る問題には、あなたが適切な救済をしていく必要があります。

■ 「アジャイル開発をやりたい」と提案してきた担当者を活かしましょう。

あなたの会社の若手の担当者がアジャイル開発を実施したいと提案してきたとき、あなたはどうすべきでしょうか？ 若手で情報処理スキルの高い人材にとって、アジャイル開発は普通のソフトウェア開発になっています。 大学時代の同期は、他社でアジャイル開発を推進して成果をあげているかもしれません。 あなたの会社がまだアジャイル開発を主流にできておらず、先に述べたあなたの意識と価値観があるのであれば、そのチャンスを活かして、人材を伸ばし、組織を変革していきましょう。 しかし、あなたがアジャイル開発に魅力を感じないのであれば、その人材があなたの会社の将来性を見限ったとしても、その判断は受け入れてあげましょう。 どちらがガラパゴスだったのかは時代が決めてくれるはずです。

■ 「アジャイル開発をやりたくない」と相談してくる PO や担当者もいます。

アジャイル開発は、ウォーターフォール開発に比べて発注側の組織と PO に大きな負担を求め、責任を負わせます。 新しい事もたくさん学ばないといけません。 自ら、全体の状況を俯え、都度、最善を考え、責任をもって優先度を判断し続けなければいけません。 開発者とはコミュニケーションを丁寧に行って高いモチベーションを維持して開発を続けてもらわなければいけません。 発注側の担当者も発注先の開発担当者と対等に議論し開発を進めていかなければいけません。 さらに、…

これらが耐えられないや非常する PO や担当者が出たらどうすれば良いでしょう

か？ ウォーターフォール開発スタイルで実現できた発注側が楽できるスタイル、決まった作法を繰り返すだけでそれっぽく進められた経験から抜け出すことができない担当者もいるかもしれません。 ウォーターフォール開発は全て消える訳ではないでしょう。 適材適所と育成のバランスを考えなければいけません。

2.4 アジャイル開発の不都合を理解する

■ アジャイル開発では動作するソフトウェアが早い段階で見えてきます。 しかし、それが商用リリースできるまでには、もっともっと時間がかかります。

ウォーターフォール開発で納品されたソフトウェアは一定の品質基準を満足しているとでしょう。 しかし、アジャイル開発では、品質確保より前に PoC (Proof of Concept,概念実証版)としてリリースされ、使い勝手などの検証や試用を実施することになるでしょう。 それらのフィードバックを受けながら、その後に品質を高める作業を実施していくことになります。 よって、「動いた」と、「商用運用可能」の間には時期的な大きな時間差があります。 動いたのを見て、「すぐサービスを開始しろ！」と、間違った指示をしてはいけません。 誤解しないように注意しましょう。

■ ウォーターフォール開発の方が適したケースもあります。

全ての場合にアジャイル開発が向いているとは限りません。 案件のタイプがアジャイル開発に合っているか？ は考える必要があります。

- **アジャイル開発価値より他の価値が重要な場合：**
  - 経営幹部としてアジャイル開発に魅力を感じない場合。
  - 他社と競争する必要が無い場合。
  - 要求仕様が固まっていて、誰でもいいから安く作ってほしい場合。
  - 人命や重要社会基盤であって、早いサービス開始よりも高品質の

保障が求められる場合。

- **他社に開発保障を求めたい場合（ただし受注会社が存在する前提）：**
  - とりあえず予算内・期間内で完成することが重要である場合。
  - 自らの完成責任を回避したい場合。
  - 全てリスクを回避したい場合。
- **時間不足：**
  - 自社の価値観をアジャイル開発適合に変えることができない場合。
  - 開発責任を委ねられる PO 人材がいない、育成もできない場合。
- **コミュニケーションが上手く行かない場合（コミュニケーション・コストが大きく、ドキュメントベースが効率的な場合）：**
  - ソフトウェア開発規模が大きくチームが 10 人を超える場合。(適切なチーム分割で回避するケースあり)
  - オフショア開発や自社への持ち帰り開発など、別ロケで開発しなければならない場合。(適切なツール使用で回避するケースあり)

■ そもそも何を作るかが決まっていない場合には、どんな開発スタイルを採用しても開発は成功しません。

最終的に、作りたいソフトウェア（ゴールイメージ）が決まっていなければ、開発は失敗します。 製造請負契約であれば、たとえ見積り仕様が1ページ1枚であってもその失敗責任は(その仕様で見積りを出した) 受注者が負いますが、アジャイル開発の場合には自社で負うことになります。 失敗に対して、短絡的に「アジャイル開発は駄目

だ」、「ウォーターフォール開発に回帰せよ」と指示する前に、失敗の原因をきちんと分析してください。 情報からは PO から も集めますが、全ての情報が得られない場合もあります。 あなたの情報収集能力と反省も問われるかもしれません。 まずは、あなたのゴールイメージの具体化に対して、あなた自身が PO にどう協力できていたのかを考えるの も一つの原因追求アプローチです。 きれいな PowerPoint 資料だけで、上手く動作するソフトウェアは作れません。

3. 発注側 PO の心得

3.1 プロダクトオーナ(PO)の責任と権限を理解し納得する

■ ウォーターフォール開発は、発注者が「受身」でも一応完成します。

従来のウォーターフォール開発のほとんどは製造請負契約とセットです。 あなたが発注仕様書を漏れなく書くことができて、かつ、予算内・期限内で受注してくれる会社が見つかり製造請負開発契約を結べれば、あなたの勝ちです。 後はあなたが不在であっても、納期には仕様を(最低限度に)満たす納品物が得られます。 仕様が十分に書かれていれば、満足できるソフトウェアの可能性だってあります。 仕様解釈のズレの発生を避けるために、丁寧なレビューを実施することはあるとしても、あなたは**常に受身**で痛いませんでした。 開発会社から提出された進捗報告を眺め、予定と実績の乖離確認を行い、もし遅延していれば遅延回復策を提案してもらって吟味することだけだったかもしれません。

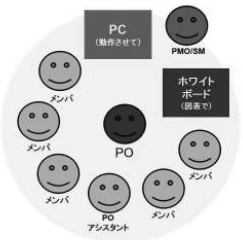

図 3-1 PO はアジャイル開発の中心

■ アジャイル開発は PO が「主体的」でなければ完成しません。

一方、アジャイル開発では、図-1 のように PO が中心に位置します。 あなたが仕様も実装も**積極的に指揮**していくことになります。 それを可能にするために製造請負契約ではなく準委任契約を締結しているのです。 あなたの期待に近いものが作れるように指揮できる代償として、発注側のあなたがソフトウェアを完成させる責任を負うのです。 あなたの判断が失敗すれば、ソフトウェアは完成しないかもしれません。

■ PO がソフトウェア完成の責任を負うことを納得しなければいけません。

ソフトウェアを完成させる責任は、明確なゴールイメージがあり、話のわかるステークホルダが相手で、質の高い開発メンバを集め、良いチームを作り、適切な期間をかければ、それほど困難ではないかもしれません。 しかし、ゴールがはっきりせず、ステークホルダが信頼できず、頻繁だけのメンバを集めて、短い納期に間に合わせる開発だったら、誰がやっても失敗するでしょう。 PO になる前に、成功のための条件をどの程度満たしているか、現在満たされていない条件をどうやって満たしていくか、を責任者の視点で考えるべきでしょう。 そして経営幹部に不足する条件を満たすための支援を要求しなければいけません。 もし、あなたの価値観が引き上げと責任を回避することを優先するなら、PO を受諾しない、アジャイル開発を採用しないのが懸命だと思われます。 また、以下の逆いが先行して主体的に行動できない場合にも辞退すべきかもしれません。

- タイムリーにステークホルダと調整して優先順位を決めることをは面倒なので避けたい。
- 現場の悩みに付き合って、都度、方針や優先順位について判断していく、という手間を避けたい。
- たくさんのレビュー依頼を負なだ、知力・体力に限界を感じる。
- あなた自身が開発するもののゴールに納得できない。
- 開発するものが全くイメージできない。 何を開発すればいいのかわからない。

■ 責任の大きさに伸り権限が与えられていない場合、PO になるのを止めておきましょう。

ステークホルダとの調整の実効的な権限や、納品物を検収する実効的な権限などを、自ら持っていることが重要です。 権限自体を持っていなくても、権限を持った上司との連携作業で進められるなら大丈夫かもしれません。 しかし、たとえば全ステークホルダと調整する権限が無い担当者がPOになる場合には、失敗するリスクは非常に高くなるでしょう。 検収権限がない場合には、自らの判断の前に都度、その検収権限者と調整しなければ、責任ある判断ができないでしょう。 PO になる前に経営幹部に必要な権限は要求するべきです。

■ PO はアジャイル開発の SPoF です。 自らの健康を維持しましょう。

PO はアジャイル開発の中心です。 PO が不在になればアジャイル開発が停止するリスクがある SPoF (Single point of Failure、単一故障点)です。 健康維持には十分注意しなければいけません。

■ 複数名が PO チームを構成することも可能です。ただし、簡単ではありません。

しかし、1 名だけで実施する体制にはプロジェクトの可用性(継続性)として問題があります。 そこで2～3 名で PO がチームを組むことが考えられます。 しかし、容易ではありません。

このとき注意すべきは、各 PO 間での情報のズレ、判断のズレを最小にすることです。 複数の範囲が発生すると逆にアジャイル開発は混乱してしまいます。 開発メンバにも主体性が生まれなくなります。 PO チームとして開発メンバから信頼されるようにならなければいけません。

3.2 発注側幹部がアジャイル開発の価値観に納得していない時

1 章で述べたように、アジャイル開発は発注側幹部の意思から始まるのが正しいスタイルです。 しかし、不幸にも逆順となることが良くあるでしょう。 なぜ、逆順になるのでしょうか。

■ PO 自身がアジャイル開発を実施したいと望むのは、開発のスピードアップやコスト低減を求めたときです。

PO 自身がアジャイル開発を望むときとは、どんな時でしょうか？ 一つは、情報処理スキルが高い人材がアジャイル開発を体験し、そのスピード感が忘れられないときではないでしょうか。 もう一つは、ウォーターフォール開発の変更対応の悪さやスピ

ード感の無さ、納品物の費用対効果の悪さ等に失望したときかもしれません。 ウォーターフォール開発では受注側が全てのリスクを負うため、見積額はリスク料金を含んだ高いもの、開発期間もリスクを見た長い期間になります。 また、当初の仕様で十分詰りきれなかった点があると、納品されたソフトウェアと期待とのズレが非常に大きくなるケースもあるのです。 自分でチームを作って自ら指揮して作った方が、安く良いものが作れると思うことがアジャイル開発を実施したいと思う強い動機となるでしょう。 そのほか、競合社の進め方を知った時ということもあるかもしれません。

■ 社外と競争する必要がない組織であれば、組織の幹部がアジャイル開発のメリットを求めません。

アジャイル開発の変化への対応のメリットは、インターネット上のサービスで競争するような企業が最大限に享受します。 しかし、世の中には異なる価値観の組織も多くあります。 年度内に与えられた予算の中で、確実に買う、ことが重要な場合だってあるはずです。 そんな組織にアジャイル開発はそもそも適合しません。

■ 社外と競争し情報システムが競争力の源泉となる企業なら、幹部にアジャイル開発という技の知恵を授けましょう。

システム開発の競争力が上がること、それは幹部にとっても企業の競争力を左右する重要な情報のはずです。 幹部がもしアジャイル開発を知らないのであればその手法の情報を提供すべきです。 さらに、深く理解してもらい、制度の整備等も合わせて進める権限も含めて理解を得ていきましょう。

■ 企業の幹部が価値を理解できなかったら、競合他社の成功を示して、気づきを待ちましょう。

競争する会社の幹部に、競争力を向上させる手法が1ミリ魅力的に感じないのであれば、その会社は真の競争をしていないのかもしれません。 もしくは幹部にはもっと優先する他の価値観があるのかもしれません。 そんなやつ、あなただけが完成責任というリスクを負うか否かは、あなたの 自身の判断です。 少なくとも、最初から大きなアジャイル開発を実施することは避け、小さな成功を積み重ねて実績を積んでいきましょう。 また、契約としては派遣契約を使って技術者を集め、内製するのがわかりやすいでしょう。 結果として、類似の他のウォーターフォール開発と比べると良いものが作れるでしょう、その姿をアピールしていくことも良いでしょう。 また、競合企業等がアジャイル開発で成功している事例については情報として伝えて行きましょう。 あとは、使命感を持って企業を変える努力をするもよし、もっと力を発揮できる環境を探すもよし。 幸運を祈るだけです。

3.3 ゴールに対して、価値のある部分をまず動かして全体の目処をつける

■ ゴールを決めるのは PO の責任です。

アジャイル開発では、開発メンバにゴールを理解してもらって協力してもらわなければいけません。 万が一、ゴールを PO が正しく理解できていないのであれば、安易に開発をはじめてはいけません。 ゴールを明確にするためには、プログラム開発以外にもっと重要なことがあるかもしれません。 一旦、開発が動き始めてしまうと、それだけで周囲を安心させてしまうかもしれません。 手段を目的化しては駄目です。 何かやっている感じの演出も駄目です。 開発の前にゴールを明確化する調査などの営みをしっかり実施しましょう。

■ ゴールまでの計画を見積る責任も、発注側の PO です。

アジャイル開発で、ゴールを決め、ゴールに向けての全体スケジュールを計画し、それに必要な体制規模と期間を決めるのも PO です。 当然、ウォーターフォールと同様に精度高く短時間で見積もる方法は存在しません。 アジャイル開発において、期間も体制も見積もれないのに完成責任を果たす最大の手法は、**重要な部分の解決を小さく実施して目処をつけることを最優先する**、ということです。 解決すべき重要な部分とは例えば以下のようなものです。

- プログラムの主要機能の実現方法の目処がついていない。
- 期待される性能を満足できるか見通しが全く無い。

- プログラムの核として利用するパッケージ・ソフトウェア／クラウド機能／OSS (Open Source Software) の理解が十分ではない。
- 他システムとの連携方法の理解が十分ではない。
- 期待するユーザインタフェースの実現方法の目処が立っていない。
- プログラムの中心となるアルゴリズムの目処が立っていない。
- プログラムのアーキテクチャが決まっていない。

解決すべき重要な部分の見通しが不明な時点で、それを解決せずに見積もっても度胸(KKD 法の D)で見積もったしかし言いようがありません。 最初にその部分の解決に目処をつけましょう。 そして、その結果を受けて見積りの精度が向上するか、チームの皆と議論してみましょう。 次に重要な部分があって、そこを解決しなければやはり全体の達成が不透明かもしれません。 そのときは、そこを解決していきましょう。 重要な部分が明確でない場合には、中核となる機能を短期間で簡易に試作してみましょう。 それによって、どこに難しさがあるのかが見えてくるはずです。 いくつかの重要な部分について解決の目処が付くと、後はそれらの経験とこれまでのチームメンバの経験を合わせて見積りができるタイミングがいつか訪れるでしょう。 そこに、残るリスクに見合った期間を足せば、やっと皆が納得できる見積り計画の完成となります。 見積もった計画を実行していく中で新たな課題が出てきたら、計画は素早く見直しましょう。

■ 基本設計を慎重するための初期プログラムコードは、無理に残さないことを原則としましょう。

基本設計を明確にするために、一旦、動作するものを組み上げてみることをすることは、ドキュメントを眺め続けて設計の精度を上げるよりも効率的なものになります。

しかし、注意点があります。 基本設計を決める際に作ったプログラムコードは再利用可能でしょうか？ 使える場合もありますが、無理に使わず捨てるのだと思いましょう。 もったいない病、ライン単金病、手段目的化病等に負けてはいけません。 現代のシステム開発では、プログラムコードよりも良い設計の方が圧倒的に価値が高いのです。 優先度判断ができない姿勢、二兎を追う姿勢は避けましょう。 同様に、基本設計を銘打諸撮するプログラムコードに UT を実施する等のバランスの悪さも避けなければいけません。

■ 未確定部分があれば、課題を分離して、仮置きして全体を進めることが重要です。

アルゴリズムや詳細方式が決まっていない部分があれば、その部分に対して簡素で交換可能なインタフェースを用意して、一旦、最も単純・原始的なアルゴリズム・詳細方式を使うことにして、全体を進めましょう。 全体を通すことで、さらに重要な問題が無いかを探っていきましょう。