の必要が生じた場合には、〇〇地方裁判所を第一審の専属的合意管轄裁判所とする。

本条第 1 項、第 2 項は、本契約に関し、紛争が生じた場合、法的救済手段を講じる前段階として、当事者間でまず十分協議し、解決に尽力すべきことを規定している。

第 3 項は、当事者間による解決が不可能な場合、当事者は、法的救済手段（仲裁又は訴訟）による解決を求めることができることを規定している。

第 4 項は、裁判所に訴訟提起する場合を前提に、専属的な合意管轄（民事訴訟法第 11 条）について規定する。なお、特許権、実用新案権、回路配置利用権又はプログラムの著作物についての著作者の権利に関する訴えについては、東京高等裁判所、名古屋高等裁判所、仙台高等裁判所又は札幌高等裁判所の管轄区域内に所在する地方裁判所については東京地方裁判所の管轄、大阪高等裁判所、広島高等裁判所、福岡高等裁判所又は高松高等裁判所の管轄区域内に所在する地方裁判所については大阪地方裁判所の管轄とされる（民事訴訟法第 6 条第 1 項）が、合意管轄も認められている（民事訴訟法第 13 条第 2 項）ので、本条の適用範囲に含まれる。

ユーザ
○○株式会社

ベンダ
○○株式会社

投資家
○○株式会社

別紙　（全体プロジェクト）

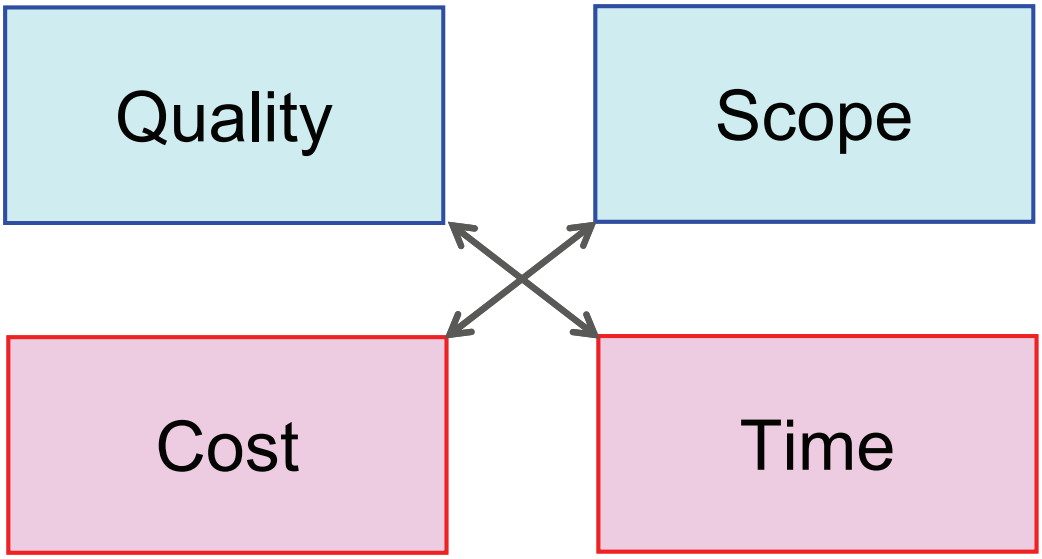
・全体プロジェクトの目的・ビジョン ・プロジェクト予算計画 ・予定スケジュール ・開発を予定しているシステムの機能 ・プロジェクト遂行のための人員体制 ・プロジェクト実施場所・設備 ・システム開発後の運用計画及び収益予測 ・具体的なシステム全体又は個別機能のイメージ（図を含む） など

付録６　次のアジャイルソフトウェアプロジェクトのための 10 の契約

米国のスクラムトレーナの Peter Stevens がスクラムによる、アジャイル開発プロジェクトで利用することを前提とした 10 種類の契約形態[2]の比較について、解説する[3]。

（1）スプリント契約（Sprint Contract）

出典資料における「スプリント契約」は、実際の商取引で利用される契約ではなく、スクラムにおいて、プロダクトオーナーと開発チームの間でスプリントごとに交わされる合意（スプリント計画）のことである。



図表 4-15　スプリント契約　（Sprint Contract）

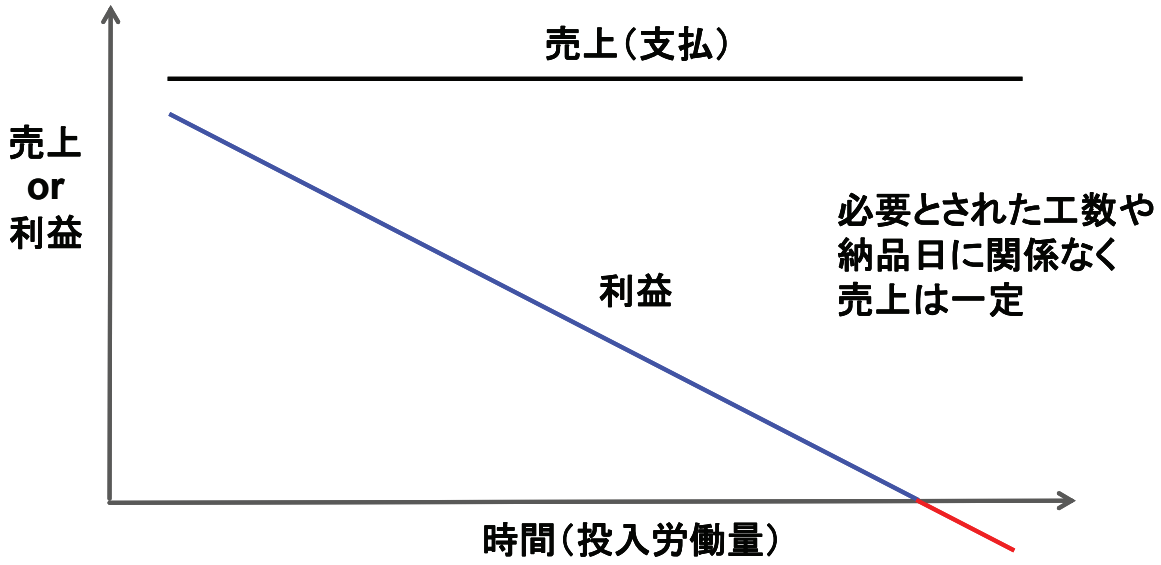
スコープ（Scope）は成果物に関する要求事項であり、品質（Quality）は成果物の品質を、期間（Time）とコスト（Cost）は開発に要する時間とコストを意味する。
スクラムでは、各スプリントに入る前に、そのスプリントで取り組むバックログと期間、要員について合意するが、これはスコープと期間、コストについて両者が合意する一種の契約である。

一般的に、1 回のスプリントは 30 日間であり、開発チームの人数は最初に決まっているので、それに合わせてスプリントバックログを調整することになる。品質要件は、無論、スプリント終了時点でスプリント終了レビューに合格することである。
このスプリント契約を実際の契約に応用することは不可能ではない。たとえば、プロジェクト全体の基本契約で、開発すべきソフトウェアの仕様と支払い金額、開発期間、検収方法については個別契約書で定めると規定しておき、スプリントごとに個別契約をすればよい。

つまり、プロジェクトの最初に基本契約を締結し、その後、スプリント毎に（つまりほぼ毎月）個別契約を結ぶという方法である。個別契約のひな形を作っておけば、個別契約書自体の作成にはそれほど時間を必要としないだろうが、毎月、契約手続きを行うのは煩雑かもしれない。

（2）定額請負型（Fixed Price & Fixed Scope）

この契約は、価格とスコープを最初に固定するので、日本における一般的な請負契約に相当する。



図表 4-16　定額請負型（Fixed Price & Fixed Scope）

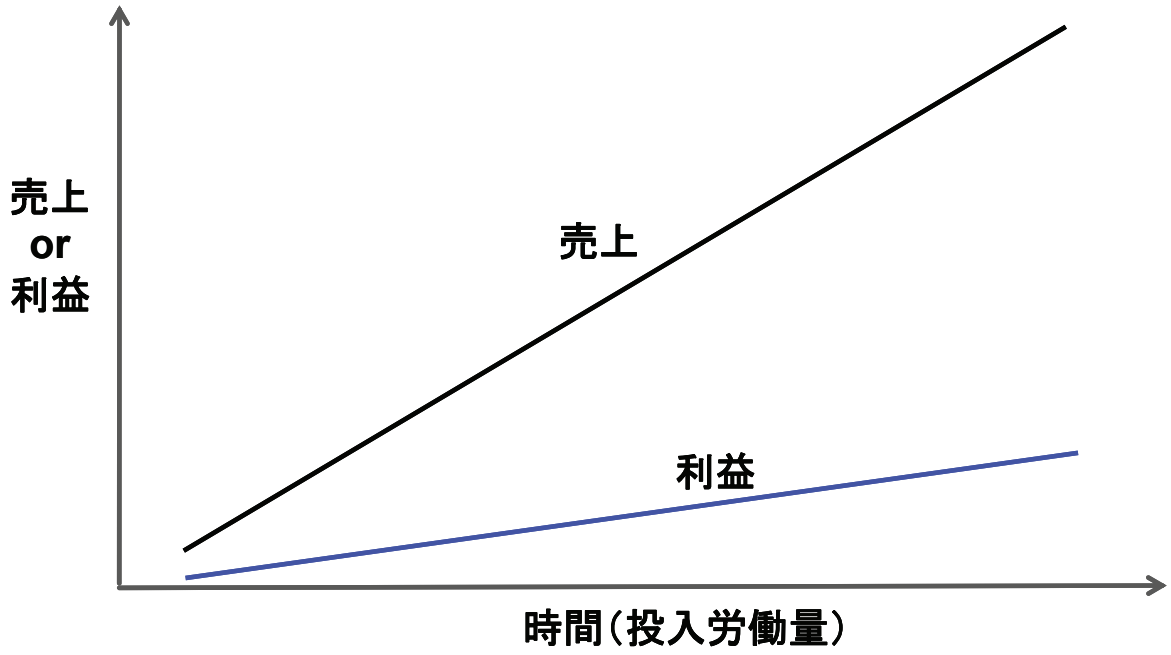
この契約の特徴は、プロジェクトが早期に終われば終わるほど、開発側の利益率が高くなることである。したがって開発側は生産性を上げようという強いインセンティブが働く。しかし、逆にスケジュールが大幅に遅延した場合、採算がとれず、赤字プロジェクトになるというリスクもある。

一方、顧客側からみると、あらかじめ決めた仕様に従ったシステムをあらかじめ決めた金額で入手できることになるので、リスクを全く負わないで済む便利な契約である。仮にプロジェクトが遅延して、コストが予定以上になった場合でも決まった金額しか支払わなくてよい。

この契約は明らかにアジャイル開発には適していない。それはスコープを最初に固定してしまうからである。

（３）実費精算契約（Time and Materials）

実費精算契約（Time and Materials）と呼ばれる契約は、日本における準委任契約に相当する。プロジェクトに投入した工数（労働量）や材料費に応じて支払い額が決まる。



図表 4-17　実費精算契約（Time and Materials）

この契約の特徴は、開発側からみた場合、プロジェクトが長引けば長引くだけ、売上も利益も増大するという点である。したがって、開発側には、生産性を上げようというインセンティブはあまり働かない。

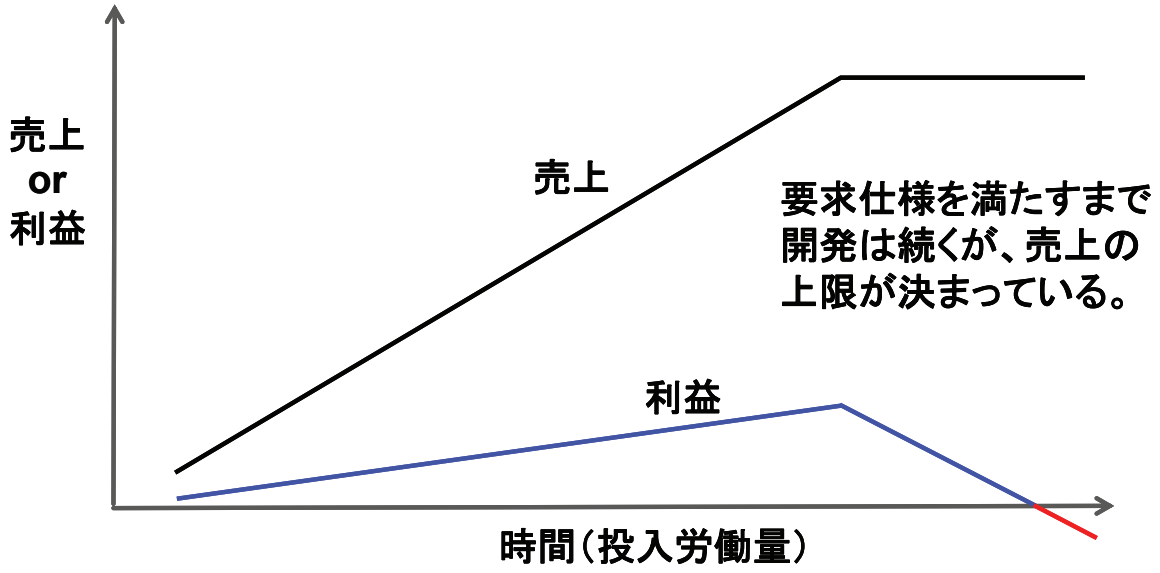
この契約方式においては、開発側には完成責任がなく、支払いの上限金額が決まっていないため、プロジェクトのリスクはすべて顧客が負う。したがって、顧客側がしっかりとプロジェクト管理を行わないと、欲しいものを適正なコストで入手することはできない。

スコープをいつでも変更できる点は、アジャイル開発に適しているが、顧客側が適切なプロジェクト管理を行うことが成功の条件であり、また、開発側に生産性向上のインセンティブを生まれる工夫をすることが望まれる。

（4）固定仕様、シーリングありの準委任契約

（Time and Materials with Fixed Scope and Cost Ceiling ）

スコープ（仕様）が固定され、かつ支払い金額に上限がある準委任契約である。要求仕様を満たすものが完成した時点でプロジェクトは終了する。



図表 4-18　固定仕様、シーリングありの準委任契約

予定より早く完成すれば、準委任契約型であるために支払い金額は上限金額より少なくなる。しかし、一般的に開発側は、利益（売上高）を最大化しようとするため、予定より早くプロジェクトが終了するケースは少ないだろう。また、予定より開発期間が長くなった場合、支払い金額にはシーリングがあるため、受け取る開発費は一定となる。つまり、開発スケジュールが遅延すれば、利益率は急減し、ある時期を超えると赤字プロジェクトになる。この点は定額請負型と同じである。

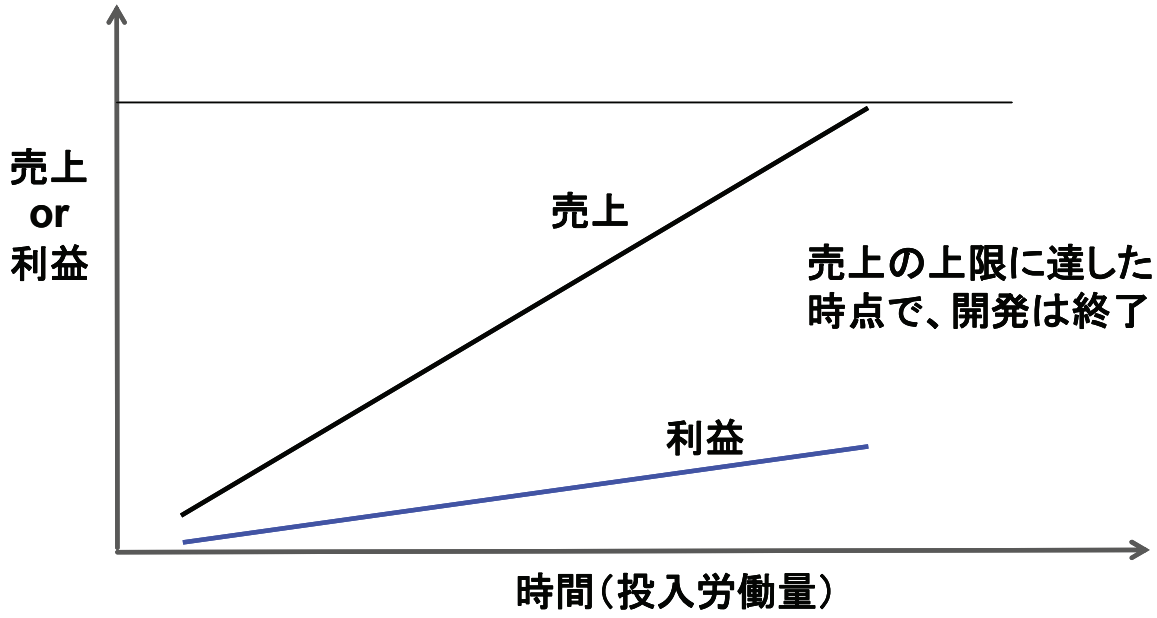
プロジェクトが早期に完成すれば支払い金額が少なくて済み、プロジェクトが長期化しても支払い金額は一定金額（上限額）を超えないという点で、明らかに顧客側に有利な契約形態である。

しかし、スコープを固定しているという点で、この契約はアジャイル開発には適していない。

（5）シーリングありの準委任契約

（Time and Materials with Variable Scope and Cost Ceiling）

支払い金額に上限がある準委任契約であるが、スコープが固定されていないという点で「固定仕様、シーリングありの準委任契約」とは異なる。



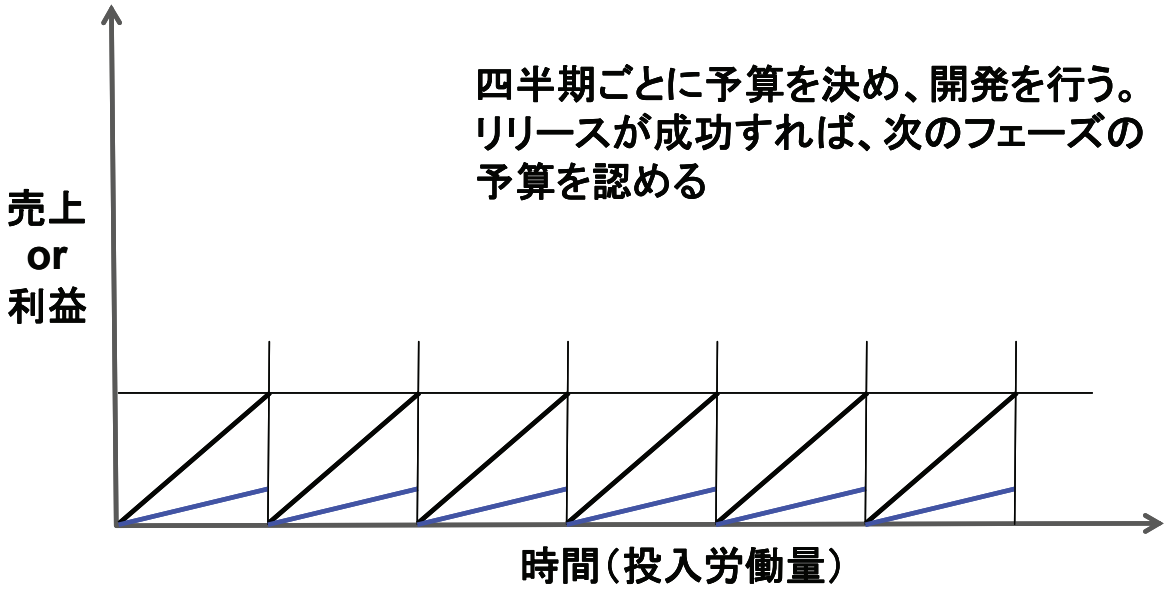
図表 4-19　シーリングありの準委任契約

顧客は希望するシステムが完成した時点でいつでもプロジェクトを終了させることができるが、その一方で予算を使いきってもシステムが完成しないというリスクを負うことになる（予算を使い切った段階で、システムが未完成でもプロジェクトは終了する）。開発側が受け取る金額（売上）と利益は、上限金額に達するまでは、時間とともに増大する。システムの完成責任は負わないが、顧客との継続的な取引を考えた場合には、上限金額に達するまでに顧客が満足するシステムを完成させようという若干のインセンティブは働くだろう。

スコープは固定されていないので、アジャイル開発に使うことができるが、開発側に生産性を向上させ、より早くシステムを完成させようというインセンティブが働く工夫が欲しい。

（６）段階的開発契約（Phased Development）

一定期間（たとえば四半期）でプロジェクト期間を分割し、その期間ごとに達成すべきスコープを定めて準委任契約を結ぶという方法である。その期間で開発した機能のリリースに成功すれば、次の期間の予算が与えられる。



図表 4-20　段階的開発契約（Phased Development）

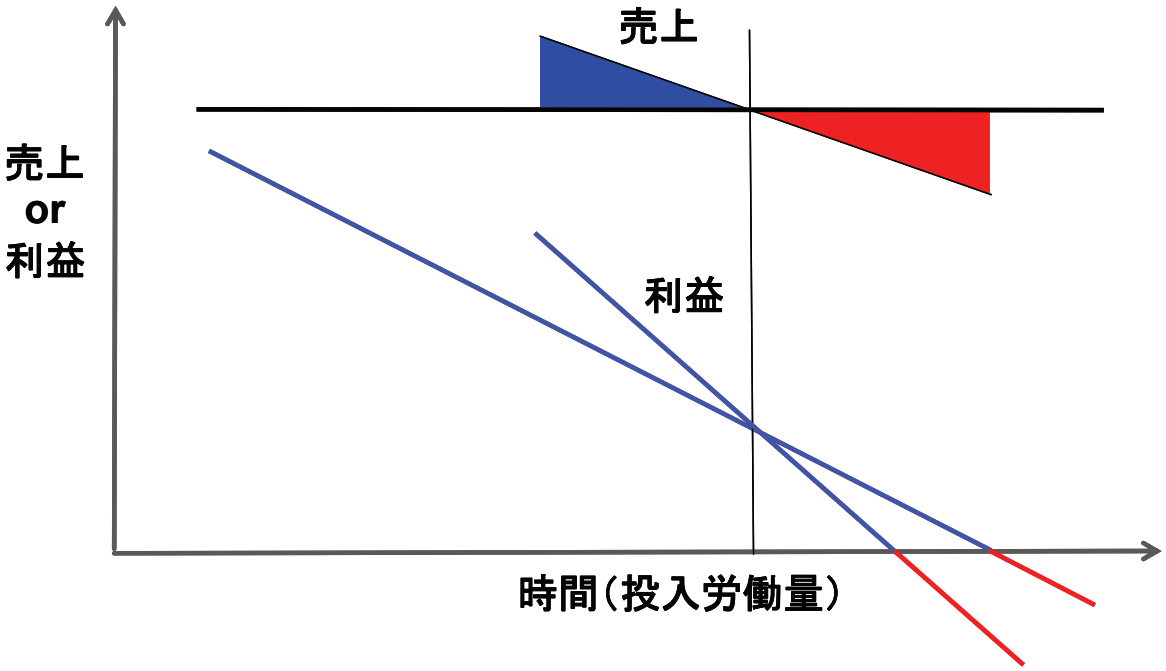
開発側にはその期間の開発を成功させようというインセンティブが働くと考えられる。また、顧客側には通常、できるだけ早く高品質なシステムを完成させたいというニーズがあるので、顧客と開発チームとの間によい協力関係が生まれる可能性が高い。

準委任契約ではあるものの、一定期間ごとにシステムがリリースされるため、最大リスクを一期間分の開発コストに抑えることができる。さらに、その期間の開発に成功しなければ次の期間の契約がなくなってしまうことを考えれば、顧客側のリスクは小さいと考えることができる。

期間を分割して契約を結ぶ点では、「スプリント契約」と同じである。
なお、期間ごとのスコープは固定しても、全体としてのスコープは固定しなくてよいので、アジャイル開発に適していると考えられる。

（7）ボーナス/ペナルティ条項（Bonus/Penalty Clauses）

開発が予定より早く完了すればボーナスを受け取り、遅延すればペナルティ（罰金）が科せられるという契約である。



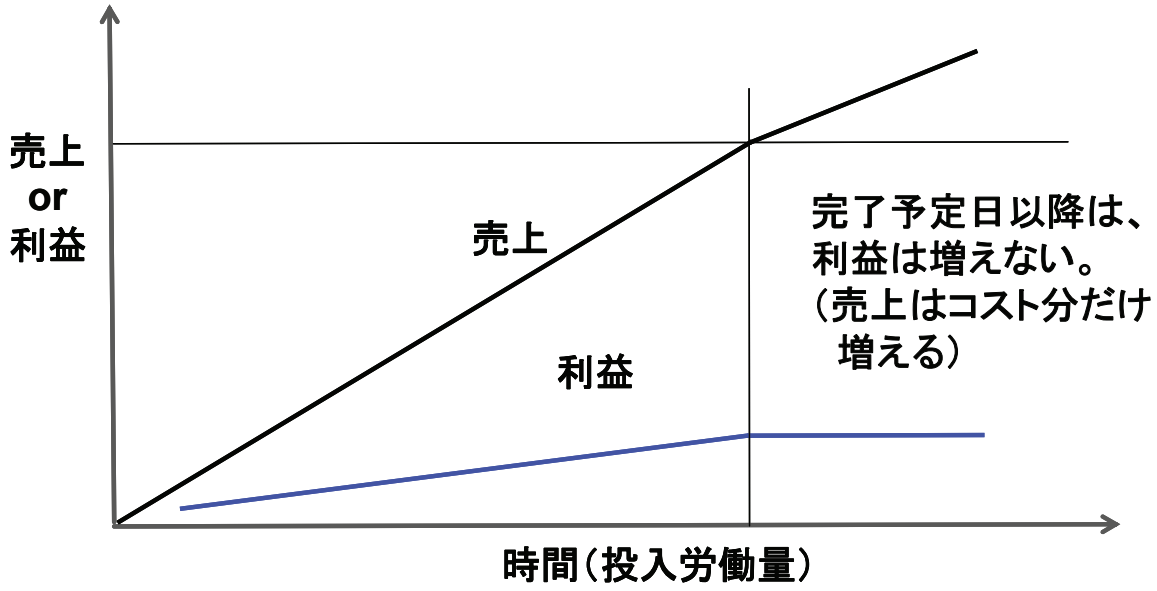
図表 4-21　ボーナス/ペナルティ条項（Bonus/Penalty Clauses）

この契約では、スコープ（仕様）が固定されていることが前提である。開発途中のスコープの変更は、プロジェクトの遅延につながる可能性が高く、開発側として受け入れられないだろう。開発側は早期に開発を完了させれば、売上も利益も増えるので早期完了に対するインセンティブはある。しかし、顧客側は開発が早期に完了すれば支払い金額が増える。もちろんシステムをできるだけ早く稼働させたいというニーズが強ければ、早期完了を望むこともあるだろうが、そうでなければ多少開発が遅延しても、支払い金額が少なくなることを期待するかもしれない。つまり、顧客が、早期完了のために積極的に開発に協力する保証のない契約だと言える。

スコープが固定で、両者に協力関係が生まれる可能性が小さいことを考えると、アジャイル開発には適していないだろう。

（8）固定利益（Fixed Profit）

この契約は、スコープが固定された実費清算型契約に近いが、予定期日を超えた場合には、超過分についてはコスト分しか支払われない。つまり開発側の利益は、予定を超過しても、予定通りに開発を終了した場合と同じになる。



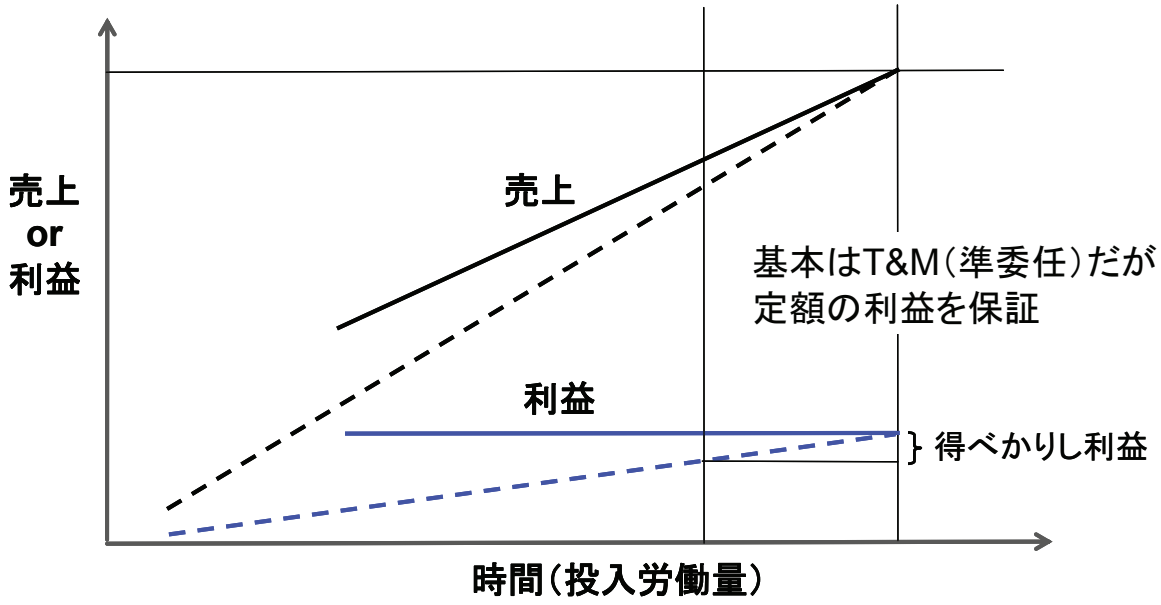
図表 4-22　固定利益（Fixed Profit）

一方開発が早期に完了した場合には、実費精算型と同じように支払い金額は少なくて済むので、顧客側には早期完了のインセンティブがある。しかし、開発側は、早期に完了すると（利益率は一定であっても）売上も利益も少なくなるので、早期完了のメリットはあまりない。逆に遅延すると顧客が支払う金額は増加する。開発側は、利益は増えないが、超過によって増えたコスト分はカバーできる。もし、開発側が売上高重視の企業であれば、遅延を歓迎するかもしれない。

スティーブンスはこの契約について、顧客、開発側の両者に早期完了のインセンティブがあると述べているが、売上高を重視する日本の企業を考えると、開発側に早期完了のインセンティブがあるとは思えない。この点に加え、スコープが固定であることを考えると、この契約もアジャイル開発には適していないだろう。

（9）Money for Nothing, Changes for Free

基本は、スコープの変更が可能で、予算上限のある準委任（実費清算型）契約であるが、早期完了（あるいはプロジェクト中止）になった場合には、開発側は、キャンセル料として残余期間で開発側が得るべき利益分を受け取る権利がある。つまり、どの時点でプロジェクトが完了、あるいは中止になったとしても開発側が得られる利益は一定となる。（予算上限があるので、予算を使い切った時点で開発は終了するが、この時も利益は一定である。）



図表 4-23　Money for Nothing, Changes for Free

この契約では、スコープの変更は自由に行うことができ、かつどの時点で開発を終了（あるいは中止）することも顧客の自由である。早期に終了（中止）すれば支払い金額が少なくなるので、顧客には早期完了のインセンティブがある。一方開発側からみれば、早期完了すれば、売上は少なくなるものの利益は変わらず、利益率を考えるとメリットは大きい。開発側が売上高ではなく利益率重視であれば、早期完了のインセンティブがあることになる。

スコープが固定でなく、両者に早期完了のインセンティブがあることを考えると、アジャイル開発には適していると考えてよいだろう。

（10）ジョイントベンチャー（Joint Venture）

ジョイントベンチャーは、その名前のとおり、顧客と開発側がコストもリスクも折半するという仕組みである。開発したシステムをパッケージソフトとして他の顧客に販売できるような場合に利用可能である。

顧客と開発企業でつくるチームを仮想的な企業と考えれば、社内開発と同じであり、アジャイル開発には適しているだろう。しかし、スコープ（仕様）や納期などについて両者の意見、優先度が一致しない可能性がある。場合によっては両者が対立し、意思決定に時間を要するかもしれない。あらかじめ、決定権限が誰にあるのかをきちんと決めておく必要があるだろう。

参考文献

[1]「第二部　アジャイル開発のモデル　付録１　ビジネス構造モデルによるアジャイル開発事例の分析例」を参照
[2] Peter Stevens,10 Contracts for your next Agile Software Project,(2009),
http://agileoftwaredevelopment.com/blog/peterstev/10-agile-contracts
[3] 前川徹, アジャイルソフトウェア開発に適した契約,2010, より転載
http://www.csaj.jp/column/index.html
[4] Mary Poppendieck& Tom Poppendieck,
Lean Software Development(Addison-Wesley,2003)［訳: 平鍋健児/高島裕子/佐野建樹,リーンソフトウェア開発~アジャイル開発を実践する 22 の方法(日経 BP 社,2004)］
[5] Lars Thorup, Bent Jensen, Collaborative Agile Contracts, agile, pp.195-200, 2009 Agile Conference, 2009
[6]「アジャイルが大企業に採用されるために解決すべきたったの２つの課題」
http://enterprisezine.jp/article/detail/2391

第五部　アジャイルにまつわる開発手法体系の歴史と動向

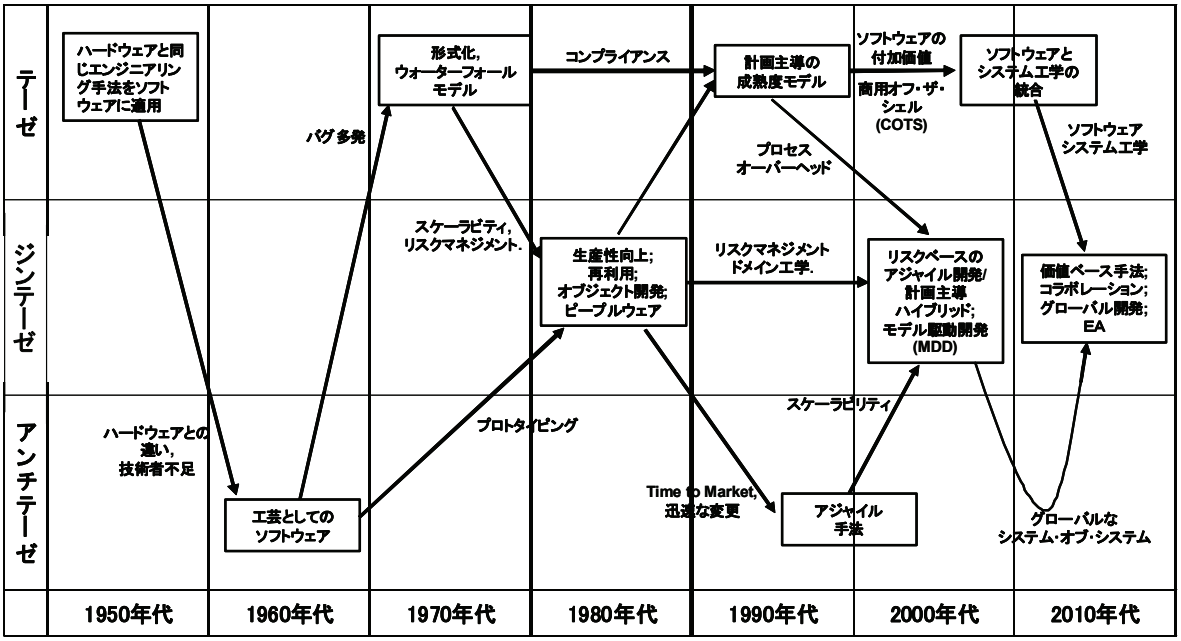
はじめに

アジャイル開発手法についてより深く理解し、その本質をさらに探求したい読者のために、アジャイル開発手法にまつわる歴史をひも解くとともに、最近の動向について簡単に紹介する。やや難しい言葉を用いている部分もあるが、読者のコンテキストに応じた最適な開発手法を模索する取組みの一助となれば幸いである。

１　歴史的経緯

ドイツの哲学者ヘーゲル（Hegel, G.W.F.） は、「真理とは、基本的に主観的なものであり、歴史を貫く思考の鎖、すなわちテーゼ（定立)、アンチテーゼ（反定立)、ジンテーゼ（総合）というトリオの連鎖の中にひそむ法則の認識である」と述べている。ソフトウェア現場が生き残るためには、定立、反定立の渦の中で、自己を見失うことなく、ソフトウェア・エンジニアリングの歴史的流れを弁証法的に分析し、洞察によって真理を主観的に見極め、それに導かれて事業を進めることが望ましい [1]。

ベーム (Barry Boehm) は、2007 年の IEEE International Conference on Software Engineering において、弁証法的手法を使って、ソフトウェア・エンジニアリングの歴史を、図表 5-1 のとおりまとめている。ベームの見方によれば、定立（上辺）はいわゆる「計画駆動手法」であり、それに対する反定立（下辺）は、1960 年代の現場の職人気質、および 1990 年代のアジャイル手法であるとしている[2]。

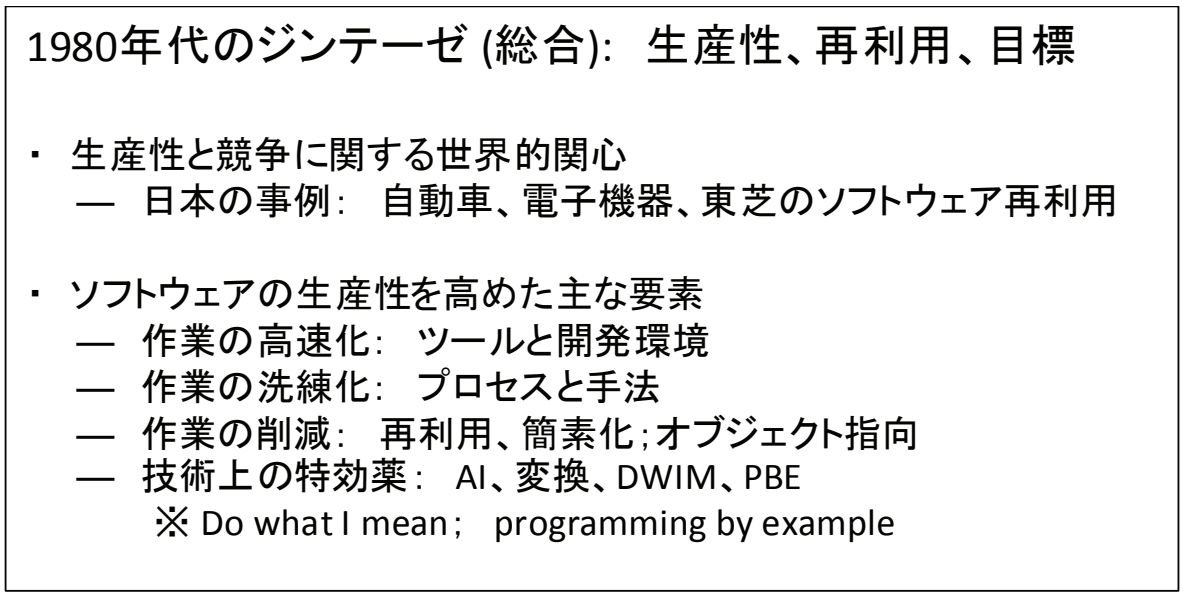


出典：[Boehm06]から引用[2]、© USC-CSE

図表 5-1　B. Boehm による、弁証法に基づいたソフトウェア・エンジニアリングの歴史分析

1990 年代の後半に生まれたアジャイル手法は、システム要求の大規模化、不透明化、複雑化、予測困難性、TTM（time-to-market）短縮、資源不足など、環境の変化から生まれた反定立であるが、決して「無計画（unplanned)」、「技術的・組織的無躾（undisciplined)」を意味するものではない。しかし、すでに 10 年を経過した現在、1990-2000 年代のアジャイル手法に代わる新しい反定立も求められている。

1980 年代は、日本の生産技術が世界で大きく評価された 10 年であった。ベームは、80 年代のジンテーゼ（総合）を、図表 5-2 で示すように要約している。ここに記載されている日本の事例の中の自動車が「リーン開発手法」になり、東芝のソフトウェア再利用が「ソフトウェアプロダクトライン」として、CMU/SEI によって国際的に展開された。



出典：[Boehm06]から引用[2]、© USC-CSE

図表 5-2　Boehm による 1980 年代のジンテーゼの内容

２　国際標準における開発プロセスモデルと知識体系

はじめに、「モデル」という語彙に対する正しい認識をもつ必要がある。「モデル」は、プログラム意味論によって定義された概念である。すなわち、閉じた論理式ϕがあるとき、ストラクチャ S において、ϕの解釈が真になる場合、S はϕのモデルであるという。

第五部３から５に記載する各モデルは S であり、企業現場で実践されているライフサイクルプロセス構成論理や製品構成論理は、ϕに当たる。国際標準における S は、複数のϕの解釈を真にするように作られる。企業が異なれば、それぞれの環境や文化に合わせて、ϕが構築されるが、その解釈を真にする S を統一することができる。

ソフトウェア・エンジニアリングでは、数多くのソフトウェア開発プロセスモデル、知識体系が扱われている。しかし、これらモデルは、あくまでも実践論理や設計・実装論理を解釈するための「モデル」であり、実際の企業現場で実践されているライフサイクル構成論理や製品構築論理、そのものではない。

ISO/IEC JCT1 SC7 (Software and Systems Engineering) は、ISO/IEC 15288:2002 (System Life Cycle Processes)、ISO/IEC 12207:1995 (Software Life Cycle Processes)、ISO/IEC TR19759 :2004 (Software Engineering Body of Knowledge)、および現在審議されている NWIP (new work item proposal) N4585 Systems Engineering Handbook (2010-2-25) （INCOSE SE Handbook）[3] などによって、エンジニアリングプロセスモデルの標準化を進めてきた。また、現在、SC7 は、Stevens Institute Technology が主宰して、策定中の Systems Engineering Body of Knowledge (SEBoK : 別名 BKCASE－Body of Knowledge for Complex and Adaptive Systems Engineering) に協力している。

SEBoK は、モデルを、(1) ライフサイクル視点、(2) サービス統合視点、(3) 組織およびエンタプライズ(ビジネス)視点、に分けて、定義している。この 3 つの視点の中で、この報告書に関係するのは、以下に示す、ライフサイクル視点から見たモデルである。
INCOSE SE Handbook および SEBoK[4] では、ライフサイクル視点から見たモデルを、次に示す 3 種類のモデルに集約して説明している（ソフトウェア・エンジニアリング現場でのプロセス構成論理そのものではないことに注意されたい)。

1) Vee Model

このモデルは、次の①から④までのモデルをまとめて、ひとつのモデルで表現する。

① Single-Step-to-Full-Capability (SSFC): CDPU;

② Pre-specified Sequential (PS): CD; →PU1; →PU2; →PU3; …….

③ Evolutional Sequential (ES): CDPU1; →CDPU2; →CDPU3; …….

④ Evolutional Overlapped (EO):

CDPU1;

⇒CDPU2;

⇒CDPU3;

⇒ （次の CDPU）

2) Spiral Model

3) Scrum Model

記号の説明：

(C: concept, D: development, P: production, U: utilization)

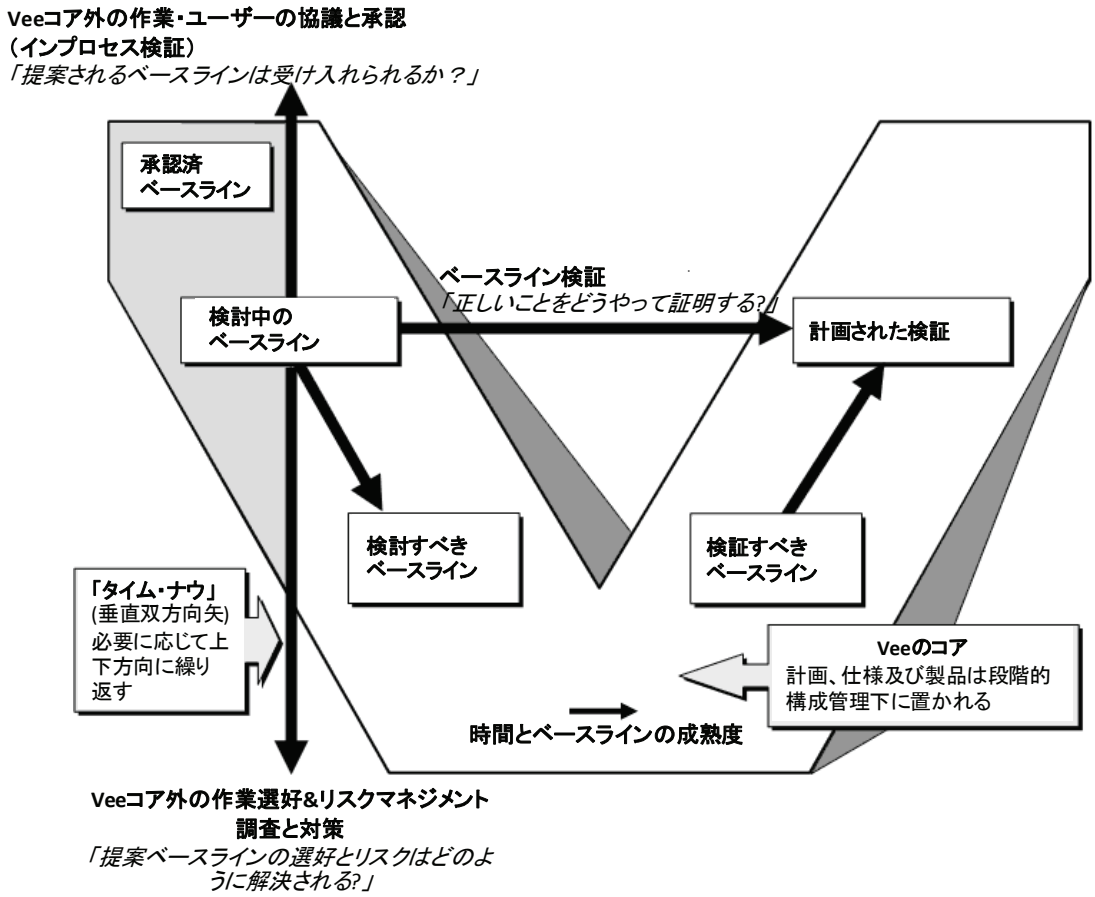
→：逐次的な連接を示す。

⇒：少し時間的に遅れて、重畳しながら並行して追従する接続を示す。

以降の節で、各モデルについて説明する。

3　Vee モデル

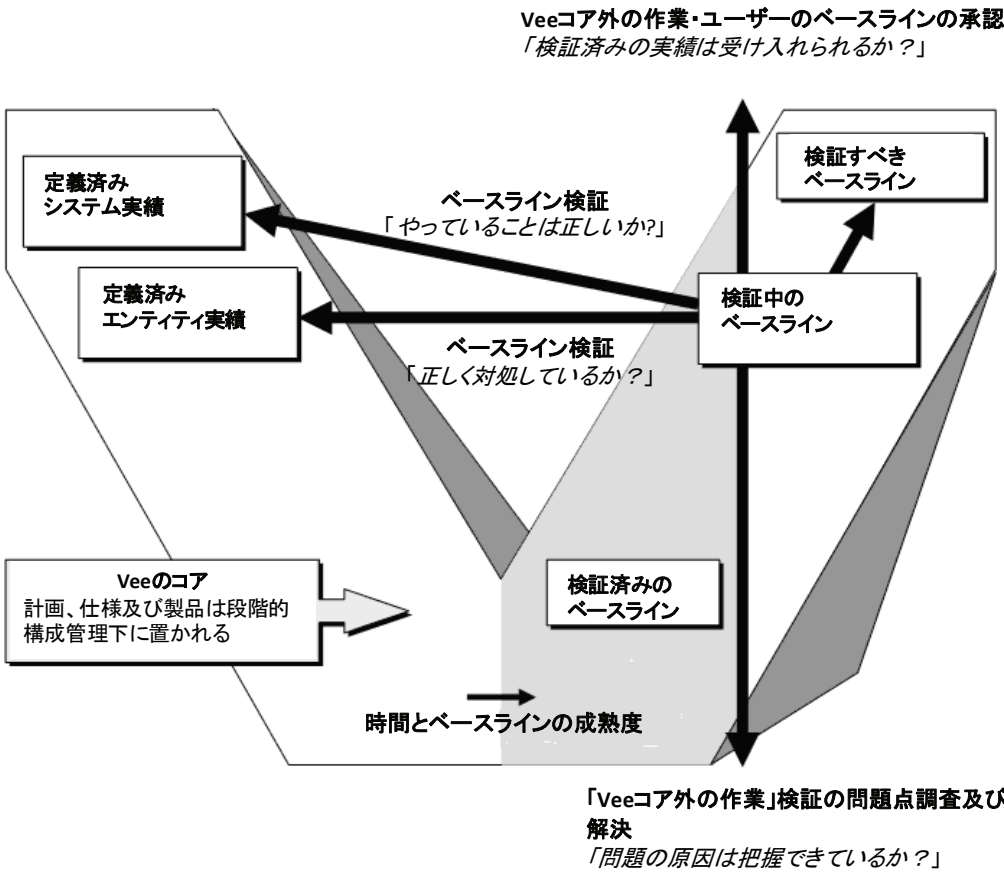
Vee モデルは、図表 5-3 および図表 5-4 に示すとおり、太い V 文字（これを「V コア」と称する。なぜ太字にするかについては、後に説明する）および垂直双方向矢（垂直双方向矢以外の矢は、補足的なもので、重要ではない）で表現される。垂直双方向矢は、必ず、矢の線上に長方形をもつ。この長方形は、ベースライン成果物を表す。Vee モデルの水平軸は、時間、およびベースライン成果物の成熟度を表す。このモデルの垂直軸は、上方向がベースライン成果物に対するユーザ承認度、下方向がベースライン成果物に対する選好（opportunity：リスクの反対）およびリスクに対するマネジメント負荷の大きさを表すとされ、具体的な測度（スケール）は規定されていない（当事者がアプリケーションに合うように決めればよい）。垂直双方向矢は、時間、およびベースライン成果物の成熟度の進展とともに、右へ平行移動する。



出典：[INCOSE10]から引用[5]

図表 5-3　Veeモデルの左半分 [1]

[1] 長方形は、ベースラン成果物を表し、単方向矢は、成果物間の適合関係を表す。

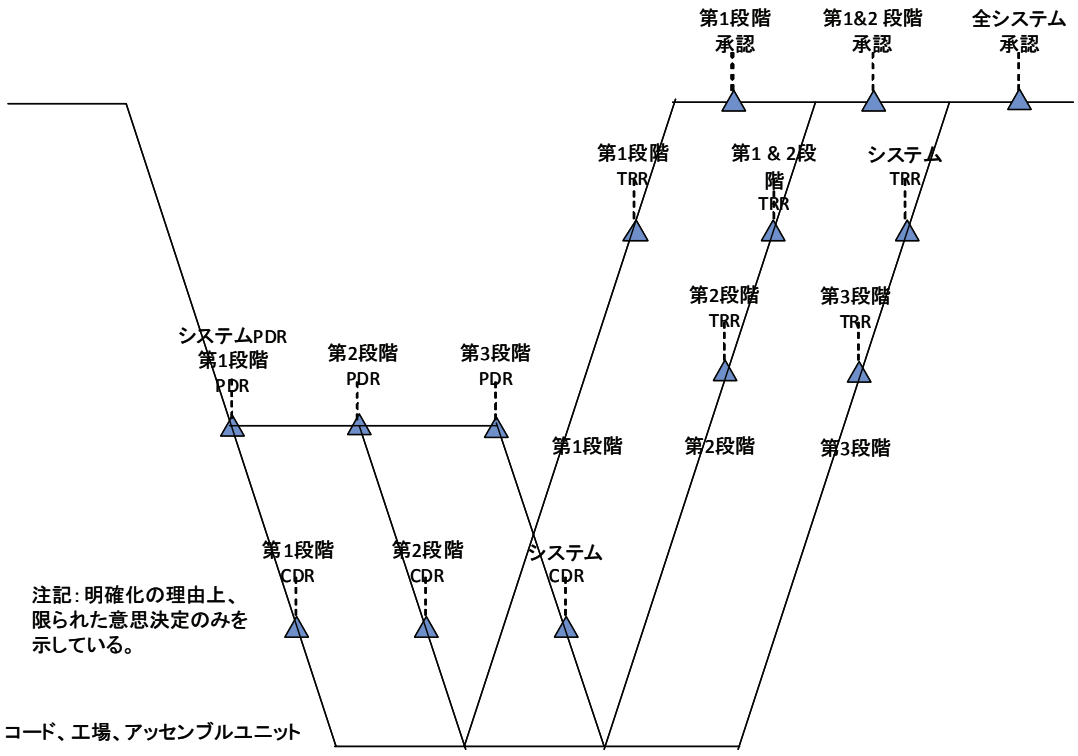


出典：[INCOSE10]から引用[5]

図表 5-4　Vee モデルの右半分

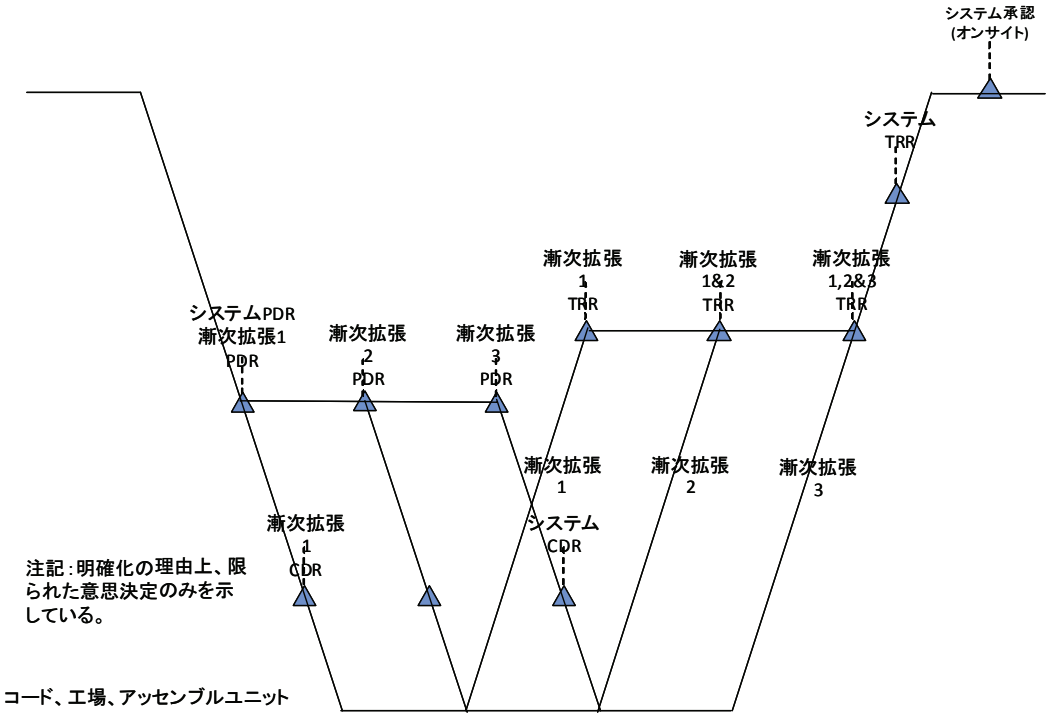
これら Vee モデルの中の V 文字は、我が国の「共通フレーム」[6]が採用しているモデルと同じ意味をもつ。V 文字を太くしている理由は、先に述べた SSFC, PS, ES, EO モデル（これ以外にも存在する可能性が高い）を包含していること、およびこのモデルによって解釈できるライフサイクルプロセス論理構成（次に述べる）が、極めて多く存在することを表す。

BKCASE では、Vee モデルによって解釈できるライフサイクルプロセス論理構成の例として、モデル指向ソフトウェア・エンジニアリングおよびリーン・システムエンジニアリング（たとえば、LEfSE: Lean Enabler for Systems Engineering）を挙げている[7] 。これらにおけるプロセス構成論理の例として、図表 5-5、5-6、5-7 を掲げている。



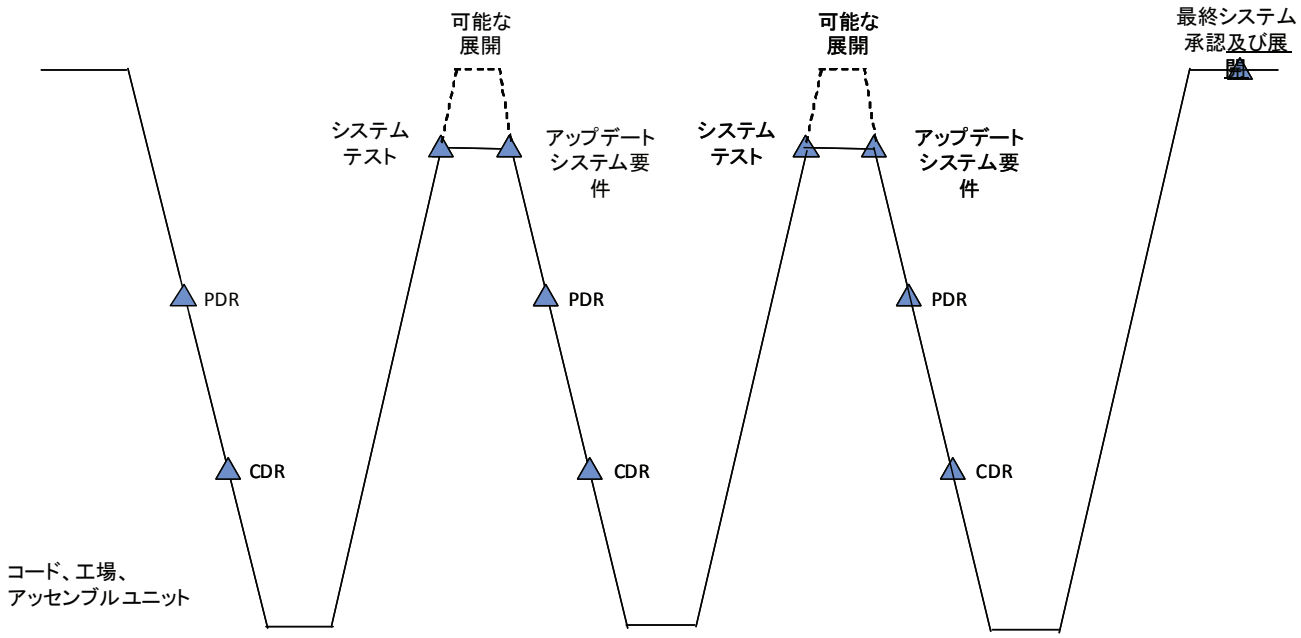
出典： [BKCASE10]から引用[4]

図表 5-5　漸次拡張・分割継続納入開発プロセス構成例



出典： [BKCASE10]から引用[4]

表 5-6　漸次拡張・一括納入開発プロセス構成例



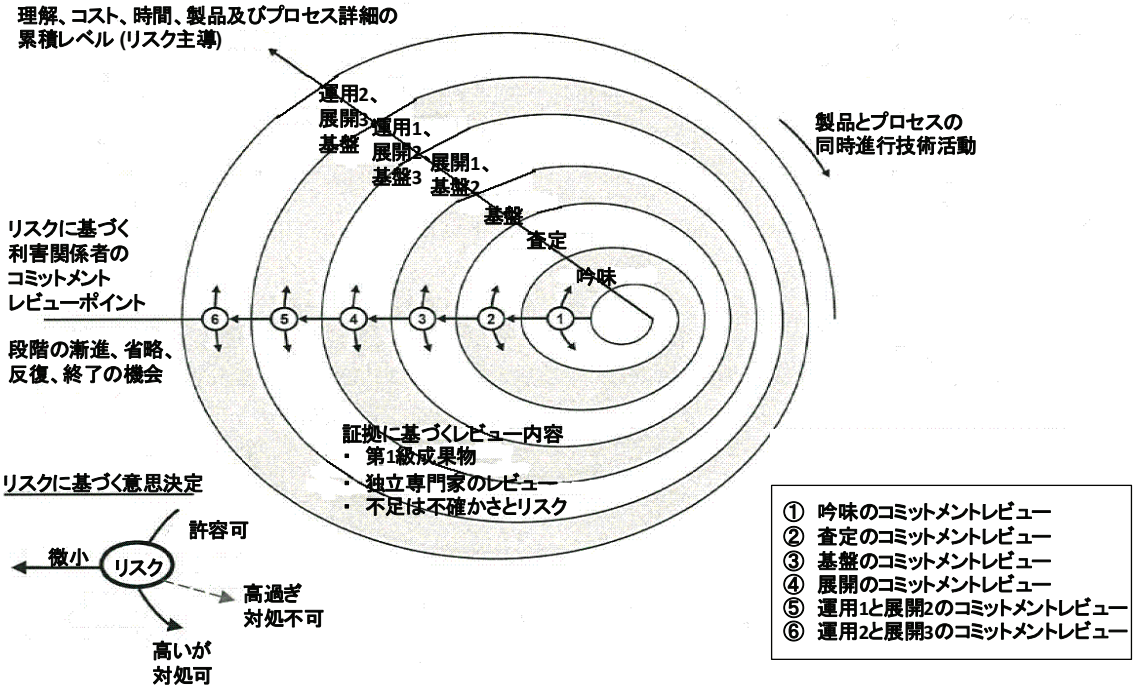
出典： [BKCASE10]から引用[4]

図表 5-7　進化型開発プロセス構成例

記号の説明：
PDR: preliminary design review　概要設計レビュー
CDR: critical design review　最終設計レビュー
TRR: test readiness review　検査レビュー
IV&V: integration, verification and validation　独立検証及び有効性確認
FVR: final validation review　最終確認レビュー

４　スパイラルモデル

ICSM (Incremental Commitment Spiral Model) とも呼ばれるように、図表 5-8 で示す①から⑥までのプロセスを、①から、順に漸進する。次のプロセスへの漸進は、リスクに準拠した意思決定指標に合格したときに行われる。合格しないときは、もとのプロセスを反復する。

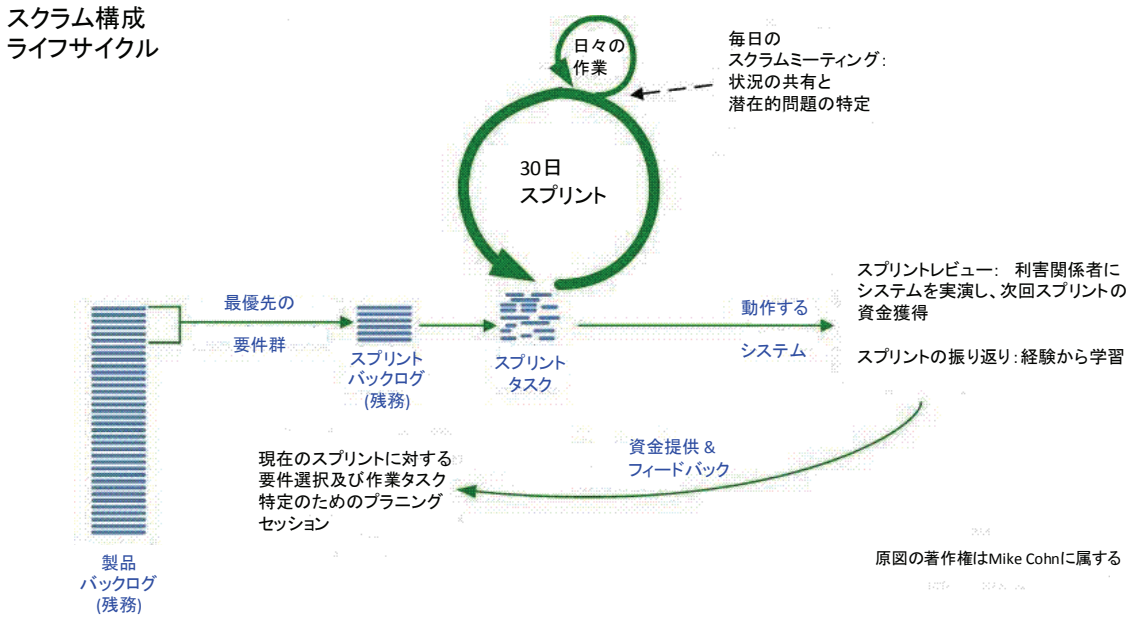


出典：［BKCASE10］から引用[4]

図表 5-8　リスクを意思決定指標として用い、反復・漸進する
開発プロセスのためのモデル

５　スクラムモデル

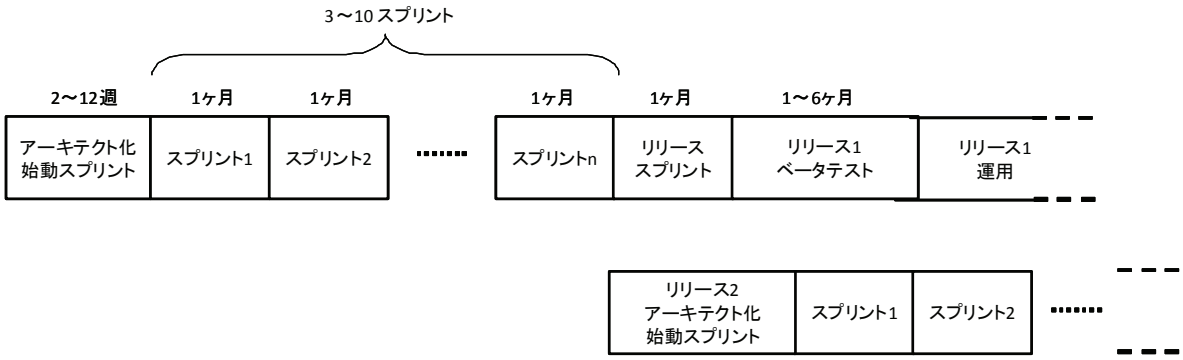
一般的なスクラム（Scrum）モデルで使われる主な意味要素は、図表 5-9 に示すように、(1) product backlog, (2) sprint backlog, (3) sprint task である。これら意味要素によって解釈できる働きをもった具体的なプロセス構成要素を集め・組み立てて、システム要求の大規模化、不透明化、複雑化、予測困難性、TTM（time-to-market）短縮、資源不足などに適応するソフトウェア開発手法を、アーキテクト化されたアジャイル手法（Architected Agile Methods）と呼ぶ。



© 2005-2008 Scott W. Ambler, Original copyright Mike Cohn）

出典：The Agile System Development Life Cycle (SDLC) [8]から引用

図表 5-9　Scrum モデル



出典：[BKCASE10]から引用[4]

図表 5-10　アーキテクト化されたアジャイルプロセス構成の例

図表 5-10 に示すアーキテクト化されたスクラムプロセス構成（論理）では、始動スプリント（sprint zero: 2-12 週）はスプリント全体計画に使われ、スプリント長は 1 カ月、3-10 カ月かけた開発スプリントの後にリリース・スプリントに入り、ベータテスト・スプリントを経て、運用に入る。開発スプリントが終わる頃、次の sprint zero を開始する。これに相当するプロセス構成は、白物家電製品に組み込まれる電子制御ソフトウェアの生産プロセスで実践されている。

また、商品（金融商品、食品、雑貨、金券など）販売のためのウェブサイト（ポータルサイト）開発では、データ以外の機能部品の再利用を行わず、データ構造を key-value pair 化し、データテーブルの垂直/水平分散化を行うことによって、データ層プラットフォームをスケーラブルにし、機能部品開発のアジャイル性を高める事例が多い。このようなアプリケーション開発では、スクラムモデルがよく利用されている。

６　新しい動き

６．１　国際標準における新しい動き

ISO/IEC JTC1 SC7 WG02 は、ISO/IEC FCD 26515「User documentation in an agile environment」と称する標準案を 2010-12-10 に発行し、2011-5-17 までに投票を行うことになっている（全 38 ページ）。

この標準案では、ユーザがソフトウェアライフサイクルで用意しなければならない文書として、下記の項目が規定されている。この標準案は、これら文書を作成することを義務付けるものではなく、文書の種類とその目的および概略の記載内容を標準化しようとするものである。

・project plans;
・sprint plans;
・requirements documents, (expressed in user stories, scenarios);
・high-level design proposals, (may not be needed for agile development);
・test plans, (test procedure);
・risk statements, (risk register);
・user stories;
・use cases;
・descriptions of personas;
・burn-down charts;
・task lists;
・scrum reports;
・end of sprint lessons learned reports;

６．２　アジャイル手法を大規模システムに適用した経験報告

（1） アジャイル開発は、「プロセス」より「人」

アジャイル手法を取り巻く現況を要約すると、次のとおりである。

ⅰ）商用システム開発にもアジャイル手法の採用を検討するところが増えている。

ⅱ）アジャイル開発は、「小さな、同じ作業スペースを共有するチーム」という枠を超えて、当初、認識されていた「comfort zone」から外へ出てきたため、当事者（people）および人的資源マネジメントに関して、新しい課題を抱えることになった。

ⅲ）初期段階では、アジャイル開発を採用するかどうかの意思決定は、主に、局所的（insular）、ボトムアップ的、自発的(voluntary）に行われ、プロジェクトチームは、アジャイル開発へ移行するか、排除するかを「自分の思うままに（on its own terms)」意思決定することができた。しかし、最近は、サプライヤ、コンサルタント、パートナー、顧客、それに公的機関（public sector bodies）でさえ、必然性や組織間のプロセス整合に関する必要性を説いた形式的要請などの発行によって、アジャイル手法を強要するようになっている。

ⅳ）このような状況を受けて、経営者、とくに人的資源管理部門、プロジェクトマネージャは、従来とは極端に異なる（significantly different）環境に向き合うために必要な、スキル、当事者課題の解明と解決に力を入れるようになった。

日本に限らず、世界各地の商用プロジェクトで、アジャイル開発手法の試行が行われている（国内の試行状況は、「非ウォーターフォール型開発に関する調査報告書」（IPA/SEC, 2010）に記載)。

「Key People Challenges in Agile Development,」（IEEE Software） [9]では、3 年間以上に亘ってアジャイル手法を試行してきた、17 の大きな多国籍団体組織で経験した、次のような内容を報告している。

①「スキル欠如が透明化できないことに起因する開発者の危惧（fear)」

②「開発者全員が、すべてのやりとり（trades）に熟達する（master）必要性」

③「社会性スキル（social skills）への依存性増大」

④「開発者におけるビジネス知識の欠如」

⑤「アジャイル・プラクティスではなく、アジャイルの価値と原理（principle）を学ぶ必要」

⑥「アジャイル手法の利用に対する開発者のモチベーションの欠如」

⑦「意思決定の譲渡（devolve）が行きつくところ」

⑧「行動が、アジャイル性に適合しているかどうかを評価する必要性」

⑨「アジャイルに特化した採用および適切に訓練された IT コース卒業生獲得に対する戦略欠如」

（2）ソフトウェア・エントロピの増加

イテレーションごとに品質作り込みを行うアジャイル手法として、Evo があることは、良く知られている（第一部２.１参照）。Evo を大規模システムに対して 14 年間に亘って試行してきた、NTNU IDI 社の実証経験が報告された[10]。内容は、おおよそ次のとおりである。

ⅰ）大規模システムの進化に対して、アジャイル手法を適用して保守・改善を続けた結果、ソフトウェア・エントロピが増大（aging 老化）し、構造が輻輳化し、管理できなくなり、アジャイル開発を続けられなくなる危険性に当面した。

ⅱ）対策として、NDepend[2]ツールを使って、イテレーションの度ごとに、構造の整列を行い、リファクタリングを行った。

ⅲ）スプリント期間中に、構造整列、リファクタリング、working software の顧客レビュー・承認、実装を行おうとしたが、当初決めたスプリント期間はとても守れなかった。

ⅳ）リファクタリングは、業界で説明されているほど、簡単な作業ではない。ごく限られたリファクタリング手法を選定して、これを自動化するツールを開発し、これを利用しながら行った。

６．３　これからへ向けて

ソリューションは、実践現場でのたゆまない問題解決の積み重ねから生まれるもので、国際標準のようなジンテーゼを知識・概念として学習しても、それだけでは開発実践論理の創生・改善にはつながらない。アジャイル手法は、図表 5-1 が示すように、反定立として登場してからすでに 10 年を経過し､今の時代に即した新しい反定立が求められている。

新しい動きのひとつとして、ソフトウェアがソシオ・テクノロジと切っても切れない調和へ向けて進む中、サンデル（Michael J. Sandel）の共同体主義[11]が示唆するような、定立/反定立を巻き込んだコミュニケーションからジンテーゼを発見し、これに裏付けられたソリューションを目指す流れが注目されている。たとえば、そのひとつの例として、UX design（user experience design：ユーザ経験による共同設計）〔user stories ⇒ research prototyping ⇒ formative testing ⇒ UX design ⇒ user stories へのフィードバック ⇒反復〕がある[12]。

日本企業では、70 年代から、設計仕様書、フローチャート、入出力仕様などを顧客が承認する商慣習に基づいた､「顧客の設計参加」という伝統的メカニズムが現在も存続している。また、最近では、SAP の Polestar チームなども似た方向へ進路を向けており、多視点共同体による、計画駆動手法とアジャイル手法の総合化が今後の動きになっていくと考えられる。社会基盤の国際入札などで成功するためには、日本の伝統的商慣習の中から、これからの国際市場が求める正義を見出し、自信をもってそれを開発手法に育てる努力が必要である。

---

[2] NDepend は、.NET の静的コード解析ツール。 コードベースのスナップ比較、ビルドのプロセスの分析、「コードクエリ言語」(CQL)を使ったコード分析等の機能がある。

７．非ウォーターフォール型ソフトウェア開発の全体像
（Perspective of Non-Waterfall-type Software Development）

７．１　非ウォーターフォール型開発の分類

ソフトウェアが中心となるシステム（software-centered systems、以下システムと称する）におけるこれからの主流は、「適応複合型システム」（adaptive complex systems）であるとされる [13],[14]。

非ウォーターフォール型ソフトウェア開発（以下、非ウォータフォール型開発と略称）は、「適応」と「複合」を次のように解釈した場合、「適応複合型開発」とほぼ等価な意味をもつ。

①適応：システムを適用する実世界が、不確実であり、短い期間をおいて変化・変動することへの適応を意味する。

②複合：システム構築に対して利害関係をもつ個人、または組織の数が大きく、また物理的に広い範囲に分散していること。さらに、システム構築に必要なソリューション構成要素の数および種類の数が大きく、分散して存在することを意味する。

（１）大分類

非ウォーターフォール型開発は、システムの全ライフサイクルを行動範囲とし、プロセス視点による分類、プロダクト視点による分類、およびソフトウェア実装基盤から見た分類に大別される。

ｉ）プロセス視点による分類

① 一括納入を条件とした｛反復開発・一括納入・据付け・運用・保守｝

② 分割納入を条件とした｛漸次拡張開発・納入・継続的統合・運用・改善｝の反復

ii）プロダクト視点による分類[3]

iii）ソフトウェア実装基盤から見た分類[4]

[3] 図表 5-11 および（２）プロダクト視点による分類を参照

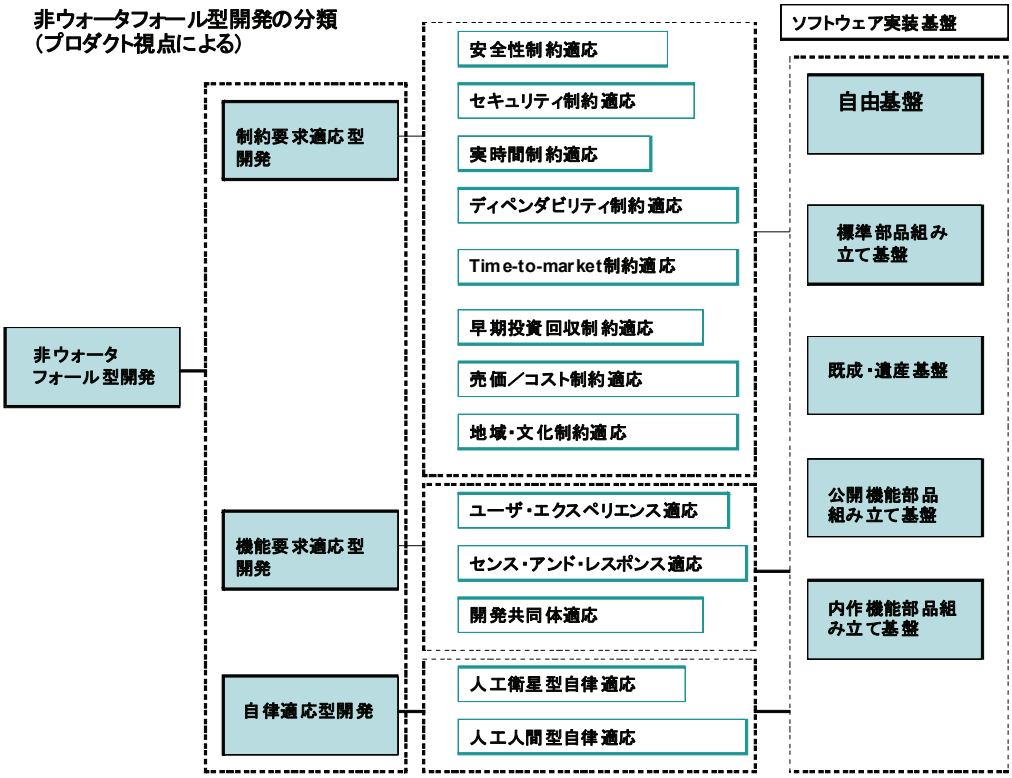
[4] 図表 5-11 および（３）ソフトウェア実装基盤から見た分類を参照

（2）プロダクト視点による分類

次の 3 つに分類できる。

ⅰ） 制約要求適応型（主な変化・変動因子が、制約（非機能要求）である場合）

制約には、図表 5-11 で示すようなものがある。いずれも自然環境、社会環境、経済環境によって可変である（たとえば、国際標準で規定される SIL（safety integrity level）の値を例にとってみても、国民年齢構成、社会の文化水準、社会の安全観念などの変化によって、時とともに変動する）。制約は、アプリケーション・アーキテクチャ、またはフレームワークを決定する。したがって、制約を可変にすることは、アプリケーション・アーキテクチャを可変にして適応することになり、アーキテクチャ設計に対する基本概念を改める必要がある。アーキテクチャを単一層（平坦構造）とし、機能要素およびデータ要素をキー・バリュー 方式(key-value pairs）にするなどして、可換化構成（reconfigurable）にする方式などが検討されている。



図表 5-11　非ウォーターフォール型開発をプロダクト視点に立って分類した構造

ⅱ）機能要求適応型（主な変化・変動因子が、機能要求である場合）

アプリケーション・アーキテクチャ、またはフレームワークを、原則として変更することなく、プログラム、またはデータを可変にして適応する方式である。プログラムは、スクリプト型言語、ドメイン特化型言語（DSL: domain-specific language）などによって記述され、ユーザが意思を自ら反映できる方法が志向される。

①ユーザ・エクスペリエンス適応：　システムとユーザ間の操作・対話における変化・

変動を問題として捉え、ユーザーストーリーをユーザ・エクスペリエンスとして定性化・定量化して、これに適応する手法。

②センス・アンド・レスポンス適応：　ビジネス環境・ビジネスプロセスの変化によるシステムの不適合を短い時間内に検出して、それに対して高速に対応する手法。とくに危機管理系システムで必要とされる。

③開発共同体適応：　システムの規模が大きいため、利害関係者の主観が文化・教育・生活慣習などの点で異なり、確定的でない場合、利害関係者から構成される共同開発室を組織して、適応する手法。スマートシティなど、大規模公共システムなどで必要とされる。契約を複数に分け、第１次契約において共同開発室を組織し、設計基準概念（design reference）を策定し、入札等によって製作者を選定し、第２次契約以降において開発・実装・据付け・運用・保守を行うなどがある。

iii）自律適応型（主な変化・変動因子が、利害関係者のスコープ外にある環境に依存する場合）

人がアクセス不可能な物理的環境の中で起きる環境変化に対して、自律的、または半自律的に適応できることが求められる。過疎地にある公共サービス機関や人工衛星のように、人手による環境変化検知および対応が難しいケースでは、システム自身が自動的に適応できるように、初期納入以前に設計しておく必要がある。

（３）ソフトウェア実装基盤から見た分類

アプリケーションは、仮想化されるプラットフォーム（ハードウェア、OS）上のソフトウェアフレームワーク（アプリケーション・アーキテクチャを含む）に実装される。ここでは、ソフトウェア実装基盤は、ソフトウェアフレームワークを意味するものとする。

①自由基盤を前提とする方式：データ基盤を除いて、その上に実装されるアプリケーションは、バックログに入力されたユーザーストーリーに従って、継続的統合環境を利用してすべて作り直す。

②既成・遺産基盤を利用する方式：すでに存在する層基盤（layered framework）上で、層内部および層間連携部だけを、バックログに入力されたユーザーストーリーに従って作り直す。

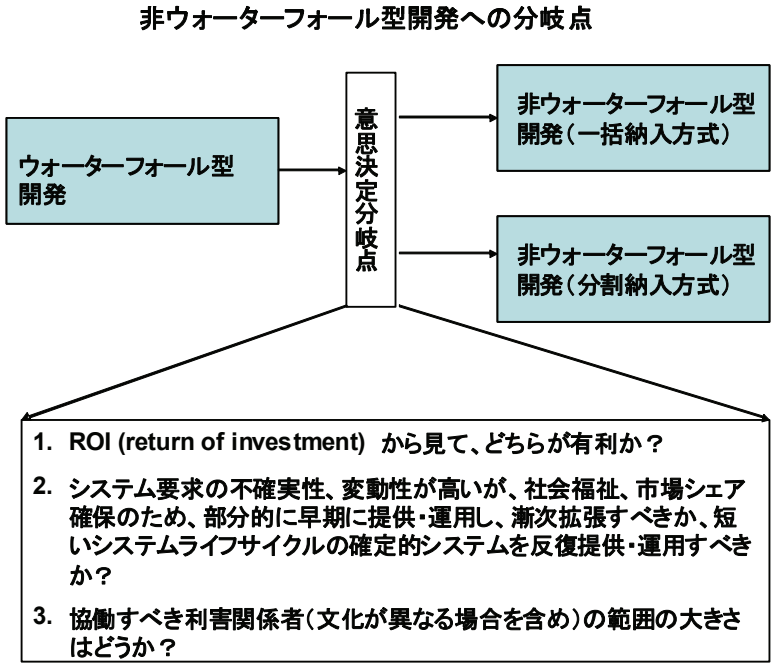
③基盤部品を使って基盤を組み立てる方式：バックログに入力されたユーザーストーリーに従って、基盤部品（標準、公開、または内作）を使って、基盤を組み立て直す。

７．２　非ウォーターフォール型開発への移行

我が国における大部分のソフトウェア開発は、情報処理推進機構が編纂する「ソフトウェア開発データ白書」が示すごとくウォーターフォール型（V モデル）に従って行われている。非ウォーターフォール開発へ移行するための意思決定は、主に、次の示す因子に依存して形成される（図表 5-12 参照）。

（１）　経済的優位性による移行意思決定

システムの全ライフサイクルにおいて、開発されたシステムを運用することによって得るキャッシュフローを予測し、それから求めた正味現在価値（NPV：Net Present Value）と初期投資額を比較して、前者が後者より大きい開発方式を優位とする。キャッシュフローは、毎年度のキャッシュイン（税引き後利益＋減価償却費＋投資残存価値）からキャッシュアウトを差し引いた金額である。一括納入方式の場合のキャッシュアウトには、運用費、保守費、改善のための追加投資費用が含まれるだけであるが、分割納入方式の場合には、漸次拡張開発および継続的統合が必要とする経費が、キャッシュアウトに加わることになる。



図表 5-12　非ウォーターフォール型開発への分岐点

（2） 社会的ニーズによる移行意思決定

①社会福祉からの要請に応えるため、システムの全体像は不確定であるが、確定できる副システムから、逐次、部分的に早期に提供・運用し、漸次拡張すべきとされる場合に、分割納入方式の非ウォーターフォール型開発に移行する。

②国策的に、国際市場シェアの早期確保のため、短いシステムライフサイクルの確定的システムを反復提供・運用すべきであるとされる場合に、分割納入方式の非ウォーターフォール型開発に移行する。

（3） 利害関係者特性による移行意思決定

利害関係者の範囲が極めて広く、確定的でない、または開発・運用が進むに従って各時点における利害関係者およびその範囲が変動するような場合に、分割納入方式の非ウォーターフォール型開発に移行する。

８　アジャイル開発におけるエンジニアリング手法

（1）アジャイルプロジェクトマネジメント

アジャイルプロジェクトの形態を、プロセス視点で分けると、以下の２つになる。

・ 一括納入を条件としたプロジェクト

このプロジェクトは、配列｛反復開発・一括納入・据付け・運用・保守｝の中のプロセスを配列順に実行する。

・分割納入を条件としたプロジェクト

このプロジェクトは、配列｛漸次拡張開発・納入・継続的統合・運用・改善｝を反復実行し、配列の中のプロセスを配列順に実行する。

上の配列に含まれる「開発」は、図表 5-13 に示すとおり、下記のプロセスの連接によって構成される。

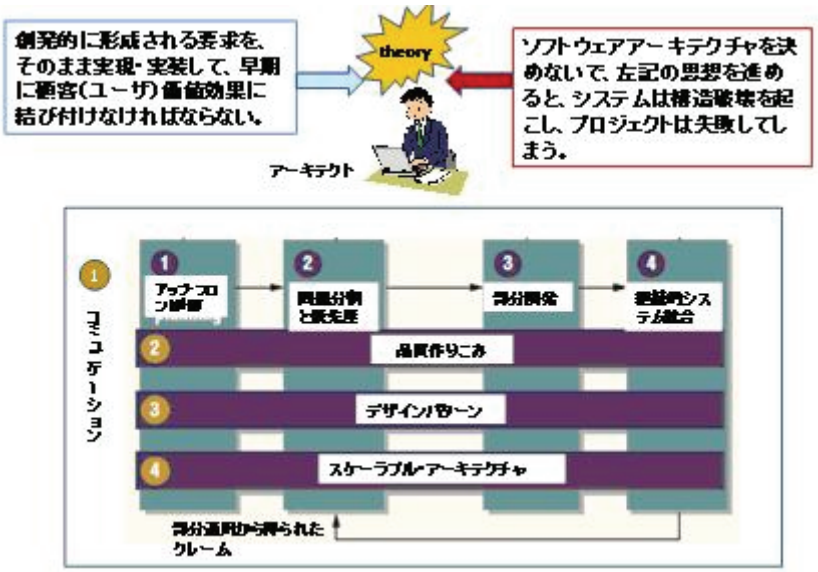
ｉ）アップ・フロント計画　(up-front planning)

ii）問題分割と優先度決定（ユーザストーリ策定とバックログ形成）

iii）部分開発（スプリント実行）

iv）継続的システム統合（実行可能ソフトウェア実装・テスティング）

v）リリース、または回帰



図表 5-13　反復開発プロセスモデルの例[15]

ⅰ）アップ・フロント計画

超上流エンタプライズ計画（広い利害関係者の関与）からの継続において、ソフトウェアライフサイクル全生涯の中における部分開発・継続的システム統合・リリース・運用・保守/改善・廃棄に対する「枠組み」を決定する。この「枠組み」には、図表 5-13 に示すとおり、品質作り込み計画、デザインパターン（記述言語を含む）利用計画、スケーラブルアーキテクチャ計画が含まれる。

ⅱ）問題分割と優先度決定（ユーザストーリ策定とバックログ形成）

アップ・フロントで識別された問題に対する優先度を決定し、ユーザーストーリー（user stories)を定義し、リリース計画を立て、バックログ・スタックに入れる。

ⅲ）部分開発（スプリント実行）

バックログ・スタックから取り出した優先度の高いユーザーストーリーに関して、実現・実装計画を立て、スプリントに対する入出力仕様および開発手法（たとえば、テスト駆動開発など）を定義し、時間軸上に配置し、スプリントを実行する。

ⅳ）継続的システム統合（実行可能ソフトウェア実装・テスティング）

スプリントからの出力を、出力仕様に基づいて検証し、継続的システム統合（continuous system integration）を行い、既存システムとの統合後のシステムに対する妥当性確認（validation）を行う。

ⅴ）リリース、または回帰

この結果が妥当である場合には、リリース・納入へ向かい、妥当でないと判定された場合には、その原因を究明するプロセスに回帰し、それ以降のプロセスを反復実行する。

（2）アジャイルソフトウェア技術者の教育・訓練

複雑なユーザ・エクスペリエンス（たとえば、ウェブ上での複数企業間での取引・折衝）を必要としない単純な小規模ウェブアプリケーションにおけるアジャイル開発においては、スプリントごとに、運用中のシステムをすべて作り変える（仕様書・コード再利用は一切行わない）プラクティスが一般化している。このような事例は、この項目の対象にはしていない。

・プログラミング教育

アジャイル開発では、ユーザーストーリーを、スクリプト型プログラミング言語（またはドメイン特化型言語）によって、直接、記述する場合が多い。このような場合、品質確保、保守・改善容易性の視点から、とくにコーディングスタイル、注釈づけ、コーディングパターン、プログラムおよびデータベース・リファクタリングに対する教育・訓練を行う必要がある。

アジャイルプロジェクトでは、図表 5-13 で示したとおり、アジャイル特有のプロセス構成が行われ、アップ・フロント計画手法、ユーザーストーリーの仕様化、品質作り込み、

設計・コーディング、検証、継続的システム統合などにおいて、アジャイル特有のマネジメント知識、実践経験、V&V[5]が必要となる。したがって、プロジェクトマネージャおよびプロジェクト・ファシリテータの教育・育成が必要となる。

（3）ユーザーストーリー間の相互依存性と継続的統合の問題

アジャイルソフトウェア開発手法では、Enhancement Agility を達成するために、ユーザにとって価値のあるストーリーを、順次、継続的に実装することを求める。しかし、この手法は、相互依存性分析に対する不適切な取組みを行った場合、これから派生する数々の問題を抱えることになる。ユーザーストーリーは、互いに相互依存性をもっているに拘わらず、互いに分離（isolate）されたものとして扱われた場合、問題が起きる。より価値にあるストーリーから順に処理していくという「greedy algorithm」という手法があるが、この提唱者でさえ、折に触れ、より高い価値のあるストーリーは、より低い価値をもったストーリ（より後の段階でバックログに入る）に依存して形成される、と言っている。問題を起こさないようにするためには、ユーザが求める価値を最適化するために役立つ、なんらかの先読みが必要になると考えられる（アジャイル手法－YAGNI では、先読みは禁じられている）。

ストーリーは、ストーリー相互だけではなく、システムを構成するアーキテクチャ構成要素に対しても、同じような相互依存性を有する。これら相互依存性は、領域安定性や技術的成熟性とは無関係に存在し、システムが初期開発段階にあるか、既に実装されて何年か実運用されているか、などのことに関係なく、存在する。アーキテクチャに関する相互依存性を識別・分析し、その知識を開発モデルに反映させる能力は、Architectural Agility と呼ばれる。Architectural Agility なしでは、Enhancement Agility は、維持できない。技術者の高度な経験・資質、およびすぐれた継続的統合環境の開発に期待するしかない。

（4）アーキテクチャの意思決定遅延と予測リスク負担の問題

Architectural Agility [16]という概念は、ウォーターフォールおよびアジャイルライフサイクルモデル、両者が内蔵する欠陥（shortcomings）へ向けた呼びかけである。Architectural Agility は、アーキテクチャ開発を、「ジャスト・イン・タイム」モデルに従って行えるようにする。Architectural Agility は、顧客がすぐにでも手に入れたいとしている特性（顧客直面特性とよぶ）を遅滞なく提供し、要求全体の形成、設計およびレビューの完成は、懸案としてあとに残す。同時に、Architectural Agility は、その後、創発的に生まれる顧客直面特性を支持しながら、安定に、一貫性をもって、継続的にアーキテクチャを進化させようとする。

ユーザーストーリーに近視眼的に取り組むと、それが度重なることによって却って複雑度を増し、アーキテクチャが想定していなかった特性まで取り込もうとする開発者の足掻きをもたらし、実現結果に捻じれを起こすという落とし穴（プロジェクト崩壊、最初から

[5] Verification（検証）& Validation（妥当性確認）。Verification（検証）は、各開発工程が適切に実行されているかどうかをチェックすることであり、Validation（妥当性確認）は、ソフトウェアが要求事項に従っているかどうかを確認するためにソフトウェアを評価することである。

のやり直し）に落ち込むことになる。

Architectural Agility を代表するスローガンは、「正確な情報に基づいた予測(informed anticipation)」である。創発するニーズを過度に予測することは、不必要に複雑化され、不必要なアーキテクチャ部分を組み込むことによって、ユーザ価値の提供を遅らせ、開発を危機に導くことになりかねない。反面、未来ニーズを過小に予測した場合には、アーキテクチャ的な道筋が作られていないことによって、新しい特徴の実現が難しくなるリスクを招く。

したがって、Architectural Agility は、「just enough」予測を必要とし、「just enough」であるためには、予測に必要な正確な情報が「informed」されなければならない。Architectural Agility は、この目的を達成するため、「リアルオプション分析（real options analysis)」、「技術的負債管理（technical debt management)」というツールを用いる。リアルオプションとは、不確実な未来を見通して経営者が選択する経営方針などを、それが未来に生みだすキャッシュフロー値に換算して定量化した予測価値（無形資産）のことである。また、技術的負債は、不適切な技術者の選択によって、生起する未来負担のことである。

９　関連分野におけるアジャイル開発への対応

ソフトウェア開発に関しては、ソフトウェアが世に出て以来ずっと各時代に応じた手法の体系化が試みられてきている。また、ソフトウェア開発に関連する種々の知識体系も作られている。ここでは、それら体系において、アジャイルがどのように取り扱われているかを紹介する。

９．１　CMMI とアジャイル開発の最近の動向

（1）SEI（Software Engineering Institute）のレポート

米国カーネギーメロン大学ソフトウェア・エンジニアリング研究所（SEI）は、2008 年 11 月に「CMMI[6] or Agile: Why Not Embrace Both! [17]（「CMMIかアジャイルか：両者を採用してみては！」）のレポートを発表した。

このレポートの中で SEI は、CMMI とアジャイル開発の歴史を振り返りながら、CMMI とアジャイル開発のアイデアやプラクティスの統合をどう進めるべきかについて提案している[18]。

SEI が勧める CMMI、アジャイル開発の賛同者双方の行動は以下の通り。

a）それぞれのパラダイムの価値を認める。
b）よくある誤解に抵抗する。
c）　どんなコンテキストのとき、何をすることが効果的であるのかを、試し、学び、レポートし続ける。

また、このレポートでは、次のような問題を扱っている。

①アジャイル手法の起源
②CMMI の起源
③正確な情報の欠如
④専門用語の難しさ
⑤CMMI、アジャイル開発それぞれのパラダイムの価値
⑥アジャイル適用時の課題
⑦CMMI 適用時の課題
⑧CMMI でもアジャイルでも解決できない問題

（2）CMMI バージョン 1.3.へのアジャイル開発の反映

2010 年 10 月 27 日リリースした CMMI バージョン 1.3 では、アジャイルに関する導入ガイドラインと導入における注意に加え、どのようにアジャイル・プラクティスを解釈すべきかが含まれている。また、スクラム、XP 等で使われている、ユーザーストーリー、プロダクトバックログ、ストーリーカード、ペアプログラミング、日次（頻繁な）ビルド、ふりかえり等のプラクティスにも言及されている[19]。

[6] CMMI:Capability Maturity Model Integration

９．２　PMBOK とアジャイル開発の最近の動向

（１）Agile Community of Practice

2009 年、米国 PMI（Project Management Institute）は、アジャイルと PMBOK を関連付け、PMI メンバがアジャイルの知識と技術を身に付けることを目的として、米国 PMI の中に Agile Community of Practice を設立した[20] 。

（２）PMBOK のアジャイル開発への適応

アジャイル開発は、プロジェクト開始時に「スコープ」「コスト」「タイム」の目標値が必ずしも決められておらず、設計や開発を反復しながら最終形を決めていくのが通常である。これに反して PMBOK は、プロジェクト開始時に「スコープ」「コスト」「タイム」等が決まっているのが前提となっている。そのため、PMBOK とアジャイル開発では考え方に大きな違いがあるといわれている。また、アジャイル開発は、プロジェクトマネジメントより、プロジェクトファシリテーションを重視する。

しかし、アジャイル開発においてもある規模以上のプロジェクトになれば、プロジェクトを何らかの方法で管理しなければならないのは事実であり、PMBOK などをベースにアジャイル手法にあったプロジェクト管理のベストプラクティスを作成することについて、今後もっと検討を深める必要がある。

（３）2011 年 2 月、PMI は Agile 認証パイロットを 5 月から開始すると発表した。

プロジェクトマネジメントにおけるアジャイル開発のマネジメントは、日々重要さを増しており、多くのプロジェクトマネジメントの専門家が仕事に使えるアジャイル技術を習得したいと熱望していた。プロジェクト経営環境におけるこのような変化への対応のため、PMI は Agile 認証パイロットの導入を決定した。本格的な PMI Agile 認証試験は、2011 年第３四半期にリリースされる予定となっている[21]。

９．３　BABOK とアジャイル開発の最近の動向

（１）IIBA が BABOK アジャイル拡張版を発表

IIBA(International Institute for Business Analysis)は、2010 年 8 月 10 日、アジャイルカンファレンス 2010 で、BABOK (Business Analysis Body of Knowledge)のアジャイル拡張版のドラフト版を発表した。

IIBA では、ビジネスアナリストを「様々なビジネスプロセス、ポリシーそして情報システムへの変更要求を引き出し、分析し、コミュニケーションをとり、そして妥当性確認をするためにステークホルダー間をつなぐ役目を果たす」と定義しており、「ビジネスアナリストとは、時にまとめ役であり、聞き役であり、調査者であり、ファシリテーターであり、コミュニケーターであり、協力者であり、革新者であり、さらにそれ以上の役割を持っている」と述べている[22]。ビジネスアナリストのこの役割は、アジャイル開発においても重要である。

（２）BABOK アジャイル拡張ドラフト版の目的と目次[23]

①BABOK アジャイル拡張版の目的

- ビジネスアナリシスのアジャイル・プラクティスへの適用
- アジャイル開発者のためのビジネスアナリシス概要
- アジャイルプロジェクトにおけるビジネスアナリシスのためのプラクティスの定義
- ビジネスアナリシスのための新しい役割、技能、能力の概要

②BABOK アジャイル拡張版の目次

- 目的
- 序論
- アジャイル・プラクティスの出現
- ビジネスアナリシスの出現
- アジャイル・ビジネスアナリスシスの必要性
- ビジネスアナリスシスのためのアジャイル的思考
- ビジネスアナリスシスへのアジャイル宣言の適用
- BABOK へのアジャイル関連の拡張
- アジャイル開発者のためのビジネスアナリスシス
- ビジネスアナリスシスのためのアジャイル・プラクティス
- アジャイル開発におけるビジネスアナリスシス
- アジャイルプロジェクトのルール
- アジャイル・プラクティスと開発方法論の歴史

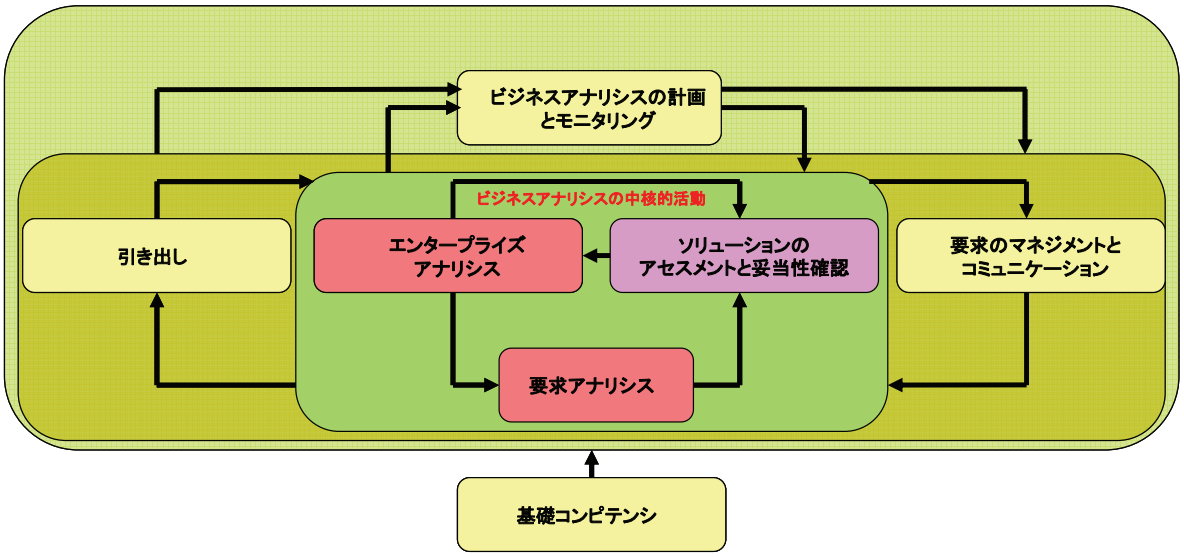
（3）BABOK とアジャイルとの関係

BABOK（最新版 Version2.0－BABOK アジャイル拡張ドラフト版も Version2.0 の範囲）では、ビジネスアナリシスを、「組織の構造とポリシーおよび業務運用についての理解を深め、組織の目的の達成に役立つソリューションを推奨するために、ステークホルダー間の橋渡しとなるタスクとテクニックの集合体」[24]と定義している。

また、BABOK ではビジネスアナリシスを、「なんらかのビジネス変革の目的で行うとしており、情報システムの変革はその手段の一つにすぎないという位置づけである」[23]。

BABOK では、ビジネスアナリシスを 7 つの知識エリアと 32 のタスクに定義している。

BABOK® V2.0 に出てくる 7 つの知識エリアを図表 5-14 に示す。



出典：ビジネスアナリシス知識体系ガイド(BABOK ガイド) Version2.0 日本語版

図表 5-14　BABOK　7 つの知識エリア

「エンタープライズアナリシス」は、経営層が想定しているビジネスの目的や目標から、それを実現するための実行戦略をビジネス要求としてまとめていくことを目的としている。「要求アナリシス」は、「エンタープライズアナリシス」からのビジネス要求にもとづいて、ソリューション要求をまとめることを目的とする。「ソリューションのアセスメントと妥当性確認」は、ソリューションがビジネス要求を満たすことを評価することで、ソリューションの妥当性を確認することを目的としている。

BABOK アジャイル拡張版は、このビジネスアナリシスの中核的活動である、「エンタープライズアナリシス」「要求アナリシス」「リューションのアセスメントと妥当性確認」を含めた知識エリア全体にアジャイル開発の考えを反映させている。

IIBA 日本支部では、BABOK Version2.0 アジャイル拡張版は、「要求のマネジメントと

コミュニケーション」の一部にプロダクトスコープを固定して要求をマネジメントするウォーターフォール型開発の考え方が一部残っていることなどから、BABOK Version2.0 を対象とした暫定版との位置づけと考えている。2011 年 4 月から本格的検討に着手する予定である BABOK Version3.0 に知識エリアの全体構造の再構想を含めたアジャル開発への対応を計画するとしている。

参考文献

[1] [Matsumoto09] 松本吉弘編,「ソフトウェア現場力ハンドブック」, オーム社, 2009
[2] [Boehm06] Boehm, B., A View of 20th and 21st Century Software Engineering, keynote address at ICSE 2006, May 25 (2006)
[3] [ISO10] ISO/IEC JTC1/SC7 N4585, Systems Engineering Handbook, NWIP, 2010-02-25
[4] [BKCASE10]Stevens Institute of Technology,System Engineering Body of Knowledge, V0.25, September 15 (2010)
[5] [INCOSE10] Forsberg, Mooz, and Cotterman 2005, Permission Pending
[6] IPA/SEC,「共通フレーム 2007 第 2 版」, IPA/SEC, 2009
[7] [Oppenheim09] Oppenheim, B. W., Lean Enabler for Systems Engineering, CrossTalk: The Journal of Defense Software Engineering (July/August 2009)
[8] [Ambler05] Ambler, S. W., The Agile System Development Life Cycle (SDLC), 2005, http://www.ambysoft.com/essays/agileLifecycle.html
[9] [Konboy2011] People Over Process: Key People Challenges in Agile Development, IEEE Software, March/April (2011)
[10] [Hanssen2010] Hanssen, G. K., et al., Software entropy in agile product evolution, Proceedings of the 43rd Hawaii International Conference on System Sciences (2010)
[11] Michael J.Sandel, Democracy's Discontent, Belknap Press of Harvard University Press, 1998, 訳: 金原恭子/小林 正弥,「民主政の不満―公共哲学を求めるアメリカ」, 勁草書房, 2010
[12] [ACM11] ACM queue, UX Design and Agile: A Natural Fit? , Communications of the ACM, January 2011 | vol. 54 | no. 1, pp.54-60
[13] Stevens Institute of Technology, System Engineering Body of Knowledge, V0.25, September 15 (2010)
[14] ISO/IEC JTC1/SC7 N4585, Systems Engineering Handbook, NWIP (2010-02-25)
[15] 松本吉弘,「非ウォーターフォール型ソフトウェア開発・保守における課題」, 情報処理学会大会, Software Japan 2010, March 11 (2010), 於東京大学
[16] Brown N., et al.,, Enabling Agility Through Architecture, The Journal of Defense Software Engineering, CrossTalk, Nov/Dec 2010, pp.12-17, U.S. DoD, Softwarfe Technology Support Center
[17] Carnegie Mellon University., CMMI or Agile: Why Not Embrace Both!, Carnegie Mellon University, 2008
[18] Manoel Pimentel, SEI publishes report integrating CMMI and Agile, 2008, http://www.infoq.com/news/2008/11/report-integrating-cmmi-agile
[19] Ben Linders, CMMI V1.3: Agile, 2010, http://www.benlinders.com/2010/cmmi-v1-3-agile/
[20] http://www.infoq.com/jp/news/2009/07/pmi-agile
http://agile-pm.pbworks.com/w/page/1526573/FrontPage
[21] http://www.pmi.org/Agile.aspx
[22] IIBA 日本支部 HP より引用
http://www.iiba-japan.org/about.php#aboutBA
[23] IIBA, Agile Extension to the Business Analysis Body of Knowledge, IIBA, 2010

[24] 宗 雅彦,「BABOK が定義する「ビジネスアナリシス」とは 」ITpro ,2010,
http://itpro.nikkeibp.co.jp/article/COLUMN/20100729/350789/?ST=upper&P=2

おわりに

激しさを増すビジネス環境の変化に伴い、情報システムの開発において、初期には全ての要求を固定できないのみならず、いったん決めた要求も短期間に変化するケースが増大している。そして、ビジネスに戦略的に活用されている情報システムに対して、この状況に迅速に対応することが一層求められるようになっている。

このような背景の中で、従来のウォーターフォール型のソフトウェア開発形態には限界があり、アジャイル開発等、「非ウォーターフォール型」の開発形態への期待が高まっている。しかしながら、非ウォーターフォール型開発形態に十分馴染んでいない人たちにとっては、非ウォーターフォール型開発に対して、プロジェクトのマネジメントやプロダクトの品質面等での不安も見られる。

本報告書は、このような不安の解消を狙い、非ウォーターフォール型開発形態についての整理された情報を提供している。特に、事例調査によって明らかとなった課題を中心にまとめられている。しかしながら、「人」に関わる課題については、本報告書では十分議論できていないので、ここで簡単に触れたい。

ソフトウェア開発の様々な局面において、個人や組織の経験に基づく『知』を拠りどころとする「選択」が行われる。すなわち、アーキテクチャ・パターンやひな型、関数、ライブラリ、部品、アルゴリズム、命令ステートメント等、開発の各フェーズでその段階に対応する構成品の選択によりソフトウェアを作り上げていく。多様な「人」が開発に携わる状況において、これらの選択が適切に行われるようにするために、暗黙知の形式知化に力が注がれている。具体的には、選択のプロセスの標準化・自動化である。

高度に自動化された製造ラインの下で標準化された手順に基づいて作業が行われる家電や車等の生産とは異なり、ソフトウェア開発では、一般に、選択の自由度が高く、プロセスは柔軟である。その意味では、ソフトウェア開発は極めて創造的な作業であると言える。反面、選択に要する時間が生産性に影響し、各選択の適切性がプロダクトの品質を大きく左右する。この影響の程度は、開発に携わる者のスキルに依存する。かといって、スキルの高いメンバのみによるチームビルディングは事実上不可能である。したがって、生産性を高め、所定の品質を確保するためには、自由度と柔軟性を低くする方向、すなわち、標準化や自動化を進めることになる。特に、標準化は、プロダクトの品質を一定値に保つために行われる。これに伴い、創造性は相対的に低下する。ウォーターフォール型の開発形態はこの方向を指向していると言える。

さて、プロダクトが予定通り完成した後、人は、自身の選択が適切であったことと、決められた手順通りに完璧に作業を行ったこととの、どちらの方に達成感や幸福感をより多く感じるであろうか？　逆に、プロダクト生産やその関連ビジネスが失敗に終わった時に、その開発に関わった人の反応、すなわち、喪失感、疲弊感、新たな闘志の大きさ、あるいはこのような反応自体の強さは、どうであろうか？

もちろん、このような感情は人により様々である。想像力を発揮することが得意な人や、手順に従ってきっちりと作業することが得意な人など、さまざまである。以前から言われ

ている、ソフトウェア開発者（特に若者）の参画意識・達成感が低いという問題は、人のスキル（想像力発揮の得手不得手）と果たすべき役割（創造性の有無）との不整合に起因する。この点に関しては、多重下請構造そのものが本質的な問題ではない。想像力を発揮することが得意な人あるいはそのことに幸福を見いだせる人に、そのような機会が与えられることが重要である。アジャイル型の開発形態はこの方向を指向していると言える。

ウォーターフォール型は『プロセス』を、アジャイル型は『人』を、それぞれ重視する開発形態と言われる。上述の分析に当てはめれば、前者は、標準化を追求し、自動化を進めやすい形態である。一方、後者は、自由度と柔軟性が高い形態である。このような特徴から、両形態では『文化』が異なると言っても過言ではない。そして、両文化を担う人たちの間の相互理解は、必ずしも十分に進んでいない面も見られる。しかし、最近では、分散開発環境に対するアジャイル開発の適用事例（イノベーションスプリント 2011）や、ウォーターフォール型テストケースへの整合のためのテスト駆動開発(TDD)の拡張提案（ソフトウェアテストシンポジウム 2011）など、さまざまな工夫も試みられている。互いに相手の文化の優れたところを自分たちの仕組みなどに採り入れようとする、異文化コラボレーションの動きが見られる。わが国がこのような取組みを得意とすることは、その歴史が物語っている。

変化の激しさが増すビジネス環境に対応するために、ソフトウェア開発には、生産性と品質を維持しつつも、一層の柔軟性が求められるようになっている。この要求には、既存の開発形態の改善（インプルーブメント）だけでは十分に応えられず、現状を大きく越えた新しいステップアップ、すなわち、イノベーションが必要である。既に言い古されている感はあるが、2004 年に報告された「イノベート・アメリカ（通称：パルミサーノ・レポート）」は、イノベーションは、異質の文化が交わり接するところで、それらの間のコミュニケーションと協働によりもたらされるもの、というようなことを述べている。今後はウォーターフォール型開発とアジャイルに代表される非ウォーターフォール型開発の両文化のコラボレーションによってもたらされるであろう『プロセス・イノベーション』に期待したい。本報告書がその活動のための端緒となれば幸いである。

付録　非ウォーターフォール型開発に関する検討委員

非ウォーターフォール型開発WG		
	氏名	所属
主査	松本　吉弘	財団法人京都高度技術研究所 顧問
委員	稲村　直穂子	株式会社ディー・エヌ・エー システム統括本部 本部長
	大槻　繁	株式会社一 コンサルティンググループ 副社長
	合田　治彦	富士通株式会社 システム生産技術本部 本部長代理
	田澤　久	楽天株式会社 開発ユニット 開発環境整備課 課長
	戸村　元久	株式会社NTTデータ 技術開発本部 プロジェクトマネジメント・イノベーションセンタ センタ長
	羽生田　栄一	株式会社豆蔵 取締役
	平鍋　健児	株式会社永和システムマネジメント 副社長、 株式会社チェンジビジョン 代表取締役
	広瀬　敏久	日本電気株式会社 主席技術主幹
	古川　正伸	株式会社東京証券取引所 品質管理部 課長
	前川　徹	サイバー大学 IT総合学部 教授
	馬嶋　宏	株式会社日立製作所 情報システム事業部 方式設計センタ
	松島　桂樹	武蔵大学 経済学部 教授
	南　悦郎	新日鉄ソリューションズ株式会社 技術本部システム研究開発センター所長
	和田　憲明	富士通株式会社 システム生産技術本部 SI生産革新統括部 SDEM推進部
エキスパート	伊久美　功一	IPA/SEC専門委員／元IPA/SEC研究員
オブザーバ	鴨田　浩明	経済産業省

開発モデルPT		
	氏名	所属
リーダ	平鍋　健児	株式会社永和システムマネジメント 副社長、 株式会社チェンジビジョン 代表取締役
委員	合田　治彦	富士通株式会社 システム生産技術本部 本部長代理
	戸村　元久	株式会社NTTデータ 技術開発本部 プロジェクトマネジメント・イノベーションセンタ センタ長
	広瀬　敏久	日本電気株式会社 主席技術主幹
	古川　正伸	株式会社東京証券取引所 品質管理部 課長
	前川　徹	サイバー大学 IT総合学部 教授
	松島　桂樹	武蔵大学 経済学部 教授
	南　悦郎	新日鉄ソリューションズ株式会社 技術本部システム研究開発センター所長
	和田　憲明	富士通株式会社 システム生産技術本部 SI生産革新統括部 SDEM推進部
エキスパート	伊久美　功一	IPA/SEC専門委員／元IPA/SEC研究員

技術・スキルPT		
	氏名	所属
リーダ	松島　桂樹	武蔵大学 経済学部 教授
委員	天野　勝	株式会社永和システムマネジメント コンサルティングセンター センター長
	川端　光義	アジャイルウェア代表
	児玉　公信	株式会社情報システム総研 取締役副社長／モデラー
	田澤　久	楽天株式会社 開発ユニット 開発環境整備課 課長
	馬嶋　宏	株式会社日立製作所 情報システム事業部 方式設計センタ
	和田　憲明	富士通株式会社 システム生産技術本部 SI生産革新統括部 SDEM推進部

契約問題PT		
	氏名	所属
リーダ	前川　徹	サイバー大学 IT総合学部 教授
委員	飯塚　顕治	新日鉄ソリューションズ株式会社 法務・知財財産部 法務グループシニアマネージャー
	高橋　雅宏	アジャイルプロセス協議会 見積・契約ワーキンググループ
	板東　直樹	アップデートテクノロジー株式会社 代表取締役社長
	平野　高志	ブレークモア法律事務所 弁護士
	古川　正伸	株式会社東京証券取引所 品質管理部 課長
エキスパート	梅本　大祐	IPA/SEC専門委員／ブレークモア法律事務所 弁護士
オブザーバ	鴨田　浩明	経済産業省
	葛山　弘揮	経済産業省
	下田　裕和	経済産業省
	柳橋　祥人	経済産業省
	新保　康夫	日本コンピューター・システム株式会社 事業推進本部
	鈴木　律郎	社団法人情報サービス産業協会 企画調査部
	茂木　智美	社団法人情報サービス産業協会 企画調査部／審査業務部

IPA/SEC		
	氏名	所属
	松田　晃一	
	山下　博之	
	柏木　雅之	
	上野　博英	
	庄司　祐子	