
NPV=PV— Initial
alue,r= interest rate,n= Years
Dynamic Systems Development
図解でわかる
アジャイル・
プロジェクト
マネジメント
Requested Procedure Stu
EMV ; Expec
PMアソシエイツ株式会社
代表取締役
鈴木 安而 著
NPV
turn On Investment
Internal R
tal Development
SCC
Return On Inve

NPV=PV— Initial

alue,r= interest rate,n= Years

Dynamic Systems Development

図解でわかる

アジャイル・プロジェクトマネジメント

Requested Procedure Stu

EMV；Expec

PMアソシエイツ株式会社
代表取締役
鈴木 安而 著

NPV

turn On Investment

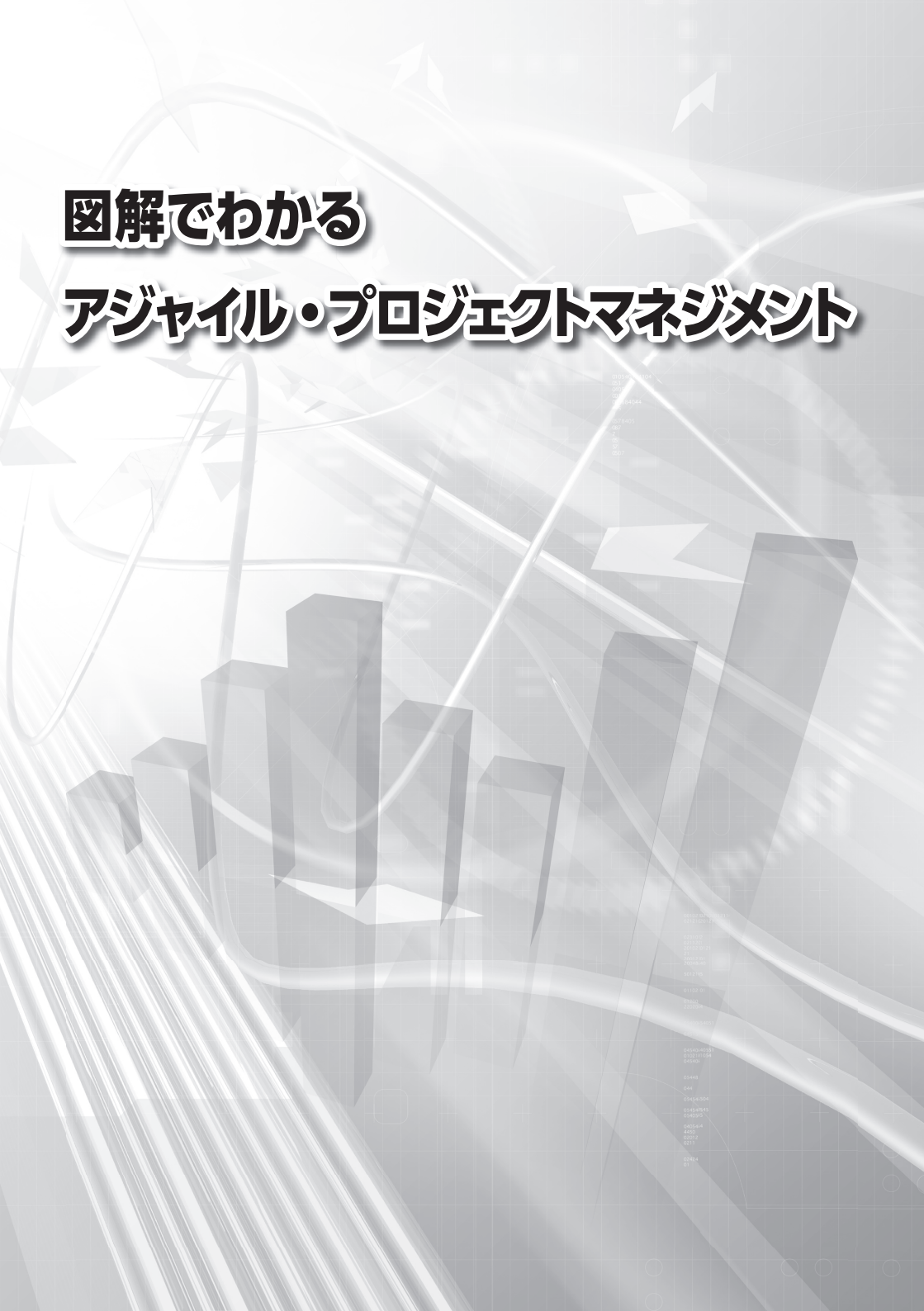
Internal R

tal Development

Return On Inve

SCC

図解でわかる
アジャイル・プロジェクトマネジメント



・本書に記載されたＵＲＬ等は執筆時点でのものであり、予告なく変更される場合があります。
・本書の使用により、万一、直接的・間接的に損害が発生しても、出版社および著者（著作権者）は一切の責任を負いかねますので、あらかじめご了承下さい。

はじめに

「アジャイル」という言葉がプロジェクトに適用されてから、早くも四半世紀が過ぎました。欧米ではソフトウェア開発プロジェクトには当たり前となった「アジャイル」手法ですが、日本では名前だけが先行して事例は少ないのが実情です。「やりたい」と興味をもっても「どうやったらいいのかわからない」し、その前に「アジャイルってなんだ？」という疑問に答える人が周りにいません。日本の文化的特徴として「マネジメント」の部分については興味がなく「ものつくり」の部分に興味のほとんどを注力するので「アジャイル」を解説しても「どうやれば早く作れるのか」という質問が多いことがあります。「アジャイル」は、決して開発行為そのものが早くなるわけではありません。プロジェクトのライフサイクルを通して手戻りやエラーが少なくなるので、結果的に遅れというリスクが少なくなる、ということが本当のところでしょう。だからなんでもアジャイルで早くなるのではなく、従来手法でしっかり目標を達成できているのならば「アジャイル」は必要ないといえるのです。

最近、ある研修の場でアジャイルについて話したところ、「先生、どうしてアジャイルアジャイルと勧めるのですか、私はアジャイルなんか不要だと思っているのですが」と、ひとりの受講生からコメントされました。その研修では、複数のプロジェクトを有機的にまとめて成果を創出するための「プログラムマネジメント」について教えていたので、受講生には結構ベテランのプロジェクト・マネジャーが多かったのです。彼は新しい手法へのアレルギーを示したわけではなく、正しく理解していなかったためにそのコメントになったのだと考えました。そこで私はすかさずステーシー・マトリックス

（本文中で説明）を示して、プロジェクトの環境に適した手法を選ぶべきである、ということを説明して理解を得たという経験があります。

あるいは「最近アジャイルという手法が若い人たちの間に流行っているが、何がアジャイルなのかよくわからない」とか「アジャイルに関与している人たちは、今までの標準を無視したような用語や考え方を持っているので付き合いにくい」などというベテランの声も聞かれました。確かにアジャイル関係の文献を英語で読んでみると用語や表現に独特なものがあり、簡単には翻訳し難いと感じたものです。アジャイル関係の用語にはカタカナ表現が多いこともそれを物語っています。私が長年翻訳に携わってきたPMBOKガイドは標準書として厳しいレビューを経て出版されているので、用語や表現が洗練されています。その考え方はISO21500へ引き継がれているので理解するにも楽で表現にも一貫性があり、翻訳には意訳が許されないというような制約条件があるものの、さすがにANSI承認の標準としての品質が保たれています。私はPMBOKガイドの邦訳ではなるべく日本語化するように努力したつもりですが、それでも「マネジメント」や「エンゲージメント」など日本語にしてしまうと真の意味が伝えにくい用語についてはカタカナにしてあります。ところがアジャイル関係では、さまざまな手法の提唱者の強い思いからさまざまな新しい用語や考え方が生み出されているので、アジャイル全体を表現できるように体系化された構造にはなっていないのが現状です。

そのような状況から、2013年に米国プロジェクトマネジメント協会（PMI）はアジャイルを含めた書籍としてIEEEとの協業で『PMBOKガイド第5版ソフトウェア拡張版』を出版しました。それは、いわゆる「ウォーターフォー

ル型」から「アジャイル型」までのプロジェクト手法全体を PMBOK ガイドと同様に「プロジェクト・ライフサイクル」と「知識エリア」という観点から表現しているので、従来の標準書の拡張版として理解しやすくなっています。この図書は 2015 年 12 月に PMI 日本支部から日本語版が出版されていますので簡単に入手可能です。しかしながら、それはあくまで PMBOK ガイドの内容にアジャイル関係の考え方や用語を追加してソフトウェア開発プロジェクトについて包括的に記述しているので、個別のアジャイル手法については深く記述しておりません。例えば実際にアジャイル手法を採用する場合にどのように進めるべきなのか、あるいはどのような問題が想定されるのか、についての具体的な記述はありません。

そこで筆者は 2 年かけてアジャイルに関する欧米の事例や日本の事例を研究しました。そこからアジャイルについての具体的な導入方法や日本における問題点を明確にし、プロジェクトを成功させるためのマネジメント手法として体系化されたものを紹介しようと考えたのです。現在のところアジャイル手法のうち最も新しい「スクラム」が、筆者の知る限り最も優れていると考えています。ただし拙速にそこから入るのではなく、なぜアジャイルという手法が考え出されたのかについてプロジェクトの歴史を理解することも大切だと考え、従来の「ウォーターフォール」からのプロジェクトの流れの解説も試みました。そして具体的な導入手順や失敗例からの教訓を紹介し、最後に「スクラム」の体系をわかりやすく解説いたしました。用語については PMBOK ガイド翻訳の経験から筆者なりの邦訳を試みましたので一般のアジャイル用語のカタカナ表現とは異なるかもしれませんが、解説も加えてありますので意味は十分に理解できると思います。

紙面の都合ですべてのアジャイル手法を紹介できませんが重要な部分はしっかりと記述したつもりです。さらにアジャイル導入の具体的ケースとして、ある女性プロマネが上司を説得し仲間を募ってアジャイルを進める事例を物語風にわかりやすく紹介しております。これからアジャイル・プロジェクトに挑戦なされる方には十分にお役に立つものと確信しております。ぜひ最後までお目通しいただき、実務にお使いいただければ幸甚です。

2016 年 5 月

筆者

目　次

【閑話休題】

序論

プロジェクト手法の歴史

近代的プロジェクトの始まり

ウォーターフォール型

変更要求

スパイラル型

アジャイルの誕生

「アジャイル（Agile）」という言葉は、プロジェクトマネジメントの世界では未だ確立した意味をもっていませんが「すばやい」とか「敏捷」という意味で使われることが多いようです。プロジェクトを素早く進めることができるなんて、なんと素晴らしいことでしょうか。実際にそんなことが可能なのか、アジャイルの本当の意味を理解するために、まずプロジェクトマネジメントの歴史を遡ってみましょう。

近代的プロジェクトの始まり

近代的なプロジェクトの始まりとして知られた「マンハッタン計画」（英語では Manhattan Project）は 1942 年に開始され、プロジェクトの成功事例と言われています。この時代は、いわゆる管理工学をもとにして発展し、建築、製造、などの分野で成果を上げてきました。その後コンピューターの時代になりましたが、天才的なプログラマーによるソフトウェア開発作業が主流であり、作業効率や品質管理に幾分の問題がありました。それを解決するために、1968 年にソフトウェア開発のためのプロジェクト・ライフサイクルの体系化が提唱され、その後 1970 年に W. W. ロイスによって発表された論文**「Managing the Development of Large Software Systems」が「ウォーターフォール型」の原型**となりました。作業効率や品質改善のために**「構造化プログラミング」**が提唱されたのもこの頃です。

ウォーターフォール型

「ウォーターフォール型」プロジェクトでは、**作業工程を「要求定義」「外部設計」「内部設計」「実装（プログラミング）」「試験」「運用」などのフェーズとして時系列に分割**します。それをスケジュール表（ガントチャート）に表して、ひとつの流れとして完了するように計画を策定し、進捗管理をしていきます。ひとつの流れということは、**原則として前工程が完了しないと次工程に進まないように設定**します。そうすることによって前工程の成果物の品質を確保し、前工程への手戻りを最小限にできるのです。この流れは進捗管理しやすいのが特徴ですが、前工程で明確な結果が出ていることが前提となります。要するに、最上流工程であ

る「要求定義」が明確でないと次工程以下の成果が出せません。そこが不明確のまま開発を進めるとどんなことになるのか、ソフトウェア開発の現場は悪夢の「デスマーチ（死の行進）」と言われる所以です。なぜ「要求定義」を事前に明確にすることはできないのでしょうか。

「要求定義」を最初に明確にすることは困難である、という認識は、古今東西ソフトウェア開発では常識となっています。そのことは次のように言い表せます。システムを発注する側はシステム自体を欲しているわけではなく、ビジネスを遂行する上でのツールを欲しがっているのです。そのツールへの要求事項は「いいものを早く安く提供してくれ」の一言に尽きます。極論すれば、細かい仕様なんかどうでもいいのです。というよりも細かいシステムの仕様を聞かれても、あるいは提示されてもわかりません。せいぜいどんな機能があるのか、について興味を示す程度でしょう。ですから最初の要求といっても大雑把な表現しかできないものです。たとえ制作側から「こんなものでどうでしょう」と聞かれても「見てみなくちゃわからない」とか「専門家が言うのだからそうするか」というような返事が精一杯です。

変更要求

その代りに作業の進捗が進むにつれ成果物の形が見えてくると、「実はこう思う」とか「そこはこうでなくては困る」とか、具体的な要求や指摘が出てくるものです。人間は、具体的なモノを目にすることによって理解が深まり、自分自身の要求も明確になってくるものです。それがプロジェクトへの変更要求となって製作者を悩ませます。「せっかく作ったのに・・」「今までの努力をどうしてくれるんだ・・」など、口には出さなくても不満タラタラとなります。作業は手直しを迫られ、余分なコストはかかるし、ヤル気も失せます。「でもお客様の言うことだから・・」とチームを奮い立たせて頑張るのですが、進捗の遅れはなかなか取り戻せません。上の人は、遅れを取り戻すために「人を入れろ！」といいます。ソフトウェア開発では、ブルックスの法則で有名ですが、人を入れると短期的には遅れが増すのが常識です。そもそも変更要求がプロジェクトを遅れさせる原因ですが、ソフトウェアは目に見えない、という特性からくる問題です。進捗状況が目に見えれば、そんなに変更要求は出ないものです。読者がマイホームを建てたときのことを思い出してください。建築現場に行って、大工さんに「その柱は

邪魔だから少し動かしてください」というような要求は出さないものですね。ところが一般に、ソフトウェアはパソコンのキーボードを叩けば簡単に出来上がるもの、と思っていますから「ちょっとの作業で変更できる」ものと勘違いしている方が多いのです。では変更要求を禁止しましょうか。それはビジネスとして良い方法とはいえません。お客様の真の要求にこたえるのがプロジェクトに課されたテーマです。その要求もプロジェクトの初期に出していただければいいのですが、後半になればなるほど変更がプロジェクトへ及ぼす時間やコストの影響が大きくなります。

スパイラル型

そこで考え出されたのが反復型という開発の考え方で、1988 年にはベリー・バームによって**「A Spiral Model of Software Development and Enhancement（ソフトウェアの開発と拡張のスパイラル・モデル）」**が提唱されました。このモデルでは開発期間を数回のループに分割して、ループ終了毎に顧客の評価を得られるようにしました。しかし一回のループが 6 か月から 2 年というように長期にわたることが特徴なので、大規模プロジェクトが対象とされていました。

アジャイルの誕生

ソフトウェア開発プロジェクトといっても大規模のプロジェクトばかりではありません。日本情報システムユーザー協会（JUAS）が調査した資料が手元にありますが、開発規模を 1 億円で分けてみると、1 億円以下の小規模プロジェクトは全体の 46% です。小規模プロジェクトの工数を 100 人月以内とすると、その開発期間はせいぜい 12ヶ月程度までですから、スパイラル開発を適用するには小さ過ぎます。もっと簡単な手法はないものか、という思いが当然出てきます。同時に、さらに効率的なソフトウェア開発の手法への期待も膨らんできました。そこでケント・ベックらは、2000 年初版の著書『eXtreme Programming Explained - Embrace Change（エクストリーム・プログラミング）－変化を抱擁せよ』において、**変更への対処法や効率的なソフトウェア開発手法を提唱**しました。これを一般に **XP** と呼んでいます。XP では「変更」は当然あるべきものとして積極的

に対応することを目指し開発の繰り返しのサイクルを数週間に短縮しました。その頃のアメリカではインターネットの発達に伴い開発スピードが最優先であるようなニーズが生まれていました。その代表にはビジネス条件が日々刻々と変化するウェブ・サイト開発があります。そうでなくてもダウンサイジング・オープン化の潮流の元で、ソフトウェア開発全般に、難易度の高い短期開発が必要となっていました。その状況下で XP はブームとなっていったのです。

閑話休題

筆者の経験談

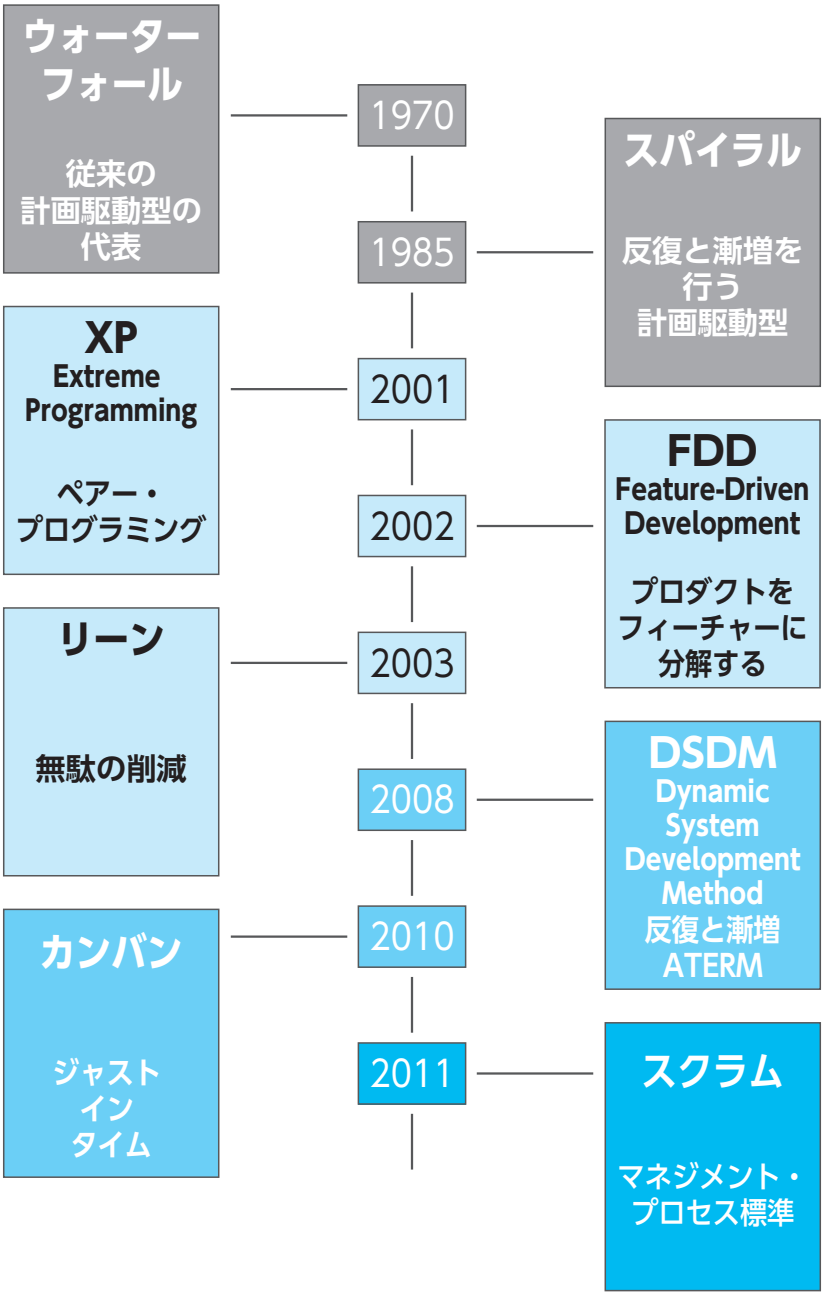
私の米国での初仕事は、あるオンライン制御プログラム開発におけるデバッガーでした。1975 年の頃の話です。構造化プログラミングが始まったばかりでまだ一般への普及というまでには至っていなかった時代です。言語は当然アセンブラーでした。

天才的なプログラマーが書いたコードを一生懸命になって読み解くのですが、コードにコメントがないものが多く、理解に苦しむような構造になっていました。

慣れてくるとプログラマーの顔が浮かんできて、「あいつならこんな書き方をするだろうな」と勝手に想像しながら進めると、コーディングの間違いが見つかるのです。本人にそのことを伝えると、「お前はなんでそんなことがわかるんだ」といぶかしがられたものです。もっとも、コーディングした本人には思い込みがあって、ミスは本人にはわからないものです。

その後は構造化プログラミング手法が普及されるに連れてミスが減ってきました。それでもプログラマーの性格によって冗長性などの特徴が顕著になります。

現在はリファクタリングという手法によってコードの解析は楽になっており、保守には欠かせない手法です。


ウォーターフォール
従来の計画駆動型の代表
1970
スパイラル
反復と漸増を行う計画駆動型
1985
XP
Extreme Programming
ペアー・プログラミング
2001
FDD
Feature-Driven Development
プロダクトをフィーチャーに分解する
2002
リーン
無駄の削減
2003
DSDM
Dynamic System Development Method
反復と漸増
ATERM
2008
カンバン
ジャストインタイム
2010
スクラム
マネジメント・プロセス標準
2011

序.1 アジャイルの歴史

第1章

アジャイル導入で変わるプロジェクトマネジメント

1.1 アジャイルの特徴とアジャイル・マニフェスト

ケント・ベックなど 13 人のアジャイル先導者たちは、**2001 年にアジャイルを世に広めるためのマニフェストを発表**し、アジャイル実務者はそれを順守するよう求めています。その考え方とは、形ばかりの運営よりも実を求めるものとして定義されていて、次のように表現されています。

> 『私たちは、ソフトウェア開発の実践あるいは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出そうとしている。この活動を通して、私たちは以下の価値に至った。
>
> 1. プロセスやツールよりも個人と対話を、
> 2. 包括的な文書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、価値とする。
>
> すなわち、左記のことがらに価値があることを認めながらも、私たちは右記のことがらにより価値をおく。』

アジャイル・マニフェスト

アジャイル・マニフェストは、日本語や英語など各国語で次の Web サイトに掲載されています。

http://agilemanifesto.org/

このマニフェストに述べられている原理原則は、**「価値のあるソフトウェアを早期に継続的に提供することによってお客様の満足を得ることを最優先とする」**にあります。このことを達成するために、次のことを提唱しています。

- たとえ開発の後半になっても変更要求を受け入れる。顧客の競争力を高めるために変化を利用する
- より短期間が望まれていることから、動くソフトウェアを数週間から数か月の頻度で提供する
- ビジネス側と開発側は、プロジェクトを通して協働する

・開発チームの中で情報を伝達する最も効果的で効率的な方法は、対面の会話である
・プロジェクトは自主性をもつように動機づけされた複数の個人によって構成され、彼らが必要とする環境と支援を提供し彼らが仕事をやり遂げることを信頼する
・動くソフトウェアを進捗管理における主な測定の対象とする
・アジャイルのプロセスは持続可能な開発を促進する。スポンサーや開発者および顧客は、最後まで一定のペースを守れるようにすべきである
・迅速さを高めるための優れた技術や設計に継続的な注意を払う
・単純化して、余計な仕事をしないようにすることが大切である
・自己組織化したチームからは、最良のアーキテクチャーや要求事項およびデザインが出現する
・チームは一定の間隔で、チームが効果的に活動できる方法を考え、それに応じて自分たちの振る舞いを調整する

ここで誤解がないように付け加えますと、**マニフェストの４つの項目の左側に述べられた内容を禁止しているわけでも認めていないわけでもない**、ということです。

例えば、マニフェストを発表してしばらくの間は「プロジェクト文書は不要なんだ！」という人を見かけました。当然、無駄なことを行う必要はありませんが、法律や企業で定められた規則への順守は、「コンプライアンス」として重要なことは言うまでもありません。例えば、プロジェクトの活動が価値の生産であるという側面からは、企業の財務決算においてプロジェクトの資産を計上する必要があります。単に開発コストを計上するだけではなく、顧客に提示して受け入れられた成果物は当然として、企業の会計基準に従い開発中のソフトウェアの価値も「仕掛品」として計上しなければなりません。そのためには会計上必要な証拠が求められるので、プロジェクト進捗状況の文書化が必要になるのです。その場合、具体的には企業における会計や財務の規則に従うことになります。

ただし、行き過ぎた文書化などのプロセスや実務は現場の生産性を低下させる要因として私たちの脳裏に刻み込まれています。例えば、上位のマネジメントが単に進捗状況を知りたいというだけで大量のレポートを求めることがあります。レポートを作成するために多くの時間が浪費され、その分プロジェクト作業の進捗に大きな影響を及ぼすことになります。したがって、このような状況を避ける

ための方策を提案したり、文書化をコンプライアンス上必要最小限に留めるように働きかけたり、現場のエンジニアに負担をかけないようなプロセスとするように働きかけたりすることも重要な活動です。これらの活動は「無駄の削減」を目標とするプロジェクト活動のひとつにもなります。

また、作業の効率化のために「標準化」を掲げてプロジェクトで使用するプロセスやツールを一律に決めてしまう組織が見られます。これはかえって個人やチームの作業効率を低下させる原因にもなっています。**本来、「標準」は「ガイド」であって規則ではありません。**一定の基準を示すことによって選択の容易さを示すものです。標準化するにしても、ある程度の許容範囲や閾値を設けてその範囲で管理すべきでしょう。エンジニアにとっては自分が最も得意とするプロセスやツールがあるものです。結果を求めることの大切さから見れば、プロセスやツールの選択は現場に任せる方が作業品質も向上するものです。多くの企業で「標準化」が「規則化」している例を見てきました。「規則化」すると、現場では自分で考えなくなってしまいがちで、新しい知見や知恵が出なくなってしまいます。これではいわゆる「指示待ち型」となってしまい、現場の思考能力を削いでしまいます。ですから、自分の組織では上司がレビューしやすくするための「標準化」になっていないかどうか、しっかり確認することが大切です。自主性を育むために自ら考える環境を構築することは、組織にとって最重要課題であるはずです。それを具現化するための手法として**「サーバント・リーダーシップ」**が望まれるようになってきました。

アジャイルの最も得意とするところは「変化への対応」です。それは「どうせ変更があるから」といって全く計画を立てないということではありません。まずは全体のおおよそのシナリオを構築しなければなりません。これを基本として個々の開発フェーズの活動を決めていきます。事前に何も準備しなければ、本当の「行き当たりばったり」になってしまいます。これではリスク管理もできずにプロジェクトが破たんすること請け合いです。新しい仕事を進める場合に重要なことは、先を見据えることです。洞察力を発揮して、考えられるあらゆる事態を想定して事前準備を行います。この活動を**「プロジェクト」**といいます。「プロジェクト（Project）」という英語は、「プロ（Pro）」と「ジェクト（Ject」」の合成語です。「プロ」は、「プロアクティブ」などのように「前」や「前方」という意味です。「ジェクト」は、「インジェクション（Injection）」などのように「投げる」ことを意味します。ここから「プロジェクト」は「前に投げる」あるいは「先を見通す」、そして「予測」というように使われます。ですから**事が起こってから**

動くのではなく、予測して事前に動くことが「プロジェクト」であり、重要なコンセプトであるといえます。アジャイルでも同じことで、開発サイクルを短くして何度も繰り返す手法でも、その短い開発期間における計画が必要です。当然、その期間の作業見積りなどが行われてエンジニアに作業割り当てがなされます。このときに機能追加やリスク対策などが織り込まれるのです。マニフェストで言っている「計画」とは、事前に決める包括的な全体計画のことで、それは変更の対象でありその内容に固執する必要はない、という意味です。「計画」に固執するあまり顧客が求める真の成果が達成できなかったら本末転倒です。実はPMBOKガイドにも同様なコンセプトがあり、「段階的詳細化」という考え方のもとに「計画策定プロセスには終わりがない」とされています。

1.2 アジャイル導入のメリット

アジャイル導入のメリットについて下記に示します。

1.2.1 時間とコストのパフォーマンス改善

アジャイルはその字の意味するように「敏捷」が目的ですが、個々の開発作業が早くなることを意味しているわけではありません。確かにケント・ベックらが提唱しているような「ペアード・プログラミング」や「テスト駆動開発」などの手法によって品質が改善され、ひいては開発の効率が向上しました。ただそれだけでしたら従来のウォーターフォール型でもその手法を採用すれば済むことです。

これは序論で示したように**「変更要求」への対応における問題を解決することによって「手戻り」を最小限に抑えることが可能になることを意味しています。**「変更」によるプロジェクトへの影響は後半になればなるほど顕著になり、スケジュール遅れやコスト増大につながります。特にソフトウェア開発では頻繁な変更によってスケジュールやコストのみならず品質にまでその影響が及んでいるのです。

ですから**「変更」を通常手続きの中に組み込んでしまえば予定外の作業工数とはならずに済む**のです。それこそケント・ベックが言うところの「変化を抱擁せよ」です。実際に 2008 年のカッター・コンソーシアムにおけるマイケル・マー氏の報告によれば、スクラム手法や XP 手法の採用は、世界中の 7,500 件ほどのプロジェクトと比較してコスト削減とスケジュール短縮に顕著な成果を上げています。

1.2.2 リスク低減

前述したように「変更」はプロジェクトに多大な影響を及ぼします。リスクは「起こりそうだが起こるかどうかわからない」事象を対象としていますが、**従来型プロジェクトでは「変更」をリスクのひとつとして捉えています。**計画が確定するまでは、顧客の要求を正確に把握すべく「変更」も要求のひとつとして扱いますが、いったん計画が確定し実行が開始されると「変更」はやっかいものという扱いになります。それをリスクとするのは、「変更」が事前に計画されていない活動だからです。

アジャイルでは、「変更」を定常的な事象として捉えることによって、**確実な**

対応が可能なようにしているのです。つまりソフトウェア開発プロジェクトにおける最大のリスクを回避することが可能になるのです。といってもすべてのリスクを「回避」できるわけではなく、多くのリスクが特定されたら、それぞれの特性に応じて「軽減」、「転嫁」、「受容」策などによって対応されなければなりません。「回避」というのは、リスク事象の発生確率か影響度を全く無いように“ゼロ”にすることですから、その対策には多大なコストや時間がかかります。通常は、リスクの影響度を「受容」できる程度に落とし込むための「軽減」策がとられます。無理に「回避」策を追い求めることは、資源の無駄使いにしかなりません。

1.2.3 生産性向上

生産性をどうやって定義するか、によって考え方は異なりますが「顧客へ引き渡す成果や価値を開発するための作業工数の割合」とすると、前述の手戻りを減らすことによって大幅に向上します。また**「サーバント・リーダー」**の下での自主的なチームを形成することによって、コミュニケーションも良くなり達成感ややりがいのある活動が期待できるので、メンバー間の補完や助け合いによるチームのシナジー効果が大きくなります。

1.2.4 高品質

成果物の品質はプロセスや作業の品質によって作り込まれます。アジャイル手法でよく採用されるテスト駆動開発や継続的統合などによって作業品質が向上し、また維持可能な作業ペースを守ることによって残業がなくなるのでミスが減ることになり、さらにエンジニアリング手法としての**ペアード・プログラミング**や**リファクタリング**などによって開発作業の効率化が図れます。結果的に成果物の品質が保証されることになります。

1.2.5 変更への対応力改善

変更を受け入れるプロセスを日常的に続けることによって、要求事項の優先順位付けやその変更への対応が速くなり、チームが有するマネジメント能力の向上が図れます。要するに、ルーティン化してしまえば効率がよくなる、ということです。

1.2.6 作業要員の関与と満足度向上

チーム・メンバーは作業計画や見積りに積極的に関与することになり、ひいてはその結果に責任をもつので、達成感や満足度が向上し、モラルが向上します。実際に、バージョンワンの調査（2011 年）やマイクロソフトにおける社員の満足度調査（2006 年）では、顕著な結果が出ています。

1.2.7 市場投入への時間短縮による投資対効果の改善

動くソフトウェアを徐々に繰り返し提供するということは、最終成果物の完成まで待たなくても、それだけビジネスへの貢献が早まることになります。当然、それだけ**投資対効果（ROI；Return On Investment）**が良くなります。

1.2.8 ステークホルダー満足の改善

ROI が改善されるということは、すなわちビジネスへの貢献ができたことになりますので、ステークホルダーの満足度は向上します。

1.3 アジャイル導入における課題

アジャイル導入における課題には次のものがあります。

1.3.1 最終成果物

アジャイル手法は繰り返される変更や追加を積極的に取り込んでいくので、最初の時点で最終成果物の詳細を定義できません。ビジネスの目標達成のための全体像を描写してビジョンとします。プロジェクトはその目標を達成するための手段です。場合によっては、複数の関連するプロジェクトの成果によって目標が達成されることもあります。この場合、全体の活動を「プログラムマネジメント」と定義しています。

1.3.2 予算

最終成果物（スコープ）を確定できないということは、**請負契約での総額を事前に決められない**ということになります。したがって見積精度は、いわゆる確定見積り（-5%～+10%）にはならないので、概算見積り（-25%～+75%）から予算見積り（-10%～+15%）程度の精度とならざるを得ません。

1.3.3 スケジュール

スケジュールは、顧客が希望する最終成果物の納品日に合わせて設定可能ですが、実装途中に追加や変更を加えた成果物のスコープに合わせて調整しなければなりません。要するに契約事項もスケジュール変更に柔軟に対応する必要があります。契約については後述しますが、さまざまな観点で見直しが必要です。

1.4 アジャイルを適用できる分野と適用しにくい分野

2011 年に発表されたラルフ・ステーシーによる「ステーシー・マトリックス」が有名です。縦軸はコミュニティにおける合意の程度を表し、横軸は確実性を表しています。右上に「混沌（カオス）」の領域がありますが、これは全くの混乱を意味しているのではなく「プロジェクトの対象とはしない」という意味です。従来型のプロジェクトは、左下に位置する環境において実施すべきであって、たとえばスコープが不明確なまま開発期間や予算を決めてしまうようなことをすべきではありません。不明確な環境を「複雑」と表していますが、この領域こそアジャイル型の出番なのです。左下の領域でアジャイル型プロジェクトを進める意味はありません。これから実施しようとしているプロジェクトの環境がステーシー・マトリックスにおけるどの位置にあるのかをステークホルダーとじっくり検討し、プロジェクトのライフサイクルを決め、リスク対策を策定する必要があります。

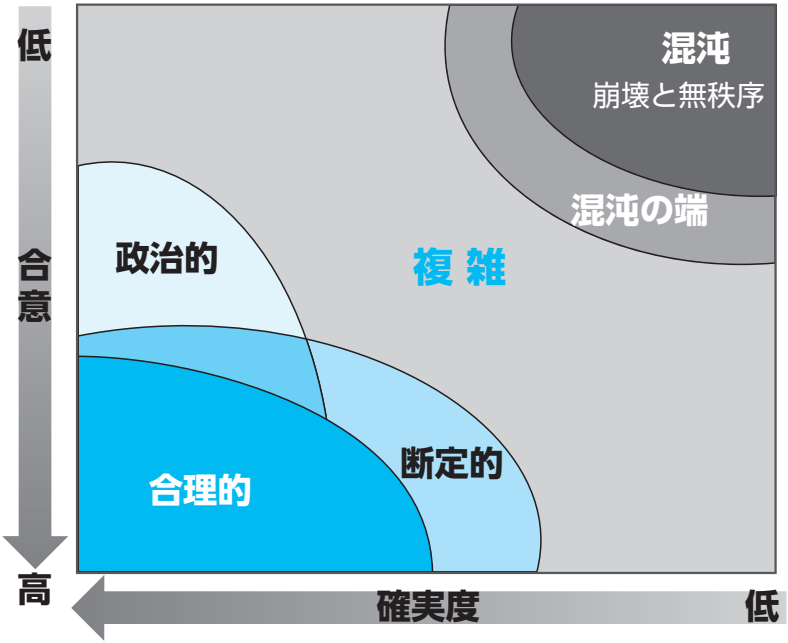


図 1.1 ステーシー・マトリックス

1.5 アジャイル導入の障壁と困難にする要素

アジャイル導入の障壁と困難にする要素として次のものがあります。

1.5.1 アジャイル導入後の課題調査

2006 年から 2011 年にかけてさまざまな調査が行われてきました。バージョンワン、スコット・アンブラー、PMI、マイクロソフトなどで行われた調査のほとんどでアジャイル導入の成功が示されていますが、中には失敗したケースもありますので、その理由や課題を見てみましょう。

1.5.2 失敗要因

代表的な失敗要因として次のものがあります。

① アジャイルへの理解不足

アジャイルの正しい理解をもたないまま、ただ単に「早くヤレ！」というマネジメント・スタイルや成果物の「質」や「量」だけを追い求めて無理をさせ、結果として作業員の健康を損なってしまうケースや離職率の増加が見られます。個人の成長なくして組織の成長はあり得ません。これはアジャイルに限った話ではありませんが、プロジェクトの制約条件間のバランスや全体最適の考え方が重要で、納期などの制約条件の柔軟性が求められます。

② 組織変革の必要性への理解不足

アジャイルに必要なのは、**「協業」**です。ビジネス側と開発側の意思統一と協業によって本来の目的を達成するように進めなければなりません。ここには「おまかせ」はありません。発注側と受注側の力関係に影響されるような仕事の進め方ではなく、同じ土俵の上で問題を解決しプロジェクトを進捗させる必要があります。上位マネジメントも、建前だけの報告書を要求するのではなく、現場の状況を自分の目で確認するくらいの活動が求められます。そのための組織変革が要求されます。

③ 組織文化とアジャイル実務との不整合

アジャイルの「協業」のためには、縦割り文化はあまり機能しません。また、チームの自主性を重んじるので自ずと新しいリーダーシップ・スタイルが必要になります。「指示型」や「指揮型」を好む組織文化にはなじみません。アジャイル型では**「サーバント・リーダー」**のスタイルが求められるので、上位マネジメントから「しっかり指示をしたか！」と叱責されるような状況ではうまくいきません。それよりも「コーチング」や「メンタリング」スタイルの方が適切です。

1.5.3 アジャイル適応拡大の障壁

たいていの組織では、最初、試験的にアジャイルを導入しますが、その結果を受けて組織内に拡大しようとします。そのときの問題を見てみましょう。

① 組織文化とアジャイル文化の不整合

組織における調達マネジメントにも影響を受けます。ベンダーとの契約を購買部門が取り仕切る場合には、**プロジェクト組織と購買部門およびベンダーがそろってアジャイルについて理解していることが重要**です。これはなかなか大変なことですが、時間をかけて教育していきましょう。

アジャイルの採用は、組織の人材育成におけるリーダーの育成方法にも影響を及ぼします。リーダーは、状況に応じたリーダーシップ・スタイルを採用することは、当然大切なことですが、**「サーバント・リーダー」は一朝一夕に育成できるものではありません。**ちょっとトレーニングを受けただけでは十分とは言えないものです。時間がかかりますので、そのためには良き相談相手としてのコンサルタントが必要になるかもしれません。

② アジャイル熟練者不足

世の中には熟練者はそう多くありません。したがって自組織で育成せざるを得ません。エンジニアとして技術的なトレーニングも必要ですが、協業のための態度や振る舞い、自主性、責任感、などの醸成が求められます。**「指示待ち」人間では役に立ちません。**当初は先輩の指導を受けながら徐々に経験を積んで育っていくものですから、メンタリングやコーチングは必須の手法です。

③ 変化への抵抗

アジャイルに限らず新しいことに抵抗する人は必ずいるものです。「今までこれでやってきたから」とか、単に「変えたくない」など、変革や改革へのレジスタンスやアレルギーは多かれ少なかれあるものだという前提に立って、関係者との折衝や交渉にあたります。成功のためにはトップダウンの説得が手っ取り早いのですが、その前には十分なネゴが必要でしょう。これらの活動を**ステークホルダー・マネジメント**と呼び、プロジェクトマネジメント国際標準として2012年9月に発表されたISO21500で採用されているマネジメント領域です。プロジェクト関係者の支持を得て、プロジェクトを確実に成功させるための活動としてアジャイルでも努力しなければならない分野です。

1.5.4 困難にしている要素

他にもいくつか障害となる要素が考えられます。

① 基本的信念への固執

従来型プロジェクトで成功を収めてきた経験者に多く見られる現象として「きちんと計画を立ててから実行しなくては成功しない」と言い張ってアジャイルを軽視する傾向があります。計画を立てることはアジャイルでも同じです。**マネジメント・プロセスとして考えると、アジャイル型を別の側面で見れば、計画駆動型の進め方の単位（フェーズ）を縮小して頻繁に繰り返すようにしているだけです。**それを効率よく推進するための手法としてチームの自主性を重んじるようになったのですから、進歩型ともいえるでしょう。前述したように、従来型もアジャイル型もその適用領域があるものです（図1.1 ステーシー・マトリックス参照）。

② 管理職の不理解

管理職のリーダーシップ・スタイルにも依存しますが、従来の計画駆動型に固執する人は、上位マネジメントへの報告のための進捗管理や文書化にとらわれ過ぎているように見受けられます。特に前述した「指示型」リーダーシップを好む管理職は、どちらかというと権限委譲が苦手な人が多いので、一から十まで管理しようとする傾向にあります。これではチームの自主性は削がれてしまうし、結果的には「指示待ち人間」や、同じような「指示型人間」を作ってしまいます。

③ トップダウン対ボトムアップ

欧米の企業に多く見られるのはトップダウン型経営ですが、日本ではどちらかというとボトムアップ型経営の方が多く見られます。どちらにも、その特徴に由来するメリット・デメリットがありますが、アジャイルはボトムアップ型の方がそのメリットを享受しやすいといえます。ところが、欧米のアジャイルの方が日本より先行している現状は非常に興味深いものです。後述しますが、例えば**カンバン方式は、トヨタの大野耐一氏が考案して実践した現場中心の生産工程がその基になっている考え方**です。その優秀さが世界中に注目され、米国で研究されてトップダウン型の工程管理になりました。「看板」とせず「カンバン」としているのは、米国から輸入されたトップダウン型を意味しているからです。

④ 予測不可能性

アジャイルでは、繰り返される変更や追加のために最終成果物を完全に予測することは困難です。ということは**コストやスケジュールについても同じことが言えます。**これが経営にとって頭の痛いところです。それをうまく進めるためには**スケジュール・バッファーや予備費を十分に用意する必要がある**のです。本来、バッファーや予備費はリスク対策のひとつですが、「余裕」と誤解されやすいものです。バッファーは「衝撃を吸収する」ための道具なので、不確実性に対応するためには必須のツールといえるでしょう。

ひとつの例として、歯車がかみ合って回転する様子を思い浮かべてみましょう。歯と歯の間にある程度の「遊び」がないとスムーズに回転しませんが、この遊びがバッファーに相当します。予測不可能性をリスクとして捉えて対応することが必要です。

⑤ 広範囲な変化

アジャイルを進めていくと、プロジェクトのみならず組織の中にさまざまな変化が起こります。プロジェクト・チームにおける新しいコミュニケーション手法の効果が認められると、その経験が組織全体に広まって自然な改善活動や改革が進んでいきます。それを管理職や上位マネジメントが認識し後押しすることによって組織全体へ波及していくのです。それがボトムアップとトップダウンの融合とでも言える大きな変化なのです。

⑥ 従来型との違いの大きさ

今まで述べてきたように、計画駆動型にどっぷり浸かっていた管理職にとっては、全く別の会社に転職したような感覚を覚えることになります。それが変革です。アジャイルはイノベーションである、と言われる所以です。その違いに戸惑うことが予想されます。

1.6 アジャイル導入物語「第１話　振り返り」

鈴木悠友（ゆゆ）は、担当するプロジェクトの終結プロセスに取り掛かかっていました。顧客の求めに応じたPOSシステムの全国展開が終わり、堂々としたメディア発表も成功し、一段落したところで、プロジェクトの完了報告書や教訓の作成作業が待っています。彼女は、あらゆる書類を目の前にして、ほっとため息をつきました。これはこれからの作業量のためではなく、プロジェクト中に起こったさまざまな問題が頭の中を駆け巡ったからです。プロジェクトに問題は付き物である、とは重々承知していたのですがこれ程とは思っていませんでした。自分の甘さを思い知らされたようでもあるのですが、今になってみると自分の頑張りを褒めてあげたいし、結果的には「成功」の一文字を獲得できたことに大きな喜びとともに自信がみなぎってきたのです。

報告書は、事実関係からいろいろな統計情報を整理して図表を取り入れた読みやすいものとしてステークホルダーへ配布しました。顧客をはじめとする主要なステークホルダーには挨拶を兼ねて手渡しとして口頭による簡単な説明を加えました。この時点ではまだ教訓をまとめていなかったので、「**教訓**はなんだったのか」と質問する役員もいましたが、口頭での受け答えだけにし、それは後日提出することとして席に戻ってきました。

さあ教訓をまとめなければなりません。関係者と一緒に考えるのが本来のやり方ですが、まず自分だけで考えてみたかったのです。彼女の頭の中を走馬灯のように駆け巡るプロジェクトのシーンが当時の苦しみとともに襲い掛かってきました。悠友は少々のめまいを感じたので濃い目の珈琲に手を伸ばしました。一瞬目を閉じて

「そういえば一番苦しかったことはなんだったのか」と自問自答したのです。

「自分も辛かったけど、やっぱりスタッフのみんなに苦労かけたことかな、せっかくやった仕事を何度もやり直させたり、徹夜させたり、泣かれたなあ」

「お客にはわがまま言われたけどこれは仕方ないとして、社内で孤立したこともあった」「でもそもそもの原因は、やたらと変更が多かったことだな」

「最初にしっかり計画を立てたんだけど、まったく役に立たなかった」

「やっぱり二年計画は無理なんだろうか」

「というより一年目から崩れたなあ」

「どの辺から崩れたんだっけ・・・」と、続いていきます。

結局、変更の多さに辟易したことを中心に考えることにしました。
「要件定義はできていたのに、どうしてあんなに要求事項が追加されたり変更されたりするのだろうか」
「お客はどう考えていたんだろうか」
当時を思い起こすと、顧客の担当者の言葉が浮かんできます。
「最初はあれでいいと思ったんだけど、時間が経つといろいろ出てくるんだよ。ああもしたい、こうもしたい、とね」
「ハードを作り直すのは大変だろうけどソフトは簡単なんじゃないの、パソコンでカシャカシャってやれば直ぐできるんでしょ？」
「変更が大変だというけれど、それを何とかするのがプロなんじゃないの？」
「進捗っていったって、ソフトの作業は見えないからわからないよ、報告書を信じるしかないよ」
「何か具体的に動かして見せてくれれば分かりやすいんだけどなあ」
わー、彼女の頭が火の海になって机にうつ伏せになったときに携帯が鳴りました。
「鈴木さん、今日は打ち上げのパーティだよ、忘れてないよね？」
いけない！うっかりしてた。悠友の友人のひとりでプロジェクト・メンバーでもある寺沢さんからの電話でした。
「待ってて、すぐ行くから」と机の上を片付けながら叫ぶと駆け出しました。
悠友はパーティ会場で挨拶した後、すぐに寺沢さんに相談しました。「千鶴子さん、後で相談に乗ってください」。寺沢千鶴子さんは、悠友より年上でプロジェクト・チームのなかの姉貴分です。ふたりは、会の後で、近くの喫茶店で落ち合うことにしました。悠友が二次会の誘いを振り切って約束の場所に急ぐと、寺沢さんだけでなく主なチーム・メンバーが揃っていました。
「どうしたの？みんなで」悠友が訝しがると仲間の間に笑いが広がりました。
「悠友さんから相談があるといったので、みんなを集めたの」
「えっ？」
「だって今頃になって悠友さんが相談なんて、変でしょ？タイミングとしてはレポート書いている頃だから、その関係かなって思ったの」
「確かに！千鶴子さんは凄いわ、なんでもよくわかるわね、だから頼りになるのよ」
みんなが頷く。悠友は珈琲を頼むと、早速いきさつを話し始めた。

閑話休題

振り返りと教訓の違い

アジャイル系のプロジェクトでは、レトロスペクティブという用語を使用します。辞書には「回顧」や「回想」という意味が出ていますが、一般に「振り返り」と言っています。この言葉のニュアンスからは、あまり堅苦しくない「思い出」みたいな軽い意味が感じられます。一方「教訓」はかなり公式度が高く、例えば教訓のデータベースへ登録したり、報告書へ記述したりするので、なかなか書きずらいものです。

プロジェクトでは、日々のちょっとしたミーティングや短いサイクルでの振り返りを重視していますが、プロジェクトの終結では完了報告書と同時に教訓を残すことが大切です。これによってメンバーに培われたノウハウが文書化され、次のプロジェクトに役に立っていきます。一般にプロジェクト・レビューなどのタイミングでも教訓の収集が行われます。