非ウォーターフォール型開発の
普及要因と適用領域の拡大に関する調査

～国内の中規模及び大規模開発プロジェクトへの適用事例調査～

調査報告書

平成 24 年 3 月 28 日

独立行政法人情報処理推進機構
技術本部 ソフトウェア・エンジニアリング・センター

はじめに

ＩＰＡ／ＳＥＣでは、「日本のソフトウェア競争力を高め、エンジニアがいきいきと働ける環境を作るため、日本国内における非ウォーターフォール型開発の普及促進と、わが国のソフトウェア産業の特性に照らした非ウォーターフォール型開発の課題解決」を目標に掲げて一昨年から「非ウォーターフォール型開発の調査」に取り組んできた。その中でも、これまで事例数が少なく十分な調査が行えていなかった、中規模、大規模開発プロジェクトに適用する開発技術及び管理技術の事例調査を実施し、調査結果を報告書としてとりまとめました。

本調査事業は、「非ウォーターフォール型開発の普及要因と適用領域の拡大に関する調査事業」として、株式会社永和システムマネジメントに委託し、実施しました。

Copyright© Information-Technology Promotion Agency, Japan. All Rights Reserved 2012

目次

〜国内の中規模及び大規模開発プロジェクトへの適用事例調査〜 i
調査報告書 i
1. 背景と目的 1
1.1. 調査に至る背景 1
1.2. 非ウォーターフォール型開発とプロジェクト規模 3
1.3. 目的 5
2. 大規模における調査課題とポイント 6
2.1. はじめに 6
2.2. 「対話」に関して調査課題 6
2.2.1. 組織体制とコミュニケーション 7
2.2.2. 組織への展開方法 7
2.2.3. 分散拠点開発 8
2.3. 「動くソフトウェア」に関しての調査課題 8
2.3.1. アーキテクチャ・共通基盤の構築手順 8
2.3.2. システムの分割と統合 9
2.4. 「顧客との協調」に関しての調査課題 9
2.4.1. 明確な問題意識と効果 9
2.4.2. 各種管理 10
2.5. 「変化への対応」に関しての調査課題 10
2.5.1. 部分適用 11
2.6. 全体に関する前提条件・背景 11
2.6.1. 組織文化調査 11
2.6.2. 従事者健康度 12
3. 調査方法 14
3.1. 調査の流れ 14
3.2. 調査方法 15
3.2.1. 【調査方法 1】問題と解決策と、その結果の関連調査 15
3.2.2. 【調査方法 2】時系列変化を調べる 15
3.2.3. 【調査方法 3】アンケート＋インタビュー 16
3.2.4. 【調査方法 4】複数ロールを調査対象 16
3.2.5. 【調査方法 5】組織文化調査 16
3.3. 調査項目 16
4. 事例別レポート 20
4.1. 事例情報の見方 20
4.2. 事例Ａ社 22
4.2.1. 特徴 22
4.2.2. 非ウォーターフォール型開発適用の背景 23

Copyright © 2012 IPA, All Rights Reserved

4.2.3. 組織構成 ... 24
4.3. 事例 B 社 ... 25
4.3.1. 特徴 ... 25
4.3.2. 非ウォーターフォール型開発適用の背景 ... 26
4.3.3. 組織構成 ... 27
4.4. 事例 C 社 ... 28
4.4.1. 特徴 ... 28
4.4.2. 非ウォーターフォール型開発適用の背景 ... 29
4.4.3. 組織構成 ... 30
4.5. 事例 D 社 ... 31
4.5.1. 特徴 ... 31
4.5.2. 非ウォーターフォール型開発適用の背景 ... 31
4.5.3. 組織構成 ... 31
4.6. 事例 E 社 ... 32
4.6.1. 特徴 ... 32
4.6.2. 非ウォーターフォール型開発適用の背景 ... 33
4.6.3. 組織構成 ... 34
4.7. 事例 F-1 社 ... 35
4.7.1. 特徴 ... 35
4.7.2. 非ウォーターフォール型開発適用の背景 ... 36
4.7.3. 組織構成 ... 37
4.8. 事例 F-2 社 ... 38
4.8.1. 特徴 ... 38
4.8.2. 非ウォーターフォール型開発適用の背景 ... 38
4.8.3. 組織構成 ... 39
4.9. 事例 G 社 ... 40
4.9.1. 特徴 ... 40
4.9.2. 非ウォーターフォール型開発適用の背景 ... 41
4.9.3. 組織構造 ... 42
4.10. 事例 H 社 ... 43
4.10.1. 特徴 ... 43
4.10.2. 非ウォーターフォール型開発適用の背景 ... 44
4.10.3. 組織構成 ... 45
4.11. 事例 I 社 ... 46
4.11.1. 特徴 ... 46
4.11.2. 非ウォーターフォール型開発適用の背景 ... 47
4.11.3. 組織構成 ... 48
4.12. 事例 J 社(参考) ... 49
4.12.1. 特徴 ... 49
4.12.2. 非ウォーターフォール型開発適用の背景 ... 50
4.12.3. 組織モデル ... 51
4.13. 事例 K 社(参考) ... 52

Copyright © 2012 IPA, All Rights Reserved

4.13.1. 特徴 52
4.13.2. 非ウォーターフォール型開発適用の背景 53
4.13.3. 組織モデル 54

5. テーマ別分析結果 55
5.1. アジャイル型開発の選択 55
5.2. 組織体制についての工夫 56
5.2.1. 準委任契約 56
5.2.2. 職能横断チーム 57
5.2.3. 役割を超えた支援 57
5.2.4. チームメンバーローテーション 58
5.2.5. 不適合メンバーの入れ替え 59
5.3. コミュニケーションについての工夫 60
5.3.1. 全員同席 60
5.3.2. 見える化 61
5.3.3. 段階的朝会 62
5.3.4. 完全透明性 62
5.3.5. チームまたぎのふりかえり 63
5.4. 展開についての工夫 64
5.4.1. パイロット導入 64
5.4.2. 認定研修・コンサルタントの利用 65
5.4.3. 漸進的な人数増加 66
5.4.4. 漸進的な展開 67
5.4.5. 社内研修・勉強会 68
5.5. 分散開発についての工夫 69
5.5.1. 同一拠点から分散へ 69
5.5.2. チケットシステム 70
5.5.3. TV 会議 72
5.5.4. リアルタイムチャット 72
5.6. アーキテクチャについての工夫 73
5.6.1. 最初のアーキテクチャ構築 73
5.6.2. アーキテクチャ・基盤チーム 74
5.6.3. 組織の共通基盤アーキテクチャの利用 74
5.6.4. アーキテクチャの改善 75
5.7. システム分割／インテグレーションについての工夫 75
5.7.1. 疎結合での分割 75
5.7.2. フィーチャーチーム 76
5.7.3. 早期からのインテグレーション 77
5.7.4. 同じリズム 78
5.7.5. 継続的インテグレーション 78
5.8. 品質についての工夫 79
5.8.1. 重視するビジネス価値 79
5.8.2. ビジネス価値の変化 81

Copyright © 2012 IPA, All Rights Reserved

5.8.3. タイムボックス優先の品質 81
5.8.4. 適切なピア・レビューの選択 82
5.8.5. 自動化された単体テスト 83
5.8.6. テストフェーズ 84
5.8.7. 第三者テスト 84
5.9. 部分適用についての工夫 85
5.9.1. 必要な部分のみ適用 85
5.9.2. 疎結合なチーム 85
5.9.3. 工程の見える化 86
5.10. 組織文化の傾向 87
5.10.1. 組織文化調査について 87
5.10.2. OCAI の二つの軸と四つの象限 88
5.10.3. 調査結果(全体) 90
5.10.4. 組織文化結果(個別) 91
5.10.5. 考察 94
5.11. 従事者の精神健康度 96
5.11.1. 調査方針について 96
5.11.2. 調査方法 96
5.11.3. 調査結果と考察 97
6. 調査ポイント以外で発見された課題 98
6.1. 全体計画の把握困難 98
6.2. 新手法への期待過大 98
6.3. 手法移行の問題 99
6.4. ビジネス企画側にボトルネック発生 99
6.5. 反復中の品質悪化のしわ寄せ 100
6.6. 他組織とのリズムの不適合発生 101
6.7. 管理工数増加や開発速度の低下 101
7. 想定との比較 102
7.1. 組織体制とコミュニケーションについての想定との比較 102
7.1.1. 職能横断チーム 102
7.1.2. 直接的コミュニケーション 102
7.1.3. 準委任契約 102
7.2. 組織への展開方法についての想定との比較 103
7.2.1. 段階的適用 103
7.2.2. 変化度が高い部分を起点 103
7.2.3. リーダーシップ 103
7.3. 分散拠点開発についての想定との比較 104
7.3.1. 同一拠点から段階的分散 104
7.3.2. コミュニケーションツールの活用 104
7.4. アーキテクチャ・共通基盤の構築手順についての想定との比較 105
7.4.1. アーキテクチャの漸進的開発 105

Copyright © 2012 IPA, All Rights Reserved

7.4.2. アーキテクチャチーム 105
7.5. システムの分割とインテグレーションについての想定との比較 105
7.5.1. サブシステム間の継続的インテグレーション 105
7.5.2. 疎結合なシステム 106
7.5.3. 機能単位のチーム 106
7.6. 明確な問題意識と効果について想定との比較 107
7.6.1. 経営者や顧客の判断 107
7.6.2. 問題意識、期待、効果の明確さ 107
7.7. 各種管理について想定との比較 110
7.7.1. テストフェーズの存在 110
7.7.2. 品質に関するプラクティス 110
7.8. 部分適用についての想定との比較 111
7.8.1. 変化の必要な部分のみの適用 111
7.8.2. 同期ポイント 111
7.9. 組織文化についての想定との比較 111
7.9.1. 異組織文化間移行の問題 111
7.9.2. スムーズな組織文化移行 112
7.9.3. 管理度合い 112
7.10. 従事者健康度についての想定との比較 112
7.10.1. 精神的健康度 112
8. 考察と提言 113
8.1. 社会的要請と適応範囲 113
8.2. 中大規模事例での工夫と課題 114
8.3. 中大規模の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 115
参考文献 116
付録１：データ編（精神的健康度） 117
付録２：データ編（工夫一覧） 118
付録３：調査票 142

Copyright © 2012 IPA, All Rights Reserved

1. 背景と目的

1.1. 調査に至る背景

アジャイル型開発を中心とする非ウォーターフォール型開発は、従来のウォーターフォール型開発の課題を解決するソフトウェア開発手法として期待されている。近年は、ビジネス環境の変化に俊敏に対応でき、ソフトウェアの開発着手から市場投入までに要する期間を短縮する手法として注目され、普及してきている。

独立行政法人情報処理推進機構 技術本部 ソフトウェア・エンジニアリング・センター (IPA/SEC) では、「日本のソフトウェア産業の競争力を強化すること」、「エンジニア一人ひとりが生き生きと働ける環境を作ること」を目指すべきゴールとして、平成 21 年度から「非ウォーターフォール型開発」の調査・検討に取り組んできた。

この 2 年間 (平成 21 年度・平成 22 年度) の調査で、5 つの「重点課題」と 4 つの「試行領域」（適用経験が十分には蓄積されていない領域）が明らかになった。

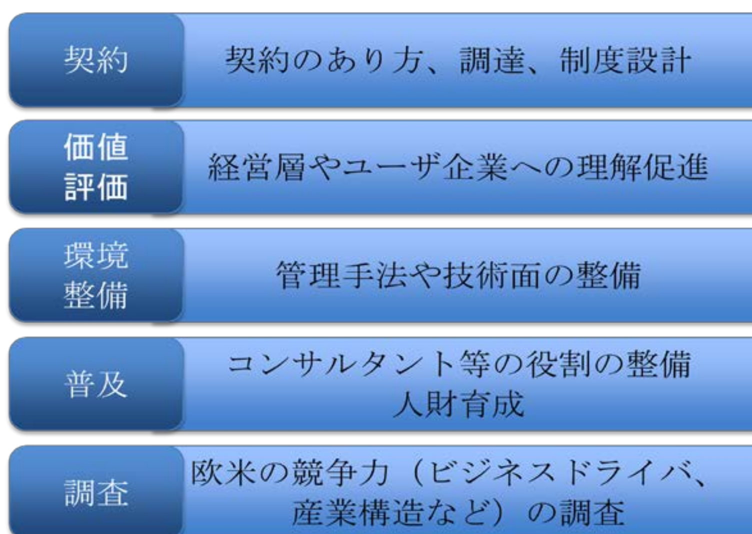

図 1-1 ５つの重点課題

Copyright © 2012 IPA, All Rights Reserved

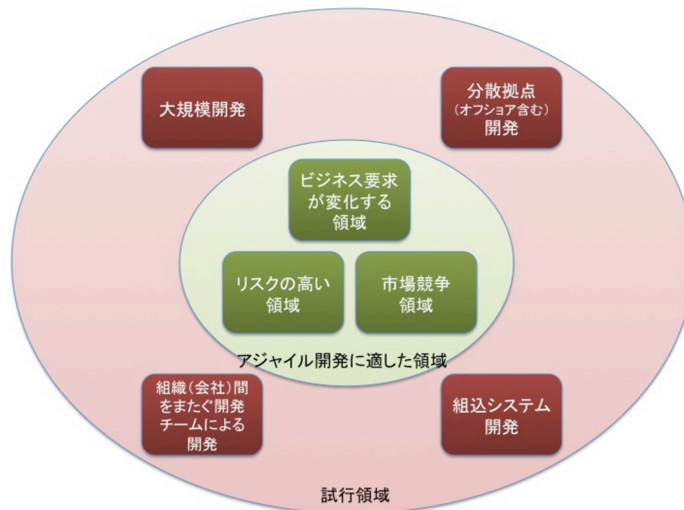

図 1-2 ４つの試行領域

５つ重点課題の多くについては、その検討結果を非ウォーターフォール型開発 WG 活動報告書[1]で公開している。以下の点については、事例数が少なく十分な調査が行えなかった。

「管理手法や技術面の整備」における、中規模、大規模開発プロジェクトに適用する開発技術及び管理技術の事例調査

また、試行領域についても、非ウォーターフォール型開発による経験が十分には蓄積されておらず、具体的な施策の提示には至っていない。本事業では、残された重点課題の解決及び試行領域への適用拡大のため、以下の調査を実施した。

- 中規模や大規模プロジェクトでの非ウォーターフォール型開発の適用状況や適用範囲を拡大するための課題についての調査

これらの調査結果を残すことにより、以下の成果に寄与することを目的とした。

- 日本のソフトウェア開発における非ウォーターフォール型開発の普及促進
    - 日本の競争力強化
- 環境の変化に強くかつ利便性の高い情報システムの実現
    - 国民生活の向上

本調査は、非ウォーターフォール型開発の中でも特に注目されているアジャイル型開発に焦点を当て調査することとした。

[1] http://sec.ipa.go.jp/reports/20110407.html

Copyright © 2012 IPA, All Rights Reserved

1.2. 非ウォーターフォール型開発とプロジェクト規模

非ウォーターフォール型開発手法であるアジャイル型開発について、2001年に作成されたアジャイル宣言[2]はその特徴、価値観を以下のように表現している。

私たちは、ソフトウェア開発の実践
あるいは実践の手助けをする活動を通じて、
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、私たちは以下の価値に至った。

プロセスやツールよりも個人と対話を、
包括的なドキュメントよりも動くソフトウェアを、
契約交渉よりも顧客との協調を、
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。

つまり、「協調とコミュニケーションを重視しながら、動くソフトウェアを少しずつ成長させながら、変化に対応していく」という形でソフトウェアシステムを開発していくことになる。この宣言が発表されてから11年が経過した。様々な批判もあったが、現在では欧米を中心に一つの選択肢として普及してきている。その現れとして、数々のITプロジェクトを調査している米Standish Group[3]の「CHAOS Manifesto 2010」では、10の成功の要因の一つとしてアジャイルプロセスが含まれているし、プロジェクトマネジメントの権威であるPMI(Project Management Institute)にも、認定制度である、PMI-Agile Certification Practitioner(PMI-ACP)[4]が登場している。単純に開発者が好む手法という領域から、プロジェクトを成功させる手法として広く認知されつつある。

一方、協調やコミュニケーションを重視する手法ということもあり、以前からその効果を発揮する領域としては少人数に限定されるという指摘も数多くあった。Philippe Kruchten氏はアジャイル型開発のスイートスポットを次のように定義している。(図 1-3)
特にその規模について見てみると「小規模人数」がスイートスポットであると捉えられている。

[2] http://agilemanifesto.org/iso/ja/
[3] http://blog.standishgroup.com/
[4] http://www.pmi.org/en/Certification/New-PMI-Agile-Certification.aspx

Copyright © 2012 IPA, All Rights Reserved

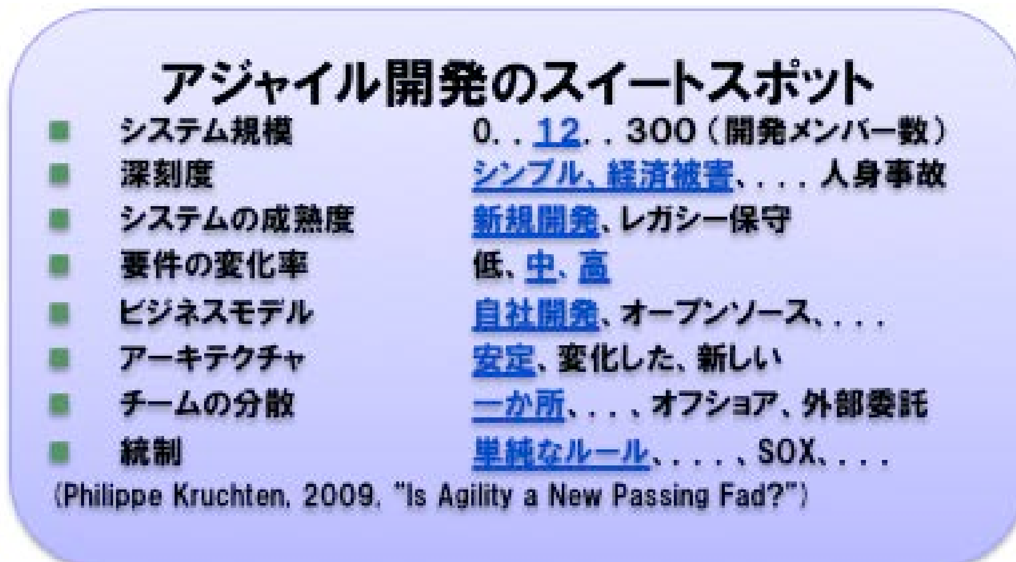

図 **1-3** アジャイル開発のスイートスポット

しかし近年において、海外におけるアジャイル型開発の大規模対応についての書籍も見かけられるようになってきた。それらの事例ではプロジェクト関与者が 100 人以上のプロジェクトを対象に扱っており、本来のスイートスポットから外れた、大規模の領域においてもアジャイル型開発が可能であることを示唆している。

他方、国内に目を向けてみると、IPA にて平成 21 年度に調査された非ウォーターフォール型開発を採用したプロジェクトの事例調査においては、22 事例のうち 30 人以上のプロジェクトは皆無であった。

Copyright © 2012 IPA, All Rights Reserved

1.3. 目的

平成 21 年度調査から二年が過ぎたが、近年、国内においても中規模、大規模での事例が発表される機会が増えてきた。他方、依然としてアジャイル型開発のスイートスポットから外れた部分についての適用の課題は残っている。アジャイルマニフェストの４つの宣言に対応した阻害要因として以下が挙げられる。

(1) 人数が増えることにより対話の困難さ
(2) システム規模の拡大に伴うシステムの分割や統合の困難さ
(3) 顧客との協調を実現するのに、従来の進め方とのギャップの埋め方や、顧客の負荷の増大
(4) システム規模が増えると、変化に適応する部分とそうでない部分の分離

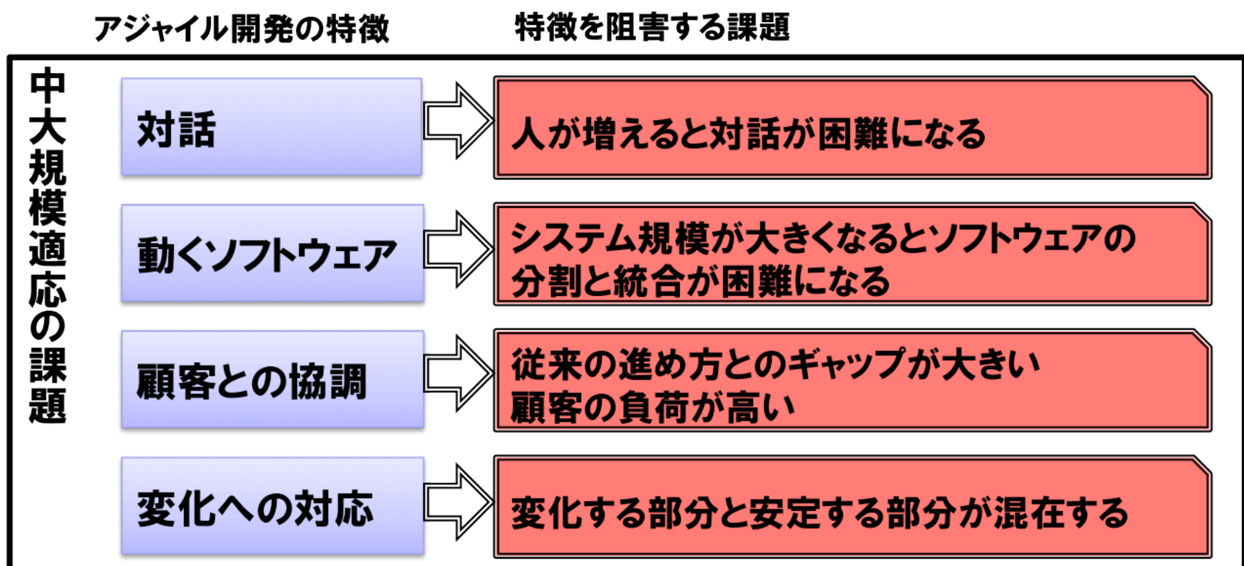

図 1-4 中大規模適応の課題

そこで今回の調査においては、非ウォーターフォール型開発のうちアジャイル型開発に着目して、中規模(30 名以上 100 名未満)、あるいは大規模(100 名以上)の非ウォーターフォール型開発を用いたプロジェクトに絞って、これら４つの課題についての調査を行い、各事例で行われている工夫を収集して成功の秘訣をまとめることで、読者に対して、中大規模プロジェクトにおける非ウォーターフォール型開発の適用の指針になることを目的とする。

Copyright © 2012 IPA, All Rights Reserved

2. 大規模における調査課題とポイント

2.1. はじめに

前章では、平成 21 年度では調査できなかった中大規模事例についての調査の背景と目的を述べた。本章では、４つの課題に対応した調査ポイントとその想定結果を述べる。

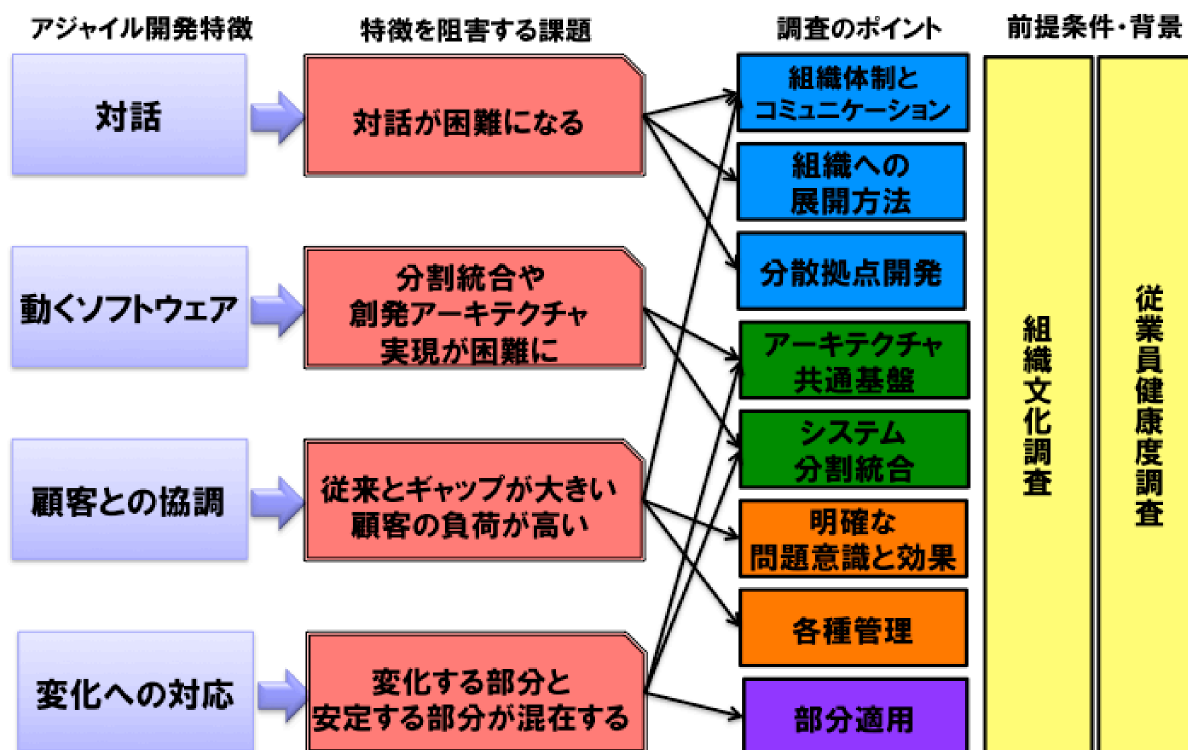

図 2-1 調査課題とポイント

2.2. 「対話」に関して調査課題

「対話が困難になる」という課題について、まず考えなければならないのが、人数が増え、チームが分割されるなど組織複雑性の増加である。組織複雑性が増すほど、協調やコミュニケーションは困難となる。アジャイル型開発は、協調やコミュニケーションを重視しているが故に、そこが機能しなくなってしまうと、その効果が大きく損ねてしまう。そこで、組織複雑性の観点から、調査ポイントとして、下記の３つの項目を設定した。

(1) 組織体制とコミュニケーション

(2) 組織への展開方法

(3) 分散拠点開発

以降に各調査ポイントについて述べる。

Copyright © 2012 IPA, All Rights Reserved

2.2.1. 組織体制とコミュニケーション

アジャイル型開発においては、効果的なコミュニケーションが必要不可欠である。しかし組織の規模が拡大すればするほど、効果的なコミュニケーションが困難になる。更にはチーム内のコミュニケーションだけではなく、チーム間や組織間のコミュニケーションが必要になる。コミュニケーションを重視しているが故に、組織規模拡大による複雑性が増す中で、どのような体制で、どのようにプロジェクト全体としてコミュニケーションについて工夫をしたのか、についての知見を収集する。

また、組織構造については、平成22年度報告資料における「アジャイル型開発のビジネス構造モデル」と「役割」をベースにして、構造と関係性を明らかにすることで、中大規模における組織システムのモデルを導出する。また、チーム毎の役割分担においては、非ウォーターフォール型開発で重要視される職能横断チームが実際の事例においてどの程度構成されているのかにも着目する。職能横断チームとは、一つのチームが同じ専門家だけ(開発者のみ、テスターのみ、DBA（データベース管理者）のみ)で構成されるのではなく、一つのチームで、あるシステムの機能を設計、実装、テストを経て出荷できるためのスキルを持つ各部門の人々で構成されたチームである。

また、顧客との協調を重視するがために、高い頻度で開発チームに次々と要件を提示し、リリースや反復期間の計画づくりに参加し、開発の最中に開発者からの質問に答え、反復の度に実現されるソフトウェアのレビューや、受入テストの実施、など、これまで以上の密度でプロジェクトへの参画が求められる。

本調査ポイントでの想定結果は次の通りである。

- 各チームは職能横断チームによって構成されている。
- チーム間のコミュニケーションおいて次のような情報共有手法を実施している
  - 見える化を活用して全体として情報共有を行っている。
  - 頻繁なチーム間の情報共有(階層的朝会の実現)を実現している。
  - オープンなスペースに全員同席している
- 顧客と開発との契約は準委任で行われている。

2.2.2. 組織への展開方法

大規模プロジェクトにおいては、最初から大勢の人数で開始する、あるいは一斉に全員で非ウォーターフォール型開発を適用することは考えにくい。プロジェクトに関わる人数を段階的に増やす、あるいは適用人数を段階的に増やしていくことが推測される。また、対象プロジェクトを最初から非ウォーターフォール型開発で始めるのか、途中から適用していくのかによって適用方法も異なる可能性が高い。無理なく大規模プロジェクトで非ウォーターフォール型開発を導入展開するための知見を収集する。また単にプロジェクト内での展開だけで

Copyright © 2012 IPA, All Rights Reserved

なく、組織的にアジャイル型開発を導入する事例がある場合は、そちらの展開についても合わせて調査する。

本調査ポイントにおける想定結果は次の通りである。

- 全体で一斉にアジャイル型開発を適用するのではなく、試行チームで評価した上で段階的に適用チームを拡大している。
- 変化の度合いが高い部分を起点として拡大している。
- 組織文化を変遷させるためのリーダーシップが存在する。

2.2.3. 分散拠点開発

参画人数が増えれば増えるほど、物理的に一つの拠点、一つのフロアに集ることが困難になる。そのため、複数の拠点にまたがって開発を進めるケースが多いと考えられる。開発拠点が分散すると、同じようにコミュニケーションが阻害されることになる。
昨今では、手軽に音声チャットやビデオ通話が可能になってはいるが、ホワイトボードの利用や、見える化での共有などにおいては、同一拠点に比べての困難が予想される。
本来分散開発については、主要調査対象になる重要なテーマであるが、本調査においては、大規模事例のバリエーションとして存在する事例について調査を実施した。想定結果は以下の通りである。

- 同一拠点で作業して、次の段階で分散してチームを作る
- IP 電話会議システム、チャットなどコミュニケーションツールを駆使している

2.3. 「動くソフトウェア」に関しての調査課題

「システムの分割統合や創発アーキテクチャが困難になる」という課題について、特に小規模事例と比べて、規模によって変動し一層の工夫が必要になる要素として以下の項目を調査課題とする。

2.3.1. アーキテクチャ・共通基盤の構築手順

アジャイル型開発においては、**YAGNI**(You Are not Gonna Need It)と呼び、将来を予測しすぎることでムダを生じてしまうことを諌めてきた。そのため、先行してフレームワーク、アーキテクチャを事前に設計する **BUFD**(Big Up Front Design)よりも、漸進的な開発の結果として創発されたフレームワーク、アーキテクチャを好むとされている。

Copyright © 2012 IPA, All Rights Reserved

しかし現実には、規模が拡大するほど、後になっての全体に関わるアーキテクチャや共通基盤の変更は、手戻り作業と付随するコストが増加するため現実的ではないという見解が一般的であり、YAGNI と BUFD のバランスが求められる。[1]

本調査ではアーキテクチャの構築手順を調査することで、非ウォーターフォール型開発での現状の知見を収集した。本調査ポイントにおける、想定結果は次の通りである。

- アプリケーション部分とは異なるサイクルで、共通基盤、アーキテクチャも非ウォーターフォール型開発を用いて漸進的に設計開発されている
- アーキテクチャ、共通基盤を担当するチームが存在して開発チームに支援をしている。

2.3.2. システムの分割と統合

システムの規模が拡大するにつれシステムにおいての統合対象が増えることになる。一般的には、サブシステム間を疎結合にして、事前にインターフェースを決めておいて遵守することで、統合時の問題を未然に防ごうとするが、それだけでは、ウォーターフォール型開発で発生しがちなビックバン結合(サブシステムを最後に結合して失敗する)を避けることは困難である。そのためビックバン結合に陥らぬように、早期から頻繁にサブシステム間を繋げて統合することが重要になってくる。

また、システムそのもの分割方針も疎結合なサブシステムとして分割されるはずであるが、規模が拡大すると、複雑性の観点から、単純に分割や統合を行うことが困難となり様々な工夫が求められると推測する。また後述するチーム構成とも密接に関わってくるため、その部分の調査を実施した。

本調査ポイントにおける、想定結果は次の通りである。

- サブシステム間の統合は継続的インテグレーション(CI) により頻繁に実施されている。
- システム間が、できるだけ疎結合になるように分割されている。
- システムは機能やサービスという単位で分割されている。チームはその分割とチームの分割がマッピングされ、チームはその機能単位で構成されている。

2.4. 「顧客との協調」に関しての調査課題

「顧客との協調」を妨げる要因として、「従来とギャップが大きい、顧客の負荷が高い」を取り上げた。本課題について、調査ポイントを述べる。

2.4.1. 明確な問題意識と効果

Copyright © 2012 IPA, All Rights Reserved

本件の調査対象となるプロジェクト、組織においては、中〜大規模ということもあり、非ウォーターフォール型開発を導入するにあたり、小規模と比べるとその採用には大きな意思決定が必要である。そのため、慣れた手法からの移行については、明確な問題意識と、その期待が明らかであることが推測される。

本調査ポイントでは、非ウォーターフォール型開発を(部分的にしろ)採用するに至った背景、問題、解決したい問題を、非ウォーターフォール型開発の採用を推進した立場の意見を調査・分析することで、意志決定者の問題意識及び期待と効果を明らかにし、今後の中大規模プロジェクトにおいても非ウォーターフォール型開発の必要性の後押しとなる動機や効果についての調査を実施した。

本調査ポイントについての、想定結果は次の通りであった。

- 非ウォーターフォール型開発への適用判断を経営者や顧客がしている。
- 解決したい問題意識、考慮すべき制約、期待、その効果が明確になっている。

2.4.2. 各種管理

アジャイル型開発は、顧客から見ると従来のやり方と大きく異なる点が多々ある。顧客との協調を重視するがために、密な頻度で開発チームに次々と要件を提示し、リリースや反復期間の計画づくりに参加し、開発の最中に開発者からの質問に答え、反復の度に実現されるソフトウェアのレビューや、受入テストの実施、など、これまで以上の密度でプロジェクトへの参画が求められる。この部分についての調査ポイントとしては、「組織体制とコミュニケーション」で調査することとする。

また、顧客の参画度合い意外にも、実現されたソフトウェアのフィードバックを元にした変化に応じた要件の見直しや優先順位づけに基づくスコープ管理、従来のガントチャートを用いた進捗管理が一般的には行われない。また早期から実現していくための品質の管理などが従来手法と異なっていると受け取ることが推測される。そのため、本調査においては、特に品質管理の観点に絞って調査を実施した。

本調査ポイントにおける、想定結果は次の通りである。

- 特に品質に気をつかうケースにおいては、反復以外にテストフェーズを設けて品質管理を行っている。
- 単体テストの自動化、継続的インテグレーションといった品質に関連するプラクティスを実施している。

2.5. 「変化への対応」に関しての調査課題

アジャイル宣言の「変化への対応」についての課題として「変化する部分と、安定する部分が存在する」について調査ポイントを述べる。

Copyright © 2012 IPA, All Rights Reserved

2.5.1. 部分適用

大規模システムになると、システムすべてにおいて変化が激しいという状況よりも、変化しやすい部分域と、変化しにくい部分に分かれるのではないかと考えた。変化しやすい領域として考えられる例としてはコンシューマ向けの Web サービスのフロント部分、変化しにくい領域として考えられるのはバックエンドの管理画面などである。大規模システムで、このように変化しやすい部分と、変化しにくい部分について明確に分かれる場合には、アジャイル型開発を適用しているケースと、そうでないケースが混在するのではないかと考えた。そこで、このような事例が存在する場合は「部分適用」事例と位置付けて、そこにおける工夫を調査した。

本調査ポイントについての想定結果は次の通りである。

- 部分適用が存在する場合はアジャイル型開発を「変化の必要な部分」にのみ適用している。
- アジャイル型開発、非アジャイル型開発の混在チームにおいては、互いの同期ポイントを反復のタイミングで合わせている。

2.6. 全体に関する前提条件・背景

2.2～2.5 までの項目が、本調査における大規模に起因する課題と、それに対しての調査ポイントであった。本節では大規模事例調査をより探究するために、前提条件や背景についての調査も必要と考えた。

2.6.1. 組織文化調査

大規模アジャイル型開発に関する書籍 [2],[3],[4]では、必ず組織文化の変化に言及する章が含まれおり、いずれも組織文化の変化への抵抗を克服することが、大規模アジャイル型開発の成功の一要因であると述べている。

組織文化は非常に観測しづらく表出化しにくい部分であるが、構成員の思考や行動を規定してしまう価値観を含む。故にその価値観がアジャイル型開発と噛み合わない場合には、導入に際して障害になり得る。

また異なる組織文化への移行自体がそもそも困難を伴うものである。ソフトウェア開発に限らないが、企業が異なる組織文化への移行に失敗し、ビジネス自体が立ち行かなくなった組織の事例も見受けられる [5]。

アジャイル型開発を適用する際には、アジャイル型開発が内在する文化と、それまでのプロジェクトの開発手法、あるいは組織自体の文化とのギャップの度合いによって、実施する施

Copyright © 2012 IPA, All Rights Reserved

策が変わると推測される。本調査では、調査対象の組織文化についても着目し、アジャイル型開発と、それ以前のウォーターフォール型開発との組織文化を比較することで文化的差異を明らかにし、異なる組織文化からの移行の際の工夫を収集する。

本調査ポイントの想定結果は次の通りである。

- 組織文化の移行が発生している場合は、一時的にパフォーマンスが落ちる、作業者の反対に合う、などの問題が起きている。
- 組織文化がアジャイル型開発の適用前と後で近い場合は文化的問題が発生しない。
- ウォーターフォール型に比べて管理度合いが低くなり、協調度合いが高くなっている。

2.6.2. 従事者健康度

平成 22 年度の非ウォーターフォール型開発WG活動報告書[5]の中で、以下のような文面がある。

> **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。**

つまり、「エンジニアが生き生きと働ける環境を作る」ことが本調査を含む、非ウォーターフォール型開発の一連の調査の目的の一つとなっている。そのため「生き生きと働いている」ことが何らかの形で把握できるように調査がなされているべきである。

アジャイル型開発においては、ウォーターフォール型開発と比べて、より顧客の近くで開発することになり、開発者自身が「顧客の実現したい本来の目的」を理解しようと努力することで、自分達の実現したソフトウェアを通じて顧客に対しての価値を提供することにやりがいを感じる傾向が強いのではないか、またチームとして一体感を持って仕事をすることで、より生き生きと仕事ができるのではないかと考えた。

一般的には、対象プロジェクトおよび組織の規模が大きくなればなるほど、顧客との距離が遠くなり、そこに関わる人たちの責任が分断される。それによってストレス度のばらつきが大きくなる。メンタル的に落ち込む人が減れば、プロジェクト推進にも影響を及ぼすだけでなく、個人の社会的活動にも大きな影響が出る。

しかしアジャイル型開発を実施しているのであれば、大規模であっても顧客との距離が近くなり、チームとしてそのやりがいを感じ、生き生きと働ける環境に近づくのではないかと考える。

---

[5] http://sec.ipa.go.jp/reports/20110407.html

Copyright © 2012 IPA, All Rights Reserved

本調査では、従事者の精神的健康度を調査することによって、アジャイル型開発が、精神的健康度の向上にも寄与することの事例や統計情報を収集する。

本調査の想定結果は、次のとおりである。

- ウォーターフォール型開発と比べて、非ウォーターフォール型開発に携わっているメンバーは、精神的健康度が高くなっている。(生き生きと仕事をしている)

Copyright © 2012 IPA, All Rights Reserved

3. 調査方法

3.1. 調査の流れ

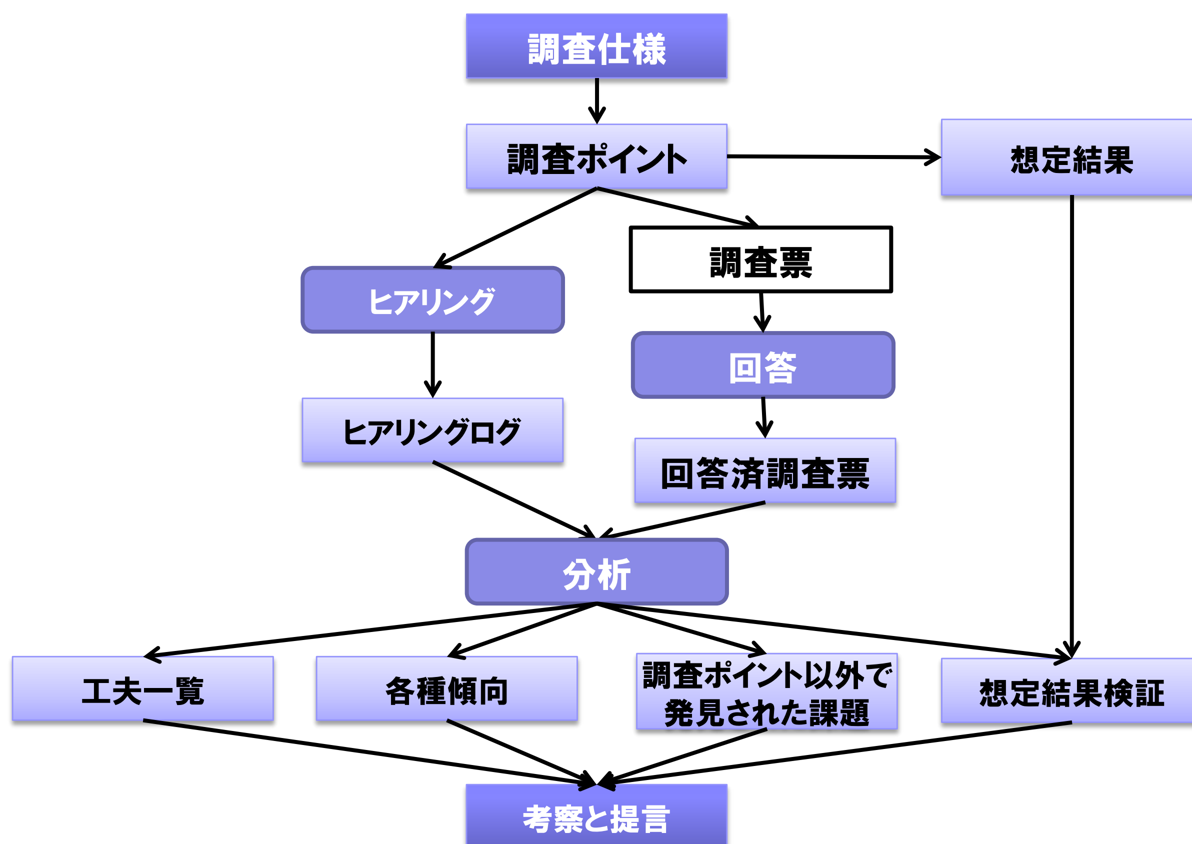

図 3-1 調査の流れ

本調査は、図 3-1 のような流れで各調査対象について調査を実施した。以下にその流れを記述する。

1. 前章の通りに調査ポイントを設定した。
2. 調査に必要な項目を洗い出した上で３種類の調査票を作成し、事前にそのうち２つの調査票を調査対象各社に送信した。
3. 事前に調査票への回答を可能な限り依頼した後に、先方を訪問し内容について個別ヒアリングを実施した。
   - その際に、３つ目の調査票について回答依頼を口頭で説明の後お願いした。
4. 調査票+ヒアリングの内容を元に分析を行った。
   - 各事例の結果を集計しての傾向分析
   - 各事例を通して発見された工夫のパターンマイニング
     (よく使われている工夫の抽出)
   - 想定と調査結果との比較
   - 調査ポイント以外で発見された課題
5. これまで結果を踏まえて考察と提言にまとめた。

Copyright © 2012 IPA, All Rights Reserved

なお、作成した調査票は表 3-1 のような目的と内容になっている。

調査票種類	質問内容等
背景調査票	非ウォーターフォール型開発の適用の背景、問題、効果について回答してもらう【主にプロジェクトマネージャ等のプロジェクト責任者向け】
実態調査票	実際のプロジェクトの基本情報、行われている各種工夫を回答してもらう【主にプロジェクトリーダ等のプロジェクト牽引者向け】
個人意識調査票	個人の意識で組織文化、精神的健康度について回答してもらう【プロジェクトメンバー等個人向け】

表 3-1 調査票の種類と目的

3.2. 調査方法

以下に具体的な調査方法について説明する。

3.2.1. 【調査方法 1】問題と解決策と、その結果の関連調査

調査方法は、去年までの調査では、解決策や工夫、プラクティスのみをヒアリングしている。しかしながら、去年度までの報告書によるようにそれぞれの組織は、それぞれ異なった文化や文脈を含有している。その結果、使用している解決策やプラクティスの羅列のみになってしまい、報告書の読者はどのようなときにそのプラクティスを採用したのか、その結果、どのような結果や影響があったのかがわからない状態になっている。つまり、本報告書の読者が解決策やプラクティスを採用するに至った文脈や経緯、結果や影響が大切である。場合によっては、新たなる問題を発生させる。

そのため、アンケートによる回答とは別にヒアリングを行った。ヒアリングでは、非ウォーターフォール型開発を適用した背景を時系列で聞きだしていきながら、そこで起きた出来事、今回の調査対象となる工夫を聞き出していった。アンケートで回答できない部分についても、個別のヒアリングを用いて具体的な工夫の聞き取りを実施した。それを既存の分野との整合性を見ながら KJ 法を用いて、整理を行った。

その結果、現れてきた各事例の工夫について、複数事例で発見されたものを、中大規模事例における貢献度の高い工夫として重要とみなしてまとめた。

3.2.2. 【調査方法 2】時系列変化を調べる

調査方法 1 でまとめた工夫について、それぞれの適用時期と順番をまとめた。ある問題に対する解決策は、新たな問題を引き起こし、次なる解決策を誘発することがある。工夫間の時系列変化を表現することで、報告書読者にとっては、プロジェクトのどのタイミングで何を始めればいいかの判断指針となることを狙う。

Copyright © 2012 IPA, All Rights Reserved

3.2.3. 【調査方法 3】アンケート＋インタビュー

本調査においては、アンケート（質問紙）を用い、特に重点的な点については、インタビューを行った。アンケートは、時間を取らせないなど非調査者の利便性が高いことと、統計的な分析が可能であること、調査数を増やせるなどのメリットがある。
さらに、その重点的な項目においては、対面によるインタビューを行い、調査票に記述されている背景や問題、歴史的な経緯を明確にした。

3.2.4. 【調査方法 4】複数ロールを調査対象

対象については、前報告書に基づき、プロダクトオーナー、スクラムマスター、チームなどの複数の重要なロールについて、それぞれ可能な限りインタビューを実施しようと試みた。
複数のロールで同じ項目を調査する理由は、ある解決策やプラクティスが、特定のロールにおいては有用であるが、別のロールにおいては有用さが薄いなどの構造を浮き彫りにするからである。

3.2.5. 【調査方法 5】組織文化調査

組織文化の調査は、被験者に「ウォーターフォール型開発のプロジェクトの組織文化」、「現在の非ウォーターフォール型開発適用後のプロジェクトの組織文化」の２つについて、それぞれアンケートを実施した。調査方法としては OCAI(Organizational Culture Assessment Instrument: 組織文化評価手法) [6],[7]を用いた。詳しい調査方法は組織文化の傾向で述べる。

3.3. 調査項目

調査方法に挙げた点を考慮し、問題、背景、結果、時間軸での変遷を聞きだすためのテンプレートを調査項目毎に質問した。テンプレートは次の通りである。基本質問項目はすべての調査項目において聞いた。追加質問項目は、基本質問のうちの「プロジェクトの目的、ビジネス価値への貢献度」が高いもののみ聞き取りを行った。ただし、先に挙げた調査ポイントに関連すると判断する項目については、評価が低くとも個別に聞き取り対象とする。

下記の項目を調査項目と位置付けた。

- 企業プロフィール
- アプリケーションシステム名または利用ソフトウェア名
- 導入背景、動機、狙い、効果
    - 適用検討までの背景

Copyright © 2012 IPA, All Rights Reserved

- 解決したかった問題
  - 導入における制約、考慮した点
  - 適用を判断した理由、その条件
  - 適用前の期待
  - 実際に得られた効果、その評価方法
  - 新たに生じた課題

- 全体像と歴史的経緯、歴史的経緯
  - ターニングポイント
  - 展開状況

- 調査対象プロジェクトの概要
  - 企業名
  - 組織名
  - アプリケーションシステム名

- 対象ドメイン
  - アプリケーションシステムタイプ
  - アプリケーションシステムの目的
  - マーケットサイズ
  - アプリケーションシステムの重要度
  - ドメインの変更に関する特徴
  - 目的実現の際に優先したビジネス価値
  - ビジネス価値の効果

- システム属性
  - 規模
  - システム環境
  - ソフトウェア／ツール／テクノロジー
  - プラットフォーム
  - 開発言語

- プロジェクト特性
  - プロジェクト期間
  - プロジェクト費用
  - プロジェクト工数
  - プロジェクト初期における要件の確定度合い
  - 1ヶ月あたりの要求変更の割合
  - 市場投入時間
  - 要求された稼働率
  - 新規性 (ビジネス)
  - 新規性 (技術)
  - プロジェクト関与者人数
  - プロジェクト関与者の内訳
  - 男女比
  - プロジェクト制約条件
  - コンプライアンス
  - 開発拠点

- プロジェクト組織特性
  - プロジェクトに関与した企業と契約関係
  - プロジェクト内組織の内訳

Copyright © 2012 IPA, All Rights Reserved

    - 組織内の評価制度
    - 組織の柔軟性

- チーム特性
    - チーム編成
    - チームあたり人数
    - チームに対しての役割分担
    - チームにおける、「平均レベル以下の、経験は少ないが、勤勉な開発者」の割合
    - チームにおける、「非ウォーターフォール型開発またはウォーターフォール型開発のプロジェクトをマネジメントできる人」の割合

- 成功度
    - 全体の成功度
    - 成功の尺度

- システムオーナと開発プロジェクトマネージャとの分担関係

- プロジェクト進行の概要
    - ライフサイクルモデル
    - リフレクション/レトロスペクティブ(反省会/ふりかえり)
    - プロセス間のコラボレーション
    - システムオーナと開発プロジェクトマネージャとの協調

- 個人特性（スキル・教育）
    - 顧客
    - 開発／保守者
    - 運用者
    - その他の役割(デザイナ、ドメインエキスパート、など)
    - 開発プロジェクトマネージャ／生産管理者

- 利用したプロジェクト管理手法
    - 統合管理
    - スコープ管理
    - スケジュール管理
    - コスト管理
    - 品質管理
    - 組織・要員管理
    - コミュニケーション管理
    - リスク管理
    - 外注管理

- 中大規模特性
    - アーキテクチャ構築プロセス
    - アーキテクチャ特徴
    - システム間インテグレーション
    - ドキュメンテーション
    - 目的・ビジョン共有
    - 組織文化
    - 部分適用
    - プロジェクト内での展開の過程
    - チーム間連係

Copyright © 2012 IPA, All Rights Reserved

    - 分散拠点
    - 異企業間開発

- 利用した開発・保守手法
    - 要求形成／追跡
    - 設計
    - 構築
    - テスティング／V&V
    - デプロイ、リリース
    - メンテナンス

- 精神的健康状態についての調査
    - ストレス度
    - 身体的健康度
    - QoL (Quality of Life)

- 組織文化についての調査

Copyright © 2012 IPA, All Rights Reserved

4. 事例別レポート

本節では、調査対象となった 11 社 12 事例の各事例について、その個別の情報を列挙する。

調査先	規模	部分適用	採用手法 [※1]	システム種別	契約関係 [※2]
A社	大規模		独自	B2Cサービス（SNS）	自社開発
B社	大規模		スクラム	B2Cサービス（ソーシャルゲーム）	自社開発
C社	大規模	○	スクラム	ゲームソフト	受託開発（契約形態:非公開）
D社	大規模	○	スクラム+独自	基幹システム	受託開発（準委任）
E社	中規模		スクラム	B2Cサービス（会員サービス）	自社開発
F社-1	中規模		スクラム+XP	B2Cサービス（医療・健康）	自社開発 + オフショア（準委任）
F社-2	中規模		スクラム+XP	B2Cサービス（エンターテイメント）	自社開発 + オフショア（準委任）
G社	中規模		XP	B2Cサービス（会員サービス）	受託開発（請負）
H社	中規模	○	XP	B2Cサービス（ECサイト）	受託開発（請負）
I社	中規模	○	XP	B2Cサービス（会員サービス）	受託開発（準委任）
J社【参考】	小・中規模組織展開		独自	B2Cサービス（会員サービス）	自社開発
K社【参考】	小規模組織展開		スクラム	B2Cサービス（SNS）	自社開発

大規模（100名以上）:4件　　中規模（30～99名）:6件　　部分適用事例:4件

※1:スクラム, XP → 両手法をベースにカスタマイズした事例も含む
　スクラム+XP → 両手法を組み合わせて実践している事例
　独自 → 特に手法を決めない、自分達で定義

※2:自社開発 → 自社組織内に開発部隊あり、一部パートナー（派遣）
　受託開発 → 自社組織内に開発部隊なし、外部ベンダに発注している

図 4-1 事例一覧

4.1. 事例情報の見方

各事例の情報は次の構成で提示している。

- 特徴
  ：事例の特徴の説明
- 事例基本情報
  ：各事例の基本的な情報
- アジャイル型開発適用の背景
  ：各事例がどのような背景で、何を解決したくて非ウォーターフォール型開発を適用したのか、その効果や新たな課題なども記載する
- 組織構成
  ：各事例のステークホルダーの構造を組織モデルで表現した。各事例の組織モデルの図を参照。

Copyright © 2012 IPA, All Rights Reserved

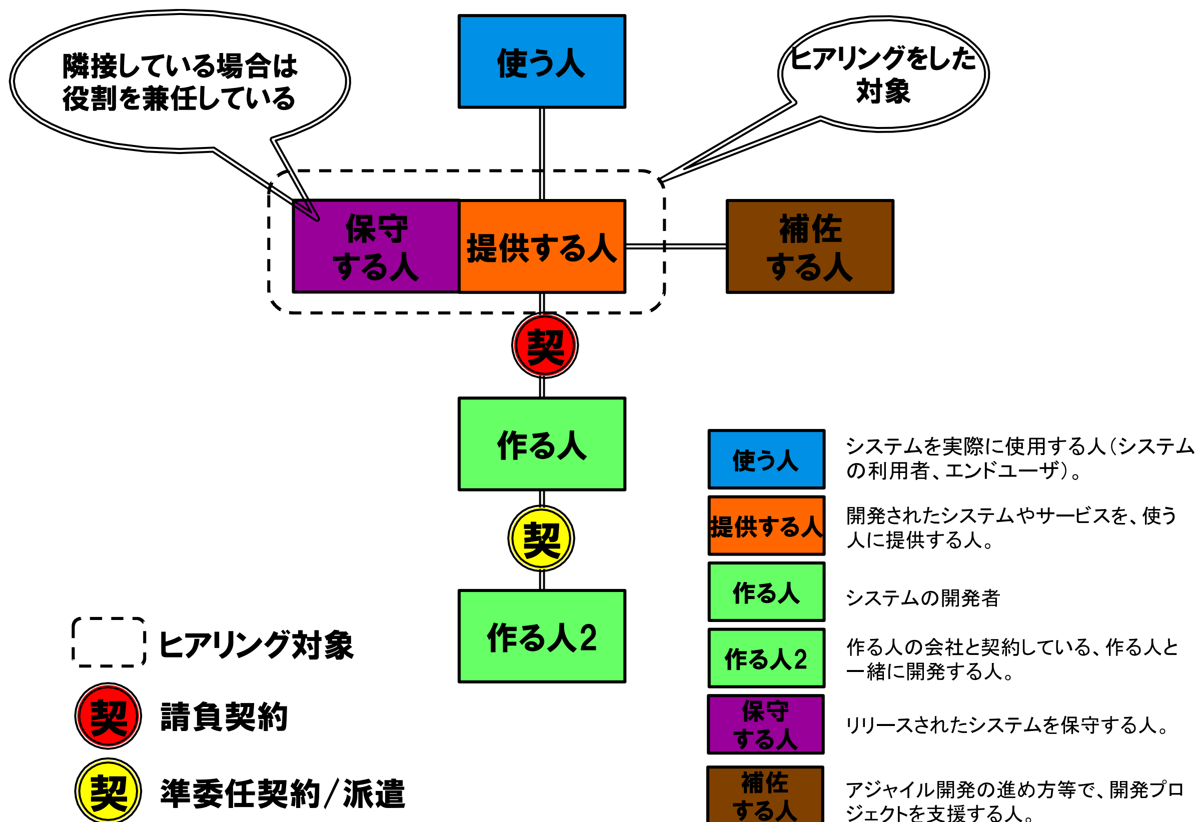

図 **4-2** 組織モデルの見方

Copyright © 2012 IPA, All Rights Reserved

4.2. 事例 A 社

4.2.1. 特徴

インターネット上の B2C サービスを提供する企業での、あるサービスの立ち上げから、数年後の現在に至るまでの変遷の記録を伺った。ビジネス的には新規要素が強く、プロトタイプを少数精鋭で実現して、徐々にシステム規模や、関与人数を増やしていった事例である。サービスの成長と共に、ターニングポイントでのパフォーマンス劣化によるシステムアーキテクチャ大幅見直し、組織体制やマネジメント手法の見直しを進めながら改善していき現在に至っている。ビジネスの特性上、ウォーターフォール型開発をすすめることはもともとなく、アジャイル型開発やスクラムといった特定の手法を意識することもなく、自然に非ウォーターフォール型開発の開発スタイルを実践し、規模拡大に適用していった点が特徴的であった。

事例基本情報

事例タイプ	大規模プロジェクト	要件の 確定度合い	コンセプトだけでほぼなし
システムタイプ	仮想空間アバターサービス	開発拠点	同一拠点 フロアは異なる
プロジェクト工数	42 人月 (サービス立ち上げ時)	重視する ビジネス価値	1:市場投入時間の短縮 2:品質の向上 3:変化への柔軟性の向上
プロジェクト期間	3 年 6 ヶ月(現在も継続中)	契約	自社内のため無し(一部は準委任もしくは派遣あり)
プロジェクト 関与者人数	プロデューサー：15 名 Flash ディベロッパー：25 名 デザイナ・イラストレーター：40 名 アプリケーションエンジニア：20 名 インフラエンジニア：3 名 計 103 人(2012/02 現在)	成功度	5:うまくいった
市場投入時間	毎週リリース	成功の尺度	顧客数

表 4-1 A 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.2.2. 非ウォーターフォール型開発適用の背景

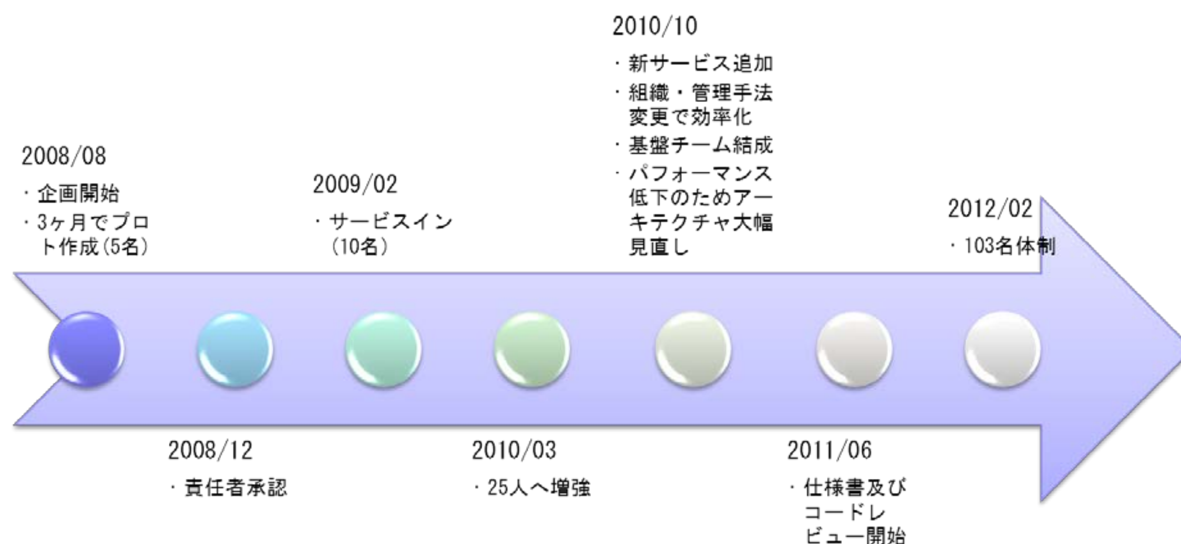

図 **4-3 A** 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景

元々、企業自体が、経営層、企画者、エンジニア、顧客、顧客見込み者の要望や社会情勢を取り込みながら開発を進める B2C サービスのため、事前に要件を固めることはできない。
そのため必然的に、ウォーターフォールによる開発は実施されずに、自然に非ウォーターフォール型開発を実践してきた。

◆ 考慮点

サービスの障害、不具合状況によっては、会社の株価にまで影響を与えてしまう可能性がある。そのため障害などの発生の際には早期に対応しなければならない。

◆ 解決したかった問題

ビジネス自体が、スピードや変化に対応していかないと立ち行かないビジネス体制である。

◆ 効果

プロジェクト開始、三ヶ月目でサービスのプロトタイプを実現し、その後もサービスの成長にともない、規模の拡大、新しい機能のリリースを実現できた。

Copyright © 2012 IPA, All Rights Reserved

◆ 生じた新たな課題

当初はマネージャが開発要員を一元管理していて、機能毎にその開発要員を割当てる体制で実施していたが、規模拡大によりマネージャの負荷が上がり要員の調整が立ち行かなくなった。そのため、機能単位にチームを構成してミッションを明確にし、マネージャの要員調整の負荷を下げた。

4.2.3. 組織構成

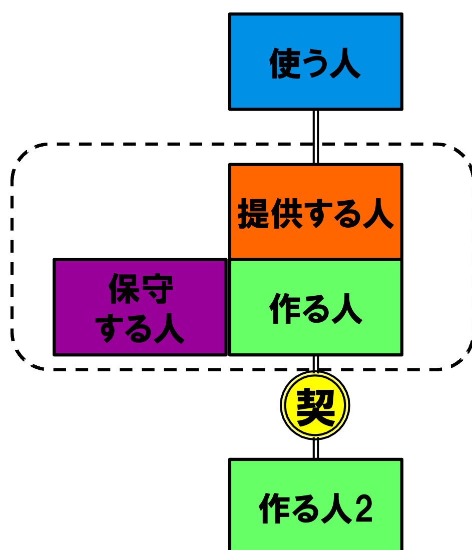

図 4-4 A 社組織モデル

- 使う人＝サービス利用者
- 提供する人、作る人、保守する人＝同一会社
- 作る人と作る人 2 の契約は準委任

Copyright © 2012 IPA, All Rights Reserved

4.3. 事例 B 社

4.3.1. 特徴

ソーシャルゲームのプラットフォームで近年に急成長している企業での、あるサービスの立ち上げから、数年後の現在に至るまでの変遷の記録を伺った。ビジネス的には新規要素が強く、少人数のチームでサービスを立ち上げ、ビジネスの拡大にあわせて、関与人数を増やしていった事例である。独自に試行錯誤しながらスクラムを導入していたが、キーマンが認定スクラムマスター研修を受講したことを契機に本格的にスクラムを導入するに至る。段階的朝会 (Scrum of Scrums) を実施している。また、プロジェクトメンバーは、スクラムで定義されているプロダクトオーナー・スクラムマスター・開発者といった役割を超えて活動している (自己組織化が進んでいる)。機能やプラットフォームといった単位ではなく、ビジネスのミッション (たとえば、アクセス数を何アクセス以上に増やすなど) でチームを分割している点が特徴的であった。

事例基本情報

事例タイプ	大規模プロジェクト	要件の 確定度合い	物によりけり
システムタイプ	ソーシャルゲーム	開発拠点	同一拠点同一フロア
プロジェクト工数	1 ヶ月あたり 45 人月投入	重視する ビジネス価値	１：変化への柔軟性の向上 ２：進行状況の可視性の向上 ３：市場投入時間の短縮
プロジェクト期間	数日～数ヶ月	契約	パートナーは準委任もしくは派遣(常駐)
プロジェクト 関与者人数	企画：20 名 デザイン：20 名 開発：45 名 テスト：15 名 計 100 名	成功度	5:うまくいった
市場投入時間	数日～数ヶ月	成功の尺度	生産性

表 **4-2 B** 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.3.2. 非ウォーターフォール型開発適用の背景

図 **4-5 B** 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ **背景**

ソーシャルゲームというサービスの性格上、ゲームが面白いか面白くないかでビジネスの成功・失敗が左右される。よって、短期間でリリースし、ユーザの反応を見ながら、サービスを成長させることが重要になる。そのため以前から非ウォーターフォール型開発であったといえるが、アジャイル型開発のプラクティスを意識的には実践されていなかった。

◆ **考慮点**

ひとつひとつの開発サイクルが短い (数日～2 週間)。企画者・デザイナー・開発者が協業してものづくりする必要があった。

◆ **解決したかった問題**

組織マネージャの意志決定がボトルネックとなる。

◆ **効果**

スクラムを導入し、チームが自己組織化したことにより、組織マネージャの意志決定のボトルネックが解消された。また、急速に変化するビジネス環境に適合するため、組織内での担当者の異動が頻繁に実施されるが、非ウォーターフォール型開発を導入後は、担当者が変わっても生産性が落ちることがなくなった。

◆ **生じた新たな課題**

Copyright © 2012 IPA, All Rights Reserved

常時問題は発生するが、そのたびに解決できる体制となっている。

4.3.3. 組織構成

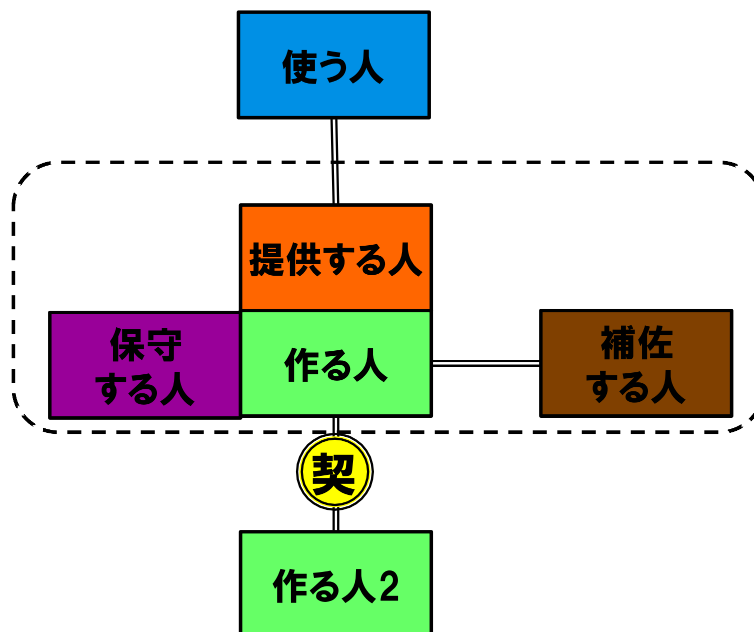

図 4-6 B 社組織モデル

- 使う人はサービス利用者
- 提供する人、作る人、補佐する人、保守する人は同一会社
- 作る人と作る人 2 の間は準委任契約。

Copyright © 2012 IPA, All Rights Reserved

4.4. 事例 C 社

4.4.1. 特徴

市販ゲームソフトの開発を受託している企業での、あるゲーム開発の立ち上げから、リリースに至るまでの変遷の記録を伺った。自社サービスに比べると、受託開発という制約の大きい状況下で、ボトムアップで徐々にスクラムを導入していった事例である。当初は少数の担当者が自己流でスクラムを実施していたが、キーマンが認定スクラムマスター研修を受講したことを契機にプロジェクト内での影響力を強めた。開発者・一部のデザイナー・企画者・サウンド担当者がスクラムを導入し、発注者や大半のデザイナーにはスクラムを実施していることは宣言していないという部分導入の事例である。発注者に対する進捗報告などは、窓口となる開発者が、内部的に実施しているスクラムの成果物を、発注者向けの資料に変換してまとめる役割を担っているのが特徴的であった。

事例基本情報

事例タイプ	大規模プロジェクト	要件の 確定度合い	開発期間とコンセプト程度
システムタイプ	ゲームソフト	開発拠点	同一拠点
プロジェクト工数	未回答	重視する ビジネス価値	未回答
プロジェクト期間	非公開	契約	非公開
プロジェクト 関与者人数	開発:15 名 CG(UI デザイン):6 名 CG(UI 以外):60～70 名 企画 8 名 サウンド:1 名 テスター (外部委託) 計 100 人(Max 時)	成功度	3:うまくいった部分もあるが、うまくいっていない面もある。
市場投入時間	非公開	成功の尺度	品質、バグ件数、開発期間

表 4-3 C 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.4.2. 非ウォーターフォール型開発適用の背景

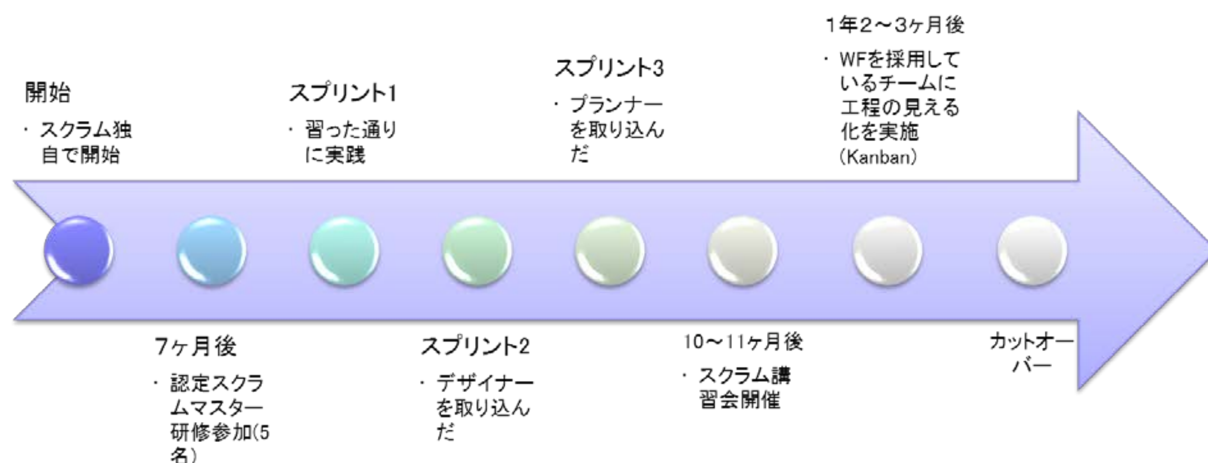

図 4-7 C 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景

プロジェクト管理のためのコストが高く、管理コストを圧縮するため、スクラムを導入した。

◆ 考慮点

発注者には開発スタイルがスクラムであることを伝えていないため、ガントチャートのスケジュール表を提出する必要があった。また、アジャイル型開発の経験者がほとんどいなかった。

◆ 解決したかった問題

管理コストを減らして、その分、プロダクトのクオリティ (ゲームとしての面白さ) をアップさせたい。

◆ 効果

問題意識が高くなったスタッフが多く見られた。チーム全体のモチベーションが維持しやすかった。管理工数は削減され、想定以上に生産性が高くなった。

◆ 生じた新たな課題

見える化が進みすぎて開発の進捗について隠し立てができなくなり、ストレスを感じるスタッフが出てきた。思ったよりも品質は向上しなかった。

Copyright © 2012 IPA, All Rights Reserved

4.4.3. 組織構成

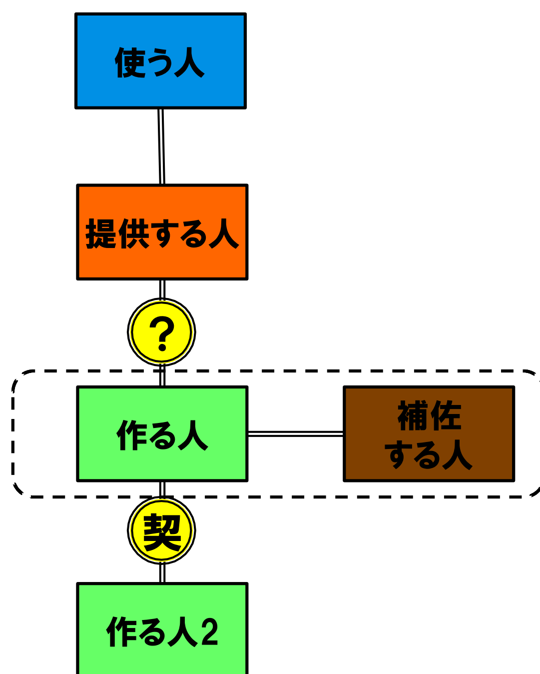

図 **4-8 C** 社組織モデル

- 使う人＝ゲーム購買者
- 提供する人＝ゲームメーカー
- 提供する人と作る人の間は契約非公開

作る人と作る人 2 の間は準委任契約

4.5. 事例 D 社

4.5.1. 特徴

大規模基幹システムの再構築の事例。詳細は非公開とする。

事例基本情報

事例タイプ	大規模プロジェクト	要件の確定度合い	未回答
システムタイプ	基幹システム	開発拠点	同一拠点
プロジェクト工数	未回答	重視するビジネス価値	未回答
プロジェクト期間	非公開	契約	準委任(常駐)
プロジェクト関与者人数	100 人以上	成功度	未回答
市場投入時間	非公開	成功の尺度	未回答

表 **4-4 D** 社事例基本情報

4.5.2. 非ウォーターフォール型開発適用の背景

本事例の非ウォーターフォール型開発適用の背景は D 社の要望により非公開とする。

4.5.3. 組織構成

本事例の組織構成は D 社の要望により非公開とする。

Copyright © 2012 IPA, All Rights Reserved

4.6. 事例 E 社

4.6.1. 特徴

インターネット上の B2C サービスを提供する企業での、あるサービスの立ち上げから、数年後の現在に至るまでの変遷の記録を伺った。トップ(経営者)からの開発コスト低減の指示の元、開発部門の事業責任者が強力に開発全体を取り仕切って、トップダウンでスクラムを導入した事例である。この事業責任者が 40 人いるチームメンバー全員の進捗を個別にチェックするなど、非常に強い影響力を発揮している点が特徴的である。また、一般的にスクラムで言われている職能横断型のチーム構成ではなく、設計・開発・テストのような職能別のチーム構成になっている点も特徴的である。これは、要員スキルにあわせた工夫といえる。

事例基本情報

事例タイプ	中規模プロジェクト	要件の 確定度合い	未回答
システムタイプ	マッチングサービス	開発拠点	国内複数拠点(東京・東北)
プロジェクト工数	未回答	重視する ビジネス価値	未回答
プロジェクト期間	2 年	契約	パートナーは準委任もしくは派遣(常駐)
プロジェクト 関与者人数	設計(基本/詳細) 開発 テストチーム プロデューサー 計 40 人(Max 時)	成功度	未回答
市場投入時間	未回答	成功の尺度	未回答

表 **4-5 E** 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.6.2. 非ウォーターフォール型開発適用の背景

図 **4-9 E** 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ **背景**
増改築を続けてきたオフコンをリプレースしたかった。ビジネスロジックなども含めアプリケーションシステムの全体的な内容を理解できない状況であった。そのため、だれも提案依頼書(RFP)を書けない状況になっていた。

◆ **考慮点**
ビジネスやアプリケーションシステムの全体がわかる人が誰もおらず、ビジネスロジックを明確に書ける人が誰もいなかった。
動いているアプリケーションシステムをベースに調査しながら、出来上がるイメージを明確にして前に進めていった。

◆ **解決したかった問題**
稼働している既存システムはあるが、仕様やビジネスロジックが不明であったため、明らかにしたかった。

◆ **効果**
失敗と経験のサイクルが早く、開発者の成長を感じられた
エンジニアが「楽しかった」と評価している
開発のオーバーヘッドが通常プロジェクトより低かった

Copyright © 2012 IPA, All Rights Reserved

◆ 生じた新たな課題

(1) 課題を報告せずに、非公式に解決しようとする人がでてきた。(裏帳簿)

(2) 受動的な人など性格やスタイルによっては、非ウォーターフォール型開発は合わず、プロジェクトから離れてもらった。だいたい 2-3 割存在した。

(3) ビジネス側と開発側の連係。お互いのミッションや役割を尊重して行動することができていない。

(4) キーマンに依存してしまい、スピードがあがりきらない

(5) 組織の要員計画により、コアメンバーが変わってしまった

(6) ビジネス側の教育が必要と感じた。

(7) 縦割りの組織体系だとうまくできない。

(8) 開発のスピードは上がっても、監査やセキュリティなどがボトルネックになっている。

4.6.3. 組織構成

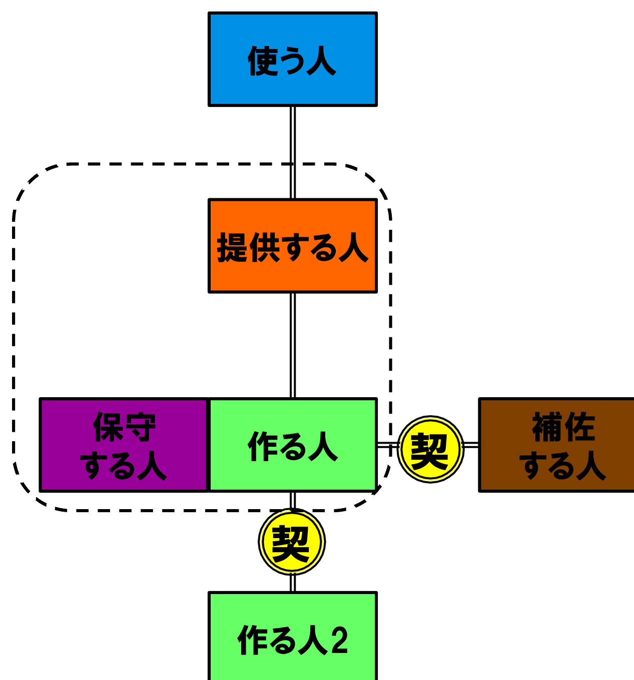

図 4-10 E 社組織モデル

- 使う人＝サービス利用者
- 提供する人、保守する人は同一会社。
- 作る人と作る人 2 との契約は準委任契約
- 補佐する人(アジャイルコーチ)は準委任契約

Copyright © 2012 IPA, All Rights Reserved

4.7. 事例 F-1 社

4.7.1. 特徴

インターネット上の B2C サービスを提供する企業での、あるサービスの立ち上げから、数年後の現在に至るまでの変遷の記録を伺った。数年前まではこの企業内では、開発部門と企画部門が分離されており、それによる開発の非効率化が問題化していた。そこで、トップ(開発部門担当の執行役員) がトップダウンでスクラムの導入を決定した。まず、手始めに社内改革によって、開発部門と企画部門を統合したのが特徴的である。その後、2 つのプロジェクトで成功事例をつくり、全社に展開している。この事例はその 2 つの成功事例の 1 つである。中国へのオフショアを積極的に活用している点も特徴的である。

事例基本情報

事例タイプ	中規模プロジェクト	要件の 確定度合い	未回答
システムタイプ	エンターテイメント サービス	開発拠点	東京・上海
プロジェクト工数	未回答	重視する ビジネス価値	マーケティングプロセス、早期の市場投入
プロジェクト期間	2 年(現在も継続中)	契約	国内は一部が準委任もしくは派遣 上海はパートナー会社
プロジェクト 関与者人数	計 40 人(2012/02 現在)	成功度	3:うまくいった部分もあるが、うまくいっていない面もある。
市場投入時間	1～3 ヶ月	成功の尺度	未回答

表 **4-6 F-1** 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.7.2. 非ウォーターフォール型開発適用の背景

図 4-11 F-1 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景

2009 年当時は組織の課題として、社内分業体制で企画部門と開発部門が完全に分離されており、企画=顧客、開発=下請けという構造が社内にできていた。ウォーターフォール型開発にも関わらず、計画がいつまでも策定できず、企画内容や要件、仕様の変更が多かった。人員を追加していたにも関わらず、その変化に対応できずプロジェクト推進が滞っていた。大きな案件が増加する中、二件の大型プロジェクトが失敗するという事態になってしまった。その事態を受け、開発部門担当の執行役員がウォーターフォール型開発は無理だと判断し、スクラムの導入を決断した。

◆ 考慮点

開発部門と企画部門を統合するといったドラスティックな組織改革をいち早く行った。開発部門と企画部門のフロアを一緒にするなど、物理的な距離も縮めた。
いきなり企画部門の人にプロダクトオーナーをやってもらうのはハードルがあったため、開発部門と企画部門のどちらにも属さない第 3 の部門をつくり、開発部門からプロダクトオーナーを出した。その後、企画部門の担当者がプロダクトオーナーを担えるようになったため、現在、その部門は役割を終えて、解散している。

Copyright © 2012 IPA, All Rights Reserved

◆ **解決したかった問題**

企画=顧客、開発=下請けという社内分業体制があり、まるで別会社のように開発部門が企画部門に見積書を出して、企画部門が開発部門に発注するという手続きを踏んでいた。それによって、無駄な事務作業によるオーバーヘッドの増加や、開発部門がリスク分を見積りに乗せたことによるコストの増加が発生するようになった。その影響で、大型の案件が 2 つ続けて失敗した。

途中で企画が変わっても追従できず、満足のいくサービスが出せない。

◆ **効果**

開発部門の社員満足度が非常に高くなった。

人が育った。

コミュニケーションが活発になった。

技術力も向上した。

◆ **生じた新たな課題**

企画部門の担当者には戸惑いが見られることがある。

急速に展開したため、実態としては、まだウォーターフォールから抜け切れていないことがある。

スクラムマスターによってやりかたが違って混乱を招くことがある (一貫した説明ができない)。

4.7.3. 組織構成

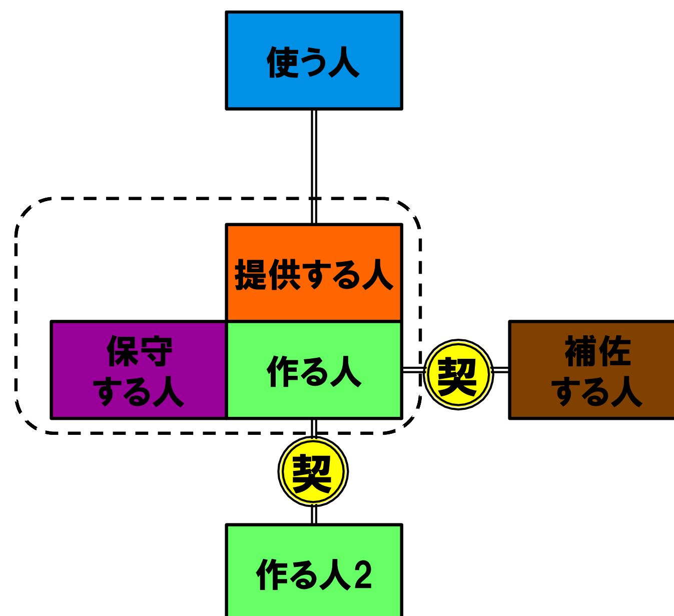

図 4-12 F-1 社組織モデル

- 使う人＝一般利用者
- 提供する人、保守する人は同一会社。
- 作る人以降の契約は準委任契約
- 補佐する人（アジャイルコーチ）は準委任契約

Copyright © 2012 IPA, All Rights Reserved

4.8. 事例 F-2 社

4.8.1. 特徴

F-1 社の事例が成功したことにより全社展開が決まり、それに伴いスクラムが導入された事例。F-1 社の事例の後を追う形で実施されているのが特徴的である。

事例基本情報

事例タイプ	中規模プロジェクト	要件の確定度合い	未回答
システムタイプ	エンターテイメントサービス	開発拠点	東京・上海
プロジェクト工数	未回答	重視するビジネス価値	マーケティングプロセス、早期の市場投入
プロジェクト期間	2 年(現在も継続中)	契約	国内開発者の 2/3 が準委任もしくは派遣 上海はパートナー会社
プロジェクト関与者人数	開発：40 名 企画:10 名 計 50 人(2012/02 現在)	成功度	3:うまくいった部分もあるが、うまくいっていない面もある。
市場投入時間	1～3 ヶ月	成功の尺度	未回答

表 4-7 F-2 社組織モデル

4.8.2. 非ウォーターフォール型開発適用の背景

事例 F-1 と同じ。

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景

F-1 社の事例と同様。

◆ 考慮点

F-1 社の事例と同様。

◆ 解決したかった問題

F-1 社の事例と同様。

Copyright © 2012 IPA, All Rights Reserved

◆ 効果

F-1 社の事例と同様。

◆ 生じた新たな課題

F-1 社の事例と同様。

4.8.3. 組織構成

F-1 社の事例と同様。

Copyright © 2012 IPA, All Rights Reserved

4.9. 事例 G 社

4.9.1. 特徴

10 年ほど前から、非ウォーターフォール型開発に取り組んでいた組織における、最新の中規模事例である。システムは実証実験的な要素が高い B2C サービスである。サービス部分とそれらサービスが利用する PaaS 基盤を並行して開発している。顧客とは請負契約を結んでいるが、開発者の日々の作業状況、開発進捗、ソースコード、不具合など開発に関する全ての情報を顧客に公開して共有していくことで信頼関係を作り、スコープ調整などにも対応できている。徹底的に開発メンバーの多能工化を進めて効果を挙げている一方、新たな課題にも直面している。

事例基本情報

事例タイプ	中規模プロジェクト	要件の確定度合い	業務プロセス未定義
システムタイプ	Android/PC 向け B2C サービス	開発拠点	東京、横浜、静岡
プロジェクト工数	40 人月(開始時は 2 名)	重視するビジネス価値	時間の短縮、有用性の検証、現場プロセスの進化を IT で加速すること
プロジェクト期間	１年半(現在も継続中)	契約	顧客との間は請負契約（一部システム要件定義契約として成果物なし）、パートナーとの間は準委任契約。
プロジェクト関与者人数	企画:6 名 開発:40 名 運用:4 名 計 50 名	成功度	3:うまくいった部分もあるが、うまくいっていない面もある。
市場投入時間	１ヶ月	成功の尺度	スコープ

表 4-8 G 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.9.2. 非ウォーターフォール型開発適用の背景

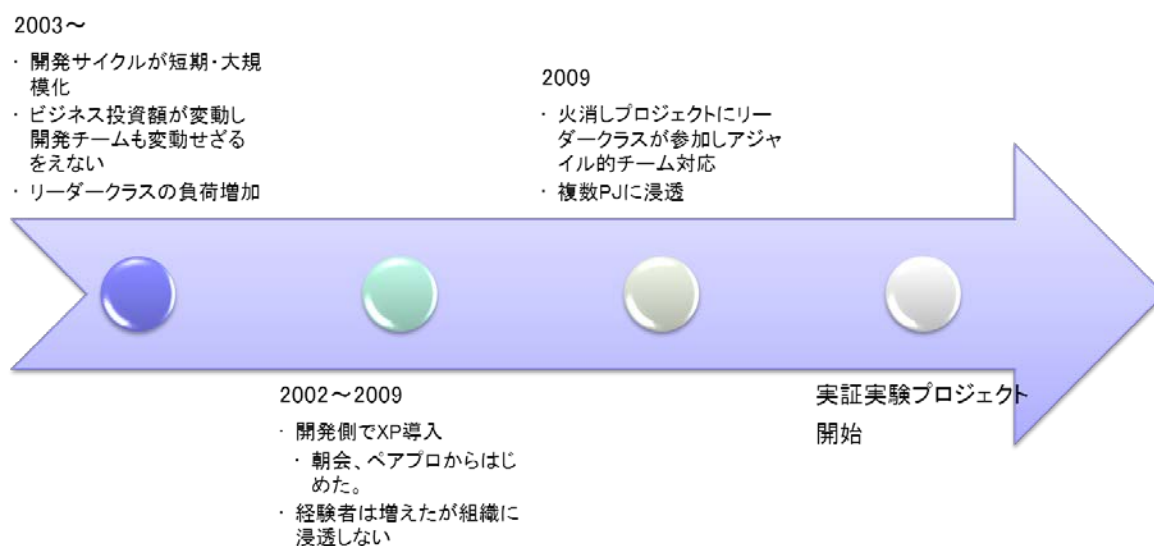

図 **4-13 G** 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景

1999 年から請け負っていた某社システムの開発サイクルが 2003 年以降、短期・大規模化するにつれ（年２回リリース→年４回へ）、受託するリスクが増大した。また、ビジネス状況で顧客の投資額が変動するため、開発チームの規模もそれに合わせて構成する必要があった。人材を流動的に配置するよう試みたが、メンバーのスキル・モチベーションの低下が進みリーダークラスの負担が増加。同時に深刻な品質低下を招く危険が出た。

◆ 考慮点

発注元でも開発プロセス (主にドキュメントの種類・形式) が規定されており、形上はこのプロセスに則っていた。

◆ 解決したかった問題

- 開発の速い段階でリスクをつぶしたい
  - 期限までに確実に動作させたい。問題点を早期に見つけたい。
- 自律的なチーム作りにより管理する負荷を減らしたい
  - プロセスの浸透・ノウハウ共有、見える化による自ら動けるしくみの実現
- プロセス順守。品質低下の防止。
  - ルール無視による作業もれ防止、ツール徹底利用によるレベルダウン防止。

Copyright © 2012 IPA, All Rights Reserved

◆ 効果

- リスクを早期に発見でき、対応を計画する余裕ができた。品質に関しても同様、開発の後半で超繁忙状態になってしまうことを防ぐことができた。
- メンバーの増減に柔軟に対応することが可能で、顧客の投資額に応じて体制を縮小・拡大することや、新たなチームを短期間で立ち上げることなどが可能となった。
- スピードを求める顧客の満足を得られやすい。

◆ **生じた新たな課題**

メンバーの選定に慎重になる必要がある。開発のスピード感についてこられない技術者が出てくる。目標を明確にしないと、モチベーションが低下して行く場合がある。責任感が無く、楽な方向に走るメンバーが出てくる。

4.9.3. 組織構造

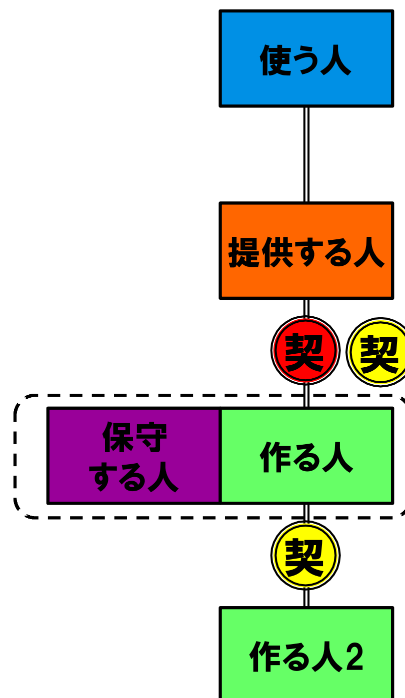

図 4-14 G 社組織モデル

- ➢ 使う人＝サービス利用者
- ➢ 提供する人＝作る人/保守する人の親会社
- ➢ 提供する人と作る人との契約は二通りある。
  - ➢ 請負契約(スコープ調整は可能、成果物責任あり)
  - ➢ 要件定義契約(成果物責任なし)
- ➢ 保守する人と作る人は同一
- ➢ 作る人と作る人２は準委任契約

Copyright © 2012 IPA, All Rights Reserved

4.10. 事例 H 社

4.10.1. 特徴

元々、出版事業がメインであったが、近年、インターネット上の B2C サービスへの取り組みを加速させている企業での、あるサービスの立ち上げから、1 年後の現在に至るまでの変遷の記録を伺った。サービス的に新規性が高い機能にアジャイル型開発を適用し、その他の機能はウォーターフォールで開発した部分適用の事例。この企業は自社に開発部隊を持たず、開発はすべて外部の企業に委託している。今回の事例では開発を担当する企業が発注元の企画を担当する企業にアジャイル型開発の導入を提案し、実施した点が特徴的であった。

事例基本情報

事例タイプ	中規模プロジェクト	要件の 確定度合い	未回答
システムタイプ	EC サイト	開発拠点	渋谷、上野、浜松、滋賀、新宿
プロジェクト工数	未回答	重視する ビジネス価値	利用者の有用性
プロジェクト期間	1 年	契約	請負
プロジェクト関与者人数	企画：10 名、開発:40 名、テスト 10 名弱、デザイナー、インフラ 計 50 人	成功度	3.75
市場投入時間	1～3 ヶ月	成功の尺度	ユーザの利用者数

表 4-9 H 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.10.2. 非ウォーターフォール型開発適用の背景

図 **4-15 H** 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ **背景**

主要な機能のうちの 1 機能を担当する開発会社(2 次請け)がアジャイル型開発を導入することを宣言した。アジャイル型開発を導入することを宣言した理由は、この開発会社がアジャイル型開発の実績を持ち合わせており、得意とする開発手法であったためである。発注元の企業も、対象となる機能が基幹に関わらないことと、サービス的に新規性が高いことから、アジャイル型開発の導入を了承した。

◆ **考慮点**

発注元でも開発プロセス (主にドキュメントの種類・形式) が規定されており、形上はこのプロセスに則っていた。

◆ **解決したかった問題**

Web システムで「もやもや」しているものなので実際に操作してみないとわからないため、実物を操作しながら開発を進めたかった。

◆ **効果**

企画者への負担が少なく、感性の高い企画者であれば、設計ができるのではないか。

◆ **生じた新たな課題**

Copyright © 2012 IPA, All Rights Reserved

部分適用であったため、段階的に統合することができず、最後にすべてのテストを実施した。

4.10.3. 組織構成

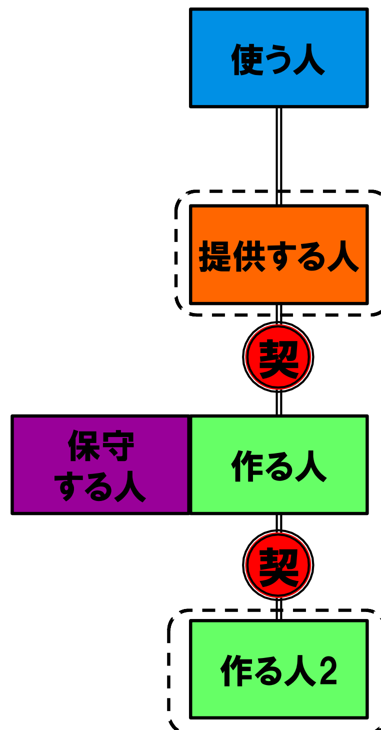

図 4-16 H 社組織モデル

- 使う人＝サービス利用者
- 契約はすべて請負
- 作る人 2 から作る人、提供する人へアジャイル型開発を提案し許可

4.11. 事例 I 社

4.11.1. 特徴

I 社の顧客であるサービスプロバイダーが、それまでの自社提供サービスの開発を特定のベンダーに委任していたため、自社の思い通りにならなかった。そのため I 社にサービスリプレース開発を依頼した。単純なサービス移行ではなく、新しい要件も加えながら開発したいとの要望を聞き、I 社から顧客にアジャイル型開発を提案して開始した。
リプレースといいながらも、元のサービスはブラックボックスのままで、暗中模索の中、要件を聞き出し顧客と一体となりながら開発を進めていった。
要件が固められない部分のみアジャイル型開発を行い、要件が明らかな部分についてはウォーターフォール型開発を実施した。国内 3 拠点での分散拠点事例でもある。
プロジェクト中盤に大幅なスコープの見直しが発生したが、アジャイル型開発のおかげで、サービスのコアに注力する計画に変更しリリースに漕ぎ着けた。顧客プロキシチームを設けることで顧客側のボトルネックを解消した。

事例基本情報

事例タイプ	中規模プロジェクト	要件の 確定度合い	要件が先に進むに連れてどんどん明確になっていく かつ、変わることもある
システムタイプ	B2C サービス	開発拠点	東京、地方を含めた 3 拠点
プロジェクト工数	500 人月	重視する ビジネス価値	マーケティングプロセス、早期の市場投入
プロジェクト期間	2 年	契約	四半期毎に契約更新 顧客とは準委任契約(時間精算無し) 開発パートナーとは準委任契約(時間精算あり)
プロジェクト 関与者人数	開発者 8 名×4 チーム インフラチーム 4 名 品質管理チーム 6 名 移行チーム 10 名 運用準備チーム 3 名 管理チーム 4 名 計:59 名	成功度	3:うまくいった部分もあるが、うまくいっていない面もある
市場投入時間	1 年半	成功の尺度	無事に市場に投入し、現在もなお保守開発を行えていること。 品質として、稼働直後若干混乱が生じたこと。 一番不安視していた移行について大きな問題がなく行えた

Copyright © 2012 IPA, All Rights Reserved

			こと。 会社の開発ベースとしてアジャイルプロセス／マインドを根付かせられたこと。

表 4-10 I 社事例基本情報

4.11.2. 非ウォーターフォール型開発適用の背景

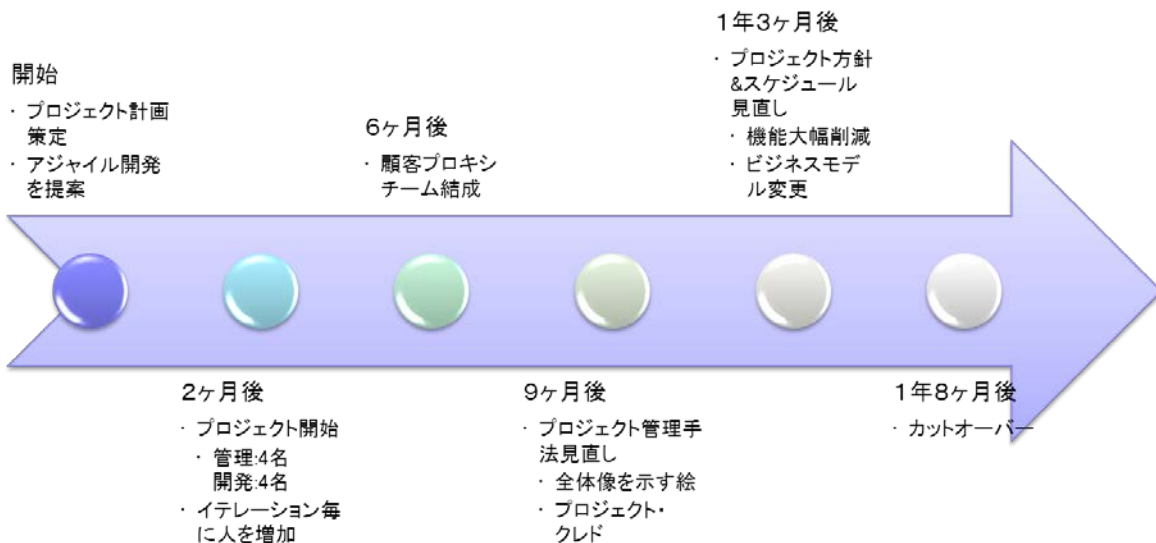

図 4-17 I 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景
プロジェクト当初から、アジャイル型開発を実施する提案をして、顧客に受け入れられた。

◆ 考慮点
既存のサービスをそのままリプレースするわけではなく、新しいコンセプトで顧客と対話しながら新しいサービスを構築していく。

◆ 解決したかった問題
すべての要件がプロジェクト開始段階で出てくることがない。また顧客と対話することで、作っていくものが変わって行くことが予想された。
そのため、以下を期待した。

Copyright © 2012 IPA, All Rights Reserved

- 優先順位づけられた開発をすることにより、顧客が良いと判断した時点でリリースができること。
- 決められた予算にもっとも近い形で開発ができるように、優先順位でもって不要なコスト、要らない機能の選択を行うこと。

◆ 効果

✧ 変化への対応。

✧ 要件、機能削減、予算縮小への対応。

◆ 生じた新たな課題

人を増やして行くことの難しさ。大規模が故に、チームをまたがったコミュニケーションの重要さ。顧客側のチーム要員が少ないため、それの補助が必要な面など。

4.11.3. 組織構成

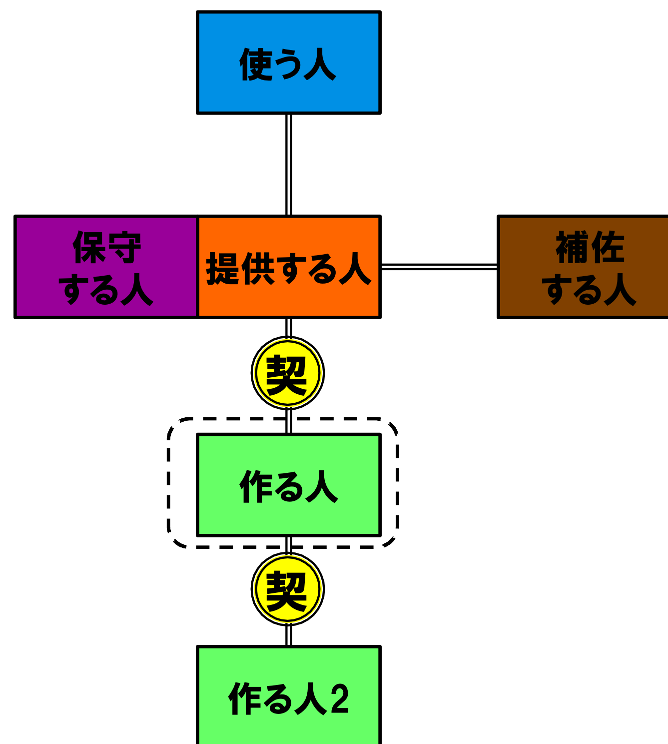

図 4-18 I 社組織モデル

- 使う人＝サービス利用者
- 提供する人と保守する人は同一
- 補佐する人(顧客プロキシ)は、作る人から提供する人へ出向していた。
- 契約はすべて準委任
- 作る人から提供する人へアジャイル型開発提案を行った。

Copyright © 2012 IPA, All Rights Reserved

4.12. 事例 J 社 (参考)

4.12.1. 特徴

元々、出版事業がメインであったが、近年、インターネット上の B2C サービスを数多く立ち上げている企業での、標準プロセスをつくって全社に展開した組織展開の事例である。この標準プロセスは、開発人数が 30〜40 人を超えると、ウォーターフォールの要素を取り入れたカスタマイズ (仕様変更のコントロールのやり方を変える、テスト・ドキュメントを厚くする) を行うのが特徴的であった。標準プロセスを定める組織横断的な部門があり、個別のプロジェクトのスクラムマスターの役割も担う。

事例基本情報

事例タイプ	組織展開 (小〜中規模プロジェクト)	要件の 確定度合い	全社標準プロセスに則るのに必要なレベルでは揃っていた
システムタイプ	顧客問題解決サービス	開発拠点	同一拠点 企画は別フロア
プロジェクト工数	36 人月 (サービス立ち上げ時)	重視する ビジネス価値	ユーザ数、短納期、低コスト
プロジェクト期間	5 ヶ月(現在も継続中)	契約	開発メンバーは準委任もしくは派遣(常駐)
プロジェクト 関与者人数	企画：5 名 スクラムマスター的役割：4 名 開発：11 名 デザイナー QA 計 20 人	成功度	3:うまくいった部分もあるが、うまくいっていない面もある
市場投入時間	3 ヶ月	成功の尺度	QCD、バグ発生率(1000FP あたりのバグ数) ただし、ウォーターフォール型開発よりもバグ発生率は増加する前提としている

表 4-11 J 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.12.2. 非ウォーターフォール型開発適用の背景

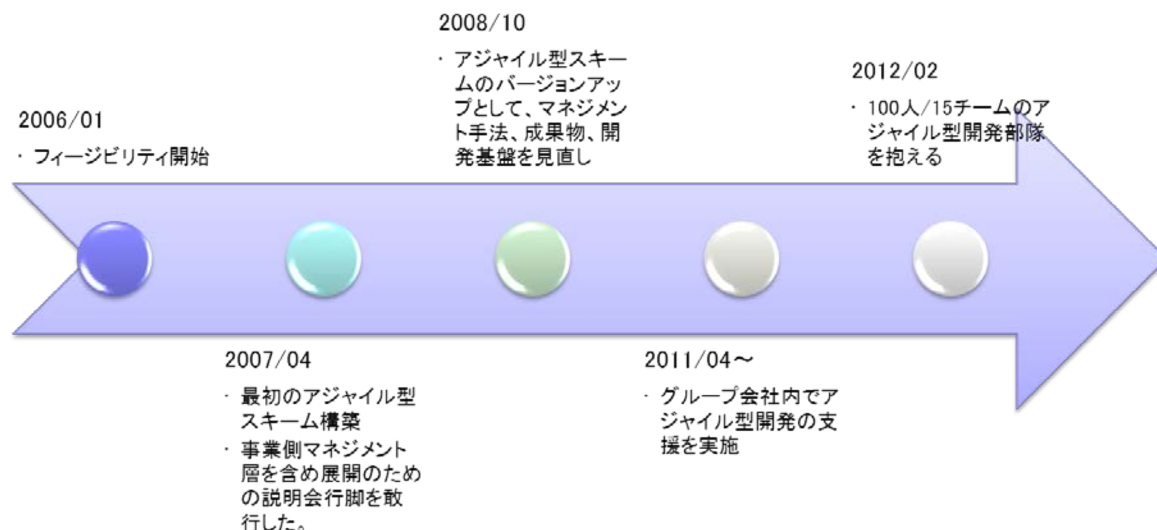

図 **4-19 J** 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ **背景**
元々、出版事業をメインとした紙のビジネスを行ってきたが、社会情勢的にネットが盛り上がるに伴い、Web の世界は速度感が違うことを実感。競合が力をつけていることにより、競争が激化。Web は出す前に考えても仕方ないことに気づきはじめた。完璧を期すると時間がかかってしまうことから、アジャイル型開発をベースとした標準プロセスの策定に乗り出した。

◆ **考慮点**
規模が大きくなるとウォーターフォールの要素を取り入れたカスタマイズを行うというように、サービスに適した進め方を使いわけている。すべてのプロジェクトに標準プロセスを適用するわけではない。

◆ **解決したかった問題**
ビジネス側と開発側が一体となり、スモールスタートできるような環境を作りたかった。

◆ **効果**
要件を定義した人が、そのままコーディングするため、情報を伝達する工数がカットできる。
それによって、工期を圧縮することができた。
ドキュメントの削減が、生産性に寄与した。

Copyright © 2012 IPA, All Rights Reserved

◆ 生じた新たな課題
標準プロセスを、40 人を超えるプロジェクトに適用しようとしたときにうまくいかなかった (ドキュメントを介しない口頭での情報伝達に限界がある)。

4.12.3. 組織モデル

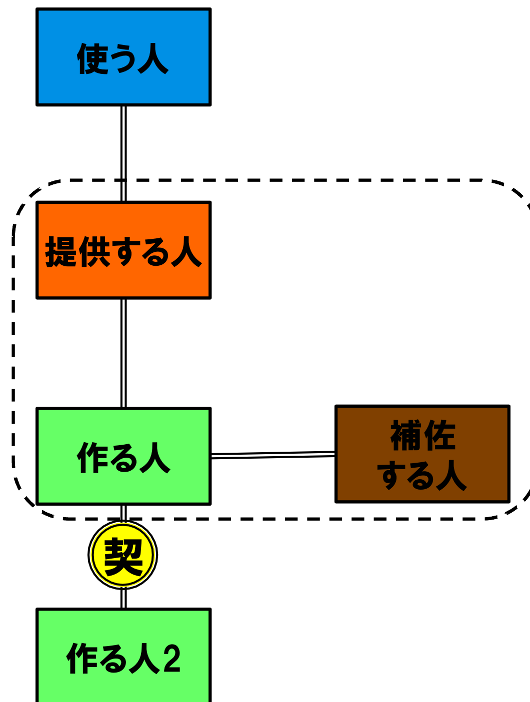

図 4-20 J 社組織モデル

- 使う人＝サービス利用者
- 提供する人と補佐する人は同一会社
- 保守する人は不明
- 作る人と作る人 2 の契約は準委任か派遣

4.13. 事例 K 社（参考）

4.13.1. 特徴

インターネット上の B2C サービスを提供する企業で、アジャイル型開発を推進するための横断組織をつくり、導入を進めている組織展開の事例である。これまでは Ad-hoc なやり方 (できたものから都度リリースしていくようなやり方) をしていたが、ここ 1,2 年でスクラムに移行している。この横断組織のメンバーが開発のコーチ役を果たす。企業の文化・風土がアジャイル型開発的なやり方と非常に親和性が高いというのが特徴的であった。

事例基本情報

事例タイプ	組織展開 ※プロジェクト情報は参考	要件の 確定度合い	必要なコア機能のみ確定していた
システムタイプ	顧客課題解決ソフトウェア （自社サービス）	開発拠点	同一拠点、同一フロア
プロジェクト工数	15 人月 (サービス立ち上げ時)	重視する ビジネス価値	事業の立ち上げ 新規市場開拓
プロジェクト期間	3 ヶ月(現在も継続中)	契約	請負(サブシステムの一部)
プロジェクト関与者人数	責任者:1 プロデュース・営業・デザイン:1 デザイン:1 開発:5 アジャイルコーチ:1 計 9 人	成功度	5:うまくいった
市場投入時間	3 ヶ月	成功の尺度	期日、コア機能の充足度

表 4-12 K 社事例基本情報

Copyright © 2012 IPA, All Rights Reserved

4.13.2. 非ウォーターフォール型開発適用の背景

図 4-21 K 社タイムライン

非ウォーターフォール型開発を適用した本事例における背景、考慮点、解決したかった問題、効果、生じた新たな課題は次のようなものであった。

◆ 背景

B2C の市場を相手にする仕事で、自然発生的なソフトウェア開発を行い、サービスを提供している。要求をただちにサービスに向けて実装している。四半期毎の結果が強く求められる。

社内のクレド(価値体系)があり、この価値体系を元にサービスを駆動している。そのため組織的に文化的なギャップはない。社内は深く階層化された組織というよりは、浅くネットワーク構造で繋がる組織である。

認定スクラムマスター研修を受講した担当者が、スクラム導入の提案を実現し、組織内で横断的組織を立ち上げた。

なお、このドメインではウォーターフォール型開発は合わないと感じている。年間計画に合わせて作ったらいい仕事はできない。

◆ 考慮点

アジャイル型開発導入のための横断組織は、全員通常業務との掛持ちでやっている。トップダウンで動く文化ではないので、横断組織のメンバーが能動的に動く必要がある。

Copyright © 2012 IPA, All Rights Reserved

◆ 解決したかった問題

スクラムを導入すると、安心・安全に着実にできるようになること。
スクラムのオーバーヘッドは大きいが、着実にできること。奇跡はないが払うべきコストとして必要と、導入を検討しているプロジェクトに説明している。

◆ 効果

横断組織は、社内およびグループ会社の過半数のプロジェクトからは相談を受けるようになった。新規事業の立ち上げを成功させた。アジャイル型開発の技術的なプラクティスが定着した。

◆ 生じた新たな課題

これまで「できたらすぐにリリースする」というやり方で開発してきた人々からは「思ったよりスピードがあがらない」という声がでている。推進者は「安心確実にサービスを提供できる手法」として社内にマーケティングしている。

4.13.3. 組織モデル

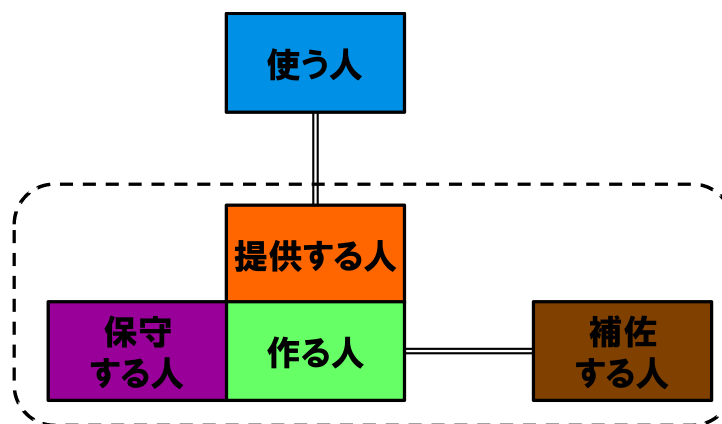

図 4-22 K 社の組織モデル

- 使う人＝サービス利用者
- 提供する人、保守する人、作る人、補佐する人は同一グループ会社

Copyright © 2012 IPA, All Rights Reserved

5. テーマ別分析結果

本章では、事例の調査結果を、調査ポイント毎に想定結果と比較し、複数の事例で観察された特徴的な工夫をまとめた。

5.1. アジャイル型開発の選択

ソフトウェア開発の進め方には、様々な種類がある。アジャイル型開発は、対話を重視するなどの理由から小規模開発に向いているとされていた。中大規模でアジャイル型開発を選択する場合には、このような事例が多かった。

1. 新しいサービスを立ち上げのため、ビジネス企画者も要件を決められない。
2. プロジェクトマネージャがそれぞれの進捗状況を確認し、管理するための工数が増加している。
3. ウォーターフォール型開発では、要件定義が完了した時には市場や社会状況が変化していたため、要求する変化に応えることができない。
4. 面白さや使いやすさなどのあらかじめドキュメント化しにくいクオリティを下げたくない

そこで変化に強いとされるアジャイル型開発を選択した。「アジャイル(agile)」は、敏捷な(able to move quickly and easily)を意味する英語の形容詞である。実際に、事例ではこれらの効果があった。

1. 実際に動くものを見ながらプロジェクトを進めた
2. 管理コストを低減できた
3. 変化に対応できた
4. 面白さなどのクオリティを維持できた

非ウォーターフォール型開発の成功度を調査したところ、37%の事例で「うまくいった」もしくは「かなりの部分でうまくいった」であり、残りは「うまく行った部分もあるが、うまくいっていない部分もある」であった。「うまくいかなかった」もしくは「まったくうまくいかなかった」を選択した事例はゼロであった。

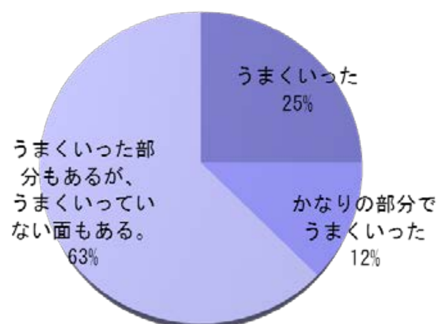

図 **5-1** プロジェクト成功度

Copyright © 2012 IPA, All Rights Reserved

5.2. 組織体制についての工夫

5.2.1. 準委任契約

アジャイル型開発では、要件や仕様、またその実現順序については当初の計画から大きな変更を伴うことが多い。プロジェクトの企画段階では準委任契約、製造段階では請負契約、運用保守段階では準委任契約がベストプラクティスのひとつである。しかしながら、アジャイル型開発では、開発対象のシステムやサービスがフィードバックを受ける頻度が高くなるため、作業内容自体が変更することは避けられない。請負契約であると契約の内容について見直し、契約変更を行う必要がある。

このような場合においては、作業量に応じた請求を行う「準委任契約」を結ぶ事例が多かった。準委任契約とは、『業務を委託する契約であり、ベンダー企業側の責任は、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』契約である。
この場合は、要件や仕様の変更についてリスクを発注側が負うことになる。それによって要件や仕様、その実現順序が変更になった際に、契約変更を回避した。

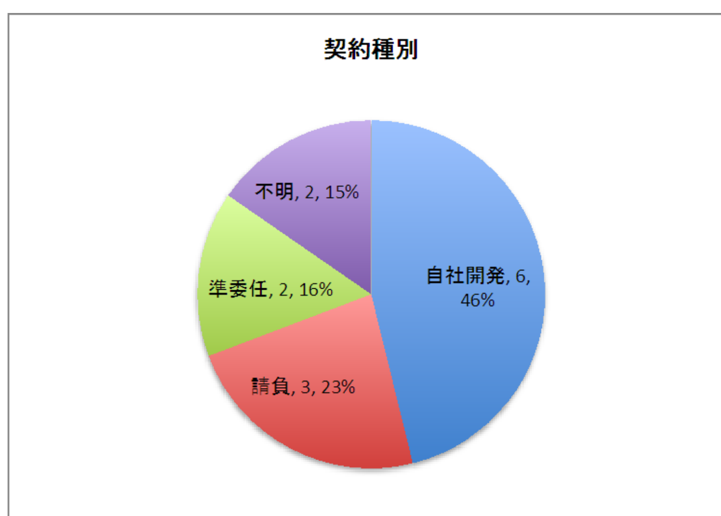

図 5-2 契約種別

今回の事例では、半数近くが自社開発によるサービス開発事例だったため顧客とベンダー間の契約については存在しなかった。残りの事例においては、請負が３件、準委任契約が２件であった。請負契約を選択している場合には、制約として他の契約方法を選択できないケースであった。

たとえば、請負契約という制約の中でも、発注側との合意の元で、開発工数を通常通り見積り契約の後、優先度の変動に従って、随時開発内容を見なおし金額は一定で、開発する機能を取捨選択して規模的に変動が出た場合随時見積り、工数を合意するという事例や、要件定義作業の一環としてプロトタイプを作成する位置づけで契約するという事例もあった。しかしいずれの場合も、請負契約でアジャイル型開発を実現するには一工夫が必要である。

Copyright © 2012 IPA, All Rights Reserved

したがって、発注側と受注側の間で、対象システムの置かれている状況を認識した上で、アジャイル型開発に向いた準委任、あるいは請負に一工夫して契約するのがよい。

5.2.2. 職能横断チーム

従来のチームの構成は、要件定義、設計、開発、テストというような職能別にチームが分かれることが多かった。職能で分離することで、専門に特化でき、それぞれの効率が増すというメリットがある一方で、職能間での繋りが弱くなりコミュニケーションの無駄が発生してしまいがちである。また、それぞれの職能においてやるべき事の実現に最適化されてしまうと、本来達成すべき目的である「ソフトウェアによる価値の創造」という観点を見失いがちになる。大規模になると、このように職能による分断が加速されることで、変化に対応しづらくなる。

このような場合、アジャイル型開発においては、チームは職能単位(企画チーム、設計チーム、開発者チーム、テストチームなど)で構成されるのではなく、ある機能を実現するために必要な職能を持つ人々を、部門を越えて集めてチームを作ることが推奨されている。

本調査においては、８割の事例が職能横断チームで構成されていた。その結果、ある機能・サービスを実現するという目的の元に、それぞれの職能のプロフェッショナルが協調してソフトウェアを通じた価値提供に集中することができた。

一方で、E 社の事例のように職能毎にチームを分けた事例もあった。この事例では設計時点での品質を担保することが強く意識されており、徹底的な設計レビューが実施された。また五月雨式で設計と開発が非常に近い時期に同時進行で行われていたため、設計者と開発者の距離が物理的にも時間的にも近かった。

したがって、できるだけ大規模においても職能横断チームを構成する必要がある。職能別チームの場合には、それぞれの距離を近くするように工夫することが求められる。

5.2.3. 役割を超えた支援

アジャイル型開発においては、プロダクトオーナー（ビジネス企画の役割）や、発注側の顧客、といった顧客役が、プロダクトバックログ（製品やサービスの要求リスト）を洗い出し、仕様を決めて、優先順位を決める責務がある。その責務を実現するためには、各関係者間との調整、要求の具体化や詳細化、開発者からの質問への回答、レビュー、受け入れテストを行う必要がある。顧客役は、更には開発されたサービスや製品を利用者や関係者に説明し、場合によっては教育なども実施する場合がある。そのため、反復する頻度に応じて、顧客役としての作業に時間をとられることになりボトルネックが発生する。このボトルネックが顕在化すると、開発者が開発する余力があっても、作る対象の要件が出てこない、顧客役に質

Copyright © 2012 IPA, All Rights Reserved

問を投げても回答が返ってこない、開発者が実現した機能を顧客役に提示しても、それを確認する時間がとれない、といったようなボトルネックになりがちである。

このようなボトルネックは生産性を低下させるため、そのボトルネックを排除して全体としての流れを円滑にする必要がある。

このような場合、ボトルネックが発生している箇所については、当初の役割を超えてサポートをする事例が見受けられた。たとえば、I 社の事例では、顧客役が明らかにボトルネックとして見受けられたため、開発チームから顧客役の支援チームを新たに構成した。更には顧客の会社へ一名が出向して、顧客役の完全支援に回る、などの解決策が見られた。

今回の調査においては、同様の顧客役のボトルネックの問題が完全には解消されずに課題として挙げられるという事例が複数あった。

したがって、顧客役がボトルネックになっている場合は、早めに検知して顧客役の支援者や支援チームを構成するのがよい。

5.2.4. チームメンバーローテーション

中大規模開発においては、必ず複数のチームに分かれる。たとえば、データベースアクセスなどを行うサーバ処理について開発するチームと、利用者へのインターフェースを提供するなどのクライアント処理について開発するチームに分かれている場合、お互いの業務知識や技術スキルに大きな違いがある。
しかしながら、全体としてひとつのサービスを構築したいために、それぞれの知識やスキルに乖離があると、整合性が取れず、トラブルを発生させてしまう。業務知識や技術スキルは、膨大で、かつ、日々更新されるため、すべてを文書化し共有することは現実的ではない。

その場合、チームや業務領域をまたいで、メンバー同士を交換することによって、知識やスキルなどを他に伝達する。メンバーを交換した直後は、一時的にチームのパフォーマンスが落ちることは避けられない。そのため、メンバー交換のタイミングや交換する人員については、各チームが連係をして注意を払う必要がある。しかしながら、長期的視点で考えた場合は、短期的なパフォーマンス低下よりも、長期的な知識伝播による効率化を選択する機会が多い。

たとえば、G 社の事例では、プロジェクトのアーキテクチャに自作の PaaS プラットフォームを利用していた。他チームは、その PaaS プラットフォームの上にサービスを構築している。PaaS チームとサービスチームのメンバーを交換することで、PaaS チームにとっては、利用者側の視点でのフィードバックを得ることができ、サービスチーム側にとっては、プラットフォームについてのより深い知識を得ることができる。メンバー交換時のパフォーマンス低下については、顧客にも連絡の後に調整しているため問題は起きない。

Copyright © 2012 IPA, All Rights Reserved

I 社の事例でも、チーム間のメンバーを積極的に交換して知識伝播を意図的に行っていた。交換するメンバーは、各チームのキャプテン(リーダーにあたる役割)同士が、それぞれの欲しい知識、人材を調整した上で交換を行っていた。

したがって、大規模プロジェクトでは、長期に渡る場合は、各チームのメンバーを積極的に交換し、互いの知識を伝播させるのがよい。

5.2.5. 不適合メンバーの入れ替え

アジャイル型開発の導入および促進を進めて行く上で、チームメンバーの中にはアジャイル型開発の方法に合わない人がいる。今回の事例調査の中でも、あるチームでは期待されている「主体的な発言や行動」をしなかったメンバーや、スキルはあるにも関わらず、状況の変化への対応スピードについていけないメンバーなど、そのチームのやり方に合わない人がいたことが報告されている。また、アジャイル型開発において採用されることのある準委任契約でパートナーとしてチームに入っている場合は、成果物に対する完成責任を負わないこともあり、チームとしてパフォーマンスが上げられないと、期待する費用対効果が得られないことになる。

そのような場合は、アジャイル型開発の方法に合わない人はチームから離れてもらい、他のメンバーがチームに新しく参画する。そのことによって、チームとしてパフォーマンスを最大限に上げることを目指す事例が５つの事例で発見された。

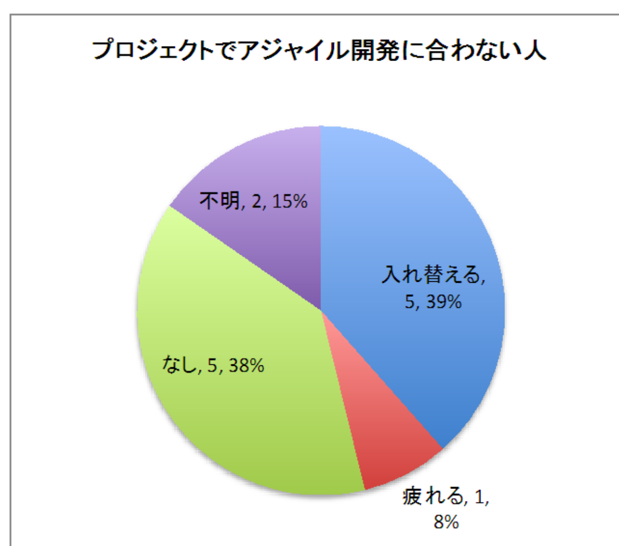

図 5-3 プロジェクトでアジャイル開発に合わない人の対応

複数の事例での合わない理由も「受動的な態度」、「スピードについていけない」という理由は共通していた。また「技術やメンバー同士の関係が密過ぎるのではないか」といった推測も挙げられていた。G 社の事例では、技術力は高い人が、スピード感についていけずに自ら辞退を申し出たという報告もあった。

Copyright © 2012 IPA, All Rights Reserved

なお、この事象は、ウォーターフォール型からアジャイル型開発への移行が発生する事例のみで報告され、逆に自社サービスを開発している事例で、自然発生的な開発プロセスからアジャイル型開発への移行の事例では報告されなかった。このことから、単純に「合わない人」の問題というよりも、組織の文化的な側面が大きな理由ではないかと考えられる。

したがって、アジャイル型開発に合わない人がいる場合は、チームのパフォーマンスを優先する、あるいは当人が望まない場合には、メンバーを入れ替えて新しいメンバーを参画させてアジャイル型開発が合うメンバーへ最適化していくのがよい。

5.3. コミュニケーションについての工夫

5.3.1. 全員同席

アジャイル型開発においては、プロセスやツールよりも個人との対話に価値をおいている。
特に、壁にタスクを貼り出すタスクボードや、全員が同じ場所で立って打ち合わせするなど、物理的に同じ場所にいることを前提にしているアジャイル型開発のプラクティスは多い。
しかしながら、ビジネス企画や開発者、デザイナーなどが別々の場所にいると、望む対話が行われにくくなる。また席か近いということは、単に話しやすいだけでなく、その場の状況、雰囲気などを皮膚感覚で感じとれるのが利点である。

そのような場合は、チームメンバーを同じ場所の近くの席に物理的に配置することによって、対話しやすい状況をつくりだす事例が複数見受けられた。

たとえば、B 社や C 社は 100 名を越える大規模事例であるが、どちらも同一フロアにメンバーを集め、チーム間の席を寄せることで全員同席の状況を作りだしていた。
D 社の事例でも、100 名を越える大規模事例であったが、同一フロアには収まりきらないながらも、開発者を同じ拠点に集めようと試みた。
E 社の事例では、元々企画側と開発側の席が別フロアであったものを、引越しなどを徐々に実施していくことで、最終的には隣接する場所に集めて既成事実を作った。
G 社の事例では、席を寄せるだけでなく、パーティションを取り除き、チームが一体感を作れるオフィス環境を作りあげた。

したがって、同一拠点内においては、関係者の席を集めてコミュニケーションがより円滑できるように物理的なデザインを行うのがよい。
その一方で、席を近づけたがゆえのデメリットも発見された。E 社の事例においては、企画側の開発側への不満を漏らした時に、席が近いため聞こえてしまい両者の関係が悪くなったという例が報告された。

Copyright © 2012 IPA, All Rights Reserved

5.3.2. 見える化

アジャイル型開発では、変化に柔軟に対応していくため、取り組んでいるタスクが変わってしまう。そのため、現在の状態が全員で把握できないと、各自が素早く適切な対応を取ることができない。

そのような場合には、取り組んでいるタスクや、抱えている課題について、現在の状態を見える状態にするため、大きなボードを用意し、そこで現在のタスクや課題がわかるように付箋紙やカードを貼付ける（タスクボード）。関わっているタスクや課題が書かれた付箋紙やカードを、自ら移動させることによって、最新の情報を維持し同時に回りに視覚で伝える。その結果、プロジェクトの管理者やリーダーだけでなく、チーム全員が、自分達のチームの状況を一目で把握できる。そのため、適切な判断がとれて管理の手間を削減できる。
今回の事例では５事例がタスクボードを利用していると回答した。

たとえば、C 社の事例では、部屋の冷蔵庫などの物品を移動して、模造紙を貼りつけるスペースを作り、巨大なタスクボードを作成し、その前にチームが集り状況を一目で把握できるようにした。最初は表計算ソフト Microsoft Excel で管理されていた項目を、付箋に書出していたが、最近は Excel での管理をやめてタスクボードをマスターとするようになってきた。またボードの前に広いスペースを設けてあり、そこで計画づくりのようなミーティングを実施することで、全体を見渡しながらの計画づくりが可能になっていた。

E 社の事例では、壁に貼り出すスペースがなかったが、ホワイトボードを３面用意して壁代りに利用していた。

G 社の事例では、オフィスの壁、ロッカー、机の前の低いパーティション、ホワイトボードなど、ありとあらゆるスペースを使って、マイルストーンまでのリリース計画、直近の反復におけるタスク、未解決の障害、バーンダウンチャート(反復内での残作業量をプロットして視覚化するグラフ)を貼り出して共有していた。

I 社の事例では、拠点は分散していたが、それぞれの拠点毎にタスクボードを使って、それぞれの拠点内での最適化を図っていた。

なお、長期的な視点するスケジュールや、大きなプロジェクトの流れについては、チケットシステムや Wiki（編集できる Web ページ）などのツールを使い、スプリントやイテレーション（反復期間）内の短期的なユーザストーリやタスクについて、壁やホワイトボード上に張り出し、見えるようにする事例が多かった。
したがって、同一拠点内においては、直近の反復のゴールやタスクを付箋に書出して、壁やホワイトボードに貼り出してチーム全体で状況を共有しながら作業をすすめるのがよい。

Copyright © 2012 IPA, All Rights Reserved

5.3.3. 段階的朝会

アジャイル開発では「朝会」というプラクティスがあり、毎朝、短時間で状況を共有する。中大規模開発において、全員で朝会をすると、朝会にかかる時間が長くなる、あるいはそもそも全員で同時に集まることが不可能になる。また、参加者ひとり当たりの発言時間が短くなり、活発な会話が出来なくなる傾向がある。さらに、中大規模開発では、複数のチームで取り組むことがほとんどであるが、チーム間の情報共有や連携が難しくなるという課題もある。

このような場合は、段階的に朝会を行うという事例が報告された。
たとえば、B 社の事例では、最初にチーム内で朝会を行い、次に各チームの代表者による朝会を行うというように、朝会を 2 段階で行うことでチーム間の連携を行った。これをスクラム用語で「スクラムのスクラム(Scrum of Scrums)」と呼ぶ。

G 社の事例では、二段階の朝会に加えて、代表者がチームに状況を持ち帰り、さらにチーム内で朝会を行う事例もあった(三段階朝会)。その結果、中大規模開発のプロジェクトにおいても、日次で全体の状況を把握しながらの推進が可能になる。

5.3.4. 完全透明性

中大規模開発において、複数チームやステークホルダーが多くなり、組織が複雑になる。そのため分割されたチームや個人個人が内部で何をやっているか、どのような気持ちでいるのかを理解が難しくなってくる。

準委任契約を結ぶ場合には、成果に対して責任を負わない代りに、発注側と受注側の双方の信頼関係が重要になる。請負契約である場合でも、開発の状況をブラックボックスにしてしまうと、その中で何が行われているかがわからなくなり、特に状況が芳しくない場合は不信感を醸成してしまうことが多い。

そのような場合には、開発に関するすべての情報を隠すことなく公開するという事例があった。

たとえば、G 社においては、ソースコードリポジトリ、チケットシステム、Wiki など開発に関係する情報を文字通りすべて発注元に公開していた。開発状況だけでなく、開発者と発注者が同じ社内ＳＮＳ（ソーシャルネットワークシステム）を使って開発作業の近況、問題が発生した時の気持ちなどもオープンにするようにした。また Wiki 上の仕様書や、その仕様書に従って作られているかのテスト結果、リリース手順の計画、リリースノートをすべて記述して発注元に見せる。発注元は、発注先の状態や取り組み、期待通りに進まないときの理由などを手に取るように確認できる。これによって、発注元との信頼関係を築くだけでなく、第三者監査などの証拠としても有用である。
H 社の事例においては、受託開発であったが、開発と発注元が同じチケットシステムを共有して計画やその進行状況を公開して信頼関係を築くことができた。

Copyright © 2012 IPA, All Rights Reserved

したがって、発注元と開発との間には、進行状況や、問題などをすべて公開して、互いの信頼関係を促進させるのがよい。

5.3.5. チームまたぎのふりかえり

あるスプリントやイテレーション（反復期間）の終わりに、そのときの良かった点、問題点、改善点などについて対話するふりかえりは、継続的な改善や、チームの学習を促進するために欠くことができないアジャイル型開発のプラクティスである。今回の調査でもほとんどのチームが実施していた。その頻度を調査した結果が図 5-4 である。(複数回答あり)

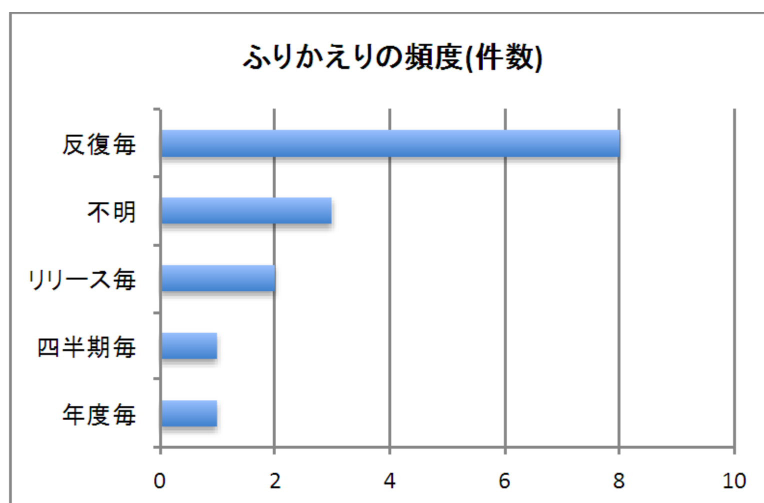

図 5-4 ふりかえりの頻度の事例件数

中大規模開発で、チームや組織が分かれているときには、コミュニケーションは主に自分の属するチームに関することが多くなる。そうなると、相対的に、他のチームの状況を知らず、相手の立場を想像することが困難になってくる。その結果、チームや組織が複数に分かれることの多い中大規模開発のふりかえりでは、他のチームや組織の悪口になってしまい、本質的な改善に結びつかない状況になってしまうことがあった。

そのような場合には、チームをまたがったふりかえりを実施するという事例があった。

たとえば、B 社の事例では、うまくいっているチーム、いっていないチームを集めて半日ものふりかえりを実施することで、問題点の洗い出しができた。

C 社の事例では、ふりかえりの度に、次にスクラムに巻き込みたい部署を選んで、ふりかえりの度にチームを拡大していった。

E 社の事例では、毎週のふりかえりとは別に、全体で年度単位での大きなふりかえり会を実施していた。また、ふりかえりで出る問題がチームで解決できない場合は、事業責任者が率先して解決に動くことで、チームの範疇を越えた問題解決ができたという事例があった。

Copyright © 2012 IPA, All Rights Reserved

たとえば、G 社の事例では、基本はチーム単位でふりかえりを実施するが、プロジェクト全体で人をシャッフルしてチームとは別の単位のグループを作りふりかえりを実施することで、プロジェクト全体としての問題意識や、その改善を実施することができていた。

同様に、I 社の事例でも、チーム横断的なふりかえりを実施して、全体としての改善を行っていた。また I 社では、三ヶ月に一度、全体で大きなふりかえりを実施することで、体制や管理方法の見直しのような、より大きな変更を行っていた。

したがって、チーム内でのふりかえりを主としながらも、チームをまたいだふりかえりも実施するのがよい。その結果、チーム間での情報や問題意識の共有が進み、部分での改善ではなく、プロジェクト全体としての改善や共通的な施策が可能になるだけでなく、ふりかえりのマンネリ化を防ぐなどの効果もある。

5.4. 展開についての工夫

5.4.1. パイロット導入

大規模事例や組織導入において、いきなり全体にアジャイル型開発を展開するということは考えられない。最初に効果やリスクを把握した上で導入する必要がある。導入の推進者は、アジャイル型開発のよさは十分理解している。

しかしながら、現場の開発者、他のプロジェクトマネージャ、そして組織の経営陣などに、いくら効果を説明しても、なかなか説明だけでは効果を伝えきれない。

そのような場合は、最初に実証実験を兼ねて、試行プロジェクトで結果を出した後で、その結果を起点に、徐々に展開していくことになる。

本調査の事例において、組織、あるいは特定のプロジェクトにアジャイル型開発を展開したい場合には、最初に小規模なプロジェクトで試してみて、効果や勘所を掴むという事例が 6 割以上であった。

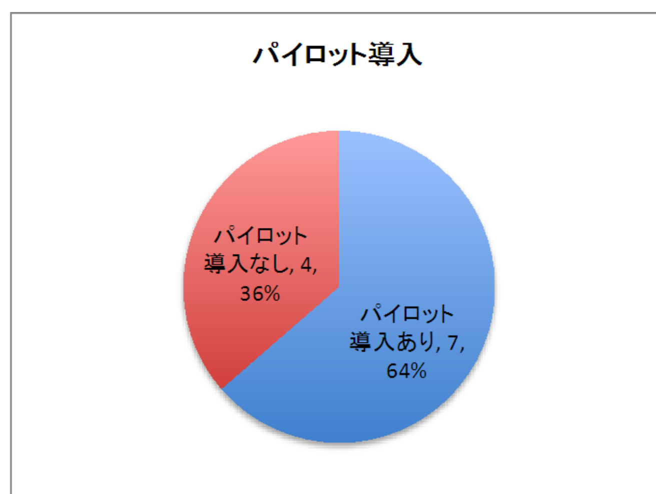

図 5-5 パイロット導入実施率

Copyright © 2012 IPA, All Rights Reserved

たとえば、B 社の事例では、最初は意図的にアジャイル型開発を導入したのではなく、海外と共同プロジェクトを実施する中で、海外企業がスクラムを実践していたために導入していた。
D 社の事例では、基幹システムの本格開発の前に、パイロットプロジェクトを立ち上げて、開発者の訓練、アジャイル型開発のフィージビリティスタディも兼ねて実施検証した。
F 社の事例では、小規模プロジェクトを２件導入して効果を実証した後に、スクラムの全社展開へとステージを上がった。
J 社の事例では、アジャイル型開発をベースとした自社標準開発プロセスを策定し、それを適用したプロジェクトで成果を出した後に、経営層へ説明した。

したがって、アジャイル型開発の本格導入の際には、まずパイロット導入で実際のプロジェクトでの手応えを掴み、効果を見えるようにした後で、本格的導入に進むのがよい。

5.4.2. 認定研修･コンサルタントの利用

自分達でパイロット導入を実施してみたが、具体的にどうしていいかわからない、または自分達が行っていることが、本当に正しいのか不安になることがある。あるいは、アジャイル型開発をはじめたいが、そもそもどうやって始めればよいかが確信が持てないことが多い。

そのような場合には、スクラムの認定研修[6](CSM 認定スクラムマスター、CSPO 認定スクラムプロダクトオーナー)や、各種アジャイルトレーニングやコーチングサービスを利用して、実際の自分達のプロジェクトの前や、最中に学んで進めていく事例が 6 割以上見受けられた。その結果、我流で試していた場合には、間違って理解していた部分を改善できる、あるいはプロジェクトの中で羅針盤としてアドバイスを送ってもらえる、という効果があった。

たとえば、D 社、E 社、F 社では外部のアジャイルコーチやメンター(実際の開発も行うアジャイル型開発経験者)を呼んで、実際のプロジェクトの中で、そのポイントを体得していった。
B 社、F 社、K 社の事例においては、認定スクラムマスター研修を受講して、スクラムの本当の意図する目的、やり型を学び、推進に拍車がかかったという効果があった。

更には、自分達で社内研修を準備する場合でも、研修で学んだ内容を元に教育コースを作ったという効果もあるという事例も複数存在した。

[6] http://www.scrumalliance.org/

Copyright © 2012 IPA, All Rights Reserved

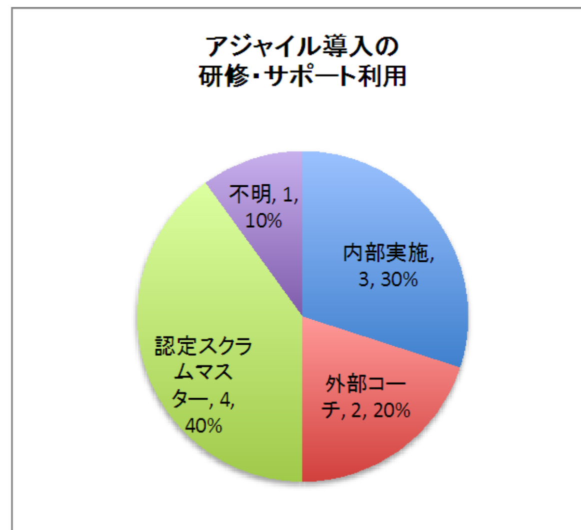

図 5-6 アジャイル導入の研修・サポート利用率

5.4.3. 漸進的な人数増加

B2C サービスのようにフィージビリティスタディの要素が強いシステムは、まずシステムを少人数で開発してリリースし、利用者のフィードバックを得ながらシステムが成長していくのが一般的である。そのため、プロジェクトに対する人員の増加は、システムの成長に合わせて自然に人を増やしていくことになる。(システムの自然な全体性拡大) 他方、受託開発のように、顧客の実現したいことが見えている場合では、その全体像を元に精緻化、詳細化して全体を作りだそうとする。(システムの意図的な全体性拡大)

しかしながら、単純に人が多くなると、全員でのコミュニケーションが困難になる。そのため組織体制や、コミュニケーションパスを確立していかないといけないが、最初から多くの人員を集めて、複数のチームを作り、体制を作ってみても、なかなかチーム間のコミュニケーションはうまくいかない。なぜなら、まずそれぞれのチームがアジャイル型開発の目指す自己組織化されたチームとして立ち上がるのに時間がかかり、更にはチームとチームの間で利害関係、信頼関係を確立できていないと、チーム間のコミュニケーションは成立し難いからである。

そのような場合には、人員は徐々に増やしていくことで、人々が共同の経験をしながら、知識を伝播しながら、人数及びチーム数を増やしていくのがよい。

自然にサービスが成長していく場合は、人員の拡大も自然と増えていく形になる。A 社の事例では、3 年以上にもわたるサービスの成長と共に、4 名からスタートした人員が 10 名、25 名、と増えつづけ最終的には 100 人規模に増えた。B 社の事例では、関与する人員を 20 名→50 名→100 名と段階的に増やしていくことで、その規模に対応した。

他方、受託開発のように最初からある程度の規模が想定されるプロジェクトもあった。その場合においても、最初から開発リソースを一度に調達せずに、意図的に少しずつ人員を増や

Copyright © 2012 IPA, All Rights Reserved

していった事例が I 社の事例であった。I 社では人を増やすタイミングを反復単位と定め、その人数も制約をかけていた。これは反復の途中ではなく、反復開始のタイミングから人を増やすことによってリズムに合わせた人員の追加と、自然なチームへの浸透の実現という理由からであった。最初は反復の単位で４名ずつ増やしていたが、途中から反復あたりの人数を２人に変更して自然な人員増加をデザインした。

１チームあたりの人数は、どの事例も 8〜10 名をその上限としていた。そのため１チームの人数が増えすぎた場合には、分割していくことになる。

したがって、人を増やすことで開発力を増強できる一方で、その人々が自律的に行動するための知識がないならば、逆に人が増えても、その人の面倒などでチームとしての機能が低下してしまうリスクがある。そのために漸進的に人員規模を増やしていくのがよい。

5.4.4. 漸進的な展開

既に大勢の人が参画しているプロジェクト、あるいは組織が存在する。新規のプロジェクトでアジャイル型開発を実践する場合には漸進的な人数増加を実施すればよいが、既にプロジェクトは始まっており人も大勢参画している。そこに対してアジャイル型開発を導入して問題を解決し成果を出す必要がある。しかしながら、すべてのチーム、人員に対して一斉に教育や、支援を行いながら導入することは難しい。またその利点などを一度に理解してもらうことも難しい。

そのような場合には、既に始まっているプロジェクトに対してアジャイル型開発への導入展開を漸進的にするという工夫をしている事例が数例あった。

例えば、B 社の事例では、最初は 20 名のプロジェクト開発メンバー中心でスクラムを導入した。次に企画も交えて 40 名のプロジェクトでスクラムを導入した。最新では 100 名を越える部門全体に対して、開発、企画、デザイナーを含めてスクラムを拡大していった。最近では開発には直接関わらない、マーケティング部門も巻き込みはじている。

C 社の事例では、プロジェクト内においてスクラムを漸進的にプロジェクトに展開していった。最初は開発チームだけでまずスクラムを実施した。反復終了時のふりかえりで、次に巻き込みたい役割を特定し、デザイナーを巻き込んでいった。次の反復終了時のふりかえりではプランナーを巻き込むことにした。C 社の場合は、壁の大きな見える化を行っていたため、チーム外から見ても、何をやっているのかが、スクラムを実施していないメンバーからも気になるようにしていた。そのため新しいメンバーが興味を持ち導入しやすかった。それ以降の反復においては、他チームにスクラムの説明会を実施して「何をするか」ではなく「なぜ必要か」という点を説明して理解をしてもらい拡大をしていった。また C 社では「スクラム、アジャイル型開発」といった手法を明言しない、あるいは展開しやすそうな所から広げていた。

Copyright © 2012 IPA, All Rights Reserved

E 社の事例では、開発と企画チームが同じ拠点だが別フロアであった。しかし引越しや席替えを繰り返して徐々に近付いていき、最後には同じ島になった。

したがって、アジャイル型開発導入と起点となる領域は、まずアジャイル型開発の必要性の高い(変化に対応したい、要件が決めにくいなど)領域が最初で、更に導入障壁が低い領域であった。導入障壁が低いとは、推進者が話をしやすい、アジャイル型開発に合いそうな人がいる、推進者の影響範囲が届く、といった導入しやすさの総称である。やりやすい領域から順に始めて拡大していくのがよい。

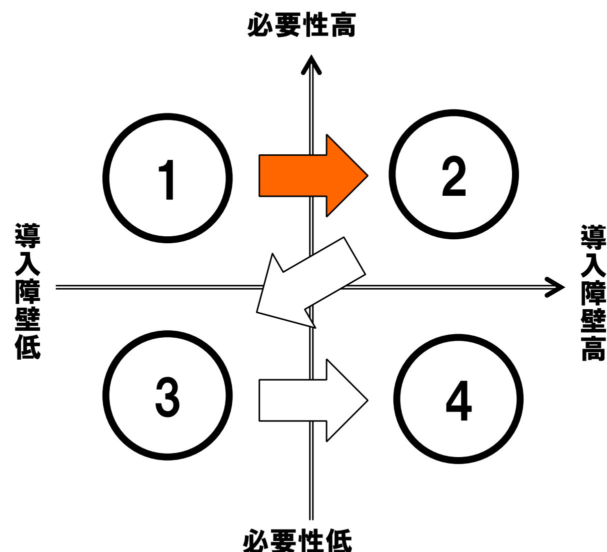

図 5-7 展開の順序

5.4.5. 社内研修･勉強会

大規模プロジェクトになると、すべてのメンバーが優秀なスキルを持つとは限らない、また各チームの知識のバラツキも発生する。アジャイル型開発は本などで学ぶことが難しい。実際に自分でやってみるか、経験者からの直接の教育などがないと理解することが難しい。より効果的な展開に関しても、回りの理解、正しい認識をしてもらうことが求められる。

そのような場合には、社内またはプロジェクト内で勉強会を実施して展開に役立てているケースが 5 事例あった。

たとえば、B 社の事例では、認定スクラムマスターの研修を受けた経験を生かして、スクラムの効果を体感できる研修を行った。また技術的な面では組織としてハッカソン(プログラマが集ってテーマに即したプログラミングをするイベント)を開催していき、その技術を高めていった。

C 社の事例では、展開に先立ってスクラムについての勉強会を 9 回ほど実施して、その必要性などを回りに伝えていった。またその勉強会も一度に大勢参加させるのではなく、10 人程度の人数でじっくり行っていった。

F 社の事例では、全社展開が決まったので、社内でアジャイル型開発の研修をカリキュラムとして取り入れて実施している。

Copyright © 2012 IPA, All Rights Reserved

G 社の事例では、毎週プロジェクトメンバー全体で勉強会を実施しており、様々な領域の学習を促進していている。

I 社の事例では、利用するフレームワークや、アジャイル型開発の研修を実開発と並行して段階的に実施した。

B 社、C 社の事例では、いずれもキーマンが認定スクラムマスター研修に参加して、自分達がアジャイル開発を指導する立場に立った時でも役に立ったと明言していた。

アジャイル型開発は、チームが継続的に学んでいくことで、成長していくことが前提となっている。展開時だけでなく、普及後にも、新しい技術や、自分達のプロセスの改善などを日々学び実践していくことが重要となっている。

5.5. 分散開発についての工夫

5.5.1. 同一拠点から分散へ

コストを削減したい、開発に関わる人数を増やしたい、などの理由から分散拠点開発を進める場合は、別拠点にチームを作る。
しかしながら、最初からチームを分けてしまうと、「自分たちのチーム」と「あのチーム」(Us vs. Them)というように全体として一つのチームと考えることが難しい。更には別拠点ということで、中心拠点とは別な組織体であることが殆どである。そのため御互いに「知らない人々」との関係になりがちでコミュニケーションにおいては妨げになることが多い。

分散開発を実施していた 6 事例のうち、４事例においては、最初は同じ拠点でチームを作り開発し、その後拠点に分散してチームを作った、という調査結果が出た。事例毎の状況を紹介する。

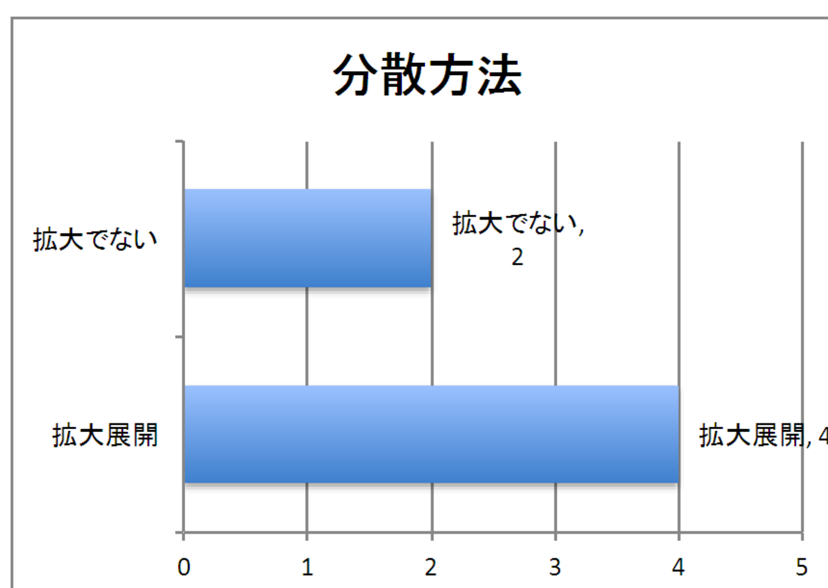

図 5-8 分散の方法

Copyright © 2012 IPA, All Rights Reserved

E 社の事例においては、東京と東北拠点での分散開発を実施したが、それに先立って東京でチームを組んで同一拠点で開発した後、東北に分散していった。分散後も何度かプロダクトオーナーが東北に行き、情報を共有していた。

F 社の２事例においては、日本と中国での分散開発を行ったが、最初に中国から人を日本に連れてきて、日本で開発を経験した後に中国に人を帰してチームを構成した。中国の人には、日本にいる間に、テスト駆動開発、継続的インテグレーションといった技術プラクティスを学んでもらった。また、日本からも技術コーチを中国に送り込んだ。

I 社の事例においては、首都圏と地方拠点での分散開発であった。首都圏で最初はチームを作り後に分散した。本事例では本格的な規模の開発が起ち上がる前から、プロジェクトの背景や経緯も含めて学んで欲しいということから、最初は地方拠点からメンバーを首都圏の拠点に呼びよせた。本事例では、地方拠点で担当するサブシステムを、システムをリリース後も運用保守を見据えて選択したという、長期的視点を持った分散展開であった。

これら事例のように、分散開発においては、細胞分裂のように、最初は一つのチームからはじめ、徐々に拡大していく中で分散していくことで、全体への知識やスキルの伝播が円滑に行われた。

5.5.2. チケットシステム

開発においては、要件、開発に付随する課題、障害など様々なやるべきことを登録しておく必要がある。同一拠点であれば、Excel や壁に貼ったタスクボードなどで共有することができるが、分散開発であると、Excel ファイルによるファイル共有によるファイル更新競合が発生しやすくなり、壁などのアナログツールを使ってチームの中で一見してわかる情報共有が実現できない。

そのような場合は、チケット管理システムと呼ばれる、Web ベースのツールを使って、要件や作業をシステムに登録してその状況や経緯を更新して、チーム内外との情報共有を計る。次に今回の調査事例においての利用状況を表にした。(比較のため Excel も含めている)

Copyright © 2012 IPA, All Rights Reserved

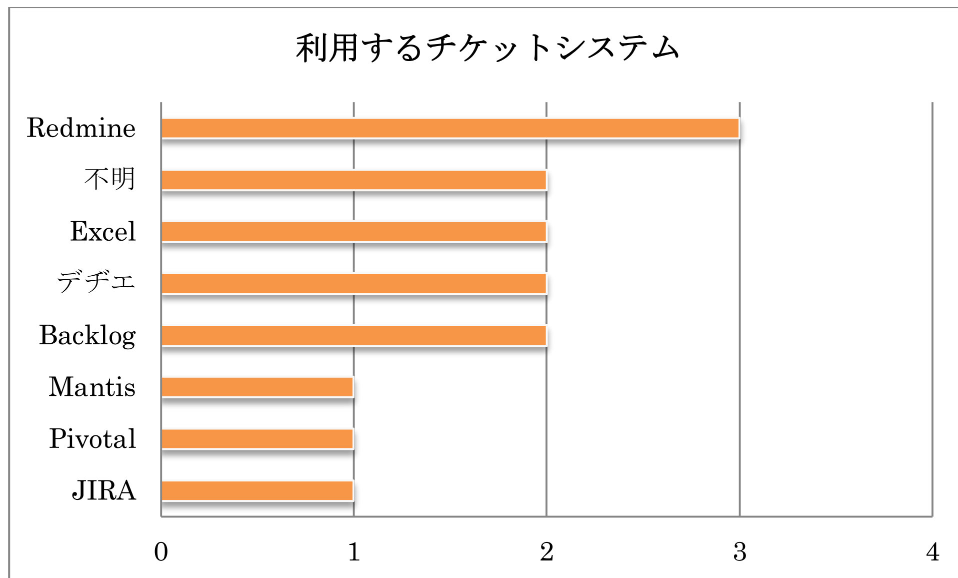

図 **5-9** 利用するチケットシステム

本調査においては、チケットシステムはRedmine[7]が一番よく使われていた。以降、特に分散開発についてのチケットシステムについての工夫を紹介する。

E社事例では、当初はRedmineを利用していた。しかしビジネス企画側と共有するには、開発者向けのツールは使い辛かった。そのためサイボウズデヂエ[8]を代りに利用することで、ビジネス企画側にも積極的に利用してもらえるようになった。

G社事例では、改修要件、新たな要望などが発生した時点ではチケットシステムには登録していなかった。まず付箋に書いて壁に貼り出しておき、実施することが確定した時点でチケットシステムに登録するという工夫をしていた。その理由としては、単にチケットシステムに登録していくと、登録されたが実際には実施されないムダなチケットが急増してしまって、チケットシステムの利用自体に影響をきたすからであった。

I社事例では、ExcelやMantis Bug Tracker[9]にて要件や障害は管理しているが、拠点毎ではその情報を元に壁のタスクボードを使い、チーム内で、よりわかりやすい状況の更新と共有を実施していた。

H社事例では、チーム間の情報共有にはRedmineを利用していたが、アジャイルチームが担当していた機能に関しては、Pivotal Tracker[10]を利用して要件を顧客との間で共有していた。視覚的に要件のボリューム、優先順位が見えるため、顧客からの評判が非常に高かった。

[7] オープンソースのプロジェクト管理ツール http://redmine.jp/
[8] Web データベースソフト http://products.cybozu.co.jp/dezie/
[9] バグ追跡システム http://www.mantisbt.org/
[10] http://www.pivotaltracker.com/

Copyright © 2012 IPA, All Rights Reserved

チケットシステムを利用する場合には、E 社、H 社のように開発者以外のステークホルダーを巻き込もうとした場合は、開発者目線の利点だけでなく特に要件を入れたり、優先順位をつけたりするビジネス側の使いやすさも考慮することが重要であった。
また I 社のようにツールに縛られることなく、その場で最適なツールを選択していくということも重要である。

したがって、チケットシステムを単に用いるだけではなく、拠点間で開発者だけでなく顧客とも状況を共有しながら、工夫をこらすことが重要である。

5.5.3. TV 会議

拠点が分散化すると、対面による対話が困難になる。しかし、昨今ではPC上で利用できるビデオチャットシステムが普及してきておりネットワークセキュリティ上の制約がない限りは、安価な拠点間でのビデオ会話が可能になっている。また本格的TV会議システム[11]を使うことで、より鮮明な拠点間コミュニケーションが実現できる。TV会議システムとビデオチャットシステムの違いとしては、複数拠点において快適に映像＋音声が配信できるかどうかであると言われている。

今回の分散開発事例では、６事例中、３事例がビデオチャットシステムを利用してコミュニケーション支援をしていた。また分散開発の６事例のうち、３事例がTV会議システムを利用して拠点間の常時リアルタイム会話を可能にしていた。H社の事例においては、コンシューマゲーム機[12]のビデオチャット機能を利用して、安価で高品質なTV会議システムを構築していた。

このような場合には、無料で実現できるビデオチャットソリューションはもちろんのこと、構築や利用に多少のコストがかかったとしても TV 会議システムを使うことで、拠点間のコミュニケーションを加速することができる、ということがわかった。

5.5.4. リアルタイムチャット

メールは記録性があるが、リアルタイムなコミュニケーションには適していない。一方映像や音声による双方向コミュニケーションシステムはリッチなコミュニケーションを実現できるが、双方の時間を拘束してしまうデメリットがある。もっと気軽にリアルタイムで拠点間でのコミュニケーションを実現したいというニーズがある。

このような場合は、テキストメッセージによるリアルタイムコミュニケーションとしてチャットシステムを使う事例がほとんどであった。

[11] http://www.polycom.co.jp/
[12] http://blog.ecstudio.jp/ec_studio_blog/080930playstation3.html

Copyright © 2012 IPA, All Rights Reserved

昨今、普及しているビデオチャットシステムは、テキストメッセージのやりとりも可能である。また、テキストベースのインターネット・チャット・システムの枠組みである IRC[13]を使っている事例も２事例存在した。

G 社事例では、IRC を開発者だけでなく顧客にも公開し、リアルタイムに全員で共有すべきことを、IRC を用いて告知していた。

したがって、求めるコミュニケーションの質によって、ツールを使いわけることが重要である。気軽なリアルタイムコミュニケーションにはテキストチャットが向いている。

5.6. アーキテクチャについての工夫

5.6.1. 最初のアーキテクチャ構築

アジャイル型開発では、BUFD(Big Up Front Design)を嫌い、創発的に生まれるアーキテクチャを漸進的に成長させていくことが推奨されている。一方で、中大規模開発において、大勢の開発者が、共通基盤としてのアーキテクチャをなしに、業務ロジックやインターフェースなどを設計実装し、システム要件を満たすアプリケーションを実現することは難しい。

このような場合、今回の事例においては、組織としてメンテナンスしている共通基盤を利用する、あるいはリードエンジニアが先行して設計開発している事例が多かった。

たとえば、A 社の事例では、初期開発の段階でリードエンジニアがスケールを視野に入れたフレームワークを設計実装し、その上でアプリケーションを開発していった。かなりのサービス規模まで当初のアーキテクチャで拡大できたが、ある段階で現状のアーキテクチャではサービスを維持できなくなり、ハードウェア構成から含めてデータストア部分の大きなアーキテクチャ改修を行い、スケーリングに対応することができた。

F 社、I 社の事例では、A 社の事例と同様に、リードエンジニアが最初に共通基盤を作り、その上でアプリケーションを開発していた。
H社、K社の事例では、標準的なLAMP[14]構成での開発だったため、アーキテクチャ的に特別な対応はしていなかった。

したがって、中大規模開発のアジャイル型開発においては、創発的なアーキテクチャよりも、アーキテクチャ要件を満たす共通基盤の利用を行うか、プロジェクト初期にアーキテクチャ部分を設計実装して、その後のアプリケーション開発に進むのがよい。

[13] http://tools.ietf.org/html/rfc1459
[14] Linux、Apache、MySQL、Perl、PHP、Python などの動的な Web サイトの構築に適したオープンソースソフトウェア群の総称

Copyright © 2012 IPA, All Rights Reserved

またアーキテクチャ部分にアプリケーション部分が従うだけではなく、アプリケーションでのフィードバックをアーキテクチャ側が受け入れて改善する必要がある。
アーキテクチャは、サービスの特性にあわせ、適切なものが選択されたい。たとえば、インターネットからの大量にアクセスがあり、かつ増加が予測される場合、スケールするためのアーキテクチャを構築すべきである。

5.6.2. アーキテクチャ・基盤チーム

中大規模開発のアジャイル型開発において、アーキテクチャは複雑になり、アーキテクチャの構築や保守も専門性が必要になる。同時に、アーキテクチャを利用するアプリケーションエンジニアからのアーキテクチャの選択や保守も難しくなる。

その場合には、アーキテクチャ・基盤についての専門チームを設けて、アプリケーションに適したアーキテクチャの構築や保守を行う。場合によっては、開発側に対して使い方の教育や支援を行う事例が複数あった。

たとえば、A 社の事例では、基盤チームを途中から立ち上げて、主にサービスのスケールアップ、スケールアウト、パフォーマンスチューニングを支援した。
F 社の事例では、同様にアーキテクチャチームを結成して、技術的な面で開発チームに支援を行っていた。
J 社の事例では、共通基盤を組織的にメンテナンスしており、開発チームへの情報提供や、時には現場でのチュートリアルなども行い、組織横断的な組織として活動していた。
G 社の事例では、PaaS 部分もプロジェクト内で開発していたため、他のサービスはその PaaS 部分を利用し、PaaS チームと密な連係をとっていた。

したがって、アーキテクチャや基盤を専門的に見るチームを構成して開発チームの支援をしてもらうのがよい。

5.6.3. 組織の共通基盤アーキテクチャの利用

プロジェクト毎にその時の要件にあったアーキテクチャを設計実装して利用する必要がある。しかしながら、速度が求められる開発になると、アーキテクチャ部分の設計や実装にそれほど時間をかけることはできない。

もし組織的に利用されてメンテナンスされている共通基盤が存在するならば、その共通基盤を利用することで工数が削減でき、アプリケーション部分の開発が効率化される。

たとえば、B 社の事例では、同社が開発しオープンソースにしている共通基盤を利用することで、必要なシステム要件を満たすアーキテクチャを実現できて工数を削減できた。
C 社、J 社の事例では、対象の開発に最適化された共通基盤が存在するため、その上での開発に専念することができた。

Copyright © 2012 IPA, All Rights Reserved

したがって、組織として日頃から利用され実績のある共通基盤が存在するならば、利用してプロジェクトの開発を効率化するのがよい。

5.6.4. アーキテクチャの改善

中大規模案件において重要なアーキテクチャについて、既存の資産を活かす、あるいは自分で開発したアーキテクチャを利用している。しかし開発が進み、サービスを公開して拡大していく中で、当初の想定とは異なる状況になっている。サービスの利用者が飛躍的に拡大していき当初の規模以上になってきた。あるいは利用していた共通基盤ではサポートしていない要件がでてきた。

そのような場合は、アーキテクチャ自身の改善も行って成長させていくという事例があった。

たとえば、A 社の事例では、ある程度までのサービス規模まで当初のアーキテクチャで拡大できたが、ある段階で現状のアーキテクチャではサービスを維持できなくなった。そのためハードウェア構成から含めてデータストア部分の大きなアーキテクチャ改修を行い、スケーリングに対応することができた。
B 社の事例では、組織的な共通基盤を利用しているが、実際にはそこに収まらないアーキテクチャ要件がアプリケーションから出てきた。それら追加要件については、プロジェクトの複数チームが共有した後、協働でアーキテクチャ要件についての改善にあたった。
上記２例はいずれも、組織的に自分達で基盤を作り育てている例であった。

したがって、アーキテクチャを事前に設計実装、あるいは共通基盤を利用していても、追加要件が必要な場合には改善をおこなってアーキテクチャ自身も成長させていくのがよい。

5.7. システム分割／インテグレーションについての工夫

5.7.1. 疎結合での分割

中大規模なシステムをサブシステムに分割するには、様々な観点があるが、大きく分けると、外部ビューの視点による分割、外部ビューに依存しない内部的構造による分割に分れる。サブシステムに分割した際に、それぞれのサブシステム間の依存関係が密であると、開発時のオーバーヘッドが大きくなってしまう。それぞれのサブシステムの独立性を高める必要がある。

そのような場合には、今回の調査事例では 3 つの指針でサブシステムを分割していた。

指針 1. サービス単位
指針 2. プラットフォーム単位
指針 3. コンポーネント単位

Copyright © 2012 IPA, All Rights Reserved

指針 1 の例では、A 社の事例では、提供サービスの機能単位でシステムが分割されていた。

指針 2 の例では、G 社の事例では、システムは 2 の PaaS、Flex、Android という分割であった。F 社の事例では、スマートフォンのプラットフォーム別に Android、iPhone、Windows Phone のように分割されていた。

指針 3 の例では、D 社は、フロント、各業務、基盤という形に分割されていた。
I 社の例では、フロント、バックエンド、決済のように分割されていた。
C 社の事例では、UI、ネットワーク、キャラクタ、背景といった単位でシステムが分割されていた。

したがって、それぞれが疎結合になるようにシステムを分割していくのがよい。その結果、それぞれが疎結合で、独立性を持って開発を進めることができる。

5.7.2. フィーチャーチーム

アジャイル型開発では、チームを水平分割(UI層、ビジネスロジック層、DB層)によってチームを区切るのではなく、「動作するソフトウェアを顧客役が確認できる」単位、つまり「顧客に価値を与える」単位であるフィーチャー(サービス、機能、特徴)単位でチームを構成していくことが推奨されている。これらのチームをフィーチャーチーム[15]と呼ぶ。その結果、各チームが反復の度に実現するソフトウェアは、顧客役が確認してフィードバックを得ることができ、なおかつ顧客役に価値を提供できる単位となる。システムを機能・サービスといった、利用者からみて確認できる単位でシステムを分割している場合には、チームもその単位で分けることにする。

本調査においては約 5 割のチームがフィーチャーチームによるチーム構成を選択していた。システム分割の内訳でみると、指針 1 のサービス単位、あるいは指針 2 のプラットフォーム単位がそれに当る。チーム編成の割合を図 5-10 でみると、フィーチャー以外には、コンポーネント(システムの内部構成による分割)、職能(設計、開発、テスト)、ミッション(ビジネス上の目的)でチームが構成されていた。

[15] http://www.featureteamprimer.org/ja/feature_team_primer_ja.pdf

Copyright © 2012 IPA, All Rights Reserved

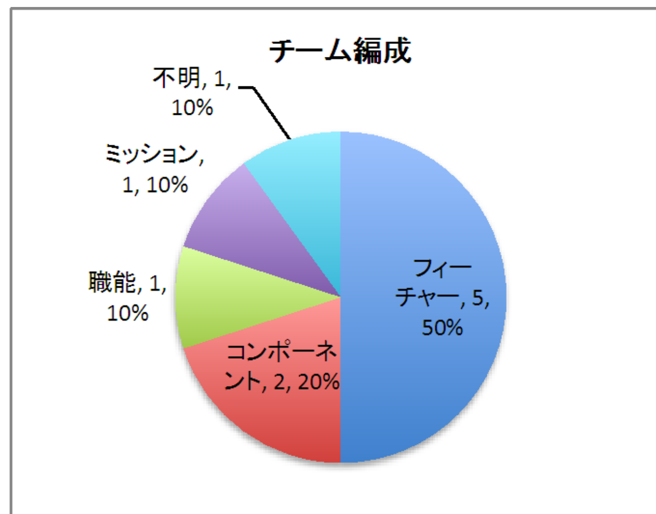

図 5-10 チーム編成(中大規模のみ)

たとえば、B 社の事例では、機能よりも抽象的な単位であるミッション単位でチームが分けられていた。ミッションとは、ビジネス上の目的のことで、たとえば「サービスの会員数を増やす」というものはミッションである。ミッションを与えられたチームは、そのミッションの達成のために、どんな機能が必要か、またはどのような誘導経路が最適か、それを実施するにはどうすればよいか、というより高次な目的レベルを考えた上での開発に着手することができるようになる。この事例はフィーチャーチームの発展形と考えることができる。

したがって、チームをフィーチャー単位で構成することで、反復におけるチームの成果物を顧客役が確認でき、価値ある単位の動作するソフトウェアになり、動いているソフトウェアを元にしたフィードバックを得ることができる。

5.7.3. 早期からのインテグレーション

サブシステムをいかに疎結合にしても、それぞれがまったく連係しないということは難しい。中大規模システム開発で恐ろしいのは、互いに想定で部分を作っていき、最後に結合してみて動かないという結末である(ビックバン結合)。ビックバン結合を防ぐためにインターフェースを当初から定義して公開しておき、テストを行うため仮のインターフェースのみを提供するスタブなどを利用して開発を進めることは一般的であるが、それだけでビックバンを防ぐのは難しい。そのため実際に動くシステムで連係しておく必要がある。しかしながら、相互インターフェースの部分の開発の進行がまちまちであるとうまく出来上がったシステム間での動作テストが実現できない。また既存の外部基盤を利用するような場合には、結合環境を整備する手間もかかる。

そのような場合、早期からサブシステム間連係テストを実施できるように、システム構築をしていくという事例が発見された。

たとえば、B 社や K 社の事例では、率先してサブシステム間でインタラクションが必要なバックログを優先的に選択して、早期よりインテグレーションを可能にした。

Copyright © 2012 IPA, All Rights Reserved

逆にH社の事例では、早期からインテグレーションを実現したかったが、他のチームはウォーターフォール型で開発しており、他チーム担当の機能の実装のタイミングとずれてしまった。また既存のシステム基盤との結合の準備できずに早期のインテグレーションが実施できなかった。そのため開発後半でその対応に追われたことが発見された。

したがって、サブシステム間のインテグレーションが必要な箇所は、各チームが早期から優先的に実現してインテグレーションを可能にしていくのがよい。

5.7.4. 同じリズム

中大規模開発においては、同時に複数のチームがそれぞれの担当の開発を行っていく。開発はスプリント、イテレーションなどの反復の単位で実施されていく。システム間連係が必要な場合は、それぞれのチームの成果を利用することで連係が可能になる。

しかしながら、各チームの反復のリズムや、利用されるインターフェースの開発がバラバラのままでいると、チーム間の相互連係が難しくなる。ウォーターフォール型開発のように「結合フェーズで繋げる」というような同期ポイントが少ない場合とは異なり、反復単位で機能を作り込んでいく以上、反復のリズムに合わせて各チームのサブシステム間連係が可能な形になっていることが望ましい。

このような場合、チーム間でこまめに調整をしてリズムを合わせていく事例が複数発見された。

たとえば、B社の事例では**早期のインテグレーション**を実現するために、こまめにチーム間で調整をしながらインテグレーションに必要なバックログの優先順位を決めていった。
G社の事例では、小さな要件はIRCで随時確認しつつ、大きな機能については、一週間単位でチーム間の適用タイミングを調整していた。

したがって、チーム間で連係するシステムの実装時期とその適用タイミングを、反復のリズムとして合わせるのがよい。

5.7.5. 継続的インテグレーション

アジャイル型開発においては、反復の度に動作するソフトウェアを実装し、変化に適応することが重要視される。そのため、常にこれまで実装してきたソフトウェア群が正しく動作しているかを確認していき、不具合があればすぐに修正しなければならない。しかしながら、毎回正しく動作しているかを手動で確認していくことはコストがかかる。

そのような場合は、インテグレーションプロセスや各種テストを自動化して、高頻度で実行していきながら、常にこれまで作りあげたソフトウェアが正しく動作しているかを継続的に確認、修正している事例が半数以上あった。

Copyright © 2012 IPA, All Rights Reserved

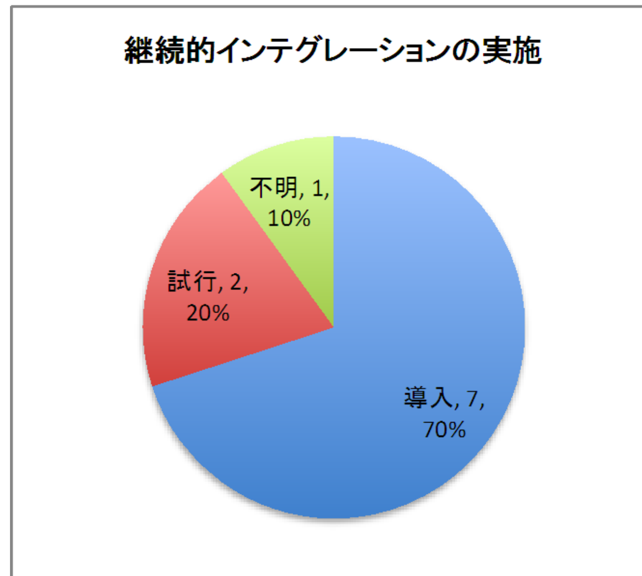

図 5-11 継続的インテグレーションの実施(中大規模のみ)

たとえば、A 社、B 社の事例では、現時点で自動化された単体テストは少ないが、現在、自動単体テスト項目を増やしながら、継続的インテグレーションツールを試行していた。
C 社の事例では、ゲームという性質上テストが自動化しづらい面はあるが、コミット時、日次という頻度で継続的インテグレーションを実施していた。
H 社や F 社の事例では共にテスト駆動開発を実践して、自動化された単体テストを用いて継続的インテグレーションが実施されていた。
G 社や I 社は、自動化された単体テストと、そうでない部分が混在しているが、自動化された部分については継続的インテグレーションを実施していた。

したがって、中大規模開発においては、できるだけテストを自動化しながら、継続的インテグレーションを実施するのがよい。

5.8. 品質についての工夫

5.8.1. 重視するビジネス価値

あるビジネス領域においては、何よりもサービス公開のスピードが重視されることがあれば、また別の領域においては「面白さ」が他の何よりも最優先されることもある。業務システムのような領域においては、何よりも「不具合を出さずに動き続ける」ことが優先される場合がある。

このような場合、システムが置かれるビジネス領域において、何が一番求められるのかによって、QCDS(品質、コスト、納期、スコープ)が変ってくる場合があった。

Copyright © 2012 IPA, All Rights Reserved

たとえば、A 社の事例では、「おもしろさ」を重視するが故に、スコープを変動させても、面白さを追求していた。しかしこの価値観は明示的ではなく暗黙的に組織の中に存在するものであった。また同じサービスの提供でも、Web サービスのように改修が後から効くものでは速度を重視するが、利用者の第一印象で、その後の利用が決まってしまうスマートフォンアプリにおいては、初期段階でそれなりの機能を含めておく必要がある、というように同じサービスの系列においても重視するポイントが変わり、それによって QCDS も変わらざるを得ない。

たとえば、ある事例においては、「市場投入時間の短縮」を最優先していた。そのため、品質にはそれほど高いものを求めないという選択を顧客と合意した上でしていた。ある事例によっては、速度を求めるため「80%ルール」(100%決定しなくても、80%決まっていれば先に進んでよいという独自ルール)で先に進めると明言しており、何が一番大事なのかを明確にしていた。

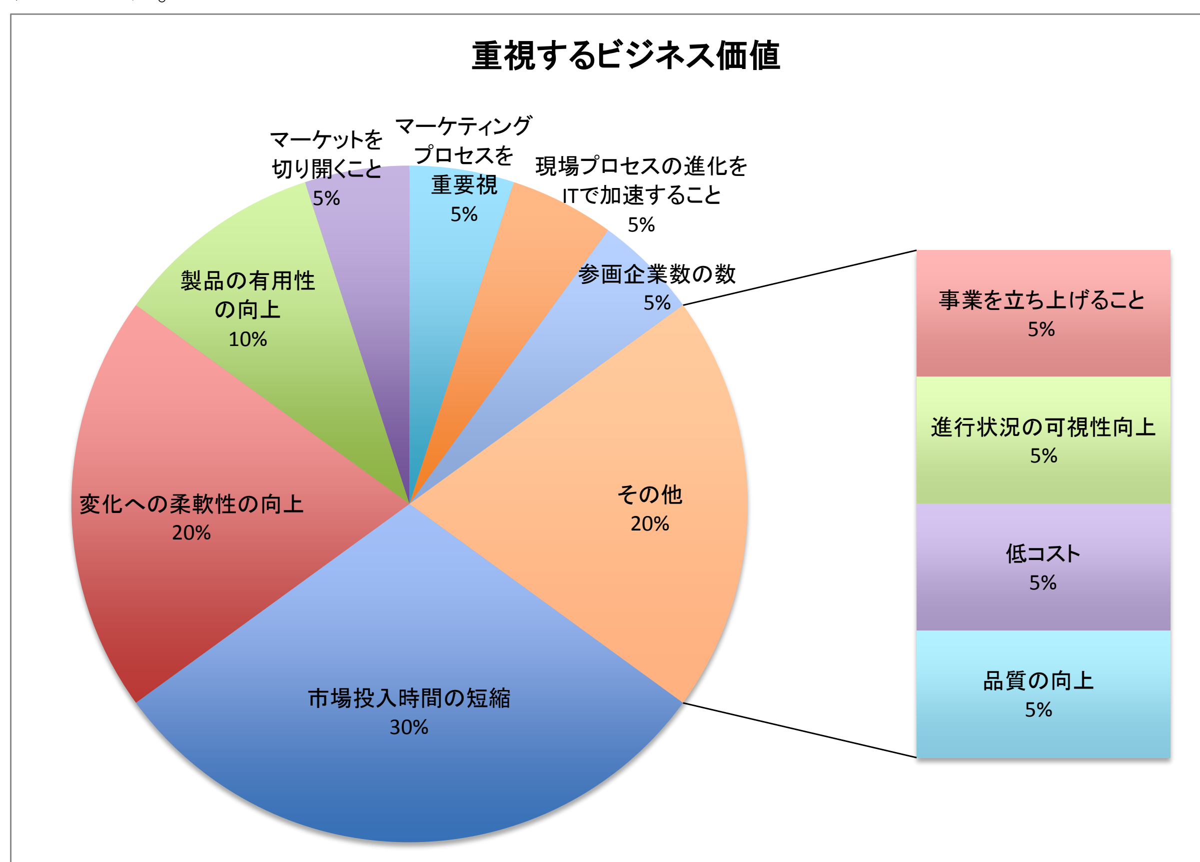

図 5-12 重視するビジネス価値

事例毎に重視するビジネス価値を上位３つ回答してもらった結果が図 5-12 である。市場投入時間の短縮、変化への柔軟性の向上、製品有用性の向上が上位３つとなるが、それぞれのビジネス領域によって重視する価値が異なることがよくわかる。

したがって、それぞれのビジネス領域において、重要となるビジネス価値を暗黙的、明示的どちらでも構わないが、開発メンバーとの間で共有しておき、そのビジネス領域に最適なバランスでソフトウェアを提供するのがよい。

Copyright © 2012 IPA, All Rights Reserved

5.8.2. ビジネス価値の変化

ビジネス領域によっては、市場の変化が激しく、プロジェクトを開始した時点では、何が重要かをあらかじめ決めることができない場合がある。たとえば、対抗サービスが発表され、そのとき提供していたビジネス価値が陳腐になってしまい、別のビジネス価値に移ることもある。また、自らが提供したサービスで、利用者の価値観が変わり、新たなビジネス価値が創発されるもある。これらのビジネス価値は、あらかじめ予期できない。
そのビジネスにおける価値観が異なると、提供する品質の評価尺度そのものも変わる。自ずと、品質、コスト、納期のバランスも変わってくる。

その場合、あらかじめ価値が変化することを前提にし、さらに変化を起こすことを促進する。たとえば、変化を促進するために、ビジネス価値や品質を「おもしろさ」のように、あえて曖昧な表現を用い、具体的に明示せずに、その提供者の即興的なセンスを発揮できるようにする。その結果、品質、コスト、納期、スコープのバランスも、ダイナミックに変化させる。

たとえば、A 社、B 社、C 社、のように「おもしろさ」を重視するビジネス領域では、そのおもしろさは事前に予測も定義もすることが難しく、ひたすら動くソフトウェアへの関りの中や、利用者のフィードバックによって生み出されるという報告があった。
I 社の事例では、ビジネス価値を変化そのもの捉えて、お客様の変化と意思決定に追従できることを最優先にしていた。

したがって、予測できないビジネス価値を扱うためには、変化を前提として対応できるようにしておき、それによって QCDS も変化させるのがよい。

5.8.3. タイムボックス優先の品質

スクラム開発ではスプリント、エクストリームプログラミング(XP)ではイテレーションという反復の単位がある。これらは単純に繰り返すというだけでなく、時間を固定して、その中で完了させる、という意味を持つ。反復の期間内では、その期間内で成果物を実現し、反復終了後に顧客役にレビューしてもらい、フィードバックを得る。

しかしながら、開発者には、期間内で実現するということを優先することができず、期間内では実現できない、よりよい品質を求めようとして作業を進めてしまい、期間内に成果を出せなくなることがある。そうなると反復毎のフィードバックを得ることができずに、アジャイル型開発の利点を享受することができなくなる。

たとえば、ある事例では、設計や実装をタイムボックスで実現していたが、それぞれの担当者から「まだできていない」との理由でレビューを断わられてしまうことがあった。事業責任者は「終ってなくてもいいからレビューをする」とのルールを設けることで、「期間内にできていません」ではなく「期間内で何ができたのか」を確認することにした。

Copyright © 2012 IPA, All Rights Reserved

また別の事例では、「すべての障害が Fix できていないので、システムテストができません」との担当の報告を受けたプロジェクトマネージャが「完璧を目指すより、完了させろ」というキャッチフレーズで、区切りを付けて先に進めていくことを促した。

したがって、致命的な状況でない限りは、反復期間が終わった時点で、その反復の成果として開発を一区切りつけレビューできるようにするのがよい。そこで出たフィードバックを次に生かして、その状況にあった品質によるソフトウェアを提供してフィードバックを貰いリズムを作り出すことができる。この決定には、製品やサービスについての意思決定者による判断が必須である。

ただし、あまりにタイムボックス内での品質を犠牲にしてソフトウェアの提供を優先しすぎると、相対的に品質を低下させ、長期的には技術的負債（障害や可読性の低いコードが増えて変更や改修の妨げになってしまうこと）になる事例も報告された。

5.8.4. 適切なピア・レビューの選択

アジャイル型開発では、フィードバックを受けるタイミングが、ウォーターフォール型開発に比べ多い傾向にある。特にエクストリームプログラミング(XP)のペア・プログラミング（二人で同席しながら、ひとつのプログラミングなどの作業を行う）などのピア・レビューのプラクティスが繰り込まれている。

ピア・レビューを行っていない事例では、導入後に品質が格段にあがる。要求されている品質に応じて、インスペクションやウォークスルーなど適切なピア・レビューを選択し、導入する。

たとえば、A 社の事例では、開発者が毎月追加されてくるようになり、よい品質のコードを書けない人が増えてきた。そのため途中からピア・レビューを実施して、設計やコードのレビューを行って品質を向上させた。
B 社の事例では、設計やコードのレビューや、ペア・プログラミングを実施して、品質向上に務めていた。
C 社の事例では、定期的にコードレビューを行っていた。
F 社の事例では、毎週のレビューによって、障害をコントロールしていた。
H 社の事例では、特に難易度が高い部分においてペア・プログラミングを実施して品質向上に寄与していた。
G 社の事例では、ペア・プログラミングを実施していたが、ピア・レビューの側面もありながら、むしろ知識の共有としての用途で実施していた。
J 社の事例では、アーキテクチャチームによるコードレビューが実施されていた。

したがって、中大規模の開発においては、様々な局面においてのピア・レビューを実施して品質の向上を実施するのがよい。

Copyright © 2012 IPA, All Rights Reserved

5.8.5. 自動化された単体テスト

アジャイル型開発において、自動化された単体テストのセットは、デグレードを防ぐ品質のセーフティーネットであり、動くソフトウェアが成長している証しでもある。単純に自動化されたテストスイートだけでなく、自動テストと共に製品コードの開発が進められていくテスト駆動開発(TDD)を実施することで、自動化された単体テストの数は加速する。自動化されたテストセットが存在することで、はじめて**継続的インテグレーション**の効果が発揮できる。
中大規模になるほど、テストするソフトウェア対象が増えるため、自動化の恩恵を受けないと、すべてをテストすることは難しい。しかしながら、ゲームのようなビジネス領域によっては面白さを自動化された単体テストでは書きづらいとの報告もあった。

このような場合、まず自動化しやすい所は単体テストを自動化しておくのがよい。図 5-13 は自動化単体テストの中大規模事例での整備状況を示したものである。半数が「あり」となっている。TDD などは行っていないが、自動化された単体テストを部分的にでも整備しているという事例が半数を占めていた。

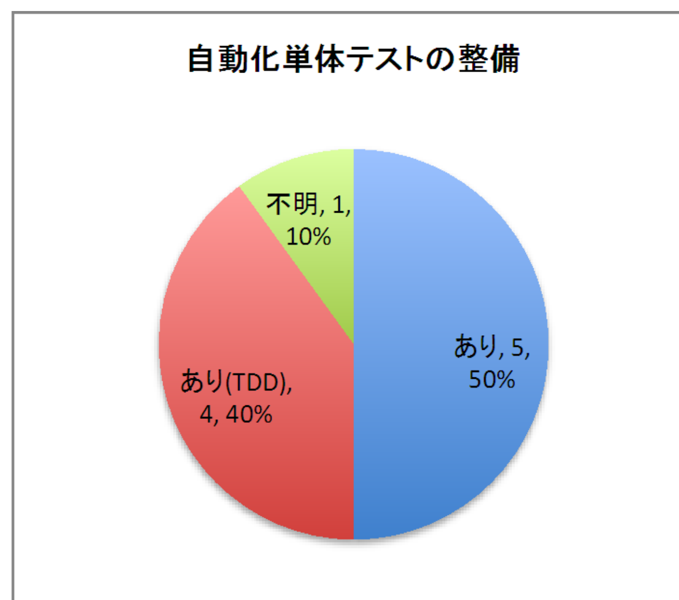

図 5-13 自動化単体テストの整備(中大規模のみ)

たとえば、B 社の事例では、元々がレガシーコード(自動化された単体テストが整備されていないソフトウェア)のため、少しずつ自動化に取り組んでいるということであった。
C 社では自動化された単体テストを導入してはいるが、まだ整備が不充分であり、ゲームという特性上テストを書きづらいという報告があった。
F 社や H 社の事例では、当初から TDD を実践していて、自然と自動化されたテストが整備された。
G 社の事例では、10 年前から自動化単体テストを導入しようと試みていたが、なかなか浸透しなかったという報告があった。最近ではやっと浸透してきたが TDD までは実践していないということであった。

Copyright © 2012 IPA, All Rights Reserved

I 社の事例では、単体テストの自動化は実施したが、TDD は「行うことが目的になりがち」、「スキルが必要」という理由から、あえて選択をしないという報告があった。

したがって、中大規模開発においては、可能なところから自動化された単体テストを整備し、可能であればテスト駆動開発を行って自然と自動化されたテスト群を成長させていくのがよい。

5.8.6. テストフェーズ

アジャイル型開発においては、各反復（スプリント、イテレーションとも言う）の中で、自動単体テストや受け入れテストも実施していくことになる。しかしながら、反復の繰り返しによって、複合的な組合せが増えて複雑さが増す。そのため、プロジェクト終盤に近づくと、通常の反復内では発見できなかった問題が発見されることがある。また、反復内で消化できなかった障害を残しておいたがために、後になって新たな障害を引き起すこともある。

このような場合には、反復漸進的な開発の後に、全体としてテストフェーズを設ける事例が複数あった。

たとえば、A 社、B 社、C 社、H 社、J 社、I 社の事例では、反復とは別に品質テストフェーズを設けていた。I 社においては、品質部隊が別にいて、反復とは別のサイクルでテストフェーズを実施していた。

したがって、中大規模開発においては、品質を更に高めたいならば、ウォーターフォール型開発のようにテストフェーズを設けるのがよい。反復の中でテストのみ実施する、あるいは反復自体を終了してテストフェーズに移行する。テストフェーズでは、利用者の視点に立ち、探索テストや高負荷テスト、操作性などを行う。

5.8.7. 第三者テスト

アジャイル型開発では、設計、開発、テストなどの様々な役割をこなせる多能工を進める。プロジェクト開始当初から、関係する法務知識やビジネス動向などのノウハウを共有している。
しかしながら、当初から提供していた機能の稼働確認をするリグレッションテストが自動化さていない場合や、携帯端末の機種ごとの確認など工数がかかるテストをチーム内で実施することは困難である場合がある。また、セキュリティや法務などの専門家チームによって、品質が担保されているかを確認が要求されているプロジェクトもある。

このような場合は、この種のテストを第三者に依頼する事例があった。

たとえば、A 社では外部パートナー企業にリグレッションテストを依頼していた。
B 社、C 社の事例では、外部パートナー企業に QA テスト(品質保証テスト)を依頼していた。
E 社の事例では、企業内のセキュリティや法務部門の監査を受ける必要があった。

Copyright © 2012 IPA, All Rights Reserved

したがって、開発チームだけでは品質が担保できない場合には、第三者テストを依頼するのがよい。

5.9. 部分適用についての工夫

部分適用とは、プロジェクトの一部のみを非ウォーターフォール型開発とすることで、複合的な手法を用いることである。

5.9.1. 必要な部分のみ適用

中大規模開発において、ひとつのプロジェクトで扱うドメインやシステムは複数に渡っている。その中でもすべての領域に同じ手法を用いる方針もある。しかしながら、領域によってシステムの特性が異なる事例が多くある。

たとえば、C 社は、ソフトウェア開発チームとデザインチームがいる。ソフトウェア開発チームはシステムの出来上がりをレビューする関係でスクラムを用いている。しかし、デザインチームは、デザイン部品を大量生産するためパイプラインのような開発プロセスを用いていた。

たとえば、D 社は、基幹システムと Web 技術を用いたフロントエンドシステムがある。ユーザビリティなどのフィードバックを得たいため、フロントエンドシステムを非ウォーターフォール型開発とした。
たとえば、H 社は、顧客データを扱うシステムと、Web コンテンツを検索するシステムがあった。ユーザビリティを確認しながら作りたいため、Web コンテンツを検索するシステムをアジャイル型開発とした。
たとえば、I 社は、業務アプリケーションとインフラストラクチャのサブシステムを同時に作っていたので、それぞれチームに分離した。業務アプリケーションは、業務要件のスコープや詳細が明確にならず、反復的に開発を進めていた。インフラチームは、あらかじめ要件が決まっていたためウォーターフォール型開発とした。サブシステムはスタブ(インタフェース利用のための仮実装)を提供し、インターフェースを早期から扱うことができるように配慮した。

このように、中大規模開発においては、すべての領域に非ウォーターフォール型開発を導入するのではなく、繰り返しフィードバックが適切な領域のみに非ウォーターフォール型開発を導入することをまず検討する。

5.9.2. 疎結合なチーム

中大規模開発において、プロジェクトが大きくなった場合、複数のチームに分割する。しかしながら、一部のチームが非ウォーターフォール型開発を行い、別のチームがウォーターフ

Copyright © 2012 IPA, All Rights Reserved

ォール型開発を行っていると、タイミングの不調和発生、スコープや仕様理解に対するチーム間認識違いが心配される。

たとえば、A 社は部分適用ではないが、同じシステム上ではあるが、チーム分割の時に、機能ごとにチームを分割した。その機能は、独立して開発およびリリースできるようになっている。機能が異なるが、同じプラットフォームを使っている。
たとえば、H 社は、機能ごとにチームを分割しているため、ソースコードやリリースの衝突がない。ただし、インテグレーション時に画面デザインなどの不一致が発生していた。
たとえば、I 社は、業務アプリケーションとインフラストラクチャのシステムを作るためにそれぞれのチームがあった。業務アプリケーションチームは反復的に開発を進め、インフラチームはウォーターフォール型開発とした。インフラチームは、スタブを提供し、インターフェースがとれるように配慮した。

したがって、チームによって異なった手法の適用があるときは、できる限り明確にチームを分離する。その間のインターフェースや共通項目について、わかりやすく間違いがないようにする。チーム間の連携のタイミングや方法についての合意も必要である。

5.9.3. 工程の見える化

アジャイル型開発においては、タスクボードなどを用いて「見える化」するプラクティスが奨励されている。タスクボード上では、タスクの TODO(未着手)、DOING(着手)、DONE(完了)の三段階状態を壁面やホワイトボードに貼り出してチームがそれを使って作業の状況を反映し共有する。しかしながら、ウォーターフォール型開発の進捗は、表計算ソフトでの WBS や、ガントチャートなど別の方法で管理されていることも多い。

そこで、C 社では、デザインを行っているチームに、デザイン工程を聞き出して、暗黙知であったプロセスを表出化させるとともに、その工程をタスクボードのように壁に見える化をして状況を把握できるようにした。最初はデザインチームの代りに推進者がそのボードをメンテナンスして見える化していたが、徐々にデザインチーム自身がそのボードを使うようになっていった。

このように、ウォーターフォール型と非ウォーターフォール型のように開発手法が異なっていても、「見える化」することで、チーム間で状況を共有できるようになった。

Copyright © 2012 IPA, All Rights Reserved

5.10. 組織文化の傾向

5.10.1. 組織文化調査について

調査方法でも触れたように、今回の調査では、組織文化を評価する手法として「組織文化診断ツール(OCAI)」を使用した。本ツールは、世界的に広く用いられている組織文化の評価方法で、二つの軸によって分けられた、四つの象限(家族文化、イノベーション文化、官僚文化、マーケット文化)によって組織文化を表現する。(図 5-14)これを競合価値観フレームワーク(Competing Values Framework)と呼び、OCAI はこの競合価値観フレームワークを用いて組織文化の診断を行う。OCAI は組織文化を明確に４つに分類するのではなく、組織内で何を重視しているか、どの傾向が強いのかを度合いによって表現するのが特徴である。

OCAI で表現する組織文化は２つの軸からなる４種類となっている。それぞれの軸と特徴を簡単に解説する。

5.10.1.1. ２つの軸

- **内部志向/統合－外部志向/分化**
  - 内部に目を向けてまとまるのか、外部に目を向けて細分化していくか。
- **安定とコントロールー柔軟性と裁量性**
  - 安定やコントロールを志向するのか、柔軟性や裁量を志向するのか。

5.10.1.2. ４つの特徴

A) 家族文化

：組織の柔軟性や人々への気遣い、顧客への心遣いを伴う組織内部の維持を重視する組織

B) イノベーション文化

：高い柔軟性と革新性が特徴的な、組織外でのポジショニングを重視する組織

C) マーケット文化

：安定性と統制の必要性を強く認める、組織外のポジショニングを重視する組織

D) 官僚文化

：安定性と統制の必要性を強く認める、組織内部の維持を重視する組織

Copyright © 2012 IPA, All Rights Reserved

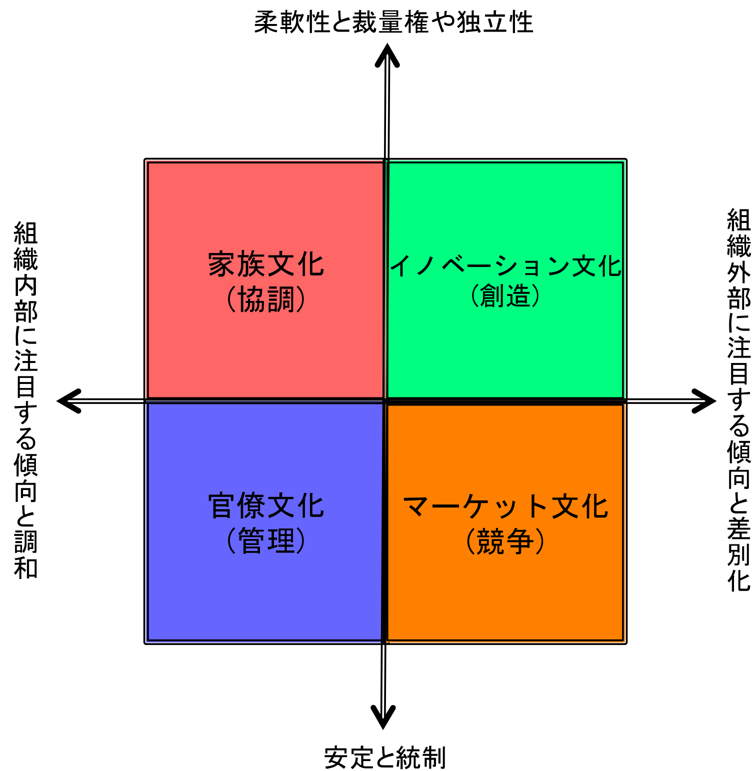

図 5-14 競合価値観フレームワーク

5.10.2. OCAI の二つの軸と四つの象限

質問は OCAI にもとづき、６つのカテゴリについて、それぞれ４つの質問項目(A:家族文化,B:イノベーション文化,C:マーケット文化,D:官僚文化)を用意した。それぞれのプロジェクト/組織の説明として近いものの順から、100 ポイントを４つの項目に按分してもらった。

集計では回答の A～D 毎に 6 つのカテゴリの得点の平均値をとることで、ある特徴についての、組織文化の点数を算出した。各点数をレーダーチャートにマッピングすることで、「官僚文化の強い組織文化」や「家族文化の協調文化の度合いが高い」という見方ができる。

本来の OCAI の使い方は「現状の組織文化」と「望ましい組織文化」という聞きかたをする。今回の調査では「直前に携わっていたウォーターフォール型開発プロジェクト」と「現在の非ウォーターフォール型開発プロジェクト」という聞きかたをした。

Copyright © 2012 IPA, All Rights Reserved

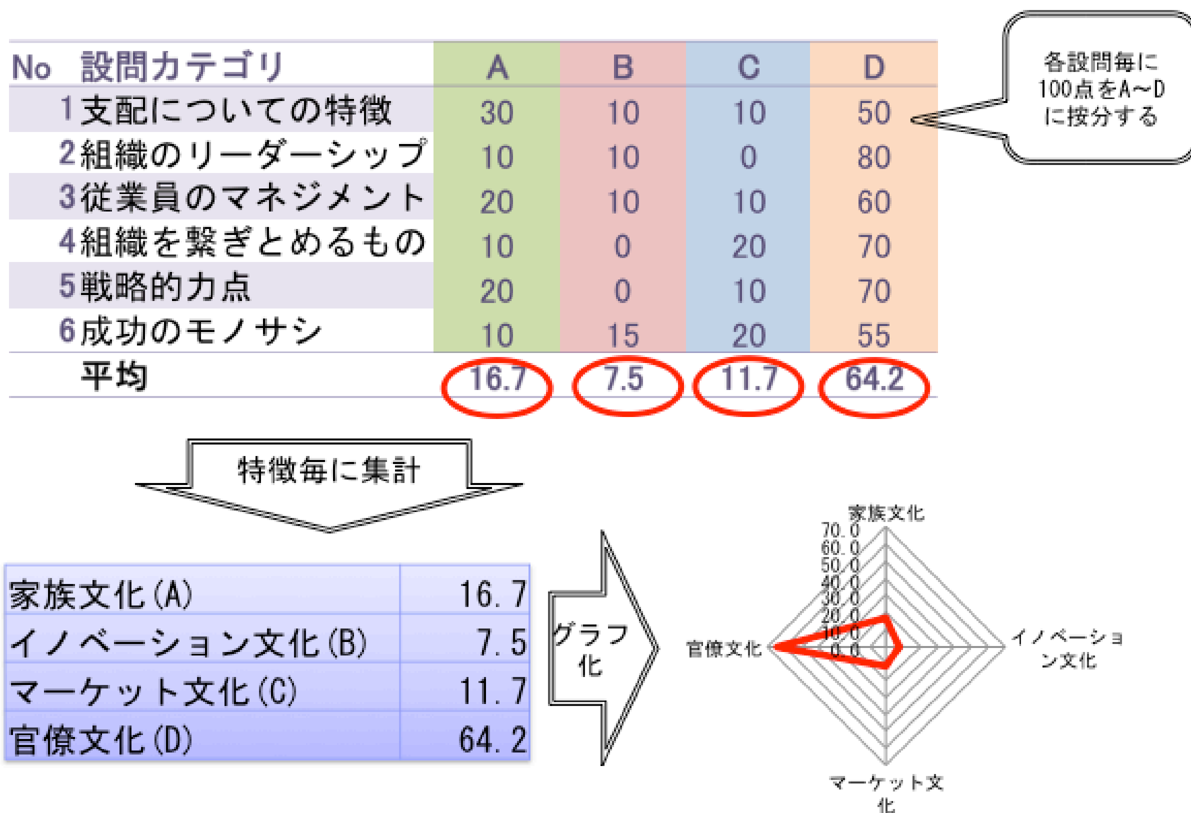

図 5-15 組織文化調査集計の流れ

６つのカテゴリについては以下の通りである。

1. 支配についての特徴
2. 組織のリーダーシップについて
3. 従業員のマネジメント
4. 組織を繋ぎとめるもの
5. 戦略的力点
6. 成功のモノサシ

この手法を用いて、ウォーターフォール型と非ウォーターフォール型の組織文化を明らかにした後、どのような文化の変遷が起きたかを分析した。また文化の変遷が発生していた場合には、同じ方向性の文化の変遷の際に生じた問題、工夫などの共通点を見出すことを狙った。

調査方法で述べたように、今回は事例毎に、プロジェクトメンバーに対してできるだけ多くの回答をお願いして個人調査表の回答を依頼した。結果、全体としては 12 社中 7 社からの回答があり、そのうち有効回答は、ウォーターフォール型(WF)で 53 件、非ウォーターフォール型(NWF)で 81 件の回答があった。最初に全体の傾向を、次に個別事例毎の結果を解説し、最後に考察を述べる。

Copyright © 2012 IPA, All Rights Reserved

5.10.3. 調査結果(全体)

全体としての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組織文化の調査結果の違いを述べる。図 5-16 の結果として、赤い線と、緑の線に注目する。赤線がウォーターフォール型開発の文化で、緑線が非ウォーターフォール型開発の文化結果である。
まず比較して目に着くのは、赤線のウォーターフォール型開発の傾向は、他の特徴に比べて管理志向が強いという点である。ついで競争志向もやや強いが、一方で協調や創造の領域が低くなっている。他方非ウォーターフォール型開発のラインの傾向としては、管理の代りに協調と創造が高くなっている(創造は競争と同じレベル)。
単純に比較してみると、ウォーターフォール型開発の方が管理志向、競争志向が突出しているだけで、非ウォーターフォール型開発の方がバランスの取れた形となっている。この結果により本調査ではウォーターフォール型開発と比べて非ウォーターフォール型開発は「管理よりも協調と創造を重視する」組織文化的特徴を持っていることがわかった。

	ウォーターフォール	*非ウォーターフォール*
家族文化	18.55	27.49
イノベーション文化	12.89	26.30
マーケット文化	28.71	27.40
官僚文化	39.87	18.04

表 5-1 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組織分化比較(全体集計)

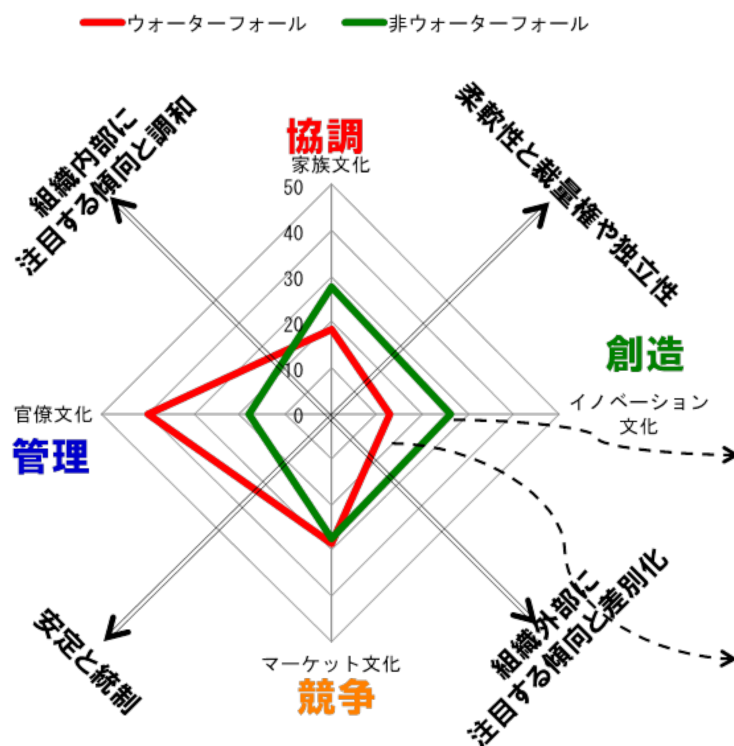

図 **5-16** 組織文化傾向(全体)

Copyright © 2012 IPA, All Rights Reserved

5.10.4. 組織文化結果(個別)

次に個別事例毎の結果を解説する。回答の回収率にばらつきがあるため、回答率が高かった３事例(ウォーターフォール型開発も非ウォーターフォール型開発も回答があり、かつ5件以上の回答)個別の結果を解説する。

事例	人数	家族文化	イノベーション文化	マーケット文化	官僚文化
B	1	31.7	11.7	20.0	36.7
C	7	19.4	19.6	31.7	29.3
G	32	18.3	12.7	27.6	41.5
F	10	17.2	9.4	31.1	42.3
K	1	11.7	15.0	40.0	33.3
H	2	23.8	10.0	23.3	42.9

表 **5-2** ウォーターフォール型プロジェクトの組織文化結果

事例	人数	家族文化	イノベーション文化	マーケット文化	官僚文化
A	7	29.6	31.3	24.9	14.2
B	11	21.9	22.8	36.6	12.7
C	7	26.3	19.2	28.7	25.8
G	34	28.7	29.8	24.7	16.8
F	15	26.2	21.0	28.6	24.3
K	4	33.1	32.7	24.0	10.4
H	3	31.1	22.2	25.3	21.4

表 **5-3** 非ウォーターフォール型プロジェクトの組織文化結果

Copyright © 2012 IPA, All Rights Reserved

5.10.4.1.　　C 社の結果

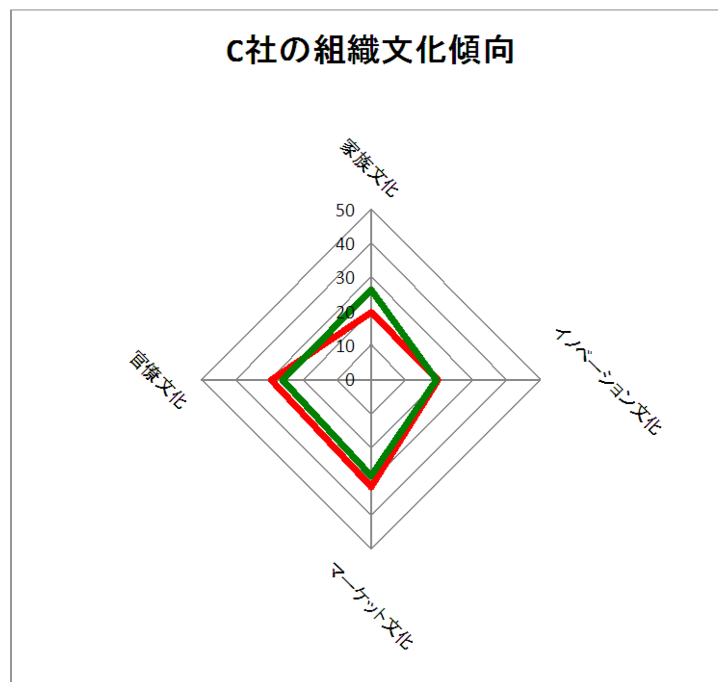

図 5-17 C 社の組織文化の傾向

◆ **解説**

図 5-17 は C 社の組織文化の傾向グラフである。C 社は元々ウォーターフォール型開発で実現している管理コストが高いプロジェクトを、スクラムを使うことによって改善した。またプロジェクト全体に一気に展開するのではなく、漸進的に部分的に展開してきた。まず特徴的なのは、ウォーターフォール型開発(赤線)での文化の特徴は全体として官僚、マーケットが高めで、家族、イノベーションはそれよりやや低い状態であった。一方、スクラム(緑線)の場合においては、官僚、マーケットが微減で、家族が増えている。全体として大きな変化ではなく、家族文化(協調型)の要素が増加してきた傾向が見受けられる。大きな変化が見られないのは、展開スピードをゆるやかにしていたために、大きな変化を起していないためかもしれない。

(回答 ウォーターフォール型開発/非ウォーターフォール型開発 7 件)

◆ **組織文化の問題**

C 社での組織文化的な問題としては、「見える化をやりすぎてストレスを感じる人が現われた」、「スプリントで疾走する連続なので疲れる」というものがあった。

Copyright © 2012 IPA, All Rights Reserved

5.10.4.2. F社の結果

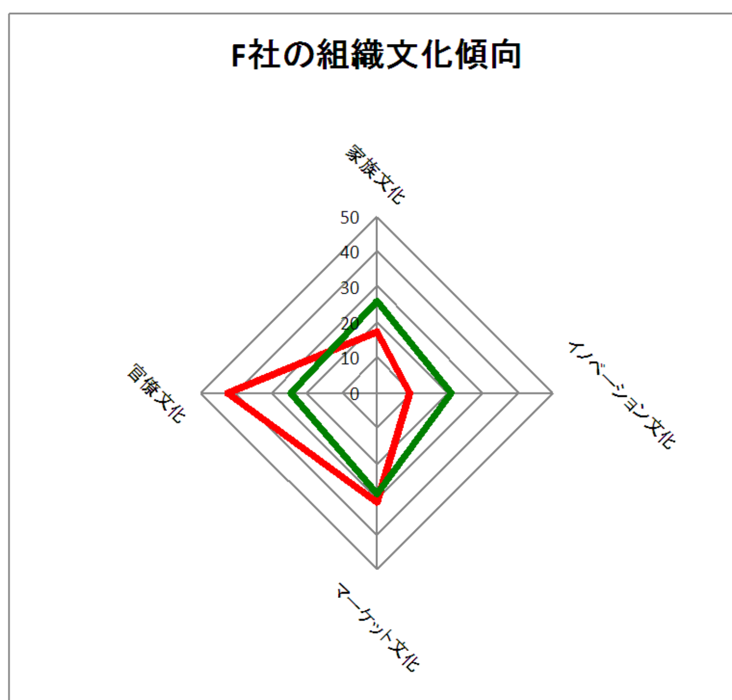

図 5-18 F 社の組織文化傾向

◆ 解説

図 5-18 は F 社の組織文化の傾向グラフである。F 社は以前ウォーターフォール型開発にて進行していたが企画の変化に対応できずに失敗したのがきっかけで、組織的にスクラムの導入に踏み切った。そのためこの結果は、F 社内の以前のウォーターフォール型開発と今のスクラムの組織文化の違いを表している。ウォーターフォール型開発(赤線)が大きく官僚文化に振れているのが特徴的である。一方、スクラム(緑線)では、官僚文化が激減し、変りに家族文化とイノベーション文化が大きく増えている。極端に管理志向だった組織文化が、協調・創造志向に傾いてバランスがとれてきた傾向になっているのが見受けられる。
(回答 ウォーターフォール型開発 10 件、非ウォーターフォール型開発 15 件)

◆ 組織文化的問題

F 社において、組織文化的問題として挙げられた例としては 「ウォーターフォール型開発が抜け切れていなくて企画からウォーターフォール型開発でやってくれと頼まれる」や「開発の力が強すぎる(企画が意見を出しづらい)」あるいは「企画側が強い(開発が意見を出しづらい)」というものがあった。

Copyright © 2012 IPA, All Rights Reserved

5.10.4.3. G社の結果

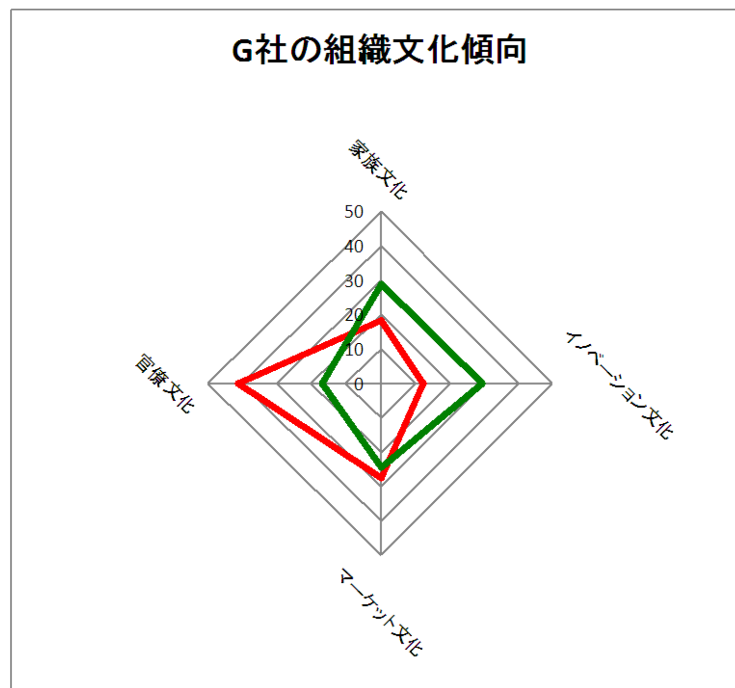

図 5-19 G 社の組織文化傾向

◆ 解説

図 5-19 は G 社の組織文化の傾向グラフである。G 社は短い納期に対応するために、自律的なチームを目指しながら、開発においては規律を守り XP をベースに 10 年以上に渡って非ウォーターフォール型開発を適用してきた。ウォーターフォール型開発(赤線)のグラフでは、やはり官僚文化が突出して高い。一方非ウォーターフォール型開発(緑線)の結果を見てみると、官僚文化が激減し、家族文化やイノベーション文化が大きく増えている。特にイノベーション文化の増加が顕著である。極端な管理志向から、全体的に協調・創造志向に移行してきているように見受けられる。

(回答 ウォーターフォール型開発 32 件、非ウォーターフォール型開発 34 件)

◆ 組織文化的問題

G 社での組織文化的問題として挙げられたのが、「合わない人が出てくるので入れ替えが激しい」というものであった。組織体制の工夫でも述べられているが、「能動的に動けない」、「技術はあるがスピード感についていけない」、「最新技術を駆使していてついていけない」といった点がその理由であった。

5.10.5. 考察

元々、アジャイル型開発は管理(コントロール)よりも協調(コラボレーション)という特徴があったため、官僚文化→家族文化への移行は想定していた。しかし予想以上にイノベーション文化の増加が見られたのは、調査事例でサービスプロバイダーが半数以上を占めるのがその理由であると推測する。

Copyright © 2012 IPA, All Rights Reserved

また組織文化的移行とそこで起きる問題としては、元の組織文化が官僚文化である場合に、家族文化・イノベーション文化への振れ幅が広いほど、F 社、G 社のような問題が発生することが予想される。

今回の調査としては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という違いに絞ってしまったが、導入前と導入後という組織文化比較という形にすることで、今回は「ウォーターフォール未経験」と回答しなかった被験者に対しても調査をすることができることがわかった。

非ウォーターフォール型開発の中でもアジャイル型開発は、単なる開発手法やプロセスと捉えられがちだが、もっと根底にある文化的な側面や価値観が大きく異なっていることをまず導入前に理解しなければいけない。
今回の事例でもあるように、導入前の組織文化の違いが、導入後に引き起す問題の差を生み出す可能性が高い。今回の調査結果では、対象が少ないため相関関係までは引出せていないが、組織文化とアジャイル型開発導入後に発生する問題を更に調べていくことで、よりスムーズな導入に寄与することが期待できる。

Copyright © 2012 IPA, All Rights Reserved

5.11. 従事者の精神健康度

5.11.1. 調査方針について

非ウォーターフォール型開発では、いきいきしていることを目指している。現在、社会的に精神的な失調やうつ状態が多い。精神的な健康状態が損なわれると休暇取得によって就業時間が減り、疲労度が高いと業務に対するパフォーマンスが落ちることは想像に難くない。しかしながら、各職場環境における従事者の健康度は調査されていない。従事者の健康度を調査することによって、非ウォーターフォール型開発の特性を見いだす。

5.11.2. 調査方法

今回は、被調査者（アンケート回答者）の男女、ビジネス構造モデルによる立場を調査した後、非ウォーターフォール型開発・アジャイル型開発かウォーターフォール型開発に加え、プロジェクトそのものの成功度および満足度を個人の主観的な観点から調査した。
さらに非ウォーターフォール型開発におけるプロジェクトの属性として、以下の３つの観点から非ウォーターフォール型開発の適用分野別マップを試作している。

1. 変化志向が大きいか安定志向が大きいかによって分類する不確実性
2. 早期に価値を実現することを段階的に目指す傾向が強いか、それとも価値の実現速度以上に安全かつ安心な価値の実現を目指す傾向が強いかによって分類する高速順応性
3. 制約条件などによる複雑性

また、それぞれ以下の２つを調査した。

1. 不確実性の調査のためにプロジェクトの規模（人月）
2. 高速順応性の調査のためにプロジェクトのリリース密度（リリース回数／プロジェクト期間）

複雑性の調査のチームの大きさ（人）は、調査対象ごとに確認した。また、要件の変化度合いと、それにプロジェクトの進め方に対する改善頻度を調査した。
いきいきしていることを調査するためにネガティブな側面である「ストレス度（どれだけ精神的、社会的、心身的疲労を感じているか）」と、ポジティブな側面「生きがい度（QoL: Quality of Life, どれだけ生活の質が良いか）」のふたつの側面と、社会的関係性（場）の側面を調査した[8]。
今回の対象プロジェクトに属し、調査主旨に合意し、協力いただける個人に対してアンケート法で調査した。アンケートは、非ウォーターフォール型開発・アジャイル型開発もしくはウォーターフォール型開発について経験したプロジェクトがある場合に記入を依頼した。

Copyright © 2012 IPA, All Rights Reserved

5.11.3. 調査結果と考察

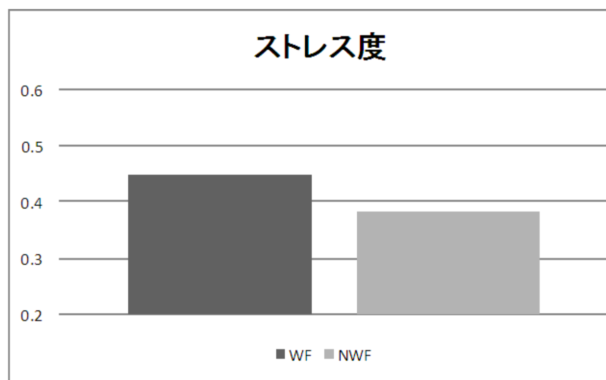

図 **5-20** ストレス度

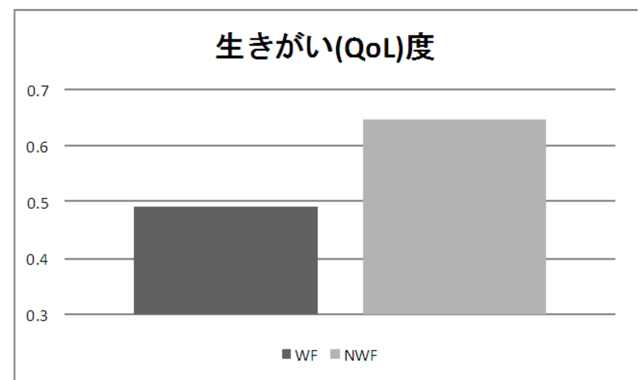

図 **5-21** 生きがい**(QoL)**度

今回、ウォーターフォール型開発に有効回答 54 件および非ウォーターフォール型開発・アジャイル型開発(以下、NWF)に有効回答 81 件を得た。ストレス度と QoL(生きがい)度を比較した。その結果、ストレス度の平均は WF 0.45 (±0.03)および NWF 0.38 (±0.02)で、WF におけるストレス度が高かった（図 5-22）。QoL 度の平均は、WF 0.49 (±0.04)および NWF 0.64 (±0.03)で、明らかに NWF における生きがい(QoL)が高かった（図 5-23）。
つまり、非ウォーターフォール型開発・アジャイル型開発の現場は、生きがい(QoL)度が高く、ストレス度が低いことがわかった。

Copyright © 2012 IPA, All Rights Reserved

6. 調査ポイント以外で発見された課題

アジャイル型開発を行っている事例のなかで、事前の調査ポイント以外で発見され、課題と認識されたものを紹介する。
(箇条書き行頭記号の●は課題、○は課題であったが解決策・回避方法が実施されているもの)

6.1. 全体計画の把握困難

ビジネスの特性によっては、期日までに期待する機能の実現が望まれている。たとえば、法令対応や広告、契約などの状況によっては、サービスの提供や改修のスコープと期日が決められている場合はある。その中でもある程度のプライオリティが決められているものの、全体的な見通しが望まれている。
しかしながら、アジャイル型開発では、要求の変化や開発状況に応じて、着手する順番や範囲を決めるため、プロジェクト開始時に全体計画の把握が困難であるとの報告された事例があった。

- ● D 社においては、フィードバックを受けて修正を続けて行きながら、全体のスコープを決めざるを得なかった。
- ○ I 社では、全体計画を把握することは困難であったが、ホワイトボードで全体像のラフスケッチを描くことで、ラフであっても関係者が納得した。

このように、計画の全体を把握することが困難である事例があるため、この課題を認識し、理解することが必要である。

6.2. 新手法への期待過大

アジャイル型開発について、様々なイベントや研修制度、認定制度がある。アジャイルコーチやコンサルティングなどの支援もある。その中で、アジャイル型開発を導入・展開し、開発を続けている上での注意点や課題点などについて指摘する場合もある。

しかしながら、推進者や支援者によっては、問題点や課題に注意せず、利点のみに着目していたと反省している事例があった。既存のやり方やビジネスの形態などに着目せず、むやみにアジャイル型開発を導入すると特に問題が発生する。

Copyright © 2012 IPA, All Rights Reserved

- ある事例では、開発現場において精査することなく、アジャイルコーチが示した他社の事例をそのまま適用してしまい、現場にマッチしないという問題が発生した。
- 別の事例では、社内教育などで非ウォーターフォール型手法の利点しか話さないことで現場に不信感を持たれたという事例があった。

このように、ビジネス状況や開発スタイルを明確にして、今ある有用な工夫と、問題点を見極めつつ、手法を選択、カスタマイズする必要がある。

6.3. 手法移行の問題

別の手法へ移行する際には、文化的な衝突が発生することが多い。ウォーターフォール型開発は、比較的「管理」を尊重する官僚的組織であることが多く、逆に、アジャイル型開発は協調を尊重する「家族文化」、創造を尊重する「イノベーション文化」の文化を持っている。この文化の移行がともなっていない。
また、アジャイル型開発などの移行先についての知識や経験が不足し、適正な運営が難しかった。特に、移行前の手法に戻そうとする意見や行動が見られる。

- E 社において、導入当初は、開発者がアジャイル型開発への移行を企画段階で拒否した。ビジネス企画者はアジャイル型開発の思想や理解が不充分であったため、慣れたウォーターフォール型開発に戻そうとした。
- F 社において、ビジネス企画側（プロダクトオーナー）が、教育コースの受講や認定を取得しているにも関わらず、アジャイル型開発のプロジェクト進行に戸惑うことが多かった。

このように、新手法に移行するときは、ビジネスの特性や構造を調査し、開発現場の文化を測定しながら、経験者の配置や、漸進的な展開など回避策を検討するとともに注意が必要である。

6.4. ビジネス企画側にボトルネック発生

非ウォーターフォール型開発・アジャイル型開発では、ビジネス企画と開発の役割が必要である。ビジネス企画と開発の役割を分け協調しながらプロジェクトを推進する方法と、ビジネス企画と開発の役割を分化せずチームが一体となってプロジェクトを推進する方法がある。スクラムのようにプロダクトオーナー（ビジネス企画）と開発者をわけたところ、プロダクトオーナーがボトルネックになる事例が多かった。

Copyright © 2012 IPA, All Rights Reserved

- C 社では、プロジェクト推進が滞ったことはないもののプロダクトオーナー（ビジネス企画）に作業が集中し、休暇を取るとプロジェクト全体が停止するリスクを認識していた。
- E 社では、開発者がプロダクトオーナーへの信頼感を失うことが報告された。
- F 社では、プロダクトオーナーの知識や経験が十分ではなく、プロジェクト推進が停滞した。当初は、チームで一体となるスタイルであったが、役割を分化しボトルネックが顕在化した。
- ○ I 社では、ビジネス企画との関係を取り持つプロキシ役を導入するなどして、上記の課題を回避していた。

このように、ビジネス企画側にボトルネックが発生しやすいことを認識し、ビジネス企画側を増員する、開発側からの支援を行うなどの対策をとる必要がある。

6.5. 反復中の品質悪化のしわ寄せ

アジャイル型開発は、反復期間をもうけ漸進的にソフトウェアを開発する。その反復期間中にてテストやレビューを行って品質の低下を防止している。しかしながら、中大規模開発において、プロジェクト後半に品質が悪化し、対応に追われる事例があった。

- C 社の事例では、タイムボックス内に動作するソフトウェアを作ることに専念していた。その完了条件に対して不具合の対応に関しては曖昧だった。そのため、不具合対応を先延しにしてしまった。その結果、溜った不具合についてプロジェクト後半に対応に負われた。

- I 社の事例では、反復内の 20～30%は、顧客からのフィードバックや不具合の修正に工数を費やすことを想定していた。しかし反復を繰り返す度に、対処しきれなかった不具合が複合的に重なることによって反復期間で想定していた以上の不具合が発見された。そのため、プロジェクト後半は不具合対応に追われた。

本来ならば、タイムボックス内で未完の不具合があったとしても、バックログ(残作業項目)に含めた上で、適切な優先順位をつけて以降に対応していくことで不具合の蓄積には対処できるはずであるが、それがうまくできなかった。

このような場合には、中大規模開発においては、プロジェクト後半に品質悪化が起こりやすいという課題を理解して対策を事前に考えておくことが必要である。今回の事例で複数報告された、プロジェクト後半をテスト行程とあらかじめ計画するなどの対策もその一つである。I 社では反復で発生した不具合を対応する時間をあらかじめ反復の計画に二割から三割程度組込むことで対応にあたった。

Copyright © 2012 IPA, All Rights Reserved

または、反復単位での品質レベルを明確にして(例えば軽微不具合以外は全対処)反復の完了条件に含めていくことで、反復内での品質の作り込みを重視し、後半に流れる不具合をできるだけ減らす、などの対策も考慮されたい。

6.6. 他組織とのリズムの不適合発生

アジャイル型開発では、反復期間を設け、リズムを作ってプロジェクトを推進している。
しかしながら、所属組織の標準プロセスや、セキュリティ監査、ユーザビリティ検査などは、反復期間を前提にしていないものも多い。つまりは漸進的に動作するソフトウェアが出来上がっていっても、開発がすべて完了しないと、リリースなど次の段階に進めない。

- E 社の事例では、すべての開発が終わらないと、セキュリティ監査や法務確認ができなかった。反復単位でリリースできる機能ができていても、セキュリティ監査が「すべて完成」しないとパスできない限り、本当にシステムリリースすることはできなかった。

このように、所属組織の標準や慣例、関係する法令などをあらかじめ調査し、スムーズにプロジェクトが完了できるように注意する必要がある。更には、組織的にアジャイル型開発の利点を生かすようなルール、リズムに変更していくことも視野に入れる。

6.7. 管理工数増加や開発速度の低下

アジャイル型開発では、反復期間内で実施するタスクをあらかじめ決定する。ウォーターフォール型と比べるとさほど気にはならないが、それまでが、むしろ状況に応じてできたらすぐにリリースをしているような開発(Ad-hoc な開発)をしていると、反復期間という制約が生じるアジャイル型開発は、むしろ速度が低下している、あるいは、短期間の開発において、一気に仕様を固めて一気に開発できる場合には、反復自体がムダではないか、という意見があった。

- H 社では、中大規模開発であったとしても、１～２ヶ月程度の短い期間で完了するプロジェクトの場合、反復を行うオーバーヘッドが無視できないのでは、という意見があった。
- K 社（参考案件）の組織では、計画作りなどのオーバーヘッドなどを嫌う意見が報告された。

このように、反復という制約にとらわれていると、より短期間で開発をしなければならない高速なシステム開発を行っている場合には、逆に管理工数増加や開発速度の低下が危惧されていた。

Copyright © 2012 IPA, All Rights Reserved

7. 想定との比較

調査にあたり設定した調査ポイントに対しての想定と実際の調査結果との比較を述べる。

7.1. 組織体制とコミュニケーションについての想定との比較

7.1.1. 職能横断チーム

(想定) 各チームは職能横断チームによって構成されている

ほとんどの事例において職能横断チームで構成されていたのは想定通りであった。一部職能別チームで構成されている事例もあったが、プロジェクトとしてのバランスがとれていれば、バリエーションとして一つの選択肢となり得る可能性があることがわかった。

7.1.2. 直接的コミュニケーション

(想定) チーム間のコミュニケーションおいて直接的な情報共有を重視している

中大規模事例においては、組織複雑性の観点から、どのような工夫がなされているかを調査した。想定結果として考慮したのは以下の３点であった。

1. 見える化を活用して全体として情報共有を行っている。
2. 頻繁なチーム間の情報共有(階層的朝会の実現)を実現している。
3. オープンなスペースに全員同席している

1 の見える化については、中大規模においても積極的に利用している事例を複数見かけた。2 の階層的朝会についても、複数の事例で見受けられた。更にはスクラムマスターのような代表者のみの横断的会議、事前プランニングなどといった、階層的なミーティング形態をとる事例があった。組織複雑性に対応するには、適切な階層化を行うことと、横のコミュニケーションをデザインする、という工夫が行われていたと考えてよいだろう。3 の全員同席については、物理的に可能であるならば、最初に考慮すべき点であることが発見された。

7.1.3. 準委任契約

(想定) 顧客と開発との契約は準委任で行われている。

今回の調査においては、半数以上が自社開発サービスであった。受託開発のうち準委任で契約しているケースは想定よりも少なかった。逆に顧客と請負契約をかわして実施している事

Copyright © 2012 IPA, All Rights Reserved

例でも、顧客と信頼関係を結ぶことで、なんとか非ウォーターフォール型開発が実現できている、という結果を得ることができた。

7.2. 組織への展開方法についての想定との比較

7.2.1. 段階的適用

(想定) 全体で一斉にアジャイル型を適用するのではなく、試行チームで評価した上で段階的に適用チームを拡大している

非ウォーターフォール型に精通している企業でない限りは、試行チームでの評価が必ず行われていた。ただし意図的に試行しただけでなく、海外企業との共同プロジェクトの中で、必然的に実施することになった事例もあった。海外と協力してプロジェクトを推進していくケースが増えるドメインにおいては、非ウォーターフォール型開発は今後ますます需要が増えるのではないか。

7.2.2. 変化度が高い部分を起点

(想定) 変化の度合いが高い部分を起点として拡大している。

今回の調査の結果、変化の度合いが高い部分(=必要性が高い)という軸と、導入障壁という二つの軸が存在することがわかった。展開という視点で見た場合には、必要性だけでなく、広げやすさも考慮して展開を行うことが重要であることが発見された。

7.2.3. リーダーシップ

(想定) 組織文化を変遷させるためのリーダーシップが存在する

今回の調査結果では、非ウォーターフォール型の導入は、特にウォーターフォール型開発を実施してきた組織にとっては単に開発手法の導入ではなく、新しい組織文化の導入に近いことが明らかになってきた。そのために導入に際してはリーダーシップをとる人物が重要となる。

ただし、今回の調査対象に限定すると「たまたま認定スクラムマスター研修にいけたから」という推進者もいれば、「10 年前から導入していた」という情熱溢れる推進者もいた。
単純に情熱溢れるリーダーシップというわけではなく、少しずつ試していきながら、成果を肌で感じた結果として、確信をともなったリーダーシップであることが感じられた。

Copyright © 2012 IPA, All Rights Reserved

7.3. 分散拠点開発についての想定との比較

7.3.1. 同一拠点から段階的分散

(想定) 同一拠点で作業して、次の段階で分散してチームを作る

非ウォーターフォール型開発ではチーム内での信頼感を醸成し、それをベースにオープンなコミュニケーションを取ることが求められる。そのため、分散拠点であっても、最初にチームとして信頼感を基礎として構築する。その後で拠点間に散らばり、コミュニケーションを円滑に進めていると想定した。調査結果からは、分散拠点開発の事例の半数以上が同一拠点からの段階的分散を選択していた。

物理的距離が離れるということは、それだけでコミュニケーションの帯域が狭められ大きな制約となる。いきなり最終形の分散拠点を形づくるのではなく、非ウォーターフォール型開発がソフトウェアを漸進的に成長させていくように、チームも漸進的に成長していくことをデザインすることが求められることを発見した。

7.3.2. コミュニケーションツールの活用

(想定) IP 電話会議システム、チャットなどコミュニケーションツールを駆使している

分散拠点開発では、その状況のなかでリッチなコミュニケーションを実現するために、様々なツールを使うだろうことは想定された。しかし単にリッチなコミュニケーションツールを使うというだけでなく、その目的に応じたツールを使いわけているという結果が明らかになった。

TV 会議システムは、拠点間において会議をあたかも同じ部屋で行うようにする目的で利用する。ピアツーピアのビデオチャットは担当者同士が気軽にコミュニケーションをとれる目的で利用する。テキストベースのチャットで複数人への同報通信、互いの時間を拘束しすぎない、より簡易的なコミュニケーションツールとして利用されていた。

今の状況の中で、何が必要か、何が足りないのかを見極めた上で、適切なツールを複数利用することが重要であることが発見された。

Copyright © 2012 IPA, All Rights Reserved

7.4. アーキテクチャ・共通基盤の構築手順についての想定との比較

7.4.1. アーキテクチャの漸進的開発

(想定) アプリケーション部分とは異なるサイクルで、共通基盤、アーキテクチャも非ウォーターフォール型を用いて漸進的に設計開発されている

アーキテクチャの漸進的な設計開発を想定していたが、結果は早期に構築した上での利用と、既存の基盤の再利用の二つのパターンであることがわかった。プロジェクトというある時間軸で見た時には、既存の基盤の再利用という形に見えるが、より広い時間軸で見た場合には、漸進的にアーキテクチャを設計実装して成長させていき、プロジェクトのタイミングで再利用しているとも言える。
また基盤として使いながらも、要件に合わない所があれば、プロジェクトチーム自身で改善していくという事例もあった。プロジェクトを越えて、組織的に中大規模事例を経験していく中でのノウハウを生かして基盤として熟成させていくことが重要であることがわかった。

7.4.2. アーキテクチャチーム

(想定) アーキテクチャ、共通基盤を担当するチームが存在して開発チームに支援をしている。
複数の事例でアーキテクチャチーム、基盤チームを構成して開発チームの支援をしていることが発見された。その中で PaaS 基盤をプロジェクト内で構築している事例があったが、そこでは基盤チームのメンバーが、アプリケーション・サービス側のチームに参加して知識を伝えていた。単に「提供し教える」立場と、「利用し教わる」立場で役割が分けるだけでなく、よりチームが密接に交わる状況を作り出すことで、更なる知識伝播が行われることが発見された。

7.5. システムの分割とインテグレーションについての想定との比較

7.5.1. サブシステム間の継続的インテグレーション

(想定) サブシステム間の統合は継続的インテグレーション(CI) により頻繁に実施されている

継続的インテグレーションを実施している事例は多数の事例で見かけられたが、実際にサブシステム間の連係を継続的インテグレーションで実現している、と報告した事例は少なかった。

そもそも自動化されたテストが十分に配備されている事例が少なかったという事実がある。
小規模事例は頻繁に見かけられる、自動化された単体・機能テストが十分に配備されている、という事例は一部であった。

Copyright © 2012 IPA, All Rights Reserved

7.5.2. 疎結合なシステム

(想定) システム間が、できるだけ疎結合になるように分割されている。

サブシステムを疎結合になるように分割している事例が大半を占めた。しかしその分割の仕方にパターンがあり 3 種類に分かれていた。

1. サービス・機能単位
2. プラットフォーム単位
3. コンポーネント単位

同じ疎結合な関係という点では同じだが、その位置付けは大きく異なることが発見された。

7.5.3. 機能単位のチーム

(想定) システムは機能やサービスという単位で分割されている。チームはその分割とチームの分割がマッピングされ、チームはその機能単位で構成されている。

調査の結果、先程の分割の三種類のうち 1.サービス・機能単位、2. プラットフォーム単位によって、チームがフィーチャーチーム[16]として構成されていた。

分割パターン 1 あるいは 2 の例では、ソーシャルゲームにおける「〜機能」や「Android クライアント」のような分割がなされ、その単位でチームを割当てる。分割パターン 3 の例では、「フロント、バックエンド」という分割を行い、その単位でチームを割当てる。前者はあるチームが「〜機能」を実現するために単独で実現することができるが、後者では利用者が使う機能を実現するには、「フロントチーム」と「バックエンドチーム」が連係をする必要がある。

前者の分割パターンは、ソフトウェアを作って利用者からフィードバックを得やすくし、長期的にチームを固定して価値提供をしようとした時に選択する指針であると考えることができる。後者の分割指針は、システム全体をいかに効率よく分割して作りあげ、できるだけ最短でシステム全体を作りあげる際に選択する指針である。今回の調査結果では、多くが前者の分割指針を選択していることがわかった。

[16] http://www.featureteamprimer.org/ja/

Copyright © 2012 IPA, All Rights Reserved

7.6. 明確な問題意識と効果について想定との比較

7.6.1. 経営者や顧客の判断

(想定) 非ウォーターフォール型開発への適用判断を経営者や顧客がしている。

今回の調査対象の事例の中で非ウォーターフォール型開発を推進したのはすべて開発側であった。そのうち、受託開発の３事例では開発側から発注側の置かれている状況を踏まえた上で、非ウォーターフォール型開発の提案をして了承されている。

自社開発のサービス提供者にとっては、非ウォーターフォール型開発は、積極的に導入していくメリットのある手法であることがわかった。他方、従来のやり方(ウォーターフォール型開発ではなく、特定のプロセスを決めずに、その場の状況に応じて素早くリリースしていく)と比べての速度低下の指摘もあった。ここは置かれている状況によって導入判断すべき所である。

受託開発の場合、発注側としては、置かれている状況の中で選びうる最善の選択をしたい。現状、発注側は開発手法として非ウォーターフォール型の存在を知らないことが多い。しかしその効果や進め方は発注側からしても、状況によっては納得できるものであるとの意見があった。(H 事例)

開発側としては、発注元の置かれている状況、望むビジネスに対応した開発手法の選択肢を持ち、発注側に提案できることが望ましい。逆に発注側にとっても、ソフトウェアシステムがビジネスを支援するものから、ビジネスそのものに変化している重要性を認識した上で、システム開発に求める選択肢として非ウォーターフォール型開発を提案依頼に含めることができるような状況に改善されることが望ましい。

7.6.2. 問題意識、期待、効果の明確さ

(想定) 解決したい問題意識、考慮すべき制約、期待、その効果が明確になっている。

各社の非ウォーターフォール型開発導入に際して、抱えていた問題とその期待と効果を列挙した。どの事例も、解決したい問題を目の前にして制約のある中で導入を行っていた。期待する効果については、回答してもらった事例がすべて５段階評価において 3 以上であった。効果としては、元々解決したかった問題が解決できただけでなく、「エンジニアが楽しい」「社員満足度が上がった」「担当が変更しても生産性が落ちない」などの予期せぬ効果も複数報告された。

Copyright © 2012 IPA, All Rights Reserved

事例	問題	制約、考慮点	期待	効果
A	経営層、企画者、エンジニア、顧客、顧客見込み者の要望や社会情勢を取り込みながら開発を進める。B2C サービスのため、事前に要件を固めることはできない	プロジェクトでは、企画者、エンジニア、デザイナーが一体となって実施した。 特に不適合者を辞めさせるようなことはない	ビジネス自体がスピードと、変化への対応速度を重視している	新規立ち上げは、３ヶ月程度で実施できた。大きなゲームをリリースできた
B	スケジューリングの難しさ、組織のマネージャの意思決定がボトルネック	1 つ 1 つの開発サイクル（数日～2 週間）が短い 企画者/デザイナーが協業してものづくりする必要があった。	何かしらの枠にはめることで、役割や担当が替わってもキャッチアップでき、進行もスムーズにできる。	スケジューリングの難しさ、組織のマネージャの意思決定のボトルネックが解決した。 担当が変更しても生産性が落ちなかった。
C	管理コストを減らしてその分プロダクトのクオリティをアップさせたい。	クライアントへは開発スタイルは伝えていないため、ガントチャートのスケジュール表を提出する必要があった。またアジャイル開発の経験者がほとんどいなかった。	見える化による問題、トラブルへの早期対応。透明性を上げる。	問題解決意識が高くなったスタッフが多く見られた。チーム全体のモチベーションが維持しやすかった。生産性が上がった。管理コストが減った。クオリティは思ったほど上がらなかった。
D	非公開	非公開	非公開	非公開
E	既存のシステムはあるが、仕様やビジネスロジックが不明だった	全体がわかる人が誰もいない。 ビジネスロジックを明確に書ける人が誰もいない。 最初に作った人がいない。		成長（失敗と経験のサイクルが早かった） エンジニアが「楽しかった」 開発のオーバーヘッドが通常プロジェクトより低い
F	企画=顧客、IT=下請けという社内分業体制。それによる、高コスト化・オーバーヘッドの増加。途中で企画が変わっても追従できず、企画側が満足できない。大型の案件が 2 つ続けて失敗した。	組織変更を行った。企画&開発一緒に、IT センターを解散した。推進者がこれでなければできないと言った。エグゼクティブのサポートもあった。企画と開発のフロアを一緒にした。まず、推進者が物理的に行った。その後、サービス毎に島をつくった。パーティションを取り除いた。コミュニケーションの場をつくった。 いきなり企画側の人に PO をやってもらうのはハードルがあったので、技術企画室という部門をつくって、IT 部門から PO を出した。技術企画室は役割を終えて、のちほど解散する。	リスク低減 フィードバックがすぐに来る 上手になっていく	社員満足度はそれほど高くないが、IT 部門は 75%がよいと言っている。これは日本でもトップ 3 に入るレベル。みんなに、良さを分かってもらっている人が育つ、コミュニケーションが活発に、技術力 UP

Copyright © 2012 IPA, All Rights Reserved

G	開発の速い段階でリスクをつぶしたい（期限までに確実に動作させたい。問題点を早期に見つけたい。） 自律的なチーム作りにより管理する負荷を減らしたい（プロセスの浸透・ノウハウ共有、見える化による自ら動けるしくみの実現） プロセス順守。品質低下の防止。（ルール無視による作業もれ防止、ツール徹底利用によるレベルダウン防止。）	以前は、顧客と握った（ウォーターフォール型開発）線表と、開発内部の線表は異なり、実質２重線表で開発を行なっていた。	以下、上記案件での効果を確認した後、他の案件での採用について記載する。 短期開発（3〜4ヶ月で第１段を確実に出す必要がある案件）において、リスク対応、進捗・品質を確実にするために、採用していた。 現在、実施している案件は、不確実性に対応するため。（新市場の開拓をミッションとするプロジェクトのため）	リスクに早期に発見でき、対応を計画する余裕ができた。品質に関しても同様、開発の後半で超繁忙状態になってしまうことを防ぐことができた。 メンバーの増減に柔軟に対応することが可能で、顧客の投資額に応じて体制を縮小・拡大することや、新たなチームを短期間で立ち上げることなどが可能となった。 スピードを求める顧客の満足を得られやすい。
H	発注側でも機能で何が出来れば良いかが暗中模索で、手探りをする必要があった。	発注元の開発プロセスがあった。形上はこのプロセスに則っていた。	機能で何が出来れば良いかが暗中模索で、手探りをする必要があった。	開発手法ではなく、マネジメントシステムがとても良い ここから先はできませんね、などユーザと開発を見て、納得していけるのがいい アジャイルのときは、ある程度動いているので、想像でき早め早めに見せてくれたので、品質は、ある程度の領域で担保されている点 初心者の担当者だったが、ウォーターフォール型開発は、仕様書通りに作って実際にダメ、ということがあったが、アジャイルは考えていたものとの違いがすくない
I	既存のサービスをそのままリプレースするわけではなく、新しいコンセプトで顧客と対話しながら新しいサービスを構築していくため		優先順位づけられた開発をすることにより、顧客が良いと判断した時点でリリースができること。 決められた予算にもっとも近い形で開発ができるように、優先順位でもって不要なコスト、要らない機能の選択を行うこと	変化への対応。要件、機能削減、予算縮小への対応。
J	・開発期間を短縮する ・開発生産性を向上させる	・開発ドキュメント，レビュースタイル ・ユーザー側，開発チーム双方のマインド醸成 ・開発パートナーと契約 ・マネジメント手法	実際に動くモノを見てユーザーフィードバックを受けながら、開発を進めるので、柔軟性もあり、システム（Web サービス）の質も上がる。	同規模でのウォーターフォール型開発と比較して期間は半減、開発生産性は倍になった。
K	サービス展開に耐えうるクオリティを保つこと、サービスインのスケジュールを守ること、着実に開発を進行したい。	開発プロジェクトをゼロから立ち上げたメンバーがいなかった 高スキルのエンジニアがいない	スケジュールを守ること、開発を着実に進行すること、技術面でのサポート	おおよそ想定内のスケジュールでサービスインができた 進捗が可視化され、スコープ見直しをプロジェクトの半ばで行えた

表 7-1 非ウォーターフォール型導入の背景一覧

Copyright © 2012 IPA, All Rights Reserved

7.7. 各種管理について想定との比較

7.7.1. テストフェーズの存在

(想定) 特に品質に気をつかうケースにおいては、反復以外にテストフェーズを設けて品質管理を行っている。

反復以外にテストフェーズを設ける事例が多数報告された。一方、反復内で対応しきれなかった障害が後半のテストフェーズの皺寄せとなった事例報告も複数あった。反復の中での品質の作り込み戦略と、全体としての品質向上の戦略の両方が必要であることがわかった。

7.7.2. 品質に関するプラクティス

(想定) 単体テストの自動化、継続的インテグレーションといった品質に関連するプラクティスを実施している。

単体テストの自動化、継続的インテグレーション、リファクタリングに取り組んでいる事例は多かったが、徹底して実施している事例は一部であった。非ウォーターフォール型開発の導入については、まずプロセス(反復漸進型)を取り入れて、反復単位の計画づくり、開発、レビュー、ふりかえりの実施から導入する。その過程で職能横断チームや、朝会、見える化、その他コミュニケーションに関するプラクティスを取り入れていく、という傾向があった。

エクストリームプログラミング(XP)で提唱されているエンジニアリングプラクティス(技術的な実践)は、プロセス、コミュニケーションに関するプラクティスよりは、優先度を下げて実施される傾向があることがわかった。
一方、E 社、F 社の事例のように当初からエンジニアリグプラクティスについても重視している事例もあり、導入の状況によっては、エンジニアリングプラクティスについても当初から着目する場合があることがわかった。

Copyright © 2012 IPA, All Rights Reserved

7.8. 部分適用についての想定との比較

7.8.1. 変化の必要な部分のみの適用

(想定) 部分適用が存在する場合はアジャイル型を「変化の必要な部分」にのみ適用している。

本想定については、展開と同様に「変化の必要な部分」というよりも、もっと広い意味での「非ウォーターフォール型に取り組む動機に適している箇所」という表現が正しかった。
D 社、H 社、I 社の事例では「要件が固められない」部分に適用された。C 社の事例では、管理コスト削減が第一目的であったので、変化への適応や、要件の不安定さとは関係なかった。逆に「ウォーターフォール型開発でやったほうがよい領域」を見極めて、部分的に導入を回避することで、全体としてのバランスをとっていた。

7.8.2. 同期ポイント

(想定) アジャイル開発、非アジャイル開発の混在チームにおいては、互いの同期ポイントを反復のタイミングで合わせている。

非ウォーターフォール型開発は、反復のリズムを刻んでいるため、ウォーターフォール型のように一定のリズムがない部分と協働作業を行うには、何らかのリズムを合わせないといけないのではと想定した。

今回の調査結果においては、部分適用について、特に同期のタイミングを合わせているといった事例はなかった。C 社の事例では、そもそもスクラムチームとウォーターフォール型開発チームは同期の必要がないように分割されていた。I 社の事例では、ウォーターフォール型開発で開発する機能については、インターフェースを定めておき、完成まではスタブ(疑似環境)で対応していた。H 社の事例では、非ウォーターフォール型で動作可能な形で開発していくサブシステムと、ウォーターフォール型開発で開発しているサブシステムの同期ポイントが作れずに、最終的には手動でマージすることで対応していた。

7.9. 組織文化についての想定との比較

7.9.1. 異組織文化間移行の問題

(想定) 組織文化の移行が発生している場合は、一時的にパフォーマンスが落ちる、作業者の反対に合う、などの問題が起きている。

Copyright © 2012 IPA, All Rights Reserved

今回の調査の結果、特にウォーターフォール型開発から非ウォーターフォール型開発へ移行している場合には「メンバーが合わない」場合や「当初から反対に合う」といった問題が発生していたことがわかった。

7.9.2. スムーズな組織文化移行

(想定) 組織文化がアジャイル開発の適用前と後で近い場合は文化的問題が発生しない。

自社サービス開発で、それまでは特に決まったプロセスがないような事例においては、文化的問題は発生していなかった。K 社の事例では「組織の文化と非ウォーターフォール型開発の文化の親和性が高い」と明言していた。
残念ながら組織文化調査は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との比較を行っていたため、元々ウォーターフォール型開発で実施していなかった組織の文化は調査できていない。

7.9.3. 管理度合い

(想定) ウォーターフォール型に比べて管理度合いが低くなり、協調度合いが高くなっている。

組織文化の傾向については、管理度合いが下がり、協調度合いが上がるだけでなく、創造度合いも同様に上がることがわかった。

7.10. 従事者健康度についての想定との比較

7.10.1. 精神的健康度

(想定) ウォーターフォール型開発と比べて、非ウォーターフォール型開発に携わっているメンバーは、精神的健康度が高くなっている。(生き生きと仕事をしている)

調査結果としては、ウォーターフォール型開発に従事している人に比べ、非ウォーターフォール型開発にストレスが少なく、生きがい度が高いという結果がでた。また非ウォーターフォール型開発導入の効果の中でも「エンジニアが楽しく仕事ができる」「組織満足度が向上した」といった定性的な報告もあった。

Copyright © 2012 IPA, All Rights Reserved

8. 考察と提言

今回は、中大規模の非ウォーターフォール型開発の事例を調査し、工夫、課題、文化、健康度を明らかにした。ソフトウェア開発案件やプロジェクトは、その状況や制約を把握し、適用範囲や文化にあった開発スタイルを選択できることを示した。その中でも、中大規模であっても、課題を理解し、工夫を参考にしながら非ウォーターフォール型開発を選択できうることを示した。

8.1. 社会的要請と適応範囲

今回、中大規模の非ウォーターフォール型開発の事例を 10 件収集できた。このことは、非ウォーターフォール型開発の代表的なアジャイル開発が日本に紹介されてから１０年を超え、アジャイル開発が少しずつではあるが、事例が着実に増えていることを表している。インターネットの台頭およびグローバル経済の発展に伴い、社会的情勢やビジネ環境、法令の変化が激しくなった。

その中で、今まで通り、ビジネスを支援する業務システム作りから、顧客と直接やり取りする業務システムへと重点が変わってきている（H 社）。実際に、今までは企画提案型のアプリケーション作りであったが、これからはユーザ参加型のアプリケーションシステム作りが注目されるようになってきた（G 社）。

これらのアプリケーションシステムは、今までは、要望や仕様を引き出し、その要望に従って作っていくシステムであった。しかしながら、ユーザ参加型のアプリケーションシステムは、実際に使ってもらって、問題点や要件が見えてくる（G 社）。要望や価値を、明文化できないが、センスの良さを信じて、作り込んでいくようになってきた。

そのような動向に呼応するかのごとく、ソフトウェアを用いたサービスや、それに伴う開発スタイルも変わってきている。そのような分野は、ウォーターフォール型開発のように全体計画を事前に立て、その計画に基づいてソフトウェアを構築することが困難になってきている。業務知識と技術知識の両方を高く、しかも継続的に学び続ける必要がでてきている（A 社、B 社、E 社、F 社、G 社、J 社、K 社など）。そのようなサービスについて提供する準備を行い、提供し続けるためには、非ウォーターフォール型開発の選択は必然に近い。そのサービス規模が拡大し、開発規模も中大規模になっていくのは自然な流れである。

Copyright © 2012 IPA, All Rights Reserved

8.2. 中大規模事例での工夫と課題

中大規模開発においての工夫は、大きく三つに分けられた。アジャイル型開発において今まで言われてきたベストプラクティス群と比較し、同様であるが特に注意を要するもの、逆のアプローチを採るもの、大規模特有のものである。

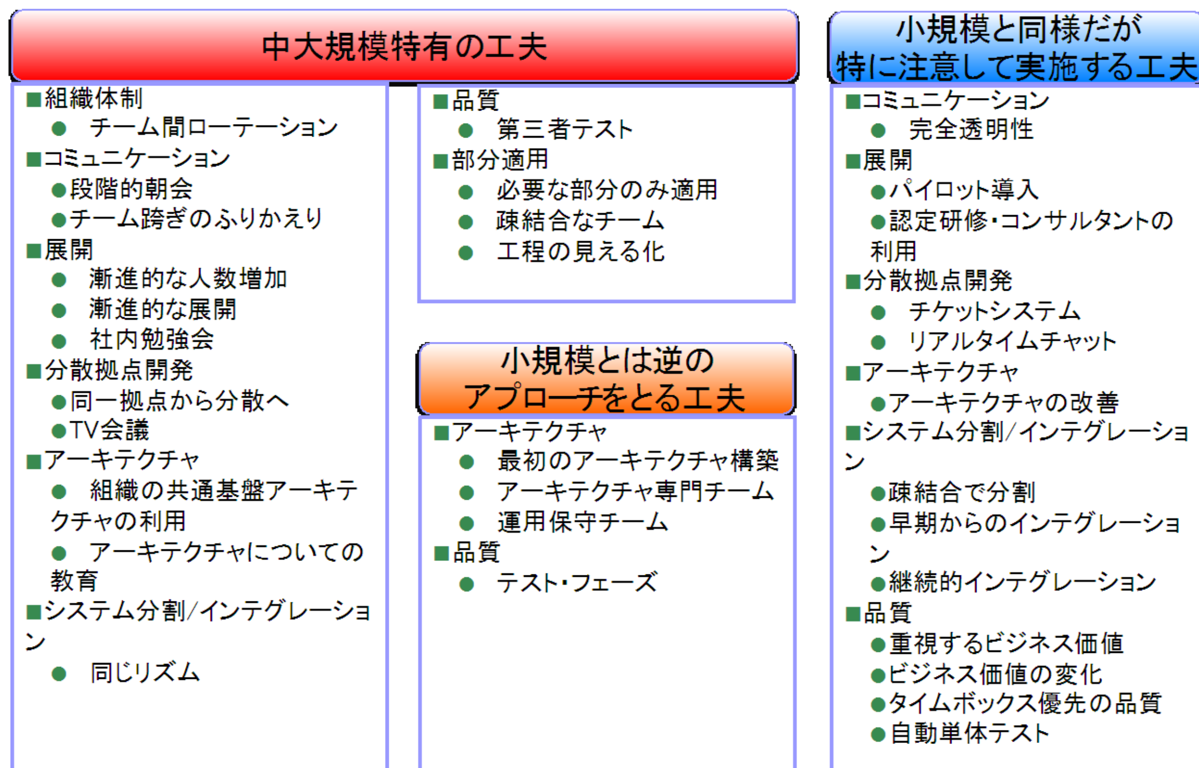

図 8-1 中大規模で発見された工夫一覧

その結果、規模やビジネス特性などの状況によって、多くのポイントにおいてふさわしい工夫が選択されている。たとえば、部分適用のポイントでは、中大規模特有の「疎結合なチーム」と、小規模アジャイル型開発のプラクティスとは逆の「テストフェーズ」の両方が選択されている。アーキテクチャのポイントでも、組織の共通基盤の利用、アーキテクチャ専門チームのように、アジャイル型開発のプラクティスだけでない。ウォーターフォール型開発や反復型開発のノウハウ、そのほか幅広いノウハウをヒントにしつつ、プロジェクトを円滑に進めていることがわかる。

つまり、そのプロジェクト状況に応じたプラクティスが選択され、組織文化が移行したことがわかった。

Copyright © 2012 IPA, All Rights Reserved

8.3. 中大規模の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

目指すべきゴールを実現するために、これらの課題と提言をおこなう。今回の調査で見えてきた中大規模開発に非ウォーターフォール型を適用する際の課題を 5 つのカテゴリ(法・環境、プロジェクト、人材、組織、リファレンス)に分けて記述した。さらに、それぞれの課題を解決に導くための提言を示す。これらの提言が過年度からの非ウォーターフォール型開発の調査の目指すべきゴールにつながることを切に願っている。

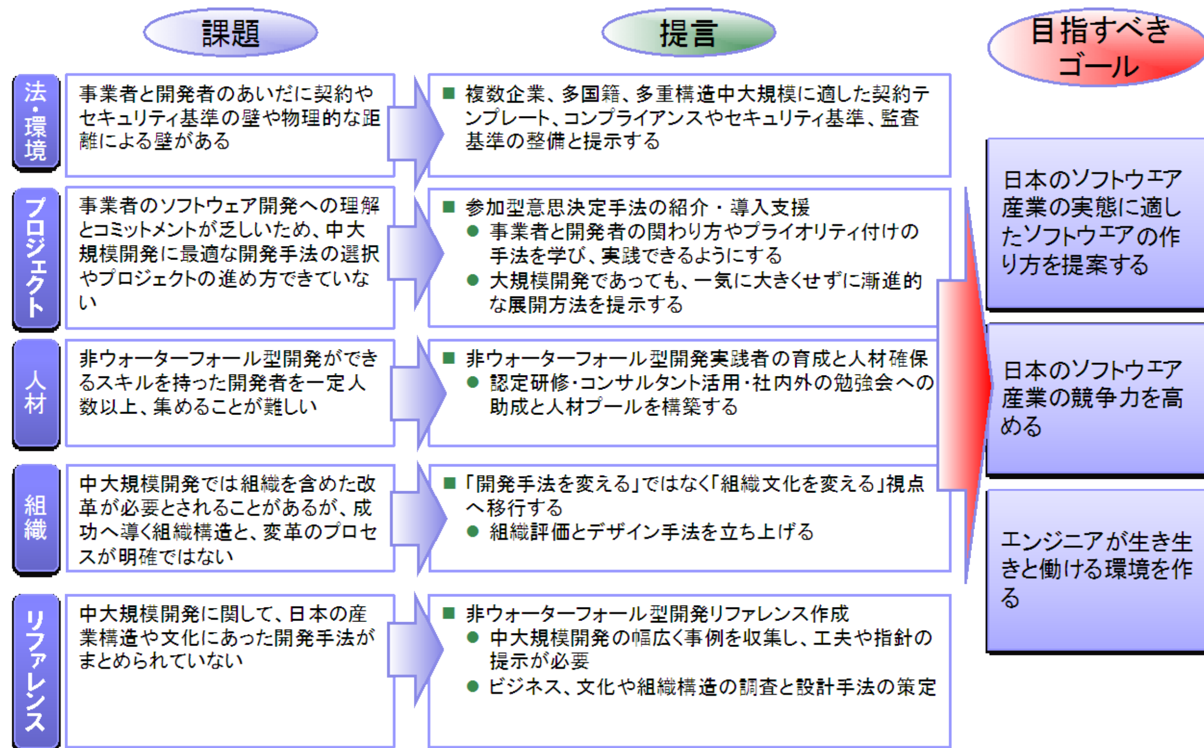

図 8-2 課題と提言

Copyright © 2012 IPA, All Rights Reserved

参考文献

[1] Coplien, J. O., & Bjørnvig, G. (2010). *Lean architecture for agile software development*. Chichester, U.K.: Wiley.

[2] Eckstein, J. (2004). *Agile software development in the large: diving into the deep*. New York: Dorset House Pub.

[3] Leffingwell, D. (2007). *Scaling software agility: best practices for large enterprises*. Harlow: Addison-Wesley.

[4] Larman, C., & Vodde, B. (2009). *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Upper Saddle River, NJ: Addison-Wesley.

[5] Schneider, W. E. (2000). *The Reengineering Alternative: a plan for making your current culture work*. Burr Ridge, Ill.: McGraw-Hill Companies.

[6] Cameron, K. S., & Quinn, R. E. (2011). *Diagnosing and Changing Organizational Culture: Based on the Competing Values Framework* (p. 288). Jossey-Bass.

[7] キムS・キャメロン, & ロバートE・クイン. (2009). *組織文化を変える[単行本（ソフトカバー）]* (p. 224). ファーストプレス.

[8] Ono, E., Nozawa, T., Ogata, T., Motohashi, M., Higo, N., Kobayashi, T., Ishikawa, K., et al. (2011). Relationship between social interaction and mental health. *System Integration (SII), 2011 IEEE/SICE International Symposium on*, 246-249. doi:10.1109/SII.2011.6147454

Copyright © 2012 IPA, All Rights Reserved

付録1:データ編(精神的健康度)

精神健康度の調査結果である。対象者の属性(性別、立場)、プロジェクトの属性(NWF/WF、プロジェクト完了時期、成功度、満足度、人月、改善頻度、要件変更度、リリース頻度)と、プロジェクトでの健康度(信頼度、互恵性、ストレス度、いきがい(QoL)度)である。

- 非ウォーターフォール型開発:NWF、ウォーターフォール型開発:WF と略す
- 「立場」は別表「組織構造モデル」を参照のこと
- 「〜度」は 0〜1 の値である。

		プロジェクト的属性								精神的属性			
性別(女0,男1)	立場	NWF:0, WF1	プロジェクト完了時期	成功度	満足度	人月	改善頻度	要件変更度	リリース頻度	互恵度	信頼度	ストレス度	QoL度
1	5	0	0	0.7	0.7	20	1	0.8	0.1	0.8	1	0.4	0.8
0	3	0	0	0.3	0.3	0	0	0	0	1	0.2	0.2	0.8
1	3	0	0	1	1	60	0	0.5	0.1	1	1	0.1	1
1	3	0	0	1	1	0	0.1	0	0.1	1	1	0.3	0.8
1	3	0	0	0.7	0.7	50	0	0.7	0	1	0.9	0.4	0.7
1	3	0	0	1	0.7	1	0.1	0.2	0.3	1	1	0.2	1
1	3	0	0	0.7	0.3	1	0.1	0.2	0.3	0.7	0.4	0.4	0.5
1	3	0	3	0.7	0.7	2	0.3	1	0.8	1	0.4	0.4	1
1	3	0	0	0.7	0.7	360	0.1	0.5	1	1	0.9	0.3	0.8
1	3	0	0	0.7	0.7	6	0.1	0.1	0	0.7	0.7	0.4	0.7
1	3	0	0	1	1	1	0	0.3	0	0.3	1	0.1	0.7
1	3	0	1	0.7	0.7	100	0.1	0.6	0	0.9	0.7	0.3	0.7
1	3	0	0	0.7	0.7	0	1	1	1	0.8	0.6	0.5	0.8
1	3	0	0	1	1	120	0.1	0.5	0	1	1	0.3	0.5
1	3	0	0	1	1	20	1	0.5	0	1	1	0.4	0.7
1	6	0	0	1	1	45	0	0.3	1	1	1	0.3	0.9
0	3	0	0	0.7	0.7	4	1	0.5	0.2	0.9	0.8	0.4	0.5
1	3	0	0	0.7	0.7	3	0.1	0.3	0.1	0	0.4	0.6	0.6
1	3	0	5	1	0.7	25	0.1	0.1	0	1	1	0.4	0.8
1	3	0	0	0	0	25	0.1	1	0	1	0.7	0.3	0.9
1	3	0	1	0.7	0.7	4	0	0.3	0	0.8	0.8	0.1	0.7
1	3	0	0	0.3	0.7	100	0	0.7	0	1	0.8	0.5	0.7
0	6	0	0	0.3	0.3	1	0.1	0.2	0.1	1	1	0.5	0.7
0	2	0	5	1	1	60	0	0.5	0.1	0.9	0.7	0.6	0.7
1	3	0	0	0.7	0.7	35	0	0.8	0	1	1	0.6	0.8
0	3	0	0	0.7	0.7	10	0	0.1	0	1	1	0.2	0.5
1	3	0	0	0.7	1	15	0	0.3	0.1	1	1	0.4	0.8
1	3	0	0	0.3	0.7	0	1	0.6	0	0.8	0.3	0.7	0.8
1	2	0	0	0.7	0.3	0	0	0.5	0	0.7	0.2	0.7	0.1
1	3	0	0	0.3	0.7	10	0.1	0.3	0.1	0.7	0.3	0.3	0.6
1	2	0		0.7	0.7	5	0.1	0.3	0.1	1	0.6	0.3	0.6
1	3	0	0	0.7	0.3	10	0.1	0.2	0.1	0.9	0.7	0.1	0.9
1	3	0	0	0.7	0.7	8	0.1	0.6	0.1	0.9	0.8	0.4	0.6
1	3	0	2	0	0	40	0	0.8	0	0.9	0.6	0.3	0.6
1	3	0	8	0.7	0.7	12	0	0.2	0	1	0.6	0.5	0.5
1	2	0	0	0.3	0.3	10	0.1	0.8	0	1	0.9	0.3	0.8
1	3	0	36	0	1	100	1	1	0	1	1	0.1	1
1	3	0	0	0.3	1	75	0.1	0.3	0.1	0.9	0.4	0.4	0.8
1	6	0	0	0.7	0.3	30	0	0.3	0.1	0.9	0.9	0.5	0.8
1	3	0	1	0.7	0.3	100	0.1	0.6	0.1	1	0.8	0.6	0.5
1	3	0	1	1	0.7	100	0.2	0.5	0.1	0.3	0.6	0.6	0.8
1	3	0	1	1	1	100	0.1	0.7	0.1	1	0.7	0.4	0.8
1	3	0	1	0.7	0.7	100	0.1	0.5	0.1	1	0.7	0.6	0.8
1	6	0	1	0.7	0.7	100	0	0.3	0.1	1	0.8	0.3	0.8
－	3	0	1	0.3	0.7	100	0.1	0.5	0.1	1	0.8	0.3	0.8
1	3	0	1	0	0.3	100	0	0.3	0.1	0.3	0.6	0.4	0.5
1	3	0	0	0.7	0.7	240	1	0.7	0.2	1	1	0.2	0.7
1	3	0	0	1	0.7	30	1	0.5	0	1	0.9	0.2	0.6
1	3	0	0	0.3	0.7	4	0	0.3	0	0.7	0.7	0.3	0.4
1	4	0	0	1	1	30	0.1	0.5	0	1	1	0.2	0.8

		プロジェクト的属性								精神的属性			
性別(女0,男1)	立場	NWF:0, WF1	プロジェクト完了時期	成功度	満足度	人月	改善頻度	要件変更度	リリース頻度	互恵度	信頼度	ストレス度	QoL度
1	3	0	0	1	0.7	200	0.1	0.6	0.1	0.9	0.8	0.3	0.7
1	3	0	0	1	0.7	240	1	0.5	0	1	0.9	0.2	0.6
1	3	0	0	0.7	1	200	0.1	0.6	0.1	1	0.8	0.4	0.7
1	4	0	0	0.7	0.7	26	1	0.7	0.1	1	1	0.2	0.7
1	3	0	0	0.7	0.7	300	0.3	0.5	0	1	0.8	0.5	0.6
1	3	0	0	0.7	0.7	11	0.1	0.3	0	1	1	0.4	0.6
0	3	0	0	1	0.7	7	0.1	0.2	0	0.7	0.7	0.2	0.6
1	3	0	0	1	0.7	100	0	0.4	0	1	1	0.6	0.3
1	3	0	0	0.7	1	12	1	0.2	0	0.9	0.9	0.4	0.6
1	4	0	0	0.3	0.3	100	0.1	0.3	0	1	1	0.2	0.8
0	3	0	0	0.7	0.7	240	1	0.7	0.1	1	1	0.2	0.7
－	3	0	0	1	1	96	0	0.6	0	1	1	0.2	0.8
1	3	0	0	0.7	0.7	10	0	0.2	0	0.7	0.7	0.4	0.5
1	3	0	0	1	1	4	1	0.2	0	0.9	0.6	0.8	0.1
0	4	0	0	0.7	0.7	7	0	0.3	0	0.9	0.9	0.3	0.5
1	4	0	0	0.3	1	7	0	0.4	0	0.9	0.7	0.5	0.7
1	4	0	0	0.7	0.7	6	0.1	0.3	0	0.7	0.7	0.5	0.4
1	3	0	0	0.3	0.7	11	0	0.5	0.1	1	1	0.6	0.5
1	4	0	0	0.7	0.7	20	0	0.3	0.1	0.7	0.7	0.5	0.4
1	4	0	0	0.7	0.7	10	0	0.7	0.1	0.7	0.7	0.6	0.4
1	3	0	0	0.7	0.7	9	0.2	0.5	0.1	1	1	0.4	0.6
1	3	0	0	0.3	0.7	26	0.1	0.3	0	0.7	0.4	0.5	0.7
1	3	0	0	0.7	0.3	200	0.1	0.1	0.1	0.6	1	0.6	0.3
1	3	0	1	0.7	0.7	0	0.1	0.3	0.1	1	1	0.5	0.7
1	3	0	0	0.7	0.7	200	0.5	0.2	0	0.7	0.7	0.4	0.6
1	4	0	0	0.7	0.7	0	0.1	0.3	0.1	1	0.7	0.5	0.3
0	3	0	0	0.7	1	9	0.2	0.5	0	1	0.9	0.3	0.7
0	3	0	0	0.7	0.7	25	1	0.7	0.1	1	0.6	0.6	0.4
1	3	0	0	0.3	0.7	10	0.1	0.3	0.1	0.9	0.4	0.4	0.7
1	4	0	0	0.7	1	9	0.1	0	0	1	1	0.3	0.8
1	4	0	0	1	1	9	0	0	0	0.8	0.7	0.3	0.7
1	4	0	0	0.7	0.7	20	0	0.3	0.1	0.7	0.7	0.5	0.4
1	4	0	0	0.7	0.7	10	0	0.7	0.1	0.7	0.7	0.6	0.4
1	3	0	0	0.7	0.7	9	0.2	0.5	0.1	1	1	0.4	0.6
1	3	0	0	0.3	0.7	26	0.1	0.3	0	0.7	0.4	0.5	0.7
1	3	0	0	0.7	0.3	200	0.1	0.1	0.1	0.6	1	0.6	0.3
1	3	0	1	0.7	0.7	0	0.1	0.3	0.1	1	1	0.5	0.7
1	3	0	0	0.7	0.7	200	0.5	0.2	0	0.7	0.7	0.4	0.6
1	4	0	0	0.7	0.7	0	0.1	0.3	0.1	1	0.7	0.5	0.3
0	3	0	0	0.7	1	9	0.2	0.5	0	1	0.9	0.3	0.7
0	3	0	0	0.7	0.7	25	1	0.7	0.1	1	0.6	0.6	0.4
1	3	0	0	0.3	0.7	10	0.1	0.3	0.1	0.9	0.4	0.4	0.7
1	4	0	0	0.7	1	9	0.1	0	0	1	1	0.3	0.8
1	4	0	0	1	1	9	0	0	0	0.8	0.7	0.3	0.7
1	3	1	8	0.7	0.3	60	0.1	0.2	0	0.7	0.6	0.5	0.3
1	4	1	60	0.7	0.7	30	0	0.3	0	1	0.7	0.4	0.3
0	3	1	24	1	0.7	3	0	0.2	0	0.4	0.4	0.5	0.4
1	4	1	1	1	1	8	0	0	0	0.8	0.7	0.3	0.7

		プロジェクト的属性								精神的属性			
性別(女0,男1)	立場	NWF:0, WF1	プロジェクト完了時期	成功度	満足度	人月	改善頻度	要件変更度	リリース頻度	互恵度	信頼度	ストレス度	QoL度
1	3	1	36	0.7	0.7	15	0	0.1	0	0.9	0.7	0.4	0.6
1	3	1	12	0.7	0.7	6	0	0.2	0.1	0	0.4	0.5	0.6
1	3	1	60	0.7	0	300	0	0.3	0	0.8	0.7	0.7	0.3
1	3	1	20	1	1	10	0.1	0.2	0	1	0.9	0.4	0.8
0	6	1	3	0.7	0	1	0	0.1	0	0.8	0.7	0.5	0.5
0	2	1	24	0.7	0.7	15	0	0.2	0.1	0.9	0.6	0.7	0.5
1	3	1	12	0.3	0	30	0	0.2	0	1	0.7	0.7	0.5
0	3	1	30	1	0.7	2	0	0.1	0	1	1	0.2	0.6
1	4	1	25	0.7	0	0	0	0.5	0	0.7	0.2	0.6	0.1
1	4	1	24	0.7	0.7	15	0	0.6	0	0.4	0.3	0.4	0.3
1	2	1	0	0.7	0.7	5	0	0.2	0.1	0.7	0.4	0.4	0.4
1	3	1	6	0.7	0.3	50	0	0.3	0	0.9	0.7	0.1	0.9
1	3	1	0	0.7	0.7	8	0	0.2	0.1	0.9	0.8	0.4	0.6
1	3	1	12	0	0	200	0	0.2	0	1	0.6	0.7	0.3
1	4	1	120	0	0	6	0	0.7	0	1	0.6	0.7	0.3
1	3	1	25	0.7	0.3	100	0	0.6	0.1	1	0.8	0.6	0.5
1	3	1	30	1	0.7	70	0.2	0.5	0.1	0.3	0.6	0.6	0.8
1	3	1	30	0.7	0.7	70	0	0.7	0.1	1	0.6	0.4	0.8
1	3	1	30	0.7	0.7	70	0	0.5	0.1	1	0.7	0.7	0.8
1	6	1	30	0.7	0.7	70	0	0.3	0.1	1	0.7	0.3	0.8
－	3	1	30	0	0.3	70	0	0.5	0.1	1	0.6	0.5	0.7
1	3	1	30	0	0.3	70	0	0.3	0.1	0.3	0.6	0.4	0.5
1	3	1	60	0.7	0.3	50	0	0.3	0	0.7	0.7	0.3	0.5
1	3	1	60	1	0.3	60	0	0.1	0	1	0.8	0.2	0.6
1	3	1	12	0.7	0	6	0	0.2	0	0.4	0.7	0.3	0.3
1	4	1	36	0.3	0.7	50	1	0.4	0	0.7	0.7	0.6	0.5
1	3	1	48	1	1	100	0	0.1	0	1	0.8	0.1	0.9
1	3	1	16	0	0.3	4	0	0.1	0	1	0.8	0.2	0.6
1	3	1	24	0.7	0.3	50	0	0.2	0	0.9	0.8	0.4	0.3
1	4	1	0	0.3	0.3	1	0	0.4	0	0.9	1	0.3	0.5
1	3	1	4	0.7	0.3	600	0.1	1	0	1	0.7	0.5	0.4
1	3	1	24	0	0.7	10	0.1	0.3	0	1	1	0.4	0.3
0	5	1	24	0.7	0.3	3	0	0.1	0	0.7	0.7	0.2	0.6
1	5	1	13	0.7	0.7	24	0	0.1	0	0.4	0.4	0.4	0.3
1	3	1	1	0.3	0.3	250	0	0.3	0	0.9	0.3	0.6	0.4
1	4	1	15	0.7	0.3	100	0	0.2	0	1	1	0.2	0.8
0	3	1	60	0.3	0.3	50	0	0.7	0	0.7	0.7	0.3	0.5
－	3	1	14	1	1	60	0	0.2	0	1	1	0.2	0.7
1	3	1	12	0.7	0.7	2	0	0	0	0.7	0.7	0.4	0.4
1	3	1	13	0.7	0.3	2	0	0.2	0	0	0	0.8	0
0	4	1	1	0.7	0.3	18	0	0.3	0	0.8	0.9	0.3	0.5
1	4	1	12	0.7	1	10	0	0.1	0	0.7	0.7	0.4	0.5
1	4	1	50	0.3	0.3	4	0	0.1	0	0.7	0.3	0.4	0.4
1	3	1	12	0.7	0.7	10	0	0.1	0	1	1	0.6	0.5
1	4	1	14	0.3	0.7	12	0	0.1	0.3	0.6	0.4	0.5	0.3
1	4	1	8	1	0.3	50	0	0.5	0	0.7	0.7	0.6	0.4
1	3	1	11	0.3	0	5	0	0.2	0	0.9	1	0.4	0.6
1	3	1	36	1	0	10	0	0.1	0	0.6	0.4	0.6	0.2
1	3	1	84	0.7	0.3	300	0	0.3	0	0.6	0.9	0.6	0.3
1	3	1	22	0.7	0.7	0	0	0.2	0	1	1	0.5	0.6

組織構造モデル

- 1: 使う人 (システムを利用する人、エンドユーザー)
- 2: 提供する人 (開発されたシステム、サービスを、使う人に提供する人)
- 3: 作る人 (システム開発者)
- 4: 作る人2 (作る人の会社と契約している、一緒に開発する人。別会社であることが多い)
- 5: 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)
- 6: 補佐する人 (開発を進める上で、プロジェクトを支援する人。コンサルタント、コーチ、ファシリテーター)

Copyright © 2012 IPA, All Rights Reserved

付録2：データ編（工夫一覧）

ある状況において直面した問題がある。解決するための制約を回避する解決策のリストである。結果および新たに生まれた課題も含む。
なお、空欄の部分は、未回答もしくは該当なしであり、工夫となっていない。

※ 参考：一覧の[大]、[中]、[参]は、大規模／中規模／その他事例

Copyright © 2012 IPA, All Rights Reserved

大 ミッションチーム

組織体制

状況	ビジネス競争が激しい環境におかれている。	制約	単に機能毎にチームを組んだだけでは足りない	結果	よりシステムの目的レベルを考えて、その実現に向うことができる。	課題	
問題	開発者が「決められたものを作る」という受け身の姿勢では困る	解決策	チームをビジネスミッション単位で構成する。(会員数を増やすこと、など)				

大 独立したデータ

組織体制

状況	システムに必要なデータ(画像)が大量に必要である	制約	画像はイラストレーターが大勢で生産する	結果	画像とソフトウェアが独立して開発を進めることができる。	課題	
問題		解決策	画像などの変更がシステムに影響を与えないようにした				

大 ベンダーロックの回避

組織体制

状況	ベンダーに委託して開発を行っている	制約	あるひとつのベンダーに業務を任せる傾向がある	結果	複数のベンダーにノウハウがシェアされ、ひとつのベンダーが抜けてもその業務やチームが止まることがなくなった。	課題	
問題	ある特定のベンダーにロックイン(ほかのベンダーに移行できなくすること)される	解決策	複数のベンダーがひとつのチームで、共同で働くようにした				

大 契約の階層化

組織体制

状況	大規模開発であるため、人員が多い	制約	複数のベンダーで開発が進められている	結果	システム子会社の人員管理の手間が減っている	課題	
問題	多い人員を管理するための手間と工数が大変になる	解決策	ベンダーが要員を採用し、契約する。システム子会社とベンダーは、一括で契約(この例では準委任契約)する				

大 プロジェクト退出

組織体制

状況	アジャイル開発で進めている	制約	人は、なかなか変われない	結果	アジャイル開発を維持することができる	課題	
問題	人員によっては、アジャイル開発になじまず、ウォーターフォール型の文化を持っている。	解決策	アジャイル開発にふさわしくない人員は、解雇または異動させる				

大 一体となるチーム

組織体制

状況	社会が流動的であり、様々な観点が必要である	制約	企画者やマーケティングだけでは、サービスや製品を改善することが困難である	結果	複数の観点から改善のアイデアが集まっている。プロダクトオーナーの役割がボトルネックとならない	課題	
問題	サービスや製品を改善しながら提供し続けている。	解決策	エンジニアやクリエーターも一体になり、ビジネスの改善を行っている				

大 成長するチームとビジネス

組織体制

状況		制約	企画部門、運用保守部門を分離すると、ノウハウが分断されてしまう	結果	コミュニケーションロスが少ない状態で運用ができている	課題	
問題	企画から運用までを素早く安定して実施したい	解決策	企画から運用保守までが同じチームで連続して関わる				

Copyright © 2012 IPA, All Rights Reserved

[大] 役割別情報交換会
組織体制

				結果		課題
状況	機能ごとにチームを分割し、サービスを提供している	制約	チームを分割しないと、規模や開発規模が大きくなり管理が困難になる	結果	会社全体でノウハウがシャアされた	課題
問題	チームごとのノウハウがシェアされていない	解決策	週に一度、同じロール(例:エンジニア)が集まり、情報交換を行う			

[大] 一体となるチーム
組織体制

状況	システム子会社であったが、ほとんどベンダーに丸投げであった	制約	ユーザ企業とベンダーが直接話すことで、相互理解と開発が効率よく進められる状況になっていた。	結果	早いフィードバックによって、学びが促進され、自信となった	課題
問題	システム子会社は、ビジネスも技術も理解していない状況に陥りがちで、コミュニケーションが成り立っていなかった	解決策	ユーザ企業、ベンダー、システム子会社が一体となったチームを編成し、アプリケーションを自分たちの手で作った			

[大] 近い席
コミュニケーション

状況	開発に携わる人々は密接にコミュニケーションをとりたい	制約		結果	コミュニケーションがとりやすくなった。	課題
問題	席が離れているだけで話す障害になる。	解決策	開発に携わる全てのメンバーを近い席に配置した。			

[大] 隣の島のチーム
コミュニケーション

状況	チーム間は疎結合になっている。同じフロアに全員座っている。	制約	できるだけコミュニケーションロスを減らしたい	結果	話しやすい環境になりコミュニケーションが円滑になる。	課題
問題	疎結合とはいっても、繋がりが強いチームの間ではコミュニケーションが必要である。	解決策	繋りの強いチームは物理的に島を隣にして近くしておく。			

[大] トップマネジメントによる認定取得の支援
展開

状況	小規模なチームへスクラムを導入していたが、その方法が我流であった。	制約	もしスクラムのプラクティスではないと、大規模に展開したときに修正が困難になる懸念があった	結果	適切なスクラムを導入できるようになった。認定スクラムマスターは、他の人への伝道者として活躍した。スクラムを大規模に導入するために、大きな転機となった。	課題
問題	我流であるために、スクラムのフレームワークが適切に導入され、運用されているかが微妙であった。	解決策	会社の理解があり、様々なロール(開発者、デザイナ、企画者、管理者)のものが、認定スクラムマスターになった。			

[大] 早い市場リリース
展開

状況	一般向けサービスを提供し、利用状況や社会変化が大きく、要件を決めているうちに社会状況が変わってしまう	制約	市場が流動的で、仕様を決定しきれない。市場に出してみて、初めて評価される。	結果	ユーザや社会状況の変化に合わせたビジネスができる	課題
問題	アイデアをいかに素早く市場に出したい	解決策	リリースにサービス停止が発生するため、一週間に一度リリースする			

Copyright © 2012 IPA, All Rights Reserved

大
チームの分割
展開

状況	規模が大きくなった	制約	いくつか投機的で、独立したサービスを提供している	結果	管理コストとリスク、ミッションを分離することができた	課題	チームごとに分割されることによって、ノウハウも部分最適化された
問題	管理コストが増大し、人のアサイン計画が困難になった	解決策	機能ごとにチームを分割した				

大
社内勉強会
展開

状況	開発者の一部だけにスクラムを導入していた。	制約	スクラムを導入したいが、他の開発者はスクラムの知識がなかった。	結果	開発者は、スクラムを理解してから導入することができた	課題	
問題	他の開発者は、スクラムを導入していない。	解決策	できる限り少ない人数で、社内勉強会を開催した（実績で9回程度）。				

大
パイロット導入
展開

状況	開発責任者が、スクラムに興味があり、大規模にスクラムを導入し運用したかった。	制約	いきなり大規模に導入すると、失敗したときのリスクが増えてしまう。	結果		課題	しかしながら、スクラムの理解や解釈が、我流であった。
問題	スクラム導入および運用の勘所や経験がなかった	解決策	小さなチームでスクラムを導入し、経験を積んだ。				

大
アジャイル導入戦略フェーズ
展開

状況	アジャイル導入を考えていた	制約	ウォーターフォール型開発に慣れ親しんでいるため、アジャイル開発の考え方に違和感を感じる要員が多くいた。	結果	規範を含め、様々な作法が導入された	課題	
問題	アジャイルの作法についてのノウハウがなかった	解決策	アジャイルに詳しいベンダーにコーチを依頼した				

大
パイロット導入
展開

状況	中大規模の組織で、サービスや製品を永続的に提供している	制約	全体に展開するためにはリスクが大きい	結果	中大規模の組織に、よりよいプラクティスが広がっている	課題	
問題	様々なプラクティスのアイデアがあり、中大規模組織に展開したい	解決策	あるチームであるプラクティスを評価し、その結果を見て、他のチームへ展開する				

大
チームの分割
展開

状況	リーダーがスケジュールと人員リソース管理を行っている	制約	リーダーの負荷が高く、適切な調整が難しい	結果	適切なスケジュールとリソース管理ができた。リーダーも管理工数が削減された	課題	
問題	規模が大きくなるにつれ、スケジュールと人員リソース管理が困難になってきた	解決策	チームを分割し、チーム単位で独立してスケジュールと人員リソース管理するようになった				

大
工程の見える化
展開

状況	開発者は、スクラムのフレームワークを導入したが、デザイナーはスクラムを導入していなかった。	制約	スクラムやアジャイルに対して抵抗感や拒絶感があった。	結果	企画者やデザイナーは、業務プロセスを変更することなく、見える化ができた。	課題	
問題	期日までにデザインが完了せず、納品されない自体があった。開発者だけスクラムを導入していたが、デザイナーの状況が見えなかった。	解決策	決して、スクラムやアジャイルという名前を出すことなく、ヒアリングを開始した。プロダクトパイプラインを、フローとして理解した。リーダーにヒアリングをしながら、カンバン形式として表現した。				

Copyright © 2012 IPA, All Rights Reserved

[大] アーキテクチャに対しての改善

アーキテクチャ

状況	自社で開発している基盤の上でサービスを構築している。	制約	各チーム毎にやるべき作業を抱えている。	結果	アーキテクチャの改善を全体として実施できるようになった。	課題	
問題	個別のアーキテクチャ要件を満たせないことがある。	解決策	アーキテクチャに求めるバックログをスプリントのタイミングで確認しながら、複数チームで分担しながら実施して改善した。				

[大] 秘伝のタレ基盤

アーキテクチャ

状況	既存の大規模システムが存在する。	制約	できるだけ効率化したい。スケーリングにも対応したい。	結果	アプリ部分で考慮する点が少なく効率的に開発を進めることができる。	課題	
問題	新しくアーキテクチャを設計構築する時間はない。	解決策	既存の大規模システムが使っている基盤を利用して開発を進める。				

[大] 粗結合のサービス

システム分割

状況	WF 型と、スクラムを採用しているチームが別々に存在していた	制約	同期のタイミング、内容によっては、困難な状況になりやすい	結果		課題	
問題	同期を取る必要がある場合に齟齬が発生する	解決策	粗結合で、整合性を取る必要がなくなるようにした。				

[大] 適用範囲の見極め

システム分割

状況	いくつかのアプリケーションがあり、会計システム、フロントエンドのウェブや、営業システムがある。	制約	同じ開発手法を用いると、不適合が発生してしまう	結果	変更の度合いに応じた開発が進められている	課題	
問題	フロントエンドのウェブや営業システムはビジネスや要件の変更が比較的多い。会計システムは、比較的変更の度合いが少ない。	解決策	ビジネス要件の変更が多いアプリケーションはアジャイル開発とし、変更の少ないアプリケーションはウォーターフォール型開発で行った。				

[大] インテグレーションバックログ優先的選択

システム分割

状況	サブシステム間でインタラクションが必要な箇所がある。そのために必要な事項はバックログに含まれている。	制約	インタラクションについては双方のチームの間で調整が必要である。	結果	早期よりシステム間インテグレーションが実現できた。	課題	
問題	インタラクションのバックログアイテムを後回しにすると統合で問題が発生しやすい	解決策	率先してサブシステム間のインタラクションが必要なバックログを選択し関連チームで調整しながら適用していった。				

[大] スプリントのための品質の悪化

品質

状況	スクラムのスプリントを導入、運用している	制約		結果		課題	
問題	スプリントに合わせるようにタスクを押し込むために品質が悪化した。	解決策					

[大] テストは WF

品質

状況	繰り返し型で開発を続けている。プロジェクトの規模が大きい	制約		結果		課題	
問題		解決策	テスト期間は WF で実施した。				

Copyright © 2012 IPA, All Rights Reserved

大 品質

完了定義の甘さ

状況	チームはイテレーション毎に動作するソフトウェアを実現している。	制約	完了定義が甘くバグがあってもそのまま直さずに完了としている。	結果		課題	
問題	バグが後半溜って大変だった	解決策					

大 品質

高い品質の提供

状況	一般向けスマートフォンのネイティブアプリを作っている。	制約	高い品質がかかると開発期間や工数がかかってしまう	結果	サービスを永続的に提供できている	課題	
問題	利用者のコメントや評価システムがあるため、品質が悪いと使われなくなってしまう	解決策	品質が十分になるまでリリースを控える。場合によっては、企画自体を停止する。				

大 品質

判断基準の不明確化

状況	サービスや製品を提供していてその優先順位をつけたい	制約	ビジョンを決めることによって、社会状況や製品特性に合わないリスクが増える	結果	日本で有数のユーザ数を誇る	課題	スクラムの手法と適合しにくい
問題	社会状況や製品特性によって、バックログの優先順位を決める基準が変わる	解決策	ディレクタやクリエーターの感性を大切にし、その時々によって判断基準を決定する。あえて明文化せず、暗黙の了解とする				

大 品質

第三者によるチェック

状況	最終的なテストを外部業者で実施する	制約	スプリントを回す必要がなくなった	結果	本行程に取って、無駄な工数を減らすことができた	課題	
問題	課題のチケット＝タスクの状態になってしまった	解決策	テスト行程からは、スクラムのスプリントを辞めることにした				

大 品質

レビューの実施

状況	チームでは、ディレクタ、エンジニア、デザイナーが一体になって開発している	制約	エンジニアもビジネスを見るため、開発に専念できない	結果	十分な品質を提供している。	課題	
問題	品質が悪化してしまった	解決策	設計レビューとコードレビューの二つを実施している				

大 品質

適切なビジネス価値の定義

状況	サービスの種類、提供するハードウェアによって求められる品質が異なっている	制約	スマートフォンでは最初の機能の充実性が大事である。パソコン用は毎週リリースで対応可能	結果	それぞれの領域で優先すべき価値が明確になる。	課題	
問題	求められるビジネス価値がサービスの種類によって異なる	解決策	提供システムが求められる品質、条件を見極めてその特性に必要なビジネス価値を設定する				

大 品質

完了条件は面白さ

状況	サービスで重要な品質は面白さである。	制約	面白さを定量的に評価することは困難である	結果	品質(面白さ)を満たしたサービスが提供できる	課題	
問題	出来上がったものが、面白くない場合がある	解決策	面白さを妥協せずにトコトン突き詰める。面白さに納得できるまでリリースを控える				

Copyright © 2012 IPA, All Rights Reserved

【大】第三者によるQA
品質

状況	サービスを早いサイクルで提供している	制約	小さいチームで動いているため、専門知識がない	結果	セキュリティや法務に準拠し、大きな問題が発生することを事前に防止する	課題	時間がかかることがある。
問題	サービスの品質(ユーザビリティ)やコンプライアンス、セキュリティを担保したい	解決策	専門のチームにより、評価してもらう				

【大】自主的管理の導入
その他(管理)

状況	WF 手法で管理のため、管理担当者がヒアリングをしながら一週間に一度、アップデートをしていた。	制約	真の情報を理解したいが、ヒアリング回数を増やすと工数が増えてしまう。	結果	リアルタイムで、真の情報を見えるようになった。	課題	
問題	報告と管理のための時間差が発生し、そのときの生の情報を理解することができていなかった。	解決策	タスクボードを作成し、タスクを開発者ひとりひとりが自主的に管理するようにした。				

【大】ボトルネックのプロダクトオーナー
その他(管理)

状況	プロダクトオーナー役を決め、バックログの順番を決めていた	制約		結果		課題	
問題	プロダクトオーナーが多忙になり、状況に応じて仕事が止まるようになった。	解決策					

【大】大きな壁
その他(管理)

状況	タスクボードで管理することに決めた。	制約	タスクボードを置く場所には、冷蔵庫を始め、様々な物品があった。	結果	開発者全員のタスク状況を一目で理解できるようになった。実績を作ってから、会社に物品をどかしたことを説明した。	課題	
問題	タスクボードを置く場所がなかった。	解決策	許可なく冷蔵庫を始め、様々な物品をどかし、そこに大きなタスクボードを配置した。				

【大】プログラムから作られた仕様書
その他(管理)

状況	要件定義や状況がダイナミックであり、ドキュメントをベースに作っていない	制約	現物を見ながら面白さ(を含む要求)を追求するため事前に仕様として記述するのが困難である	結果	納品する仕様書とソースコードの乖離が少なくなった	課題	
問題	ソースコードが真であり、ドキュメントとの乖離があった	解決策	簡単な仕様書を開発を行い、動いているソフトウェアをベースに、仕様書を修正した。				

【大】適切な導入戦略
その他(管理)

状況	大量生産のチームと、企画者にフィードバックをもらいながら作るチームがあった	制約	大量生産のチームに、スクラムを当てはめるメリットが見いだせなかった	結果	チームの特性にあわせた開発スタイルになった	課題	
問題	ひとつのプロセスを当てはめることが不適切であった	解決策	適合するチームだけにスクラムを導入した				

Copyright © 2012 IPA, All Rights Reserved

【大】タスクボードの導入

その他（管理）

状況	管理するために、管理専門の担当者が開発者に状況をヒアリングして Excel や Microsoft Project を更新していた。	制約	進捗管理を行うと、その分の管理工数が増えてしまう。	結果	スプリントごとに期日があるため開発者はのんびりしなくなったため、開発工数が減り、開発速度があがった。	課題	数ヶ月レベルでのメリハリがなくなり、開発者は疲れてしまった。
問題	プログラマは週に一時間程度の進捗管理をしており、工数を取られていた。	解決策	全体が見渡せるような一般的なタスクボードで管理した。				

【大】探されたプロダクトオーナー

その他（管理）

状況	独自の方法でスクラムを導入し運用していた。	制約	おおよそプロダクトオーナー役は、ディレクタであると予想していた。	結果	プロダクトオーナーが実際にバックログの順番を決めることになった	課題	プロダクトオーナー役のディレクタは、多忙になってしまった。
問題	プロダクトオーナーの役割はなく、実施していた。	解決策	ディレクタにプロダクトオーナー役がなった				

【大】利用者との擦り合わせ

その他（管理）

状況	ユーザ企業の特定部署と、システム子会社、ベンダーの三者で開発を進めている	制約	要件にそぐわない場合は、業務側を変更する方針である。	結果		課題	現在も対応中である。
問題	ユーザ企業のエンドユーザの要件を聞かない方針で開発を進めた結果、ユーザの業務との乖離が大きくなりすぎてしまった。	解決策	ユーザとのレビューを開始し、要件の擦り合わせを開始した。				

【大】段階的朝会

その他（管理）

状況	チームが複数に分れている。各チーム毎に朝会をしている。	制約	部分の朝会と重複したくない。	結果	部分の朝会と、全体の朝会を実施することができる。	課題	
問題	朝会を全員でやると時間がかかりすぎてしまう。	解決策	各チームで朝会を実施し、その後に代表者だけの朝会を実施する。				

【大】必要なものだけドキュメンテーション

その他（管理）

状況	ドキュメントがあまりない状態である。	制約	無駄なものは作りたくない。	結果	本当に必要なもののみドキュメント化することができる。	課題	
問題	何をドキュメント化するかが決まっていない。	解決策	ソースをみてもわからないものはドキュメント化する。（業務フロー）				

【大】インセプションデッキ

その他（管理）

状況	プロジェクト全体でゴールやビジョンを共有したい。	制約	単に「書いてある」だけでなくメンバーが「積極的に見て共有する」ようにしたい	結果	プロジェクト全体での目的、目標が明確になった。	課題	
問題	プロジェクト憲章や計画書だけでは伝わりにくい。	解決策	プロジェクトの開始時にビジネス上達成すべきゴールやステークホルダー、リスク、アーキテクチャといった初期時点で考慮すべき点をプレゼンスタイルでチームで作成し共有する。				

Copyright © 2012 IPA, All Rights Reserved

【大】その他（管理）

チーム間のふりかえり

状況	各チームでふりかえりを実施し、問題やうまく行っている点が明かになっている	制約	全員でのふりかえりは困難である。	結果	プロジェクト全体で共有することができる。共通ルールなどに反映することができる。	課題	
問題	他のチームにふりかえりの内容が展開できていない。	解決策	各チームの代表者が、それぞれのふりかえりの内容を持ち寄って全体のふりかえりを実施する。				

【大】その他（管理）

失敗による問題の見える化

状況	開発協力会社でスクラムを導入していた。サポートを受けてスクラムを回してみた。	制約		結果	何がうまくいかなかったか問題を明らかにすることができた。	課題	
問題	最初はうまく行かなかった。	解決策	ふりかえりにより失敗の原因を特定した。				

【大】その他（管理）

WF への翻訳

状況	四半期ごとのマイルストーン定義について痛くされた受託開発である。	制約	タスクボードのまま報告すると、発注者の意思とギャップが生まれてしまう。	結果	発注元へは、WF の方法で報告することができた	課題	新たな工数発生や、スクラム→WF の翻訳ギャップがある
問題	週ごとの進捗を、マイルストーン定義に従って報告する必要があり、実際に運用されているタスクボードとのギャップがあった	解決策	付箋紙で書かれているタスクボードの内容を Microsoft Project や Excel で書かれたガントチャートやスケジュール表にあわせて、データを起こし、経験に基づき翻訳して記入するようにした				

【大】その他

アジャイルの弊害

状況	あらかじめ計画作りをしない方針で開発を進めていた	制約		結果		課題	
問題	そもそも何を作ろうとしていたのかがわからなくなっているのではないか	解決策					

【大】その他

試行錯誤ビジネス

状況	ビジネスが試行錯誤であり、あらかじめ予測することが困難である。	制約	新規開発の割合が少なく、ほとんどがソフトウェアの保守行程となっている	結果		課題	
問題	要件定義に時間をかけて、実装しても、正しく作られるとは限らない	解決策	ビジネスのフィードバックを得られるようにアジャイル開発を選択した				

Copyright © 2012 IPA, All Rights Reserved

中 徐々に分散

組織体制

状況	離れた拠点と分散開発をする必要がある	制約		結果	分散でも問題なく開発ができた。	課題	
問題	いきなり分散で始めてもうまくいかない。	解決策	最初は同じ拠点で仕事をした。その後で別拠点に分かれた。				

中 組織の壁

組織体制

状況	違う会社が合併してできてきた。見えない壁ができてきた。	制約		結果		課題	
問題	縦割りの組織体系だと横の組織の連係がうまくいかない。特に DBA、監査、セキュリティは融通が効かなかった。	解決策					

中 擬似的な壁面

組織体制

状況	近くに掲示できる壁面がない	制約		結果	タスク、ふりかえりの内容などの見える化がしやすくなった。	課題	
問題	掲示を貼り出したりする場所がなく、全体を見通しにくかった	解決策	ホワイトボード 3 面を使用し、掲示できる擬似的な壁面を作成した				

中 みんなが使いやすいツール

組織体制

状況	Redmine をチケット管理として導入しようとした	制約	Redmine はビジネス側にはとっつきにくい	結果	ビジネス側が積極的に使ってくれるようになった。	課題	
問題	影響などの関連を表現しづらく、社内で使われていた社内ツールがあった	解決策	すでに使われていたツール(サイボウズ デヂエ)を使って管理した				

中 顧客サポートチーム

組織体制

状況	お客様の要件や状況を引き出しながら、反復的に開発を進めている	制約	お客様の担当人数を増やすことは、困難であった。	結果		課題	
問題	お客様の担当者は2名しかおらず、確認や企画が十分にできなかった	解決策	お客様プロキシ(お客様側に1名、開発側に3名)を用意した				

中 チーム間ローテーション

組織体制

状況	複数のチームでサービスを開発している。	制約	特定のチームの中だけで知識が閉じてしまう	結果	各チーム間で知識が伝播できる	課題	一時的に開発速度は落ちる
問題	チームを跨いで知識が伝播しない	解決策	チームメンバーの交換をチーム間で行う				

中 一体のチーム

組織体制

状況	開発と企画が別々にいて、ドキュメントでコミュニケーションしていた	制約	企画が、開発を見下すような傾向があった	結果	場が作られ、コミュニケーションギャップが減った	課題	
問題	コミュニケーションギャップが大きかった	解決策	組織変更および席替えをし、企画と開発を一体のチームとした				

中 開発と監査のアンマッチ

組織体制

状況	システムは動作する状態の途中経過のものが存在する	制約		結果		課題	
問題	すべてのシステムが完成しないと監査やセキュリティの確認ができないと言われてしまった。	解決策					

Copyright © 2012 IPA, All Rights Reserved

【中】一人だけキーマン

組織体制

状況	POがすべてのレビューを受け持っている。	制約	トップダウンを避けた	結果		課題	
問題	POに負荷がかかってしまい、そこで物事が止ってしまうことが多くなった。	解決策					

【中】個人の責任希薄化

組織体制

状況	チームで一体となって仕事をしている。	制約	個人の責任で置きた問題もチームの責任になる。	結果		課題	
問題	個人の責任が薄くなってしまう。	解決策					

【中】コアメンバー不在

組織体制

状況	プロジェクトの要員は、専任ではなく、他の案件との絡みで調整が入る。	制約		結果		課題	
問題	プロジェクトメンバーの出入りが激しく、せっかく慣れたメンバーを外されてしまいコアメンバーが残らなかった	解決策					

【中】常時テレビ会議

コミュニケーション

状況	海外へオフショアをしている	制約	遠隔地であり、面と向かってのコミュニケーションが物理的に不可能	結果	コミュニケーションギャップを少しでも解消する取り組みである	課題	
問題	コミュニケーションギャップが発生している	解決策	常にテレビ会議システムが稼働し、手を挙げれば話ができるような状態にしている				

【中】全体で朝会

コミュニケーション

状況	フロアに全員が集ることができる。	制約	空気感も含めて伝えたい。	結果	伝達ミスが減った。	課題	
問題	特定の人が伝達役になると、うまく伝わらないことがある。	解決策	全チーム全メンバーで朝会に参加する。発表者はチーム毎一人にする。				

【中】IRC チャット

コミュニケーション

状況	顧客は別の拠点にいる。インフラチームは別フロアにいる。	制約	メールやSNSだと連絡手段としてはリアルタイム性に欠ける	結果	拠点を跨いでもコミュニケーションをとることができた。	課題	
問題	直線コミュニケーションをとりづらい	解決策	IRCで常に全員がリアルタイムでコミュニケーションをとれるようにした。				

Copyright © 2012 IPA, All Rights Reserved

中 一体となっていくチーム

コミュニケーション

状況	ビジネス側と開発側のフロア別だった。	制約	コミュニケーションコストが高いと、頻繁で密なコミュニケーションが取られにくい	結果	対面でコミュニケーションがとれるようになった
問題	フロアが別だとコミュニケーションコストが高く、コミュニケーションギャップが発生しやすい	解決策	徐々に引っ越しをして近づいていき、最後には一緒の島とした	課題	ビジネス側の障害などに対する発言で、開発側がモチベーションを下げてしまうことがあった。開発側がビジネス側を軽視してしまうことがあった。

中 チーム勉強会

展開

状況	チームはフロアに揃っている	制約		結果	各メンバーが学んだことを発表する機会ができた。技術的な発表資料は提案などにも利用できる
問題	メンバーに様々なことを学んでもらいたい	解決策	毎週 1 時間、チーム全体で勉強会の時間を設ける	課題	

中 プラクティスのエバンジェリスト

展開

状況	局所的にチームでプラクティスを実施した	制約	プラクティスの成功には経験が必要である	結果	経験が組織全体に伝播して行っている
問題	他のチームやプロジェクトにも広げたい	解決策	成功した経験者が他のチームに実地で教える	課題	経験者が掛け持ちし、負担が高まってしまった

中 頭は一つ

展開

状況	技術的に優れた人を複数人集めた。	制約	優れた技術者は拘りをもっており、妥協をしない。協調しての作業が苦手である。	結果	技術的議論で先に進まなくなることがなくなった。
問題	技術的な議論ばかりしていて、開発が進まない	解決策	技術的に優れたキーマンは一人にする。	課題	

中 チームを跨いだローテーション

展開

状況	チーム毎の担当する機能が異なる	制約	クライアントの知識と、サーバの知識は異なり交換がきかない	結果	すべてのチームが別のチームの知識を得ることができる。プロジェクト内でメンバーが交換可能になる。
問題	知識がチーム毎に固定化されてしまい、他のチームの技術的情報がわからない。	解決策	チームのメンバーを別のチームに移転して知識を伝播させる。	課題	一時的に開発速度は落ちる

中 メンバーの交換

展開

状況	アジャイル開発においては、前向きな態度と、助け合い、努力できる人材が求められる。	制約	ウォーターフォール型よりも、短い単位での作業の集中が求められる。チームで行動することが求められている。新しい技術を積極的に利用している。	結果	プロジェクトに合う人だけが残って、チームとしても、本人としても適切な場所で仕事ができる。
問題	やり方について行けない人がいた。	解決策	合わない人は、どんどん変えていく。	課題	

中 スクラム導入

展開

状況	ベテランをそろえコンサルも入れてウォーターフォール型開発をしていた	制約	途中で企画が変わり、IT は下請になってしまった。	結果	成功し、全社展開とした
問題	管理工数が増えて、巨大なトラブルを起こしてしまった	解決策	困難なプロジェクトにスクラムを導入した	課題	スクラム開発にそぐわない開発者や企画者が多い

Copyright © 2012 IPA, All Rights Reserved

中 展開

パイロット導入

状況	中規模案件でアジャイル導入を考えているが、リスクが不明である	制約	アジャイル導入および運用の勘所や経験がなかった。いきなり大規模に導入すると、失敗したときのリスクが増えてしまう。	結果		課題	
問題	中規模案件でアジャイルを導入し運用したい	解決策	小さなチームにアジャイルを導入し、経験を積んだ。				

中 展開

運用準備チーム

状況	リプレース案件で、リリース後新しいシステムを使っていただく	制約	要件が変わる、お客様の教育や業務を記述する時間がとれない	結果		課題	
問題	新しいシステムを初めて使う利用者は、使い方を知らない	解決策	運用準備チームを配備し、教育を行った				

中 展開

チーム人数の拡大

状況	プロジェクト開始当初は、人数が少なかったが、規模が増えている。	制約	いきなり人数を増やしてしまうと、業務もスキルも知らないので困る。	結果	最終的に1チーム8名×4チームとする。	課題	
問題	ひとつのプロジェクトで従事している人数を増やしたい	解決策	1イテレーションごとに人数を増やす。ただし、一度に2名程度にする				

中 展開

リーダーへの教育

状況	非 WF を組織として導入していきたい。リーダークラスが WF に慣れていて非 WF は馴染んでいない。	制約	メンバーが非 WF 経験者でもリーダーを説得することは難しい	結果	組織の中で、非 WF で開発できるチームが増える	課題	
問題	メンバーに非 WF 経験者が入ってもチームとしてはリーダーの慣れた WF しかできない。	解決策	リーダークラスを非 WF 教育する				

中 展開

ペア作業

状況	期単位で発注額が変ってしまい、予算が減ると人を放出しないといけない。	制約	新しく来た人に再教育しないといけないがその時間がない	結果	仕事の中でスキル、規律を学んで成長することができる	課題	
問題	教育している時間がない	解決策	ペア作業を実施する				

中 展開

プロジェクト退出

状況	決めてもらわないと動けない人がいる。仕事の仕方として受動的な仕事しかできない。	制約	プロジェクトの目的実現に力点が置かれている	結果	20～30%の人がスクラムのフレームワークに合わず、合う人だけがプロジェクトに残した	課題	
問題	スクラムフレームワークに合わない人がいる	解決策	合わない人はプロジェクトから外していった				

中 展開

コーチへの妄信

状況	スクラムのセミナー受講やコーチを導入している	制約		結果		課題	
問題	いいことばかりを言っている。	解決策					

Copyright © 2012 IPA, All Rights Reserved

中 危機の中での体験

展開

状況	50 人のプロジェクトが炎上していた。火消し役としてプロジェクトに参画した。	制約	組織にとって非常に大きな影響を与える位の赤字である	結果	リーダークラスにアジャイル手法のよさを体験できた。	課題	朝会しか身に付いていなかった。
問題	プロジェクトがまったくうまくいっていない。	解決策	アジャイル手法を用いて立て直した				

中 オフショアの導入

展開

状況	日本で B2B サービスを提供している	制約	日本人だけで開発している	結果	オフショア導入によって、コスト削減と規模の拡大ができた	課題	物理的には慣れているため、コミュニケーションギャップが発生してしまった
問題	人件費が高く、規模を拡大できない	解決策	中国にオフショアにした。 日本に来て、同じチームで仕事をした後、中国に戻り、他のメンバーを教育し、プロジェクトを実施している				

中 現場が伝達

展開

状況	スタッフ部門の人が新しい手法を現場に導入しようとしている	制約	現場に入らない人が言うことは聞いてくれにくい	結果	現場が自分達の仕事の中で非 WF を学び理解することができる。	課題	
問題	現場はスタッフ部門の言うことに耳を傾けない。	解決策	現場に経験者が入り実地で教える				

中

展開

状況	チーム内、チーム間の情報共有の結果、メンバーが多能工化している。人が抜けてもチームとしては前に進むことができる。	制約	人に依存しないようになっているがために、外部から見ると人を抜いても大丈夫と見られてしまう。	結果		課題	
問題	外部のプロジェクトのヘルプなどで人が抜かれてしまう。	解決策					

中 認定制度の活用

展開

状況	スクラムのフレームワークを導入したい	制約	ウォーターフォール型開発が前提であった	結果	スクラムへの理解が深まり、導入が容易になった	課題	
問題	社員がスクラムを理解していない	解決策	経営層から開発、クリエーターまで、大量に　認定スクラムマスターおよび認定スクラムプロダクトオーナーを取得した				

中 アーキテクチャチーム

アーキテクチャ

状況	全社的にスクラムのフレームワークを導入している	制約	中大規模の組織に展開したいが、アーキテクトを増やせない	結果	適切なアーキテクチャの選択ができている	課題	
問題	どのようなアーキテクチャが適切かわからない	解決策	アーキテクチャ専門チームがいて、チームへの相談や構築を行っている				

Copyright © 2012 IPA, All Rights Reserved

中 アーキテクチャ

アーキテクチャ

状況	プロジェクトの最初	制約	プロジェクトを通じて、アーキテクチャが変わると、アプリケーションの手戻りや修正が発生してしまう	結果		課題	
問題	アーキテクチャがなかった	解決策	半年ぐらいアーキテクチャの専門家によってアーキテクチャを作った				

中 システム分割

全チーム同じリズム

状況	複数のチームでサービスを開発している。	制約	クライアントとサーバ側ではリリースにかかる時間が異なる	結果	同じタイミングで同期可能になる。	課題	
問題	サービス全体としてリリースしたい。	解決策	反復の時期を合わせる。				

中 システム分割

見える化付きの朝会

状況	チームは毎日朝会をしている。	制約	口頭で内容を共有しているが、全員で共通で見られるものがない	結果	状況がよくわかり朝会の進行が飛躍的に向上した	課題	
問題	チーム内での情報共有が十分行なわれていない。	解決策	タスクボードを壁に作り、その前で朝会を実施した				

中 システム分割

基盤準備チーム

状況	メール配信などの事前に決められている機能が必要である	制約	他は反復を含む非ウォーターフォール型で開発している	結果		課題	
問題	あらかじめ要件が決定している	解決策	該当機能をウォーターフォール型で作る。非ウォーターフォール型開発チームのため、テストモジュール（スタブ）を入れる				

中 品質

テストのイテレーション

状況	プロジェクト最後の状態になってきた	制約	イテレーションとイテレーションで開発された間に複合的な問題が隠されている	結果		課題	
問題	品質が悪化していた	解決策	新規に機能を入れることをやめ、テストだけに集中するようになった				

中 品質

第三者確認

状況	ソースコードレビューというフェーズは設けていない。ペアプロは実施している。	制約	リーダーがレビューを実施するのは時間がかかる。	結果	多くの人の目で成果物を検証できる。全員が徐々に担当以外の部分を学習できる	課題	
問題	更に品質を向上させたい	解決策	第三者に完了したコードやテスト結果を確認してもらう。第三者責任を重くする。				

Copyright © 2012 IPA, All Rights Reserved

【中】100 点は対象外

品質

状況	フィードバックを繰り返しながら作りあげたいと考えている。品質についてはそれほど厳密なものを考えていない。	制約	品質において 100%を求めていない。	結果		課題	
問題	100 点を目指すなら WF の方がよいのではないか	解決策					

【中】未完成レビュー

品質

状況	タイムボックスに収まらずに動作するソフトウェアができていない	制約	完成したものでないと、他人に見てほしくない	結果	中途半端な状態でも実際の状況が確認できた。	課題	
問題	週単位で成果の進捗を見たいが、見せられるものがない	解決策	できていなくても、途中経過のものでレビューをする。週に二回レビューする				

【中】RD 契約で開発

その他（契約）

状況	顧客とプロジェクトの契約を結ばないといけない	制約	一括できちきち決めていると時間がかかってしまう。	結果	アジャイル開発がしやすいようになった。	課題	
問題	アジャイル開発がやりづらい	解決策	RD(要件定義)の形で契約をして納品物はなしにする。				

【中】裏委託契約

その他（管理）

状況	顧客とは一括請負で契約している。	制約	見積は出してもらわないといけない	結果	アジャイル開発がしやすいようになった。	課題	契約上はグレーのままである。
問題	パートナーと一括請負の契約を行うとアジャイル開発がしにくい	解決策	後で作業内容を調整して変更に対応させる				

【中】タスクボード

その他（管理）

状況	バグや要件がたくさんある	制約		結果		課題	
問題	状態を見える化したい	解決策	ツールを使い、バグと要件（バックログ）を整理した				

【中】丸見えな開発状況

その他（管理）

状況	一括契約の中で顧客と一体で開発を進めたい。	制約	一括であるがスコープなどに関して調整の余地をもたせたい	結果	顧客と強い信頼関係が構築できた。調整がしやすくなった。	課題	
問題	開発の状況が見えにくい	解決策	開発側のすべての情報を顧客が見られるように透明化する				

【中】悩みの共有

その他（管理）

状況	ふりかえりを一週間毎に実施している	制約		結果	新鮮な気持ちでふりかえりを実施できる。	課題	
問題	KPT ばかりだと飽きてしまう	解決策	様々なふりかえりの仕方を試してみる				

Copyright © 2012 IPA, All Rights Reserved

【中】その他（管理）

完璧よりも完了

状況	制約	結果	課題
まだバグが残っている。システムテストフェーズに入っている。	開発者はバグを修正しないとテストはできないと考えている。	開発者も納得してシステムテストに入ることができた。	
問題	**解決策**		
「もうシステムテストフェーズだから、バグがあってもテストを始める」と言っても、なかなか伝わらない	シンプルなキャッチフレーズで必要性を伝える。「完璧を目指すのはなく、まず完了させろ」		

【中】その他（管理）

チケットは確定のみ

状況	制約	結果	課題
チケットシステムを使用している。障害や追加要件が発生する。	チケットシステムのメンテで時間をとりたくない。	チケットシステムのゴミがなくなった。	付箋を大量に使用する。
問題	**解決策**		
チケットシステムに何でもいれてしまうと、いつまでたってもクローズされないゴミが溜ってしまう。	チケットシステムの前に付箋に内容を書き、優先順位をつけて実施が確定したものにのみチケットシステムに登録するようにする		

【中】その他（管理）

連係のための設計書

状況	制約	結果	課題
既存システムのリプレースであった	規模が大きいと複数のチームで仕事をする	設計を確実に実施できた	
問題	**解決策**		
規模が大きく全貌が掴みづらい	業務フローや設計書を作成して連係した		

【中】その他（管理）

気持ち週報

状況	制約	結果	課題
週報に作業内容の状況などを書いてもらっていた	タスクボードや日々の朝会などで状況が把握できる		
問題	**解決策**		
週報とスクラムのスプリントの活動が被ってしまう	週報はタスクなどではなく、個人の気持ちを書いて送ってもらうようにした。		

【中】その他（管理）

三段階朝会

状況	制約	結果	課題
チーム毎に朝会をやっている。全体でも内容を共有したい。	短時間で終わらせたい、全体での共有の後にもチーム単位で共有したい。	短時間で実現でき、全体朝会の内容についてもチームで考えることができた。	
問題	**解決策**		
個別朝会を先にやると、全体朝会のフィードバックに対応できない。全体朝会を先にやると、全体での共有に時間がかかる。	朝会を三段階にして実施する。最初はチームが全体に報告する内容をまとめる。次に全体で内容を共有する。最後に全体のフィードバックを元にチームで考える。		

【中】その他（管理）

解決できない問題

状況	制約	結果	課題
チームがふりかえりを実施しその中で問題を深堀している	解決できない問題を議論は、愚痴になりやすい	解決できる・できないに関わらず、ふりかえり時に議論できるようになった	
問題	**解決策**		
ふりかえり時にチームでは解決できない問題があがっていた	プロジェクトチームでやれること、やれないことを切り分けた。チームでどうにもならないことはいったん預かるようにした。		

Copyright © 2012 IPA, All Rights Reserved

中 その他（管理） 完全透明性

状況	顧客とは一括請負で契約している。信頼関係があれば様々な調整がしやすくなる。	制約	同じ拠点ではない。	結果	開発者の状況、悩みなどが顧客と共有することができ信頼関係を醸成することができる。	課題	
問題	開発側がブラックボックスだと顧客から見た時には不信感が生まれやすい	解決策	すべての開発に関しての情報を顧客に公開し隠し立てをなくす。				

中 その他（管理） クレドを作る

状況	プロジェクトが長期的になり、緊張が高い	制約		結果		課題	
問題	メンバーが疲弊してきた	解決策	クレドを作り、みんなの考えをまとめた				

中 その他（管理） 全体図を書く

状況	長期にわたるプロジェクトにおいて、非ウォーターフォール型開発を実施している	制約	全体像が明確ではない	結果		課題	
問題	全体像が見えなくなってくる	解決策	簡単な全体像を手書きで書くことによって把握する				

中 その他（管理） 多段式反復

状況	長期にわたるプロジェクトにおいて、非ウォーターフォール型開発を実施している	制約	通常の反復期間（二週間）では、見通し、見直すことができない	結果	大幅な影響を修正しつつリリースを迎えることができた	課題	
問題	長期的に適切な方向にすすみたい	解決策	一定期間（三ヶ月）にいったん大きなふりかえりと計画づくりを行った				

中 その他（管理） 全て丸みえ

状況	顧客と開発が同じ SNS で情報共有をしている。顧客との共有用と開発者専用のグループを分けていた。	制約	開発者は SNS に日記を書きたい。現状の不満なども書きたい。書き込みもしない人は一定数いる。	結果	顧客側から見て開発の状況がよくわかり、開発側への理解度が上がり顧客との調整がしやすくなった。	課題	
問題	開発者専用のグループに分けること自体がムダである。	解決策	顧客と開発で全く同じ書き込み情報を共有することにした。日常的な日記も公開した。				

中 その他（管理） 一行テスト

状況	Wiki 上に仕様書がある。それを元にマニュアルテストをしたい。	制約	エビデンスが欲しい。	結果	仕様書とテストの乖離がなくなった。	課題	
問題	仕様書からテスト項目を作ると、仕様変更の際にテスト項目も修正しないといけないが、修正モレの恐れがある。	解決策	仕様書を Word 文書にして、テストを実施する。実施した箇所は塗り潰していく。				

中 その他 ボトルネックのプロダクトオーナー

状況	スクラムの役割とスプリントを導入した	制約		結果		課題	
問題	プロダクトオーナーがボトルネックになっている	解決策					

Copyright © 2012 IPA, All Rights Reserved

中 スクラム文化の障壁

その他

状況	スクラムのフレームワークを導入した	制約	教育やルールシェアを実施している	結果		課題	
問題	企画の人が戸惑っている。WFのママである	解決策					

中 アジャイル耳タコ

その他

状況	メンバーはスクラムのフレームワークの経験が浅い	制約	慣れ親しんだやり方は安心できる	結果	以前のやり方に戻らずにスクラムのやり方を続けることができた。	課題	
問題	気がつくと以前のウォーターフォールのやり方に戻ろうとしてしまう	解決策	スクラムについて、考え方や、進め方を何度もいい続けた。				

中 裏帳簿

その他

状況	自分のやり方を貫きたい、表に課題を出したくない。	制約		結果		課題	
問題	課題管理帳を裏で作られてしまった。	解決策					

中 受け身の事業側

その他

状況	事業側が能動的に開発側と関りを持とうとしない。	制約		結果		課題	
問題	事業側が「できあがったものだけみる」という受け身の姿勢である	解決策					

中 ふりかえりのギャップ

その他

状況	ふりかえりを毎週実施している。	制約		結果		課題	
問題	ふりかえりの中で問題を深く考える人と浅すぎる人のギャップが大きかった。	解決策					

中 コーチへの盲信

その他

状況	外部からアジャイルコーチを招聘して実施している	制約		結果		課題	
問題	コーチのアドバイスを無条件に取り入れようとしてしまう	解決策					

中 すれ違うコミュニケーション

その他

状況	事業側と開発側は同じフロア、近くに移ってきた。ミーティングは一緒に行っている。	制約	情報を共有しているつもりだが、内容が伝わっていない。	結果		課題	
問題	ビジネス側と開発側の間の意思疎通がうまくいかなかった。議事録に書いてあることでも「言った言わない」という問題が発生した。	解決策					

中 信頼の欠如

その他

状況	事業側と開発側は同じフロア、近くに移ってきた。ミーティングは一緒に行っている。	制約		結果		課題	
問題	開発側が事業側を低くみるきらいがあった	解決策					

Copyright © 2012 IPA, All Rights Reserved

中 その他

見えないバックログ

状況	全員でバックログを書いていたが、事業部と話すにあたり、一人とりまとめる人を立てた。	制約		結果		課題	
問題	バックログがとたんに見えづらくなった(透明性が落ちた)	解決策					

中 その他

運用されないバックログ

状況	スクラムのバックログを導入した	制約		結果		課題	
問題	バックログがきちんと運用されていない	解決策					

中 その他

分割

状況	いくつかのベンダーがシステムを分割して実施している	制約		結果		課題	もしかして WF であれば、画面仕様を決めるので、そのようなことはなかったのかもしれない
問題	インテグレーションしたところ、インターフェース(画面のイメージ)がバラバラになってしまった	解決策					

中 その他

アジャイルの選択

状況	市場を調査しながら、まだないサービスがある	制約	そのサービスが市場で受け入れられるかわからない	結果		課題	
問題	市場に提供したことがない新しいサービスを提供する	解決策	アジャイル型で、市場のフィードバックをもらいながら開発する				

中 その他

アジャイルの選択

状況	ビジネス企画とシステム開発が調整しながら実施している	制約		結果	データ設計が苦手な人でもうまく行くのではないか	課題	
問題	ビジネス企画側が要件定義書を記述できない	解決策	モックやラフスケッチをみながら開発した				

中 その他

アジャイルの選択

状況	ウォーターフォール型とアジャイル型開発の品質について比較している	制約	イテラティブに反復的に開発されている	結果	その時点での品質は担保されている	課題	
問題	アジャイルの品質管理について	解決策	イテレーションで、動くものを作るようにした				

中 その他

アジャイルの選択

状況	インターネットのウェブシステムについて開発および保守開発をしている	制約	ウェブシステムは「もやもや」しているのを見ながら進めるのがよい	結果	マネジメントシステムがとてもよい(Pivotal Tracker)。工数と要望がイコールであり、ユーザと開発が納得していけるのがよい。	課題	
問題	アジャイルを導入しようか悩んでいる	解決策	リスクが少ない部分についてアジャイル開発を導入した				

Copyright © 2012 IPA, All Rights Reserved

【中】その他

アジャイルの選択

状況	顧客データを扱う基幹システムと、お客様が毎日触るようなフロントシステムがある。パッケージシステムを利用している	制約		結果		課題	
問題	どのような開発スタイルがいいのかを検討する	解決策	基幹システムやパッケージシステムは、ウォーターフォール型開発とし、フロントシステムをアジャイル型で進める。巨大なDB 設計やチェックルーチンなどが入るような開発にはアジャイルは向かないと判断した				

【参】組織体制

経験者の確保

状況	素早く開発しないといけない。非ウォーターフォールのスキームを利用することは決まっている。利用するフレームワークも決まっている。	制約	経験者でないと、プロセスやフレームワークの両面で立ち上がりが遅れてしまう。	結果	立ち上がりのオーバーヘッドを削減してスタートダッシュできる	課題	
問題	素早くプロジェクトを立ち上げたい。	解決策	半数近くを非ウォーターフォール型開発経験者にする				

【参】組織体制

一体となるチーム

状況	開発側が、協力会社である。	制約		結果	顧客と開発側が、対等にコミュニケーションがとれる	課題	
問題	力関係上、顧客に対して、積極的なコミュニケーションがとりづらい	解決策	顧客と開発者の間を取り持ち円滑なコミュニケーションを支援するロールを顧客と同じ会社の人間で設けた				

【参】コミュニケーション

要件確認会

状況	素早く開発しないといけない	制約	ドキュメントを作るのは必要だが、時間がかかる	結果	中間成果物が不要で素早く開発に進むことができる。要件定義者と開発者の間の伝達が不要になる。	課題	
問題	要件定義をする人と、開発する人が分かれると知識移転のオーバーヘッドが高い	解決策	週に 6～8 時間を要件確認の時間にあてて、顧客から開発まで関係者が一同に揃って内容を共有する				

【参】コミュニケーション

チーム要件定義

状況	要件を顧客との間で明確にして定義する必要がある。	制約	要件はコーディングする人が理解しないとけない	結果	ドキュメント作成分の時間が節約できた	課題	20 人を越えると難しくなる
問題	要件定義者と開発者が別々だとコミュニケーションのためのドキュメント作成に時間がかかってしまう	解決策	要件の確認検討はチーム全員でやってしまい中間成果物は作らないようにする				

【参】展開

結果駆動での展開

状況	新しいスキームを社内に展開したい。	制約	説明だけでは伝わらないし、新しいスキームを全体で適用するとリスクが高まる	結果	100 人規模にまで広めることができた。	課題	
問題	誰もその良さを知らない。文化的に異質である。	解決策	パイロット案件で試してその実績を得る。「生産性倍、期間半分」というキャッチフレーズを用いて社内で広告する				

Copyright © 2012 IPA, All Rights Reserved

参 ウォーターフォール・チューニング

展開

状況	サービスが順調に成長して、規模が拡大してきた。求められる品質レベルも高くなってきた。	制約	市場への公開スピードよりも、安定性が求められる。	結果		課題	速度は遅くなる。
問題	80%ルールでは求められる品質を満たせない。	解決策	要件を固めた上で、文書化して進めるようにする。チーム間の調整もドキュメントをベースとする。				

参 小さく生んで大きく育てる

展開

状況	競合他者に先んじたサービスを提供したい	制約	ヒットするかどうかはリリースしてみないとわからない	結果	無駄な投資をすることなく、効率的なサービス展開を実施できる	課題	
問題	いきなり予算を掛けてサービスを構築することはできない	解決策	小さく素早くリリースして、経営的な判断を行う。規模を拡大していく。あるタイミングで再構築を実施する				

参 スケール可能な標準アーキテクチャ

アーキテクチャ

状況	他の製品やプロダクトがあり、動作している	制約	ミドルウェアやフレームワークが組織を通じて共通化されていると使いやすい	結果	基盤構築の時間が省け、提供する品質を一定化し、高速に開発することができる。	課題	フレームワークに精通する人材を確保することが不可欠になる。
問題	納期が短期なので基盤などを作りこむ時間がとれず、ミドルウェアやフレームワークの標準化をしたい	解決策	サービスの実現に必要な基盤として、データベース、ミドルウェア、フレームワークとして実現しておく。スケール可能なアーキテクチャにする。				

参 フレームワーク・サポート体制

アーキテクチャ

状況	フレームワークを使って高速に開発したい。	制約	フレームワークの使い方を知る必要がある	結果	現場で起きたフレームワークに関するトラブルを早期に解決できる	課題	
問題	開発中にフレームワークについて調査し、学習する時間をかけられない	解決策	社内にフレームワークのサポートセンターを用意しておく。現場に出張チュートリアルをする				

参 適度な品質

品質

状況	素早く開発し、市場優位を得たい	制約	QCD で、品質と提供スピードのバランスがある	結果	素早く開発できている。ただ、結果として品質もそれほど悪くならず、目標品質よりも高くなるケースも多い。	課題	
問題	品質を上げようとしすぎると時間がかかる	解決策	品質レベルを低いことを納得していただくことで、市場への提供時間を早くした。				

参 80%ルール

品質

状況	重要度の高いものから早めに市場に提供したい	制約		結果	素早く開発を前に進めることができる。	課題	80%ルールを運用することが人によって難しい
問題	100%を目指して物事を決めようとすると時間がかかる	解決策	80%決まっていれば前進できるようにルール付けしておく				

Copyright © 2012 IPA, All Rights Reserved

参 方針とリスクの事前告知

その他（契約）

項目	内容	項目	内容
状況	社内のコンペでアジャイルを提案して案件を受注したい	制約	顧客は社内の別グループである。外部の業者とのコンペになる。
問題	非ウォーターフォールを用いるとスコープが調整対象となり契約時点の内容と異なる可能性がある	解決策	契約時に「非ウォーターフォールなので変更する可能性があります」と最初に伝えておく。
結果	事前に非ウォーターフォールの制約を認識してもらうことで、後で問題にならないようにする	課題	社内だからできると考えられる。

参 準委任契約パートナー

その他（契約）

項目	内容	項目	内容
状況	スクラムのフレームワークを導入して開発したい	制約	顧客との間は一括だが、開発に協力会社を頼む必要がある
問題	請負契約だとチームとして連係して作業が難しい	解決策	開発パートナーとの間は業務委託(準委任)契約を結ぶ
結果	チームとして柔軟に開発することができる	課題	顧客側との契約は一括なので、そことの整合性をとる必要がある

参 残業バッファ

その他（管理）

項目	内容	項目	内容
状況	スクラムのスプリントで作業を進めている	制約	
問題	メンバーの作業時間が平準化されていない	解決策	残業の予想をしておき、必要あればバッファから消費して対応する
結果		課題	

参 親方バッファ

その他（管理）

項目	内容	項目	内容
状況	タイムボックスを用いて、期間を固定し、スコープを調整可能にしている	制約	計画が変わるのを許容したいが、どこまで許容できるかがわからない。
問題	当初の計画をたてても、うまくその通りにいかない。	解決策	スケジュール全体に CCPM の親方バッファを設けて、バッファでマネジメントする
結果	バッファが分散せずに、効率よくスケジュール管理ができる。	課題	

参 リリースによるスコープの調整

その他（管理）

項目	内容	項目	内容
状況	サービス公開までに時間があまりない	制約	
問題	リリースの期日は変えられない	解決策	一週間を固定の期日にしてその中で実現する要件のスコープを調整し動くソフトウェアを実現する。
結果	最終的期日を動かすことなく、週単位で動くソフトウェアが実現できる	課題	

参 スケールに応じた要件定義書

その他（管理）

項目	内容	項目	内容
状況	サービスが軌道に乗ると徐々に規模が拡大し開発者も増えていく	制約	多くの人数に情報を伝えたいが、直接伝えることが難しい
問題	人数が多いと全員で要件検討会を実施するのが困難になる	解決策	要件定義者が要件をドキュメントとしてまとめて開発に伝えてコミュニケーションを円滑にする
結果	一度に要件定義会に集まらずとも、要件の共有ができる	課題	

Copyright © 2012 IPA, All Rights Reserved

参 ケーススタディ

その他

状況	非ウォーターフォール型の考え方に馴染がない人が多い	制約	納得して取り込むことは難しい。	結果	単なる説明よりも、理解しやすく、その目的、効果などが理解できる。	課題	
問題	単に説明するだけでは、なかなか伝えられない	解決策	単なる説明だけでなく、ケーススタディを用いて具体的なケースでのふるまいとその理由を関係者に説明する。				

参 プロジェクト退出者

その他

状況	独自の非ウォーターフォール手法を前提に開発要員を募集する	制約	すべての要員が非ウォーターフォールの経験者ではない	結果	プロジェクトの速度を落すことなく目的を達成することができる	課題	
問題	やり方について不適合者がいる。	解決策	合わない人はプロジェクトから抜けてもらうようにする				

参 リリースバーンダウンチャート

その他

状況	現在の進行状況を確認把握したい	制約	リーダーやマネージャの管理工数がかかってしまう	結果	メンテナンスコストが削減して、状況が把握しやすくなった。	課題	
問題	ガントチャートを使って進捗管理すると工数が掛りすぎる	解決策	バーンダウンチャートを使いマイルストーンまでの進捗状況を可視化する				

参 スモールスタートダッシュ

その他

状況	ネットで競合他者との競争が激化してきた。	制約	システム複雑化でトラブルが発生し顧客に被害を与えてしまう	結果	生産性が倍になり、速度が半分になった。	課題	
問題	要件や仕様の完璧を期すると策定に時間がかかってしまう	解決策	品質はほどほどにスピードを重視して開発しスモールスタートで素早く公開する				

Copyright © 2012 IPA, All Rights Reserved

付録3：調査票

今回の調査で調査先各社に回答を求めた調査票の雛形を添付する。
責任者(プロジェクトマネージャ層)向け、牽引者(プロジェクトリーダー層)向け、メンバー向けの以下の 3 種類の調査票を用意した。

・ 背景調査票
・ 実態調査票
・ 個人意識調査票

Copyright © 2012 IPA, All Rights Reserved

■ **背景調査票**

プロジェクトの責任者に対し、非ウォーターフォール型開発導入の背景等を中心に以降の質問を行った。

【非ウォーターフォール型開発背景調査票　項目】

項目		質問事項
導入背景/動機/狙い/効果	適用検討までの背景	非ウォーターフォール型開発に至るまでの背景を詳しく記述してください。
	解決したかった問題	非ウォーターフォール型開発によって解決したかった問題を記述してください。複数ある場合は、優先順位で並び変えてください。
	導入における制約、考慮した点	非ウォーターフォール型開発を導入検討する上で、制約となったこと、あるいは考慮しなければならなかった点を記述してください。
	適用を判断した理由、その条件	非ウォーターフォール型開発の適用を判断した理由や、その条件を記述してください。
	適用前の期待	適用前に非ウォーターフォール型開発の適用によって期待していた効果を記述してください。前述の問題解決以外にも期待していた点があれば記述してください。
	実際に得られた効果、その評価方法	適用前の期待と比べて、実際に得られた効果を記述してください。定量的な数字があればそちらを記述してください。
	新たに生じた課題	適用後に発覚した新たな課題があれば記述してください。
全体像と歴史的経緯	歴史的経緯	実際の非ウォーターフォール型開発の適用前〜適用後の間の全体的な歴史的経緯（評価、適用、展開、開発モデルの変更、管理手法の見直し、知識体系、アーキテクチャの導入、など、大きな変更点）を記述してください。
	ターニングポイント	非ウォーターフォール型開発を適用して実施していく際で、ターニングポイントになった出来事（時期、内容とその効果）を教えてください。複数個あればすべて記述ください。「当時はそう感じていなくても、今考えると、あの出来事がターニングポイントだった」というものでも結構です。
	展開状況	プロジェクト全体に対しての非ウォーターフォール型開発の展開状況を、全体の％、人数、チーム数、適用分野などの観点で記述ください。

Copyright © 2012 IPA, All Rights Reserved

■ **実態調査票**

プロジェクトリーダ等の主にプロジェクトの牽引者に対して、非ウォーターフォール型開発における実態調査のため、以降の項目それぞれに対して以下の質問を行った。

① 実態・実績・利用しているプラクティス
② 独自の工夫
③ 成功度への貢献度
④ 取組に至る背景、考慮した制約
⑤ 解決したかった問題点
⑥ 非ウォーターフォール型開発によって体験した利点／課題
⑦ 実施開始／終了のタイミング

【非ウォーターフォール型開発実態調査票 項目】

項目		質問事項
対象ドメイン	アプリケーションシステムタイプ	顧客問題解決ソフトウェア（顧客の要望を受けて開発するソフトウェア）であるか、ソフトウェアプロダクト (ソフトウェアフレームワーク、プラットフォーム、パッケージソフトウェア、組込みソフトウェア等) であるか
	アプリケーションシステムの目的	本システムの目的
	マーケットサイズ	対象アプリケーションシステムの市場規模 (年間あたり)
	アプリケーションシステムの重要度	対象アプリケーションシステムの重要度レベルを選択してください(アプリケーションシステムに欠陥があった場合の損失により定義)
	ドメインの変更に関する特徴	業務ドメインの特徴を主に変更に関してご回答ください。
	目的実現の際に優先した ビジネス価値	以下の例を参考に、優先したビジネス価値の上位３つを順に記述ください。売り上げを第一優先にしたい場合は「何を重要視すると売り上げが伸びるか」という観点でお答えください。 市場投入時間の短縮、 製品の有用性の向上、 品質の向上、 進行状況の可視性の向上、 変化への柔軟性の向上、 コスト削減、 製品寿命の向上、 その他(内容を記述)
	ビジネス価値の効果	優先したビジネス価値に対して、どのような効果があったのかを可能な限り定量的に記述ください。
システム属性	規模	ユースケースの数、ユーザ ストーリー数、ステップ数等の指標を用いて記述
	システム環境	プラットフォーム、ソフトウェアフレームワーク等、開発したシステム環境について記述

Copyright © 2012 IPA, All Rights Reserved

項目		質問事項
	ソフトウェア／ツールテクノロジー	開発したシステムに組み込まれているソフトウェア、ツール、テクノロジーについて記述
	プラットフォーム	動作プラットフォームについての特徴を回答ください(単一プラットフォーム、マルチプラットフォーム、既存サービスとの連係あり、etc)
	開発言語	Java、C、Ruby、等
プロジェクト特性	プロジェクト期間	プロジェクトの開始から終了までの期間
	プロジェクト費用	アプリケーションシステム開発に要した費用
	プロジェクト工数	人月単位で記述
	プロジェクト初期における要件の確定度合い	最初の設計に入る段階において要件がどの程度決まっていたか
	1ヶ月あたりの要求変更の割合	全体の要求数に対して、変化した要求の数の割合
	市場投入時間	仕様策定後、製品を市場に投入するまでの期間
	要求された稼働率	アプリケーションシステムに要求された稼働率（運転時間内においてどれくらいの割合で稼動しているか）を記述
	新規性 (ビジネス)	ビジネスの新規性についてご回答ください。
	新規性 (技術)	開発時に利用した技術の新規性や技術に対する経験の度合いを回答ください。
	プロジェクト関与者人数	プロジェクトに関与した人の人数を回答ください。(実人数であって、のべ人数ではありません)
	プロジェクト関与者の内訳	関与者の役割毎の内訳をお答えください。
	男女比	関与者の男女比率をお答えください。
	プロジェクト制約条件	プロジェクトにおいて、品質(Q)、コスト(C)、時間(D)、スコープ(S)の制約条件をお答えください。
	コンプライアンス	システムを開発する上で遵守すべきコンプライアンスなどがあれば記述してください。(ISO9000、SOX、など)
	開発拠点	開発拠点の場所と、そこいる人について記述してください。同一拠点で同フロアに集っていた、同一拠点だがフロアが異なっていた、全く別の拠点に分散していた、など。(場所もお願いします)
プロジェクト組織特性	プロジェクトに関与した企業と契約関係	プロジェクトに参画した企業と、その間の契約形態についてご回答ください。(顧客との間、開発企業間、コンサルティング会社、など)
	プロジェクト内組織の内訳	同一組織内の一部署だけ、異なる部署が混在している、パートナー企業が参加している、一部を外部にアウトソースしている、などを回答ください。

Copyright © 2012 IPA, All Rights Reserved

項目		質問事項
	組織内の評価制度	個人を適正に評価する仕組みがあるかどうかご回答ください。それによって個人がどの程度モチベーションが高まっているかどうかもわかる範囲で記載ください。
	組織の柔軟性	プロジェクト内の柔軟性の度合いをリストから選択ください。
チーム特性	チーム編成	チームをどうやって立ち上げたか、どうやって人を集めたか、チーム内の役割をどう割当てたか、についての工夫を教えてください。
	チームあたり人数	チームあたりの平均人数をご回答ください。正確な数字がある場合は、それぞれのチームの人数を記述ください。
	チームに対しての役割分担	各チームにどのような役割を与えていたかを詳しく記述してください。インフラ、DBA、UI のようなレイヤ構造でチームに割り振った、あるいは、各サブシステムの機能単位でチームに任せた、など。
	チームにおける、「平均レベル以下の、経験は少ないが、勤勉な開発者」の割合	「別表 1」ワークシート記載の表におけるレベル 1B の人の割合
	チームにおける、「非ウォーターフォールまたはウォーターフォール型開発のプロジェクトをマネジメントできる人」の割合	「別表 1」ワークシート記載の表におけるレベル 2 および 3 の割合の合計
成功度	全体の成功度	プロジェクトの成功度合いを表現すると、5 段階でいくつになるでしょうか？　前述の「ビジネス価値」やその他考慮点などを踏まえて、5 段階でお答えください。
	成功の尺度	上記成功度を測る成功の尺度に何を用いているか、具体的な定量値があればそちらも回答ください。複数あれば複数ご回答ください。 (QCD、それ以外も)
システムオーナと開発プロジェクトマネージャとの分担関係		事業責任、ガバナンス、組織の歴史・環境・文化という視点から、分担関係を記述 （システムオーナとは、開発対象システムの企画・投資に関して責任を持つ職責者を指す）
プロジェクト進行の概要 (特にスピード、要求や市場の変化への対応の視点から記述)	ライフサイクルモデル	各プロセスにかけた期間や人月、イテレーションを行ったプロセス・回数・期間、等を記述
	リフレクション/レトロスペクティブ(反省会/ふりかえり)	プロジェクトにおいての、リフレクション/レトロスペクティブを用いた改善の頻度、改善の内容や効果について回答ください。
	プロセス間のコラボレーション	プロセス間でコラボレーションするために実施したこと、利用したツール、等を記述

Copyright © 2012 IPA, All Rights Reserved

項目		質問事項
	システムオーナと開発プロジェクトマネージャとの協調	システムオーナと開発プロジェクトマネージャがどのような責任分担において互いに協調したか、等を記述
個人特性 （スキル/教育）	顧客	・顧客の属性やスキル（コミュニケーション能力があるか、協力的か、顧客の意思をきちんと代表しているか、権限を持っているか、献身的であるか、知識を持っているか、等） ・顧客に対してどのような教育を行ったか ・非ウォーターフォール型開発の経験度合い、習熟度合い
	開発／保守者	・開発／保守者の属性やスキル（コミュニケーション能力があるか、開発経験、変化する状況に対応する能力、等） ・開発／保守者に対してどのような教育を行ったか ・非ウォーターフォール型開発の経験度合い、習熟度合い
	運用者	・運用者の属性やスキル（コミュニケーション能力があるか、運用経験、等） ・運用者に対してどのような教育を行ったか ・非ウォーターフォール型開発の経験度合い、習熟度合い
	その他の役割(デザイナー、ドメインエキスパート、など)	・属性やスキル（コミュニケーション能力があるか、開発経験、変化する状況に対応する能力、等） ・どのような教育を行った ・非ウォーターフォール型開発の経験度合い、習熟度合い
	開発プロジェクトマネージャ／生産管理者	・開発プロジェクトマネージャ／生産管理者の属性やスキル（コミュニケーション能力があるか、開発経験、変化する状況に対応する能力、マネジメント能力、等） ・開発プロジェクトマネージャ／生産管理者に対してどのような教育を行ったか ・非ウォーターフォール型開発の経験度合い、習熟度合い
利用したプロジェクト管理手法(見える化も考慮してください。よく知られたプラクティスを利用した場合には、その名前を記述ください。)	統合管理	プロジェクトのさまざまな要素を調和のとれた形に統合するのに必要なプロセス。 プロジェクト憲章作成、プロジェクト・スコープ記述書暫定版作成、プロジェクトマネジメント計画書作成、プロジェクト実行の指揮・マネジメント、プロジェクト作業のコントロール、統合変更管理、プロジェクト終結のプロセスからなる。
	スコープ管理	プロジェクトを成功のうちに完了するために、プロジェクトに必要とされる全ての作業を含み、かつ、必要とされる作業のみを含んでいる

Copyright © 2012 IPA, All Rights Reserved

項目		質問事項
		ことを確実にするために必要なプロセス。 スコープ計画、スコープ定義、WBS 作成、スコープ検証、スコープ・コントロールのプロセスからなる。
	スケジュール管理	プロジェクトを所定の時期に完成させるために必要なプロセス。 アクティビティ定義、アクティビティ順序設定、アクティビティ資源見積り、アクティビティ所要期間見積り、スケジュール作成、スケジュール・コントロールのプロセスからなる。
	コスト管理	プロジェクトを承認された予算内で完了させるために必要なプロセス。 コスト見積り、コスト予算化、コスト･コントロールのプロセスからなる。
	品質管理	プロジェクトが意図するニーズを満足させることを確実にするために必要なプロセス。 品質計画、品質保証、品質管理のプロセスからなる。
	組織・要員管理	プロジェクトに関与する人々を最も効果的に活用するために必要なプロセス。 人的資源計画、プロジェクト・チーム編成、プロジェクト・チーム育成、プロジェクト・チームのマネジメントのプロセスからなる。
	コミュニケーション管理	プロジェクト情報の生成、収集、配布、保管、廃棄をタイムリーかつ確実に行うために必要なプロセス。 コミュニケーション計画、情報配布、実績報告、ステークホルダー・マネジメントのプロセスからなる。
	リスク管理	プロジェクトのリスクを識別し、リスクに対応することに関するプロセス。 リスク・マネジメント計画、リスク識別、定性的リスク分析、定量的リスク分析、リスク対応計画、リスクの監視コントロールのプロセスからなる
	外注管理	プロジェクト母体組織の外部から物品やサービスを取得するために必要なプロセス。 購入・取得計画、契約計画、納入者回答依頼、納入者決定、契約管理、契約終結のプロセスからなる。
中大規模特性	アーキテクチャ構築プロセス	システムアーキテクチャの設計、構築、変更について実施した工夫、参考にした手法などを回答ください。
	アーキテクチャ特徴	アーキテクチャに求めた要件を記載してください。
	システム間インテグレーション	サブシステム間や外部システム間でのインテグレーションについて、実施した工夫や、考慮し

Copyright © 2012 IPA, All Rights Reserved

項目		質問事項
		た点などを回答ください。
	ドキュメンテーション	どのような情報を文書化したのか、逆にしなかったのか、その範囲と、工夫した点について教えてください。
	目的・ビジョン共有	大規模の開発関与者間で、システム全体の目的やビジョンの共有について工夫、考慮した点などを記述
	組織文化	非ウォーターフォール型の適用によって、文化的に大きく変った場合、それについての工夫などがあれば回答ください。
	部分適用	非ウォーターフォール型開発を部分的に適用している場合、従来型開発と、どのような条件で切り分けているのか、その時にどのような工夫をしているのかをご回答ください。
	プロジェクト内での展開の過程	非ウォーターフォール型開発を、プロジェクトにどのように展開していったのかを、時系列、対象人数、チーム、分野などの観点で記述ください。
	チーム間連係	チーム間で情報共有や、作業の連係、支援などについて、どのような工夫をしていたかを回答ください
	分散拠点	分散拠点での開発の場合、どのような工夫をしていたかを、具体的にご回答ください。
	異企業間開発	異企業間の合同プロジェクト、あるいは多くのパートナー企業とのプロジェクトの場合、どのような工夫をしていたかを、具体的にご回答ください。
利用した開発・保守手法(自動化も考慮する)	要求形成／追跡	オブジェクト指向分析やトレーサービリティ技術の利用等、要求形成／追跡に関して行ったこと、適用した手法、等を記述
	設計	コンポーネント利用設計やデザインパターンの利用等、設計に関して行ったこと、適用した手法、等を記述
	構築	テスト駆動開発や自動コード生成ツールの利用等、実装に関して行ったこと、適用した手法、等を記述
	テスティング／V&V	リグレッションテストツールの利用やブラックボックステストの実行等、テスティング／V&Vに関して行ったこと、適用した手法、等を記述
	デプロイ、リリース	システムの配備、リリースについて行ったこと、工夫を記述ください。
	メンテナンス	開発したアプリケーションシステムに対して行ったこと、適用した手法、等を記述

Copyright © 2012 IPA, All Rights Reserved

■ 個人意識調査票

個人（メンバー）に対して、非ウォーターフォール型／ウォーターフォール型開発それぞれの経験手法での意識調査として以下の 2 点の質問を行った。

① プロジェクト組織について（採点方式）
② 個人の気持ちについて

① プロジェクト組織についての質問項目

（各項目に対し、WF・NWF 型開発それぞれの小計が 100 ポイントとなるよう配点。）

1.支配についての特徴		WF 型	NWF 型
A	組織はとても暖かな場所である。まるで家庭のようだ。そこにいる人々はお互いのことをよく分り合えているようだ。		
B	組織はとてもダイナミックでかつ新しいビジネスが生まれるような場所である。人々は危険に身を晒し、リクスを取ることを喜んでする。		
C	組織はとても結果指向である。第一の関心事は仕事を完了させることである。人々はとても競争的でかつ業績指向である。		
D	組織はとても管理的で階層的な場所である。公式の手順が大抵の人々の行動を決めている。		
	小計(合計が 100 になるように記述する)	0	0

2.組織のリーダーシップについて		WF 型	NWF 型
A	組織内のリーダーシップでは、メンタリング、ファシリテーション、人の育成の実現を考える。		
B	組織内のリーダーシップでは、起業家精神、物事の革新、リスクを負ってもこれらの実現を考える。		
C	組織内のリーダーシップでは、現実的、積極的、結果指向への集中、これらの実現を考える。		
D	組織内のリーダーシップでは、調整、組織編成、または円滑な運営効率の実現を考える。		
	小計(合計が 100 になるように記述する)	0	0

3.従業員のマネジメント		WF 型	NWF 型
A	組織のマネジメントスタイルは、チームワーク、合意形成、参加意識、という言葉で特徴づけられる。		
B	組織のマネジメントスタイルは、個人の危険を厭わない態度、イノベーション、自由、個性、という言葉で特徴づけられる。		
C	組織のマネジメントスタイルは、猛烈な競争意識、高いレベルの要求、達成や業績、という言葉で特徴づけられる。		
D	組織のマネジメントスタイルは、安全な業務、従順さ、予測性、安定した関係性、という言葉で特徴づけられる。		
	小計(合計が 100 になるように記述する)	0	0

Copyright © 2012 IPA, All Rights Reserved

4.組織を繋ぎとめるもの		WF 型	NWF 型
A	組織を団結させているものは、忠誠と相互信頼である。この組織に対してのコミットメントは非常に高い。		
B	組織を団結させているものは、イノベーションや開発へのコミットメントである。最先端であり続けることを重要視している。		
C	組織を団結させているものは、業績や目標達成の強調である。積極性と勝利は共通テーマである。		
D	組織を団結させているものは、公的な規則と方針である。円滑な組織運営の維持つづけることが重要である。		
	小計(合計が 100 になるように記述する)	0	0

5.戦略的力点		WF 型	NWF 型
A	組織は人材開発を重要視する。高い信頼関係、開かれた関係性、人々の参加しやすさ。		
B	組織は新しい資質の獲得と、新しい挑戦の創造を重要視する。新しい物事への試行や、機会の探索に価値を見出す。		
C	組織は競争的な行動や業績を重要視する。広げた目標へ命中させること、市場で勝利することが最高である。		
D	組織は永続性や安定を重要視する。効率、管理、円滑な実行は重要である。		
	小計(合計が 100 になるように記述する)	0	0

6.成功のモノサシ		WF 型	NWF 型
A	組織が定義する成功とは、人材の開発、チームワーク、従業員のコミットメント、人々への気づかいに基づくものである。		
B	組織が定義する成功とは、類いなき個性または最新の製品を持つことに基づくものである。つまり、製品リーダーや革新者である、ということである。		
C	組織が定義する成功とは、市場で勝利し、競合を凌ぐことに基づくものである。競争的市場でのリーダーシップが鍵となる。		
D	組織が定義する成功とは、効率に基づくものである。信頼のおける受け渡し、円滑なスケジューリング、低コストの生産、これらが決め手である。		
	小計(合計が 100 になるように記述する)	0	0

Copyright © 2012 IPA, All Rights Reserved

② 個人の気持ちに対しての調査項目

回答者情報		WF型	NWF型
1	そのプロジェクトは何ヶ月前に完了しましたか？ 現在も仕掛かり中のプロジェクトは 0（ゼロ)をお書きください。		
2	あなたのプロジェクトにおける立場を教えてください		
3	プロジェクトの規模（人月）をお書きください		
4	一年あたりプロジェクトや組織での改善頻度をおしえてください（毎日の場合は 365 を記入） 開始した当初から変更がない場合は 0(ゼロ)をご記入ください		
5	要件・仕様の変更や訂正が何パーセントぐらいありましたか 開始した当初から変更がない場合は 0(ゼロ)をご記入ください		
6	一年間あたりのリリース頻度はどのぐらいでしたか（計画を含む、毎日の場合は 365 を記入）		

プロジェクトのことを思い浮かべながら、あなた自身の認識や気持ちを下記の選択肢の中からお書きください。 （4. あてはまる、3.どちらかといえばあてはまる、2.どちらかといえばあてはまらない 1.あてはまらない）		WF型	NWF型
1	あなたは、プロジェクトは順調である（成功した）と思う		
2	あなたは、プロジェクトに満足している		
3	誰かに助けてもらったら、自分もまた他の誰かを助ける		
4	人から親切にしてもらった場合、自分も誰かに親切にする		
5	助け合っている人々を見ると、自分も困っている人を助けようという気持ちになる		
6	ほとんどの人は信頼できる		
7	ほとんどの人は他人を信頼している		
8	ほとんどの人は基本的に善良で親切である		
9	心配ばかりしている		
10	ひとつのことに気持ちをむけていることができない		
11	人と話をするのがいやになる		
12	見知らぬ人が近くにいると気になる		
13	何となく全身がだるい		
14	寝つきが悪い		
15	しあわせを感じている		
16	やってみたいと思う具体的な目標をもっている		
17	物事にこだわっている		
18	がんばりがきかない		
19	人と会うのがおっくうである		
20	周囲のことが気になる		
21	なかなか疲れがとれない		
22	眠りが浅く熟睡していない		
23	自分の生活に満足している		
24	将来に対して夢を抱いている		

Copyright © 2012 IPA, All Rights Reserved