■ **新メンバの参加障壁を低減するための準備を、タイミングは遅ぴっつ行いましょう。**

アジャイル開発でメンバが追加・交代する際に必要なドキュメントの整備を計画しましょう。 詳細設計のドキュメントはソースコードで十分です。 システムの概要、アーキテクチャ図、外部 API 仕様などが求められますが、あまりにも初期の時期にそれらを整備するのは難しいかもしれません。 システムの基本設計が安定したはじめたら、それらのドキュメントは速やかに整備するのが一つの考え方です。

5.2 自らがゴールを示し、皆に理解してもらってチームを作る

■ **もし PO 自身のゴールイメージが不明確なら、まだ開発に入ることは止めましょう。**

アジャイル開発では PO 自身の判断が成功の鍵を握ります。 車で言えば運転手です。 最終的なソフトウェアのゴールイメージを持たなければ目的には到達できませんし、開発プロジェクトは迷走するでしょう。 PO 自身が開発したいもののゴールイメージを全く持てない場合には、開発メンバにゴールを伝えることもできません。 開発に入るタイミングはもう少し後、ゴールイメージを具体化した後にすべきです。

■ **良いソフトウェアを作るために必要なものは、開発者の高いモチベーションです。**

■ **ゴールを共感してもらって、モチベーションを高く維持することに注力しましょう。**

ゴールに共感してもらうことが非常に重要です。 皆が一つの目標を共有できれば良いチームになれますし、小さい成功を繰り返せばよりチームのモチベーションが高まっていくでしょう。 ゴールが理解されなければ、高いモチベーションを維持して欲しいと思っても、それは難しいでしょう。

■ **ゴールを正しく理解してもらうために、何度も説明しましょう。**

ゴールは、まだこの世界に存在しないもののことも多いでしょう。 そのゴールイメージを正しく伝え理解してもらうことは大変難しい作業です。 一度で伝わることはないと覚悟し、機会を見て何度も説明しましょう。 プロトタイプ・システムが動き出すまでは、皆のイメージはなかなか合わないものだと思いましょう。

■ **ゴールが変化した際には、丁寧にその理由を伝え理解してもらう努力をしましょう。**

ゴールイメージが不安定なときには、途中状態も含めて皆に共有していきましょう。 共有によって、将来の変更への備えが得られる場合があります。 たとえば、この辺りはまだ議論が再燃しそうだから、後で見直す想定で開発を進めていこう、などの判断ができると効率的になります。 ただし、変更の可能性が多すぎると前進できなくなります。 汎用化しすぎることなく作りなおす覚悟を持って仮に決める姿勢が大切です。

また、競合サービスの影響など外的要因によって突発的に、ゴールが変化することもあります。 その変化に至った理由をできるだけ皆に理解してもらい、モチベーションの低下に繋がらないように配慮していきましょう。

5.3 一人ひとりを知って適材適所に配置する

アジャイル開発では優秀な技術者を揃えたいところですが、全員が同じスキル・特性とは限りません。 チームとしての力を発揮するために、一人ひとりのスキルや興味をできるだけ生かして適材適所の配置に努めましょう。 4 章でも紹介したとおり、コミュニケーションの質を上げるには、メンバの一人ひとりを良く知ることが重要でした。 そこで得た一人ひとりの人物モデルを活用して適材適所を考えることが大切です。 ただし、目の前だけを見てはいけません。 チームカの可用性（継続性）や育成（スキル向上）を考えた配置とのバランスも大切です。

5.4 状況・問題点・今後の方針を積極的に聞き、現場を正しく理解する

■ **良い優先度判断をするためには、正しい情報を集めるように努力しましょう。**

良い判断をするための大前提は、実際の状態を正しく認識していることです。 常に、現場の実態を見て、現場の実態も勘案した優先順位を考え続けていかないと、良いソフトウェアが作れないだけでなく、失敗するリスクが高くなります。

■ **朝スのヒアリングは、ポイントを絞って短時間で終えましょう。**

朝のスタンディング・ミーティング（朝会）では、昨日の実績、今日の予定、現在作業進捗が悪いならその課題が何か、今後の見通し・完了予定時期、を簡潔に確認し、短時間で終了しましょう。 朝会で課題を報告した開発メンバに対しては、必要なメンバを別途招集し、詳細に相談をする場を設定します。

■ **昨日の星に発生した大きな問題が翌日の朝会で最初に報告された場合、なぜ遅報が遅いのかを確認し、改善していきましょう。**

朝は緊急情報を報告する場ではありません。 悪い情報は速報を入れるという習慣がまだ浸透していないと判断した場合には、次回から改善を促しましょう。

■ **開発の作業現場に足を運んで、現在の状況を確認し、困っていることが無いかを確認しましょう。**

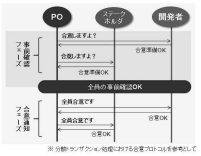
PO は、ステークホルダとのシステムのゴールの調整やプロジェクトの予算措置、各メンバからの相談に対する判断など、きわめて忙しい存在です。 しかし、時々、開発の現場にも足を運んで、現在の開発メンバの状況を自らの目で確認し、積極的に課題を検出していきましょう。 モチベーション高く皆が開発を行っているでしょうか？ 作業環境や生活面で困っていることは無いでしょうか？ 生活利便性の解決はモチベーションの向上に効果があります。

5.5 ステークホルダとの調整を実施する

システム開発は、ステークホルダとの調整と共に進みます。 ステークホルダとは、開発資金を提供する人、システムを構築・運用する人、サービスを売る人、最初に開発システム／サービスを利用する予定のユーザ等、多岐に渡るでしょう。 それらのステークホルダと開発システムの仕様については調整をしなければいけません。

■ **ステークホルダとの合意は、チームの代表として現場も見て判断しましょう。**

ステークホルダと開発チームとの間に立って調整をするのは PO の役割です。 しかし、PO はステークホルダと開発チームとの相互の主張を確認した後に判断することが基本であることを忘れないでください。 重要な合意の際には、2 フェーズ・コミット（図 5-2）による合意確認が大切です。 PO は最終決定の責任者ですが、現場が反対する決定をした後に開発メンバをどう説得するか、納得させるかについては事前に考えなければいけません。 開発現場・チームの実態を正しく知って、代表として判断をしなければ、チームのリーダとして認められることはありません。



■ 図 5-2 ビジネス上の2フェーズ・コミット

■ **アジャイル開発ではリリース・タイミングの自由度が高いことを活用して、開発チームとステークホルダの調整を実施しましょう。**

ウォーターフォール開発・製造請負契約では、リリース回数は通常 1 回です。 しかし、アジャイル開発・準委任契約では PO の判断で何度でも可能です。 今開発に入れるか否かという 0/1 の判断ではなく、常に優先度の高いものからリリースしていくことで、開発チームとステークホルダの双方の都合を調整できることがあります。 品質レベルを含めて双方に信頼される調整をしなければいけません。

5.6 丁寧な対話を続け、バランスの良い優先度判断を続ける

4 章で示したように、コミュニケーションは継続して対話相手の人物モデルを改善することによって質が改善していきます。 それがモチベーションの向上や、チーム力の向上に繋がり、良いソフトウェア開発に繋がっていきます。 しかし、ちょっとした一言によって、モチベーションを大きく損なう可能性も持っています。 十分な関係が出来るまではより丁寧なコミュニケーションを継続するようにしましょう。

■ **アジャイル開発では変化は必然です。 期限があるときに、追加分があれば、削る（先送りする）分も同時に作りましょう。**

開発の途中での追加要求による追加変更作業量が A なら、従来予定の作業から A 以上の作業量を削減しなければ、開発終了時期が遅延します。 アジャイル開発では削った分（＝優先度を下げた分）をバックログとして、先送りします。 その決断を PO はしなければいけません。 必要ならステークホルダと急いで調整をしましょう。

5.7 人格者のリーダとして信頼される

チームのメンバから信頼されるリーダになることを目指しましょう。 まずは、仲間になることが第一歩です。 同じ目線を共有できるようになりましょう。

小さい成功の積み上げや、開発チームの代表としてのステークホルダとの調整などの実績を通して、リーダとして認めてもらえるように努力を続けましょう。

開発上の課題や商用サービスでのトラブル対応などについても、騒ぎず、冷静にかつ先頭に立って、常にバランスが良い判断を続け、開発メンバにとって風通しの良い組織が構築できれば、きっと信頼され人格者として認められることでしょう。 そこへの到達は容易ではなく、時間がかかるかもしれません。 諦めることなく続けていきましょう。

5.8 仲間とともに Win-Win と成長を目指す

PO はチームメンバとの Win-Win の関係の姿勢を出し、実践することが重要です。

■ **開発メンバの自己研鑽を推奨しましょう。**

アジャイル開発を支持するソフトウェア技術者は、新しい技術を常に習得するモチベーションを持っているケースが多々あります。 そういった自己研鑽意欲を活かしてあげなければいけません。 いつかプロジェクトにも良い効果をもたらしてくれるでしょう。 プロジェクトの成功と技術者としての成長の両方を目指して行きましょう。

■ **開発メンバの超勤削減と健康管理も PO の考えるべき範囲です。**

発注者先の勤務管理責任は発注先の管理者にあります。 しかし、超勤を発注者側だけに任せておこことは危険です。 超勤は、一定レベルであれば生産性を高めるかもしれません。 しかし、アジャイル開発では原則として避けるべきです。

- ❖ 準委任契約の場合には、超勤した分が支払額の増加につながります。 休日出勤の対応をお願いしたらいくらかかるか、は知っておかなければいけません。
- ❖ 超勤を前提としない進め方をベースに計画をしましょう。 超勤は、いざというときに遅延を回復するバッファとしましょう。 そのバッファを最初から使い尽くしてはいけません。
- ❖ 超勤が長期間続くと風邪を引きやすくなるなど、健康面でのリスクが高

まるでしょう。 さらに、超勤が続くと、燃え尽きることがあります。 モチベーションが無くなる状態は避けなければいけません。

- ❖ アジャイル開発は、幅広い情報処理の知識を持った高スキルの技術者を期待します。 技術勉強会等に参加するなどして開発物以外のことを自己学習する時間的な余裕を与えましょう。

PO はゴールイメージをもとに、時間・コスト・成果物（機能・品質）のバランスを自分で判断していく必要があります。 ソフトウェア開発はバランスの産物です。 バランスを間違えると良くないソフトウェアになってしまいます。 バランスが悪いもの、それは乗用車にロケットエンジンを積み込んでしまうようなものです。

■ **相互に休みやすくするコツを実施しましょう。**

休暇の取得は、全員が持つ権利です。 相互に助け合う体制になると、気持ちに余裕を持って休暇を取得できるようになるでしょう。

- ❖ 皆が休暇を取得しやすくするために、日々の自分の最終状態を常に他のメンバが共有できる場所に置きましょう。 チケット管理システムなど利用し、帰宅前に記載を意識すべき作業として位置づけるべきです。
- ❖ 急に休暇を取得する場合には、作業に関する引継ぎ事項の伝達と、休暇からの復帰する時期（明日出社可能かいの見通しも含めて連絡する習慣にしましょう。

6. 品質について考え直す

本章について、最初に、お詫びがあります。 アジャイル開発における品質確保の手法について、常識らいものがあるのかについて、著者には全く知識がありません。 本書全体が著者個人の意見を中心に記載していますが、特に本章の内容は将来見直される可能性があります。

6.1 ウォーターフォール開発では形式的な合意形成の事例がある

【現実 6.1】 品質に 100%は存在しません。

ソフトウェアにバグが無いことは証明も確認もできません。 テストはいくらでも継続することも可能です。 さらに、そもそも要求仕様が真の要求を 100%満足するものになっていることさえも保証できていないかもしれません。

【現実 6.2】 品質とは、費用対効果のバランスを考えて割り切るものです。【品質トレードオフ原則】

品質 100%の世界が無いとすると、どんな試験をどこまで実施し、どこで止めるか、

が重要なポイントになります。 **QCD(品質・コスト・納期)のバランスをとった判断が必要で大切**になります。

■ **ウォーターフォール開発の良さは、試験の進め方および試験終了に関する合意形成の形式的な見本が多数あることです。**

ウォーターフォール開発では、たとえば試験の進め方は以下のようになります。

- ❖ 各工程のドキュメント作成とレビューによる合意を行い品質の高いドキュメントを積み上げる。 必要ならレビュー時間・コメント密度などの指標も活用しドキュメント品質を上げていく。
- ❖ 試験工程は「V 字モデル」で示されるとおり、各工程で生産したドキュメントに対応する試験観点・試験項目を用意する。
- ❖ ソースコードはクロスレビューをする。
- ❖ 単体試験は C0(命令網羅率), C1(分岐網羅率), C2(条件網羅率)などの網羅率の目標値を決める。
- ❖ 試験観点については網羅性をレビュー合意する。
- ❖ 目標の試験密度を決め、試験項目で正常系試験数と準正常・異常系試験数の比率を決め、試験観点から試験項目を展開する。
- ❖ 長安（長期安定化試験）、過負荷長安について試験時間の目標を決める。

そして、試験を実施した結果について、たとえば以下のような分析と対策を実施して、試験終了判定を実施します。

- ❖ 各目標値を満足しているか、基準範囲内に含まれているか、乖離があれば乖離理由は何かを分析します。
- ❖ バグ検出数・傾向の分析をします。 高密度なら品質低ではないか、低密度なら試験不足ではないか、バグ出現は特定の機能や観点部分に偏

りないか、摘出した工程は適切だったか、などの分析を行います。
- ❖ 長安実施中のリソース傾向、過負荷時の状態、過負荷終了後に定常状態に戻った後のサービス回復／リソース回復状況などを分析します。
- ❖ バグや異常動作については、原因分析（なぜなぜ分析）を行って弱点やミス発生理由を検出します。 偏り（特定の機能か、特定の使われ方か、特定の作成者かなど）のある弱点がないか分析します。 そのバグが作り込まれた本当の理由はなにか、そのバグはどの工程で検出すべきだったのか、それを防ぐために本来各工程で何をすべきだったのなど、を分析します。
- ❖ 各弱点や原因に応じて本質的な品質向上対策を類似部分についても水平展開を実施します。
- ❖ 品質不足、試験密度不足、観点漏れがあれば、追加で強化試験を実施します。
- ❖ 強化試験結果の再分析とバグ原因への対策完了で試験終了を判定します。
- ❖ バグ収束曲線の傾向分析を行い収束傾向から試験終了を判定します。

ウォーターフォール開発では、試験前に合意した各種目標値を満たすことで試験終了と判断することができます。 その基準を満足することが、決して、サービス開始後に重要な問題を起こさないということを保証するものではないことも多くの人は理解しています。 実際、ウォーターフォール開発で実施したシステムでも、不具合・故障は発生しています。 それでも、ほぼ形式的に試験終了判断を可能にする合意形成メカニズムは有効です。 他者と同等レベルの形式に従って試験を実施したので、問題が発生しても、それは仕方なかったという横並び論理が使える枠組みとして割り切るのでしょう。（実際に、商用サービスで大きな問題が発生してしまうと、そんな理由は何の気休めにもならないのが実態かと思いますが・・・。）

■ **ウォーターフォール開発では、受注側に完成責任（瑕疵担保責任）があるので発注側に安心感はありますが、実は期待するほど強力な仕組みではありません。**

ウォーターフォール開発では、受注する側に欠陥の無いものを納品する義務が発生しています。 その枠組みがあるから、困ったら無償で直してもらえばよいと考えれば良いのですしょう。 ある意味では正しい考え方ですが、一方で注意しなければいけない点はあります。

- ❖ 受注側の責任による欠陥であることを客観的に示せるでしょうか。 仕様として誤解など明記されているでしょうか。 伝え方が悪かったなどグレーゾーンであることがあります。 伝えた記録はきちんと残っているでしょうか。
- ❖ 要件 FIX、仕様 FIX 後に仕様変更を押し込むなどの強引な行為は無かったでしょうか。 途中から言い方を変えて受注者を混乱させていなかったでしょうか。 変更に伴って期間・費用を適切に見直した契約変更を実施しているでしょうか。
- ❖ 経験豊富な開発会社であれば、都度、発注者にレビューを依頼し確認を積み上げ続きます。 発注者側でレビューは完全にできているでしょうか。 最終品質も、「ここまでこういう試験をしている」と発注者に報告した記録と、発注者側の受入試験に合格した証跡があるはずです。 受入確認は十分だったのでしょうか。

これらに発注者側が完璧に対応していれば、瑕疵担保責任によって無償改修してもらえるでしょう。 しかし、発注者側にも仕様記載漏れ、レビュー確認漏れ等が発生すると、訴訟に発展していきます。 痛み分けとなった場合には、双方に弁護士費用と不毛な時間が発生します。 「伝家の宝刀」だと思って抜いたら自らも大きく傷つ