非ウォーターフォール型開発に関する調査

研究会報告書

平成 22 年 3 月 30 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

研究会報告書目次

１ 非ウォーターフォール型開発研究会の目的

非ウォーターフォール型開発研究会は、平成 21 年 11 月に、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（IPA/SEC）によって組織され、主としてエンタープライズ系ソフトウェアにおける、ニーズ発芽から、契約、開発、実装、据付け、運用、成長／成熟／衰退から廃棄に至る全ソフトウェアライフサイクルを対象として、IPA/SEC より非ウォーターフォール型開発調査事業を受託した株式会社三菱総合研究所と連携して、次の目的に従う研究を行った。

(1) 非ウォーターフォール型開発・保守に関する動向・現状の把握と整理、および
(2) 非ウォーターフォール型開発・保守を現場へ適用・普及させるために必要な要諦と整理

２ 動向・現状の把握と整理

２．１ 研究会における事例紹介

本研究会では、第 1 回の会合から第 3 回の会合にかけて、研究会委員が実際に取り組まれた非ウォーターフォール型開発事例を中心とする発表を行うとともに、発表内容について議論することにより、動向・現状の把握と整理を行った。以下に、各回における発表内容の概要および議論内容の概要を示す。

２．１．１ 第 1 回研究会（2009 年 11 月 16 日開催）における事例紹介

(1) 永和システムマネジメントにおけるアジャイル開発の現状（平鍋　委員）

海外および国内でのアジャイル適用状況とその事例や、永和システムマネジメントにおけるアジャイル適用状況についての紹介があった。

以下に、紹介された内容の概要を示す。

- 米国では、日本のように SIer に開発を丸投げするというスタイルを取っていない。また、コンサルタント、コーチ（現場に入って人を指導）が多く、導入支援が行われている。
- VersionOne 社が、Web による全世界のアジャイルユーザを対象にしたアンケートを実施している。
- アジャイル開発の特徴として、リスクを初期に下げる、可視性を常に高く保つ、変化への対応性をキープする、といったことが挙げられる。
- 日本のアジャイル市場の特徴としては、小規模が多く、ユーザ企業主導にはなっていない。
- 永和システムマネジメントでは、3 年で約 30 プロジェクトをアジャイル開発した実績を持つ。開発メンバは 1 チームあたり 2-3 名である。

発表に対する質疑として以下のものがあった。

Q：アジャイル型の開発が業務システムにも少しずつ広まっていると考えてよろしいか。
A：そのように思う。
Q：アジャイル型開発の方がウォーターフォール型開発よりも品質が高くなるのか。
A：エンジニアリングプラクティスの 1 つにテスト駆動開発があり、コードとテストを両方同時に作っていく。できた成果物は、動くコードと動くテストがほぼ同じソース行数程度にできていく。つまり、動いていることを確かめられる状態で納品される。その意味では、品質が担保されると言える。

Q：障壁の中で一番大きいものは何か。
A：マネジメントとカルチャーに関するものが大きいだろう。

(2) DeNA での非ウォーターフォール型開発取り組みについて（稲村　委員）

携帯電話向けソーシャルゲーム開発に非ウォーターフォール型開発を適用した事例が紹介された。

以下に、紹介された内容の概要を示す。

- 携帯電話向けの開発は、ほとんどが 100 人日以内の規模である。また、スピード・機動力がポイントであり、ユーザビリティが生命線である。
- 基本的にはエンジニア 1 名、企画 1 名で開発を行った。
- 企画スタートからサービスインまで約 3 か月であった。
- 4 回以上のプロトタイピング開発およびフィードバックを行い、納得いくまでサービス品質を高めた。実装が淘汰され、メンテナンス性の高いコードになる。
- 1 人で開発を行うため、開発ドキュメント作成コストがほぼ 0 であった。

発表に対する質疑として以下のものがあった。

Q：テストは外注しているのか。
A：ソーシャルゲームやそれ以外の Web の機能に関しては、社内でテストを行っている。企画を考えた人など上流工程を行った人がテストを行っている。品質の問題もあるので、現場でどうしてもテストが行えない場合は、QA 部隊がテストを行っている。
Q：ビジネスの成長モデルでアジャイルはどういうステップとなっているのか。
A：2、3 年間は 2、3 人で開発を行っていたが、5 人以上になると 1 人 1 人に担わせる範囲や責任など教育をしないと失敗する。その場合は計画をして、多少時間が掛かっても要所でウォーターフォール型の開発を行うなど、状況を考えながら開発を行った。
Q：アジャイル開発を採用した時のニーズとは何か。
A:従来型開発では機能作成に3人月かかることに問題意識があり、少人数で開発したところ、うまく開発できた。その成功体験をもとにアジャイル型開発を実施している。
Q：利用しているフレームワークは自社開発なのか。
A：自社開発であり、オープンソース化している。

(3) 非ウォーターフォール型開発に関する富士通の取組みと課題（合田　委員）

富士通におけるアジャイル開発の実態およびアジャイル開発への取組みの傾向について報告された。

以下に、紹介された内容の概要を示す。

- 2000 年以降、社内で発表されたアジャイル開発事例として、15 件の概要について紹介された。
- 大規模プロジェクトでは、コンカレント開発、イテラティブ開発、ラピッドプロトタイピングが実施されている。
- 自らが仕様を決められる領域での適用が中心である。
- 中小規模への適用が中心である。
- カスタムソフトウェアの開発にアジャイル的なアプローチを適用する際の主な課題として、契約の問題、定量化技術の整備などが挙げられる。

発表に対する質疑として以下のものがあった。

Q:アプリケーションの保守を行うための工夫は何か行っているか。
A:全体を俯瞰できるドキュメントの整備は行っている。ERD のようなデータに着目した構造の設計情報や機能的な配置や構成に係る情報、ソースコードそのものを格納しているコンフィグレーション情報など、紙でしか押さえられないものは紙の形で残している。
Q:顧客からアジャイル開発を望まれるケースはあるのか。
A:正確には把握していないが、最近そのような傾向は強まっている。
Q:課題として何が残っているか。
A:契約の問題がある。契約は請負契約であったが、請負計画の実体はなく、派遣そのものに近かった。

2.1.2 第 2 回研究会（2009 年 12 月 22 日開催）における事例紹介

(1) ウォーターフォールは間違いだ！（前川　委員）

次の事項について紹介があった。

- ウォーターフォール型開発に関する課題について、提唱者といわれる Royce 博士が示した５つのリスクの回避方法
- アジャイル的な開発の拡大状況とともに、良品計画におけるマーチャンダイジングシステムの具体的な取組み
- アジャイル開発における契約の問題

  ウォーターフォール型について、本来活動が示されているだけで、時間の概念は入っていないところが、解釈が変わってしまった所などの議論がなされた。また、契約に関して、様々な契約方式が紹介されるとともに、特に、契約履行上係争となったときに裁判所が判断できるものでないと困る、何を作るかがはっきりしていないと判断ができないこと等、アジャイル的な開発での論点に関する議論があった。

(2) 楽天の開発について（田澤　委員）

楽天におけるシステム開発についての紹介があった。

- 背景として、ビジネスの状況、システムの企画から開発、リリースまでのシステム開発部の役割の紹介
- 現状課題として、工数と工期の関係の結果等が示され、ウォーターフォール的な取組みでの課題とアジャイル開発への取組みとして継続的インテグレーション（CI）環境の活用の紹介
- また、施策として、標準ツールの提供（プロジェクト管理ツール、コストレポート）、見える化（プロジェクト単位で「進捗・工数・コスト」予実情報の見える化）、報告の場を提供（プロジェクト進捗報告会）についての紹介

(3) 非ウォーターフォール型開発に関する NEC の取組み（広瀬　委員）

NEC における取組みについて、次の事項の紹介があった。

- SI プロジェクト全体状況、特に、標準化推進活動（NEC における共通フレームへの取組み）
- アジャイル開発事例の状況と、現状課題、課題認識について、6 例の概要とともに紹介があった。また、課題として、①標準化の課題、②組織・風土・仕組の課題、③契約上の課題、④技術的な課題等が示された。

・　ソフトウェア開発における課題認識として、特徴に応じた開発パターン分類（市場創出試行型、成果定義型、成長型）が示されるとともに、それぞれの特徴、ポイントと課題の紹介があった。
・　現場での活動事例として、ソフトウェア開発現場での草の根改善活動、P&P（プロセス&プラクティス）のマップ（共通フレームへのマッピング）が紹介された。

(4) アジャイルプロセス協議会の挑戦：なんちゃってアジャイルの先にあるもの（大槻委員）

アジャイルプロセス協議会の取組み、ソフトウェアセル生産、知働化について紹介があった。

・　アジャイルプロセスおよびアジャイルプロセス協議会の歴史について紹介された。
・　アジャイル・ソフトウェアセル生産方式の概要が紹介された。
・　人働説から知働説への転回の必要性について示された。
・　複数のプロジェクトについて、チーム外から見た特性とチーム内から見た特性の軸を用いて分析を行った。

事例発表に対する質疑として以下のものがあった。

Q：軸はオリジナルであるか。
A：私が考えた軸である。
Q：具体的にセルをどう定義するのか。
A：各社各様で行われる。シェフと言われる人や社長等、リーダー的な人が経験から行う。
Q：何をやるか全貌がわからないときにスタートするのがアジャイル。そういう状態でセルはあらかじめ用意できるのか。
A：用意はできない。わかる時点で作っていくものである。

２.１.３ 第３回研究会（2010 年１月 19 日開催）における事例紹介

(1) デンマーク事例紹介「It-contracts - agile methods」（平鍋　委員）

2009 年 12 月にコペンハーゲンで開催された Agile ’09 に参加した平鍋委員から、Agile ‘09 での Agile 開発向けの標準契約書に関する講演の概要が事例として紹介された。

以下に、紹介された内容の概要を示す。

・ 講演の発表者は、コペンハーゲンの弁護士である。
・ デンマークでは、ウォーターフォール型のソフトウェア開発用の標準契約書としてタイプ K と呼ばれるものがあったが、新たに Agile 開発を対象とした標準契約書が作られた。
・ タイプ K の契約書は、発注側と受注側に信頼関係があることと、受注側が契約の遂行意思を持っていることを前提として作られている。
・ 一方、アジャイル向けの契約は、契約実施期間中に双方が協力し合うことが前提となっている。
・ アジャイル向けの契約は、最終結果を決定するものではなく、プロジェクト管理と実施プロセスを規定するものとして位置づけている。

事例発表に対する質疑として以下のものがあった。

Q：既に具体的な契約事例はあるのか。
A：実例はまだない。政府の中で採択されたかどうかも聞いていない。

Q：この標準契約書を推進しているのはこの法律事務所の人か。
A：推進しているのか、作成を依頼されているのかはわからない。
Q：対象は、アジャイル開発というよりはイテラティブ開発ではないか。
A：そのとおりである。

(2) リスク管理システムの開発の事例（南　委員）

リスク管理システムの開発に反復的なインクリメンタル開発を適用した事例について報告された。

以下に、紹介された内容の概要を示す。

- 1994 年に開発開始し、1996 年にサービスインした。その後、現在も管理対象の増加に伴い、機能の拡張を行っている。
- 開発当初はオブジェクト指向開発・スパイラル型開発を標榜し、具体的な仕様を定めずに開発を開始した。
- 最初の反復で開発システムに機能面・性能面での問題があることが判明し、開発方針の見直しを行った。
- 依存関係に注目して、階層化とモジュール化を徹底的に行うことにより、標準化を実施した。
- 依存関係（利用関係）を元にリリース順序を制御、2 週間から 1 ヶ月を単位に継続的にリリースを行った。
- 結果的にサービスインは半年遅れたものの、現在に至るまで継続して利用されるシステムとなった。

事例発表に対する質疑として以下のものがあった。

Q：ドメインモデルの考え方は、事例によって捉え方が違うと思う。具体的にはどうやっているのか。
A：画面の都合によらない業務ロジックをドメインモデルと定義している。
Q：上位層からフレームワークで順番にやったということであるが、下位層からのフィードバックはどう行っていたのか。
A：吸収できるものはなるべくアプリケーション側で吸収した。ドメイン層までに関わってきたらドメインモデルで吸収する。
Q：契約は最後まで準委任であったのか。
A：そのとおりである。
Q：顧客側のシステム開発に対する知識や進め方の熟練度などはこのプロジェクトではどうであったのか。任せっきりだったのか、詳しい人がいたのか。
A：詳しい人が、システム部門にもユーザ部門にもいた。

(3) 豆蔵での適用事例（羽生田　委員）

豆蔵における非ウォーターフォール型開発の事例が 3 件紹介された。3 件の内 2 件はアジャイル型の開発であり、1 件はユニファイドプロセスをベースとして作られた豆蔵プロセスによるものであった。

以下に、紹介された内容の概要を示す。

- 最近の開発プロジェクトは、規模に比べて短期開発である、要求仕様が複雑である、開発初期段階ですべての要求が出せない、等の傾向が見られる。

・ このため、反復的に開発を行いこまめにリリースすることによって、リスクを軽減する必要がある。
・ 豆蔵プロセスは、方向付け・推敲フェーズの部分で、アーキテクチャのベースラインを探索的に定義する。作成・移行フェーズでは、反復的に開発を実施する。
・ 方向付けで決まったプロジェクトゴールがぶれない範囲内で、変更・追加を許容している。
・ アーキテクチャベースラインを決定する部分では優秀なアーキテクト集団を確保する必要があるが、後半はオフショア開発も可能である。
・ 推敲フェーズは、準委任（コンサルティング契約）での実施が基本である。
・ 反復型開発では、既開発分の回帰テストが必要となるため、プロジェクトの初期段階からテスト自動化を考えておく必要がある。

事例発表に対する質疑として以下のものがあった。

Q：ユニファイドプロセスはリスクを見極めて、最初にリスクを回避するという説明だったが、開発のスピードを早めるというのが主眼ではないのか。
A：後半には、生産性を上げることも目的の一つである。
Q：アーキテクチャをしっかり作っているから後半で生産性が上がるということか。
A：その通りである。ただし、決定したアーキテクチャとは違う方向には行けない。
Q：組込みシステムの場合、部品等の不具合により設計を変えなければならなくなるのではないか。
A：その場合は、対応せざるを得ない。

(4) グローバルソフトウェア製品開発における事例と考察（馬嶋　委員）

日立におけるグローバルな系列会社向けのシステム管理ソフトウェアの開発をアジャイル型で実施した 3 件の事例が報告された。

以下に、紹介された内容の概要を示す。

・ 米国を中心としたグローバルなソフトウェア開発は、国内の開発スタイルと大きく異なっており、特に開発スピードの速さが求められる。
・ このため、グローバルな企業向けのソフトウェア開発に対応するために、アジャイル型開発を実施した。
・ ユーザインタフェースの設計について、UX（User Experience）を追求したいというニーズがあった事例では、動作するプログラムを頻繁に提供する必要があったため、GUI の開発部分をアジャイル（Scrum）形で実施した。

事例発表に対する質疑として以下のものがあった。

Q：開発システムのセマンティクスがわかっている人がコードを書くというのは生産性を高める基本だと思うが。
A：その通りだと思う。優秀な人を集めて成功するなら集めればいい。結局は開発者に依存する問題だと思う。
Q：セマンティクスをわかっている人は、ユーザ側にいるべきではないか。
A：プロダクトマネジャは大きいユーザに引きずられる傾向もあるので、プロダクトマネジャとソフトウェア開発は共同的な体制を組む必要がある。

２．２ 研究会のメーリングリストにおける議論

本研究会では、研究会委員をメンバとするメーリングリストを設置し、メールによる意見表明も活発に行った。メーリングリストにおいて議論された論点を以下に示す。なお、議論内容については、本報告書の付録 C に掲載する。

表 1　メーリングリストにおいて議論された論点

1	非ウォーターフォール型開発手法の体系化
	(ア)ウォーターフォール型開発の課題
	(イ)アジャイルの特徴
	(ウ)アジャイル開発のライフサイクルモデル
	(エ)アジャイルにおける支援システム・ツール
	(オ)アジャイルコミュニティ
	(カ)アジャイルプロセスの計測
	(キ)アジャイルの理論化・形式化
	(ク)非ウォーターフォール型開発に関する意見（外部）
2	各手法の利害得失の分析と適用領域
	(ア)状況に応じた適切なプロセスの選択
	(イ)マップの軸
	(ウ)保守・運用・保全・引き継ぎ
	(エ)非ウォーターフォール型開発の適用状況
	(オ)アジャイルの適用領域
3	非ウォーターフォール型開発の品質確保のあり方
4	非ウォーターフォール型開発への顧客参画のあり方
5	契約に関する課題整理
6	我が国のソフトウェアの国際競争力の強化
7	技術者が生き生きと働ける環境作り、優秀なソフトウェア技術者が集まるソフトウェア産業へ変革
	(ア)アジャイルの良さ
	(イ)人材
	(ウ)産業構造
8	技術者の個人成果、および事業成果の評価のあり方
	(ア)個人評価
	(イ)チームの成熟度
9	非ウォーターフォール型開発の普及

２．３ 研究会におけるアンケートの結果

本研究会では、研究会での議論やメーリングリストを用いた意見表明に加え、委員相互の認識を共有することを目的として、アンケート形式による意見表明も行った。以下に、アンケートの設問を示す。なお、各設問に対して寄せられた意見については、本報告書の付録 D に掲載する。

表 2　アンケート設問の観点

No.	設問の観点
1	非ウォーターフォール型開発の適用分野について
2	非ウォーターフォール型開発を適用するための前提条件について
3	非ウォーターフォール型開発に対する信頼性について
4	非ウォーターフォール型開発は保守、引継の困難さについて
5	非ウォーターフォール型開発に適した契約について
6	日本のソフトウェア産業構造のあるべき姿はについて
7	委員各位の共通認識について

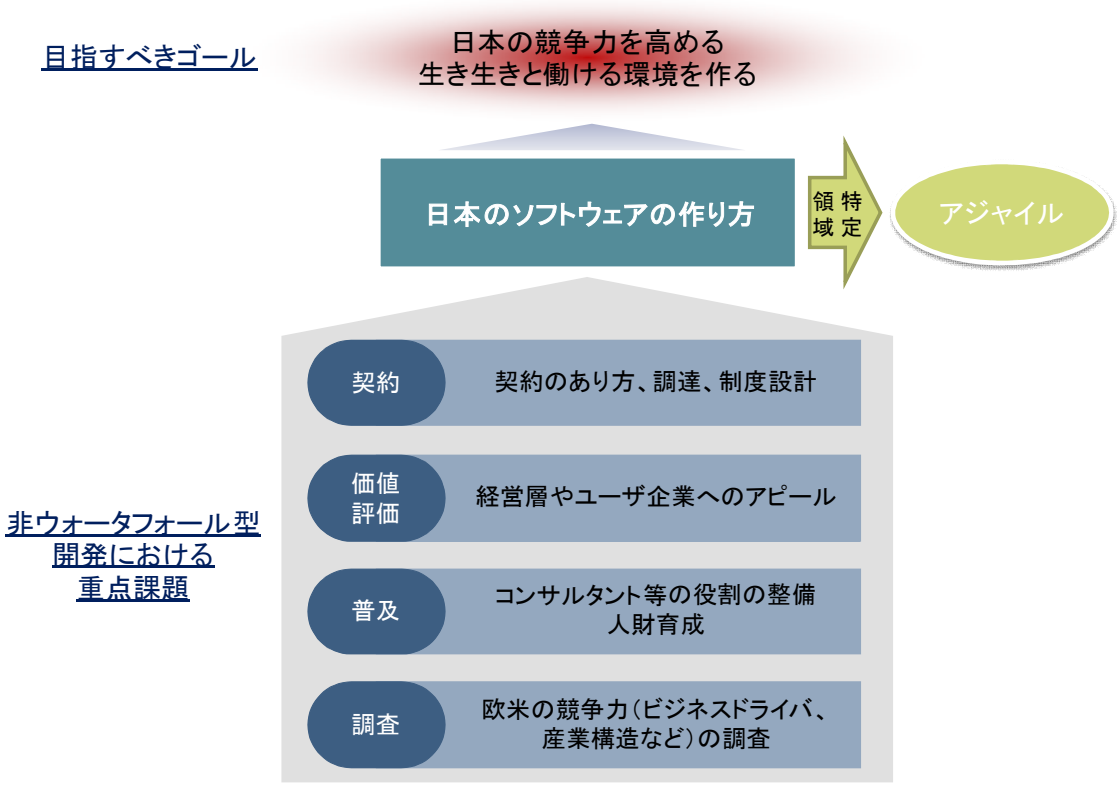
３ 現場へ適用・普及させるために必要な要諦と整理

非ウォーターフォール型開発・保守手法を現場へ適用・普及させるために必要な要諦について、開催された 5 回に亘る研究会発表およびメールによる意見表明によって調査した。寄せられた意見の記録は付録 C および付録 D に示すとおりである。

以下の 3.1 章から 3.3 章に、調査した結果を項目別に整理して記載する。

３．１ 非ウォーターフォール型開発における課題と目指すべきゴール案

産学官の連携により、我が国のソフトウェア産業が目指すべきゴールとして、グローバルな視点から見た我が国のソフトウェア産業の競争力を強化すること、および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を整備していくことがあげられる。そのためには、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を見出していくことが求められる。その１つのアプローチとなるものが、非ウォーターフォール型開発の普及と考えられる。その中では、アジャイル型開発の適用に適した領域を見定め、その活用を普及促進していくことが求められる。



以上のような認識の下、本研究会では、目指すべきゴールの達成に寄与するために、非ウォーターフォール型開発を適切な形で現場に根付かせる上で重要な課題を整理した。

３．２ 非ウォーターフォール型開発に関する課題の解決策例

前節に示した課題に対し、本研究会では解決の道筋についても活発な議論が行われた。それらについて、課題ごとに示す。

３．２．１ 契約に関して

- 開発・保守方式を選定する場合、現行の契約型（準委任型、請負型）に従う必要があるが、現行契約型の適用が難しいため、新たな契約型を創意しなければならないときに参考となる資料を提供する必要がある。
- 共通フレーム 2007 に従って二者間契約を行うため、および工事進行基準に従って会計管理を実施するために必要なガイドラインを提供すること。
- 契約や制度的な問題について、今後どういう取組みを行うべきかのメッセージを発信すること。
- 外注に適したアジャイルの契約は何かを整理すること。
- リスクやインセンティブを考慮した契約上の仕組みを考えること。

３．２．２ 価値評価に関して

- 参画するプレーヤ（とくに顧客、ユーザ）の分担すべき責務と相互のコミュニケーションに関するガイドラインを策定すること。
- 要求は未確定でも、事業戦略・執行戦術とその優先度が明確に示される必要があるため、これを促すガイドラインを策定すること。
- アーキテクチャを初期段階で仮決めするために必要最低限の非機能要件が提示されている必要があるため、これを促すガイドラインを策定すること。
- 要求の全体は未確定でも、どの開発・保守方式を選択すべきかを判定するために必要なレベルの機能要件は明示されていることが必要であるため、これを促すガイドラインを策定すること。
- 部分的ソリューションコードを、早期に実装・運用することによって顧客が得る価値を評価できるような基準を策定すること。
- 今までソフトウェアに関わりの無かった人に対して、産業的なインパクトやグローバルな視点など、非ウォーターフォール型開発における利点を周知すること。

３．２．３ 普及に関して

- 非ウォーターフォール型開発を実施できるような場を提供すること。
- ナレッジのシェアや事例の普及のための実証実験を行うこと。
- イテレーションを担当する技術者の資質・能力に関する評価基準を策定する必要がある（要求を聞いて、ソリューションコードを直接書ける資質、能力は、かなり特殊である）。
- 割り当てる職種、教育に対して、個人の資質・技能を、なんらかの指標（たとえば、IT スキル標準、Myer-Briggs Type Indicator など）を用いて最適化するためのガイドラインを策定する必要がある。

３．２．４ 調査に関して

- 欧米のエンタープライズ系における開発についての調査を行うこと。

- 非ウォーターフォール型開発をまだ適用していない現場における、エンジニアの調査を行うこと。
- ユーザがどのくらいアジャイルを認識しているか、あるいはユーザがどのくらい実施したいと思っているかなどを把握すること。

３.２.５ 全般的な事項に関して

個別の課題の解決に特化した解決策例ではなく、各種の課題に幅広く関連する事項を列挙する。

(1) 非ウォーターフォール型開発手法の体系化

- アジャイル駆動型、計画駆動型、アジャイル／計画組合せ駆動型、各開発・保守ライフサイクルそれぞれに対するプロセスおよび文書化のためのガイドラインを策定すること。
- 要求を逐次的に分割するに当たって必要となる、順序の決め方、分割方法に関するガイドラインを策定すること。
- 各イテレーションチームに提示すべき仕様、指示（かんばん）の記述形式に関する基準を策定すること。
- ALM 環境（Application Life Cycle Management Environments）の整備。
- 手法および開発を支援するツールの整備。
- イテレーションごとの V&V 手法、イテレーションの継続的完全化手法（Continuous Integration）および部分完全化時点でのシステム V&V 手法に関する基準を策定すること。

(2) 非ウォーターフォール型開発における開発チームのあり方

- チーム結成および分散配置されたチーム間コミュニケーション、リスクマネジメントに関するガイドラインを策定すること。
- 分散・自己組織化論の展開
- Distributed Problem Solving (DPS)－自律的エージェントを用いる。

(3) 各手法の利害得失の分析と適用領域

- 開発から保守へ移行する過程において必要となる、知識、遡及可能な文書、問題などの伝達に関するガイドラインを策定すること。

(4) 問題解決法に関する調査と開発・保守手法への反映

新しいパラダイムに従う問題解決に役立つ手法を、解法、分析／分割・合成法、協調求解法に分けて、主なものだけを記載する。これからの非ウォーターフォール型開発・保守手法の基底には、確固たる問題解決方法論を敷き詰めておく必要がある。

- マス・コミュニティ（ユーザ）が求めることに対する先読みに関するもの
  - モンテカルロ＋木探索　（コンピュータ囲碁などで利用）
  - オプション評価法　（金融資産のオプション取引で利用）
- 問題分析／分割／解合成法と問題解決順の決定に関するもの
  - Design Structure Matrix (DSM)
  - Probability Collectives（確率的類似度によるクラスタリング）

- Estimation of Distribution Algorithms (EDA)
- Mutual Information Maximizing Input Clustering (MIMIC)

- 設計遅延手法に関するもの
  - アーキテクチャ準拠ソフトウェア工学
    - up-front design とは⇒　初期段階では、非機能要件・制約が分からない：　アーキテクチャの基本要素だけを用いて、概念設計し、検証し、運用し、最後に製品設計へマッピングする。
- 支援環境に関するもの
  - Continuous Integration Environments
  - ALM 環境など

注：　上に記載した「解法」は、問題領域にイテレーションを適切に布石するための暗黙知、およびイテレーション内で、要求からコードを発想するための暗黙知を表出化するために役立つ。

(5) その他の観点
- プログラマの社内開発の流動性という課題に取り組むこと。
- コーディングをする領域の増加を考慮した、ソフトウェアエンジニアリングの再定義を行うこと。

３．３ 非ウォーターフォール型開発手法の事例調査

今日、我々のビジネスのみならず日常生活においても欠くことのできない情報システムは、情報システムそれ自体、情報システムを構築する技術、さらには情報システムを構築するプロジェクトに与えられる制約条件等、様々な点において**「複雑性」**を増している。

このような状況下においてこれまで日本の各企業は、初期段階で要求が定まらない（不確実な）システムに適用するソフトウェア開発・実現・実装・運用・保守を、数々行ってきた（例えば：大規模網システム内で生起する事故早期検出・事故点切り離し・網再構築・早期復旧に用いるシステムなど[1]）。

本研究会が取り組む課題は、このような**「複雑性」**、**「不確実性」**に加えて、さらに新たに生起した課題、すなわち開発始動直後から不断に生起する要求の変化によって適合的に生起する部分要求を逐次的に受け止め、そのたびに開発・実装・テスティングを行い、先行システムに対する継続的統合(Continuous Integration: CI)を行い、運用への投入を行うことによって達成する time-to-market の短縮、すなわち**「高速適応性」**を実現するソフトウェア開発・保守のあり方である。

この課題に対する抽象論は各所に見られるが、本研究会は、あくまでも現場に密着し、我が国で実際に実施され、実務に貢献している実例の調査結果をベースにして議論を展開し、これによって一般的現場への適用・普及に役立つ要諦を得る、というアプローチをとった。調査を行った事例について、以下に報告する。

---

[1] 不確実な要求から始める大規模システム開発に対しては、線形／非線形計画法、最適化技法、ヒューリスティック技法、探索法などを含むシステム工学的技法が適用されている。

３.３.１ 調査事例

本研究会において調査を行った非ウォーターフォール型開発が適用された 17 件の開発事例に加え、調査報告書において独自に対象とした 5 件について概要を示す。

(1) 小売業における業務システム開発事例

流通小売分野向け業務アプリケーション開発を出発点に、シェルスクリプトを主力言語として開発速度を高め、運用での変化に対応している事例である。取組みとしては、アジャイル的なアプローチを取捨選択し、活用している。これまで、システム全体で 160 メニュー、約 8 万ステップの開発の実績。

(2) ソーシャルネットワーキングサービス（SNS）システム開発事例

社内において技術情報を共有するためのコミュニケーション基盤として、SNS を開発。社内での実利用を経て、現在では SaaS によるサービス提供も行う。

(3) サプライチェーンマネジメントシステム開発事例

本事例は、個別事業を対象としたものではなく、本事例を実施した企業における標準的な取組みを説明するものとなっている。

(4) 研修運営システム開発事例

RUP をベースとした独自の反復型開発プロセスにより、システム開発を行った事例である。

(5) 開発案件管理 Web アプリケーション開発事例

顧客システムの保守案件を管理する Web アプリケーションソフトウェア開発。このシステムのユーザは、別のシステムの保守要員および顧客の情報システム部門である。

(6) 携帯ソーシャルゲーム開発事例

スピード、要求の変化への対応、利用率が優先される KR(携帯ソーシャルゲーム）システムの開発事例である。2 ヶ月でα版作成、その後半月でβ版、さらに半月で最終展開を行っている。小さく初めて大きく育てるようなビジネスモデルを実現する場合の典型的な例。

(7) 製造業向けプロトタイプシステム開発事例

製造業向けプロトタイプシステムの事例である。納期が機能スコープを優先された（イテレーションが終了すると、すべての顧客要求が実装されていないものの、優先順位が高い機能は実装され、動作可能なソフトウェア出力される）。9 名で 2 ヶ月の開発である。また、開発側において、アジャイル的なアプローチを体得するなどの試行的な意味合いがあったもの。

(8) 携帯端末向けブログシステム開発事例

既存のブログシステムに対する性能面の問題から、全面的に更改を行った事例。開発するシステムに対する基本的な要件は固まっているものの、新たに追加していく機

能に対する要件は全く未知の状態であった。

(9) パッケージソフトウェア開発事例

ワークフローを管理するためのパッケージソフトウェアの基盤となるエンジン部分を開発。

(10) 共通認証システム開発事例

公共団体向けの共通認証システム等の開発事例である。システムの性質上、セキュリティが重視される。また、アーキテクチャとして SOA および OSS の採用が重視されたものである。2 ヶ月程度のイテレーションを 8 回、1 年半の期間である。

(11) プロジェクト管理システム開発事例

１週間単位のイテレーションを実施した。結果的にプロジェクト生産性が約 1.5 倍に向上したため、余力分を品質と保守性の向上のための作業にまわすことが可能になった。

(12) アプリケーションプラットフォーム開発事例

ドメインの知識と従来の開発手法を良く知る顧客企業の開発メンバと当社のメンバが組むことで、相互に補完しあう形でのアプリケーションプラットフォームの開発を実施した。

(13) 教務 Web システム開発事例

大学の基幹業務の一部である「教務システム」の開発を行った。学校行事に沿った各データ登録とバッチ処理から成る。

(14) 教育機関向け統合業務パッケージ開発事例

顧客システムの保守案件を管理する Web アプリケーションソフトウェア開発。このシステムのユーザは、別のシステムの保守要員および顧客の情報システム部門である。

(15) 検索エンジン開発事例

ショッピングサイトで商品、店舗検索をするためのエンジン開発の事例である。①検索性能の向上、②検索結果表示の戦略性向上、③店舗登録情報の早期反映が開発での方針である。主なシステムオーナは、市場事業になるが、開発部内からの改善、提案もあるため、自分自身がオーナになることもある。プロジェクトによって多少メンバの入れ替わりはあるが、ショッピングサーチ開発グループとしては、比較的メンバは固定している。発案からリリースまで、2～8 週間と短期間である。

(16) システム管理ミドルウェア開発事例

システム管理のためのミドルウェアの GUI 部分の開発に関する事例。別企業にて開発した現行バージョンのバージョンアップを実施。

(17) 株式取引のための Web アプリケーション開発事例

セルと称する小集団枠組による株取引のための Web アプリケーションシステムの開発事例である。

(18) プラント監視制御用計算機システム開発事例

プラント監視制御用計算機システムの開発事例である。プラントを監視制御する標準的なソフトウェアプラットフォームを提供するものであり、過去に開発されたプラットフォームソフトウェアを異なるプラットフォームへ移植し、維持する。現在使用中の顧客システムのリプレースの為、アプリケーションソフトウェア再利用による信頼性が重視される。移植のための開発に 1～2 年を要し、移植後は約 10～15 年は使用し続けられる予定。

(19) 生産管理システム開発事例

新工場にて使用する生産管理システムの構築に関する事例。システム要件の細部は、新工場の建設に併せて固まるものであるため、要件未定ながら開発を進めなければならない状況であった。

(20) Web メディア開発事例

完全内製ではないユーザ企業が、自社システムのビジネス価値を高めるために、アジャイル開発の考え方を自分たちの現場に適用させる試みについて組織的に取り組んでいる事例である。

(21) アジャイル型開発の支援環境開発事例

アジャイル型開発の支援環境を、当該支援環境のβ版を用いて、開発を行った事例である。10 人程度の開発チームが複数存在し、各チームリーダもメンバである PMC と呼ばれるプロジェクト全体の会議体が全体のラフなプランを策定し、それぞれのチームが担当した部分（コンポーネント等）を反復的に実現している。なお、開発チームは国内だけでなく、海外にも分散している。リリース間隔は、約 1 年である。

(22) 業界共通電子データ交換基盤構築事例

アジャイル型開発手法を適用し、特定業界内にとどまらない全ての業界にて利用可能な業界共通電子データ交換基盤（OS、ミドルウェア、通信等）を開発した事例。他業界におけるデータ交換ニーズは全く不明であり、開発期間も約 11 ヶ月と短期であることから、不明な要求が明らかになってからでも十分に対応できるようモジュール構造化されている。プロジェクトのメンバは 15 名であった。

３.３.２ 調査事例の整理方法

調査した事例から、ウォーターフォール型開発と比較し非ウォーターフォール型開発を際立たせている特徴として、「不確実性」および「高速適応性」への対応がうかがえる。さらに、社会的背景となっている「複雑性」という観点を加えた 3 つの属性を軸として事例を分類し、分類ごとにどのような取組みがなされているのかを分析することにより、各分類において求められるプラクティス、留意すべき点を明らかにすることがで

きると考えられる。そこで、「不確実性」、「高速適応性」、および「複雑性」の 3 つの軸を定義し、調査した事例のマッピングを行う。これら 3 軸のうち、「不確実性」と「複雑性」は、要求されている問題が先天点的に抱えている性質について示すものであり、「高速適応性」は、プロジェクトに対して実行時に与えられる動的な制約に関する性質について示すものである。以下に、各軸の定義およびマップの説明を示す。

・水平軸（不確実性）：

不確実性の大きなものほど右寄りに、小さなものほど左寄りに配置する。ここで、不確実性を表す属性は、次のとおりに定義する。

属性	スコア				
	1	3	5	7	10
市場の不確実性	社会インフラを形成する基幹システムであるため、不確実な部分は逐次段階的であっても確実化してから実装する必要がある	目標市場に関する既成概念のマイナーチェンジで対応することが可能	目標市場の最初の推測の修正を繰り返しながら開発する必要がある	かなり不確実な要素が存在する市場	未知かつ未試験の新市場
技術の不確実性	既存アーキテクチャの拡張で解決する	構築方法が分かっている	構築方法がはっきりとは分からないが、既成技術、既成アーキテクチャを適用できる	既成技術、既成アーキテクチャでは解決できない部分が含まれる	新規技術、新規アーキテクチャ
プロジェクトの期間	24 ヶ月またはそれ以上	18 ヶ月またはそれ以上	12 ヶ月前後	6 ヶ月前後	1-3 ヶ月
依存関係／スコープの柔軟性	十分に定義されたスコープにおける責務遂行が義務付けられている	スコープが厳密に定義されていなくても、責務は義務付けられている	スコープと責務にはある程度の柔軟性が認められる	スコープには高い柔軟性があるので、義務遂行に関する契約上の配慮が必要	二者間で協議のうえ、最も適切な契約を選択する

不確実性の度合いは、属性ごとのスコア（x_i）を決定のうえ次の式に従って算出する。

$$\text{不確実性} = 2^{\sum \log_{10} x_i} \quad \cdots (1)$$

すなわち、不確実性の度合いが低い事例は次のようなものである。

(1) 定められた業務の実行および管理を、安定的に支援するもの
(2) ネットワークによって連携する分散組織間の協調を安定的に支援するもの

また、不確実性の度合いが高い事例は、次のようなものである。

(1) 「事業体自身のアジャイル化」を狙い、事業環境から得られる実時間情報と密接に連携して、企業戦略、業務執行計画およびガバナンスを支援するもの
(2) 事業体内の識見、知識、経験を相互に流通し、事業体のなかの生産性向上、品質向上、知識・スキル向上を主な目的にするもの

・垂直軸（高速適応性）：

運用（または市場投入）によって早期価値実現を段階的に目指すものを上寄りに、早期というより安全かつ安心な価値実現により重点をおくものを下寄りに配置する。

ここで、高速適応性を表す属性は、次のとおりに定義する。

属性	スコア		
	1	5	10
リリース密度（リリース回数／プロジェクト期間（月））	年 1 回	月 1 回	月 2 回以上
CI 環境の商用フレームワークからの独立度	商用フレームワークと密接に連携	商用ではないがある種のフレームワークと連携	独立した環境
ソフトウェア実装フレームワークの複雑度	クラウドや複雑・大規模な商用ソフトウェアフレームワークおよびプラットフォーム	標準的なマルチティレイヤ・ソフトウェアフレームワーク／マルティ・プラットフォーム	独自のシンプルな実装環境

高速適応性の度合いは、属性ごとのスコア（y_i）を決定のうえ次の式に従って算出する。

$$\text{高速適応性} = 2^{\sum \log_{10} y_i} \quad \cdots (2)$$

すなわち、高速適応性の度合いが高い事例には、次の特徴がある。

(1) 複数のリポジトリから構成され、CI を目的とした軽量な環境を用いて、開発の終わったイテレーション成果物から逐次システムに統合し、運用可能にするもの
(2) 商用ソフトウェアフレームワーク（ソフトウェア・サブシステムをレイヤに分けて

配置し、アプリケーションを実行するために必要なサブシステム統合を行うために必要なセキュリティ・運用管理・通信管理機能を合わせ具えた枠組み、たとえば .NET または JavaEE）のなかのひとつのレイヤに対するイテレーションおよび CI を目的とするもの：たとえば、ウェブアプリケーションやワークフロー・アプリケーションだけを対象とするもの

また、高速適応性の度合いが低い事例には、次の特徴がある。

(1) 自社の社内開発環境上において、イテレーションおよび逐次 CI を繰り返し、システム全体 V&V および顧客（ユーザ）満足度が計画値を越えた段階で、一挙にリリースし、運用に入れるように計画されたもの

(2) 運用に入ったシステムに対して、ソフトウェアの全ライフサイクルに亘って、継続してイテレーションおよび CI を繰り返し、ソフトウェアの成長、成熟、衰退、破棄までの構成管理を行うよう計画されたもの

・第 3 軸（複雑性）：

プロジェクトの複雑性の高いものと低いものに分類し、各々を別平面にマッピングする。

ここで、複雑性を表す属性は、次のとおりに定義する。

属性	スコア				
	1	3	5	7	10
チームの大きさ(人)	1	5	15	40	100
ミッションクリティカル	投機的	少数のユーザ	確立された市場	多数のユーザを持つミッションクリティカルシステム	重大な障害に耐えられるセーフティクリティカルシステム
チームの場所	同じ部屋	同じ建物	車で移動可能な範囲	時差 2 時間以下のタイムゾーン内	世界中に複数のサイト
チームの能力	熟練者による確立されたチーム	熟練者による新しいチーム	1 人の熟練者に対して限定的な経験しか持たないメンバ数を適正化	1 人の熟練者に対して限定的な経験しか持たないメンバ数をコントロール	1 人の熟練者に対して限定的な経験しか持たないメンバ数が大きくなる
ドメイン知識のギャップ	チームメンバは熟練者同様ドメインについて知っている	チームメンバはドメインについてかなり知っている	チームメンバはドメインについての支援を必要とする	チームメンバはドメインに触れたことがある	ドメインについて知らないチームメンバを入れざるを得ない

依存関係	依存関係はない	限定的かつ／もしくは十分に分離されている	中程度の依存関係	重大な依存関係	周辺システムに対する統合・完全化が必要

複雑性の度合いは、属性ごとのスコア（z_i）を決定のうえ次の式に従って算出する。

$$\text{複雑性} = 2^{\sum \log_{10} z_i} \quad \cdots (3)$$

・マップ：

「不確実性」と「高速適応性」を、直交する 2 座標にそれぞれ割り付け、開発・保守手法ドメインをこの 2 座標によって形成される 4 つの象限に分割し、x 座標を式(1)によって算出される値、y 座標を式(2)によって算出される値として事例を各象限にマッピングする。これによって、まず、事例のマクロな特徴を把握できるようにした。

また、各事例の「複雑性」を考慮するため、式(3)によって算出される値に従い複雑性の高い事例と低い事例を別平面にマッピングすることとした。

事例をマッピングした結果は、調査報告書に記載する。一例をあげると、第 1 象限には事例(1)、第 2 象限には事例(6)、第 3 象限には事例(4)、そして第 4 象限には事例(7)がそれぞれマップされる。

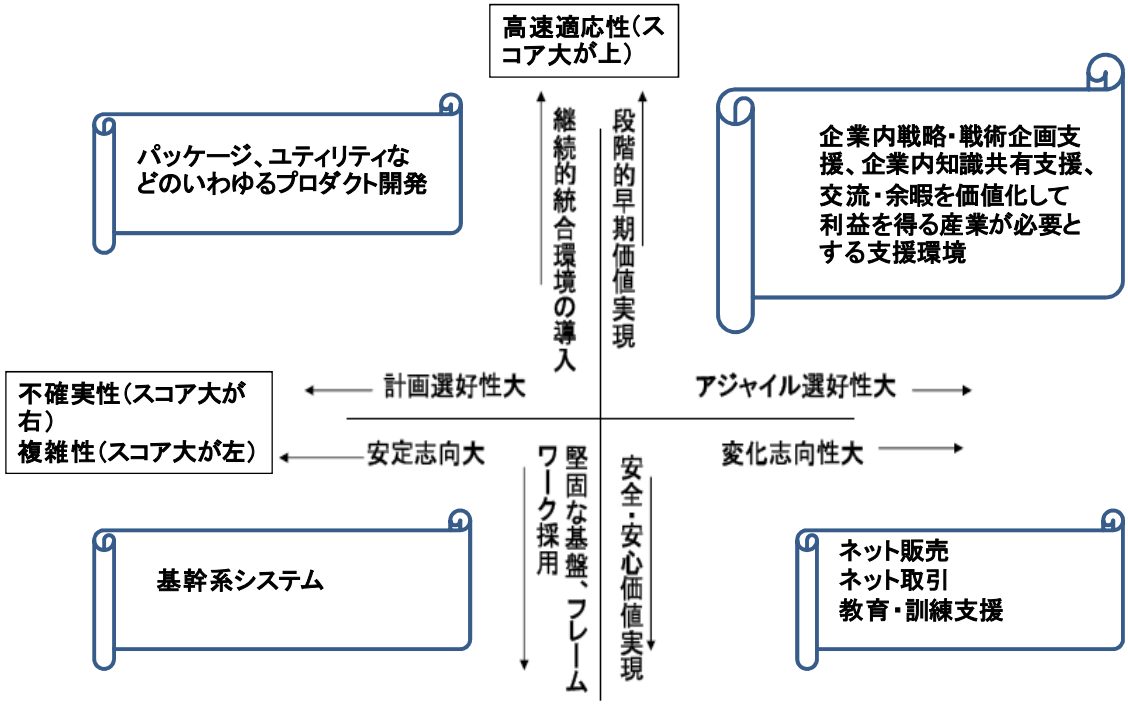


図 1　第 1 マッピング

現状、非ウォーターフォール型開発は、第 1 象限に該当するような領域に対してはその

適用が進んでいるが、第 3 象限に該当するような領域に対しては今後一層の普及を進めていかれる可能性があると考えている。

４ むすび

5 ヶ月に亘る委員のプレゼンと白熱した討議およびメーリングリストによる活発な意見交換を通して、非ウォーターフォール型開発・保守に関する動向・現状の把握と整理、および現場へ適用・普及させるために必要な要諦と整理を行うと共に、我が国のソフトウェア産業における競争力の強化と技術者が生き生きと働ける環境を作っていくための課題を明らかにした。時間的な制約もあって、これらの課題についての議論が不十分であり解決の方向が煮詰まっていないため、今後検討していく必要がある。

課題

・契約の問題への取組み(契約のあり方、調達、制度設計)
・経営層、顧客へのアピール(価値評価のガイドライン等)
・普及・展開施策(コンサルタント等の役割の整備、人財育成)
・欧米の競争力の調査(ビジネスドライバー、産業構造を含む)
・非ウォーターフォール型開発の日本におけるコンテキストにあわせた形での着地点
・非ウォーターフォール型開発に必要な技術の明確化(QCD、開発モデル・開発プロセスの見える化、等)
・非ウォーターフォール型開発に必要なツールの整備

付録 A： 研究会委員構成および研究会開催

研究会は、次の委員から構成された。

氏名		所属
松本　吉弘	座長	財団法人・京都高度技術研究所 顧問
稲村　直穂子	委員	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	委員	株式会社一 副社長
合田　治彦	委員	富士通株式会社 システム生産技術本部長代理
田澤　久	委員	楽天株式会社 開発部開発生産性強化グループ グループマネージャー
羽生田　栄一	委員	株式会社豆蔵 取締役
平鍋　健児	委員	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	委員	日本電気株式会社 主席技術主幹
前川　徹	委員	サイバー大学　IT 総合学部 教授
馬嶋　宏	委員	株式会社日立製作所 ソフトウェア事業部企画本部統括部長
松島　桂樹	委員	武蔵大学 経済学部 教授
南　悦郎	委員	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

研究会は次のとおり開催された。

開催回数	開催日程
第 1 回研究会	11 月 26 日
第 2 回研究会	12 月 22 日
第 3 回研究会	1 月 19 日
第 4 回研究会	2 月 16 日
第 5 回研究会	3 月 5 日

付録 B： 研究会における発表資料

■ 第 1 回研究会（2009 年 11 月 16 日開催）における事例紹介
(1) 永和システムマネジメントにおけるアジャイル開発の現状（平鍋　委員）
(2) DeNA での非ウォーターフォール型開発取り組みについて（稲村　委員）
(3) 非ウォーターフォール型開発に関する富士通の取組みと課題（合田　委員）

■ 第 2 回研究会（2009 年 12 月 22 日開催）における事例紹介
(1) ウォーターフォールは間違いだ！（前川　委員）
(2) 楽天の開発について（田澤　委員）
(3) 非ウォーターフォール型開発に関する NEC の取組み（広瀬　委員）
(4) アジャイルプロセス協議会の挑戦：なんちゃってアジャイルの先にあるもの（大槻　委員）

■ 第 3 回研究会（2010 年 1 月 19 日開催）における事例紹介
(1) 金融デリバティブリスク管理システムの開発の事例（南　委員）
(2) 豆蔵での適用事例（羽生田　委員）

付録 C： 研究会のメーリングリストにおける議論

本研究会では、研究会委員をメンバとするメーリングリストを設置し、メールによる意見表明も活発に行った。メーリングリストにおいて議論された内容を、論点ごとに示す。

1	非ウォーターフォール型開発手法の体系化
	(ア)ウォーターフォール型開発の課題
	(イ)アジャイルの特徴
	(ウ)アジャイル開発のライフサイクルモデル
	(エ)アジャイルにおける支援システム・ツール
	(オ)アジャイルコミュニティ
	(カ)アジャイルプロセスの計測
	(キ)アジャイルの理論化・形式化
	(ク)非ウォーターフォール型開発に関する意見（外部）
2	各手法の利害得失の分析と適用領域
	(ア)状況に応じた適切なプロセスの選択
	(イ)マップの軸
	(ウ)保守・運用・保全・引き継ぎ
	(エ)非ウォーターフォール型開発の適用状況
	(オ)アジャイルの適用領域
3	非ウォーターフォール型開発の品質確保のあり方
4	非ウォーターフォール型開発への顧客参画のあり方
5	契約に関する課題整理
6	我が国のソフトウェアの国際競争力の強化
7	技術者が生き生きと働ける環境作り、優秀なソフトウェア技術者が集まるソフトウェア産業へ変革
	(ア)アジャイルの良さ
	(イ)人材
	(ウ)産業構造
8	技術者の個人成果、および事業成果の評価のあり方
	(ア)個人評価
	(イ)チームの成熟度
9	非ウォーターフォール型開発の普及

(1) 非ウォーターフォール型開発手法の体系化

(ア) ウォーターフォール型開発の課題

- 仕様検討、設計、開発、テストを、1 サイクルでやる（＝ウォーターフォール）のは無理があるだろう。（田澤委員）
- 『前世紀末，ソフトウェア業界で膾炙された用語に，「ウォーターフォール型開発」 がある．この用語の解釈はさまざまであるが，日本に限らず，ほとんどの基幹企業は，これら解釈を客観的に受け止めるだけで，要求定義，設計，テスティングプロセスを，互いに協調しながら並行に進める方式（米国ではこれを Sashimi モデルと呼ぶ）をと

ってきた。』（松本座長講演要旨から）

- 「俺たちは過去、ウォーターフォールでうまくやってきた。だから今でもできるはずだ。うまく行かないとすれば、プロセスの問題ではなく、きちんとやらない人の問題だ。」といった論理が通っている危険性はないだろうか。（前川委員）
- （環境が変わり、アジャイルが可能となっている）スケールアウト、分散ファイルシステム、分散メモリキャッシュ、疎結合アーキテクチャが、圧倒的なスケールアウトを可能にしてくれる。圧倒的な生産性を実現すれば、量の生産から解放される。（田澤委員）
- 不確実性に対応するために一見、ウォーターフォール的でありながら、実態はアジャイル的というのもあるだろう。おりしも、松本先生の来年３月のご講演の要旨には、『前世紀末, ソフトウェア業界で膾炙された用語に,「ウォーターフォール型開発」がある. この用語の解釈はさまざまであるが，日本に限らず，ほとんどの基幹企業は，これら解釈を客観的に受け止めるだけで，要求定義，設計，テスティングプロセスを，互いに協調しながら並行に進める方式をとってきた.』とあるように、旧来の開発工程を、協調と並行化という方法で、実態をアジャイル的なものに工夫している。（大槻委員）
- ウォーターフォールと建前では通しつつも、アジャイル的な（並大抵ではない）改善が施されていることもある。（大槻委員）
- 元請け会社も、ウォーターフォールが建前であるという認識であればよいが、私がお会いした方々は、そうではなく、実態もウォーターフォールでやっている（やるべきだ、やれる）という認識の方が多かった。（前川委員）
- デカップリングという考え方がある。計画で作る部分と、マーケットイン、いいかえると受注生産、Make to order とを、どの段階でカップリングするかが生産戦略、あるいは製品提供プロセスの核である。（松島委員）
- 右から左に振り子を振らせるために、現在のウォーターフォールの問題点を挙げ、あたかもアジャイルで問題が解決されるような啓蒙を、（あえて）しているが、おとしどころが、両者をいかに組み合わせるかであることも、分かってきたと思う（松島委員）。
- 仕様を出す側も、作る側も、そうそう完全なことは出来ない（①決めた仕様は変えない→まず無い。②仕様通りにシステムが作られている→これが意外と出来ていない）（田澤委員）
- デカップリングの考え方、興味深い。 計画と生産（遂行）との分離というのは、ウォーターフォール型でもWモデルで、テスト計画とテスト遂行を分離するという考え方があるが、こういうのもアジャイル的な改善と言えると思う（大槻委員）
- 融合とか、統合とか、意味不明な（失礼）用語でまとめるのではなく、具体的にどこで接合させるのか、なにを共有するのかという開発手法のデザインの問題を議論したい。（松島委員）

(イ) アジャイルの特徴

- アジャイルとは、「規模と複雑性」「ビジネス連動」「計画駆動と俊敏性のバランス」が共通認識として挙げられる（大槻委員）
- アジャイルとは、「ビジネス連動」つまり、競合他社やマーケットの状況に合わせていつでも出荷可能にしておくというところが目的である（大槻委員）
- ビジネス連動、マーケットインで、いつでも出荷可能にしておく環境を構築して継続

的に開発を実行するスタイルを、アジャイルと呼んでいる。アジャイルプロセスというか、CI (continuous integration)がピッタリなイメージである。（田澤委員）

- アジャイルプロセスのことを考えてみると、その本質は、＜明確になった要求（仕様）は、明確になった時点で迅速に実現する＞ということだ。ゼネコン型の伝言ゲームでは原理的に旨くいかないということになる。（大槻委員）
- 製品開発におけるビジネス連動、マーケットインを考慮した「広義のアジャイルプロセス」に関する意見、同意する。（大槻委員）
- ぼくが特に不安視しているものの１つは、「アジャイル＝よいもの」、という過度な期待である。それと、アジャイルという言葉によるエンジニアリングの放棄である。（平鍋委員）
- アジャイルを導入するとソフトウェアエンジニアリングの課題が明確になる。ウォーターフォール型の場合では顕在化しなかったものだ。（合田委員：第１回研究会）
- 経営の観点から、アジャイルの価値を検討することが重要である。「IT 経営におけるアジャイル開発手法の価値」という論文を記述した。（松島委員）
- やはり経営的視点は大事である。アジャイルのオプション価値があるというのは、全くその通りであるが、採算性の観点から言うと、コストもかさむ（小ロット化や回帰テスト、小ロットごとの検収確認、不確実性に対応するためのリソース（人員）バッファなど）と思われる。無論、このコスト増大に余有る価値があると信じているが、コスト（費用）については、どうなのであろうか。（大槻委員）
- コストが本当に高くなるのか、あるいは、うまく運用できなかったからなのかは、いろいろな意見があるだろう。コスト超過の可能性が少なくなれば、ベンダとしても、メリットがあると考えられる。（松島委員）

(ウ) アジャイル開発のライフサイクルモデル

- 共通フレームなどで使われているＶモデルは、とくにウォーターフォールを標榜するためのものではなく、主としてＶ＆Ｖのためのモデルであるに関わらず、ウォーターフォール開発モデルであるように誤解される向きがある（松本座長）
- Ｖの左上（要求）がぐらぐら動いている様子を想像してみる。その場合、「要求全体」を分析->設計->実装->テストしても、時間がたつとぶれるので問題である。左上のうち、比較的動かないところを、すばやく、Ｖを移動して右上に持っていく活動が必要である。そして、それを繰り返すこと。それがアジャイルである。WF が静力学なのに対して、Agile は動力学である。（平鍋委員）
- サービスを運営してきて思うのは、わりとぎりぎりに出てきたアイディアでも恐れずに、いったんまな板の上に上げて検討することが必要だということである。運用含めたトータルコスト、サービス品質にとってはプラスに転じることも結構あった。（稲村委員）
- 共通フレームについても、改めて見直してみたが、ウォーターフォールを標榜していると誤解されても仕方が無いなと感じた。ウォーターフォールを定義できる辞書を整備して、最後に、段階的や進化的プロセスも説明できるよという書き方になっている。要求定義が不十分であることが、リスクとして扱われている。アジャイルプロセスでは、このことはチャンスである。（大槻委員）

・ 私は「進化型開発プロセス」と言う場合には、ソフトウェアやシステムが実世界（業務世界）に及ぼす影響によって、顧客の認識も変わって、新たな要求がでてくるところも、捉えたいと考えている。（大槻委員）
・ 問題なのは、T（テスト）という作用を受けて＜認識が変わって R（要求）が変わる＞ことである。このことを旨く表現したい。（大槻委員）
・ 自然環境を「顧客要求」ととらえれば、大部分のソフトウェア事業者は、これに反抗することはなく、適合できるように、それぞれ自分の DNA を改造している。せっかくの機会であるから、当研究会で合意できるモデルを作りたい。（松本座長）

(エ) アジャイルにおける支援システム・ツール
・ IBM や Microsoft が ALM (agile application life cycle management)の分野に出てきたのは、Rally や VersionOne がこのツール市場のリーダーになってきていて、世界的にも市場が広まってきたことが動機である。また、TPS のかんばんを支援ツールに対応させた製品として TRICHORD がある。（平鍋委員）
・ ALM と、ALM を支援するツール環境の世界的現況調査、および大規模システムに適用するに当たって、日本企業が参照できる国産 ALM ならびにツール環境は、当研究会のアウトプットとしてまとめる必要があるだろう。（松本座長）
・ この手のツールに関しては、規模への対応、国産ツールの動向、プラットフォーム依存性などについて整理しておくのがよいと思われる。（大槻委員）
・ 現在、比較的工程を区切って行われるタイプの開発の製造工程、試験工程をモニターする計測ツール、というか計測プラットフォーム EPM(Empirical Project Monitor)という道具立てを提供し、普及活動をしている。国産、非ＷＦ開発支援システムの仲間入りができるだろうか（神谷さん）
・ 支援システム、ALM ツールなどの国内外動向について、検討していく際に重要な観点だと思っているのが、「ライフサイクル」である。アプリケーション、プラットフォーム、組織（ユーザ／ベンダ）、ツールのそれぞれのライフサイクル（寿命）の問題である。従って、SEC で提供している／するツール（EPM？など）についても、ツール事業そのものの事業継続性が問われている／いくと思われる。（大槻委員）
・ 全く核心を突いたご指摘と思う。戦争で個々の戦闘と同様に兵站がその死命を制するように、ツールやソフトウェア製品では継続した開発の実現法が命となる。(神谷さん)
・ 開発支援ツールを継続提供できそうなベンダは、IBM（ラショナル）、マイクロソフトくらいしかないだろう。あとは、OSS である。使い方のコンサルテーションも重要である。事業部で、ClearCase を全面採用したときには、日本 IBM（当時はラショナル）にコンサルテーションをかなり受けた。（馬嶋委員）
・ 日本がソフトウェアに関して「世界標準確立へ向けての施策策定」を検討していくのであれば、どういった領域にどれくらい投資が行われていくかという観点も必要である。独自のイノベーションで差別化を図り、他方オープンイノベーションで標準化を推進するという総合的な戦略が必要である（大槻委員）
・ グローバル展開に掛けてツールを開発しているが、最も難しいのがマーケティングのメッセージやコンセプトづくりと、「インストールを絶対失敗させない」などの製品哲学だと思っている。日本からだと、なかなかこれができない。（平鍋委員）
・ GeneXus と称するツールの利用により，生産性は確実に向上するが、それは少ない工

数で実現できるということになることから，ベンダとしては，価格を人月工数で算定する現状の慣習では，開発受託額が小さくなることになり，おもしろくない、と言っている例もある。（山下さん）

- 確かに、１案件からの売上は減るが、フループットが上がる訳であるから、早く開発してもうけた時間で、他の案件をやれば良いだけである。やはり、ユーザ企業自らが、自前で開発できるように自らを鍛える必要があるだろう。生産性の向上により、多くのサービスがローンチされ、多くの方々に喜ばれ、皆が元気になり、当社も潤う（田澤委員）
- 実際には、ライセンス費用が高額であったり、学習コストの問題、柔軟性の問題、あるいは 4GL での反省にもあるように、メジャーになりえなかった場合のアプリケーションの保守の問題とかが導入の障壁として認識されていることが多かったように思う。しかし、最近の景況下だと、SIer にとって、受注できるかどうかも極めて重要であり、そこでは工数（開発生産性）や単価などを下げることで受注額そのものが下がったとしても、その案件が受注できるのであれば、それは十分動機付けになる。（南委員）
- 「バグ作って生活安定」という話が昔はあったが、今後の競争社会では自然に淘汰されてゆくだろう。背景に、「部分最適」と「全体最適」の話があり、複雑な分業体制の中で、「部分最適」が「全体最適」に繋がらない話である。むしろ、「部分最適」は「全体最適」につながらないことの方が多い、というのが現実と思われる。（神谷さん）
- ムーアの法則を織り込むように、ソフトウェアの生産性向上（言語の高級化）を織り込み、成長・進化していく仕組みを考えないと、ビジネスとしては難しいと思われる。しかも、支援環境面の進化の仕組みが、提供ベンダにも依存しているので、結局、外国のツールベンダのライセンス料ばかりがかさむということになりかねないであろう。（大槻委員）
- このように，様々な事例を集めて整理・分析することにより，非 WF 型開発の本質が見えてくることを期待している。さらには，WF 型と非 WF 型に共通の，ソフトウェア開発の本質にまで結び付けられれば，すごいことだと思っている。（山下さん）
- GeneXus のような言語は、現在の分類では、DSL（domain-specific language）に属するか、と思う。DSL も、アジャイルに対する formalism 適用のひとつとされているから、事例として取り上げて良いと思う。しかし、現代アジャイル族の言いたいことは、DSL で要求を書くならば、コードを作ってしまったほうが、よほど楽だし、効率的に CI できるということであろう。（松本座長）

(オ) アジャイルコミュニティ

- アジャイルに関する大きな国際会議には、IEEE が 2006 から毎年開いている Agile Conference 200X と、Agile Alliance が主宰する Agile200X がある。（松本座長）
- 時系列でいうと、XP200x, Agile Development Conference 200x(3<=x<=6), XP/Universe 200x が大きなカンファレンスとしてあった。XP/Universe が Agile/Universe と名前を変え(2005 年?)、 それと、Agile Development Conference 200x がマージしたものが、Agile 200x である(x>=6)。（平鍋委員）
- アジャイルは、もともと「知行合一」が基本なので、論文が少ないのは 「机上で考えた理論」というものへのアンチテーゼである。 文献として当たれるものは、「その人自身」が一番信頼できるソースである。 例えば、Agile2009 の参加者が 1500 人で、

その 1/3 がカンファレンスの スピーカーだ、という驚くべき参加型である。（平鍋委員）

- アジャイルコミュニティの大多数は、宣伝モードに入っていて、面白い新しいことをやっている人達は、コミュニティから出たいと感じていると思われるがどうであろうか（松本座長）
- そういう面はあると思う。型にすると停滞するというのも事実だろう。（田澤委員）
- どのコミュニティでも、参加者は宣伝モードなのだろう。ある程度レピュテーションを上げられれば、出たいと思うのかもしれない。むしろこれは、「アジャイル」という言葉が、どれだけ社会の中で消費されてしまったかによるだろう。（大槻委員）
- Cockburn の言葉：「自分は agile を葬るために Agile2009 にやってきた」の真意は何であろうか（松本座長）
- コバーンのスライドを見る限りでは、シェイクスピアの言葉を借りての、アジャイルのプロモーションに見える。（大槻委員）
- 本人に聞いてみた。マーケティングフックと真意の２つがある。マーケティング面はシェイクスピアのもじり。真意は、さらに２つあって、
  - １．すでにアジャイルはキャズムを越えていて、小さなチーム環境で「うまく行くか？」を問うのは適切ではない（うまく行く）。より広い問題を問うべき。（90 年代の Agile を葬る）
  - ２．アジャイルのようなアイディアを、より大規模で難しいプロジェクトに導入していく過程で、ソフトウェア開発のより広い理論（協調ゲーム、リーン、工芸モデル）からの視点を築くべき。（平鍋委員）
- 「theory」という語は、アジャイルコミュニティにはふさわしくないかもしれないが、Cockburn が書いてきた文章の裏には、theory への接近が常に見え隠れしていたので、今後とも、この人との接触は厚くしていく価値があると思う。（松本座長）
- 彼は書籍も多いが、協議会の APMBOK(Agile Project Management Body Of Knowledge)WG では、ここのところ、"Crystal Clear"の超訳や検討にも取組んでいる。
- ビジネス価値とコマ切れリリースの関係がほとんど議論されない。（松本座長）
- Barry Boehm 先生の Value Based Software Engineering の一派といった、他のコミュニティネタなのであろう。あまり交流がないのではないかと思う。（あるいは、意図的に住み分けているかもしれない）（大槻委員）
- 自分がやった経験的プラクティスは紹介される（自己紹介だけの目的で投稿する人が多い）が、第 3 者にとって学習可能なプリンシプルが議論されない。（松本座長）
- 基本的なスタンスが 「これクールでしょ？」 なので、皆がある例を参考に自分流を常に作っているという事なのだと思われる。ただし、セールスフォース社の事例は参考になると考えている。（田澤委員）
- ポジションペーパで自己紹介して、カンファレンスの中で暗黙的に伝承されているのだろう。また、本当に価値のあるプロセスや方法で、それがチームや組織のコアになっているのなら、それを第三者に判るように伝えるというのは、ビジネス的な観点ではあり得ないであろう。（大槻委員）
- ここは、逆方向で、「経験を持たない人の机上プリンシプルを排除」というのがはたらいているのは確かだと思われる。経験論文の重視、参加の尊重、エピソード（物語）の重視がなされている。（平鍋委員）

- Alistair や Scrum 開祖の Jeff Sutherland, Collaboration Explained の Jean Tabaka, Agile Retrospective の Esther... とにかく、アジャイルは権威でなくて、コミュニティだ、と感じることができる。（平鍋委員）
- Kanban（かんばん）の WIP (works in progress) limit（仕掛り制限）の明白なものと暗黙のものを識別する基準がないか？（松本座長）
- 当社では WIP limit の様なものを、Redmine で実装しようと試みている。また、この機能に内部統制要件も加えて、JSOX 監査にも対応出来るタスク・承認・証跡管理の統合ツールにしようとしている。（田澤委員）
- Lean（Kanban）と制約条件の理論（ドラム-バッファ-ロープ）はどうなっているか（松本座長）
- Kanban についての規範的プリンシプルがない。（松本座長）
- Kanban は現在、形式化が一番急激に進んでいるメソッドである。（平鍋委員）
- 暗黙なものを形式知化するとか、規範的プリンシプルにするというのは、未だ難しい領域なのだと思う。（大槻委員）
- この分野のリーダーは、David Anderson と Alan Shalloway,後続は、Karl Scottland, Corey Ladas である。（平鍋委員）
- 価値やビジネス駆動が全体傾向として読み取れ、我々もこの方向の検討はしていくべきだと考えている。（大槻委員）
- 現在、IEEE および ISO/IEC SC7 WG20 では、SWEBOK 2010 の策定を行っているが、ソフトウェアエンジニアリングの立場から、先行しているプロジェクトマネジメント知識体系（PRINCE2TM、PMI's PMBoK(r)、APM's APMBoK、IPMA's ICB）のなにを引用し、なにを独自に SWEBOK のなかに新たに起草すればよいか、の議論がなされている。これらの動きを視野に入れ、すでに確立している知見は引用したい。（松本座長）

(カ) アジャイルプロセスの計測

- アジャイル関連で、触れられにくい話題として、アジャイルプロセスの計測がある。計測の中でも、進行中のプロジェクトの計測、つまり「インプロセス計測」をテーマにしているが、アジャイルプロセスは、この計測の格好の対象と考えられる。現在ＳＥＣでは、従来型のソフトウェアプロジェクトのインプロセス計測に取り組んでいるが、この研究とほぼ同じ時期、つまり 2002 年頃から、ハワイ大学の Philip Johnson 教授が Hakystat と名付けた、ソフトウェア開発ツールに中に埋め込む型の計測ツールを開発し、主にアジャイル開発のプロセスを計測して発表した。計測対象は、ソースコードの記述量、修正量、ビルド回数、構成管理など特定のツールの使用頻度など開発や開発管理ツールの中で自動計測できる事象である。（神谷さん）
- どこまでやるかはあるが、SVN (Apache Subversion), Hudson, Maven (いわゆる CI 環境)で出来るだろう。 Redmine まで組み合わせれば、開発者のタスク数までトラッキング出来る。（田澤委員）
- 研究会の中での発表にもあったが、こうしたソフトウェア開発環境と一体となったトラッキングの環境には大変興味がある。（神谷さん）

(キ) アジャイルの理論化・形式化

- 現在のアジャイルは、『数さえこなせば解るようになる』という「術」の域に留まっているとしよう。この「アジャイル術」を、嘉納治五郎のように理論化して、「アジャイル道」を、世界に先駆けて創始することが、我々に課された責務ではないだろうか。（松本座長）
- 日本の「道」を開くことが「theory」に通じるというのは、すごく刺激的な発想である。（大槻委員）
- 国産技術と定理（theory、または道理）を大きな声で昂揚させるべきである。（松本座長）
- theory を創ること（形式知化）は、私は、そこは、どうやっても欧米に勝てないと思う。これは日本語、英語問題、および、コンセプト作りに長けた民族かどうか、ということを考えるとそう思える。（平鍋委員）
- 現場が馬鹿にしないで実践してくれるような、地についた、アジャイル道理、または定理（術ではなく）を、着実に、提供していかねばならないのではないだろうか。（松本座長）
- Cockburn の言う theory は、現場というより、当面、組織上層や原価管理、生産管理をやっているスタッフの意識改革のために必要だと考えて注目している。（松本座長）
- 「現場が実践してくれるような」というのは、アジャイルはもともと勝算があって当然であり、むしろ、マネジメントとの摺り合わせが問題であると思われる。（平鍋委員）
- 「日本の製品を買う目的は技術の習得であって、日本に profit を与えるためではない」、と明言する複数の某国との取引では、暗黙知を無償で提供することはできない。なんらかの方法で、国際的に提供できる形式知、または Jazz のような環境にして、金を取るようにしなければならない。（松本座長）
- 「Agile は一部では使われているものの、日本の得意とする高信頼性、高品質を最高価値とする開発文化、価値システムとは合わない。よって、アジャイルではなく、１５年前に戻って、経験や理論によって裏打ちされている知見から、再度、ソフトウェア工学を組み立てよう。そちらの方が、日本が国際的に優位に立てる。」 というやり方は逆におもしろいかもしれない。（平鍋委員）
- 現在、私は、添付の論文にある「形式手法のアジャイルへの適用」に、以前から興味をもち、勉強しているところである。（松本座長）
- 最近（3 年くらい前から）、テスト駆動（TDD: Test-Driven-Development）から、振る舞い駆動（BDD: Behavior-Driven-Development）という動きがあり、「テストを書く」、でなくて、「振る舞いを書く」というというメンタリティでテストコードを書く動きがある。特に、シンタクスを工夫して、すごくユーマンリーダブルな、テストスクリプト、というか、シナリオを書ける。有名なものは、Cucumber であり、これに、日本語語彙テーブル miso をかませると、とても読みやすくなる。アジャイルの考え方である TDD の延長に、もしかしたら、形式記述の考えとの接点があるのではないか、という勘がある。（平鍋委員）
- アジャイル開発・保守には、人の話し言葉のような実行可能な言語（naturalistic programming Language と呼ばれます)が必要である。たとえば、Pegasus project というのがあり、このようなことを研究している。（松本座長）
- 組織のコアになっているアジャイル技術は公開しない傾向にある。例えば、形式手法のモデル検証で振舞いの証明やモデルチェックする方式（MDD?)というのは、我々が

教科書や論文で知り得るレベルを遥かに超えた（ドメイン特化）方法が使われているようである。（大槻委員）

- Formal Method のアジャイルの適用についてだが、有望な観点だと思う。導入にあたっては、もう一つ、新しい（形式）言語を覚えることに対する抵抗を和らげる必要がある。一方、形式手法に興味を持って勉強したがるエンジニアは、スーパーシェフ的な人なので、人の選別に使うこともできる。（大槻委員）
- 日本語処理については、京大を始め各企業にエキスパートがおられるので、そのような方々の協力を取り付けることができれば、世界の先端を行く executable 日本語 Naturalistic Programming Language を実現することができるのではないか、と考える。（松本座長）
- 最近の議論では、ソフトウェアアーキテクチャを決めないで始動し、アーキテクチャ決定をできるだけ先送りしながら、品質を落とさず、失敗しない非ウォーターフォール型手法が求められている。（松本座長）

(ク) 非ウォーターフォール型開発に関する意見（外部）

- エンタープライズ系では、ユーザ・ベンダ企業をはじめ、地域・中小企業や大学・有識者の方々合わせて 17 者へのヒアリングを行った。意見の中から，非ウォーターフォール型開発に関するものをピックアップして紹介する（山下さん）

- ソフトウェアの基本はアーキテクチャである．大規模になるとアジャイルでは対応できない．アーキテクチャ設計をきちんとした上で，コンポーネントごとに，再利用，アジャイル開発等，用途用途で最適な設計を進めることになる．優れたアーキテクチャデザインがあり，コンポーネントをどう実装していくかが大事である．［大学］
- アジャイル開発形態に関しては，文化の問題が大きい．過去の経験上，“契約”が重要課題の一つ．過去に関わった事例では，半期単位での支払いとした．また，“ユーザのプロジェクトへの参画意識”もうまくいくかどうかのポイントの一つ．公共系ユーザは一旦発注後は完成までほとんど関与しないが，流通系ユーザは早くリリースできればうれしいのでどんどん関与してくる．ある意味では，保守(開発)と似ている．［大手ベンダ］
- アジャイル型開発について，ウォーターフォール型と分ける必要があるのか疑問．要件の固まり具合への対応の問題．両者の混在あるいは組合せで，多様化する技術のいいとこ取りの傾向．システム実現の速度感の違いと認識．むしろ，出来上がったシステムの保守を素早くできるのかが課題である．［大手ベンダ］
- アジャイルで変わっていくと思うが，ベースは変わらないと思う．［中小ベンダ］
- アジャイル型開発形態について，個人としては，小さなウォーターフォール型の繰返しと思っている．実際には，完全なウォーターフォール型開発はなく，プロセスが上流から下流に並行して流れている．型に嵌めない方がよい．アジャイル型は規模の小さな開発が対象であり，開発規模が大きくなるとウォーターフォール型になる．また，メトリクスをきちんと収集できないと進捗管理が十分できないため，機械的にどうやって管理するかが課題．［地域ベンダ］
- アジャイル型開発については，ユーザの協力が必要だが，東京ではやっているところもあるが，ここでは協力はするが任せてしまう．今までとなりの人が何をやっているのか分からなかったが，プロジェクト内のコミュニケーション促進（※これはアジャ

イル導入効果？）という点では，取組み（『スクラム』の一部を採り入れたもの）を開始している．［地域ベンダ］
- エンドユーザについて，短納期要求，見えないと不安，という傾向がある．アジャイル型開発の場合，テストの繰り返しのために時間がかかってしまうことが課題．［地域ベンダ］
- 社内標準は WF 型であり，アジャイルは社内審査に通らない．よって，本体では未導入であるが，下請の企業は使い始めているようだ．（大手ベンダ：100 社ヒアリング）

(2) 各手法の利害得失の分析と適用領域

(ア) 状況に応じた適切なプロセスの選択

- アジャイル、ウォーターフォールいずれもラベルではあるが、プロジェクトのステークホルダーと開発チームとチームメンバの心理とがお互いに見えるようにする工夫がまずあって、それらの間で満たすべき最低限の論理（それをアジャイルプラクティスというのか、セル生産方式というのか、改良ウォーターフォールというのかは別として）を設定し、そこから逆算して、このプロジェクトはプロセスのスペクトルのこの辺というべきなのではないかと思う。（羽生田委員）
- スペクトルの軸の一つとして、要求、技術等が固定されているか、変動するかという「不確実性」の観点があると見てよさそうである。アジャイルプロセス（それが何とラベル付けされようが）のもたらす（ユーザから見た）価値の一つは、仕様決定の意思決定を遅らせることができること、（要求）仕様変更や状況変化に迅速に対応可能なことだと思う。 いわば、キャンセル可能な（高価な）航空券の価値、つまり、オプション（選択肢を持てる）価値である。（大槻委員）
- 不確実性への対応手法については、アプリケーションフレームワークとか SOA での議論においてされるような「摺り合わせ（インテグラル）」と「組み合わせ（モジュラー）」の使い分けのフレームは使えないだろうか、と考えている。作業効率からすると、「組み合わせ」の方が恐らくは良いだろうから、可能な限り「組み合わせ」の世界にしたいところだが、「組み合わせ」には「間違った分け方をするリスク」があるので、このリスクを誰かがとるか、リスク軽減のために「摺り合わせ」を導入するか、の判断が必要になる。（南委員）
- スペクトルの属性、向き不向き、適用領域とともに、その「分量」に非常に興味が湧く。ただ、この「分量」というのがむずかしい。当該手法を用いている人の人数か、ソフトウェアの開発量か、産業的な市場規模か、内製と外注があるし、エンタープライズと組み込みがあるし、なかなか決め手が難しい。（神谷さん）
- 問題は、目的とする事業のもつ特性スペクトラムに適合するプロセスをどのように選定し、組み合わせるか、である。（松本座長）
- スペクトルというのは対象領域（コンテキスト）と方法論と両方についてあると思うが、M x N x O x ...という膨大な対象領域に X x Y x Z x ...といった方法論をマップする、という整理はさすがに厳しいので、M と X, N と Y、のように分解した結果同士で議論できるようになると良い。（南委員）
- 「変化」も特性の異なる要素ごとに分けたテーマ設定が必要である。例えば、(a) 技術変化 (b)ニーズ変化 (c)調達環境変化、が考えられる。これらは、対応方法や評価軸も異なるであろう。

(イ) マップの軸

- MAP の軸については、システム開発の規模（規模は無関係だという意見もあるかもしれないが)、要求仕様が変化するリスクの高低（開発中に仕様がどの程度変化する可能性があるのか）の 2 軸で整理してはどうかと思っている。システムの重要性については、あまり関係がないのではないか。（前川委員）
- 規模というよりも、そのシステムの想定されるライフサイクルの長さ、によって違いがある様に思えてきた。それがシステムの重要さということなのかもしれない。（田澤委員）
- （事例整理について）この目的はレーダーチャートの軸（メタデータ）を見つけるためのデータ収集と思う。現在の埋めるべき項目は仮のもので、そこから「データを集めてメタデータを探る」というアプローチと見た。また、項目名に縛られて出るべきメタデータが出ないことを避けるために、調査主体の方には「ぜひ、うちにインタビューに来てください」ということで、口頭ヒアリングの実施をお願いした。（平鍋委員）
- 私も規模よりもライフサイクルの長さ（想定される寿命）がポイントだと思う。保守することで使用に耐えうる状態をキープできる期間、という意味である。更に保守なども考えると、そのシステムの寿命と、開発者のライフサイクル（そのシステムに関わり続けられる時間）の相対的な長短が更に重要かも知れない。（南委員）
- あと、どう作るのか（プロセス含む)、を判断する場合には、
  - アプリケーションアーキテクチャ
  - 利用するプロダクト
  - データモデル
  - UI 周りの仕様
  - 作る人のスキル
  - 非機能要求（性能、信頼性など）

  のどこに主なリスク（作ってみないとわからない）がある/ないと想定したか、というのも関係がありそうである。これは単純に要求が厳しいかどうか、ではなくて、実現の確信度合いがどうだったか、ということである。しかし、過去の事例を紐解くに、最初からきっちりリスクについてどこかに記述されていれば良いが、後付で調べるのは難しいかも知れない。（南委員）
- マップの軸として、”Adaptive Agility, T. Little et. al, IEEE ADC 2004”では、不確実さと複雑さを採用している。（松本座長）
- Complexity = そもそもの難しさ、Uncertainty = 不確実・変化の度合いという軸自体は納得感がある。（南委員）
- 座標軸を定める目的は､大別して 2 つある｡事例の特性（attributes）による仕分けと、次に取り組む開発プロジェクトに対して推奨する実践プラクティス（優先度つき）の提示である。たとえば軸によってプロジェクトの特性を 4 象限に分け、各象限ごとに推奨するプラクティスを、must, should, may に分けて提示する。（松本座長）

(ウ) 保守・運用・保全・引き継ぎ

- 開発者も順次新しいプロジェクトを抱えることになるので、保守・引き継ぎを考えねばならない。たとえば 3 年程度だと、そのまま属人的に何とかなるのだが、5 年-10 年となると部署の異動などもあってなかなかそうは行かない。（南委員）
- モチベーション・生産性が高い開発者は得てして多様なプログラミング言語、多様なソフトウェアを使う傾向にある。保守・引き継ぎ、あるいは運用を考えると標準化（統一までの必要はないが）の必要があるが、この整備と展開が大変である。（南委員）
- 保守専門の会社では、お茶屋さんのような専門別グループの維持・管理手法が必要となる。（松本座長）
- アジャイルというと、新規の価値創造にばかり目が向くが、保守の仕事は、イテレーションや Scrum-Sprint への対応と同じである。これからの時代は、開発では収益が得られない。保守とサービス、およびアジャイルによる知能の切り売り販売で稼ぐべきである。幸い、クラウドの登場で、切り売りしても、大規模システムに容易に組み込めるようなアーキテクチャになってきた。（松本座長）
- アジャイル開発プロセスは、ウォーターフォール型と違って、運用の実態が反映されやすいように思われるから、良さそうである。（田澤委員）
- 保守については、ある程度までは開発者が直接絡まないと、現場のニーズから開発者のマインドが乖離してしまう問題があるので、もし引き継ぎ（別会社などに）が前提だとすると、どの時点で引き継ぐのが望ましいか、という課題もある。（南委員）
- ソフトウェアファクトリの考え方や、ワークパッケージ（作業の前提条件、後置条件、リソース、期間などで定義）を定義して、組織化し規模に対応することの重要性は実感した。（大槻委員）
- 運用というのはシステムや製品が投入された実世界、つまり、システムの文脈（コンテキスト）を扱い、開発者はシステム要求や実現を扱うので、別世界ではあるものの、これを繋げるところがポイントになるような気がしている。（大槻委員）
- 開発の話題ばかり、あるいはソフトウェア製品開発、ばかりに目が行くが、IT 支出のかなりは運用/保守である。この生産性が議論されていない。（松島委員）
- サプライヤとしては、製品全ライフサイクルを見通して、プラスのキャッシュ・フローを叩きだせる顧客以外は相手にしてはならないし、予測される全キャッシュ・フローから求めたキャッシュ・フロー・現在価値（PV）に見合うだけの開発費を出してくれないようなケチな顧客からは受注しないほうがよい。（松本座長）
- ユーザシステムを、データベースを含めて全システム仮想化し、開発段階で仮想化されたユーザシステム上で、テスティングを行うと良いと記述している論文がある。（松本座長）
- 「保守」って呼ぶべきかどうかは迷うところだが、誤りを訂正、予防的な対策、システムの実行が実世界に及ぼす影響のフィードバック、適応など含めて、アジャイルプロセスのスコープっていうのがあるのではないか。こうすると、広義のリファクタリングみたいなアーキテクチャの見直しとか構造の最適化とか外部の振舞いに影響を及ぼさないような作業も位置づけられると思う。（大槻委員）
- 運用を持たず開発だけを担当しても、アジャイルに入って来られないと、本音では思っている。（田澤委員）
- 新たな要求の発生が、いろいろなところ（ユーザ、運用者など）から、いろいろなタイミングで起こるところが本質だと思う。（大槻委員）

・ 運用を持たず開発だけではアジャイルに入ってこられない、ということは、ベンダから見れば顧客と継続してつながりを持ち続けることと理解した。（伊久美さん）
・ アジャイルが、時間軸や規模感でなく、ライフサイクル全体を捉えた開発ということであれば、社会インフラの開発にも、アジャイルが良いように思う。（田澤委員）

(エ) 非ウォーターフォール型開発の適用状況
・ Oracle 社も、DBMS の 8i を作っていたとき、アジャイルプロセスでやっていたと聞いている（田澤委員）
・ もう一つ有名な例が、Microsoft 社の Excel でやっていた Sync and Stabilize という常にビルドアップした実行可能なコードを保持するイテラティブな方法である。これも今思えばアジャイルプロセスであった。（大槻委員）
・ 韓国のサムスン SDS と LG CNS（韓国の２大 SIer）にヒアリングに行ったとき、彼らは通常、システムを３回作ると言っていた。設計ー開発ーテストのプロセスを大きく３回繰り返している。こういう「繰り返し」もありかなと思っている。（前川委員）
・ アジャイルがすでに大規模システムに浸透し始めている状況が報告されている（IBM 内の普及度、米国での普及度）。いずれにおいても支援ツール（IBM の場合は、Jazz）および ALM (Agile application life cycle management : ボーランドやオラクルがリード、最近マイクロソフトも注力している）の確立が大きな役割を担っていることが分かる。（松本座長）
・ 非ウォーターフォール型開発、と言うとき、Web アプリやベンダの製品開発が挙げられるが、「ニーズや顧客の要求へ迅速に対応する」、「変化する要求を反映させる」のは、SI 現場にも必要である。現場では、新規プロジェクトというよりも、機能追加や更新プロジェクトが多い。こういうところへ、どう対応するか、ということも重要である。（伊久美さん）
・ IBM のスマートプラネットでは、不確実な問題をどのように捉え、どのような部分から部分解を逐次実現しながら、部分解を統合し、全体解に結びつけるか、が話題とされている。このような手法は、ソフトウェアの視点からは、アジャイル手法と親和性をもつため、このような大規模な問題に対するソフトウェア開発・保守には、積極的にアジャイル手法の挑戦が行われている。（松本座長）

(オ) アジャイルの適用領域
・ 私個人としては、Agile プロセスよりも、Agile な人、文化、組織というようなもののほうに興味がある。それができていれば、プロセスはおのずと Agile になる。逆に、いくら Agile なプロセスにしても、Agile な人、文化、組織じゃないと、本質的に Agile にならない。だから、Agile の x x プロセスがどういうプロジェクトに適するか、というような議論には、あまり興味がない。（馬嶋委員）
・ 同意する。（田澤委員）
・ それぞれは、万能ではなく、それぞれの適用分野があるらしい。（神谷さん）
・ 限定された適用分野があるかどうかはわからない。私はアジャイルというと、TDD 型の CI 環境がまず頭に浮かぶが、そういう場合、確実に言えるのは、世の中でいう”WEB システムの様なものがアジャイルに向いている”というのは幻想である。WEB システムも色々あるが、一過性のプログラムに CI 環境は不要である。ライフサイク

ルが長いサービス、システムにこそ CI は向いている。（田澤委員）

- Agility を重視して、ビジネス上必須であれば Agile 開発にならざるを得ない。よって、Web アプリだと向いている/向いていない、信頼性がどうこう、というような「特徴（？）」をあげて、「だから、Agile に向いている/向いていない」というようなことを言うことの意味が理解できない。（馬嶋委員）
- 今回、事例調査を通じて分かった収穫のひとつは、一般にアジャイルに背を向けていると思われている、基幹系アプリ（金融、証券など）、社会インフラ系アプリ（宇宙、エネルギー、交通など）を提供している大企業において、現実的にはアジャイルがかなり浸透しているということである。（松本座長）
- 先日、クルーシュテン博士が、「アジャイルは単に廃れつつある流行語なのか？」というセミナーをされ、アジャイル開発のスイートスポットを探っている。（平鍋委員）
- 当研究会報告書では、「スイートスポット」の位置を表出化するために、開発・保守ドメインを、下記に説明する、４つのサブドメインに分割することを提案している。非ウォーターフォールが求める「（対象システムの不確実度）－（平均的な不確実度）」と「（平均的な複雑度）－（対象システムの複雑度）」をそれぞれＸ軸とＹ軸にとると、第１象限から第４象限までの４つのサブドメインが生まれる。これらサブドメインに、研究会で提示された 19 の実例をプロットすると、各事例のもつレーダーチャート特性差が表出化できるとともに、各サブドメインでは、そのなかに入る事例におけるスィートスポットの位置が、ほぼ似た位置にあることが分かる。（松本座長）

(3) 非ウォーターフォール型開発の品質確保のあり方

- 私が少々気になっている記述は、第一回会合の資料２で、

  --

  アジャイル的開発の品質・信頼性の確保の問題、適用にあたっての情報不足、契約面の課題、上長の理解が得られないなどにより、適用が限られている。

  --

  非ウォーターフォール型開発では、ニーズへの対応が迅速に行われるが、品質問題が担保されない。

  --

  というのが、どこから出てきたのかが不可解でならない。（大槻委員）
- 昨年の 12 月初旬より、委員をはじめ、いくつかの企業・団体・大学を訪問し、意見を拝聴した。その中から、当面重要と思われる課題として第一回会合の資料２の５つの課題を挙げた次第である。（伊久美さん）
- つまり、インタビューで意見をきいてこられた中に、「品質・信頼性の確保の問題」「品質問題が担保されない」というのがあったということ。おそらく、そういう意見を言われる方々に対して、正しく普及・啓蒙活動をしていくというのが、今後の施策として効果的だと思う。（大槻委員）
- 信頼性向上の工夫に対してはその効果を測ることができない恐れがある。（山下さん）
- 信頼性の向上は、サービスの安定性をもたらし、だれもが安心して、当社のサービスを利用出来る様になり、無停止継続時間、サービス利用時間の伸び等で、その効果は、幾らでも計測出来る。（田澤委員）
- 少し前に、証券の誤取引が問題になったが、当初、想定しなかったような品質要件が、

さまざまな環境要因によって持ち込まれる時代になってきた。私たちの問題に照らしていえば、まさしく、要求定義をきちんとかけなどいという時限を超え、環境変化に迅速に対応する必要性が、急激に増大していることを象徴する出来事であった。(松島委員)

(4) 非ウォーターフォール型開発への顧客参画のあり方

- アジャイルはいまひとつ、スポンサー(ビジネスステークホルダ)に説明しづらいことがあり、何とかしたい。(田澤委員)
- 最近の身の周りでは、ビジネスステークホルダーの IT に対する理解を深めてもらうことが第一歩かと感じている。なお、最も効くのは成功事例とそのメンバのアサイン、ゆるぎないコミットメントであったりする。(稲村委員)
- 論文やウェブサイトで観察している範囲では、アーティストや画家のコミュニティと似た側面があり、「自由に絵を描かせてくれる顧客と結びつく必要がある」、「いったん、顧客がうるさいことを言い出すと決別する」、「下手な絵描きが増えてくると、コミュニティから逃げ出す」など、繊細な特性をもつ集団であるとも感じている。(松本座長)
- 自分たちで手を動かす必要のある企業は、非ウォーターフォール型開発に自然となっていく、なっていかないとやっていけなくなっている様に、多くの事例を見て思う様になった。従来の WF 型が良い、と思う、固執するのは、自らが手を動かさない方々なのではないだろうか?丸投げするユーザ、下請け構造上位のゼネコントップなどである。(田澤委員)
- ユーザ側が、仕様を決めないのは怠慢ではないか?と、ユーザ側にいる私でも、いまだに思うことがある。でも、実際決まらない。あとは作る側(システム側)も、あとから出てくるアイディアがある。それも反映したい。(田澤委員)

(5) 契約に関する課題整理

- ウォーターフォールが建前になってしまう原因の多くは、コンプライアンスとか、法務や契約の問題が多いと思われる。(大槻委員)
- 請負契約とアウトソーシングは、原理的になじまないはずである。にもかかわらず、安易にも、単なる外注までもがアウトソーシングと称され、また SI などという都合のよい用語の普及によって、あたかも発注者はなにもせずに、外注できるかのように誤解されてしまった。(松島委員)
- デンマークが政府調達の契約方法として「繰返型開発契約 I03,04,05」というのを準備している法律家と会った。もらった資料は、アジャイルとウォーターフォールのコンセプト説明、契約に際する課題の整理、そして、契約の雛形からなる。デンマークでは、政府調達契約は非常に透明性の高いもので、(開発手法を宣言したり、体制を明示したりする。) 1 つの例として興味深いものである。(平鍋委員)
- 大規模システム発注者から小ベンチャーまでが無理なく移行できるような、なんらかの(要求の不確実を前提とした)契約方式を皆様方に生みだしていただき、行政指導の一翼に貢献できれば、たいへん有り難い。(松本座長)
- 非ウォーターフォールが、国力増強という観点から何を果たすべきかという事であろう。(広瀬委員)

- いろんな意味で、とかくＷＦｖｓ非ＷＦの対立軸になりがちなこの話題について、デンマークの雛形契約は契約上の問題整理でも参考になる資料だと思う。（立石さん）
- アジャイルソフトウェア開発のための契約方式は、私の一つの研究対象であるが、最近、ウェブ上で見つけた以下の文献がとても参考になった。
  http://agilesoftwaredevelopment.com/blog/peterstev/10-agile-contracts
  （前川委員）

- アジャイルプロセス協議会では、以下のような考えに至っている。
  - 請負契約では、体制、金額、期間等は記載するが、詳細のプロセスや成果物の縛りがないようにする。つまり、アジャイルプロセス以外の工程や中間成果物の約束をしない。
  - 詳細機能実現の約束をしない。むしろ、重要な機能（要求）が実現できない場合にのみ契約不履行となる「赤点契約」とする。
  - 発注側が要求や検収に迅速に実施できる協働関係が築ける契約。
- 我々技術領域の人間ができることは、非ウォーターフォール型開発に関する契約雛形作成のための要求仕様をつくることである。（大槻委員）
- IT にかかわる契約は原則、出来高払い方式の契約にし、仕様が明確なものに限定して、確定金額契約を採用するという、方向を確認しておく必要があると思う。
- 発注側が真剣に、品質と納期を考えればアジャイルに行きつくしかない。（松島委員）
- 大手メーカ社の方々が動いてくれないと始まらないと思う。（田澤委員）
- 先斗町などへの接触があった。これは要求が不明確な接客業のひとつであるが、料亭、宴席計画者、お茶屋さん（芸妓、舞妓、などを舞、笛、鼓、琴などスキルごとに育成し、合わせて生活の場を提供している）が微妙な連携を保ちながら、外国公人から下々の企業人に至る、さまざまな階級の客に応じていた。なにか、業界にとらわれず、自然に体現できるような実例を求めている。（松本座長）
- アジャイルプロセスの場合には、初期構築だけの考慮ではだめで、保守（訂正、改善、拡張など）のライフサイクル全体にわたるものを契約の対象として考えるべきだと思う。（大槻委員）
- 成功ビジネスモデル形成の実例、リッツカールトンの「おもてなし」とか、「おまかせ」についてである。このあたりは＜クレド＞のようなものを、ソフトウェアの世界でも設定できるといいのではないかという議論をしている。
- ウォーターフォール型に見えるけど、アジャイル的なものを入れて旨く行っているなと感じたのは、10 月末に開催された IPA フォーラムの 7&i さんのご発表である。初段で膨大な仕様書を作って発注したようだが、ユーザインタフェース部分などは、現場の店舗担当者（アルバイトのお兄ちゃんもいる）が、実際にいじってみて大丈夫かという確認をし、だめならそれをベンダ側に手直しをさせるという（検収の）合意形成のプロセスを決めて対応したといったようなことをおっしゃっていた。（大槻委員）
- あるベンダが定額請負契約でアジャイル開発を行い、いつまでもユーザが情報システムの完成を認めなかったために、延々とイテレーションが続き、大赤字になったという話を聞いたことがある。（前川委員）
- オフショア関連の研究をされている米国の大学の先生にきいたところでは欧米（米国？）の 20%くらいが Fixed Price で、のこりはほとんど Time & Material と聞いたように思う。

- Time & Material にするということは、受注側には最終的な完成義務はなく、完成に向けての善良なる管理義務のようなものが生じるだけである。よって、発注側の責任が大きくなる。もちろん、信頼関係がないと成り立たないだろう。（馬嶋委員）
- 民間業者による time and material contract 定義だが、一応参考になる。（松本座長）
  http://www.businessdictionary.com/definition/time-and-materials-T-M-contract.html
- 発注側企業、受注側企業、下請け（もしあれば）、開発者個人などの構造において、雇用の関係まで踏み込まないとこの課題は解決しないのではないか、と思っている。（南委員）
- 今年、ノーベル経済賞を受賞したウイリアムソンの貢献は、内部組織と外部組織の違いからの会社とは何かであるか、この考え方は、内製か外注化の意思決定、さらにアウトソーシングに関する理論的根拠を提供することになった。信頼できる業者を探し、育成し、そこに、出来高払いの契約をすることだと思う。（松島委員）
- ノールウェイの PS2000 という標準契約が示されていて、我々にも大変参考になると想像する。（松本座長）
- 確定金額方式にいろいろなオプションを付加し、両者の調整の余地を残すという形式は理解できる。ただ、米国のように、契約でことをすべて律するというプラクティスと日本のように、本来的にあいまいなプラクティスを、それこそ善良な管理者義務と誠実さでカバーするという方式では、簡単ではないと思う。恣意性が結局入り込んでくるしかないだろう。海外のいわゆるバリューベーストという、利益を分配する方式やＳＬＡにように指標を設定してモニタリングをするのも、両者に多くの管理工数が発生し、実施が困難に思う。（松島委員）
- トヨタ系列ではどんな契約方式かを確認したところ、基本契約で基本的な事項と単価を設定し、数か月の注文書がくるようである。そしてかんばんで納期調整をする。これをアジャイル風に理解すれば、ＳＥの単価、あるいはもうすこし大括りなイテレーション単位、モジュール的な単位での単価を設定し、イテレーションごとに、発注者が注文する（両者捺印ではなく、一方的な注文）というのが日本では、実施しやすいように思う。（松島委員）
- JISA 浜口会長の説明。（前川委員）

  1. 欧州における情報システム開発の現状を調べたところその契約の 7 割(8 割だったかも）は、Fixed Price 契約ではなく、Time and Material 契約である。
  （注）Fixed Price 契約は、日本の定額請負契約に相当し、Time and Material 契約は準委任契約に相当する。（ただし、浜口会長は Time and Material 契約は派遣契約に相当と説明）
  2. Fixed Price 契約でない理由は、Fixed Price 契約ではあまりにもリスクが大きいからである。
  3. 欧州では、ユーザ企業（浜口会長は「クライアント」と表現）が、責任を持って主導的にシステムを開発しており、ベンダはこれをサポートする役割である。
  （つまり、インハウス開発、とすると、やはり契約は派遣契約か？）
  4. 欧州の関係者は、情報システムの要求仕様が最初に決まることはなく、要求仕様は時間とともに固まっていくものだと考えている。
  5. 日本の情報システム開発も変化の時が来ているように思う。

・ JISA 浜口会長の説明。私は 80 年代に、シーメンスおよびスイスの ABB 社がソフトウェア工場を立ち上げるとき、計画策定の指導をしてくれという依頼を先方から受けて、何回かに亘って渡欧した経験があり、そのときのことで関連する事実（ソフトウェア工場を設計する前提条件）を簡単にご紹介します。（松本座長）

> １．欧州では、（具体的な国を挙げるのは避けたほうがよいが、）周辺の中東地域からの季節労働者が溢れていて、労働市場が流動性に富んでいる。
> ２．技術者資格認定制度がしっかりしていて、技術者の企業を渡っての移動雇用を可能にする基盤がある。
> ３．ビジネス向け IT システムが、日本のようにサイロ（他人には untouchable であるという固定観念）化していなくて、SAP のような準汎用ビジネスパッケージをできるだけ活用して、流動労働力を最適活用することがベストであるという観念が強い。
> これらの前提条件下では、個別プロジェクトは、必要に応じて、人を追加雇用したり、解雇できる利点がある。したがって、fixed price 契約というのは、ベンダにとっては、そうでない人月ベース契約に比べてリスクが大きいと考えられる風土がある。
>
> 保守的な京都においてすら、もっとも伝統的な企業が社内経営情報システムに、SAP を導入した。次第に、ビジネス IT システムを神聖化する意識は薄れていくのではないか？
> おそらく、AgileJapan2009 で紹介されている良品計画社の例のように、店舗レジ、物流管理、メーカ EDI 管理、勘定システムのような流動労働市場からでも調達できる部分と、そうでない内製管理システム部分を分離して、後者は、サイロ、かつアジャイルに、できるだけフラットな CI 環境（たとえば Hudson）の上で構築するという方向が今後志向されるではないか？（松本座長）

・ 契約問題の棚上げ（軽視？）IEEE の論文集で、契約方式、価値問題、コストを扱ったものはそれぞれ 1 件しかなかった。（松本座長）
・ 外注しない自前開発なので、契約というものが必要無いのであろう。（田澤委員）
・ 例えば、契約問題と RUP や契約問題とウォーターフォールを扱ったものはあるのであろうか？私は契約問題と技術問題（プロセス問題）は、結構どこでも難しい接点ではないか、と思っている。（平鍋委員）
・ 契約問題に限定したワーキンググループをつくってたたき台を作成するというような進め方を考えてよいのではないか。（前川委員）
・ 現状の請負契約のアジャイル向きのものを策定していくことのみならず、「制度設計」という少し広めの設定をした方がよい。これは、例えば、官庁の入札制度を視野にいれて改革をしていくべきだという認識である。（大槻委員）
・ アジャイルプロセスでは、従来のユーザ／ベンダという対峙以外の関係や、新しいステークホルダも視野に入れていかなくてはならないので、「スキル」の検討と同様に、既成の民法、商法、社会的慣習とは切り離して、産業振興に効く組織間の関係や契約について、ゼロから見直してみる必要がある。（大槻委員）
・ 既存の法令や公序良俗に反しない限り、典型契約以外のタイプの契約を結ぶことも可能である。たとえば、「ソフトウェアの使用許諾契約」は典型契約外の契約である。し

たがって、アジャイル開発に適した契約モデル、契約書のひな型、利用の手引などを新規に作成し、それを普及させることは可能である。（前川委員）

- 工事進行基準の実践には、非ウォーターフォール型開発として問題がないだろうか。（松本座長）
- アジャイルプロセスとしてイテレーションを細かくとり、かつ、多くのプロジェクトでは、実質的なプロジェクトデータを採取していくので、工事進行基準は、問題ないという意見が多い。ただし、官庁案件で推奨されている工事進行基準を EVMS(Earned Value Management System)の方法で管理しようという動きに対しては、懐疑的である。出来高で管理するというそれ自体はよいとしても、当初計画が変更されることを、積極的に織り込んでいく必要があると考える。（大槻委員）
- ウォーターフォールモデルに比べ進捗の把握が正確なので、仮に、売上総額が決まっている案件であっても問題は少ないであろう。業界では、工事進行基準より、むしろ国際会計基準(IFRS)がどういう影響を与えるのかが話題になっている。だがこれも、SEC より、他の組織で議論するような問題である。（前川委員）

(6) 我が国のソフトウェアの国際競争力の強化

- 最近、日本の新幹線が世界中に出て行きそうなことが伝えられている。もちろん、ソフトもついてゆくのであろう。やっぱり考えるべき舞台は世界だ！と、思う。（神谷さん）
- 国内向けの開発はウォーターフォールを理想とする開発の仕方でもいいのではないか、と思う。海外向け、または海外企業と競争する国内向けサービスとかの領域で、Agile 開発が必要と思っている。（馬嶋委員）
- IPA SEC として、国が考えるべきソフトウェア領域というのをなにかしら定めて、それに必要な技術、プロセスなどを検討していく、というなかでの本研究会活動にならないものであろうか？（馬嶋委員）
- ★開発のサービス化★を目標にするとよいと考えている。クラウドの従量課金サービスのように、開発サービスを提供できるようにすれば、世界中どこにでもグローバルにも展開できるようになるだろう。（大槻委員）
- 私のいる事業部では、インドベンダへのアウトソースを活用している。インドベンダのリソース確保能力は高いし、（文句は言われるが）国内に比べて増減の融通は、はるかにききやすい。インドベンダがまさに、そういう「開発のサービス化」をグローバルに実現している、と思うのだが、いかがであろうか？（馬嶋委員）
- おそらく、これだと人工提供的なところに留まってしまうだろう。私が、＜開発のサービス化＞でイメージしているのは、もっと高度というか、ドメイン特化型のものである。例えば、新幹線の輸出などに伴って、システムやソフトウェアをいっしょに持って行く場合には、運行管理システム、列車制御システムといった日本のノウハウがつまった部分を現地側で構築・運用していくところを＜開発のサービス化＞として提供するといったことである。（大槻委員）
- なんでも開発する型というよりは、交通運行管理システム開発サービスのような、ドメイン特化で、例えば、列車などを持つ企業と連携しながらビジネスしていく、のような感じとイメージした。（馬嶋委員）

・ ユーザ企業は、なにを内部に残し、何を外部に依存するかの議論、あるいは戦略なしに、コスト削減目的で拡大してしまった。
１．重要なノウハウは内部に残し、プログラミングなどのローテク、ローコスト領域を外部に依存。
２．専門的な IT ノウハウ、スキルは習得に時間とコストがかかるので外部に依存。
内部は、企業経営にかかわる業務スキルを中心に整備する。
もちろん、第３に、全部、外部に依存するというのもある。（松島委員）
・ SEC では機会をとらえて国際会議に参加したり、国際会議の日本開催を計画したり、SEC BOOKS の一部の英訳を進めたり、あるいは国際標準化活動に委員を送ったりと、多少の努力はしている。（神谷さん）
・ 日本の産業が勝つのは「最終製品」とし、アジャイル等の方法論は道具としてしまい、むしろ、この産業構造、文化背景、のなかで、日本のソフトウェア産業の育成方法⇒最終製品としての価値を出すためのソフトウェアのあり方、として考えることとなるように思う。今期の活動は、空中戦を避けるための、まず、その「地上絵」（マップ）作りを行なう。そして、最終ゴールを情報サービス産業の国内での安定と海外への展開へ向け、その上で、マイルストーンとして上記の「例えば」のようなことを優先順位づけ、具体的なサブ活動に分割して、来期への提言を行なう、というのが、３末の姿ではないかと思っている。（平鍋委員）
・ ソフトウェアの国際競争力に関して、関連業界団体が共同で試算した輸出入額（2000 年まで）で計算すると、限りなくマイナス１に近い。つまりまったく競争力がない。（前川委員）
・ 1990 年代から、モジュール化＋オープン化、それにエクイティ中心の金融環境がつくる世界は、日本の産業界にとって不得意科目だったように思える。新時代になって、いわば仕切り直しとなれば、日本得意の局面も出てくるのではないであろうか。（神谷さん）
・ 計量経済学や組織論（産業アーキテクチャモジュール）といった観点は、次世代を見通すための一つのコンセプトツールとして有効だと思っている。このような新しい世界で日本が得意な局面を探して行くというのは元気がでていいだろう。（前川委員）
・ カナダは、米国のソフトウェア産業に対抗するために、米国ではできない領域を見定めて、3 次元動画技術をトロント周辺に誘致、育成した。日本もこういった（グローバル対抗の）戦略があると良いと思う。既存の（基幹）産業領域を守る話と、新規領域を開拓していく話とは、分けた方がよいだろう。（大槻委員）
・ 「情報」を処理する基礎技術自体の研究・開発という観点のテーマが必要である。ブレークスルーすべき技術、海外勢に遅れをとりつつある技術があるのか、無いのかという観点（議論）が必要である。（広瀬委員）
・ ソフトウェア業の将来像に関して、「激しい設備投資競争とグローカラゼイション（globalization-&-localization）によるボリュームゾーン戦略」、「擦り合わせ集積の危機（トヨタの米国企業から調達する部品で起きた失敗の如く、擦り合わせはグローバライゼーションを狙う企業では通用しないなどのこと）」、「ブラックボックスとオープン化」などについて議論をしたい。（松本座長）
・ 世の中の流れとしては、モノから離れた「情報」をバーチャルに扱うソフトウェアシステムから、モノの動きを司る部分も含めた全体システムへと重点が移っている。モ

ノの動きを司る全体システムをグローバルな競争領域として、国をあげての売り込みをしていくというのが流れと理解。そのようなシステムには、　必ずソフトウェアが重要な位置を占めるはずで、そのソフトウェア開発を成功させることを SEC のミッションの一つとする。また、この領域のソフトウェア開発の現状とあるべき姿を今後、より深く検討して、Agile 開発を含む Software Engineering 手法の適用を図り、開発を成功に結びつけることを SEC に提言する（馬嶋委員）

- いわゆる「ソフト産業」という範囲を実態に合わせて、再定義(再整理）すること広義に理解した。それらの必要性・必然性が分かりやすくクリアにできればいい。（広瀬委員）
- 各企業の経営者の方々が、戦略立案のベースになるようなものを作れると理想的である（大槻委員）
- 信頼性や安全性といった画一的なシステム要件を掲げていくことよりも、品質を含めてソフトウェアへの要求が社会的コードによって変化していくという前提で、戦略を検討していくというのが正しい姿だと思う（大槻委員）
- SWEBOK へ向けて、日本からの貢献が期待される。（松本座長）
- IEEE から SWEBOK のとりまとめを請け負ったグループのリーダーである Abran 教授によると、SWEBOK 作成に日本人は関わっていない。日本としても、こうした動きの継続的なフォロー、積極的な参加、日本でのアクションの反映は重要であると考える（神谷さん）
- PMBOK, BABOK, APMBOK などの BOK を、権威付けた知識体系にする意義がそもそもあるのかという議論も必要である。開発プロセスそのものを知識体系として整備する意味は本来無いはずである（大槻委員）
- 日本、あるいは日本企業にとって都合の悪い規格が国際標準になってしまって、大変なことになるという可能性はあるのか。もし、それほど悪影響がないのであれば、まずは、国内産業の問題を優先すべきである。（前川委員）
- 標準化は標準の成立にかかわった企業の技術が優れていると評価され、他社がキャッチアップするまでの間、強力な競争優位を獲得する。その一方で、一定の技術を確立したが標準が採用にならなかった企業にとっては、標準の吸収、移行に関する膨大な開発コストが必要となるばかりか、これまでの市場と顧客を失うことになる。これをもとに、さて、多くのコストをかけて標準の成立に努力することが得策なのかどうか。それとも、決まった標準を、迅速に受け入れる方が得策か。（松島委員）
- ISO9000 や 14000 のように半ば強制されるような国際標準の場合は、要注意である。企業の競争優位に影響を与えるような国際標準の場合には、日本企業が不利にならないように、積極的に標準化活動に参加する必要がある。今議論しているものはそのどちらかに当てはまるのだろうか。（前川委員）

(7) 技術者が生き生きと働ける環境作り、優秀なソフトウェア技術者が集まるソフトウェア産業へ変革

(ア) アジャイルの良さ

- 「気持ち良い」というのは、主観的なものであるから、何とも言えないが、アジャイルプロセスでは、ファシリテーションによって、気持ち良く開発（エンジニアが生き生きしているという意味）をしている事例である。（大槻委員）

- 私がこの会に出席している動機は、日本の優秀なエンジニアのライフスタイル革新（いつまでも 3K とか言わせない）と、プロジェクト成功率の向上（ビジネスステークホルダと顧客の満足）。その 2 つを、AND で取ることである。（平鍋委員）
- エンジニアは、もっと楽しいはずだし、実際一度楽しい開発を経験してしまうと他の職は出来ないくらい楽しいはずである。（田澤委員）
- いちばんは信頼関係の構築である。（田澤委員）
- プログラミングは、本来、レゴやおもちゃの組み立てみたいに楽しいものである。（松島委員）

(イ) 人材

- 米国と日本では、システム開発における理想形が違うと感じている。（馬嶋委員）
  - ✓ 米国：ビジネスに重要なユースケースを理解し、それをもとに、アーキテクチャ設計、Code 作成、初期テストを、高速なスピードでこなしている。
  - ✓ 日本：ビジネスに重要なユースケースを上流エンジニア(コンサルタントも含む)が設計し、それを SE が機能設計して、その機能をリーダーがコンポーネント分割して、サブリーダーが何人かのプログラマを使ってコンポーネントを仕上げている。
- 日本の形態は、マネジメントのしやすさ、個人能力差によるアウトプットの平準化などメリットがあるが、米国と違いユースケース(ビジネス)と最終成果物である Code との間が遠いので、本質的な高速スピード、イノベーションが生みだされにくいだろう。（馬嶋委員）
- ある企業で、業績悪化のために、社長がアジャイル的な理解も示しながら外注せず内製化しろといった例があるが、外注慣れしたＳＥは、コーディングができない。入社以来、本番用のコーディング経験がまったくないのである。（松島委員）
- 平準化すると競争力は、なくなってしまうのではないのだろうか？当社は SIer 側ではないので、アウトプットの均質化は、あまり望んでいない。（田澤委員）
- 「コーディングなんか誰でもできる仕事だ」とか「コーディングは設計書どおりにプログラムを書くだけだ」という SE がいるのも事実である（前川委員）。
- （コーディングが全くできない SE がいる元請け会社が多いという件について）ユーザ企業も、しっかりしないといけない。そういうところに発注しなきゃいいだけである。（田澤委員）
- 現場にとっても新人本人にとっても、とにかく戦力化することが必要である。「できる人」が指導するため、「できる人」のパフォーマンスが平準化される。
- 人毎のパフォーマンスの違いは、なるべく見ないようにするか、できない人をできる人が如何に補って全体のパフォーマンスを上げる。（馬嶋委員）
- Google って新卒とかそういう軸で採用をしていない。私も新人という採用枠があるのは反対である。〜〜が出来るから採用される、とあるべきだ。新卒新人を採用するのは、労働集約型の仕事のスタイルだからではないかと思う。（田澤委員）
- IT 産業(特に SIer)は、知識集約型のはずなのに、あいかわらず労働集約型である。某米国 IT ベンダの方にアーキテクトは育成するのか？」と質問したことがあるが、「育成なんてしない。だって、アーキテクチャ設計ができる人を採用するから。」という単純明快なご回答であった。（馬嶋委員）

- 人材を育成するのか、できる人間を採用するのかと。能力を高めるのは、本人の責任のはずという基本が、おざなりになっていたのかもしれない。（松島委員）
- CACM の J.Kramer の小論"Is Abstraction Key to Computing?"によると、抽象能力は先天的で、高度の抽象能力を持てるようになる人は 30%～35%なのだそうである。つまり、育成も必要だけど、選別もしなくてはならないということである。
- 手順化できて、属人性を排することができる領域は労働集約型でもかまわないと思う。（大槻委員）
- 普通科の 3、4 年生あたりは、まずい。面接に出ていても、ひとりも採りたいと思わなかった。工専卒、工専から大学への編入生は、ほんとうに良く出来る。（田澤委員）
- 地方の高専が苦戦している。IT 業界は売れ残る。雇用問題が、日本の重要課題である。（松島委員）
- 技術者(SE/PG など）および IT 業界のモチベーションをどう高めるべきか（下がらないようにすべきか)。不確実な時代に対応できるような技術者を育成するにはどうすべきかを考える必要がある。（南委員）
- 現実に、「他の企業で育成された人しか採用しない」、というのであれば、産業界全体では成り立たない。（神谷さん）
- 大手企業の方からも、新しい先端企業の方からも口を揃えて、人材の不足を聞いた。いづれにしても、大学は、ちょっと変わる必要があるだろう。いわゆるアカデミックポイントを追うということと、産業界のニーズに直接応えてゆく、ということは違うことのように感じている。（神谷さん）
- Google や、メジャーリーグは、優秀な人しか取らないという側面があるが、優秀な人が、そこで活躍したい、働きたい、と思っている方が強いと思う。そういう人たちが集まってくる様になれば、ルーキーという採用はしなくなる、というのは自然な流れの様に思う。（田澤委員）
- エンジニアは往々にして「もっと自由に動きたい」「責任範囲を広げたい」と考えているようである。（稲村委員）
- 当社では、既に昨年、インド、中国の新卒（中途ではなく）を採用している。その理由は、優秀さと、バイタリティーである。当然、ダイバーシティもある。システムはある程度コピー出来ても、商習慣、生活スタイルは、日本人が勉強するよりも現地の方の方が、当然一日の長がある。（田澤委員）
- グローバルビジネスをするためには、組織内が当然グローバルになることを求められるわけで、そういう意味で参考になる。（馬嶋委員）
- モノの動きも司る新しい領域では、なんらかのイノベーションが必要で、そのためには、「自ら考えて行動する生き生きとした」ソフトウェアエンジニアが多く必要。それを実現する/育てるには Agile 開発が最適である。（馬嶋委員）
- いろいろな役割を持つエンジニアやコンサルタントを、既成の認定や資格制度とは切り離して、日本独自の人材配置や育成プランをたてていくというのであれば、すごく意味がある。「知働化研究会」では、システム・エンターティナーとか、インタープリター（ユーザとベンダとの間の翻訳）とか役割に関する議論が活発である。（大槻委員）

(ウ) 産業構造

① 開発の平準化

・ 今日のようなソフトウェア産業の構造ができたのは、企業のソフトウェア開発に大きな山谷があり、要員の山谷が避けられないから、歴史的にこうなった、と認識している。常時開発を、それこそ平準化し続ける方式、そして内製方式、というのが考えられるが、毎日少しずつシステム更改、というようなことが技術的に可能だろうか。皆が、開発を平準化し、自社要員による内製体制をとった場合、現状のソフトウェア産業は形を変えざるをえなくなる。（神谷さん）
・ 技術的に可能だと思うし、可能にすべきである。そうしないと競争出来ない。同じ土俵にさえ上がれない。また、産業は形を変えざるをえないと思っている。（田澤委員）
・ 業務システムを、それこそ良いアーキテクチャで設計すれば、バックログをためないように、タイムリーに、日々アップグレードして、連続的に更新してゆける。開発要員は平準化し、あとは自社内に専門職制度を整備すれば、自社要員を抱えて、恒常的に内製型で開発業務を進めてゆける。こうなると、日本のソフトウェア産業の構造は、変わらざるをえない。（神谷さん）
・ 開発者の作業リズムが平準化する。勉強時間も紡ぎ出せるので幸せである。（田澤委員）

② 産業構造の歴史
・ 日本の産業構造が生まれた経緯と歴史を押さえておくことは、そこに働く慣性力 （およびその意味）を理解することに繋がり、ひいては変革のヒントになると思う。（平鍋委員）
・ ユーザ企業の情報システム部門が独立するケースや、SI としての資本力強化による統合などは、まわりまわって、システムのアーキテクチャやプロセスに影響を及ぼす。コーンウェイの法則というのであったか。アーキテクチャの構造が、それに関わる組織の構造に対応してしまうというのもある。（大槻委員）
・ 日本の IT 産業(EDP->SI)の成り立ちについては、結構な量で読むのが大変であるが、以下の文章（日本 IT 書紀）が面白く読めて、かつ参考になった。IT なのに江戸・明治から始まって 1980 年近辺までで終わる、というあたりが斬新である。
http://itsyoki.justblog.jp/blog/archives.html
（南委員）
・ 産業構造に考えをめぐらすことはものすごく重要なことだと思っている。ソフトウェア産業以外の分野は、ここ 10 年スパンでみれば、ものすごく新しくなっている。（神谷さん）
・ そういう、想定ユーザと具体的な成果が思い浮かばないと、SEC 研究会活動もやりがいが薄くなると感じる。（馬嶋委員）
・ 業界全体の変化が、開発方法論の変化として現れていると仮定した場合、開発手法の変化（改革）を通して、我々が着手すべき事柄がクリアにできるのではないかと思う。すなわち、変化を強いられて困っているポーションにより多くの教訓があると思うので、ヒアリングもピュアなソフト業界だけでなく、他業界のうちソフトの開発（含むサービス）を強いられている部分について、ソフトの目で、ヒアリングしてみればいいのではないかと思われる。（広瀬委員）

③ 産業構造変革の必要性

- 多重下請けをやめて、いきなり末端の企業に全部依頼しても、現実には末端企業には組織機能的に欠けているところが沢山あり、すぐには依頼に応えられない、というのが実情だろう。それから、システム構築にはファイナンス的な問題もある。まさしく、ゼネコンはシステム完成までのファイナンスをしている側面があり、大手インテグレータは、自身が金融業の性格を持っている。多重下請け構造をなくすには、こちらの金融的な工夫も必要であろう。（神谷さん）
- 私が気になっているのは１点のみ。日本式多重下請け構造のままで、世界と戦えますか？ということである。（田澤委員）
- 日本式多重下請け構造は変革の必要があると思われる。多重下請け構造が、きちんと価値を生んでいるのであればいいが、「丸投げ」的な請負契約、あるいは派遣契約に近い「偽装的」請負契約が多い現状では、あまり付加価値を生んでいないように思う。（前川委員）
- 徐々に産業構造を好ましい方向に変えていきましょう、なんてことでは、なぎ倒されるだけ、ということであろうか。（神谷さん）
- 内製指向（あるいは Could/Inhouse/Agile と表現）、もしくは、「限りなく全体最適に近づく分業」。ある企業があるサービスを行なう場合に、IT を中に抱えるか、外に出すか。ソフトウェアというのを手段とみるか、「重要な手段」と見るか。産業を横に切るか、縦に切るか。産業をまたがって重要になってきている「ソフトウェア開発」を、共通のモノとして、それを、強みとして提供する、というやり方もあるだろう。SIer の仕事。それには新しいタイプの契約が必要であろうが。私は、まだどちらもある、と思っている。未来の産業構造は、次の２つの形のどちらかであろう。（平鍋委員）
  - ✓ インハウスでアジャイルが活発化する。ユーザ企業が開発パワーを持つような形になり、受託開発、というものや、SIer という業態がなくなる。
  - ✓ 契約問題が解決する場合、ユーザ企業がアジャイルでの開発を開発会社に発注する形態がとれる。そして、スキルの高いエンジニア、チームのモチベーションとコミュニケーションをうまく作れるマネージャを持つ開発会社が、より高い競争力を持つようになる。
- 私は別に内製指向ではない。ただし、外に真の協力者が見あたらないのは事実かもしれない。それ以上に、メンバが育つといったメリットがあるのが内製の魅力と思っている。（田澤委員）

④　コンサルティング

- 実際に新しい試みにトライする企業群、あるいはプロジェクト群が、まさか講演を聴いて、報告書や本を読んだだけで、突撃するのは無理だろう。そこで、次のプロセスとして、開発企業を選ぶことのほかに、信頼できるコンサルティング企業の知恵をリーズナブルなコストで、いわば知恵を購入できる環境が整えば、あたらしいムーブメントが加速すると思う。（神谷さん）
- 発表の中で、米国コンサルタントと日本の位置づけの違いなどについて話があり、とても注目している。何か新しいことをする際のコーチ、指導役としてコンサルタントが就任、雇用されるのは良いと思う。（田澤委員）
- ソフトウェアを第六次産業化するために必要なプレーヤ構成（構造）のあり方（世界を引率できる）を議論していただけると有り難い。（松本座長）

・ 米国の場合、ほぼ、コンサルタントが「知の運び屋」になっている。現場を渡り歩いて、それを形式知化して本を書き、それをまた実践検証していくような感じである。実はアジャイルは one-size-fits-all ではないので、人でないと、知識を運べない。したがってコンサルタントは重要である。また、コンサルタントを地位として認める産業構造が重要である。（平鍋委員）
・ IPA SEC 活動として、コンサルティング面を出口とする戦略は十分ありだと思う。Agile 開発にかぎらず、いろいろ書籍化しているノウハウも本当に普及できると思われる。（馬嶋委員）

(8) 技術者の個人成果、および事業成果の評価のあり方

(ア) 個人評価

・ 『日本の形態は、マネジメントのしやすさ、個人能力差によるアウトプットの平準化などメリットがある』の意見に対して、SIer 側はアウトプットの均質化が重要な側面かもしれないが、ユーザ側は、すごく面白いものを作ってしまうか、おそろしく早く安く作ってしまうか、そういうところを望む。（田澤委員）
・ 自分のスキルアップが、ビジネスの価値になれば、開発者は（私は）喜んで勉強して良い仕事をする。会社は社員のためを考える。社員は会社のためを考える。の好循環が回るような構造にしたい。受託開発ではそれが難しいのではないか。（平鍋委員）
・ 仕事の目標を何にすえるかで人の成長、成果は明確に変わるものだと感じている。（稲村委員）

(イ) チームの成熟度

・ チームの成熟度が議論されない。もともと、個人評価とか、explicit 化とか、資産化とか、文書化などを忌避する集団によって支えられているためか？（松本座長）
・ そういう面はあると思うが、浅いレベルで考慮、実践している集団の意見なので無視して良いだろう。チーム熟成度に関しては、教育効果が言われているので、議論の対象になっていると思われる。（田澤委員）
・ 伝統的な意味での個人評価、explicit 化などを忌避する集団なのだと思う。（大槻委員）
・ 成熟度=CMM というわけではないが、アジャイルは CMM(I)をバッシングすることがある。これは、成熟度の計測と取得がビジネス化してしまい、成功しているチームとそのレベルがほとんど有意な統計的関係を持たない、という主張。もう１つ、改善の観点を含まないプロセスが死んでしまうことと、上位レベルにしか改善の観点がない CMM から、3-5 を薄く横断するプロセスとしてアジャイルを定義する見方もあるようだ。逆に、Watts Humphrey 自身はアジャイル擁護的な発言があり、ラブコール的に見える。

(9) 非ウォーターフォール型開発の普及

- 開発手法については、手法自体の良さをいくら主張しても限界があって、それによるBig な成功例が必須である。トヨタのリーン開発についても、トヨタの成功があってこそ、あれだけのムーブメントになった。（馬嶋委員）
- 日本からの情報発信、日本の貢献を考える前に、まず、（アジャイル開発を含む）非WF 型開発の普及促進を考える必要がある。非 WF 型開発が定着、発展していけば、自然に日本からの情報発信もできるし、ソフトウェア工学上の貢献も可能である。（前川委員）
- 非 WF 型開発の事例を整理することは、それなりに普及啓発に役立つ。これは継続的に委託調査で進めるとよいのではないかと思う。（前川委員）
- 民間が自分でやれることには、絶対に手を出すべきではないと思っている。（松本座長）
- アジャイルプロセスを普及・促進するための活動を IPA が行うことには意味がある。そのためには、既存ですでに推進してきている個人・組織・団体ともっと密に連携し、IPA でないとできにくい部分に集中して啓蒙・推進していくのがよい。（羽生田委員）
- 日本における IT 産業の在り方やマクロ構造の見直し・転換を促し、より健全で意味のある産業にしていくために、今何をやるべきかは、テーマとしては大きく過ぎる。重要なことはわかるが、別途テーマを設定した別委員会にして、行うべきである。ミクロとマクロの中間くらいの領域として、「プログラマ」の働き方とソフトウェア開発の「契約」の問題にテーマを絞るのがよい（羽生田委員）
- 実質的な IPA でやるべき課題(具体的）を「特定」し、WG 形式で形にする、という方向に同意する。具体的には、契約問題、スキル（教育）問題（含む：コンサルティングロール)、ツール問題を挙げてもらいたい。（平鍋委員）
- 次年度以降の課題設定と WG 方式で進めることに、私も一票投じる。（大槻委員）
- ソフトウェアエンジニアリングに関する国家戦略、国内取引きに関する制度設計と契約、価値・ビジネス駆動のソフトウェアエンジニアリングが WG 候補として挙げられる。（大槻委員）
- 「価値・ビジネス駆動」については、アジャイルプロセスを外から見て（ユーザ側から）のご利益や効用を明確にすることが実践部隊のプラクティスなどの整備とは別に必要だと考えていることと、経営層の意識改革に効く、材料を整備していくのが、産業振興上も効果的であろう。（大槻委員）
- 現在のアジャイル成功事例の整理方法からだけでは、皆さんが持っておられる問題意識が浮かび上がってこないのではと思う。（広瀬委員）
- 国が手をつけるべき課題、民間にまかせるべき課題をあげて、
  - 国が手をつけるべき課題について、最適なやり方 (WG とか専門家への発注とか) で SEC が推進する。
  - 国で手をつけるべき課題かどうかの継続検証など戦略を立て続けるチームを設立する（SEC が負ってもよい)。

  というような骨子が浮かぶ。（馬嶋委員）
- 契約についての WG 開催に賛成である。内製、外出しいずれにせよ、分担し少人数で策定し委員会に FB を求めるかたちで推進する方が機動性は高まる。（稲村委員）
- 「契約」について、今後、検討していくことに賛成である。（大槻委員）

付録 D：　研究会におけるアンケートの結果

研究会における委員相互の認識を共有することを目的として実施したアンケート形式による意見表明の結果を、設問ごとに示す。

No.	設問の観点
1	非ウォーターフォール型開発の適用分野について
2	非ウォーターフォール型開発を適用するための前提条件について
3	非ウォーターフォール型開発に対する信頼性について
4	非ウォーターフォール型開発は保守、引継の困難さについて
5	非ウォーターフォール型開発に適した契約について
6	日本のソフトウェア産業構造のあるべき姿はについて
7	委員各位の共通認識について

1.　非ウォーターフォール型開発の適用分野は限定されるものか？

(1)　非ウォーターフォール型開発を適用することができる分野は、例えばWebアプリケーション開発や社内で活用するミドルウェアの開発のような特定の分野に限定されるものでしょうか、あるいは限定されないものでしょうか。そのようにお考えになる理由とあわせてご回答下さい。

回答

- 得意不得意はあると思うが、限定されない。(以降すべての質問で、非ウォーターフォールという言葉でパッケージ適用、を意味しません。どちらかというとアジャイルに近いものを非ウォーターフォールのイメージで回答します。)
- 限定されないと思います。
  どのような開発プロセスをとるかは、ビジネスの要請と人材や体制などによって決まるもので、ご質問のような「分野」で決まるものではないと思います。
- なんちゃってアジャイルプロセスから、実践的なアジャイルプロセスに移行してきている現状からみて、どんな分野でもアジャイル的な要素を取入れていると予想されます。ビジネスやマーケットとの連動が必要で、ソフトウェアをとりまく要求に不確実性が高い分野では、アジャイル的な要素が占める割合は高いでしょう。特定のアジャイルプロセス型開発方法というのがあって、それがどこかの分野に適用されるという見方は意味がありません。
- ソフトウェア開発手法で、もともと適用分野を問わない、というような手法はなく、まして先進的な手法は「向き」「不向き」があるのだから、手法に沿って適した適用分野があるのは当然。非WF型開発にあたかも適用限界があるということを誘導したげな設問で、現段階では適切な質問ではない。
- 旧来のいわゆる「ウォーターフォール」型は将来とも、何も変えなくていいとは、誰も思わないでしょう。旧来の型を変化させる新しい革新は必須です。旧来型からの進化形を非ウォーターフォールと呼ぶのだと思います。(アジャイルとは呼ばないのは、この為と思料)
- それぞれの立場でのリスクを適切にハンドリングできるのであれば、基本的には非ウォーターフォール開発で OK。後は適用の可不可の議論ではなく、何を優先するか、

メリット・デメリットのトレードオフに基づく判断の問題だと思う。

- 禅問答の様で申し訳ないのですが､限定する理由が無いためです｡そういう意味では、WF 型の開発方法も何かを限定するものではないと考えます。
- 非ウォーターフォール型の開発プロセスの定義にもよりますが、ウォーターフォール型の開発プロセスを繰り返すようなスパイラル型まで含めれば、特定の分野に限定されないでしょう。ありとあらゆる分野に適用できると思います。
  （注）非ウォーターフォール型の開発プロセスとは何かをきちんと議論すべきです。用語を定義せずに議論するのは時間の無駄になります。

(2) 現状広く適用されている分野ではないが、今後適用の拡大が期待される分野はどのようなものか、具体的な分野とその特徴をご回答下さい。

回答

- Web 開発、サービス開発では必須になると思う。特にクラウドとの関係でも。この分野は「もうそうなっている」のかも。それ以外では、組込み系開発、および、基幹系開発にも適用が進むと期待される。
- 日本の競争力がある分野(鉄道、原子力、車など)でのソフトウェア開発では、イノベーションに近いものが求められると思います｡現状がどうかは把握できていませんが、そこでは、なんらかの非ウォーターフォール開発が必須となると感じます。
- 社会制度がソフトウェアに直接影響を与え、かつ、現状でウォーターフォール型開発を建前としている政府、行政サービス、官庁系システム群は、アジャイルプロセス型への抜本的な移行が期待されていくでしょう。これには、政府調達制度、予算制度そのものの仕組みから変更していく必要があります。
- 継続的に機能拡充をすすめてゆく必要性の高い分野。つまり一般的な業務システム、サービスシステムの分野。次つぎと継続的に新製品を提供していく必要のある組込みソフトウェアの分野。
- IPA として、ある程度の中長期的展望を示す必要があるとするならば、現時点で適用が出来ていないポーションが主な攻めるべき分野でしょう。
- 企業情報システムにおいては、少なくとも情報系（意志決定支援など、知的業務の生産性・質の向上を目的としたシステム領域。使っても使わなくても良いシステムの領域）については、反復・インクリメンタル、あるいはアジャイルなどが適している。
- 電気、水道、ガス、交通管制など、ライフライン系のシステム開発。
  システムのライフサイクルが長く一過性のものでないため。
- 現在、ウォーターフォール型の開発プロセスを採用しているすべての分野です。
  理由は､「ウォーターフォールモデルは間違いだ！」だからです。

2. 非ウォーターフォール型開発を適用するための前提条件は何か？

(3) 非ウォーターフォール型開発を成功させるために必要な前提条件にはどのようなものがあるでしょうか。そのようにお考えになる理由とあわせてご回答下さい。前提条件はないとお考えの場合も、その理由をご回答下さい。

回答

- 利用者、顧客、開発者の協力関係。それがないと、フィードバックがうまくかからない。(長時間待たされる、ドキュメントで会話する、などの逆方向に進みがち) 非ウォーターフォールでは広範囲すぎるので回答不可。Agile 開発だとすると、コーディング力があり自律した人材が集まっていることが望ましい。但し、「前提」というほどの強さかというと、そうでもないと思う。
- ユーザ／ベンダ、経営／現場、企画／監査といった組織や部門間をまたがる協調（協働）関係や信頼関係が前提条件です。
- 前例踏襲でなく、新しい手法を試みてゆく革新的な組織風土。新しいことを試みる際のリスクを吸収していけるマネジメント上の余裕。継続的な革新努力（プロセス改善など）をおこなう関係者一同の組織的なマインド。
- 非ウォーターフォールが未定義のままなので、前提条件と明確に定義できませんが、各私企業の一般的経営努力で済む範疇と、明らかに政府等の力が必要となる範疇とに別けられると思いますが、後者の政府関連の話は、まずは、法的な整備と思います。特に、ソフトが産業として成り立つための整備（および整理）が必要です。それぞれの立場におけるリスクの認識の一致・理解と、そのハンドリングの方法（契約方法含む）の整備。
- 関係者が近い距離（チーム制など）でコミュニケーションが取れること。
  WF 型の様な手続きよりも、コミュニケーションを重視するため。
- 関係者が非ウォーターフォール型の開発プロセスをきちんと理解していれば大丈夫だと思いますが。

3. 非ウォーターフォール型開発は信頼性に課題があるか？ (1) 非ウォーターフォール型開発は従来型開発に比べ、成果物に対する信頼性を確保することに課題が存在するとしたら、それはどのようなものでしょうか。そのようにお考えになる理由とあわせてご回答下さい。課題はないとお考えの場合も、その理由をご回答下さい。

回答

- ウォーターフォールでは信頼性に課題があるか？という質問と同じ答えになると思う。すなわち、テスト戦略に大きく依存する。一般的には、信頼性は向上する。動くテストが保守されながら多く作られるため。
- 直感的ですが、プロセスと信頼性はそう大きな関連はないと思います。あるとすると、新しいプロセスを導入すると、そのマネージメントノウハウが薄いので、信頼性に影響するということはあると思います。それは、「変化」に伴うリスクで、プロセスの特性ではないと思います。
  それとは別に、イノベーション的なものよりも信頼性をかなり重要視するプロジェクトをわざわざ Agile 開発を適用するか？　というと、それは、少なくとも日本では疑問だと思います。
- 信頼性に関する要求を明確にすることが課題となるでしょう。
  「アジャイルプロセスでは信頼性が確保できない」と言う都市伝説を排除することが課題かもしれません。

- 信頼性の問題は伝統的な開発でもかかえているので、非ＷＦだからといって特別に存在するとは考えられない。手法が新しいので、信頼性確保の手法が未成熟ということがあるかも知れないが、伝統的な方法でも必ずしも成熟しているとは言えないので、一般論は無理である。非ＷＦ型で信頼性が確保されていることを外に見せる努力は必要。
- ウォーターフォールが完全に品質が良くて、非ウォーターフォールが全くダメという議論にはならないのではないでしょうか？
  「品質」という概念を開発過程における品質の作りこみの担保のあり方と捉えることが重要という事だと思います。結果の良し悪しだけを判断基準にしない。また、失敗の予防学と、失敗回避学という軸が、開発手法という軸とは別に独立に注力されるといいなと思います。
- 開発手法自体に起因する課題は無いと思う。しかし、そもそもの導入の動機がコスト削減である場合は、テスト工程の作業が不必要に削られる可能性も高く、テスト工程において何らかのコスト削減手法も同時に導入しなければならないと考える。
- 非ウォーターフォール型開発の方が、従来の開発方法よりも信頼性が確保出来ると考えています。テストが頻繁に行われ、動くものを重視しているためです。
- 非ウォーターフォール型開発だから信頼性が低くなることはないと思います。それはどれだけきちんとテストしたか等に依存する問題だからです。

4. 非ウォーターフォール型開発は保守、引継が難しいか？

(1) 非ウォーターフォール型開発は従来型開発に比べ、保守や引継に関して困難な面が存在するとしたら、それはどのようなものでしょうか。そのようにお考えになる理由とあわせて記載して下さい。困難な面はないとお考えの場合も、その理由をご回答下さい。

回答

- ウォーターフォールでは保守・引継ぎに課題があるか？という質問と同じ答えになると思う。すなわち、保守戦略に大きく依存する。一般的には、ドキュメントで引き継ぐのであれば、非ウォーターフォールでもドキュメントを残すタスクを入れるべき。（これでウォーターフォールと同等の引継ぎ性能がでる。）さらに、人を徐々に入れ替えたりすることで、よりよい引継ぎができる。（ウォーターフォール以上の引継ぎ性能）
- どうでしょう？従来開発でも、保守や引継ぎは大抵困難で、プロセスの問題というよりは、人の能力の問題では？
- アジャイルプロセスは、初期構築と保守との区別をせず、ライフサイクルプロセスとして扱うのが本来の姿です。保守で他組織へ引継ぐということが必要であれば、ウォーターフォール型と同様に適切な文書化や伝達をしていくことは変わりありません。
- 信頼性の話と同様に、新しい手法なので、保守や引き継ぎの環境が未成熟と言うことがあるかも知れないが、伝統的な開発でも、必ずしも成熟しているとは言えないので、一般論は無理である。ここでも、新手法でも保守、引き継ぎに心配がないことを外部に見せる努力は必要と考えられる。
- ウォーターフォール型はドキュメントがしっかりしていて、非ウォーターフォール型はしっかりしていないという前提が置かれているような気がします。

その前提はさておき、保守を行うための資料と開発のために必要な資料とは必ずしも同じではないと思います。技術開発の延長としての必要悪として保守を捉える考え方とは違って、保守が独立した産業になる時代ですので、きちんとした保守のための資料は、どちらとも必要であり、これもまた、新しい議論すべき事業（技術）分野ではないでしょうか。

- アジャイル型の開発では、保守・引継に必要なドキュメントが十分用意されていないこと､あるいはそれらを作成するコストの必要性が欠落していることが多い｡ただし、開発手法自体に起因する問題ではなく、アジャイル型開発が採用されているコンテキストとの相関の問題なので、開発手法の向き不向きの議論ではなく、保守・引継が必要な場合に何が必要となるかを議論すべきである。また、利用者間、利用者側と運用者側との間などに相反する利益が有る場合は、保守時にも継続的に仕様が揺れる可能性があるのではないかと思う。
- 従来のものと変わらないと思われます。
- ウォーターフォール型の開発プロセスと非ウォーターフォール型の開発プロセスで違いが生じるとは思えません。この質問が出てきた背景を知りたいです。

5. 非ウォーターフォール型開発における契約はどのようなものが良いか？ (1) 非ウォーターフォール型開発を適用して開発を行うのに、従来型の契約方法では問題となるのはどのような点でしょうか。

回答

- 以下の 2 点
  - 仕様が揺れる。
  - 中間の納品を検収する必要性。
- Fixed Price では、成果があらかじめ客観的に決まらないと契約にならないのが問題。よって、Time&Material 系の契約にすればよい。
- 仕様をあらかじめ固定化すること。
  （アジャイルプロセス以外の）プロセスや中間成果物を指定されること。
  発注側が要求や検収に迅速に対応することが明記されていないこと。
  発注側が開発者側の管理権限を持つこと。
- ユーザ主導、内製中心の場合、専門技術を派遣の手法で確保することが考えられる。この場合、専門家としての高級派遣のようなメカニズムが必要だが、現在完備しているとは言い難い。派遣を禁止したり、「派遣は低価格」と固定されていると、実質的に実施出来ない。
- 開発ケーパビリティそのものを売る場合と､開発結果としてのサービスを売る場合(含内製）とを分けて議論したほうがいいと思います。
  ソフト産業の定義にもよりますが、工事進行基準、をまず、クリアする必要があります。
- 作業量が決まっていないこと、また期待された成果がでるかどうかわからないことによるリスクが適切に処理できない。
- 参加型の開発スタイルになると思われるので、派遣契約が中心になると考えます。
- 開発プロセスの問題ではなく、要求仕様が変化することを前提にした場合、定額請負

契約だと（途中で変更契約をしないと）作業の完了を第三者が判断できなくなり、トラブルの原因になります。

(2) 非ウォーターフォール型開発による開発に適した契約とはどのようなものとお考えになりますか。

回答

- 以下の 3 点
  - 1. 短期での納品と検収を、顧客が受け入れる義務があるもの。
  - 2. 完成物の仕様をあらかじめ固定しないこと。
  - 3. 顧客がわの参画責任（1.に関わる）が明らかなもの。
- Time&Material 系の契約。
  ただ、日本のソフトウェア産業は、Time&Material 系の契約である「派遣業」から、責任をもって完成させて効率向上分は利益を高くする「請負業」へ転換してきた、という歴史的経緯があって、Time&Material 系の契約へ戻るのは経営的に困難（一次取り纏めベンダ）。
  ユーザ企業の情シ部門も、請負で発注するというのに最適化されていて、これも戻るのは結構困難なのではないか？
- 以下の 4 点
  - （1）請負契約では、体制、金額、期間等は記載するが、詳細のプロセスや成果物の縛りがないようにする。つまり、アジャイルプロセス以外の工程や中間成果物の約束をしない。
    →工程（成果物の提出順序）制約のない契約
    →発注側に管理権限のない契約
  - （2）詳細機能実現の約束をしない。むしろ、重要な機能（要求）が実現できない場合にのみ契約不履行となる「赤点契約」とする。
    →制約を約束する赤点契約
  - （3）発注側が要求や検収に迅速に実施できる協働関係が築ける契約。
    →短期に確実に検収が実施される契約
    →発注側も関与し協働関係が築ける契約
  - （4）保守契約に類似するライフサイクルにまたがる契約
    →訂正（訂正保守、予防保守）、改良（適応保守、完全化保守）契約
- ユーザ主導、内製中心の場合、高級派遣体系がほしい。法律事務所から法律専門家が派遣されるようなモデル、高級戦略コンサルティングファームの様なモデル。
- 現時点で既に、法的にも手法的も問題なく実施されている分野は、問題ない（対象外）とすれば、現時点で問題なのは、いわゆる「ウォーターフォール型」で工事進行基準の厳格な適用が求められている分野が課題を抱えている（考慮に値する）分野です。
- 利用者側は仕様を早期に決める/絞り込むと支払いが小さくなり、開発者側は仕様を早期に/正しく/多く実現すると受け取りが大きくなる契約。保守・引き継ぎの必要性やシステムのライフタイム、達成すべき非機能要件などによって変わるコストが金額に反映される契約。
- 前問と同様、派遣契約になると考えます。

・ 非ウォーターフォール型の開発プロセスの種類が多いので、ケース・バイ・ケースだと思います。たとえば、インハウスで Agile 開発の場合には派遣契約になるでしょうし、ウォーターフォール型の開発プロセスを繰り返すスパイラル型なら定額請負でも大丈夫だと思います。

6. 日本のソフトウェア産業構造のあるべき姿はどのようなものか？
(1) 非ウォーターフォール型開発が日本のソフトウェア産業の構造や競争力に与える影響にはどのようなことがあるとお考えになりますか。

回答

・ 契約問題の解決によっては、大きな影響がある。現在のままでは、早晩、ユーザ企業側がエンジニアを直接雇用して、開発を行なうスタイルになる。契約問題が解決すれば、競争力のある SI やソフトハウスが活気を持ち直し、開発現場が明るくなる。
・ 影響はありません。
これは、逆で、ソフトウェア産業の構造が変わらないと Agile 開発がメインストリームになることは難しいと思います。内部の開発方法が、ビジネス構造へ影響するというような逆はありえません。
・ ビジネスプロセスとエンジニアリングプロセスとが連動していくことになるので、エンジニア領域のビジネスマインドが増長されてくることによって、結果として、競争力が上がってくるかもしれません。
基本的には、ソフトウェアがもたらす（非金銭的なものを含む）富というものを考えるようになり、パラダイムシフトが起きてくるでしょう。例えば、ソフトウェアを作ると使うが同じように考えられる（地産地消）ような仕組み、ユーザと開発者との区別が無くなる世界になっていくでしょう。
・ 日本のソフトウェア開発のスピード感を増し、コストと品質を上げ、良質の職場を実現し、産業全体の国際競争力向上に貢献する。ソフトウェア産業の活性化、そのソフトウェアを用いたサービスの活発化、そのソフトウェアを組み込んだ製品の競争力の向上により、新たな良質の雇用の場を生む。
・ この問題設定そのものが抱えている問題点がある。日本のソフト産業に元気がない理由がプロセスのあり方や、開発手法に由来するのではなく、それらとは直交する軸であるはずの、産業としての技術領域や、対象領域の変化が原因であるのではないのでしょうか。
例えて言えば、開発手法というメガネを通して見た（分析した）時に風景が昔とどう変わっているのかが重要であって、メガネ自体の変化の議論はその後でいいはずと考えています。
直感的に言えば、開発期間の短期間化、ソフト寿命の短命化、が発生しているのです。その現象に対処すればそれでいいのでしょうか、現時点では、それだけでは不足で、新しい技術領域への投資が国として必要となっていると思います
・ 開発単位をうまく制御する（開発に適したサイズに分割する）能力が重要になる。一方で、契約単位は、単体のプロジェクトよりも大きな単位での契約が主流になるのではないか。また、オフショアの使い方に工夫が必要になると思う。一方で、オフショア側からは日本語の点を除けば参入障壁が低くなるのではないか。

- もともと日本のソフトウェア産業は、品質への想いが強いので、そこにスピード、柔軟性という要素が加わることで、競争力を増すと考えます。
- 質問の趣旨がよくわかりませんが、ソフトウェア産業の競争の源泉は人材です。ソフトウェア技術者の処遇が改善され、明るく働きがいのある職場になれば、優秀な人材が集まってきて、ソフトウェア産業の競争力は向上すると考えられます。

(2) 非ウォーターフォール型開発がより多くの場面で適用されるようになることを前提とした場合、日本のソフトウェア産業構造としてあるべき姿はどのようなものとお考えになりますか。

回答

- 未来の極端なシナリオは、以下の２つ。
  - 契約問題がうまく解決しない場合、インハウスで非ウォーターフォールが活発化する。ユーザ企業が開発パワーを持つような形になり、受託開発、というものや、SIer という業態がなくなる。
  - 契約問題が解決する場合、ユーザ企業が非ウォーターフォール型での開発を開発会社に発注する形態がとれる。そして、スキルの高いエンジニア、チームのモチベーションとコミュニケーションをうまく作れるマネージャを持つ開発会社が、より高い競争力を持つようになる。

  上記２つの形は、ともに、現在よりもよい方向と思える。ともに、実力のあるエンジニアと管理者に競争力が出て、ソフトウェアの開発現場も明るくなる。
- 「ソフトウェア産業構造としてあるべき姿」というのは、開発プロセスをもとに規定するようなものではないと思います。
- 問題領域（実世界）とコンピュータ領域との間に設定されている既存のユーザ／ベンダの境界はなくなっていくでしょう。日本のおもてなしの精神、ものづくりなどの強みがソフトウェアやプロセスに埋込まれていくことでしょう。
- 平準化した常時開発の比重の増加。内製開発比重の増加。専門職制度の拡充と普及。システムインテグレータ産業の派遣型高級コンサルティングファームへの転換（新ＳＩ産業)。業務システム分野でのサービス型情報システム（SaaS 型、クラウド型）形態の比重の拡大。これらを通した、多重下請け構造の解消。上記、新ＳＩ産業の海外展開。
- ソフト産業の再定義が必要になると思いますが、変化する（もしくは変化させるべき）軸という観点で考えますと、一昔前でのソフトのライフサイクルを構成する各フェーズが、現在では、専門分野化が激しくなっている感じがします。それは、ソフトを必要とする分野が非常に広がり扱うべき技術領域が多岐にわたるようになったために、急激にすすんでいます。企画フーズしかり、コーディングフェーズしかり、等々。。
  ここで重要なのは、これらに共通的な（より高い抽象レベルでの）ソフト技術の再定義が必要となっています。
  たとえば、単純な例では、ソフト開発現場でのプロジェクト管理や、モジュール分割技法等々は、ソフト開発以外でも、十分通用します。逆に、ソフト的な考え方を適用すると上手くいく場合があるのでは？
  産業構造を変える原動力としてのソフト産業（がもつ潜在力）は、より詳細な製造工

程ではなく、上流工程への展開と思います。
- 雇用リスク（雇用側、被雇用側）や仕事量の変動リスクを構造としてどこでヘッジするかにもよるが、基本的には 3 層構造（利用者-インテグレータ-開発者、利用者-情報システム部門-開発者）くらいが妥当ではないかと思う。
- 参加型開発になると考えます。マネジメント（偏重）指向は淘汰されてしまうと考えます。
- 本当の意味でプロフェッショナルの集団になることだと思っています。

7. 以下にあげる論点は委員の認識が概ね共通化したと考えておりますが、異なるご意見がある場合にはその内容をご回答下さい。

■ 論点

A) 非ウォーターフォール型開発の定義
- ビジネス連動、マーケットインで、いつでも出荷可能にしておく環境を構築して継続的に開発を実行するスタイルである
- 明確になった仕様を明確になった時点で迅速に実現するものである
- 繰り返し開発を行うものである

B) 非ウォーターフォール型開発を利用するユーザの価値の一つは、仕様決定の意思決定を遅らせることができる（仕様変更や状況変更に迅速に対応可能である）ことである

C) 実際には厳密にウォーターフォール型開発によって行われた事例は少なく、非ウォーターフォールの要素が含まれていることが多い

D) 現状では非ウォーターフォール型開発に適した契約方式が存在しない

E) 業務システムを連続的に更新していくことができるような仕組みを考案し、開発を平準化することが必要であり、これに応じて現状のソフトウェア産業界は形を変えなければならない

回答

- 上記非ウォーターフォール型開発の定義が、アジャイルの定義に近い。パッケージ利用、なども非ウォーターフォールとよぶ、というのが当初の仕様だったような。。。。
- 以下の 3 点
  - （A について）どういう文脈での「定義」でしょうか？　本当に「定義」するなら、アジャイル宣言をもとにしたほうが、私はしっくりきます。「定義」ではなくて、だいたいこういうことをみんなが想像するよね、ということであれば、反論はありませんが。
  - （C について）これはこれで正しいと思いますが、これもどういう文脈でしょうか「ウォーターフォールが理想だけど、プロジェクトの特性からやむなくいろいろ工夫している」というのと「ウォーターフォールは理想ではなく、ｘｘｘｘだ」としているプロジェクトでは、全然方向が異なってくると思いますけど。
  - （D について）レベルによりますが、Time&Material 系の契約だと阻害要因はないと思います。推奨する、というレベルではないですが。モデル契約みたいなものが世の中にあるほうがいいに決まっていますが、発注側、受注側、両者が Agile

でやろうと決めたら、契約はいかようにもできると思いますが。

- 最後の項目の「平準化」については、よく判りません。人の流動性が組織内／間で確保されていれば、制約にならないでしょう。
- いずれも異議あり。議論と定義が足りない。もっと時間をかけ、エビデンスを集め、議論すべき。上記のような結論に至る状況にない。
  5番目の項目は、「・・・すべき」という議論ではない。「・・・となる可能性がある」、といった程度である。
- 以下の４点
  - 品質を担保する行為と、ビジネスとしての要件への対応をクリアに分けて議論したほうが、実りがあると思います。
  - 外的な要件で分類（定義）したら、当然ながら、開発手法はそれに適合するように変化します。前述の定義は、このような外的要件の分類のように見えます。
  - 開発手法への要求（要件）が、どのように変化しているかの分析が必要かと思います。
    感覚的ですが、外見は、従来の情報処理のように見えて、実は、別物（言いすぎですか？）
  - 究極の選択にならないようにお願いします。どちらを選んでも不幸とならないためには、ユーザ視点での調査を深堀する必要があります。
- 企業システムにおいては、ソフトウェア産業だけでなく、利用企業の情報システム部門を含めて形を変えねばならないと考える。
- 仕様決定を遅らせることが出来る訳ではなく、括弧書きされている内容で良いと思います。
  また、適した契約は、派遣契約があるので、存在しない訳ではないです。
- 非ウォーターフォール型の開発プロセスは、いろいろな種類があり、これらの認識が共通化したとは思っていません。そもそも、委員会は、委員の発表とその質疑応答で、これらについて議論していないのではないでしょうか？

8. 非ウォーターフォール型開発に関連して人材像もしくは人材育成について検討すべきとお考えのことがある場合には、その内容をご回答下さい。

回答

- ヒューマン系スキルの重要性が上がる。
  チームビルディング、コーチング、ファシリテーション、参加型意思決定、アプリシアティブインクワイアリ、ソリューションフォーカス。
- 日本の SIer では、機能設計/アーキテクチャ設計をする人はコードを書かないのが普通だと思います。機能設計～コード作成までを最新の技術で「できる」人を多く育てていくことが必要と思います。また、４０代、５０代でも、長年の経験と最新技術を身につけて、機能設計～コード作成までできるようにしていくことが重要です。
  でも、それを日本 SIer に求めても無理があるので、別の競争力のある分野（鉄道、原子力、車など）でのソフトウェア開発でそうなるようにしていくことが必要と思います。

- アジャイル・ソフトウェアセル生産で言う、スーパーシェフ／スーパーコックの育成が必要でしょう。また、問題領域（要求定義よりもっと上流？）の分析、実世界と開発世界との翻訳ができる人がもっとたくさん必要になるでしょう。問題を解く人より、発見できる人が必要になります。
  コミュニケーション能力や調整能力よりも、基礎力や洞察力を持った人材の方が必要になる世界になると考えています。
  均一な工業的世界観での同質な教育や資格制度は無用の長物となります。
- 歴史ある大手企業も、新しい企業も同様に人材の不足を指摘している。新開発方式を念頭においた人材育成が必要。現在進められつつあるＦＤ（ファカルティ開発）の運動のなかに取り込んでゆく必要がある。求める人材像を先端企業で描き、教育環境に伝えてゆく必要がある。
- ウォーターフォール、非ウォーターフォールに関わらず、日本がおかれている状況を考えますと、従来に比べて、近隣諸国の追い上げもあり、より上流フェーズ、たとえば、コンセプトの創造、ビジネスモデルの創出、新価値の創造（でっちあげ？）に強い人材が必要となると思います。
  アンケートの中でも答えましたが、「ソフト開発」のノウハウを抽象化・共通化することが軸となると思います。
- 出世するにはマネージャになる、という以外の道が、より明確に打ち出され、世の中に認知される様になっていくと思います。また、そうしないといけないと思います。
- これから考えていきたいと思います。