# プロジェクト
# レスキューマニュアル

**できれば炎上するまえに読んでください！**

PMファクトリー　2016年

目次

## はじめに

重大な問題が分かっていたとしても開発工程の前段階では危機的な問題として取り上げられることは本当のところめったにないのだ。 だれでも「まだ時間があるからどうにかなるだろう」なんて思っているうちは誰も危機的な問題として取り上げようとはしない。
危機的な状況を招くほとんどの原因は開発の上流工程にあるのだ。しかし本書は、これらの原因の追求については後半の第5章以降で触れたいと思う。 なぜなら本書の重要な目的は、今現在、大混乱に陥っているプロジェクトをどうにかしようということなのだ。 つまりお客様および開発がこうむる被害を最小限にくい止めるには今からでもどんなことができるのかという話を最初にしたいのだ。

普通、プロジェクトマネジメント本やガイドブックは開発の前工程から順次後工程へと5W4Hで「・・・しなければならない」との記述の連続になっている。 だけど結局そんなことができなかったから、またはやらなかったから今現在プロジェクトはボウボウと炎上しているワケでこんな箇条書きの対策をいまさらとれるわけもない。
そういうわけで本書は、最も被害状況がひどい開発の最終工程から逆に前工程へと遡って話を進めたい。

開発の最終工程から逆に遡ってみた時の各工程での主だった危機的な状況はつぎの通りだ。

### （1） 炎上の市場稼動工程

最終工程の「市場稼動工程」で問題が爆発している状況は悲惨だ。 お客様のところでは取引ミスが発生したり、誤ったデータが膨大な範囲で拡散したり、その果てにはシステム全体が動作不能になったりして膨大な金銭的、時間的ロスを出したうえに一番大事な信頼まで失っているのだ。 客先および開発組織はパニック状態に陥っており、客先からは損害賠償の懸念が表明されている。 毎日損害発生が続いているけどキミならこのピンチどう切り抜ける？

### （2） 爆発の評価工程

市場稼動工程におけるパニック状態よりはまだましだけど、不具合が続出してテストがあちこちで止まっている。 とうとう納期までに総合テストが完了できず上司はもちろんのこと客先もカンカンに怒っており契約解除もにおわせている。 もう爆発してしまっているけどキミならどうする？

### （3） 混乱の製造工程

なんだか分からないが設計の基本的なところがコロコロ変更され製造チームはなんどもやり直しをさせられた。 製造チームは何が本当の仕様か疑心暗鬼でやる気もうせてしまった。 製造工程は大幅に遅れ、まだ未製造の部分がかなり残っているが、みんな何とかなるだろうと思っている。 このような状況を上司も客先も知らされていない。 キミならこの混乱からどう抜け出す？

### （4） あせりの設計工程

仕様凍結がずるずると遅れており設計が完了できていない。 もうスケジュールの期限になったが未凍結の仕様がどれだけ残っているかもサッパリ分からず未着手の設計も5割以上も残っている。 あせっているキミはどうする？

### （5） 待ちの要件定義工程

客先は要求仕様を提示できない、あるいは要求内容が二転三転してなかなか仕様が固まらない。 客先は仕様を提示してくるのは開発の仕事だと思っている。 客先もSEも開発もだれもが仕様を決めるのは自分の仕事じゃないと思っている。 もう設計開始の時期がきたがまともに仕様らしきものが決まったのはわずかばかりで残りの仕様があとどれくらいあるのかも想像できない。 このような状況にあっても客先も開発組織も何らの危機感も抱いていない。 まだまだ時間はいっぱいあるから、後工程でなんとかなるだろうと思っている。

キミは黙ったままでいるの？

## 第1章　みんなの生の声

プロジェクトの全ての工程でいつも深刻な問題が発生しているが次に開発現場の担当者の生の声に耳を傾けてみたい。 現場の人々には、仕事のやり方・品質・ドキュメント・スキル・マインドの 5 つの項目についてそれぞれ今どんな問題を抱えているか、何故その問題が出てくるのかを聞いてみた。 背筋が寒くなるような話ばかりで、自分の組織はそれほどひどくないと思っているのなら匿名のアンケートをとってみるとよい。これは創作ではなく事実なのだ。

### 1.　プロセス・やり方の問題

**◎どんな問題点があるのか**

- 当たり前の事を当たり前に出来ない。
- 時間が思うようにとれない。
- 開発着手したが遅れが生じ間に合わないという時や、抱えすぎて廻らない時にテスト作業、ドキュメントを省略している。
- 開発各プロセスが省略されているために品質が低下している。

- マネージャを含めて、仕事の優先順位の管理ができていない。
- 情報の整理及び管理能力が低い。
- 個人依存の開発スタイルから抜け出せない。
- システム検討をしていても積極的にホワイトボードに書く人が少ない。

- レビューの内容が不十分。
- レビュー自体が適正に行えていないし、フォローもできていない。
- 記録シートの作成などの形式だけのレビュー。

- 形式だけの影響度表作成しか行われていない。
- 影響度表を考慮した設計・製造・評価が行われていない。
- ソフトウェアの品質を評価チームによる人手だけのテストに頼っている。

**◎何故問題が起きるのか**

- いまだ、人的スキルに多くを頼っている。
- 知っている担当者（協力会社）へ丸投げするほうが安く、納期も短縮できた。
- 社内は要件定義とプロセス管理、外注は設計・製造という位置付けが定着した。
- 担当・協力会社変更、ローテーション問題など引継ぎ体制、内容が不十分。
- 発注元からレビューされることが無かった。
- 要件定義ベースの読み合わせを設計レビューとした。

- プロセスのプライオリティ付けの誤認識のためドキュメント作成にあてる時間を減らしている。
- 初期構築時には何らかの開発者向け設計書はあったが更新していない。
- ドキュメントは管理者が私的に保有したか、または管理者が変わり所在不明。

・ 設計内容を知るメンバーが担当変更になり資料格納場所も不明になってしまった。
・ 全容を解説する資料が無くても、バージョンアップ開発が出来た。
・ ドキュメントがなくてもバージョンアップ開発時にはソースベースで調査・改修を実施した。
・ プログラマーにとって、メンテナンス箇所のみ解れば良く、設計書を必要としない。
・ 発注元でドキュメントの管理もせず、外注に要求もしないので設計書作成の工程を省略した。
・ 自社固有の技術として隠蔽する方が次の仕事に繋がる。

・ システム構造・関係・流れを学ぶ資料がない。外注がいれば自分が知る必要はない。
・ 自ら設計する技術が無くなった。
・ 社内製造者が皆無の状態で技術的精査が出来ない状態がつづいた。
・ 技術基盤（設計・製造技術）を培う時期に、管理作業（外注、テスト）のみの仕事状態が続き、設計・製造を経験したことが無く、開発技術に何が必要か解らない。

## 2. 品質・障害対応問題

### ◎どんな問題点があるのか

・ 障害クレームの発生件数が減少できない。
・「何が発生しているのか分からない、何に困っているのか分からない」と言う事が多い。
・ 障害対応ノウハウの整理と管理ができていない為、一部の人間しか調査・回答が行えなかったり、対応時間に多くの時間を費やしてしまったりしている。
・ 障害報告書も技術的な側面からの報告が出来ない。絵あるいは図を書いて説明できない。
・ ログに関して十分な情報がログから判明しない。後ろ向きな作業に時間がかかる。

### ◎何故問題が起きるのか

・ 障害対応がその場限りであり、それを財産として活かしていない。
・ 過去に発生した障害の歯止め対策が以降の開発で実行されていない。報告書に再発防止策を記載する事に留まっている。
・ 不具合対策にて分析まで掘り下げていない。
・ 防止策が何故必要となったかが薄れ、内容が乏しくなっている。
・ 真因特定の曖昧さが歯止めの曖昧さにつながっている。
・ 不具合の分析や真因特定の結果が、人のせいになってしまうことも多々ある。
・ 歯止め策をみると非常に曖昧な内容になっている。「～を徹底する」「レビューする」等の記述が目立つ。また具体策が提言されていたとしても、大げさな内容が多く、「本当にそんなことができるのか」との疑問がある。

・ 横展開してない・ルール化されていない。
・ トラブル時の資料を配布してもほとんど活用されない。
・ 同じ部内なのに、無関心、知らない。

## 3. ドキュメント問題

### ◎どんな問題点があるのか

・ ドキュメントが分かりにくい。
・ 作成者以外が見ても判るレベルではない。
・ 客先要件の実現方法が分かるようになっていない。
・ 仕様書の内容が貧弱である。
・ 設計書の記述が不足している。
・ ドキュメントがしっかり出来ているものはほとんど見られない。
・ インプット、アウトプットの記述を明確にする必要がある。
・ 各グループにおいてドキュメントが全て完璧に揃っているグループは残念ながらない。

・ システムを理解できる資料が乏しい。
・ 端末、サーバの連携した仕様が設計書に記述されていない。
・ どうしてシステムを構築したり変更したりすることになったか等のコンセプトや背景，開発に至った経緯情報がない。
・ 弱いと痛切に感じる点は、複合システムにおける全体図，製品間の連携部分の説明がない。

・ カスタム機能に限定した記載しかなく、全貌が見えない。
・ 機能仕様書は小規模開発であっても作成しているが、変更する目的が不明確となっている。なぜ、標準仕様を変更するか、なぜ変更が必要かの説明が漏れている。
・ 内容が要件や仕様より、ロジックに偏った記述が多く、実運用がイメージできない。
・ 誰のために書かれたものなのか判らないドキュメントが多いと思う（設計書の延長線上のものが多い）。

・ 開発時には設計ドキュメント、運用フロー、システム構成図、テーブル構造はあると思いますが、データフロー、プログラム処理構造図がない場合が見られる。
・ データフロー，構造図等の各種ドキュメントに関しては、内容が不十分、作成できていない部分も多々ある。
・ データフローに関しては、ひどい場合は存在しない。
・ データフロー・構造図などは設計ドキュメントしては皆無に近い。

・ 影響度表に関しては、現在使用しているものの効果があまり出ていない状況だが、改善も行われていない。一部ツールを作成しているが、まだ部内でも展開できていない。
・ 修正影響度表は出荷用資料としては作成しているが、有効活用はできていません。

・ ドキュメントの精査および更新が完全で無い為、対策案の各種チェックを行っても不具合が防げていない。
・ 障害対応後、最新のドキュメントに更新していない。
・ 障害発生時の解析の資料が遺されていないため利用できるものになっていない。
・ 同じような資料が多く存在し、その資料が更新されてない。
・ 流用の際の判断（工数、リスク）の裏付け不明なため利用できるものになっていない。

**◎何故問題が起きるのか**

・ 時間がない（納期に間に合わない）を理由に各工程のドキュメントを省いたり、内容がプアーなものが見受けられる。
・ 見積もり作業やクレーム回答を行うのが精一杯で協力会社への依頼も口頭で行う場合がある。検討、指示、依頼が曖昧で漏れがある。担当者にしかわからない。
・ 資料が最新ではない、忙しい（人がいない，時間がない）等の理由で、必要な資料も十分にメンテされずに開発が終了してしまう（終了できてしまう）。

・ 資料がない。
・ 皆が参照できる様に整理，管理できていない為、＝「資料がない」という状況になっている。
・ 発注元あるいは協力会社の成果物としての認識が不足している。
・ 各種の資料の重要性・必要性を認識していない。
・ ドキュメント管理が個人（担当）レベルになっている。
・ 端末開発は協力会社（もっと言うと人に）に依存しすぎていたため、初期設計書以降の工程に関する資料は皆無です。
・ ドキュメントの書き方が統一されていないため、見ても分からないことがある。どこに保管されているのかもわからない。
・ 資料を探すのが面倒。資料の格納先に関する全社的な指針がないため、その時点の担当者の感覚で資料を格納している。
・ 資料が分かりづらい。思いこみで作成しており他の人から見ると分かりにくい資料になっている。作成した資料の評価を行っていないため、資料の記入内容に最低限の一貫性がない。
・ 必要性が徹底できていない。

・ 本来、開発者がドキュメントを作成する場合、客先（対外部）向けの記載と製造者（対内部）向けの記載を区別して書かなければならない筈だが、開発者が客先の本当の要望（やりたい事）を理解していないために書けない。
・ 過去のドキュメントが有効活用されていない。
・ 管理方法も、過去のドキュメントに蓄積していくのではなく、開発毎に個別に作って最新の状態が一目ではわからない。
・ カスタムする機能に限定した内容しか記載されていないものが多い。標準の仕様書がプアなため、どうしてもカスタムに限定したドキュメントしか作成できない。
・ バージョンアップ時に以前の運用の考慮が漏れる。
・ 全体を理解できる人が少なく全体の記述が無いためで致命的なミスが後工程で発生してしまう場合がある。全体記述が無いので資料の再利用ができない。
・ 機能を実現する為のモジュール、レジストリ、フロー、初期値等いろいろな物が設計書に記述されていない。その為に出荷時のミスを起こしてしまう。

## 4. スキル問題

### ◎どんな問題点があるのか

- 開発者の意識、スキルが足りない（社内、協力会社ともに）。
- 開発リーダーのベースシステム経験が不足している。
- 長い間の悪習で”まず書く”、”絵を書いて考える”ということができない、苦手な人が多い。
- システム的な捉え方ができる人材がいない。システム全体評価の精度が低い。 システム全体をコーディネイトできる、あるいは意識して取組んでくれる人材がおりません。 PMレベルでも、「俺は端末しかわからない、あるいはサーバしかわからないのできません」と開き直っているひとが多いです。
- なぜその仕様が必要かなどプログラムを実装する上で、お客様の「目的」を理解することが必要不可欠と考えます。お客様の目的が理解できれば、必要範囲、逆提案も可能となり、開発効率UP・顧客満足度向上に繋げることもできる。

### ◎何故問題が起きるのか

- スキル不足、モチベーション不足で書けない、。他人の作成した部分だから分からない。
- なぜ、その仕様が必要か理解されていない。画面上、その項目を出力する意味を判っていない（知ろうともしない）。
- 図、マトリックスを書き考える事のスキル、習慣がない人がPM、PLでも多い。
- なぜそのドキュメントが必要か理解されていない。
- 個人スキルの2極化（できる人／できない人）が進み、全体偏差値底上げスピードが、市場要求に対して遅れている（ポテンシャルを引き出せていない）為。できる人への集中化、カスタム開発掛け持ち過多。作業量の平準化ができていない為（全体人数でみれば、まだキャパはあるのに）。

- 協力会社依存度が高く社員が理解できていない。任せきりの発注の為。
- 最大の要因は①特定の協力会社、人に長期に渡り依存した開発による。 ②モディファイ開発主流による新規システム立上経験の不足。 ③外部資料が多く、設計図は製造者任せ。
- 開発要員に新しいメンバーを投入した時の初動教育のチェック不足（協力会社任せ）。
- 自分自身も追われ、なかなか人材育成、特に協力会社に関して時間がとれない。
- マネジャー責任によるものが殆どだと考えます。

## 5. マインド・意識問題

### ◎どんな問題点があるのか

- トラブルの事の重大さに対する認識が薄い。
- 認識の尺度が各人、上位管理層、PM、PL、担当者で異なっている。
- 社内、協力会社ともに開発時の意識、スキルが足りない。
- なぜ再発してしまうのか？自分が作ったものじゃないから分からない、他人任せという風潮があるように感じます。再発防止というと、個々の担当が責められるイメージが強いのかも知れません。発生してしまったものは過去の自分が作ったもの。これから作るものは「今の自分」が作るんだから同じ過ちはしない！という意識改革が必要だと思います。

- ドキュメントの重要性が理解されていない。
- まずもの作りありきで、ドキュメントは二の次と言った考えも根深くあるように思います。見る人に分かり易いようにという意識の欠如。何を書くべきかの勘違い。運用フロー、データフロー、APL構造図は特に必要と考えます。

### ◎何故問題が起きるのか

- 社内開発者の協力会社依存の体質による仕様作成能力の低下、設計能力の低下、プログラミング能力の低下、およびレビュー能力の低下などが根底にあり、自分たちと協力会社（ﾌﾟﾛｸﾞﾗﾑ技術者）の役割、責任を明確に出来ない。これを繰り返しており、運用、仕様への意識・こだわりが無くなって来ている。開発者としての運用、仕様へのこだわり低下（＝無責任な仕事）が品質低下の大きな要因。
- 自身が本当の担当とは考えていない為。自分は追加しただけ、ドキュメントがないのは元を作った人の責任などと思っている。漠然としている部分はあるかも知れないが、ドキュメントの必要性、何を書くべきかは、大多数のメンバーは理解している。特に、各システムの標準にて基幹となるバックサービスの処理は、追加開発／トラブル調査において、何より自身が知りたいはず。全体の意識は向上しているが、人の作成した領域には入りたがらない為。

- 早く製造しないと納期に間に合わない。余計なことを聞く（はっきりさせる）と仕様が増える。仕事が増えるので言われたことだけやれば良い。　といった具合に、貝に閉じこもっていく傾向があり、「はっきりさせる」ことに臆病になっていると思います。　「はっきりさせる」ことで製造範囲を明確にすることが、自分たちを守ってくれることに繋がることを理解させる必要があると考えます
- ドキュメントの必要性を感じていない、理解していない。書こうともしていない。

## 第2章　炎上の市場稼動工程からの脱出法


レスキューを頼まれて行った先のプロジェクトルームはひどい状態だった。 机につっぷして寝てるヤツ、充血した目でパソコンの前に座っているヤツ、会議机の上は散乱した資料や山盛りの吸殻の灰皿だらけだ。 不眠不休でほとんど寝てないせいか全員無言で疲れきった青白い顔ばかり。 あちこちにあるモニターは青白い画面を表示したまま放置されているものが多い。
頼まれたとはいえとんでもない所にきてしまったものだ。

このプロジェクトの状況はこうだ。 このプロジェクトはソフトウェアの新規開発プロジェクトであった。約1年余りの開発期間で、数ヶ月前に総合テストがやっと終了し、製品版のソフトウェアが市場にリリースされたのだが、導入直後から重大障害を多発・連発し、プロジェクトはパニック状態に陥ってしまい組織として機能できなくなっていた。
顧客の店舗では障害が続出してシステムは満足に稼動できず、客先は激怒している。 もう裁判しかないとか損害賠償請求だとかの声がでており万事休すの状態だ。 キミならこのピンチどう切り抜ける？

こんなときはだれでもパニック状態で冷静な仕事はできないものだ。 事実、この時のプロジェクトリーダも開発メンバーも連日の徹夜作業、客先での謝罪説明、上司からの厳しい叱責の雨アラレで完全に思考停止の虚脱状態にあった。
こんな緊急非常事態になったときに真っ先に必要なことは何だろう？ ポイントはやっぱり「冷静になる」ってことだな。 テンパッタ頭でいきなりアッチもコッチも直しはじめるとアッチもコッチももっとひどい状態になるのがオチだ。 実際そんな状態になっているのだ。 不具合対応で、「いきなりソースコードに触ってはいけない」という言い伝え知ってるよね。
冷静になって問題の真因を見極めることができればこれで問題の半分は解決したも同然だ。
これで最小の労力で最大の効果が引き出せる下地ができあがるのだ。
とは言ってもこんな大変な状態のとき当事者は冷静になれないものだ。 こんなにひどくなってしまったら、もう別の冷静なリーダーや開発者たちをどこからか引っ張ってくるしか手はないよね。

無理にでも冷静になったら次にやることは「腹を括る」ってことだな。 実際、もう腹を括るしかないよね。 期限はとうに過ぎてしまっているし、店舗では毎日問題を起こし続けているし、お客も上司もカンカンで、あとは首を洗って待つしかないのかな？ 本当にそれだけか？ 見方を変えれば、こうも考えられないか。
もう期限に追われなくてもいい。 この 1 年間スケジュールはどうなっているか、期限は大丈夫かと毎日毎日せっつかれてばかりいた。 もうその最終期限は過ぎてしまったのだ。 期限はもう自分たちになにも迫ることはできないのだ。
カンカンになっているお客も上司も直接的には何もできない、問題を直接的に解決できるのは自分と自分のチームだけなのだ。 つまりある意味自分たちに主導権があるのだ。 実際、お客はワタシにこう言ったものだ「私たちにできることは何でも言ってください。システムが正常に動くためならチームになんでも協力します」と。 これは本当にあったことだ。
さあ腹を括ろう。 終わったら首にでも何にでもしてチョーダイぐらいの気持ちでこの山を登る覚悟を決めよう。

## 1. 心構え

もう一度いうけど最初にやることは次の二つだ。
**□ 「冷静になる」こと**
**□ 「腹を括る」こと**
この二つを忘れないようにしよう。

さて心構えができたら次は対策実行前の準備にかかろう。

## 2. 対策実行前の下ごしらえ

やるべきことは次の三つだ。

### (1) まずプロジェクトメンバーを休ませよう

とにかくみんな疲れているのだ。何ヶ月も不眠不休の状態が続いておりすでに何人かは倒れてしまったのだ。今が最悪の状態だから、この際今まで頑張ってきたメンバーを4、5日休ませたからってもうこれ以上は状況は悪くなりようがないのだ。休ませよう。

### (2) 仕事のキャッチアップをしよう

二つ目は、新たに開発者およびテストメンバーを投入し、リーダーから担当者レベルまで、それぞれのレベルで現行メンバーとペアを組ませ仕事のキャッチアップをしよう。 キャッチアップが済んだら現行メンバー休ませよう。 大規模なレスキューが必要な場合には新規メンバーの投入は思い切って大規模に投入するのが鉄則だ。 チビチビ投入する「逐次投入型」は効果がないから絶対に避けよう。 古い話だけど太平洋戦争での日本軍の敗戦の理由の一つがこの「逐次投入型」の戦術なのだ。「戦力の集中」が基本だ。 [失敗の本質 p8、p37、p89、p189]

### (3) 基本データを押さえておこう

次に必要なのが、プロジェクトの基本データである品質、人、モノ、カネなどのその時点でのデータを押さえておくことだ。 これは仕事のキャッチアップと同時に行なうと良い。 当然、外注会社を使っている場合はそれも含もう。
これはあとになって誰が何にいくら費用を使ったのか全く分からないようになってしまっては困ることになるからだ。 大規模なプロジェクトではレスキューのドサクサで自分の知らない億単位のカネが突然請求されることもあるから怖いのだ。

この 3 つがレスキューの最初に同時にやらなければならないことだろう。 このアクションはできれば1週間以内で実行したい。 対応速度の速さは余計な問題の発生を防いでくれたり対策の効き目を倍増してくれるのだ。

## 3. 品質安定化対策アクション

続いて直接的な対策を打つ行動にとりかかろう。

### （1） まず市場における品質データの把握だ

数字を知らなければ本当の事実状況は分からないし有効な対策もできないのだ。 現在までに市場および内部テストで発生した不具合の数・種類・原因・対策などについてすべてのデータを集めよう。

### （2） 品質データが集まったら分析をしよう

ポイントは次のようだ。

- □ どの業務の不具合が多いのか。
- □ どのモジュールの不具合が多いのか。
- □ 不具合を多く発生させているチームや担当は誰か。不良なメンバーは外す。
- □ 不具合の原因を作り込んだ工程はどこか。 要件か、設計か、製造か？

ここで大事なことは開発チームの弱点はどこに有ったのかを素早く明確に知ることなのだ。
多分開発メンバーは自分たちの弱点を正確には分かっていないだろう。弱点のある工程にエキスパートをすぐに投入しよう。

### （3） プライオリティ順に不具合を修正していこう

分析が済んだら顧客にとっての重要な機能のプライオリティ順に不具合を修正していこう。
修正にあたって大事なことは現行メンバーと同じ過ちをしないことなのだ。 レスキューメンバーは現行メンバーの弱点を正確に把握した後に、自分たちが持っている優位な技術ノウハウおよびプロセス遂行能力をフルに発揮しよう。

### （4） 十分な時間をかけ効率的にテストをしよう

次に修正したソフトウェアをどうテストするかだ。
修正したソフトウェアは新たに大量投入した評価チームで十分な時間をかけ効率的にテストをしよう。 レスキューの最初の状態では総合テストのやり直しを行うことになる。 必要なテスト時間・期間の目安は、現状の不具合発生率から予測される潜在不具合残存予測を行い、予測未発見数に達するまでどれくらいの評価時間が必要かで割り出そう。 経験からいうと高品質なソフトウェアをリリースするためには、総合テストは予定チェック件数を2サイクル以上回す必要がある。 完全1サイクルも未達成のソフトウェアは絶対に市場で問題を起こすのだ。

### （5） 評価済みの修正ソフトウェアを市場に投入しよう

修正ソフトウェアの投入ごとに市場品質データの変化を毎日収集し、改善状況を確認しよう。
不具合が全て収束するまで（1）から（5）のサイクルをスピード感をもって繰り返すのだ。 対応速度の速さは問題を一挙に粉砕する力をもっているのだ。 不具合が終息したか否かの判断は市場の不具合発生率や不具合内容のレベルで判断できるだろう。

## 4. 緊急対応行動のポイント

以上基本的な緊急時の対応方法について説明してきたが、つぎに対応行動におけるいくつかのポイントを示しておこう。

### （1） プロジェクトメンバーは同じ場所に集結しよう

社内開発チームも外注チームも地理的に同じ場所に集結して作業を行なうのが原則だ。
スピード感のある効果的なレスキューは開発者全員がすぐ近くにいて、毎日直接会話を行なうところからしか実現できないのだ。 何でもITだネットだという迷信を信じてはいけない。
すぐに伝えたいことは相手に直接話しをすることだ。 電子データは機器同士にはリアルに伝送されるが、人にはバッチでしか伝わらないのだ。 リアルに伝送されたデータは明日にしか相手に見てもらえないかもしれないのだ。 またメールは感情や気持ちを伝えることが苦手なのだ。
上記のことを、このレスキューの場面ではなくもっと最初の上流の工程から実行していればこんなレスキューはきっといらなかったに違いない。

### （2） 修正影響度表を作成しよう

不具合の修正にあたっては他のロジックへ悪影響が及ばないように事前に「修正影響度表」を作成しておき評価テストで確認しよう。 簡単なサンプルを下記に示したが各自で工夫して作成してみるといいだろう。

XXサーバ XX様 カスタム対応 修正影響度表

◎：対象(+新規/変更) △：関連

| 大分類 | 中分類 | No. | 小分類 | | システム | | 修正範囲 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 業務項目 | 修正概要 | XX | | GUI | 印字 | サービス | DB構造 | パラメータ | その他 |
| 1.画面系 | 1．メニュー関連 | | | | | | | | | | | |
| | | 1 | 業務選択メニュー | | | | | | | | | |
| | | 2 | H/T運用メニュー | | | | | | | | | |
| | | 3 | POP/プライスカード運用 | | | | | | | | | |
| | | 4 | 精算日報業務 | | | | | | | | | |
| | | 5 | 棚卸業務 | | | | | | | | | |
| | | 6 | マスタ運用 | | | | | | | | | |
| | | | | | | | | | | | | |
| | 2．H/T運用 | | | | | | | | | | | |
| | | 1 | H/Tモニタリスト | 2．1次卸売変対応 | ◎ | | ◎ | ◎ | | ◎ | ◎ | |
| | | 2 | 返品伝票出力 | | | | | | | | | |
| | | 3 | 他店舗発注確認 | | | | | | | | | |
| | | 4 | 卸売変リスト | 2．1次卸売変対応 | ◎ | | ◎ | ◎ | | ◎ | | |
| | | 5 | 卸返品仕入伝票出力 | 2．1次卸売変対応 | ◎ | | ◎ | ◎ | | ◎ | | |
| | | | | | | | | | | | | |

**[図1．修正影響度表]**

### （3） レスキューには精鋭メンバーを投入しよう

プロジェクトマネジャー、プロジェクトリーダー、評価テストリーダーなど要になる人材は精鋭を揃えよう。 余っている人間をとりあえず投入するなどの愚策は間違ってもやってはいけないのだ。 とんでもない悲惨な状況にはハイパーレスキューが必要なのだ。

### （4） 毎日やろう

次のことを毎日継続して続けよう。

- 問題・課題を毎日掘り起こし、対策アクションを作成しよう。
- 市場不具合状況／傾向の推移を毎日分析して全員で把握しよう。
- 市場不具合対策内容について毎日レビューしよう。
- 市場障害原因の解析および対策方法について毎日メンバーを指導しよう。

### （5） レスキュー体制

1） 指揮系統は一本化するようにしよう。
元のリーダーたちとレスキューに入ったリーダーたちが入り乱れた指揮系統は最悪だ。
レスキューに入ったリーダーが指揮をとろう。

2） 効率的かつ適正なプロセス遂行の指導や技術ノウハウの指導を行なう「キーマンチーム」を設置しよう。 このチームは開発チームの技術的知恵袋としての活動が期待されるのだ。

3） 開発残件が残っている場合は、開発者は「障害対応チーム」と「開発チーム」に分けよう。同じ開発者が障害修正と機能開発の二つの業務を兼任することは緊急対応の効率および正確性を著しく損ねてしまうのだ。

4） 評価テストチームは、結合テストおよびソフトウェアリリース管理を主体にする「開発評価チーム」と、客先業務運用評価を主体とする「運用評価チーム」に分けよう。 「開発評価チーム」は開発側の立場に立った技術的テスト業務を担い、「運用評価チーム」は客先における実運用的テスト業務を担うのだ。

## （6） リスクを管理しよう

プロジェクトが炎上しているのだからリスク管理はほとんどなされていなかったと判断してもいいだろう。 開発費、スケジュール、品質、技術的問題などの主要リスクの管理を徹底的に行なおう。 リスク発見時は即刻報告が行われるように全員に徹底しよう。

サンプルを下記に示したが各自で工夫して作成してみるといいだろう。

| | | | | | | 影響 | | 影響度 | | | | Weighted | | | | 予防対策後（予想） | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 項番 | リスクID | カテゴリ（タスク） | リスク（失敗モード） | 原因 | 発生率 | ローカル影響 | エンド影響 | CO 品質 | CO コスト | CO 納期 | 検出法 | 発生日数 | RPN | CO 品質 | CO コスト | CO 納期 | 予防策 | 発生率 | CO 品質 | CO コスト | CO 納期 | 発生日数 | RPN | リスクが問題化したときの対処計画 | 備考 |
| 1 | 1-1 | マルチベンダー開発におけるリスク | 担当外の保守の責任を負わされる | 保守契約または他ベンダーとの合意がされていない | | 開発者負担増加 | コスト超過 | | | | 進捗レビュー | | | | | | 保守契約を交わす他ベンダーとの合意を得る | | | | | | | 他ベンダーと保守内容の交渉をする | |
| 2 | 1-2 | マルチベンダー開発におけるリスク | マルチベンダー環境でのテストができない | 他ベンダーが情報（モジュール）を提供しない/提供が遅れる | | 待ちが発生 | 納期遅れ | | | | テスト事前準備 | | | | | | 他ベンダーからの提供時期について合意をとる | | | | | | | 顧客と費用、納期について交渉する。 | |
| 3 | 1-3 | マルチベンダー開発におけるリスク | マルチベンダー間のI/Fが決まらない。 | 他ベンダーのI/F検討遅れ | | 開発の出戻り | 納期遅れ | | | | 仕様レビュー | | | | | | I/Fレビューを行う | | | | | | | I/Fレビューを行う | |
| 4 | 2-1 | 注工程共通 | 開発途中の仕様変更 | 顧客とのコミュニケーション不足 | 1 | 開発の出戻り | コスト超過・納期遅れ | 2 | 1 | 1 | 仕様レビュー | 1 | 12 | 4 | 4 | 4 | ・顧客とコミュニケーションを良くとる<br>・要求仕様を文書化して顧客の承認をもらう | | | | | | | 顧客と費用、納期について交渉する。 | |
| 5 | 2-2 | 注工程共通 | 開発途中の仕様変更 | 仕様書加不明確 | 1 | 開発の出戻り | コスト超過・納期遅れ | 1 | 1 | 1 | 仕様レビュー | 1 | 10 | 2 | 4 | 4 | テスト仕様書作成による不明確部分の洗出 | | | | | | | 顧客と費用、納期について交渉する。 | |
| 6 | 2-3 | 注工程共通 | 進捗状況が悪い | PJメンバのスキルが低い | 1 | 開発の遅延 | 品質低下 | 2 | 1 | 1 | メンバからの報告 | 1 | 16 | 8 | 4 | 4 | PJ開始前にメンバのスキルをチェックする | | | | | | | トレーニングの計画と実施 | |

**［図表2. リスク管理表］**

### （7） モグラたたき状態からの脱出法 「仮説と検証の実行」

障害多発のシステムで多く見られる現象の一つにいわゆる「モグラたたき状態」がある。 つまり、修正しても修正しても別の問題が発生してくる状態だ。

このモグラたたき状態からの脱出方法のひとつに「仮説と検証」の実行という効果的な方法がある。「仮説と検証」っていっても何のことだか分かりにくいと思うので例をあげてみよう。 例えば障害の原因が何だか良く分からない場合に、その原因を仮に「未修正の不具合がまだ残っている」せいだとしてみる。 次にその原因の元となっている要因を考えられるだけ挙げてみるのだ。 そうすれば下記のように何となく真の原因らしきものが見えてきていくつかの対策案も浮かんでくるのだ。

**［仮説］** 未修正の不具合がまだ残っている。
**［要因］**
①全店舗のソフトウェアのバージョンが統一されていないため。
②今まで原因不明として調査を打ち切ってしまったものの中にやっぱりバグが含まれていたため。 未再現クレームの原因としてソフト基本構造の脆弱さが想定される。
**［対策］**
①全店舗のソフトウェアのバージョンを統一する。
②不具合多発業務を中心に基本設計書の再整備およびソース解析を行って弱点箇所の強化を行なおう。 弱点箇所として例えば、排他処理不正、非同期制御不正、タイミングずれ、リトライ処理抜け、タイマー値不正、レジストリ不正、設定間矛盾、メモリリソース・データ長・伝文長・カウンタ・ポインタなどのミニマム／マクシマム制御不正が考えられるのだ。

上記のような推定要因全てに対して同時に複数の対策を打つような多重性アクションの実行が有効だ。

### （8） 単発効果より複数効果をねらうこと

これは世によくいう「一石二鳥」というやつだ。 問題解決のうまいやつはいつもこれをやっているのだ。何か問題が発生したとき、みんな目の前のその問題の症状だけに目を奪われてしまうものだ。とにかく火が出たら水というふうに、とにかく目の前の火だけを消すことにやっきになるものだ。だけど現実的には目の前に火の手があがったときはすでにあっちもこっちも火の手があがっていることが多いものだ。 自分の尻にも火がついていることには意外とみんな気がつかないものだ。

例えば、ロジック不正のバグを直すときに単に不正なロジックを直すだけでなく処理速度も同時に向上するような直し方をするとかはバグのおわびにお客も喜ぶようなお土産つきの一石二鳥の上策だろう。

またこんなのはどうだろうか、小売店向けシステムでその店で販売されている商品をスキャンしたらエラーになってしまうようなとんでもないバグを出してしまった時。 ただ単にスキャンできるように修正しましたって持っていくより、お客様専用の商品コードテストブックを作成しこれで全ての商品をチェックしましたって持っていった方がいいと思わないか。バグ修正に加えてお客様専用の商品コードテストブックということでお客様は単純によろこぶだろうしこちらのテスト精度もあがるって仕掛けなのだ。

さらにもう一つ、値引きと儲けを同時に成立させる方法もあるぞ。 お客様からA機能の開発を見積ってくれと当社の営業経由で頼まれた。 普通に見積るとだれでも100万円くらいが値ごろ感だったとしよう。 実はこの機能は以前他の開発で似たようなものが開発されていたものが、ある事情でお蔵入りになっていた。

そこで営業部には100万円かかるところ営業渡しは50万円でいいから、お客様には80万円で販売するように話をつけた。 この商売は成立したのだ。 開発は原価ほぼゼロで約 50 万円の利益、営業部は 30 万円の利益、お客様は2割引で20万円助かった。 もとは開発の不良資産100万円だったものが3者ともにある意味で現実の利益を手にしたわけで、これは「一石三鳥」なのだ。

この「一石二鳥」のやり方のコツは、いつも単数思考ではなく複数思考でものごとを考えるということなのだ。対象は一つより二つで、一人ではなく二人でというふうに考えてみると仕事は楽しくなってくるものなのだ。

## 第3章　爆発の評価テスト工程からの脱出法

市場稼動工程におけるパニック状態よりはまだましだけど、不具合が続出してテストがあちこちで止まっている。 とうとう納期までに総合テストが完了できず上司はもちろんのこと客先もカンカンに怒っており契約解除もにおわせている。
爆発している評価テスト工程を収束させなければ間違いなく最悪の炎上の市場工程へと突入してしまうだろう。

あるプロジェクトの状況はこうだった。
やっとのことで製造工程を終了し評価テスト工程に進んだが単体評価、システム接続、総合評価と進むにつれて不具合は収束するどころか増加の一途をたどっていった。 不具合の内容も単純ミス（30%くらい有ったかも？）が頻発しており、評価テストの中断状態が何度も有った。
この様な状態で開発グループのリーダーは計3名しかおらず開発者40名、評価テスト者60名の総勢100名近くのメンバーに対して対応・指示する事が不可能になってしまった。 その結果、不具合修正数が発見数に追いつかなくなってしまい一時期は不具合残件が200件程度までになってしまった。
更にターゲット機器のパフォーマンスは到底商品レベルに達せず改善も手付かず状態で総合評価に突入してしまった。
すでに単体評価の後期に外注プログラマーの不足人員は充当済で作業者の数は十分な状態にありプロジェクトの指揮ができる中堅技術者をどう手当てするかが焦点となった。
キミならどうする？

### 1.　原因

対策を考える前に何故評価工程に入ってから大量の不具合が発生してしまっのかいくつかの原因について考えて見たい。

### （1） 統合プロジェクトマネジメントの不在

一貫して複数の組織および関連会社をコントロールするプロジェクトマネジャーがいなかった。
ソフトウェア開発の世界でも近年中国を代表としたオフショア開発は増える一方だよね。このような状況下で要件定義は元請のA社担当、設計は国内協力会社のB社担当、製造は中国のC社担当、評価は国内協力会社のD社担当という風に一つのプロジェクトの中の各工程を別々の会社が担当するケースがどんどんと増えていってるよね。つまりワンプロジェクト・マルチベンダーになってしまってるわけで、昔みたいに一社で全工程を開発する場合に比べてプロジェクトの管理やコントロールは非常に難しくなっているわけだ。
だけどこの状況がどんなに危険なことかを本当に理解している開発マネジャーやリーダーはあまり多くないのが実状なのだ。 極端な例をいうと、元請A社のプロジェクト責任者は他の工程は各協力会社に任せたのだから各社が責任をもってきっちりと仕事をしてくれるのが当然だと思っており、下請け各社の状況を何も把握していないような場合があるのだ。これがいわゆる”丸投げ”なのだ。 まとめ役が責任を放棄しているも同然なので下請け各社間に何んのコミュニケーションも存在しなくても不思議ではないよね。こんな状態でプロジェクトが成功するならこれは奇跡的なことだろう。100%失敗するだろうね。
キミは、自分は下請けのB社だから元請A社に何を言ってもむだだから黙っていてもいいと思

いますか。 キミならどうする？

### （2） 仕様決定者の不在

元請A社で全ての仕様を決定しなければいけないのに決定する人が少ないあるいは不在だった。 多くの仕様が未凍結あるいはあやふやなまま適当な設計が行なわれ適当な製造が行なわれノーチェックのまま評価工程に突入してしまった。 こんなことは有り得ないという人もいるでしょうが、こんなことが日常的にあちこちのプロジェクトで繰り返されているのが現実なのだ。
こんなので仕事がうまくいくわけもないのだ。

### （3） コミュニケーションの不在

複数の会社あるいは組織間の毎日ベースの情報交換が行なわれていないためチームがバラバラなのだ。それと同時にメンバーのモチベーションは大幅に低下しているのだ。
各工程の担当会社・組織間が自工程の前後と密なコミュニケーションの実施を怠り、さらにお互いの約束事が全くあいまいであったり守られていないのだろう。
こんなので仕事がうまくいくわけもないのだ。

### （4） 有効な開発管理・進捗管理がなされていない

元請A社は下請けB・C・D各社のプロセスを統合して管理していなかった。 下請けB・C・D各社も自律性がなくいつも受身的仕事のやり方しか行なわず自社担当分のプロセス管理もできていなかった。 こんなので仕事がうまくいくわけもないのだ。

原因として上記 4 点を挙げたけど、結局その根底にある共通の原因は開発プロセスが連帯・結合的な『共有分業』になっておらず無責任さに起因する『分離分業』になっているためなのだ。
最悪なのは、元請A社の責任者は、それでも外注にカネを払って仕事を全部まかせたのだから”丸投げ”なんて言われたくないなどといかにも正当論めいた認識しかもっていないことなのだ。 カネにまみれたマネードライブ思想の成れの果てがこれなのだ。 もう一度みんなに言っておきたいけど「あなたの仕事は何ですか？」。

## 2. 基本行動指針

上記の原因の対策を行うにあたっての基本的な行動指針は次のようだ。

① 関係各会社・組織との密なコミュニケーションを実施すること。
② 関係各会社・組織との約束を明確化・明文化することおよびその遵守。
③ 自工程の前後の工程・プロセスにも積極的に口を出すこと。

## 3. 総合テストにおける対策アクション

### （1） プロジェクト内の日次会議の実行

最初に実行しなければならないことはチーム内のコミュニケーション網の再構築なのだ。 混乱しているプロジェクト内ではコントロールされた指揮命令系統はすでに機能しておらずチーム内のコミュニケーションも有効に働いてはいないだろう。
誰も会話をしていないような沈黙のプロジェクトは危険だ。 沈黙のプロジェクトは必ず失敗するといっていいだろう。
毎日ベースの徹底的な課題掘り出しや対策会議を実施しよう。スクラム開発手法の日次スクラム会議のやりかたは非常に有効だ。つまりプロジェクトメンバー全員で、昨日実行したこと・今日実行する予定・今かかえている問題点について毎日短時間のコミュニケーションを繰り返していくことだ。この方法は単純だが忍耐が必要であり非常に強力な意思疎通を可能にして効果的・効率的プロジェクトの土台をつくりあげることができるのだ。
混乱しているプロジェクト内のコミュニケーションを回復するためには決められた時間に毎日ベースの日次会議を設定し、次の3つのことをすぐに実行しよう。

① まずキーマン全員を集めよう。
② 次に、集まったキーマンたちから現在の状況に関するQCDおよび人・モノ・金に関するデータを集め整理しよう。 不明なデータがあればすぐに調査するように指示を出そう。 本当のデータは信頼できる人からしか得られないと心得よう。
③ 次に、整理・分析されたデータからプロジェクト推進の障害になっている問題点を読み取りその優先度の高いものから対策アクションを実行しよう。

日次会議を効率的にやるには、昨日やったことは何で、今抱えている問題は何で、今日は何を実行する予定かの 3 点に絞って報告および情報交換を行なうようにすることだ。
効率的な日次会議はプロジェクト内のコミュニケーションを活性化すると同時にプロジェクト活動にリズム感を取り戻させてくれるのだ。 活動においてリズム感があることは非常に重要なことだ。 死んでいるプロジェクトにはリズム感はないのだ。
プロジェクトリーダは各サブリーダや担当者間の緊密な連携を築くために、毎日朝・昼・夜と担当者間を歩いてまわり彼らの不満・問題点・健康状態につき本音の情報収集に努めよう。

### （2） 顧客との日次会議の実行

プロジェクト内のコミュニケーションの回復と同時に重要なのが顧客とのコミュニケーションの回復だ。 失われた信頼を取り戻すことは容易なことではないが不可能ではない。 顧客の信頼回復のポイントは次のようだ。

**【顧客の信頼回復のポイント】**
**① 顧客との日次会議の実行**
顧客との共同評価を実行しつつ日次会議において毎日発生してくる課題・問題を顧客と共有し、毎日その場で判断および方針を決定していこう。 毎日ベースの顧客との即断・即決的なマネジメントサイクルはプロジェクトを必ず成功させてくれるのだ。

**② 顧客に対する説明のポイント**

対策および体制強化について見える形で提示すること。つまり、誰が、何を、どのように、いつまでに実行するかを明確に表明することだ。

**③ 顧客に対する説明は新リーダーにて**

すでに信頼を失った人間に顧客に対する説明を行わせてはいけない。今まで何度も顧客の期待を裏切ってきた人間をいつまでも顧客に対する説明者として使ってはいけない。 顧客に対する説明はレスキューに入った新たなリーダーが実行しよう。

**④ 顧客に協力をお願いしよう**

顧客にもプロジェクトの一員になっていただこう。 総合評価における実運用テストは顧客側のノウハウに依存するところが大きいのだ。 顧客に実運用テストのシナリオ作成および実運用テストへの参加を頼もう。 またプロジェクト遂行のボトルネックになっているだろう不明確な仕様等についての確定を一緒に進めるようにお願いしよう。

**⑤ 実行結果についての顧客への報告**

顧客に約束した対策等についてその結果をタイミングよく報告しよう。

前述の①および②の重要なポイントはチーム内あるいは顧客との良好なコミュニケーションの構築なのだ。

### (3) 独立したシステムインテグレーションチームを作ろう

開発メンバーを開発業務に集中させるために、開発チームから独立したシステムインテグレーションチームによる構成管理すなわちソース管理、モジュール管理、バージョン管理等を実行しよう。

### (4) キーマンチームを作ろう

未再現不具合や対策困難不具合の解決およびパフォーマンス問題の改善等の為に開発専任チームとは別に、開発技術者の相談役として上級技術者によるキーマンチームを作ろう。

キーマンチームは効率的かつ適正なプロセス遂行の指導や技術ノウハウの指導を行ない、開発チームの技術的知恵袋としての活動が期待されるのだ。

### (5) プロジェクトマネジャーの増強

プロジェクトマネジャークラスの不足はこの工程のみならず全工程のボトルネックになってしまう。すぐにでも他の優先度の低いプロジェクトあるいは協力会社などから適任者を追加投入しよう。このプロジェクトの優先度が非常に高い場合は他のプロジェクトの迷惑を恐れずに適任者を引抜く位の過激さも必要なのだ。

多くの外注メンバーを抱えるプロジェクトにおいては外注を指揮できるレベルのマネジャークラスの投入が効果的だ。 また社内からの人材補充が難しい場合は外注各社からマネジャークラスを採用し、プログラム製造等の直接的作業を禁じ外注各社分担分の開発管理のまとめに専念させ、実質的に社内のサブリーダとして開発管理が機能する様にすればよいのだ。

### （6） 開発サイドの総合テストおよび客先実施の運用テストの並行実施

一般的に客先による評価テストは形式的なものかあるいは全く行なわれない場合が多いものだ。 実際のところ開発側は客先における実運用にうとく、客先は開発技術に関しては素人なのだ。 このような事実からみても総合テストは開発側主体のテストだけでは全く不十分で、客先主体による実運用に則した運用テストが実行されなければ完結できないのだ。

客先が運用テストを実行してくれないのならば、少なくとも客先から実運用のシナリオを入手し開発側にてこのテストを代行しよう。

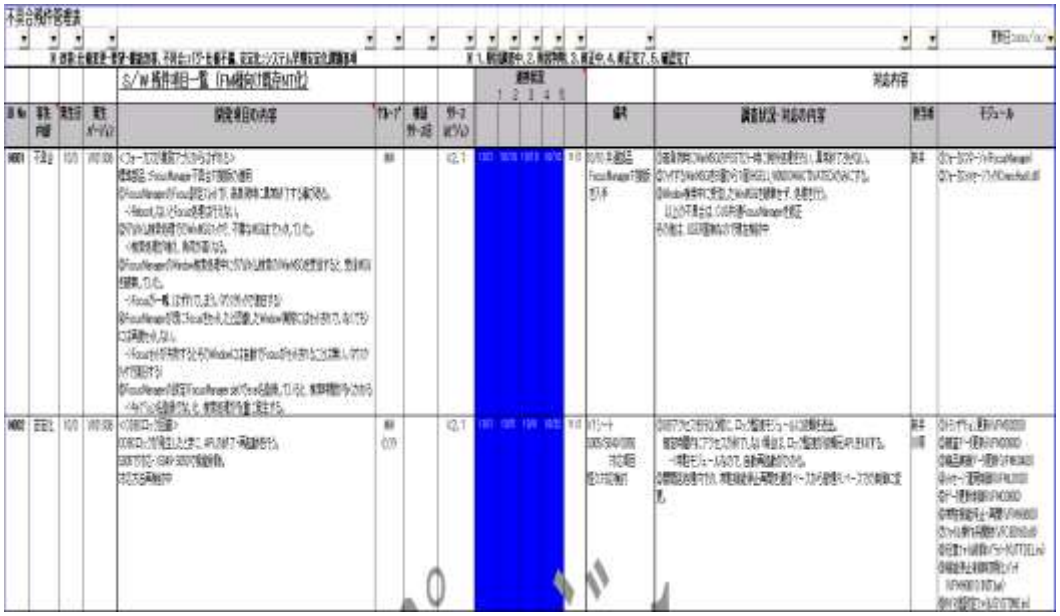
### （7） 不具合の修正と評価の進め方

不具合が続出している状態では早晩評価業務は行き詰ってしまい、評価のやり直しが続出したり、評価メンバーにおける手待ち時間が発生したりで数百万円単位で損失がでかねないのだ。

#### ① 不具合残件管理表の作成

まずは開発組織に対して重要機能の重要不具合順にどれを誰が何日までに修正およびリリースするのか全残件不具合に対して約束・記入させる必要があるのだ。 この表で不具合修正および評価の優先順位を決定するのだ。 また評価のボトルネックになっているバグを特定し、その為に止まっている評価項目数を示すことが必要だ。 これを実行すれば評価業務の進捗が大幅に改善されるだろう。

この残件管理表はプロジェクト全体で共同管理し日次会議で毎日更新しよう。



［図表3. 不具合残件管理表］

**② 評価進捗計画表の作成**

不具合残件管理表の作成ができたら不具合修正計画日程にしたがって、評価チームでの評価進捗予定表を作成しよう。 開発側の修正遅れがあれば、当然評価側の進捗も遅れるので日次会議において両者間で合意を取りつつ進捗管理をすすめれば両者の歯車がかみ合い始めるだろう。

**③ 不具合票の運用**

不具合の管理はデータベースだけの管理だけでは不十分なのだ。 実際の紙ベースの不具合票を一品一葉で作成し、不具合発生の都度評価から開発に渡し、修正結果を最終的に評価チームに戻す運用が必要なのだ。 データベースによる管理は当然分析等で必要だが電子ファイルでは皆の危機感が薄くなるのだ。 紙ベースでトレイに一杯入っている状況を実際にプロジェクト全員に見えるところに置いおこう。日次会議でもそのトレイを目の前にして話をすべきだ。全員の目に見える実感がプロジェクトを動かすのだ。

**④ 不明瞭仕様管理表の作成**

最終的に総合テストにおいて仕様の確認を誰が責任をもってするのか改めて元請会社に確認しなければならないだろう。 このままではあいまいな仕様がそのまま市場に出て多くのクレームが発生する可能性が大きのだ。

いままでも評価中に元請会社からしっかりした回答が得られていない仕様がたくさんあったはずだが、それらを全て「不明瞭仕様管理表」として都度元請会社に提示しておこう。

### （8） 総合テスト用ソフトウェアのこまめなリリース

総合テスト用ソフトウエアは一定期間たとえば1週間ないしは10日ごとにリリースを繰り返すことでプロジェクト活動に一定のリズム感を生み出し、評価がスムーズに進捗するようにしよう。

### （9） 大規模な評価メンバーの投入の実施

新規OSあるいは新規技術を多用したシステムにおいては通常規模の評価メンバーの投入では不十分なのだ。 一般的な派生開発における評価コストは全開発コストの10%以下だと推定されるが新規大型プロジェクトにおいては30%程度の評価コストが必要になってくるのだ。

### （10） 市場稼動版ソフトウェアのリリースの可否について

このソフトウェアが市場で問題なく稼動できるかどうかは実際に機器を操作した評価チームの生の声を聞くのが一番正解のようだ。 表向きは開発責任者や品質保証部門が市場導入の可否判断を行なうことになっているのだが、今までの経験から見てこれはあまりあてにならない。

## 4. 結合テストにおける対策アクション

結合テストにおける主な問題点とその対策は次の通りだ。
プロジェクト内の日次会議の実行および顧客との日次会議の実行については総合テストにおいて説明したのと同じだ。 チーム内および顧客とのコミュニケーションは日次ベースであることが望ましい。 毎日ベースのコミュニケーションを通じたマネジメントサイクルはプロジェクトを必ず成功させてくれるのだ。

### （1） 結合テストにおける客先検証の実施

一般的には結合テストでは客先との共同検証は行なわれないが、客先の協力が得られるようならば実行するように提案しよう。
結合テストレベルでの客先との共同検証の実施は早期の段階で客先の要求と製造物の一致を確認できる意味で非常に重要なのだ。 共同検証は7日～10日ごとのソフトリリースの繰り返しで実施すればよい。

### （2） 評価体制の強化

プロジェクトチームの強化については総合テストにおける対策アクションとして前述した、独立したシステムインテグレーションチームの設置、キーマンチームの設置、プロジェクトマネジャーの増強などが結合テストにおいても有効なのだ。 もう一度読み返してほしい。
そうはいってもプロジェクトにおいて十分なノウハウ保有技術者がそろっているとは限らない。 すぐにノウハウ技術者を充当できない場合は適切な人材を増強できるまでのつなぎとして評価体制の強化で乗り切るしかないのだ。 知恵の不足はやっぱり体力でカバーするしかないのだ。

### （3） メンバーのシフトあるいは増員

結合テストにおけるソフトウェアの品質の悪さに対する対策は総合テストの対策で説明した”不具合の修正と評価の進め方”がやっぱり役に立つのだ。 もう一度読み返してほしい。
また更なる一手として不具合が収束しない品質の悪い業務を洗い出し、その業務を中心に開発および評価メンバーの投入あるいは増員を実行しよう。 メンバーの配置を常に柔軟性をもって適時に実行できるようにするためにリーダーは各メンバーの状況を日々把握していなければならないのだ。 キミは部下の日々の状況が分かっているか？

### （4） 交代制勤務の実施

製造品質の低さやテスト機器の不足により結合テストが思うように進まない時は評価のための絶対時間を確保するために開発および評価チームの交代制勤務の実施が効果的だ。
交代制なしで強行した場合は開発メンバーの長時間勤務で時間を確保せざるを得ず、メンバーの疲労の蓄積によって更なる品質の低下および時間の浪費を招く悪循環に陥りやすい。

### （5） 管理業務のシステム化

プロジェクトの不具合管理、進捗管理、課題管理、成果物管理等情報の一元化および共有化の実施を行なうためのITシステムを導入しよう。 ただし最も重要なことは人と人による直接的コミュニケーションであり、ITシステムはあくまでもサポートのための道具だということを忘れないでほしい。 メールを出した、掲示板に貼り付けたなどはコミュニケーションをとったことにはならない。

## 5. 評価工程に関するリスク検出チェックリスト

評価工程におけるリスク排除のために次に示したリスク検出チェックリストを使うといい。

### （1） ヒトに関するリスク要因

☐ リーダーのプロジェクトマネジメント能力の不足
（外部交渉、タイムマネジメント、現場主義、見える化能力など）
☐ 計画性のないテスト業務
☐ テスト体制の不備
☐ メンバーの技術能力、ヒューマンエラー（うっかりミス）

### （2） モノに関するリスク要因

◎要因:ドキュメントおよびテスト環境の不備
☐ 要求仕様書の不備
☐ 詳細設計書の不備
☐ チェックリストの不備
☐ テスト計画書の不備
☐ テスト手順書の不備
☐ テスト環境の不備

## 第4章　混乱の製造工程からの脱出法


なんだか分からないが設計の基本的なところがコロコロ変更され製造チームはなんどもやり直しをさせられた。 製造チームは何が本当の仕様か疑心暗鬼でやる気もうせてしまった。 製造工程は大幅に遅れ、まだ未製造の部分がかなり残っているが、みんな何とかなるだろうと思っている。 このような状況を上司も客先も知らされていない。 キミならどうする？

製造工程およびその前の設計工程、要件定義工程における問題は評価工程および市場稼動工程における爆発炎上の予防策を中心とした話で進めよう。

### 1.　製造工程における問題

製造工程における開発能力の具体的問題を列挙してみる。

・ ベースプログラム構造の理解不十分なまま、あるいは今回の変更対象の機能の理解不十分なまま、あるいはデータ処理の仕組みの理解不十分なまま、あるいは変更部分の他との関連影響度の理解不十分なままソースコード変更に着手している。

・ 複数の開発者が関係しているにもかかわらず、早い段階で内容等の相互確認やレビューをせず、みなバラバラにソース修正にとりかかり、寄せ集めたソースで一気にテストをしている。

・ 開発者同士お互いに誰がどこを変更したのか誰も知らない。

・ 関数を理解不十分なまま無造作に使用している。

・ 検討不十分なままソースコードの”コピー＆ペースト”を無節操に行なっている。

・ 開発中に発見された、以前からの潜在バグについて検討・レビューもなく、いきなりソースコードの修正を行なっている。　　[Shimizu]

### 2.　製造工程における対策アクション

#### （1） コーディングガイドラインによる製造

プログラム製造は設計書だけに頼って複数のプログラマーがバラバラな手順で行なっても成功しないだろう。プログラム製造はコーディングガイドラインおよび設計書に基づいて実施しよう。 コーディングガイドラインは製造工程における問題に対処するコーディングの手引きであり手順書でもある。 また製造工程の次工程である評価工程においてはコード受入れ時に受入れ検査の基準として製造されたプログラムの妥当性チェックにこのコーディングガイドラインが使える。

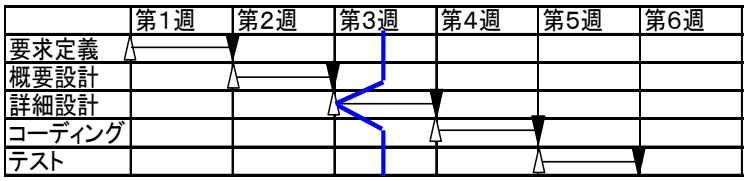
#### （2） 熟練開発者によるプログラマーの指導

コード製造にあたっては設計チームから独立した技術者によるプログラマーの指導が重要なポイントだ。 特に大きなプロジェクトにおいては数名の開発熟練者による専任のキーマンチームによる設計チーム・製造チームの技術コンサルテーションが非常に効率的かつ高品質のソフトウェア開発を達成する原動力となる。

### （3） 見える進捗管理の実施

ソフトウエアの進捗管理は一般的にガントチャートに進捗線を入れたもので管理されることが多いが、この表ではタスクの正確な進捗を見ることができない。 進捗線が1週間遅れの様に記入されていたとしても本当に1週間なのかどうか記入した本人ですら分からないのが実状だ。

冗談のようだが本当のことである。 ガントチャート上の最後の5%がいつまでたっても終了しなかったことを経験したことがないか。次のような警句もある。『ソフトウエアの開発スケジュールは、納期の直前までは順調に進む』。[Kinoshita]




［図表4. ガントチャート進捗表］

ガントチャートは大まかな進捗を見るのには適した表現方法であるが、おおまかさを補うために下記に示したモジュール進捗管理表等の併用が有効である。モジュール進捗管理表は縦軸に開発タスクを最小レベルまでブレークダウンしたソフトウエアモジュール名を記載し、横軸にはそのプログラムの見込みサイズ、開発各工程の終了予定日等最終リリースまでを表し実績を記入して予定日との差異を管理するものだ。最小プログラムモジュール単位までブレークダウンされているため主観の入る余地が少なく正確な進捗が見えやすいものとなっている。

機能モジュール進捗管理表(サンプル)

| 業務グループ | プログラム名称 | EXE, DLL, OCX等名称 | 開発Vol (Kbyte) 見込 | 実績 | 担当者 | 開発状況 | コーディング(C) 完了(予定日) | コーディング(C) 完了(実績日) | 単体評価(U) 完了(予定日) | 単体評価(U) 完了(実績日) | 結合評価(K) 完了(予定日) | 結合評価(K) 完了(実績日) | 不具合残件数 | 開発(残件)項目・処理日程 9月30日 | 10月10日 | 10月30日 | 備 考 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | XXX受信プロセス | xxxxx.exe | 50 | 60 | A, B | C | 6/30 | 6/30 | 8/15 | 8/15 | 8/30 | 8/30 | | △△△機能 | | | △△△後の継続処理 |
| | XXX送信プロセス | xxxxx.exe | 45 | 40 | A, B | C | 6/30 | 6/30 | 8/15 | 8/15 | 8/30 | 8/30 | | △△△機能 | | | ○○○のパフォーマンス向上 |
| | 通信プロセス | ○○.exe | 25 | 30 | A | U | 6/30 | 6/30 | 7/15 | 7/15 | 7/30 | 7/30 | | | | | |
| | リカバリー応答プロセス | ○○RECOV.exe | 35 | 30 | A, B | U | 6/30 | 6/30 | 7/15 | 7/15 | 7/30 | 7/30 | | | ○○○処理 | | スケジュールのスラップ分け無計 |
| | △△受信プロセス | △△RECV.exe | 25 | 20 | A | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |
| | △△送受信プロセス | △△SEND.exe | 20 | 30 | A | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |
| | △△起動開台せ | △△INI.exe | 30 | 10 | B | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |
| | FTPプロセス | ○○FTP.exe | 20 | 30 | C | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |

| | | | | |
|---|---|---|---|---|
| 最小モジュール単位までブレークダウンする | 最終把握の見込み／実績を行う | 担当者の明確化 | 工程別進捗を予定／実績日程にて行い、色彩にても直感しやすくする | 品質状況の把握を行う | 残件処理日程を明確にする | 備考記述による問題点のフォロー |

［図表5. モジュール進捗管理表］

### （4） 遅延リカバリーの有効な対処方法

プロジェクトマネジャーに限らず皆さんの口癖は”あ～時間がない！”なのだ。 それもプロジェクトの後工程になるに従って悲鳴のように聞こえる。 最後の段階になっては、いくら人を投入しても遅れを挽回するのは非常に困難である
ことは衆知のことである。 次の文章をあながち冗談と言い切れるだろうか。
『遅れているプロジェクトに人員を補充すると、スケジュールはさらに遅れる。補充しないで放置すると、プロジェクトは崩壊する。』［Kinoshita］

ある仕事量を消化するためには”人数X時間”が必要である。この観点から時間が少なくなった分を補うためには人数を増やせばいいという発想が生まれる。 しかしながら単純な製造工程の遅れの場合はこの対策でも効果はあるが、多くのプロジェクトの遅延の原因はシステムに熟知した設計者の不足あるいはプロジェクトを指揮できるプロジェクトマネジャーの不足の場合が多い。やたらプログラマーの数を増やしても一向に進捗しないのはそのためだ。遅延部分に適任・適数の設計者・プロマネの投入で大幅に遅延回復が可能だろう。

### （5） 遅延を発生させないためのポイント

- ☐ 正確な要件定義に基づく正確な見積りの実行。
- ☐ リスクの把握、見積り条件の明示は必須。
- ☐ 仕様変更のコスト・期間は別途見積りとする。変更管理の徹底が必要。
- ☐ 単純な製造工程の遅延はプログラマの増員で対応する。
- ☐ 設計品質による遅延は設計者の増強で対応する。
- ☐ 開発プロセスによる遅延はプロマネの増強で対応する。

後工程における対処は前工程の何倍ものコストと時間がかかることを肝に銘じておくこと。

### （6） 悪い状況だからこそ早めの報告を 「ほうれんそう」の励行

言い古された言葉だが、状況が悪ければ悪いほど早め早めの報告・連絡・相談（ほうれんそう）が必要だ。 しかしながら現実は、すでに当事者においては完全にリカバリー不能の最悪の状態に陥りながらもまだ何とかしようとしており、ついに外部からの指摘により問題が発覚する場合が非常に多いのだ。
何故早く相談しないのだろうか。 心理的障壁として“他人・上司に悪い状況を報告したくない “いつも良い子でいたい症候群” “悪くなったのは自分のせいではない”等々がありプロジェクトマネジャー自身がこの穴に落ち込むことだけは絶対に避けなければいけない。 また開発プロジェクト内の日々のフェイス・トゥ・フェイスの会話を絶やさない様にし、風通しを良くすることが必要だ。

### （7） 頑張ることだけが良いことではない

先週までの報告では順調だったプロジェクトが突然1ヵ月遅れになってしまったとの報告を受けたことがないだろうか。理屈に合わない話である。先週の報告が正しければ遅れは最大1週間であるだろうし、1ヶ月遅れの報告が正しければ先週すでに3週間遅れていたことになる。

このようなことが起きる原因は、悪い状況をなるべく報告したくないためぎりぎりまで頑張っていたというような場合が多い。遅すぎた報告は実状を見えなくする行為である。見えないということはスピードあるいは速い対応を要求されるソフトウェア開発においては致命傷となるであろう。

不測の問題が連続して発生したプロジェクトにおいてプロジェクトマネジャーは、既に現状体制ではリカバリー不能の状態に陥りながらもまだ何とかしようとしており，ついに外部からの指摘により実状が発覚する場合が多いものである。体力・気力勝負で完全にだめになるまで頑張り通すことは褒められることではない。ソフトウェア開発は体力・気力で解決できるほど単純なものではないだろう。メンバーの多くが「頑張ります」の発言をするようになったプロジェクトは危険状態である証拠と認識した方が良いだろう。

人間の心理として，上司に悪い報告をしたくないとか，自分だけの責任ではないとか，良い評価に傷をつけたくないとかが有り，本当の最悪状態になるまで問題を上げてこない傾向がある。プロジェクトマネジャーはプロジェクト内の人間関係の風通しを良くし，早め早めの報告・連絡・相談が行われる組織環境を確立すべきであろう。プロジェクトにおいてさまざまな問題が発生してくることを当然のこととして受け止め，日々問題の発見・対処のサイクルを講じるようなアジャイル的変化即応型プロセスの実行が必要であろう。

## 3.　製造工程に関するリスク検出チェックリスト

製造工程における主なリスクとしては次のようなものがある。リスク排除のためにこのチェックリストを使うといい。

### （1） ヒトに関するリスク要因

☐ リーダーのプロジェクトマネジメント能力（外部交渉、タイムマネジメント、現場主義、見える化能力など）
☐ 安易な既存プログラムの改修
☐ コーディングルール・手順書など製造工程の統制不足
☐ メンバーの技術能力、ヒューマンエラー（うっかりミス）
☐ 構成管理技術力不足

### （2） モノに関するリスク要因

ドキュメントの不備（要件定義書・設計書・チェックリスト・手順書など）

☐ 要求仕様書なしの製造
☐ 基本設計書なしの製造
☐ 詳細設計書なしの製造
☐ コーディング規約（ルール）なしのコーディング作業
☐ 構成管理手順書の不備
☐ ソースコード修正管理書の不備

### （3） 情報に関するリスク要因

☐ あいまいな開発範囲（スコープ）
☐ 情報の不備・不足（マネジメント情報、技術情報、過去の失敗情報）

## 第5章　あせりの設計工程からの脱出法

　仕様凍結がずるずると遅れており設計が完了できていない。もうスケジュールの期限になったが未凍結の仕様がまだ3割も残っており未着手の設計も5割も残っている。 このような状況を上司も客先も知らされていない。 キミならどうする？
　設計工程における問題は次工程である製造工程問題の予防策および評価・市場稼動工程における爆発炎上の予防策を中心とした話で進めたい。

### 1.　設計工程における問題点

#### （1） 設計ドキュメントの実態

　実際に設計ドキュメントがどんな風に扱われているかもう一度開発者たちの生の声を列挙してみよう。

◎ドキュメント品質のレベル

・ 誰のために書かれたものか分からないドキュメントが多い。
・ 作成者以外が見ても分かる内容・レベルになっていない。
・ 分かりにくいドキュメントばかりしかない。
・ 時間に余裕がなく、とりあえず自分が分かるレベルの内容記述だけになった。客先・SEや他の開発者・協力会社が理解できる内容にはなっていない。
・ 客先要件の実現方法が明確に書かれていない。
・ 機能を実現するために必要な情報が設計書に記述されていない。
・ 開発内容に対するコンセプトや背景や経緯についての資料がない。
・ 既存ソフトの流用の可否を判断できる資料になっていない。
・ システムの全体を表した資料がない。
・ モジュール間の連携部分の説明が貧弱。
・ 品質は上流工程（要件定義書・設計書）で決定する。テストだけでは品質は改善しない。
・ 客先要件や仕様の記述が乏しくロジック記述に偏っている。運用テストに使えない。
・ 障害解析に使用できるレベルの内容になっていない。
・ 障害報告書で技術的側面から図表等を使用した報告ができていない。

◎ドキュメントのメンテナンスおよび管理

・ 内容が更新されていない。
・ 前任者がドキュメントを更新していなかったから、自分もできなかった。
・ 整理・管理されていないため調査・検索ができない。

◎ドキュメントに関する意識

・ 納期が間に合わなくなったらテスト作業やドキュメントを省略してしまう。
・ 協力会社に設計以降の工程を丸投げする方が安く・納期短縮もできた。
・ バージョンアップ開発時はソースベースで調査・改修でも何とかなった。
・ 発注元からドキュメントを要求されることもなくドキュメント作成工数が削減できた。
・ 時間も人もいなくて、限定した内容だけの記述しか作成できなかった。更新が必要な資料にも手をつけられなかった。

・ 発注元が協力会社に丸投げする状況が続き、発注元の担当者が自ら設計する技術がなくなってきた。

これらの記述は筆者が創作したものじゃない。何ともヒドイ状況だ。いろいろ事情はあるだろうが、結局は正しい仕事をしていないのだ。これはプロの仕事とは言えない。

## 2. 設計遅延に対するアクション

### (1) 仕様凍結遅れで設計業務が開始できない場合

設計能力自体の問題で設計業務が遅れることはあまりない。一般的に多いのは基本的な要件の仕様がいつまでたっても決まらず設計開始の時期が来ても設計に取り掛かれない場合が多い。こんな時あせってとりあえず決まったものから設計を進めようとするのはチョット待った。
やりやすいものから手をつけてはいけないのだ。プライオリティの高いものからつまり顧客の要求度の高いものから手をつけていくのが鉄則だ。この鉄則は絶対に破っちゃだめなのだ。
基本要件に影響しない枝葉末節の仕様から着手してもどうせ基本仕様が決まるまでに二転三転してやり直しが増えるだけだ。
重要基幹部の仕様が決まらないうちに端々の機能を自分流の想像で作り始めてしまい、本当の仕様が決まった時には内容が想像とは全く違ったものになり、それまでに作った設計図とコードはすべてやり直しなんてことがよくあるだろう。やり直しする時間があるくらいなら最初から何にも手を付けずにおくのが正解だろう。
こういった場合に開発チームが取るべき対策は設計者全員を要件定義に投入し顧客要求度の高い順に仕様を凍結させていくことだ。オレは設計者で要件定義はSEの仕事だなんて言ってる場合じゃないだろう。 またウチの会社は設計業務以降しか請負ってないので仕様が凍結されるまで待ってますっていう態度もマズイと思う。結局下請けが時間の帳尻あわせをさせられるハメになってしまうのだから。そうなることが分かりきっているのにみんないつも”待ち”の姿勢なんだ。仕様凍結に参加するしか進捗する方法はないのだ。もちろん有料で参加するのだ。

### (2) 仕様の難易度が高くて設計業務が遅延している場合

この場合の対処方法は二通りある。一つはレベルの高い設計者の投入であり、もう一つは顧客との話し合いが可能なら仕様の簡易化ないしは一部機能のトレードオフだ。
設計遅延対応に投入費用を惜しんではいけない。ここで投入しないと後の工程でもっと高いものにつくことになるのだ。

### (3) 手抜きはいけない

設計業務が遅れ出すとみんながよくやることは必要な設計ドキュメントの作成の手抜きをするということだ。ひどい場合は設計書自体が作られないことだってある。これじゃまともに動くものが作れるわけがない。
プログラマーに渡されたものは文章だらけの仕様書と若干の参考資料だけで、運用フローもデーターフローもプロセスフローもソフト構造図も何もない。 設計工程での手抜きは製造工程・評価工程での大幅なやり直し作業ないしは手待ち時間を発生させ膨大な損失を生むと同時に品質劣悪な欠陥ソフトウェアを作ってしまう結果になるのだ。

### （4） 開発文書は先に作るものじゃないの？

大切なのはプログラムだけか？ ドキュメントは一応作っておけばいいものか？。 本来、要件定義書や設計書は何のためにあるのだろうか。 開発ドキュメントは、開発者が正しく設計・開発できるように客先要求を開発の言葉に変換したものだ。何をどういう風に作れば良いかを可視化したものであったはずだ。決してプログラムを作りながらとか作った後に作るものではないはずだ。開発ドキュメントは目標にたどりつくための地図のようなものだ。先に地図がなければ目的地には到達できないだろう。どこの世界に目的地らしい所についた後で、後付け的に地図をつくる人がいるだろうか。そのような人やチームがあれば道に迷うか遭難するかのどちらかだ。

### （5） ドキュメント・文書の効用

直接当人に話をしなければいけない人事的・私的な案件を除いては、口頭によるよりも文書・メモ等によるコミュニケーションが結果として余計な時間をセーブできる効用が大きい。文書やメモの効用として下記が考えられる。

- 証拠が残り、「いった・いわない」の問題がなくなる。
- 客観的な情報を伝えやすい。
- 相手の時間にしばられない。
- 重要な事とそうでない事の区別や選択がしやすい。
- 文章にする事で論理的な思考がきたえられる。
- 何回も説明を繰り返す必要がなくなる。

### （6） 設計遅延に対するアクションのポイント

ポイントは次に示す通りだ。

□ やりやすいものから手をつけてはいけない。プライオリティの高いものからつまり顧客の要求度の高いものから手をつけていくのが鉄則だ。
□ 設計者全員を要件定義に投入し顧客要求度の高い順に仕様を凍結させていくことだ。
□ 難易度の高い開発にはレベルの高い設計者の投入が必要。
□ 困難な仕様に対しては仕様の簡易化ないしは一部機能のトレードオフの提案も考える。
□ 設計遅延に費用や人材の投入を惜しんではいけない。
□ 設計ドキュメント作成の手抜きは絶対にしない。

## 3.　設計工程に関するリスク検出チェックリスト

設計工程におけるリスクには次のようなものがある。これらのチェックリストに基づいて、設計工程に着手する前にこれらのリスクを排除しておく必要がある。

### （1） ヒトに関するリスク要因

◎要因：他者依存的姿勢（自律性の放棄）
- □ ：協力会社への丸投げ（開発統合責任・製品最終責任の放棄）

◎要因：組織能力不足（未熟な組織文化、戦略の欠如）
- □ 無定見なオフショア開発

◎要因：リーダーのプロジェクトマネジメント能力（外部交渉、タイムマネジメント、現場主義、見える化能力など）
- □ 未経験技術者の大量投入
- □ 基本設計レビューの未実施
- □ 安易なフレームワークの改修

◎要因：メンバーの技術能力、ヒューマンエラー（うっかりミス）
- □ フレームワーク構築能力
- □ 異常系技術力の不足
- □ タイミング設計能力の不足
- □ 排他処理設計能力の不足
- □ 非同期制御設計能力の不足
- □ リトライ設計能力の不足
- □ ログ技術力・認識不足
- □ オープンソフト対応技術力不足
- □ メモリーリーク対応技術力不足
- □ 性能（パフォーマンス・レスポンス）技術力不足

◎要因：プロセス管理の有無
- □ プロセス管理不足
- □ 基本設計レビューの未実施
- □ 性能要件（パフォーマンス・レスポンス）レビューの未実施

◎要因：コミュニケーション能力（阻害・ギャップ）
- □ 協力会社・オフショア先などとのコミュニケーション能力

◎要因：関連部署との連携不足
- □ 組織間の協調性・コミュニケーション

### （2） モノに関するリスク要因
- □ 要求仕様書なしの設計
- □ 貧弱なドキュメントやガイドライン

### （3） 情報に関するリスク要因
- □ あいまいな開発範囲（スコープ）
- □ あいまいな要件
- □ 情報の不備・不足（マネジメント情報、技術情報、過去の失敗情報）

## 第6章　待ちの要件定義工程からの脱却

開発側は、客先が要求仕様を提示できない、あるいは要求内容が二転三転してなかなか仕様が固まらないと言っている。 客先は仕様を提示してくるのは開発の仕事だと思っている。 客先もSEも開発もだれもが仕様を決めるのは自分の仕事じゃないと思っている。 もう設計開始の時期がきたがまともに仕様らしきものが決まったのはわずかばかりで残りの仕様があとどれくらいあるのかも想像できない。
このような状況にあっても客先も開発組織も何らの危機感も抱いていない。 まだまだ時間はいっぱいあるから、後工程でなんとかなるだろうと思っている。
キミならどうする？

要件定義工程における問題は次工程の設計・製造工程の問題発生の予防策および評価工程・市場稼動工程における爆発炎上の予防策を中心とした話で進めたい。

第５章でも要件定義工程における仕様凍結遅れの問題および対策について触れたが、要件定義工程はソフトウェア開発工程の中で最も重要かつ難しい業務である。
要件定義はプロジェクトの総力をあげて対応すべき業務であり必要なら設計者たちも積極的に投入すべき仕事なのだ。

### １．要件定義工程の問題点

要件定義工程の主な問題点を下記に示した。

・ 要件定義ができる人材が不足している。
・ 要件定義ベースラインの確立および変更管理がなされないまま開発行為に突入し大きな問題を発生させている。
・ 要件定義工程は利益・品質・納期確保の最大の源泉であるという認識が不足している。
・ ソフト製造会社とはいえ要件定義作成などの最上流工程への参画を要請されている。 オフショア製造が拡大する中、ソフト製造開発会社から脱皮し、より顧客に近い業務へとシフトしなければ生き残れない。
・ 従来、客先の要求は、要件定義、要求仕様等さまざまな呼称にて呼ばれていたが、その内容についての分析・規定は現時点においても明確なものは存在していない。そのために作成者のレベルにより要件定義書のレベルもまちまちであり、極端な例では“〇〇システムと同様”、“△△打合せ議事録”等の記述にて済まされている場合もある。

## 2. 仕様凍結をスムーズに進行させるには

### （1） 顧客との密接な仕様検討会の実施

仕様の答えは顧客が持っているのだ。 顧客と話をしなければ仕様は永久に決まらない。 だから顧客とのフェイス・トゥ・フェイスによる仕様検討会を集中的に実施する必要がある。例えばある大きい規模の開発では週1回ベースにて約20回の仕様検討会および集中合宿検討会2日を実施してやっと基本的な仕様の凍結に成功した。

### （2） プロトタイプ機による仕様確認

正規の開発に入る前にビジネスロジックなしで良いからある程度の操作および動作確認ができるプロトタイプ機にて顧客と話をすることができれば、書類では確認できない内容についてある程度正確に顧客の要求を把握できる。

### （3） 要件定義書の内容の明確化

ここでは要件定義書は顧客の要求するソフトウェアの機能を下記項目について記述したものだとしておく。 要求される開発の規模や範囲に従って必要な項目は全て記述・作成しなければならないのだ。

・ システム企画の背景・目的
・ システム構成図（全体・店舗機器・ネットワーク）
・ 構成（ハードウエア構成、ソフトウエア構成）
・ コード体系
・ すべての要求仕様および機能の記述
・ 業務運用フロー
・ 入出力情報定義（入力画面、出力画面、リスト、帳票書式、等）
・ データ定義概要（集信／配信データ、オンラインフォーマット、メモリバランス、データ件数、データ保有期間等）
・ 要求性能概要
・ 条件／制限事項（運用条件、制限事項、信頼性、拡張性、セキュリティ）
・ スケジュール（稼動スケジュール、導入展開スケジュール）
・ 従来システム構成、従来システムからのデータ移行など

### （4） 要件定義業務の実行

顧客要件は顧客の要求度の高いものから定義しなければならない。顧客にとって価値の高いものが要求度の高いものなのだ。開発の全工程に渡ってこの優先度に従って開発実行ないしはリソースの配分が行われることが顧客の満足度を最高のレベルで獲得するための合理的な開発となる。

顧客要件は顧客価値の重要度順に、たとえばMust項目、Need項目、Want項目のように3分類しよう。

- **Must項目**； 無条件必須項目である。
- **Need項目**； 条件付必須項目である。条件とは顧客との話し合いで分割リリースが可能なもの、あるいは他の代替機能の提案によってはトレードオフ可能なもの。
- **Want項目**； 条件付必要項目。条件とは顧客との話合いで次回開発時に実行可能なもの（次回開発時の話し合いではドロップされる可能性がある）、あるいは他の機能を実行する代わりにドロップ可能なもの（トレードオフ項目）。

Must項目の要件の提示および仕様化は開発着手前に完全に凍結されている必要がある。Need項目、Want項目についても、実施が確定したものについては、それぞれの開発着手の前には要件の提示および仕様化は開発着手前に完全に凍結されている必要がある。要件開発における要件リスク管理表のサンプルを下図に示した。

要件開発リスク管理MAP
作成更新日 XXXX年XX月XX日

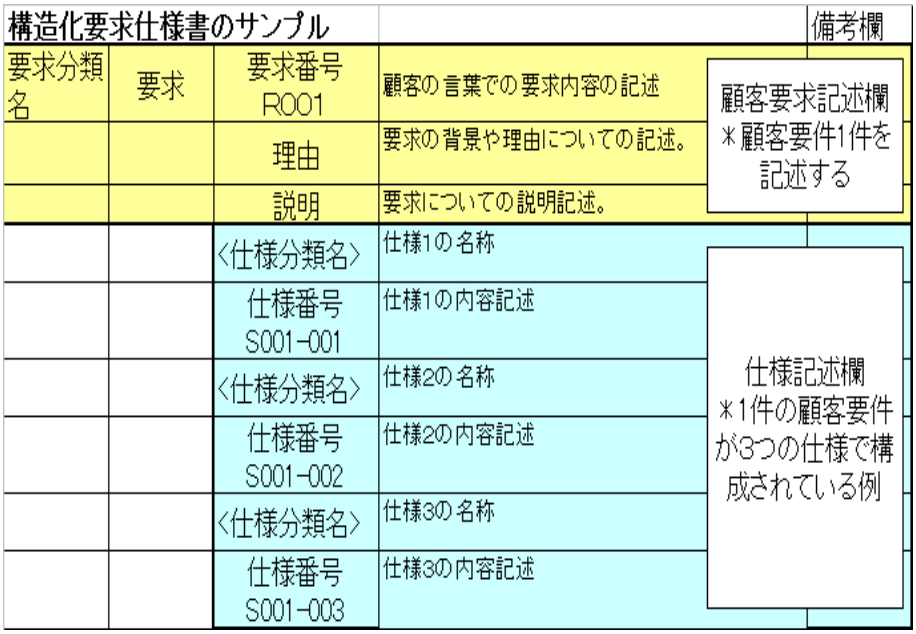
| 要求番号 | 要求内容 | 顧客重要度 | 技術難易度 | 仕様凍結度 | 仕様凍結目標期日 | リリース | 未凍結仕様問題記述 |
|---|---|---|---|---|---|---|---|
| 1 | 要件概要記述:顧客要求内容の概略ポイントを記述する。 | MUST | 中 | 80% | 2010/10/1 | 1 | ・未仕様凍結の理由および内容の記述:どのような要求内容が決まっていないのか理由も含めて記述する。<br>・仕様凍結目標期日までに間に合いそうか、そうでないか等対策も含めて記述する。 |
| 2 | 要件概要記述 | MUST | 高 | 30% | 2010/10/1 | 1 | ＊顧客重要度が高い項目で仕様凍結度の低いものは高リスクである。これ以上仕様凍結度が進捗しないのなら、顧客との話し合いにて重要度を下げ2次リリース等に移動するアクションが必要。 |
| 3 | 要件概要記述 | MUST | 中 | 60% | 2010/10/1 | 1 | ＊顧客重要度が高い項目で仕様凍結度の低いものはリスクである。リスクの度合いによる色分けを行う。背景色赤は危険・至急アクション要、黄色は要注意・アクション要の項目である。 |
| 4 | 要件概要記述 | NEED | 低 | 50% | 2010/11/1 | 1 | |
| 5 | 要件概要記述 | NEED | 中 | 30% | 2010/12/1 | 2 | ＊顧客重要度および仕様凍結度の状況につき顧客との話し合いにて、2次のリリース項目に決定した。 |
| 6 | 要件概要記述 | WANT | 低 | 20% | | ドロップ | ＊顧客との打合せにて開発項目からドロップされた。 |

**[図表6. 要件リスク管理表]**

要件開発リスク管理表は、顧客要件の重要度の管理および仕様凍結状況の管理に使用するものだ。 顧客重要度が高い項目で仕様凍結度の低いものは高リスク項目であり、これ以上仕様凍結度が進捗しないのなら、顧客との話し合いにて重要度を下げる等のアクションが必要だろう。 リスクの程度による色分け表示は分かりやすい。また仕様凍結進捗管理のために、未凍結の理由・内容・対策等の記述も必要となる。

要求仕様書は、それぞれの顧客要件について、その要求内容の記述と、これを仕様化した仕様内容の記述のペアとして記述される。

要求内容は、顧客要求内容の記述ごとに、その要求に対応する仕様化記述を階層構造的に表示したものが有効だ。記述内容は、顧客要求記述、要求内容の説明、要求内容の理由・背景、要求番号、要求分類番号、要求分類名、要求の顧客価値度、等で構成される。また仕様内容は、要求内容とリンクした形にて仕様記述、仕様番号、仕様分類名、仕様分類番号、等で構成される。構造化要求仕様書のサンプルを下図に示した。




**[図表7. 構造化要求仕様書]**

### 3. リスクヘッジについて

要件定義工程でのもう一つの重要な仕事はプロジェクトに関係するさまざまなのリスク排除だ。

#### (1) リスクヘッジのポイント

リスクヘッジのポイントをいくつか列挙してみよう。

□ できない事をやろうとしてはいないか。できないことはやってはいけないし、やらせてもいけないのだ。一人で判断が困難な場合はチーム・組織で判断しよう。
□ リスクは基本的に人により発生させられるものであると言う認識を持とう。 例えば”低開発費”自体がリスクではなく、”低開発費を招きそうな人間”がリスクなのだ。
□ リスクと課題の違いが分かるかな。
まだ発生するか・しないか分からないが発生しそうな問題を「リスク」と言い、実際に発生してしまった問題を課題と言うのだ。 だからリスク管理表と課題管理表は別ものなのだ。
□ 取ってはいけないリスクと取るべきリスクについて
判断が難しいが、自分あるいは自組織の実力にて何とかこなせるレベルを越えるものは”チャレンジ”の域を超えて”無謀”な領域に入る。 反対にチャレンジ可能なリスクは積極的に取らなければビジネスの拡大はないのだ。

#### (2) リスクヘッジについて

リスクの回避は上流工程における実施がもっとも効果的だ。
上流工程におけるリスク回避に必要な重要成果物は下記3点だ。

1. **要件定義書** ： 何(**What**)を作るのかの定義書。
2. **基本設計書** ： どう(**How**)作るのかの定義書。
3. **見積り回答書**(開発費、開発期間、見積条件・範囲) ： いくらでいつまでに作るのかの契約書。

この3つの成果物の精度でプロジェクトの成否の大半が決定するのだ。

「超短納期」、「少ない開発費」、「難度の高い仕様・技術」がリスクなのだろうか。自分のプロジェクトが、超短納期・少開発費・高難度仕様であると認識された時点では、これらはすでにリスクではなくなり、プロジェクトにおける現実化した課題になってしまっていると思った方がよい。

これらの課題は自然発生したものではないだろう。誰かが要求し誰かが受け入れたものなのだ。誰かとは、顧客のA氏であり、開発責任者のB氏であろう。このような困難な状況は、強引なユーザと能力不足の開発責任者の組み合わせの中で発生しがちな事象なのだ。これを決めたのは人なのだ。

**常にリスクはモノではなく人にあると言う強い認識と対応が必要なのだ。**

金・時間・モノにリスクがあると思っているプロジェクトマネジャーは結果として既に確定してしまった開発予算・開発期間を何とかしようと七転八倒の苦労を強いられるだけなのだ。

低開発費・短納期等リスクを回避したい場合はそれを要求してきそうな人がキミにとってのリスクであり，リスクヘッジをしたければ、それらのリスクが課題として決定する前に、その人と交渉するしかないのだ。

多くのリスクは人に依存しており、リスクヘッジを可能にするのはリスク管理表ではなく、 自分自身との戦い，あるいは交渉相手との戦いの中にしか存在しないのだ。 人との交渉を回避したい臆病さはリスクを増大させてしまうと思った方がいい。「リスク管理」は結局、自分自身をも含めた「人の管理」に他ならないのだ。

### （3） リスクの可視化

いずれにしろ何がリスクなのかを明示したリスク管理表の作成は必須だ。
リスク管理表における管理対象項目として下記がある。

・リスク記述欄 ： 品質、開発費、納期、対外部署間問題、工数問題、人材問題、その他問題
・物件名、ユーザ名、依頼元名、工程名、開発番号、開発費、開発スケジュール概要、
　責任プロマネ名、担当メンバー氏名、などだ。
リスク管理表のサンプルは、第7章 ６．（4）を参照のこと。

### （4） 自ら一歩前に出ればリスクは半減し、一歩後退すればリスクは倍増する

リスクを感じたらどうすべきだろうか。リスクはある程度問題が見えてから対処した方が良いと思っている内に本当に問題化してしまうことが多い。リスクは放置しておくと時間の経過とともに増大し複雑化するものだ。

ものごとの実行順は不確実なこと、リスキーなことから先に着手した方が良い。リスクの解消にあたっては個人的にも大きな労力と緊張感を強いられることが多いが、一歩前に出て前線あるいは現場において当事者である相手と直接交渉する方法がリスク回避の最も効果的な方法となる。経験則から見ても自ら一歩前に出ればリスクは半減し、一歩後退すればリスクは倍増するものなのだ。

### （5） リスク減少開発方式　アジャイル

**アジャイル開発はリスク減少システムなのだ。**

アジャイル開発はその特徴である、顧客価値の優先順位の高い順にインクリメントな開発（幾つかに分割されたソフトウェアを順次リリースしていく開発方法）が実行される点において、都度正しく動作するソフトウェアを顧客に見せる点において、フェイス・トゥ・フェイスの密なコミュニケーションを実施する点において、スプリント（短期の開発）における目標の明確な設定が行われる点において、下記のリスクを確実に減少させるシステムであると言えるのだ。

- 顧客満足の未達成リスクの減少
- 未完成機能の発生リスクの減少
- 不正確な見積りと目標に対するリスクの減少
- 問題の長期化リスクの減少
- 開発期間内の開発未完了リスクの減少
- 開発者における過度の労働リスクの減少と顧客における過度の期待に対するリスクの減少

（[Agile Scrum]、p120～121）

誤解して欲しくないが、ウォーターフォール開発がダメだということではない。顧客との密接なコミュニケーションや高い開発力の維持や変化に柔軟に対応する能力は、どちらの開発方式においても必須であり、その能力のないチームがアジャイル方式を型どおりに実行しても必ず失敗するだろう。

## 4. 要件定義工程に関するリスク検出チェックリスト

要件定義工程は何を開発するのかを決定する重要な工程であることははみなさん百も承知のはずだと思う。何を開発するのかが決まらなければ、どのように開発するのかを決める設計工程に進むことはできない。想定仕様による事前着手はプロの仕事ではない。

次に要件定義に関するリスクを挙げてみる。

### （1） ヒトに関するリスク要因

◎要因：他者依存的姿勢（自律性の放棄）
- □ 顧客からの要求仕様提示待ちの姿勢の結果、納期遅延・品質悪化を招く。

◎要因：上位マネジメントの関与不足
- □ 顧客交渉戦略･能力不足
- □ 大幅な仕様追加要求の丸呑み
- □ 大幅な開発費削減要求の丸呑み
- □ 大幅な納期短縮要求の丸呑み

◎要因：ユーザーの参加・協力度不足
- □ 顧客の参加・協力度が低い

◎要因：組織能力不足（未熟な組織文化、戦略の欠如）
- □ 見積りプロセスの手続きルール違反

◎要因：見積り能力
- □ 既存資産流用の可否判断
- □ 顧客要件の精査
- □ フィット＆ギャップ調査能力（要求仕様と保有技術の乖離の調査分析）

◎要因：要件定義能力
- □ 要件定義能力不足

◎要因：リーダーのプロジェクトマネジメント能力（外部交渉、タイムマネジメント、現場主義、見える化能力など）
- □ 顧客交渉戦略･能力不足
- □ 大幅な仕様追加要求
- □ 大幅な開発費削減要求
- □ 大幅な納期短縮要求
- □ 適時の支援依頼発信能力不足
- □ 変更管理未実施による開発費の増加
- □ 顧客に対する不都合な事実の隠蔽（故意による事故）
- □ 要件定義体制不備（業務経験・知識不足の要員で要件定義を実施）
- □ 要件定義遅延のため不十分なテストで客先リリース
- □ 顧客要求の優先順位コントロール失敗による開発失敗
- □ タイムリミット管理能力不足

◎要因：メンバーの技術能力、ヒューマンエラー（うっかりミス）
- □ 新技術知識不足
- □ 顧客業務知識不足

◎要因:プロセス管理の有無
- □ プロセス管理の不足
- □ 仕様凍結未確認のまま設計着手
- □ 基本設計の承認なしで詳細設計着手

◎要因:コミュニケーション能力(阻害・ギャップ)
- □ 顧客のパートナー化失敗
- □ 顧客とのコミュニケーション能力
- □ プロマネと上司間のコミュニケーション
- □ 営業と開発部門間のコミュニケーション能力
- □ 営業部との交渉力

◎要因:関連部署との連携不足
- □ 組織間の協調性・コミュニケーション

**(2) モノに関するリスク要因**

◎要因:ドキュメントの不備(要件定義書・設計書・チェックリスト・手順書など)
- □ 要求仕様書なし

**(3) 情報に関するリスク要因**
- □ あいまいな開発範囲(スコープ)
- □ あいまいな要件
- □ あいまいな仕様
- □ 情報の不備・不足(マネジメント情報、技術情報、過去の失敗情報)

## 5. 事前準備工程に関するリスク検出チェックリスト

要件定義工程は重要な工程であったが、さらに重要な工程として事前準備工程がある。ほとんどの開発関係者においては、開発の最初の工程は要件定義工程だとの認識が一般的なのだろうと思う。開発のスケジュール表に事前準備工程などの線引きなど見たこともないし、第一そんな線引きをしたとたん上司からきつい叱責をくらうのがおちなことは当たり前のことだ。

だけど事前準備は必要なのだ。事前準備とは、日本の伝統的なモノづくりで言われるところの「しこみ」にあたるものだ。その時になって、あれもないこれもないなどとアタフタしないためにも絶対的に必要な下準備なのだ。

その事前準備工程はどこで実行されるのかと言えば、前の開発その前の開発すべての前の開発の遂行中に実行される改善活動やプロジェクトの振り返りで実行されることが、将来のプロジェクトのための事前準備となる。 余談だが井上陽水の『瞬き』という曲にある「未来のあなたに幸せを贈る 記憶と思い出を 花束に添えて」というのがその意味をよく表しているように感じる。

以下に事前準備工程において日々排除されるべきリスクを挙げてみる。これらのリスクのうち排除できなかったものが次なる開発で間違いなく問題を引き起こすことになるだろう。

### （1） ヒトに関するリスク要因

◎要因：他者依存的姿勢（自律性の放棄）
- □ リーダーシップの欠如
- □ マルチベンダー下における分担責任のあいまいさ
- □ 他社パッケージの未検証採用
- □ 基幹技術の選定（なりゆきまかせ）
- □ 開発組織の自律性不足

◎要因：上位マネジメントの関与不足
- □ 顧客交渉戦略･能力不足
- □ 不適切なプロマネの選任

◎要因：ユーザーの参加・協力度不足
- □ 顧客の参加・協力度が低い

◎要因：組織能力不足（未熟な組織文化、戦略の欠如）
- □ 不適切な事前着手（短納期）
- □ 新旧システムの同時並行開発（開発量・時期の重複）
- □ 顧客との名目だけの共同研究開発
- □ 能力不足のプロマネの選任
- □ 頻繁な更新によるシステムの劣化・スパゲッティ化

◎要因：見積り能力
- □ 安易な現行機能搭載の保証
- □ フィット＆ギャップ調査能力

◎要因：要件定義能力
- □ 要件定義能力不足

◎要因：リーダーのプロジェクトマネジメント能力（外部交渉、タイムマネジメント、現場主義、見える化能力など）
- □ 顧客交渉戦略･能力不足
- □ 不適切な事前着手
- □ 新旧システムの同時並行開発
- □ 企画能力不足
- □ 適時の支援依頼発信能力不足
- □ 問題の放置
- □ 未経験言語採用の準備不足
- □ 開発目的の誤り（受注優先）
- □ 技術者のトレーニング不足
- □ 開発体制の不備（経験者・能力・知識・人数）
- □ 未経験分野／業界参入の準備不足
- □ 新技術採用の準備不足
- □ パッケージベース開発におけるフィット＆ギャップ調査不足
- □ タイムリミット管理能力不足
- □ 技術方式選定対応における柔軟性

◎要因：メンバーの技術能力、ヒューマンエラー（うっかりミス）
- □ オープンシステム知識の不足
- □ 開発言語能力不足
- □ 新技術知識不足
- □ 顧客業務知識不足

◎要因：プロセス管理の有無
- □ プロセス管理の不足

◎要因：コミュニケーション能力（阻害・ギャップ）
- □ 顧客のパートナー化失敗
- □ 顧客とのコミュニケーション能力

◎要因：関連部署との連携不足
- □ 組織間の協調性・コミュニケーション

**（2） モノに関するリスク要因**
- □ 開発ガイドラインの不備（開発プロセス、設計手順書、コーディング規約、単体・総合評価テスト手順書等）
- □ 開発のベースの有無
- □ 開発環境の不備

**（3） 資金に関するリスク要因**
- □ 赤字受注
- □ 開発費不足（見積りの失敗、開発の失敗）
- □ 資源投入戦略の誤り（開発費投入時期ミス）

**（4） 情報に関するリスク要因**
- □ あいまいな開発範囲（スコープ）
- □ あいまいな要求仕様
- □ 情報の不備・不足
- □ 未経験分野／業界の業務知識・技術情報不足
- □ 新技術の技術情報不足

## 第7章　プロジェクトを成功に導く黄金律

銀の弾丸って知っているかい。そうだあの狼男を倒せるただ一つの武器といわれるやつだ。しかしソフトウェア工学においてはフレデリック・ブルックスがソフトウェアのすべての問題を解決する万能な解決策、つまり銀の弾丸などないっていっている。

でもそんなことはない。別に生身の人間に空を飛べって言っているわけじゃない。ノーベル賞級の発明をしろといっているわけじゃない。ただソフトウェアのアプリケーション開発を正しくやろうっていっているだけなのだ。銀の弾丸はないのかもしれないけど成功に導く黄金律はあるに違いないのだ。

本章においては、プロジェクトを成功に導く基本的な法則をアジャイルの視点も加えながら探っていくことにする。

### 1．　プロジェクト成功の黄金律

まず初めにソフトウェア開発を成功に導く黄金律について説明をしよう。

製品開発であれ商品開発であれ我々は不確実性に満ちた予測不可能なビジネス環境下におかれていることは間違いない。 予測不可能な状況への最適なアプローチの仕方は”変化即応”なのだ。環境のすばやい変化を常にキャッチしすばやく対応し続けるということだ。

予測不可能なビジネス環境下におけるソフトウェア開発方式の代表例の一つがアジャイルスクラムだ。また同様に予測不可能なビジネス環境下におけるビジネスアプローチの代表例の一つはセブンイレブンにおける仮説と検証というアプローチだ。

ここでは詳しい分析内容については触れないが優れた組織のビジネスアプローチあるいはソフトウェア開発アプローチは、その考え方・行動様式において驚くほどの共通性・類似性をもっているということであり、これらのアプローチは先が見通せない不確実性の環境の中にあっても顧客満足度と開発組織の健全性の最大化を実現し、先を見通した結果とほぼ同じ結果を生み出すことのできる驚異的な方式であることを示唆している。

セブンイレブンのビジネスコンセプトおよびその行動指針の分析とアジャイルスクラムの示す価値と原則から導かれるプロジェクト成功の黄金律は次のようだ。

**【プロジェクト成功への黄金律】**

- **顧客の要求に対して即応すること**
- **プロジェクト全員で同じゴールを目指すこと**
- **価値ある商品の提供**

### （1） 黄金律①　顧客の要求に対して即応すること

常に変化し続ける顧客要求に即応するための行動指針は次のようだ。

**□　行動指針1　変化する顧客の要求内容を常時把握し続けること。**

変化する顧客の要求を常に把握し続けるためには、顧客との強調および変化への対応が必要だ。 つまり開発者は顧客とプロジェクトにおける密なコミュニケーションを通して日々変化す

る顧客の要求を把握することだ。顧客要求の変化に沿った開発を実行することは、変化を味方につけることによって、お客様の競争力を引き上げることになる。

**□ 行動指針2 正しく機能するソフトウェアの短いサイクルでの提供（仮説と検証の実行）**

ソフトウェア開発においてお客様の本当の要求をすばやく確認する方法は、動作するソフトウェアをできるだけ短いサイクルでお客様に提供し検証していただくことに尽きる。お客様の確認前のソフトウェアはいわば「仮説」の状態でありお客様の「検証」を受けて初めて顧客要求が満足されたかどうかが分かる。短納期で顧客の要求する優先度の順に提供されたソフトウェアは顧客の満足度および信頼度を最高レベルで満足させるだろう。

### （2） 黄金律② プロジェクト全員で同じゴールを目指すこと

プロジェクトの成功は、価値観、意識および目標について全員でそれを共有し、同じゴールを目指すところから達成される。チームプレーは連携・連帯の中でしか成果をだすことはできないのだ。

**□ 行動指針3. プロジェクト関係者間の信頼関係の構築**

信頼関係の重要性については本章の「2. コミュニケーションって何だ」等で詳細について触れるが、プロジェクト活動を成功させる基本はプロジェクトメンバー同士の信頼関係ないしはプロジェクトと顧客間の信頼関係にある。

信頼関係はどうやって構築されていくのだろうか。 信頼関係は相互のコミュニケーションの中でお互いの約束を守ることによってしか作り上げられない。何回も何回もお互い同士の約束を守っていくことで信頼関係は強いきずなとなっていくのだ。

古い話だけど、孔子は次のように言っている。

「人にして信なくば、その可なるを知らざるなり。[為政第二]」

つまり、人として信義がなければうまくやっていけるはずがない。車に車軸がなくてどうして車を動かせるのかと言っているのだ。またこうも言っているのだ。

「信なくば立たず。[顔淵第十二]」

弟子の子貢が政治の中で大切な三つのもの、つまり食料、軍備、民の信頼があるが、やむをえず捨てるなら何かとの問いに対して、孔子はこう答えた。まず軍備を捨てる。次には食料を捨てる。軍備を捨て食料を捨てれば人は死ぬが、昔から誰にでも死はある。しかし民の信頼がなければ社会は成立しないと言っているのだ。

オー！なんとすごいことを言っているのだ。 しかも二千年も前に。信頼関係がなくなったら人間が人間でいられなくなると言っている。情けないけど自分ならまず軍備を捨て次に信頼関係を捨て食料は最後まで捨てないで野獣化して結局死んでしまうかも知れない。

**□ 行動指針4. 直接的なコミュニケーションの実施**

コミュニケーション、特に顔と顔を合わせた直接的なコミュニケーションは重要なのだ。ITネット全盛の現代においてそんなの古いとか非効率的だとか言う声も聞こえてきそうだが、これは古いとか非効率的だとかいう問題じゃないのだ。本音・本物で会話が成立するのは直接に顔を合わせたコミュニケーションだけなのだ。本当のホントのデータは普通では決してネットには流れないし、心を許していないネットで自分の本音を言うことは非常に危険なことはみんなよく知っていることだろう。

だからこの時代であってもセブンイレブンは2週間に一度の割合で千数百人もの店舗サポート

メンバーを本部に集めて、いわゆるダイレクト・コミュニケーションを実施しているし、アジャイル開発においては顧客と開発者間のないしは開発者同士のフェイス・トゥ・フェイスのコミュニケーションを重要視しているのだ。
直接的なコミュニケーションは本音・本物の情報交換を可能としお互いの信頼関係を築くためにはどうしても必要なものなのだ。

**□ 行動指針5. 全員による意識・情報の共有化**

大勢の人間で構成された組織で同じ目標を達成するためには全員の目指す方向を一つにまとめなければうまくいかない。アジャイル開発のスクラム方式では正に次のように言っている。

> 「スクラムを組んでフィールドを駆け抜けよう。 前後にボールをパスしながらチームが一丸となって前進しようとする。 最初から最後までメンバーがともに動く。 フェーズは相当に重なり合い、開発プロセス中に生じる振動やノイズを吸収できる。 プロジェクトチームの中心メンバーは初めから終りまで一緒に走り、すべてのフェーズを結合する責任をもつ。」[Takeuchi_Nonaka_1986]

これはチームプレーの基本原理であり、日本におけるモノづくりの原点を端的に示したもので、これ以上にプロジェクトのあるべき姿を表した文章は他では見たことがないと言ってもいい。

### （3） 黄金律③ 価値ある商品の提供

価値ある商品とはもちろん顧客にとって価値のあるということだが、顧客価値の実現方法については「第6章 2.（4）要件定義業務の実行」、「第7章 3.（12） 顧客価値はどこにあるか」などで触れてきた。
顧客に対して現在よりさらに価値のある商品、他にはない価値をもっている商品を提供することができなければこの競争社会では生き残れない。
価値ある商品の提供はモノ真似ではない、独自性の追求の中で実現できるのだ。

**□ 行動指針6. モノ真似でない、独自性の追求を行う**

顧客との直接的なコミュニケーションの中から発見された顧客価値の高い要望を実現するためには意欲に満ちた開発者を集めてプロジェクトを構成し、メンバーへの信頼をもとに必要な環境と支援を与えることが必要なのだ。
自己組織的なチームは現状に満足しないブレークスルー思考に基づき技術的卓越性を蓄積し続けながら優れた設計に対する不断の研究を怠らないだろう。その結果自己組織的なチームにおける環境対応力の機敏さが高まり、現状を越える独自性をもった最良のアーキテクチャ・要求・設計が生み出されるのだ。

以上、プロジェクト成功の黄金律を示したが、この黄金律に到達するのに越えなければいけない四つの壁がある。以下にこの四つの壁である、コミュニケーションの壁、マネジメントの壁、タイムマネジメントの壁、プロジェクトの壁について示すことにする。

## 2. コミュニケーション問題の解決

チームの中のコミュニケーションが崩れるとどんなことがおきるのか挙げてみよう。

### （1） コミュニケーションの阻害がひきおこす病気

プロジェクトチーム内でのコミュニケーションの断絶はチーム内でさまざまな病気を引き起こしているのだ。

#### ① 空気読みの蔓延

みんなが言っているいわゆるKYだ。 なぜKYが病気なのか分かるかい？ KYって結局表向きはその場の雰囲気を壊さないようにみんなに配慮した発言なり行動をすると言われているよね。 本当にそうだろうか？ 本当は単に”出る杭は打たれる”のが恐ろしくて、または仲間はずれにされるのが怖くて空気を読んでいるだけなのじゃないのかな。 結局自分の利益第一主義で他人の気持ちを読んでいるだけで、他人の気持ちを思いやるのとは正反対なんじゃないのかな。 KYが病気って言うのは次に書いたようなことがチームの中で起きてしまうからなのだ。

#### ② 指示待ち人間の増加

言われたことしかやらない、自分で考えない、言われたことも半端にしかできない、挑戦しない、責任回避する、保険をかける、一歩後退、自己中心的、人の意見を聞かない、他人に興味をもたない、などおよそ人間の醜い面ばかりが露呈されてしまいプロジェクトの崩壊を促進するのだ。

#### ③ チーム内の信頼関係の崩壊

お互いの意思疎通が断絶すれば人と人の間の信頼関係は生まれないだろう。 ”一見のお客はお断り”というのは信頼関係のない人はお客ですらないと言うことなのだ。信用・信頼関係がない個人あるいは組織は存立できない、つまりプロジェクトとして活動できないということなのだ。

#### ④ チームがチームでなくなり、単なる烏合の衆となってしまう

みんなが各自勝手なことばかりやるようになると組織的な活動はできなくなりプロジェクトは進まなくなってしまうのだ。

#### ⑤ その結果、当り前のことが当り前に実行できなくなる

プロジェクトとして一致団結していたときにできていた当り前のことが当り前に実行できなくなり、組織の力は低下し、ばらばらになった隙間から品質とお金がどんどんこぼれ落ちてしまうのだ。

まとめて言うとコミュニケーションの阻害はメンバーの業務遂行の品質を低下させ、そのことが製品品質の低下を招き、それは更にプロジェクト組織を崩壊させ、最終的には顧客の信用を失墜させてしまうことになるのだ。 実に恐ろしい病気なのだ。

**ティーブレイク**

**指示待ち　脱却なるか**

６日の中国戦を終えて、日本代表の遠藤がこんなことを口にしている。

「言われたことを言われた通りにやるという日本人の悪い癖が出た」。４日前のベネズエラ戦の反省で両翼から仕掛けることが強調されていた。せっせと左右にボールを運んではクロスを中央へ、という整然とした攻撃の繰り返し。意外性に乏しい前半を指した言葉だ。

・・・・

冒険をしたがらない現代の気質はサッカー選手も同じだ。「日本人は自分の責任でリスクを負わず、保障を求めたがる。取られてもいいからといわれないと1対1で勝負しない。でも本当は取られてはいけない。ぎりぎりのところでチャレンジするのがスポーツの面白さなんだ」。岡田監督はこんな風に選手に呼び掛けている。

年明けの鹿児島合宿から、チームづくりは一歩深められた。指示を待つのではなく、選手自らが判断して試合を作っていくことをこれまで以上に促している。「その方がいろいろな事態に対処できる。チームとして成熟していく段階と期待している」。

潮　智史　朝日新聞編集委員　［朝日 2010.2.9］

**(2)コミュニケーションからしか信頼関係は生まれないのだ**

誰も積極的には話しをしない。お互いの関係もギクシャクとして仕事も順調じゃない。そんな状況を見かけたりしないか。 相手が信頼できないからとか嫌いだとかの理由で話をしない状態を続けていては、いつまでたっても信頼関係は構築できないのだ。 話をしなくても仕事はできるとの考え方はおそらく間違っているね。 良いコミュニケーションは良い仕事の基盤なのだ。

また信頼関係は誠実なコミュニケーションを続けていく中でしか結ばれないだろう。 信頼関係がないからコミュニケーションができないという考え方は誤っているのだ。 他人が一歩前に出てくるのを待つのじゃなく、まずは好き嫌いの感情を横においておき、自ら一歩前に踏み出す勇気を持とうじゃないか。

**(3)コミュニケーション改善のポイント**

コミュニケーションの改善に有効な手立ては次のようなものだ。

□　オープンに友好的に話せる場を確保しよう、そして自由な議論が行なえるようにしよう。
□　好奇心をもって対話しよう。 聞くだけ無駄と思わずに人の話に興味を持とう。
□　自分の感情を少し抑えて発言者への気配りをしよう。
□　対話に集中しよう。 出席した時間がもったいないなどと考えず、この場の議論に集中しよう。

すべてのコミュニケーションにいえることだが、「私」という言葉を捨て、「あなた」 に置き換えてみよう。 相手の視線に立った見方をすることが、自己中心的で感情的なふるまいを抑えるのに役に立つ。その結果として、相手の行動を促し、成果を挙げることにつながる。［プレジデント 2007］

### （4） コミュニケーションの復活

コミュニケーションの復活は直接的会話の復活しかないのだ。第２章の４．緊急対応行動のポイントや第３章の２．基本行動指針および３．対策アクションも全て直接的コミュニケーション復活のための行動なのだ。 もう一度直接的コミュニケーションってどういうものだったのかを示そう。

【直接的コミュニケーションのために】
１．メンバーは同じ場所に集結しよう。
２．毎日直接話をしよう。
３．関係各会社・組織との密なコミュニケーションを実施しよう。
４．プロジェクト内の日次会議をしよう。
５．顧客との日々のコミュニケーションをしよう。

### （5） 良いコミュニケーションとアジャイル開発

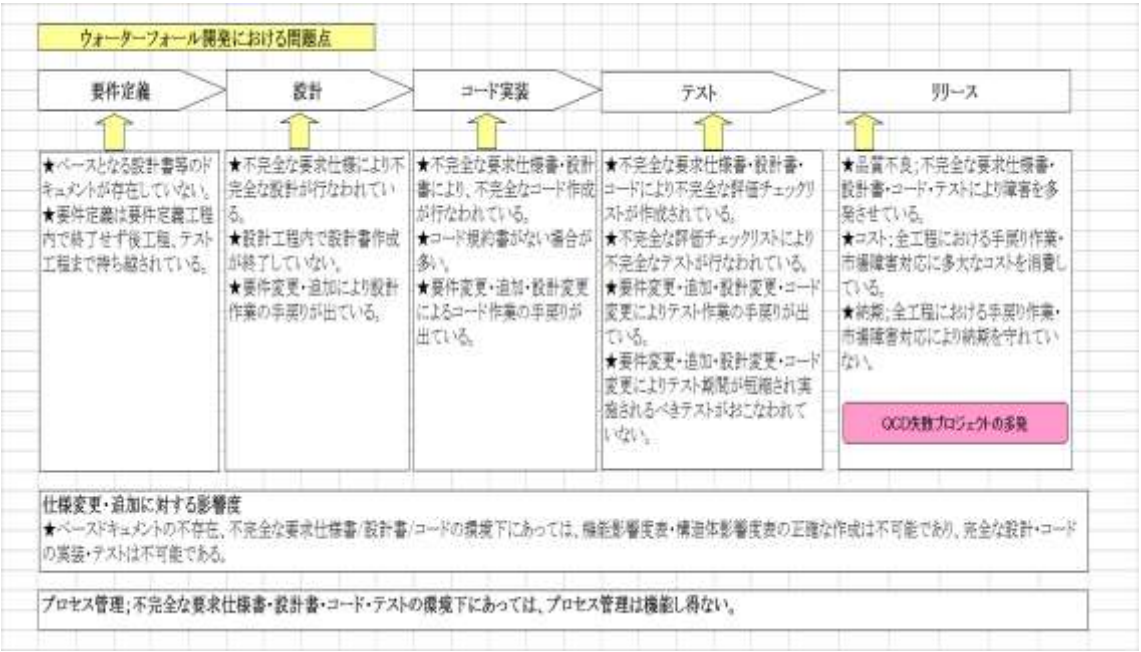
注目されているアジャイル開発手法でもコミュニケーションの重要性について次のように指摘しているのだ。

□ アジャイルの価値1. プロセスやツールの価値も認めるが、**個人とその相互作用を**。
□ アジャイルの価値3. 契約交渉の価値も認めるが、**顧客との協調**を。
□ アジャイルの原則4. ビジネス側の人々と開発者たちはプロジェクトを通じて**毎日共に作業**しなければならない。
□ アジャイルの原則6. 開発チームに対してあるいは開発チーム内において最も効率的かつ効果的な情報の伝達方法は**フェイス・トゥ・フェイスの会話**である。

上記価値および原則に示されたように、アジャイル開発の基本的な価値および原則は顧客と開発チームの協業および直接会話におかれているのだ。

### （6） コミュニケーション・ギャップ問題について

ウォーターフォール開発における問題点を下図に示した。




**[図表8．工程（プロセス）間のギャップ]**

コスト削減圧力にたまりかねて中国を筆頭にオフショア開発は拡大する一方で、一つのプロジェクトの各工程を多数の会社にて分割担当するのが当り前のようになってきた。 例えば要件定義は元請けのA社、設計は外注のB社、コード製造は中国のC社、テストは外注のD社のようにワンプロジェクト・マルチベンダーによる開発が激増してきている。

このような環境の変化の中で顕著に現れている不都合なことはみんなも知っていると思うが”品質の低下”なのだ。中国にやっとプログラム製造を出したけどできあがったプログラムはまともに動作しないってことしばしば聞かないか？ ここで言いたいことは中国の品質が悪いってことじゃない。

ソフトウェア開発は複数の工程で構成されているよね。またこれらの各工程間の入出力は厳密に定義され、その約束に従って自分の工程の正しい成果物を次の工程に渡す約束になっているよね。そうしなければこのプロジェクトは成功裏のうちに完結できないよね。

この開発のやりかたは我々の多くが長年慣れ親しんだ開発スタイルであるウォーターフォール開発とよばれているものだ。ウォーターフォール開発の最大の弱点は各工程間で約束どおりの成果物のアウトプット／インプットが難しいところにあるといっていいだろう。 問題は開発の分業化された各工程の間で発生しているのだ。

ウォーターフォールの各開発工程の間において次工程との情報伝達がうまくいっていない場合、次工程に対して不十分なあるいは誤った成果物を渡してしまうことになる。またタイムリミットにせかされて、不完全なものであると知りつつ、その成果物らしきものを次工程に渡していることがある。最初の工程における不具合は次の工程に行くほどその被害内容は比例級数的に拡大するのだ。

工程間の問題は、要件定義／設計／コード実装／テスト／リリース工程間にて重大な種々の問題を発生させ続けている。各工程業務はその専門担当組織に分業・分割されている場合が多く、特にこの場合のコミュニケーション・ギャップは組織間の壁の存在により非常に深刻な状

態となる場合が多いのだ。

１社で全工程の開発を行なった場合でも各工程は社内の異なった部署で担当されることも多くこれらの組織間のコミュニケーションを保つことはプロジェクトマネジャーにとって非常に困難な業務の一つであったが、各工程を別々の会社が担当した上に、海外の会社が混じってしまった場合のプロジェクトマネジメントの困難さは想像を絶するものがあると言っても言いすぎじゃないだろう。 実際のところみんな誰も”コスト削減”の錦の御旗を恐れてか、公式の場ではこの問題を口に出しては言わない。キミの会社ではどうだろうか？

特に「ワンプロジェクト・マルチベンダー＋中国」のプロジェクトにおける工程分業の姿は、あるべき“共有分業”ではなく最悪の“分離分業”の形に陥りやすく、ウォーターフォールの弱点をさらに拡大し、各工程間でのコミュニケーションの断絶を致命的なレベルにまで拡大させ、極度の品質悪化を招いている場合が多いのだ。 キミはどう思っているのかな。

### （7） コミュニケーション・ギャップ問題の具体例

次にコミュニケーション・ギャップ問題の具体的例を列挙してみよう。

- 要件定義は要件定義工程内で終了せず後工程、テスト工程まで持ち越されている。
- 不完全な要求仕様により不完全な設計が行なわれている。
- 要件変更・追加により設計作業の手戻りが出ている。
- 設計工程内で設計書作成が終了していない。
- 不完全な設計により、正しい機能影響度表およびソフト構造体影響度表が作成できない。
- 要件変更・要件追加・設計変更によるコード作業の手戻りが出ている。
- 不完全な要求仕様書・設計書により、不完全なコード作成が行なわれている。
- コード規約書がない場合が多い。
- 不完全な要求仕様書・設計書・コードにより不完全な評価チェックリストが作成されている。
- 要件変更・要件追加・設計変更・コード変更によりテスト作業の手戻りが出ている。
- 不完全な評価チェックリストにより不完全なテストが行なわれている。
- 要件変更・追加・設計変更・コード変更によりテスト期間が短縮され実施されるべきテストがおこなわれていない。
- プロセス管理表が正しく機能できず、形式的にならざるを得ない。
- 必要なドキュメントが存在していない。

どうだろうか、キミのところではこんな問題が出ていないだろうか。

これらの問題は、全工程における手戻り作業を発生させ、さらに多くの不具合を作り込み、市場において重大な障害を発生させ続けており、顧客に対しては顧客価値の毀損およびその信用を傷つけ、開発側においては多大なコストの浪費および人的疲弊を招いている。

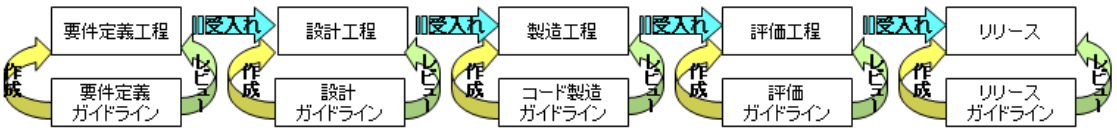
### （8） コミュニケーション・ギャップの解消

工程間のコミュニケーション・ギャップ問題を解消するにはまず各組織間の密なコミュニケーションの実行が必須なのだ。これはすでに第3章2．基本行動指針の②および③で述べたがもう一度示そう。

□ 関係各会社・組織との約束を明確化・明文化することおよびその遵守。
□ 自工程の前後の工程・プロセスにも積極的に口を出すこと。

次にそのコミュニケーションを実現するための各組織間の共通の約束ごとが必要だ。共通の約束ごととは言葉をかえると各工程における成果物のあるべき姿を定義したガイドラインが必要だということなのだ。

例えば、要件定義ガイドライン、設計ガイドライン、コーディングガイドライン、結合テストガイドライン、総合テストガイドライン、リリースガイドライン、市場品質ガイドラインなどだ。これらのガイドラインはそれぞれの工程の成果物のあるべき姿を具体的に示した内容でなければいけない。これらのガイドラインはそれぞれの工程担当組織における業務遂行のガイドライン・手順書であると同時に次工程に渡す前の成果物のレビュー基準としても使用され、さらに次工程における受入れ検査の基準としての役割も果たすのだ。




［図表9．工程別ガイドラインの役割］

**ティーブレイク**

**確実にキャッチされる球を**

キャッチボールで大事なのは投げることではなく、受けてもらうこと。話すことも書くこともそれと一緒で、情報量を増やしても、伝わらなければ意味がない。「いくら豪速球を投げてもダメなんです」。マスメディアにかかわる人間は「伝える」ではなく「伝わる」に意識を置かねばならない。 そうした思いが一層強くなったのだという。

「あ、そうそう」。古いファイルから取り出したのは、落語家の桂小南（2代目）の揮毫だった。 ただ一言、「噺家は喋るな」とある。矛盾しているように見えても、言われてみれば、たしかに言葉数の多い落語家は必ずしもおもしろくない。反対に古今亭志ん生（5代目）の最晩年は、高座にいるだけで観客が満足したという伝説がある。

「要は間を大事にするということでしょう。なんでも日本の芸は間だな、と思いますね」。それから山川さんは声を落として「だから究極は、アナウンサーは、黙っていた方がいいのかもしれません」と言った。

山川　静夫　エッセイスト・元NHKアナウンサー［朝日 2010.2.1］

## 3. プロジェクトをマネジメントする

マネジメントとは部下を管理・監督・監視することだと思ってはいないかい。 そうではなくてマネジメントって困っている問題を何とか解決することだろう。困っている問題を”どうにかする”ことがマネジメントの本当の意味なのだ。

”どうにかする”のは開発管理表でもなければ、コンピュータシステムでもなければ、資金でもなく人間がどうにかするのだ。キミたちがどうにかするしかないのだ。 人間が問題（仕事）をどうにかするにあたって重要なことは仕事に対する認識とやる気の2つだ。

### （1） 認識とやる気について

**◎認識について**

いろんな人がいろんな所で部下のやる気を起こさせようと必死になっているが、やる気は起こさせようとしても起こさせられるものではないのだ。ニンジンをぶらさげて動くのはウマぐらいなものだろう。

まずは認識だろう。目の前に問題があったとしても本人がそれを問題と思わなければ問題は解決されないままでいつか爆発するだろう。ある事象が問題であると認識されてはじめてそれはその人にとっての問題となるのだ。「心ここに無くば見えても見えず」なのだ。

そんなことは分かっているという声が聞こえてきそうだけど本当に分かっているかい？気が付いたら尻に火がついていたようなことがなかっただろうか。

マネジメントの失敗のほとんどは問題認識の失敗とその誤認識に基づく不適切な行動に原因があるのだ。

**◎やる気について**

よくモチベーションだとかマインドだとかいわれるけど他人に言われただけで本気ややる気が起きる訳もないだろう。少しのインセンティブがもらえるとしてもそんなことで本気を出すのはウブな青少年くらいなものだろう。

問題が自分自身の問題だと認識されない限り人は本気もやる気も出さないものだ。その問題をどうにかしないと自分が非常に困ったことになると分かって初めて人は動くのだ。

人は他人の痛みは百年でも我慢できるが自分の痛みは一秒たりとも我慢できないものだ。だからといって人を動かすために不安や恐怖をあおりたてた恐怖政治をすすめているわけではない。他人から迫られ強制されたものでは人は決して本気にもやる気にもなれないことを知っていたほうがいい。

これ以降はマネジメントの原点となる人の「認識」に焦点を当てて話をすすめたいと思う。

### （2） 認識のポイント

我々のソフトウェア開発で具体的にはどんな不都合な認識や行動が行われているのか。 みんなもすでに気づいているものも多くあると思うが以降、我々が陥りがちな誤認識の実例を見ながら、自分も救い仲間も救うような建設的な認識はどんなものかということについて一緒に考えてみよう。

認識の転換は、自分自身との闘いなのだ。 建設的な認識が手に入ると仲間同士だけではな

く多くの人々との活発なコミュニケーションが復活・促進され人間力および組織力回復の源となるのだ。
　認識改善の目的は個人およびチームが主体性をもって生きいきと活動できる自律性の回復なのだ。
「誰かが何かをやってくれないかな」から「自分に何ができるだろうか」への思いの転換は、希望に満ちたソフトウェア開発への出発点となるに違いないのだ。
　認識のポイントは、「人に関すること」、「マネジメントに関すること」、「品質に関すること」、「コストに関すること」、「納期・時間に関すること」、「プロセスに関すること」など多岐にわたっている。

### (3) プロジェクトの成功は人による ［人に関する認識］

　まず人について取り上げてみよう。
　プロジェクトの成否はまず必要な開発費およびスケジュールにかかっているとみんな思っているだろう。 だけど本当にそうなのかな。
　次に示したのはプロジェクトの成功要因に関する米国での調査結果だ。[CHAOS]

　1位：幹部のサポート　18%
　2位：ユーザの貢献度　16%
　3位：経験のあるPM(プロマネ)　14%
　4位：明確な目的　12%
　5位：単純なスコープ　10%
　6位：標準ソフト・インフラ　8%
　7位：要件のシンプルさ　6%
　8位：プロジェクト・プロセスの有無　6%、
　9位：見積りの信頼性　5%
　10位　その他　5%

　成功要因の1位から3位まで、人間に関する項目であり合計48%も占めているのだ。1位および2位を引き出すのは3位のプロジェクトマネジャーの能力次第だから結局プロジェクトの成否の半分はプロジェクトマネジャーであるその人にかかっているということなのだ。

【アジャイルの視点】
　アジャイルの4つの価値の内の3つ、および12の原則の内の8つが直接的な人間の行動に関するものだ。プロジェクトの成功は人間のどのような行動にかかっているか、下記の人間行動に関するアジャイルの価値および原則を十分かみしめて見る必要があるだろう。
［添付資料1．アジャイルソフトウェア開発宣言］

□価値1．プロセスやツールの価値も認めるが、個人とその相互作用を。
□原則5．意欲のある人々でプロジェクトを構築する。彼らが必要とする環境とサポートを与え、そして仕事が完了するまで彼らを信頼する。
□原則6．開発チームに対してあるいは開発チーム内において最も効率的かつ効果的な情報の伝達方法はフェイス・トゥ・フェイスの会話である。
□原則8．アジャイルプロセスは持続的な開発を促進する。スポンサー、開発者、そしてユーザはたえず一定のペースが維持できるようにしなければならない。

□原則9. 技術的な優位性および良好な設計に対する継続的に払われる注意は俊敏さを増強する。
□原則11. 最良のアーキテクチャ・要求・設計は自己組織化チームから生まれる。
□原則12. アジャイルチームは定期的に、より効果的な方法につき振り返りを行い、それに従ってその行動を変更し調節していく。
□価値3. 契約交渉の価値も認めるが顧客との協調を。
原則4. ビジネス側の人々と開発者たちはプロジェクトを通じて毎日共に作業しなければならない。
□価値4. 計画に従うことの価値も認めるが、変化への対応を。
原則2. 要求変更を歓迎する。たとえそれが開発の後期であったとしても。アジャイルは顧客の競争力を高めるために日々の変化に対応する」

**（4） 障害の真因は人間に起因する。モノには」起因しない** ［人に関する認識］

ソフトウェアに関して、ほとんど全ての障害報告書における障害原因の記述において、“人”が主語として書かれることはまずないだろう。例えば一般的な障害報告書には次のように記述されているだろう。

障害原因 ： ○○業務の△△機能のロジックにおいて、Aデータを参照すべきところBデータを参照していたため誤った計算結果となってしまった。
対策 ： ロジックを正しく修正した。開発レビューの強化および評価チェックを強化し再発防止を行う。

この障害報告書には全く主語がないのである。これではまるで悪かったのは“ロジック”であると言っているようなものであろう。本当は悪いロジックを作成した○○技術者のはずであろう。世の中で事故と呼ばれるものの報告書おいては全て、その事故を引き起こした原因者の名前が明記されており、その行為が何故行われたのかについて検証が行われている。原因者が分からなければ、その原因も分からないだろう。

ソフトウェア障害において、特定の個人名まで出す必要はないが、仕様作成者、設計者、プログラマ等の役割名程度は明記すべきであろう。主語が明記されないと“誰が”があいまいになり、何故そのようなミスをしたのかも掘り下げることは不可能になるだろう。事実、このような障害報告書しか書けない部署においてはほとんど真因の追究も行われないため今後の再発防止もできず、防止策の情報共有もできないであろう。障害の真因とはこのような結果を招いた人・組織自身の考え方および行動のありかたの欠陥にある。このような視点に立って真因の特定およびそれに対する再発防止策を今後のハドメとして提示する必要がある。

責任の所在を不明瞭にするところから“無責任”が発生するのであり、成果を享受する個人および組織は、それに対する責任を明確にしなければならない。

障害の真因の明確化方法については、科学技術振興機構（統括　畑村　洋太郎）による「失敗知識データベース」［http://shippai.jst.go.jp/fkd/Search］ を参照されることをおすすめする。

### （5） 自律は自分の頭で考えるところから始まる ［人に関する認識］

障害発生時，予算超過時，スケジュール遅延時などいろんな問題が発生している時に何んらの解決案の提案もなく、上司にただ「こうなっちゃいました。どうしましょうか」と報告している人を見たことがないかい。

報告者は起こっていることだけを伝えれば責任は果たせると思っているようだし、上司はすぐに指示を与えることだけが義務であるかのように思っているようだ。 このような関係は上司依存の甘えとしか思えないのだがどうだろうか。 こんなやり方ばかり続けていたら部下は報告さえすれば責任は取らなくても済むものと思い自分で何とかするという姿勢をなくしてしまうのだ。これでは部下の自己判断による自律性を妨げてしまうだろう。

自分の意見・意思のない報告者に対しては、「だから何？」の繰り返しの問いかけをしよう。

このような実践のなかから自律的な人材、自律的なチームが生まれてくるのだ。

【アジャイルの視点】

アジャイルの原則１１にも「最良のアーキテクチャ・要求・設計は自己組織的なチームから生まれる」とあるように、自己組織的な個人やチーム(注)は、言われたことしかやらないような硬直的な個人や組織とはちがって、チーム内での話し合いや自律的な判断に基づいて各人が責任を自覚しながらなすべきことを行なうのだ。 このような自律的に行動する個人がチームで協調作業を行えば、問題発生時等の対応も含め顧客の要求の変化への対応もすばやくなることで環境変化への適応性が強くなり更に独自性の追求が可能となるのだ。

注(注). 自己組織的なチームとは、命令や指示に基づく縦割りの組織ではなく、チーム内での話し合いや自律的な判断に基づいて各人がなすべきことを行なう組織のことだ。このような自律的に行動する個人がチームで協調作業を行なうことで、変化への適応性が強くなり独自性の追求が可能となるのだ。

**考える人、考えない人**

残念なことに、世の中には考えることをやめてしまった人たちも大勢います。運動をやめた人と同じで、身体を1ミリも動かさないのではなく、「鍛えない」という意味です。人は身体を鍛えずとも生きていけますが、鍛えればたくましい身体になります。

考えるのをやめた人たちは、いつもと同じように人生を過ごし、現在の快適な生活を破壊するリスクがあるものすべてに懐疑的です。ちょうど、運転中に居眠りしてしまった人が、まっすぐな道を行く間は無事であるように、考えるのをやめた人たちもいつも通り物事が過ぎている間は順調な生活を送っていられます。

問題は、曲がり角に差しかかったときでしょう。突然の激しい衝撃に耐えられないのです。

あなたのまわりの考えるのをやめた人たちは誰でしょう。眠りの森から彼らを呼び戻してください。

［スウェーデン式アイデア・ブック　p57］

### （6） 改善活動は仕事そのものである ［マネジメントに関する認識］

改善活動は仕事に余裕があればやりたい程度のものと考えてはいないだろうか。仕事で手がいっぱいの時など改善活動なんてやっていられない、ということをよく耳にする。

ソフトウェアは目には見えにくいものである。ソフトウェアはハードウェアと違って、その傷、汚れ、ひずみなどをすぐには見ることができない。ソフトウェアにおける改善活動とは、自分たちのソフトウェアの傷・汚れ・ひずみを修復する“仕事”なのだ。もし目の前にあるソフトウェアに傷・汚れ・ひずみなどを眼で見ることができたとしたら、あなたはそれを商品として、そのままお客様に渡せるのだろうか。

みんながいう”仕事で手がいっぱい”の仕事って何のことだろうか？　相変わらずのやり方の仕事のことだろうか。 相変わらずの仕事のやり方で相変わらずのトラブルを引き起こす準備に忙しいってことなのか。

**“改善活動”は仕事そのものであり、仕事の中で実行されなければならないものだ。**

【アジャイルの視点】

言われたことしかやらない自己組織的でない人やチームにおいては、改善活動はほとんど実行されないだろう。アジャイルにおいては、チーム内での話し合いや自律的な判断に基づいて各人がなすべきことを協調して行なう（価値1．個人とその相互作用）ことで自律的に行動する個人やチームが育成され（原則11．自己組織化チーム）、日々の顧客価値の実現のための開発活動の中で改善活動が実施されるだろう。

**いつものやり方　無意識の習慣に気づく**

人間は好奇心旺盛ですが、同時に恐ろしいほど習慣に支配されています。

好奇心が強くなければ、いまだに洞穴生活を続けていたことでしょう。ですが、習慣の支配力は強いもので、ある方法で何かを覚えると、無意識のうちにその方法ばかりを使い続けてしまうのです。私たちの脳は、四六時中あらゆることに疑問を持つような仕組みにはなっていません。問題なのは、前と同じように考えてしまいすぎることです。

実際、模倣は本能であると言えます。・・・真似ることは自然であり、覚えたことに逆行するのは不自然です。

情報についても同じことが言えます。100人に同じ情報を与えると、99人はそれを事実として受け入れます。しかし、そのうち1人は「ほほう。でもこれをこうしたらどうだろう」と考えます。自然界と同様、99%は何の変化ももたらさない一方、100に1つは突然変異し、進化につながるのです。

［スウェーデン式アイデア・ブック　p22］

**（7）「何故？」を失った時から衰退が始まっている**

顧客要件に対して「何故？」を問いかけなくなってはいないか。また開発業務の諸々のことに対して「何故？」を問いかけなくなってはいないか。

開発とは世の中にないものを創造する仕事ではないのか。言われた通りのものを作ったり、定型的なものを処理する仕事は基本的に単純労務作業あるいはルーチンワークと呼ばれる作業だろう。

「何故？」を問いかけなければ、本当の顧客が望んでいるものや、顧客価値のあるものは発見できないだろう。 開発業務においても、常に「何故？」を問い続けなければ「本当のこと」にたどりつくことはできないのだ。

「何故？」は問いかける方も、問われる方も知的労力を必要とし疲れるものだ。 人間は、意識をしていないと徐々に疲れる作業を回避し始めるものだから、ついには創造的仕事を放棄して楽なルーチンワークに専念するようになる。そして、そのルーチンワークが開発業務であるかのように思い込むようになるのだ。

「何故？」の問いかけは「新しい認識」を生み出し、環境適応力のある柔軟性のある新しい自分を創りだす。「何故？」の問いかけが少なくなったら、自分の想像力・開発力が衰退し始めたものと思った方がいいだろう。

【アジャイルの視点】

「何故？」の問いかけはコミュニケーションの質を深める。その結果、個人とその相互作用（価値1）を活発にし、チームの自己組織化（原則 11）を促し、顧客との協調（価値3）を深め、変化への俊敏な対応（価値4）を可能にするのだ。

**（8） 上司は部下の仕事をやってはいけない** ［コストに関する認識］

上司が部下の仕事をしている状況は、危機に瀕しているプロジェクトで必ず見られる光景の一つだ。担当レベルの仕事が遅延し始めるとリーダーが遅れている担当の仕事を代わりに実行し始め、この状況がプロジェクト全体に及ぶような状況ではプロジェクト内における役割分担が崩れリーダー不在、マネジャー不在のプロジェクトに陥ってしまうことが多々ある。

組織活動において、その役割の者がその役割を果たさないような状況になると、その組織は全く機能しなくなる。これがプロジェクトの崩壊の姿だ。

上司が部下の仕事をしてもいい場合は、スポット的あるいは教育的意味合いにおいてのみであろう。

また平常時において上司が部下の仕事をしている場合もあるだろう。一つは、自分の役割が能力的に遂行できず何か仕事をやっているように見せかけたい場合、あるいは部下の処理速度に不満で部下の仕事を取り上げて自分でやってしまう場合などがあるが、いずれにしろこれらは不適切な行為であるだろう。なぜなら高いコスト単価の上司が安いコストの部下の仕事を実行していることになり、実質的に会社に対してコスト的な損害を与えていることになり、また部下の成長の機会を奪っているからだ。

【アジャイルの視点】

部下における仕事の大幅遅延は、プロジェクトにおいての協調関係が崩れチーム内のコミュニケーションが行われず、結果各メンバーが孤立状態におかれた場合に多く発生する。ウォーターフォール開発における矛盾点・問題点は開発メンバー内の弱い立場の者あるいは初心者等において顕著に現れてくるものだ。

アジャイル的な開発はメンバーを孤立化させないだろう。アジャイルは、最初から最後までメン

バーがともに働き、自律的協調作業の中でチームが一丸となって前進しようとするアプローチである。

**スクラム**
例えば、ラグビーにおいては各々のポジションの役割は明確に決められているが、いったん体制が崩れた場合には各メンバーはその役割を超えて臨機応変な対応が求められる。メンバーは自分の役割と組織を補い合う自律性が要求されると言われている。仲間が守るべきポジションが崩れた時には、巨漢のフォワードもバックスに参加し、体の小さいスクラムハーフも、相手の巨漢に果敢にタックルを仕掛ける。[Squse]

これこそアジャイルスクラムのスクラムたるゆえんを表していることばだろう。

**ティーブレイク**

**マラドーナも人間だった**

こんなひどいアルゼンチンは記憶にない。マラドーナ監督の下で個の能力を生かした自由なスタイルが特徴とされてきたもののメッシやテベスが孤立し、散発的なシュートもことごとく止められた。

彼らが単独突破を試みて失敗をくり返したのは、エースを生かすために自己犠牲的なサポートをしてくれる仲間を欠いたことが原因だ。メッシも空を飛べるのでない限り、フリーな体勢でパスをもらえず、「ドイツの壁」に阻まれるだけだ。

アルゼンチンはメッシに依存しすぎた。内容が悪い試合でも、メッシが試合を決めるプレーをしたり、おとりになったりすることで勝ち進んできた。もし、マラドーナ監督がメッシも「神の手」を持っていると考え、奇跡が起こると信じていたとしたら準備不足という以前の問題だろう。

ドイツはこの試合でアルゼンチンに対して、サッカーとは何か、どうプレーすべきかの授業をした。いかなる時間帯も11人全員がプレーし、献身的に組織的プレーを繰り返した。

イビチャ・オシム　前日本代表監督　[朝日 2010.7.5]

## （9） 契約書がないまま開発に着手してはいけない ［マネジメントに関する認識］

**◎上流工程におけるマネジメントの問題**

注文書（契約書）がないまま開発を実行しているプロジェクトを見かける事がないか。マネジャーに何故と聞いたら”納期に間に合わないからとりあえず開始しています”と言うのがおおかたの答えだ。このような状況のプロジェクトは大体要件定義も完了していなく、開発体制も未整備なまま、お金や時間が管理できない状況下にある。もう少したてば何とか改善されるとの甘い判断のもと大きく焦げ付いたプロジェクトを経験したことがないだろうか。着手すべき条件が整っていないのに外部の圧力に負けて開発着手してしまうのではマネジャーとしての仕事をしているとはとうてい言えない。 ビジネスライクに言うと仕事を受けていない段階において納期の心配をすることはない。

それでも失注したらどうしようと言うのは、また別の次元の問題だ。マネジャーとしては早く条件・契約が成立するように阻害要因の解決に向けて利害関係者間の交渉・調整にちゅうちょしてはいけない。 ある意味において戦えるマネジャーである必要がある。

## （10） 人依存のレビューはレビューにならない ［プロセスに関する認識］

レビューの情景を想像した場合、知識保有者のリーダがメンバーをレビューしている光景を思い浮かべるだろう。しかしながらレビューする内容に関しての知識保有者がリーダーとして常に存在しているわけではないだろう。むしろ現実はその反対の場合が多いのではないだろうか。

実際、多くのプロジェクトを見ても形式的なレビューあるいは自分の得意な分野のみのレビューが行われているのが実態なのではないかと思われる。

多くの場合レビューは人の能力に依存しており，人が変わればレビューの質が変化しても仕方ないとのあきらめの雰囲気がある。これでは，できる人依存型の開発からの脱却は不可能であろう。 平均レベルのリーダによるレビューにて，レビューの有効性を確保するには，ソフトウェア変更部に関するソフトウェア構造および機能の影響度表によるレビューおよび過去の障害対策にて得られた重要ノウハウおよび仕様・設計等のノウハウから関連内容を抜粋したレビューノート等によるレビューを実施することが有効であると思われる。

またレビューを形骸化させている事象の一つとして、実施すべきタイミングでレビューが実施されていないと言うことがある。レビューは実行すべきタイミングにて実行されなければ何ら意味のないものになってしまうだろう。不適切なプロジェクトマネジメントにおいて時間がなかった等の言い訳のもとに、すでに製造工程に入った後で設計レビューを実施してもレビューの意味も効果もなく，多数のレビュー参加者の時間を浪費させているだけなのである.

【アジャイルの視点】

「原則１２．アジャイルチームは定期的に、より効果的な方法につき振り返りを行い、それに従ってその行動を変更し調節していく」

上記原則に示されたように、アジャイルにおいては顧客価値の重要度の順に短期の開発が繰り返される毎に動作するソフトウェアを前にした顧客およびチームでの振り返り会議が実施される。このようにレビューの対象範囲が限定され、しかも動作するソフトウェアにて行われるレビューは効果的なのだ。

## （11） けじめがついていない ［プロセスに関する認識］

開発プロセスとは、顧客要件に従い顧客価値の重要度順に機能するソフトウェアを実現するための手順あるいは段取りのことである。開発プロセスはどのような方式においても、その基本は要件開発／設計／製造／評価の工程を通して実行される。問題はこれらの各工程間における約束の曖昧さや約束の不履行にある。各ガイドラインに基づいて作成された見積書、要求仕様書、設計書、評価チェックリストは各工程間を結ぶ約束事である。

プロセス管理表はこれらの約束事が適正に履行されているか否かを検証するための管理表である。社内における業務であるとはいえ、これらのドキュメントは個人間・組織間の契約書ともいえるレベルのものである。社内的な約束ごとでさえ契約という認識のもとに行動している個人・チームは少ないだろう。それゆえ不完全なドキュメントあるいは成果物を後工程に渡すようなことが常態化しているのであろう。“時間がなかったから仕方なかった”との言い訳には何らの合理性もない。

けじめのつけられない個人の集合体がけじめのつけられないチームを形成し、顧客価値の実現どころか、品質も利益も大きく損なっているのであろう。完全なプロセスとは”けじめ”の連鎖した集合体のことを意味する。

【アジャイルの視点】

「価値3．契約交渉の価値も認めるが、顧客との協調を」は契約を軽視している訳ではないのだ。契約は重要だ。仕事は公式な約束、すなわち契約に基づいて実行されなければならない。社内における約束ごとは各種のガイドライン、規約類あるいは計画書、設計書等によって行われるのだ。アジャイルチームのメンバーにおける行動は自主的なコミットメントベースを重要視する。自己組織化チームはメンバーの自主的かつ妥当なコミットメントに基づき行動しコミットメントの責任を認識している。またスクラムは、自分たちのコミットメントを果たすのに必要なあらゆる権限を全員に提供する。

コミットメントは自主的な約束であり、自律性の発露なのだ。一方多くの萎縮した個人あるいは組織においては、どうやって相手に言質を取られないかに腐心している。後日、それは”あなたが言ったこと”ではないのかと追求されることを恐れているからだろう。このようなネガティブな心情にとらわれた個人あるいは組織においては自律性の確保は不可能であり、自らの実行を約束するコミットメントは行われないだろう。

## （12） 顧客価値はどこにあるか ［プロセスに関する認識］

製品における顧客価値がどこにあるのかについて考えて見たい。

プロセスをマネジメントするにあたって、いわゆる5W4H(注)の明確化がポイントとなってくるが、その中で何が特に重要なのだろうか。

顧客価値の実現の視点から見た場合において最も重要なことは、“What”すなわち何を作るかを定義する要件開発工程が最も重要なのだ。何を作るのかが決まっていなければ，それ以降の工程・行為は意味を持たないことは誰にでも分かることだろう。その次に重要なことは、“How to”すなわち“What”を実現する方法を決める設計工程だ。その他の項目も最終目標の実現には必須の項目だけど，いずれも「What」および「How to」の付属要素であり，人・モノ・金・時間を定量的に定義するものだ。またWhyについては全工程に渡ってのプロジェクトの実行行為の検証つまり、何故と問いかけることの意味合いを持つものだ。

プロセスのマネジメントにおいては、5W4Hの顧客価値の視点に基づく重要性の差についての判断を持ちながらプロセスを遂行する必要が有るのだ。
注.「5W4H」; 5W4Hとは、5つのW:When ,Who ,Where ,Why ,Whatと4つのH:How to ,How long ,How much ,How many である。

【アジャイルの視点】
下記は5W4Hと、アジャイルの原則および行動を対比させたものだ。

**◎ What(要件)、How long・How much・How many(仕様)**
下記は、アジャイルにおける要件開発に関する行動の原則だ。
□ 原則4. ビジネス側の人々と開発者たちはプロジェクトを通じて毎日共に作業しなければならない。
□ 原則6. 開発チームに対してあるいは開発チーム内において最も効率的かつ効果的な情報の伝達方法はフェイス・トゥ・フェイスの会話である。

アジャイルは、顧客要件開発において顧客と開発者における毎日の協調作業が必須であり、コミュニケーションの方法はフェイス・トゥ・フェイスの会話を重要視している。これらの密着したコミュニケーションの中で顧客要件の掘り起こしおよび定量化(How long・How much・How many)により仕様化を行い、それぞれの要件について顧客価値の重要度・優先度を決定しようとするものなのだ。

**◎ Why(理由・背景)**
「価値3. 顧客との協調」に関する行動は、顧客ニーズの把握であり、更に顧客ニーズの優先順位の決定を行う活動である。この段階で把握された顧客ニーズは、顧客に対して「正しく機能するソフトウェアの提供」が行なわれるまでは、真の顧客価値に対する「仮説」段階なのだ。また「価値4. 変化への対応」に関する行動は、顧客においてすら確定していない真の顧客ニーズの掘り起こしを行なうもので、その継続的な活動は日々変化するであろう未確定な顧客要件を明確化するものなのだ。この段階で把握された顧客ニーズに関しても、顧客に対して「正しく機能するソフトウェアの提供」が行なわれるまでは、真の顧客価値に対する「仮説」段階だ。
これらの仮説は、アジャイルにおける「正しく機能するソフトウェアの提供」によって顧客と共に「検証」されるであろう。
これらの「仮説と検証」の行為を支えるものは、常に現状に対する「何故？」の問いかけなのだ。顧客は顧客の言葉でしかその要求を表現できないのだ。これらの顧客の言葉を要求仕様の言葉に変換し、さらに背景に隠れている要求を掘り起こす作業が要件開発なのだ。これらの作業における顧客との密着したコミュニケーションを深めるのが「何故？」の問いかけであり、ここから顧客要求の本当の理由や背景が明確になってくるのだ。
真の顧客価値の発見は、「何故？」の問いかけによって可能になるのだ。

**◎ How To、How long・How much・How many(設計)**
アジャイルは顧客との密着したコミュニケーションの中から獲得した要求仕様に基づいて、より顧客価値の高いものから開発を実行するだろう。下記は設計に関するアジャイルの行動原則である。
□ 原則9. 技術的な優位性および良好な設計に対する継続的に払われる注意は俊敏さを増強する。

□ 原則10. 単純化、すなわち不要な仕事の削減の最大化の技術こそが、アジャイルの本質である。
□ 原則11. 最良のアーキテクチャ・要求・設計は自己組織化チームから生まれる。

アジャイルは、自律的なメンバーにより、常に優位性のある技術および良好な設計についての研究をおこたらず、無駄を排除し、最良のアーキテクチャ・要求・設計を生み出そうと志しているのだ。

**◎ When、How long（時間）**

下記は、アジャイルにおける時間の使い方に関する行動の原則等であり、アジャイルの本質が時間プロセス制御であることを明確に示しているものと言える。

□ 原則1. アジャイルの最優先事項は顧客を満足させることであり、これは価値あるソフトウェアの早期かつ継続した提供を通して実現される。
□ 原則3. 動作するソフトウェアを数週間から数ヶ月の単位で、より短い時間軸にて繰り返しリリースする。
□ 原則12. アジャイルチームは定期的に、より効果的な方法につき振り返りを行い、それに従ってその行動を変更し調節していく。

短期スプリント開発、インクリメントな継続した開発、日次スクラム会議、定期的な振り返り会議等の実行。

**◎ Who（人）**

下記は、アジャイルにおける人の行動に関する価値および原則である。アジャイルは個人の自律性を尊重し、その中で顧客あるいはチーム内における直接的会話を通した協調作業を実現することで、価値ある製品を開発すると同時に開発者の人間的成長をも獲得しようとするものだ。

□ 価値1. プロセスやツールの価値も認めるが、個人とその相互作用を。
□ 原則4. ビジネス側の人々と開発者たちはプロジェクトを通じて毎日共に作業しなければならない。
□ 原則5. 意欲のある人々でプロジェクトを構築する。彼らが必要とする環境とサポートを与え、そして仕事が完了するまで彼らを信頼する。
□ 原則6. 開発チームに対してあるいは開発チーム内において最も効率的かつ効果的な情報の伝達方法はフェイス・トゥ・フェイスの会話である。
□ 原則8. アジャイルプロセスは持続的な開発を促進する。スポンサー、開発者、そしてユーザはたえず一定のペースが維持できるようにしなければならない。
□ 原則11. 最良のアーキテクチャ・要求・設計は自己組織化チームから生まれる。

**◎ Where（場所）**

アジャイルにおいてその即応性を実現するための場所の確保は重要である。アジャイル開発の成功率は、顧客および開発メンバーが物理的に近接した場所にいることが重要であることを示している（参照；添付資料2.　米国におけるアジャイル導入率調査結果）。

下記は、アジャイルにおける場所に関する原則だ。

□ 原則4. ビジネス側の人々と開発者たちはプロジェクトを通じて毎日共に作業しなければならない。
□ 原則5. 意欲のある人々でプロジェクトを構築する。彼らが必要とする環境とサポートを与え、そして仕事が完了するまで彼らを信頼する。

### （13） 顧客価値の基本は製品の品質にある ［品質に関する認識］

ほとんどの製造会社における第1位の基本方針は「顧客価値の満足」であるだろう。「自社の利益」を上げている会社は見たことがない。「顧客価値の満足」の中心になるのは「製品の品質」であろう。また製造会社における開発プロジェクトの成否判断は主に、そのQCDの目標達成度によって行われている。すなわち開発プロジェクトにおける第1位の達成目標は「品質」であろう。

しかしながら実際の状況は下記のソフトウェア開発プロジェクトの失敗率が示す通りである。

・QCDいずれかに失敗したプロジェクト　68. 9%
・品質で失敗したプロジェクト　48. 1%
・コストで失敗したプロジェクト　36. 8%
・納期で失敗したプロジェクト　45. 4%
［日経BP2009］

「品質」で失敗しているプロジェクトがもっとも多く、コストで失敗しているプロジェクトが最も少ないのである。この数値は見方によっては、利益を守るために品質が犠牲になっているようにも見え、品質は利益の許す範囲においてのみの品質であるようにも見える。

良品を継続して提供しつづけることで顧客満足および信用の獲得を達成し、結果として利益を享受するのが本来のあるべき企業の姿である。まちがっても利益追求が先にあるのではないだろう。

利益の確保は目標ではなく結果であるべきである。利益追求は自己価値の追求である点において、自己中心的志向であり社会的公器であるべき企業の第一の目標とするのにふさわしいとは言えず、またその合理性を発見することは難しい。利益は顧客価値の満足の対価であるという顧客価値の追求にこそ合理性があると思われる。

Qの達成なくしてはC・Dの本当の達成はあり得ないだろうし、顧客の満足を得られる条件下ではじめてQCDの成立の意味があると思う。

【アジャイルの視点】

原則1が言うように「アジャイルの最優先事項は顧客を満足させることであり、これは価値あるソフトウェアの早期かつ継続した提供を通して実現される」ことがアジャイルの真の目標であろう。アジャイルにおける四つの価値およびその他の原則はすべてこの一点「顧客を満足させること」のためにあると言っても過言ではないだろう。

## （14） 価格競争力ではなく、価値競争力について

みんなコストのことだけで頭がいっぱいなのではないだろうか。経済圏のグローバル化やオフショアの強烈なコストインパクトでみんな東奔西走を強いられている。確かに同じ価値の商品ならば安くなければ生き残れないだろう。

利益は売価と製造コストの二つだけの要素で決定されるだろう。商品は市場における顧客が認める価値によって売価が決定される。顧客価値によって決定された売価は所定の利益を確保するために必要な製造コストを決定するだろう。

利益をあげる方法は二つしかないだろう。一つは商品の価値を上げて売価設定権を保つこと、もう一つは製造に要する労働の価値を下げてコストを下げることだけだ。

一般的に“コスト競争力”と言われるが、本質的には“価値の競争力”であるとの認識が必要だ。ただ安くすればいいと言うものではないだろう。ビジネスの基本はやはり“商品の価値”におかなければならない。利益を得るために良い商品を提供するということではなく、良い商品を提供する結果として利益がもたらされると言う姿勢が本来の姿である。

商品の価値を高めること、すなわち顧客が本当に欲するものをタイミング良く提供するためには開発チームの知恵と俊敏さと柔軟性が必須だ。現在のような不安定な先行き不透明な時代にあってはなおさらアジャイル的開発手法を通した人間力・チーム力の強化なくしては勝ち残ることは困難なのだ。

労働価値の削減、すなわち賃金カットやリストラだけの施策の行き着くところは組織能力の低下を伴う売上減・利益減の縮小均衡だろう。

【アジャイルの視点】

価値の競争力すなわち顧客価値の実現に関しては、そのことを第一の目標にあげるアジャイル開発方式が適しているだろう。アジャイルが示す四つの価値、「個人とその相互作用」、「動作するソフトウェア」、「顧客との協調」、「変化への対応」は、いずれも顧客価値の実現に関するものだ。これら四つの価値およびこれらに属する十二の原則の実行は顧客に対して価値ある製品の提供を確実なものにする。

### （15） 分かっているつもりのプロジェクト ［マネジメントに関する認識］

意外に思われるかも知れないが、自分のプロジェクトに関して下記の質問に全て即答できるプロジェクトマネジャーは少ない。

- 現在の品質状況は？
- 現在、予算の何%を消費したのか？
- 開発プログラムサイズの容量は？
- 現在、何%のコーディングが終了しているか？
- 開発者A君、B君、C君の今日作業内容は何？
- このプロジェクトのリスク処理状況は？
- 開発残件は何？
- プロジェクト計画書どおりに進んでいないものは何？

これらの質問は、手帳を見ながらでも良いから即答して欲しい内容である。開発状況に関して最もその実情を把握しているはずのプロジェクトマネジャーにおいてすら実は分かっているようで分かっていないのが現実の姿であろう。

多くのプロジェクトマネジャーにおける認識のあいまいさあるいは仕事に対する情熱のなさを示しているようにも思える。これでは，プロジェクトの成功はおぼつかないものになるだろう。毎日、基本的なプロジェクト指標のデータの把握に努める必要がある。

【アジャイルの視点】

プロジェクトの基本的指標について答えられないプロジェクトマネジャーとはどのような人たちなのだろうか。おそらくプロジェクトの目標すらあいまいな状態で、日々チームメンバーと問題についてコミュニケーションすることもないのだろう。下記のアジャイルの価値および原則はウォーターフォール開発においても有効であり、即刻実行すべきものなのだ。

- □ 価値1．プロセスやツールの価値も認めるが、個人とその相互作用を
- □ 原則5．意欲のある人々でプロジェクトを構築する。彼らが必要とする環境とサポートを与え、そして仕事が完了するまで彼らを信頼する。
- □ 原則6．開発チームに対してあるいは開発チーム内において最も効率的かつ効果的な情報の伝達方法はフェイス・トゥ・フェイスの会話である。
- □ 原則9．技術的な優位性および良好な設計に対する継続的に払われる注意は俊敏さを増強する。
- □ 原則11．最良のアーキテクチャ・要求・設計は自己組織化チームから生まれる。
- □ 原則12．アジャイルチームは定期的に、より効果的な方法につき振り返りを行い、それに従ってその行動を変更し調節していく。

### （16）「とりあえず手をつける」ことは余裕をもたらす［マネジメントに関する認識］

忙し過ぎて何も手がつかないことが常態化していないだろうか。忙しいから手をつけられないのではなく、手をつけていないから忙しくなるのではないだろうか。

複数の仕事が重なった場合、特に優先順位がほぼ同じ場合にはどうしてもやりやすい方から手をつけがちだが、未着手の仕事の方が途中で優先順位が高くなってしまった場合、非常に困ったことになる。このようなケースに対応するために、重要と思われる複数の案件はどれも「とりあえず着手しておく」ことを日常から心がけておいた方が良いだろう。仕事中の気分転換ということで用もないのにブラブラと他人の仕事の邪魔をしに行く人がいるが、仕事中の気分転換方法としては別の案件に取りかかることが結構気分転換になるものである。気になる案件について常に未着手のものがないという状態は、急に優先順位が変わった場合にも有る程度の対応が可能となり、何と言っても自分自身の精神状態を優位で良好な状態にしておける大きな効用がある。余裕があるマネジャーとは仕事が絶対量的に余裕が有るわけではなく未着手な案件が少ないという”安心感”から来るものだろう。

【アジャイルの視点】

「とりあえず手をつける」ことは「価値4．変化への対応」行為の一つだろう。日々の変化に対応するためには、とりあえず手をつけることから始めることは良い習慣となるだろう。この習慣は、マネジメントにおける俊敏さを育成し、それらの継続した積み重ねから顧客の要求の変化に対するチームの柔軟性が生まれてくるだろう。

### （17） 責任について ［マネジメントに関する認識］

『誰が』の記述がない、あっても今度は期限が明示されていない報告書・会議がごく当たり前の様に行われていることがないか。 ただ口頭で話すばかりで、いっさい議事録が取られない会議が行われてはいないか。『誰が』の記述がないと言うことは責任者が不明と言うことだ。 期限の明示されないアクションは実行されないのと同義語かも知れない。

だれもが当事者・責任者になりたがらない組織やチームになってはいないか。責任と言うことについて重苦しく考える必要はない。 単純に考えると「この仕事は私の担当です」と言えるかどうかなのだ。

**責任について**

自由と義務を均衡させる手段の第1歩として人間の責任に関しもう一度自分の頭で何をすべきか個々人で考えなければならない。日本人全体を覆う無責任主義下における思考停止傾向から脱却し「考える日本人」にならなければならない。

［朝日 1998］

**ティーブレイク**

**世界は「見方」で出来ている**

事実うんぬんと書いてきたが、むき出しの事実なんてものは、そこらに転がってやしない。誰かが実感を持って「そう見た」ものがあるばかりだ。賛同者が多ければ真理なんぞになり、少なければ妄想となる。世界はそんな「見方」で出来ている。実感の無い所に実を見ず、そのかわり実感さえ沸けば実が無くともそれを見る。だから詐欺なんてことも成り立つ。

紙っぺらに描いたものを、花だ美女だと言ってのける絵描きなんて商売も、かなり詐欺師寄りだ。ただ、紙の上に嘘八百を描きつつも、観る者の胸の中には「ほんとう」が像を結ぶように四苦八苦している点、少しはマシか。

山口　晃　画家　［朝日 2008］

## （18）「それは私の担当ではありません」に学ぶ　［人に関する認識］

「それは私の担当ではありません」（ピータ・グレン著）の文章を抜粋してみよう。

本書は顧客サービスについて書かれたものだが仕事に対する人の取組み方や考え方に関して実によくその本質を言いあてている。我々のソフトウェア開発に従事するものに対しても仕事に対する考え方を一新してくれるに違いない。

**このひどい仕事ぶり！**

私たちはいいサービスを受けてもいないし、提供してもいない。品質が落ちているのもサービスが悪いのもあたりまえと、たいしてがっかりすることもなくなっている。 この国では不平をもらすことで一致団結している。「それは私の担当じゃありません」というのが国歌になっているのだ。

今まさに変革を起こす時に来ている。サービスは落ちるところまで落ちているのだ。上に立つ者は何もしないし、商品は用をなさない。製造業者は不良品の修理に1ドルにつき約20セントを費やし、サービス業界では不手際をカバーしたりクレームを処理したりするのに、1ドルあたり30セントも費やすことになる。

最初からきちんとやっていればこんなことはしなくてすむのだ。私達は「サービス経済」の中で暮らしているのに、サービスはなされていない。でも誰も気にもしない。「私の担当じゃありません」というわけだが、実際、誰の担当でもなくなっている。［ﾋﾟｰﾀ･ｸﾞﾚﾝ］

これは米国での話しなのだがまるで現在の我々の状況を物語っているようだ。 実際ソフトウェア開発者たちは毎日といっていいくらいバグと呼ばれる不具合対応に追われ膨大な時間を取られるためますます開発を行なうための時間は短縮され、これがまた次の不具合を呼ぶという悪魔の連鎖にはまっているのだ。何ともヒドイ仕事を続けているのだ。

どこの誰を見てもこれは「私の責任じゃありません」みたいな顔をしている状況は本当に奇妙な理解しがたい光景だとは思わないかい。

著者は続いてこうも言っている

革命はたった一人でも始められるのだ。 ドクター・マッティンリーいわく。多くの人を動かすことができなくても、たとえ100人のうち3人しか動かせなくても、できるだけのことをすべきだ。 そういうわけで、この本は100人のうちのその3人に捧げたい。十分に闘う気があり、日々人生を、それがどんな人生であれ十全に生きている人たちに。 ファイターは「それは私の担当じゃありません」などとはけっして言わない。人生そのものを担当しているのだ。

ヘレン・ケラーはこう言っている。「科学によってほとんどの害悪に治療法が見出されるだろう。しかし、最も悪いもの～人間の無関心にきく薬はない」

行動を起こすためには動機づけしなければならない。だが、動機づけされていない人は多い。そういう人は哀れである。自分の人生を生きようという気があまりないのだ。

私はいつも人が自分で思っている以上のものをその人の中に見出そうとしている。 自分に対する評価を捨てるべし。 たいていは、そうあるべきもの、そうでなければならないもの以下にしか自分を見ていないし、質が落ちているのはそのせいなのだ。 自分のやっていることに専念し、自分自身を愛して欲しいと思う。 社会には手本となるような人がいて欲しいものだが、今日それは難しい。[ﾋﾟｰﾀ・ｸﾞﾚﾝ]

この一連の文章は過去の米国の話なのだろうが今日の日本の実話としか思えないのだ。

心ある人は開発者たちに悪魔の連鎖を断ち切らせようと口をすっぱくして業務改善を実行したらと迫るが、開発者たちは口をそろえて実務に忙しいので改善に手が付かないと言い訳ばかりしている。この状況は一体何なのか。開発者たちの言っている”実務”って一体何なんだ。いままでのやり方でやればまた問題を引き起こすような傷のついたやり方のことを”実務”って言っているのと同じにしか聞こえないがキミはどう思う。

**「ノー！」をやっつけろ**

これほど気をくじかれる言葉はない。お客としてもサービスする側としてもしょっちゅう「ノー！」を耳にするが、大事なのは、その時どういう行動に出るかである。手軽で安易で、非生産的であり、敗残者が好んで使う「ノー！」。これは「私の担当じゃありません」を短くずばりと言いかえたものだ。

「イエス！」を探し求めよう。やる気を持続させること。山頂に到達した時こそ、もっと登ろうとしなければならない。動機を持つことが生きるということなのである。[ﾋﾟｰﾀ・ｸﾞﾚﾝ]

何でノーとしか言えないのだろうか。プロに頼まれた仕事は最初から難しい仕事だってことは当り前のことだろう。それでも返ってくる言葉は「できません」「今忙しくてできません」のオンパレードなのだ。 何故「これこれの条件がそろえばやります」とか「忙しいけどあさってからなら取り掛かれます」って答えられないのか。プロみたいな顔をしているけど実は草野球レベルの素人根性の人間ばかりが増えているのかも知れない。これは別に担当者レベルだけの話をしている訳じゃなく、上は経営層から部長、課長、主任にいたるまで日本における実業界を広く覆い尽くしているのだ。

**良い顧客になるには**
革命的な顧客になるための信条:
・ 何を求めているのか、どんな方法でするのを望んでいるのかを知り、そのとおりにすることだ！
・ これが求めていること、望む方法だ。そのとおりにやってもらいたい。
・ すべきことを、すべき方法でやってもらわなければならない。
・ 望むことを、望む方法で、必要な時に、そのとおりにやってもらうことだ。
いくじなしには何も(！)手に入らないのだ。[ﾋﾟｰﾀ･ｸﾞﾚﾝ]

**良いサービス提供者になるには**
・ 彼らが好むこと、好む方法を見出し、そのとおりにすることだ。
・ 従業員の自主性に期待するのは間違いである。あてにしてはいけない。多くの人には動機づけが、しかも一度ならず動機づけすることが必要である。これは終わりのない仕事である。
・ 人生には成功よりも泣き言のほうが多く、情熱は失われていくのだ。自己啓発できる人はごくまれなのだ。
・ 考えと現実の間、気持ちと行動の間には暗がりがある。
顧客サービスを提供する人～実はもっと生産的な人生を望んでいる人すべてのことだが～は、その暗がりを警戒し、追い払い、行動するという楽しい世界に生きなければならない。
・ 賢明な管理職は顧客になってみる。
顧客サービスをどうとらえるかということはどう生きるかということなのだ。[ﾋﾟｰﾀ･ｸﾞﾚﾝ]

上記は顧客とサービスの提供者両方について何が価値のあることかについて同じことを語っているのだ。セブンイレブンやアジャイルが言っているビジネスあるいはプロジェクトで成功する黄金律の一つは、顧客が欲しがっているものを知りそれを直ぐに提供すること、つまり”顧客の要求に対して即応すること”なのだ。 これは実に単純なことで誰でもすぐに理解できることだが中々実行できないのだ。 何故実行できないかって言うと人間はみんな誰かに何かしてもらうことは好きだけど、自分が誰かに何かしてあげるためには非常な努力が必要だからやりたくないってのが本心なのだ。だから自分でやりたくないって人に代わって有料でやってあげましょうってのがプロの仕事の始まりだった訳なのだ。
だけど世の中豊かになって楽なことばかりに慣れてしまうとプロの気持ちも緩んでしまうわけよ。さあキミならどうする。

## 4. タイムマネジメント:時間を制御する

### （1） タイムマネジメントの目的

タイムマネジメントの成果というとプロジェクト活動においては、約束した期間の中で約束した成果物の量および質についてどれくらい効率よく生み出せたかということになるだろう。

みんなが最も知りたいことは納期を守る方法や効率的な開発方法についてだろうが、まずなぜタイムマネジメントが必要なのかについてその目的をキチンと知ることから始めたい。

その目的は下記プレジデント紙における記事が言い尽くしているのでそのまま引用する。

**タイムマネジメントの究極の目的は「競争力の強化」である**

『ボストンコンサルティンググループ（ジョージ・ストーク・ジュニア）は1980年代に日本企業の強さを分析したところから「タイムベース競争」というコンセプトを提唱していた。当時日本企業は新車の開発から発売まで36ヶ月でできたが、アメリカ企業は60ヶ月もかかった。24ヶ月もの差は仕事の重複が少なく、同時並行でできることは必ず同時並行でやり、早くから下請けまで含めてすり合わせを行うなどいくつもの時間の使い方の知恵が競争力の源泉だと指摘した。日本の競争力の源は川上から川下まで直結した時間管理にある。

タイムベース競争の基本原理は「時間こそが顧客と企業の双方にとって最も貴重な資源である」という考え方である。

時間を短縮し、スピードや柔軟性を高めて「顧客が求めるときに直ちに求めるものを提供する」という考え方に立つと、様々な利益が生まれる。生産性が向上しリードタイムが短ければ顧客にとっての付加価値は高く、それに応じた価格設定が可能になる。新製品や新サービスの開発期間が短くなれば、顧客ニーズや需要も予測しやすく、リスクが軽減される。リードタイム短縮で予測精度が高まれば、在庫や欠品リスクが減り、結果として競争優位に立ってシェアを拡大できる。全体最適のタイムマネジメントのためのキーワードは競争力なのだ。顧客・営業・協力会社・関連部署等すべての中でどことのすり合わせに、どれだけの時間配分をするか、どれだけ時間の節約や効率化ができるか、それによってどんな付加価値がつくれ競争力が上がるかという発想に立ってタイムマネジメントを行う。』［プレジデント 2004］

### （2） タイムマネジメントのポイント

タイムマネジメントのポイントを整理すると次のようになる。

- □ 時間の使い方の知恵が競争力の源泉だ。
- □ 時間こそが顧客と企業の双方にとって最も貴重な資源だ。
- □ 顧客が求めるときに直ちに求めるものを提供する。
- □ リードタイムが短ければ顧客にとっての付加価値は高くなる。
- □ 付加価値の高さに応じた価格設定が可能になる。
- □ 新製品や新サービスの開発期間が短くなれば、顧客ニーズや需要も予測しやすく、在庫リスクや欠品リスクが軽減される。
- □ 結果として競争優位に立ってシェアを拡大できる

次に実際のプロジェクトの現場におけるタイムマネジメントについて考えてみよう。

### （3） スケジュールは最初から遅れている ［時間に関する認識］

**【現在の問題点】**

開発プロジェクトにおける持ち時間は最も厳しい真の制約条件だ。

多くの開発プロジェクトにおいては、その要求仕様が全て決定する前に納期が決定されていることが残念ながら多い。納期決定後に次から次へと仕様変更あるいは追加仕様が盛り込まれているのが実態なのだ。その実態から見て、ほとんどの開発プロジェクトにおいて、スケジュールはプロジェクト開始前から遅れていると認識した方がよい。このような状況下でウォーターフォール開発においてはどのような対策が行われているだろうか。

人員の追加投入、仕様の軽減化等の対策が行われても遅延を回復できない場合も多く、開発の各工程における残り時間はついにタイムアップとなってしまう。この結果、前工程は後工程に不完全な成果物を渡さざるを得ない状況になり最終的に粗悪な品質のソフトウェアを出荷してしまうことになる。表向き納期は守ったが品質が犠牲になったしまった場合、本当の意味で納期を守ったとはとても言えない。

最初から持ち時間が決められており後から後から追加仕様・仕様変更が発生してくるような状況は後だしジャンケンと同じで最初から負けが決まっているようなものだ。何でこんな愚かなことをいつまでも続けているのか。会社が決めたことだから、上司が決めたことだから仕方がないのか。仕方ないで結局納期も品質も利益も人の健康も犠牲にして誰も喜べないようなやり方をずっと続ける積りなのか。こんなやり方は正しい仕事とはいえない。

**【解決方法】**

解決のキーワードは持ち時間なのだ。現状のウォーターフォール的開発方法を続けるのなら最初の要件定義段階でプロジェクトの実力つまり所定の期間内に動員できるリソースつまり人・モノ・金のレベルの範囲内で可能な開発内容はどこまでかその限度を見極める必要がある。プロジェクトリーダーはこの見極めを上司まかせ会社まかせにしてはいけない。次から次へと多くの機能開発を求められるのならば、このリソース条件を変える交渉をタフに行なうしかない。仕方ないとあきらめたらそれで終わったと思った方が良い。あきらめてしまえば間違いなくプロジェクトは最初から失敗することが運命づけられてしまうのだ。

持ち時間の有効な使い方は、顧客との密なコミュニケーションをベースとした要求仕様における顧客価値の重み付け順の開発、あるいは顧客要件の正確な仕様化等による要件スコープの明確化によってしか実現できない。プロジェクトは”ガンバロウ”の世界ではない。

【アジャイルの視点】

アジャイルは経験的プロセス制御であると言われており、その主要な活動は短期間に繰り返し実行されるスプリント開発を中心に行われる。開発の順番は顧客価値の高い仕様の順に実行される。そのスプリント開発のメカニズムにおける開発プロジェクトの四つの制約は下記の通りである。

プロダクト開発プロジェクトはすべて以下4つの変数によって制約を受ける。

1. 与えられている時間
2. コスト（人とリソース）
3. 出荷時の品質
4. 出荷時の機能

スプリントは特に最初の3つの変数を決定する」[Agile Scrum　p61］

四つの制約の筆頭に、時間の制約をあげているのである。またスプリントの進捗管理は、横軸に「残り時間」を縦軸に「残り作業量」を表すバーンダウンチャートによって行われている。これは、アジャイル開発がプロジェクトに残された時間を明確に意識した管理を行っていることを示している。重要なことはガントチャート等が示す遅延時間ではなく時間の制約を直接的に示す「残り時間」である。

アジャイル（俊敏性・即応性）はその名前が表しているように、その本質は残時間プロセス制御にある。プロジェクトで失ってはいけないものの筆頭は”時間”である。その他のリソース、資金・モノ・人材は全て人為的なもので何とかしようとすれば何とかなるものであるが、失った時間は取り戻せない。 また、時間の制約に関して次のピーター・ドラッカーによる主張はその核心を突いているだろう。

「成果をあげる者は仕事からスタートしない。計画からもスタートしない。時間からスタートする。何に時間がとられているかを明らかにすることからスタートする。・・・成果をあげる者は時間こそが、真に普遍的な制約条件であることを知っている」 [Drucker]

### （4） 不良なコミュニケーションにより時間は失われる ［時間に関する認識］

形式だけのレビュー会議、何もアクションを決めない会議、報告者の意思がない報告、不要な雑談メールの配信などが行われてはいないだろうか。これらの行為はプロジェクトの貴重な時間を無駄に浪費するだけだ。プロジェクトで最も貴重なものは、“時間”であるということをしっかりと認識しておく必要がある。人材や機材や開発費も貴重なリソースではあるが、ある程度は人為による調整が可能である。しかしながら失われた時間は取り戻せないものなのだ。

【アジャイルの視点】

プロジェクトにおいて一旦プロセスが崩れ始めると実行されるアクションは後追いの行動に陥り、本来の目標は忘れ去られ形式を整えるための行動ばかりが行なわれるようになってしまう。このことは特に各工程のタイムスパンが長いウォーターフォール開発では顕著に現れてくる。例えば、要件定義工程に源を発し 6 ヵ月後のコード製造工程において大きなプロセス崩れが現出してしまった場合には、もはやこの過ぎ去った 6 ヶ月を取り戻すことは不可能だ。このようなプロジェクトにおける要件レビュー、設計レビュー、コードレビューはおそらく形式を整えるだけの実効性のないレビューであっただろう。

アジャイルにおいてもプロセス崩れが発生する可能性はあるだろう。しかしその場合に失われる時間は最大でも短期間であるスプリント期間の長さ以上にはなり得ない。また日々の直接的なコミュニケーションの実行を原則とするアジャイルにおいて形式だけの会議・レビューが行なわれる可能性は低い。形式だけの行為はアジャイルとはいえないだろう。

### （5） ドキュメントは時間の拘束から逃れる有効な手段の一つだ ［時間に関する認識］

ドキュメントは大量の情報を正確に複数の他者に伝えることを目的として作成される。多くの失敗プロジェクトは有効なドキュメントの作成にも失敗しているのだ。その言い訳は必ず“作成する時間がなかった”なのだ。ソフトウェア開発は情報システムの開発であり、そのプロジェクトが情報伝達のポイントであるドキュメントの作成に失敗している事実は皮肉なものだ。多くのソフトウェア開発の現場でドキュメントが更新されないまま放置されていることは関係者衆知の事実だ。

ドキュメントのもう一つの大きな役割は、開発者およびチームが不要な時間的拘束から逃れられることにあるだろう。ドキュメントは複雑かつ大量の情報を正確に任意の時間に複数の他者

に伝達できる特徴をもっている。またドキュメント情報は口頭による情報伝達と異なり不揮発性であり任意の時間に何回でも利用が可能である。ドキュメントがなければ情報伝達のために複数の人間、複数の組織に対し繰り返し繰り返し説明が必要となり、個人あるいはチームは時間的な拘束をたびたび受け、膨大な時間を失ってしまうだろう。それにもかかわらず“時間がなかったから作成できなかった”との言い訳には全く合理性が見られないと言わざるを得ない。時間を生み出したかったら必要なドキュメントに基づいた開発を実行することが最も効果的なことだ。

ドキュメントの作成においては、顧客価値の重要性の順位に従ってこのプロジェクトに真に必要なドキュメントは何かの判断が重要だ。不要な時間を削減したければ必要なドキュメントの作成は必須だ。どちらが得かよく考えよう。

### （6） 自分で期限を切れば余裕が生まれる ［時間に関する認識］

いつも締め切り期限に追われてはいないだろうか。他者によって決められた期限ばかりで行動している限りはこの締め切り前のドタバタ症候群からの脱出は難しいだろう。 公式の期限より少し前の期限を設定し、それを達成するためのアクションを実行する習慣を身に着けることが有効であろう。１日前倒し、２日前倒しの計画・実行は自分に良い意味でのプレッシャーを与え、良い知恵も浮かぶものである。他律的なプレッシャーには重圧を感じ、自律的なプレッシャーは知恵とやる気を生むだろう。自分自身で期限を切ることで突発的事象にも対応できる時間的余裕を持たせることもできるだろう。

自己およびチームの自律性の中からしか時間的ないしは精神的余裕は生み出せないものなのだ。

【アジャイルの視点】

「原則１１．最良のアーキテクチャ・要求・設計は自己組織化チームから生まれる」に言う自己組織化チームは、命令や指示に基づく縦割りの組織ではなく、チーム内での話し合いや自律的な判断に基づいて各人がなすべきことを行う組織のことである。このような自律的に行動する個人やチームでなければ自己の時間を制御することはできない。真の意味で、自己の時間は他人に貸すことはできないし、その逆もまた同様なのだ。

自律性とは自己の時間を生き抜くということだ。時間プロセス制御がその本質であるアジャイル方式を成立させるものは自己ないしはチームの自律性に他ならない。

### （7） 時間を制するものは全てを制する ［時間に関する認識］

コスト、利益、品質、顧客満足等、人間が生み出すことのできるあらゆるものは時間を通してのみしか具現化できないものであり、また人間における能力の獲得についても同様だ。時間は開発プロジェクトのみならず全ての人間活動における真の制約条件であり、時間は全てのものの源なのだ。

プロジェクトの成功の基本を顧客価値の実現に対する時を待たない“即応性”に求めることは合理的なことだ。各人の有限な手持ち時間の中で最も重要なことは、時間的な無駄を排除し、日々変化する顧客価値の要求に寄り添えるような活動を行うことなのだ。

プロジェクトの失敗は表面的にはQCDの失敗として現れるが、その本質は個人ないしはチームにおける許容時間のタイムアップだろう。自分のあるいは自組織の持ち時間がなくなる前に他人のあるいは他組織の時間を自分のあるいは自組織の時間として取り込むことが成功する唯一の方法だろう。

### （8） 時間をムダにする方法 ［時間に関する認識］

下記にG. M. ワインバーグの「コンサルタントの道具箱　勇気と自信がもてる16の秘密」から時間制御に関する役に立つ言葉を抜粋してみた。

- ○ 混乱は時間の無駄。混乱に飛びついてどたばたする前に冷静に事実を把握する事だ。
- ○ いらだちは時間の無駄である。現実を受け入れないこともしかり。
- ○ 急がば回れ。信用は時間の代わりになる。正しく使えば、金もしかり。
- ○ 結論に飛びつくのは時間の無駄である。原因追及もしかり。大騒ぎせず見極めること。
- ○ 時間の無駄の原因は休暇ではない。計画不足である。
- ○ 余裕は、緊急事態発生時の予備時間として使えば時間の節約になる。頼りになる人たちに頼ることもしかり。
- ○ 新しいメンバーに考えを伝授するには時間がかかるが、無駄な時間ではない。
- ○ 研修も時間を食うように思われる活動の一つだが、早い時期に行えば、実際には時間の節約になる。
- ○ テストは時間を食うように思われるかも知れないが、早い段階にうまくやれば、かかった以上の時間を節約できる。
- ○ プロジェクトが計画に合わないのではない。計画がプロジェクトに合っていないのだ。人間のもろさを見込んでいない計画には欠陥がある。
- ○ 計画とは予測である。計画を早い段階で何度も確かめ、修正する予定を立てておく。
- ○ 完璧主義はスケジュールの敵である。妥当性はスケジュールを救う。
- ○ 妥当であれば、大幅な時間の節約になる。完璧主義のせいでパニックに襲われたことが何度かある。完璧な質を求めるのではなく、許容できる質を考えるようにするのが主な治療法である。
- ○ お金で幸福は買えないかもしれないが、信頼性は買うことができ、ひいては時間を買うことができる。
- ○ 質は妥協が可能だが、どこまでも妥協できるものではない。
- ○ 依頼主が、「すべてうまくいけば」プロジェクトは予定通りに進むと話すのをよく耳にする。私はプロジェクト・ビジネスに足を突っ込んで40年以上になり、何千件というプロジェクトを見てきた。「すべてうまくいく」プロジェクトには、まだ一度もお目にかかったことがない。

［ワインバーグ］

**ティーブレイク**

**金々節**

「金だ金々　金々金だ　金だ金々　この世は金だ　金だ金だよ　誰がなんと言おと　金だ金々　黄金万能

金だ力だ　力だ金だ　金だ金々　その金欲しや　欲しや欲しやの　顔色目色　見やれ血眼くまたか眼・・・学者、議員も政治も金だ　神も仏も　坊主も金だ　金だ教育　学校も金だ」

添田唖然坊　大正末期の演歌師 ［朝日 2010.7.26］

## 5. プロジェクトとは何か

その規模の大きさはともあれ我々は何らかの組織に属しているものだ。 その組織というものは良いにつけ悪いにつけ我々に大きな影響を及ぼす存在だ。 組織やプロジェクトはある目的を達成する使命をおびた人間の集団なのだ。 プロジェクトも組織の一種だ。 組織ってものの性格を知らないと組織の中で生きていくのがつらくなるからここでちょっと組織というものについて考えて見よう。

### (1) 未熟な組織文化

組織にも未熟なものと優れたものがある。未熟な組織とは簡単にいうとその構成員にとってもその組織と関係する他の人や組織にとっても困った存在のことをいう。いわゆるダメな組織だ。未熟な組織の特徴は次のようだろう。

【未熟な組織の特徴】

- 情緒的人間関係を重視した親分・子分的派閥にて構成されている。
- 情緒的固定観念等による短絡的・独善的な行動パターンが多い。
- 非科学的な情緒的思考による空気・雰囲気が支配的。
- 科学的データを軽視する。
- 単一情報だけで突っ走る情報軽視、認識力の弱さ。
- 裏付けのない楽天主義。
- 精神論だけの組織。
- 知恵が出せず体力・気力だけで勝負する組織。
- 組織内外のコミュニケーション機能不全。

### (2) プロジェクト崩壊の情景

ダメな組織やプロジェクトはいずれ崩壊していくものだ。 キミは目の前で組織やプロジェクトが崩壊したのを目撃したことがないかい。 組織が崩壊していくときの情景は次のようだろう。

【プロジェクト崩壊の情景】

- 部長が課長の仕事をし、課長は主任の仕事をし、主任は社員の仕事をしている。
- 何ができていて何ができていないのか誰に聞いても分からない。
- いつ終わるのか誰も分からない。
- 誰が何の責任を負っているのかも分からない。
- 仕様は未決定のまま、仕事は外注へ丸投げされている。
- 外注から突然巨額の請求書を突きつけられる。
- 何故？の質問に誰もが沈黙してしまう。
- 人はたくさん居るのに進捗がはかばかしくない。
- 徹夜続きでみんな疲れ、暗い表情で笑いも会話もない。
- 人が足りない、マシンが足りない、お金が足りない。
- けんか、なぐりあい、欠勤、職場放棄などの問題行動の発生。

### （3） プロジェクト崩壊を招くマネジャーのタイプ

組織を運営していくのはマネージャの責任だ。マネージャは、プロジェクトには常に危機が訪れ、問題は毎日発生してくるということがあたりまえのことだという認識と覚悟を持つ必要がある。
問題はプロジェクトが危機に陥った時にそのマネジャーがどう対応するかなのだ。 日々問題の解決を迫られる組織、特に開発プロジェクトにおいて機能不全に陥りやすいマネージャのタイプは次の通りだ。

- **手配師型マネジャー**
  単に工数穴埋めのためだけの人の手配、情緒的人員配置等に陥りやすく、表面的帳尻が合えば良しとするだけで、とうてい問題解決のための有効な組織編成ができない。
- **形式調整型マネジャー**
  表面を取りつくろうだけの成果調整、数字調整に走りやすく、真の問題が見逃され時間とともに問題が深刻化し、顕在化した時点で時すでに遅しの場合が多い。
- **独善型マネジャー**
  他者とのコミュニケーションが機能せず、いったん誤った方向に進み出したら最悪の状態に至るまで止められない。
- **権力誇示型マネジャー**
  昔、流行した歌に「あんたが大将」と言うのがあったが、まさにそのタイプのマネジャーだ。常に自分の権力を誇示することでしかリーダシップを発揮できないと勘違いし、結果的に組織員の面従背腹をまねきリーダシップを失い組織崩壊の道をたどる。

こんなマネジャーだけにはなりたくないものだ。

### （4） プロジェクト崩壊の主な原因 ［マネジメントに関する認識］

何故プロジェクトが崩壊するのだろうか。 ここでいくつかの原因を考えてみたい。

#### ① あいまいな目標

プロジェクトリーダーが組織の内外に対してその目標をはっきりと説明できない場合、そのプロジェクトは間違いなく失敗する。
プロジェクトのキックオフ会議でよくプロジェクトリーダが決意表明するのを見かけるが、その目標についてスローガンばかりでさっぱり具体的な目標を語れないリーダが多すぎる。
スローガンは単なる”ガンバロー”の世界の話だ。スローガンはあいまいな言葉だから達成結果を検証できる性格のものじゃない。一方本当の目標は数値に裏づけられた具体的な内容であり達成結果が検証可能であり、この目標は一旦公開したら達成しなければならない約束となってしまうものだ。

あいまいな目標ではプロジェクトメンバーは何をいつまでに実現するように目指せばよいか分からず各人勝手の解釈で適当にバラバラに進み始めるものだ。 これではプロジェクト開始時点でこのプロジェクトは失敗する運命におかれたということだ。
同じ目標を目指すためのポイントは次のようだ。

**【同じ目標を目指すためのポイント】**

□ プロジェクトリーダはメンバーに対して目的・目標の明確化を行い、全員の共通認識にすること。

□ 目的遂行のために各自の具体的業務内容の認識が正確になされるようにすること。

□ 目標の軌道修正時においても逐次メンバーに対する説明を怠らないようにすること。

□ 目標は抽象的なものではなく具体的なものにする。すなわち形容詞、副詞等のあいまい語を排し数値で示すようにする。抽象的表現は単なる”スローガン”に過ぎず目標とは言えない。

**② 戦略の欠如**

プロジェクトには戦略が必要だ。プロジェクトはいわば戦場なのだ。戦場で戦うには戦い方をあらかじめ用意しておく必要がある。これがプロジェクトの基本戦略だ。

組織がある所には必ず多くの問題・課題がある。これらの問題・課題に個別に対応していたのではいつまでたってもなかなか解決できないことは経験の示すところだ。いわゆる”モグラたたき”状態に陥いってしまう。

開発組織における問題・課題の多くはQCDすなわち品質、コスト（利益）、納期（期間・時間）の三つと”人間”の問題に集約することができる。この“QCD＋人”の四つの要素に関して現状の組織の問題を洗い出し、将来それらをどの様に改善していくかをはっきりと示したものが基本戦略となる。

**③ コンティンジェンシー・プランの欠如（不測の事態に備えた対応計画）**

仕事において常に不測の事態に直面する開発プロジェクトは、不測事態発生時に瞬間的に有効的な対応をとれるようにするためのコンティンジェンシー・プランを持っていることが重要だ。

例えば、ある施策を実行するにあたって短期決戦で行こうと決めた場合においても、それが失敗しそうになった場合に他の施策で代替えする等の案を当初から考えておくことだ。

別の言葉で言うと「戦略オプションの広さ」を持つと言うことだ。

目標は動的に動いていると言う認識をもち、戦略遂行に際して当初の計画と実際の成果とのギャップをどこまで小さくすることができるかということによって、プロジェクトの成否が分かれる。

**④ 時間・タイミングに対する認識不足**

複雑で困難なプロジェクトほど、整然としたプロセスに基づいた共同作業を実施し、なおかつ実行すべきタイミングを逃さない様にしなければ仕事の成功はあり得ない。マネジャーは時間の有効利用、タイミングの重要性について常に目を光らせていなければならない。 良い意味で「機を見るに敏」でなければならないのだ。

**⑤ 情報戦略の貧困さ**

重要事項決定にあたっては複数の情報源をもとに決定すべきだ。単一情報に基づくアクションは手ひどい結果を招く危険性が極めて高いことを承知しておかなければならない。

**⑥ 資源投入戦略の誤り**

プロジェクトが危機状態にもかかわらずマンパワーの投入先の誤り、またはさみだれ的投入しか行われないことは何ら良い効果を生まない。プロジェクトマネジャーが不足なのにプログラマーを大増員するようなことは愚かなことだ。またプログラマー10名を同時に投入しなければなら

ないのに今週2名、来週3名などの逐次投入は何にも危機を回復しないだろう。

**⑦） 現場主義の喪失**

現場にあまり行かないあるいは居ないマネージャは、ほとんどプロジェクトの実態を知らないと思ったほうがいい。 指揮官が現場にいなくて何を指揮できると思っているのだろうか。

プロジェクトマネジャーの現場主義の心得を挙げてみる。

**【現場主義の励行】（マネジャーの心得）**

□ 毎日やろう （課題バラシ、担当者の巡回フォロー、対応策の指示）
□ 毎日聞こう （担当者の不平・不満、関連部署の状況、お客様の話）
□ 毎日見よう （リスク・課題、進捗、メンバーの健康状態）
□ 毎日まとめよう （課題と対策アクション）
□ 見える様にしよう （仕事の内容・範囲、口頭ではなく書類で残すこと、管理表の日々更新すること）
□ マネジャーは心理的にも、地理的にもいつも担当者のそばに居よう。
□ お客様を早い時点で仲間として巻き込もう。

情報とは、一旦誰かの頭を通過して言葉に置き換えられたものなので、必ず現実との間にズレがあることを承知しておこう。不正確な情報によって行動を間違えないようにするためには、現場で確認し、複数の情報源に当たってみることだ。現場の最前線こそが収益の源なのだ。

これらができるだけでもプロジェクトの危機は大幅に減らすことができるのだ。

**⑧ 組織エネルギーの内部消耗**

組織のエネルギーを無駄に費やしてはいないか。我々は役に立つ製品をお客様に届けることを第一義の目的として活動をしている。 言葉を変えて言うならば我々は組織の資源として保有している、人・モノ・金・時間などのリソースを製品に転換する作業を行っているのだ。これを開発と呼んでいるのだ。この意味分かるよね。

企業活動において組織員は最小の資源投入で最大の成果を生み出さなければならない責務を負っているのだ。 つまり目的とする製品をお客様に届けることに役立たない活動や阻害する活動は全て組織エネルギーの無駄使いなのだ。

組織活動における主なエネルギーの内部消耗の例は次のようなものがある。

【組織エネルギーの内部消耗】

- 派閥争い、勢力争い
- 個人間の私憤による闘争
- 責任の押し付け合い
- いわゆる「組織の壁」による非効率化
- 内部の事情調整ばかりの組織運営
- お客様より上司の目ばかり伺う平目型的行動
- 形式主義、体裁主義による意味のない計画、いわゆるお役所仕事
- 優先順位のない場当たり的活動

数十年間いろんな開発組織を見てきたけど上級マネジメントに行くにしたがってこの内部消耗活動に費やされる時間は大きくなるね。 一体何をやっているんだろうね。まねしない方がいいと思う。正しい仕事に情熱を燃やそうじゃないか。

### （5） データドリブンな組織になってるか ［マネジメントに関する認識］

開発プロジェクトにおいて、経験とデータはどっちがより重要だと思う？ どっちも重要だけど、未熟なチームでは経験（者）依存となり、成熟したチームでは個人の経験を尊重しつつ、より「データドリブン」な組織運営を行っているのだ。 データドリブンとは組織運営を組織の持つデータに基づいて行なうという意味だ。

経験者依存の組織では人力に頼らざるを得ず労働集約型の力ずくの開発となりやすく、長時間労働を始めとした人間消耗型開発となってしまう。経験について、その有効性を否定している訳ではないが、経験は言葉を変えると個人的なノウハウとも言え、保有する個人においては非常に貴重かつ有効なものだ。しかし経験は属人的な暗黙知であるため、他人と共有することが極めて困難なものだ。また単なる勘や組織の空気に基づく行動は失敗の大きな原因となるものだ。個人の有用な経験やノウハウをチームが共有できる公開知にするためにはデータ化が必須なのだ。

「経験」は個人的なもので、「データ」はチームみんなのものになる。

プロジェクトの遂行にあたっては科学的思考に基づく行動が最も効率的かつ効果的に成果を生み出す。データは科学的アプローチの基本なのだ。データがなければ目標は見えない。 目標が見えなければ的確な行動も取れない。的確な行動がとれなければ成果を生み出すこともできない。データは数値データだけではなく文字・文書・図・画像等あらゆる表現手段も含むのだ。特に数値データは、情報におけるあいまいさや情緒性を排除できるので最も基本的なデータと言える。プロジェクト活動における基本的なデータはQCDおよび人・モノ・金に関するものになる。

【アジャイルの視点】

ソフトウェア開発における顧客価値はデータで表される。またデータはチーム内で共有されて初めて顧客価値を実現する力となり得る。アジャイルにおける四つの価値はすべて共有されたデータによって実現されるだろう。すなわち、「個人とその相互作用」もデータを介して実行され、「動作するソフトウェア」もデータを通して実現され、「顧客との協調」もデータを介して行われ、「変化への対応」もデータに基づいて実行されて初めて顧客価値の実現に成功することができる。

### （6） 丸投げ ［人に関する認識］

丸投げは関係者間の不信感を増長し組織力を弱めるものだということを知っておいたほうがよい。一括請負にて協力会社に仕事を依頼したので、もう自分には責任はなくなったと考えている人はいないだろうか。

仕事は分業あるいは他社との協業によって行われることが多い。自分の仕事はここまで、自社の責任はここまで、とみんな線を引きたがるものだ。多くの場合、その線は相手が想定している領域の外に引かれる。お互いの境界が大きく離れた状態を丸投げというのであろう。 この溝はその大きさに比例して必ず問題を起こす。相手にまかせたから自分は知らない、協力会社に一括請負で発注したから当社に問題はないという認識は誤りだ。

全く関係のない独立した仕事の場合には、関係ないとの主張も正当化できるように思えるが、そもそも独立した仕事においては両者の仕事上の相関関係は発生せず、その主張は正当化できない。

### ① 丸投げについての開発者たちの生の声

丸投げについての開発者たちの生の声をいくつか拾ってみる。

- 知っている担当者（協力会社）へ丸投げするほうが安く、納期も短縮できた。
- 端末開発は協力会社（もっと言うと人に）依存しすぎていたため、初期設計書以降の工程に関する資料は皆無。
- 協力会社依存度が高く社員が理解できていない。任せきりの発注。
- 社内開発者の協力会社依存の体質による仕様作成能力の低下、設計能力の低下、プログラミング能力の低下、およびレビュー能力の低下などが根底にあり、自分たちと協力会社（ﾌﾟﾛｸﾞﾗﾑ技術者）の役割、責任を明確に出来ない。これを繰り返しており、運用、仕様への意識・こだわりが無くなって来ている。開発者としての運用、仕様へのこだわり低下つまり無責任な仕事が品質低下の大きな要因だ。

なんと恥ずかしいことを言っているのだろうか。

### ② 何故丸投げが起きるのだろうか

では何故丸投げがおきるのか考えてみよう。

#### 分業化の進展

すいぶん昔の話だがソフトウェア開発という仕事が始まったころは自分の組織の中で開発を完結させるというスタイルが普通だった。当時のソフトウェアは現在に比べて規模も小さく内容も複雑ではなかった。顧客の要件も直接開発者が聞いてそのまま設計書にまとめることができた。もちろんプログラムも自分らの手で作成し評価テストも自分らで実行した。不明な仕様なんて存在しなかったし設計書のない開発などありはしなかった。開発者たちは自分で作ったソフトウェアの構造を熟知しておりその弱点もよく知っていたので評価テストも効率的かつ効果的に自分ら自身で実行したものだ。このような自組織完結型の開発は客先にも喜ばれ本当に楽しかった。

その後ソフトウェア開発のスタイルはどう変化したのか見てみよう。

いろんな製品においてマイクロコンピュータの採用はどんどんと進んでいき、元はハードウェアが担っていた機能はどんどんとソフトウェアが担っていくようになっていったのだ。その結果ソフトウェア開発の規模や複雑さは初期のころに比べて飛躍的に拡大されていきとうてい一組織内の開発では手に負えなくなってしまった。ここからソフトウェア開発の分業化が始まったのだ。

まず同じ会社内での分業化が進んでいき最初に開発業務と評価業務の二分業化が行われた。その後についてはみんなも知っているように要件定義、設計、製造、評価と現在のような分業化が進み各工程の協力会社による分業化も同時に進んでいったのだ。

#### 共有分業と分離分業

分業化がすすんでもそれぞれの分業組織間でのコミュニケーションが密に行われていれば大きな問題は発生しないだろう。第 3 章の1．や第 7 章の1．でも話をしたけど工程間・組織間のコミュニケーションギャップを克服しないまま分業だアウトソーシングだと血の通わない現実が分かっていない施策を強行していくと分業はいつのまにか共有分業から最悪の分離分業となってしまうのだ。何度もいうようだけど共有分業ってのは前工程の人ないしは組織が約束通りの成果物を次工程の人ないしは組織に確実に渡していくような本来の分業のあり方のことをいい、分離分業とは分業間の人ないしは組織がお互いにほとんどコミュニケーションも行わず明確な

約束ごとも取り交わさず、結果としていい加減な成果物を次工程に流すような最悪の分業の姿のことなのだ。
現在、日本のソフトウェア開発組織に限らず多くの開発組織はこの分離分業の状態にどっぷりと浸かっているのだ。品質問題、コスト問題、納期問題から解放されたかったらさっさと分離分業をやめ正しい共有分業の姿に戻せばいいのだ。

### ③ オフショア開発における丸投げ

オフショアにおける丸投げは国内におけるそれよりもはるかに深刻な結果を招くだろうし実際に招いてしまっている。国内の協力会社はいままでも人的な努力でコミュニケーション・ギャップ問題や劣悪な開発ドキュメント問題を補完してきたがもうとっくにその限界を越えてしまっている。国内協力会社における人的な補完は日本文化特有のものでありオフショア各国の組織においてはそのようなことなど全く期待できないのだ。
オフショア問題は国内における分離分業問題に加えて契約文化・意識・言語の差異を克服しなければならない問題を抱えたことになり、国内ですらコミュニケーション分断を克服できない人あるいは組織にはとうてい越えられない問題なのだ。

### ④ 業務委託が責任放棄にいたる悪魔のステップ

**step1** 仕事が忙しくて間に合わない。
**step2** 外部の業者から人を派遣してもらい、仕事を手伝ってもらう。
　このときはまだ仕事の責任は自分にあると思っているだろう。派遣者のミスが自分の指示の悪さに有ったとしたら自分が責任を負うのは当然だと思っている。
**step3** 派遣者で穴埋めしてきたが、仕事の忙しさが限度を越えてきたので、業務のまとまった部分を外部の業者に委託する。最初のうちは委託した業務についての進行状況や品質や問題点について委託先と綿密にコミュニケーションをとっており余り問題は発生しなかった。
**step4** さらに忙しさが増し複数の仕事を抱えざるを得ない状況になってきた。もう委託先の仕事の状況を見る余裕すらなくなってきた。委託業務の細かい指示もできなくなり、自分の専門領域の技術を学習する時間も無くなり、自分の主な仕事は多数の外注に仕事を割り振ることだけになってきた。
**step5** 外注から納入される成果物に多くの欠陥が発生するようになった。
**step6** 多数の欠陥品に対して自分では到底その責任を負い切れないので、委託先外注に対してその非を責め続け製品責任の転嫁をせざるを得ないようになってしまった。
**step7** step4～step6の悪循環がくりかえされると同時に、学習不足の結果は自分の専門的なスキルを劣化させ、外注へまともな指示すら出せなくなってしまった。
**step8** 回復不可能な大障害が発生してしまった。

この悪魔のサイクルはstep3で止めなければいけなかったのだ。悪魔のサイクルを止める役割・責任はマネージャにあり、仕事の分散だけではこの問題は解決できないだろう。仕事のプロセスの見直しや仕事のやり方の見直しも必要だろう。更に業務力の強化も必要だろう。多くのムリ・ムダの排除のためのリスク排除活動も必要だろう。人間の代わりにツールでできることはツールにやらせるべきである。業務はその価値の高い順に実行しなければならない。今やるべきことは直ちに実行しよう。今日やらなくてもいいことは明日にしよう。仕事に関係している仲間とは密なコミュニケーションを行おう。
担当者に直接的な責任はないが、そのひどい状況およびそれを放置していたらとんでもないことになるリスクについて諦めることなくマネジャーに発信し続ける義務はあるのだ。

### ⑤ 丸投げを防ぐには（共有分業のすすめ）

さてオフショアも含めてこれらの丸投げ問題を解決する方法について考えてみよう。

まずは分業組織間のコミュニケーションを復活させることだ。キーワードは共有分業の実現とコミュニケーションギャップの克服だ。

次に分業組織間の共有分業の重要な手段であるドキュメントによるコミュニケーション能力の向上が必要となる。

開発ドキュメントの最たるものは要件定義書と各種設計書だろう。明確に要件が定義された要件定義書がなくてどんな設計ができるのだろうか。どのように機能を実現していったらよいのかを漏れなく表現した設計書がなくてどんなプログラミングができるのだろうか。

結局、国内協力会社に設計工程を分業するにしろ、中国に製造工程を分業するにしろ、どれだけ品質の高いドキュメントでコミュニケーションができるかにかかっているのだ。

分業化・アウトソーシングを推進したい開発組織がまずやるべきことは、要件定義能力および設計書作成能力の向上とその人材の拡大なのだ。中国にどれだけの量の製造を委託できるかは、自分の組織が保有している要件定義能力人材の数・質および設計書作成能力人材の数・質のキャパシティに比例しているのだ。

自分の組織の中で要件定義能力保有者および設計書作成能力者をそれぞれA、B、Cランクに分類してみて、実際のところ国内・海外も合わせて半期あたりどれくらいの量の仕事を品質・コスト・納期を確保しながら達成できるのかを試算してみるといい。

試算された数値を50%も100%も越えるようなアウトソーシングは現在の自分の組織では無理を通り越して無謀なことだと自覚しよう。

いくら利益至上主義の経営者が製造工程をすべて中国にオフショアしろと命令しても自分の組織能力以上のアウトソーシングは顧客価値の最も高い品質を犠牲にし、結局不具合・障害対応でコストを浪費し顧客の信頼も失ってしまうのだ。身の程知らずのアウトソーシングは身を滅ぼすのだ。

丸投げを防ぐポイントは次のようだ。

**丸投げを防ぐポイント**

1. 分業組織間の会話および文書によるコミュニケーションの復活
2. 要件定義能力の向上とその人材の拡大
3. 設計書作成能力の向上とその人材の拡大
4. ドキュメント作成文化の習慣化と定着

同じプロジェクト内で、他の人あるいは協力会社にまかせたからとかといっても、自分の責任、特にマネジメントの領域における責任を免れることはできない。協力・協業とは共に力を合わせることであって、全部の負荷を相手に投げつけることではないはずだ。仕事はその工程・フェーズごとの重なり合いの中で実行される共助でしか目標は達成できない。

いわゆる米国発のアウトソーシングとは、発注元が仕事の中味を十分理解しており協力関係の中で発注先をちゃんとコントロールしているもののことであり、仕事の中味すら分かっておらずコントロールも放棄し利益だけを抜き去る丸投げとは全く別物であるとの理解が必要だ。会社において丸投げという職責は存在せず、丸投げするだけの人は会社にとって不要なのだ。

部下に丸投げ、外注に丸投げ、中国にオフショアの丸投げの組織においては実行責任の所在が不明になり、実際の現場がどの様な状態であるか誰にも分からず、更に興味すら持たないと

いう悲惨な状態に陥る。 現場に行かない、現場を知らないマネジメントに率いられた組織は崩壊するだろう。

【アジャイルの視点】
アジャイルの価値および原則が示すように、アジャイルチームは意欲のある仲間および顧客との信頼関係の中で協業する。およそ丸投げの世界とは反対の極にあり、人間的な行動であるといえるだろう。

**ティーブレイク**

**仕事考**

言っちゃなんだけど、不景気になったって忙しいんだ、ウチの工場は。売上げ？ 年商 6 億円っていってるけど、金のことは女房に任せっきりでオレはよく知らない。 ただね、職人が金のために働いたら、いいモノができるわけがない。1千万円で受注した品物を、なんとか600万円で仕上げて、もうけたいと考えたら終わりだよ。 1千万円で受注して、1200万円かかったって納得できる製品をつくる。赤字でも「さすが岡野さんだ」と認められれば、いずれ利益としてはね返ってくる。金を追っかけるから金が逃げる。仕事を追っかける。そうすれば後から金はついてくるよ。

本当はね、大きな会社ならウチが開発したような製品は、挑戦すればできるんだよ。なぜできないか。責任をとる人間がいないからだ。優等生ばっかしだから。 難しい仕事っていうのは失敗の連続だ。でもサラリーマンは自分が先走って結局できなかったら、月給少なくなったり、とばされたりさ、大変だろ。怖いから、やらないほうがいいもんな。違うかい？ オレは成功率が 6 割あれば挑戦する。たった 6 人の会社だ。「オレの責任だ、言った通りやってくれ」って言い切れる。

会社をもっと大きくしないのかって？ そんなことはこれっぽちも思わない。 6 人を 200 人にして苦労するより、楽しく働けて、職人に臨時ボーナスもだせて、ノルマもない。そういう東京・下町の町工場がいいんだ。

岡野工業「代表社員」 岡野雅行 ［朝日 2005］

## （7） 七つのチェックポイント

最後に、自分の組織あるいはプロジェクトの調子が悪いと感じた場合、下記7つの項目をチェックしてみよう。
似たような条件・環境の組織・プロジェクトにおいても、かたや全く成果を出せない場合と一方卓越した結果を出す場合とがある。だだ運が悪かったで終わらせるのではなくこれらの7つのチェックポイントを実践することで、個人のガンバリズムから脱し組織として戦える実践部隊の構築が急務なのだ。

**【プロジェクト・組織の健康度チェックリスト】 七つのチェックポイント**

**□ チェック1．明快な目標の設定とその共有**
目標・ビジョン・基本戦略の設定が明確で、かつ全員に共有されているか。
勘と度胸の精神論・気合いかけ声ではなくデータに基づいた明確な目標が現場に伝え続けられているか。

**□ チェック2．権限の委譲と責任の自覚**
適切な権限委譲による職場の活性化と責任（職責）の自覚・理解による目的意識・使命感の喚起が行われているか。権力を抱え込んでいないか。 任せることが信頼につながる。

**□ チェック3．現場主義に基づいた双方向コミュニケーション**
現場主義に基づいた双方向コミュニケーションが成立しているか。
・ 上司から部下へ適時わかりやすい指示と激励が行われているか。上司から部下へ丸投げになっていないか。マネジメントが現場に出向き実態把握と刺激と顧客の生の声を全従業員に伝え続けているか。
・ 部下から上司へタイムリーな報告と相談が行われているか。悪い情報が迅速に報告されているか。部下は指示待ちになっていないか。

**□ チェック4．マネジメントのリーダシップ**
権力主義、自己保身主義、形式主義、派閥等を廃し信頼感に基づいたリーダシップが発揮されているか。健全なリーダシップにより組織の一体感、問題意識の共有化、行動ベクトルの一致、全員参加が行われているか。

**□ チェック5．不適正問題の早期発見および対処**
Gap＆Fit。 目標達成において計画と成果の差異を常時チェックし速やかにそのギャップを小さくする活動を行っているか。問題点の隠蔽は組織崩壊の元。問題がオープンにされる風土は不正問題の未然防止につながる。

**□ チェック6．人材育成・能力開発の実施**
なりゆきまかせのOJTになっていないか。人材育成・能力開発・キャリアパスの計画があるか。有能人材の抜擢・登用を行っているか。 独自技術開発への取り組みを行っているか。

**□ チェック7．適切な評価と励みになる処遇が行われているか**

正当な評価・処遇は全ての活動のモチベーションに大きく影響する。私情を排して評価を行うにはApple to Apple比較の原則に基づいて極力あいまいさを排除しなければならない。

（＊注） Apple to Apple比較

いわゆるアップル・トゥ・アップル比較とは、ものごとを比較する場合、比較の対象どうしをそろえる、つまり同類のもの同士を比較しなければ正確な比較分析はできないということなのだ。たとえばリンゴの品種の優劣を比較したい場合に「紅玉」と「ゴールデンデリシャス」のそれぞれの特徴の優劣を比較することだ。リンゴの「紅玉」の優位性を証明するのにナシの「二十世紀」といくら比較しても「紅玉」の優位性の証明にはならないのだ。

こんなこと分かっていると思っているかもしれないけど結構みんなこの間違いを平気でやっている。企画部などがよくやる競合他社商品との比較などで、もともと比較対象にならない他社の微妙にワンランク下の製品と自社製品を比較して、自社製品が優れているなんて平気でみんなにプレゼンなどしている。もし意識的にやってるとしたらこれはインチキな背信行為なのだ。

## 6.　プロジェクトの見える化に役立つ主な管理表

プロジェクトの悪さ加減が最初に目立ちはじめるのはやはり結合テストや総合テストにおいての場合が多いと思わないか？ いつも思うけど何故もっと早く悪さ加減が目に見えてこないのだろうか。 いや本当のことを言うとプロジェクトのみんなは開発の前工程でこのプロジェクトはヤバイと言うことに気が付いているのだ。 みんな心の中では気づいているくせに公式の場ではただ”頑張ろう”と言って根拠のない楽観主義に逃げているだけなんだろう。
プロジェクトリーダーはそんな根拠のないガンバリズムの楽観主義に陥ってはいけないのだ。
プロジェクトの”見える化”はそんな楽観主義あるいは無責任主義を許さないためにあるのだ。

プロジェクトを健全に進めるために役に立つ色々な管理表があるが、実際のところみんなこれをやりたくないのだ。 これを実行すると言い訳とか敵前逃亡とかができなくなってしまい結構キツイものがあるのだ。
以上のことを承知した上で以下に説明するプロジェクト管理表を実行しようじゃないか。

### （1） プロセス管理表

プロジェクト管理の基本中の基本はプロセス管理だ。 プロセスは分かりやすく言うと仕事の段取りなのだ。 プロセス管理表はプロジェクトの全ての仕事を時系列順に並べた開発の手順のチェックシートなのだ。

各々の詳細な工程に対してやるべき作業内容、誰が実行するのか、作成される成果物は何かなどを記述し実施予定日・実施日や備考などを記録していき個々の作業が適切な内容で確実に実施されたかどうかを保証する証拠となるようにするのだ。下図にサンプルを示した。

プロセス管理表（開発手順チェックシート）

システム区分：　開発番号：　開発ランク：　システム名：

S/W Version：　初回作成日：xxxx/xx/xx　変更作成日：　ﾌﾟﾛｾｽ管理者：PL

| | NO | 手順 | 連絡 | ポイント | 作業 | 担当者 | 作成物 | Rev | 備考 | PL Check | 実施予定日 | 実施日 | レビュー結果 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 商品企画プロセス | 1 | 要求仕様書説明 | | | 要求仕様書説明 | SE・開発部 | 要求仕様書 | レビ | 要求仕様書は、打ち合わせ3日前に受領し、検討を行う。検討できない場合は、打合せは不可<br>要求仕様の確定時は、リーダーに連絡する。要求仕様の確定状況をヒヤリングする | | | | |
| | 2 | 見積もり依頼<br>見積もり回答 | | | 仕様、ボリューム、<br>要員検討<br>見積依頼 | PL、SPL | 見積回答書<br>リスク管理表作成<br>2003案物件見積DB<br>要求仕様書の修正1 | | 原則として土曜翌日内に回答する<br>メールでくる複数見積もり回答については、仕様の不明なものが多いため仕様不明のものは、回答しない回答できないことを営業に連絡する<br>リスク管理表は、社内ワークフロー物件対象 | | | | |
| | 3 | 要求仕様の明確化 | | | 要求事項の明確化 | PL | 要求事項の明確化チェックリスト作成 | | 要求仕様の明確化を行い、内容のチェックリストを作成する | | | | |
| | 4 | 注文書<br>事前着手指示書<br>受領 | | | 開発指示 | 営業部→開発部 | 注文書<br>2003案物件見積DB | | 開発指示の発行をSalesforce、着手時に発行されていない場合は、事前着手指示を受け付ける<br>開発指示は、デザインレビュー1までに必ず発行させる | | | | |
| | 5 | プロセスの選定、分析、改善 | | | 商品企画プロセスを実施してまとめ<br>所感<br>改善策の検討 | PL、SPL | 要求仕様DBの仕様準拠<br>レビュー報告 | | 次の工程への引継ぎと次のリスクの引継ぎ<br>・営業/SEの担当確認 | | | | |
| | 6 | 予算企画 | | | 予算の入力 | PL | 仕様かり表 | | 予算の入力を行う | | | | |
| | 7 | ランク申請 | | | ランク申請DB | 開発部→品質保証部 | ランク申請書<br>ランク決定書 | | | | | | |
| | 8 | 仕様説明 | | | 要求仕様書を基に仕様説明 | 開発部　→開発メンバー | 議事録 | | | | | | |
| | 9 | 構成管理 | | | 画面、プログラムの引き出し | | モジュール一覧<br>ドキュメント一覧 | | 仕様説明後、構成管理より必要な物を引き出し担当者に渡す。 | | | | |
| | 10 | 仕様検討<br>承認 | | | 上記の仕様打ち合わせ | SE<br>PL、開発メンバー | 議事録<br>要求仕様書の承認 | | 保証項目発生時：上へのTop<br>作業追加発生時：再度仕様検討 | | | | |

［図表10. プロセス管理表］

各工程が複数の会社にて分業化されたワンプロジェクト・マルチベンダーの場合は各工程の担当会社ごとに担当工程のプロセス管理表を作成・運用し、発注元の会社においては全体をまとめた統合プロセス管理表の作成・運用が必須だ。これを実施しない会社は最初から責任を放棄したものと考えていいだろう。

### （2） 日次フォローシート

日次会議における進捗実績、進捗予定および問題や課題の記録だ。 これを利用して本当の進捗を実感としてつかみ必要なフォローおよびヘルプを実行しよう。各自作成してみるといいだろう。

### （3） 週次フォローシート

週次の振返り会議における週間実績、次週予定および問題や課題の記録だ。 一週間の日次会議のまとめが週次会議なのだ。 この一週間の実績を振返って見て次の週の行動計画を立てるのだ。 週次フォローをキッチリと実行できているプロジェクトでは進捗が数ヶ月遅れとなる可能性は極めてまれだろう。 しかしながら振り返り会議はもとよりプロジェクト完了時の全体振り返り会議を実行しているプロジェクトは少ない。キミは実行するのか？それともしないのか？

### （4） リスク管理表

開発全工程にわたるリスクの管理表。 プロジェクトにおけるリスクとしては次のようなものがあるだろう。 仕様凍結遅れ、仕様変更、進捗状況、責任の所在、開発規模、他組織との関連、要員の確保、外注コントロール、短納期、プロセスの遵守、見積り精度、性能問題、保守性問題、プログラム構造、設計ミス、製造ミス、評価ミス、構成管理ミスなど。これらの情報を網羅したリスク管理表を作成・運用しよう。

| | | | | | | 影響 | | 影響度 | | | | | Weighted | | | | 予防対策後（予想） | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | 影響度 | | | | | |
| 通番 | リスクID | カテゴリ（タスク） | リスク（失敗モード） | 原因 | 発生確率 | ローカル影響 | エンド影響 | COCOMO 開発 | COCOMO コスト | COCOMO 品質 | 検出法 | 発生確率 | RPN | COCOMO 開発 | COCOMO コスト | COCOMO 品質 | 予防策 | 発生確率 | COCOMO 開発 | COCOMO コスト | COCOMO 品質 | 発生確率 | RPN | リスク対策発動したときの対処計画 | 備考 |
| 1 | 1-1 | マルチベンダー開発におけるリスク | 担当外の保守の責任を負わされる | 保守契約または他ベンダーとの合意がされていない | | 開発着手遅延 | コスト超過 | | | | 進捗レビュー | | | | | | 保守契約を交わす他ベンダーとの合意を得る | | | | | | | 他ベンダーと保守内容の文書化する |
| 2 | 1-2 | マルチベンダー開発におけるリスク | マルチベンダー環境でのテストができない | 他ベンダーの機器（モジュール）を借用しない／借用が遅れる | | 待ちの発生 | 納期遅れ | | | | テスト事前準備 | | | | | | 他ベンダーからの借用時期について合意をとる | | | | | | | 顧客と費用、納期について交渉する。 |
| 3 | 1-3 | マルチベンダー開発におけるリスク | マルチベンダー間のI／Fが決まらない。 | 他ベンダーのI／F検討遅れ | | 開発の仕戻り | 納期遅れ | | | | 仕様レビュー | | | | | | I／Fレビューを行う | | | | | | | I／Fレビューを行う |
| 4 | 2-1 | 仕工程共通 | 開発途中の仕様変更 | 顧客とのコミュニケーション不足 | 1 | 開発の仕戻り | コスト超過・納期遅れ | 2 | 1 | 1 | 仕様レビュー | 1 | 12 | 4 | 4 | 4 | ・顧客とコミュニケーションを良くとる<br>・要求仕様を文書化して顧客の署印をもらう | | | | | | | 顧客と費用、納期について交渉する。 |
| 5 | 2-2 | 仕工程共通 | 開発途中の仕様変更 | 仕様書が不明確 | 1 | 開発の仕戻り | コスト超過・納期遅れ | 1 | 1 | 1 | 仕様レビュー | 1 | 10 | 1 | 4 | 4 | テスト仕様書作成による不明確部分の洗出し | | | | | | | 顧客と費用、納期について交渉する。 |
| 6 | 2-3 | 仕工程共通 | 進捗状況が悪い | PJメンバのスキルが低い | 1 | 開発の遅延 | 品質低下 | 2 | 1 | 1 | メンバからの報告 | 1 | 16 | 8 | 4 | 4 | PJ開始前にメンバのスキルをチェックする | | | | | | | トレーニングの計画と実施 |

［図表11. リスク管理表］

### （5） 課題管理表

リスクが実際に問題化したものが課題なのだ。 日々発生してくる課題をその対策アクションとともに記録しよう。

課題管理のポイントは各々の課題に優先度をつけること、実行担当者名および期限を明記することだ。 自分なりの課題管理表を作ってみよう。

課題とリスクの違いがよく分かってない人が意外と多いのだが、リスクはまだ問題が表面化してはいないが表面化する危険性のある潜在的な課題のことだ。 リスクはそれが表面化する前にその芽を摘み取っておくような対策や行動が必要だ。 例えば仕様凍結が遅れそうな心配があるなら、こちらから先に仕様提案をするとか、仕様を決める責任部署に人員強化の依頼を出しておくとか、顧客との要求仕様決定合宿会議などをすればいいだろう。

一方課題は必ず解決されなければならない問題なのだ。 課題はキミを待ってはくれない。

期日までにその問題を解決しなければ、その問題はさらに大問題へと成長してしまいQCDに大きな傷を負わせプロジェクトを止めてしまう結果を招くかもしれない。

キミの毎日の仕事は課題の解決だと言っても言いすぎじゃない。 プロジェクトにおいてはリーダーから担当者にいたるまで実に多くの課題・問題を抱えているのだ。 何も問題を抱えていませんと発言する人は何も仕事をしていない人だと思った方がいい。 課題管理表なしではまっとうに仕事はやり遂げられないものだ。 キミはこれやっているよね？

課題管理表　更新年月日:xx/xx/xx

| 課題(件名) | 分類 | 優先度 | 担当 | 期限 | 発生日 | 完了日 | 依頼・指示者 | 課題内容 | 対応状況 |
|---|---|---|---|---|---|---|---|---|---|
| | 【プロジェクト管理問題】 | A | | | | | | | |
| | 【顧客・人材問題】 | B | | | | | | | |
| | 【物品問題】 | C | | | | | | | |
| | 【損益問題】 | A | | | | | | | |
| | 【Q:品質問題】 | B | | | | | | | |
| | 【C:コスト問題】 | C | | | | | | | |
| | 【D:スケジュール問題】 | A | | | | | | | |

［図表12. 課題管理表］

### （6） 仕様変更管理表

顧客あるいは仕様作成者と約束した仕様凍結日以降に提示された仕様変更あるいは追加仕様については仕様変更内容はもちろんのこと必要な開発工数なども記録しておく必要がある。

後日、スケジュール調整や追加開発費の請求などにおける重要な証拠資料なのだ。

これをまともにやっているプロジェクトは少ないのだ。キミはやれるよね？

［図表13．仕様変更管理表］

| No. | 仕様変更票番号 | 発生日 | 業務名 | 仕様変更の内容 | 区分 | グループ | 修正担当 | 影響度 | 難易度 | 対応予定 | 対応日 | バージョンページ | 開発工数 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | 仕様変更 | APL | | | B | | | | |
| 2 | | | | | 仕様変更 | APL | | | A | | | | |
| 3 | | | | | 仕様変更 | APL | | | A | | | | |
| 4 | | | | | 仕様変更 | APL | | | B | | | | |
| 5 | | | | | 画面変更 | APL | | | A | | | | |
| 6 | | | | | 仕様変更 | APL | | | A | | | | |

### （7） 進捗管理表／モジュール進捗管理表

モジュール進捗管理表はソフトウェアの部品レベルでの進捗管理表だ。 ソフトウェア開発における部品の単品管理なのだ。モジュール管理表のポイントは下記のとおりだ。

・管理するソフトウェアをexe、dll、ocxなどの最小部品レベルまでブレークダウンしよう。

・各部品のプログラムサイズの見込みおよび実績を記入しよう。

・責任所在をはっきりさせるために作成担当者名を明記しよう。

・各部品の開発進捗状況が見えるようにしよう。 今コーディング中か、単体評価中か、結合評価中か、それぞれの工程についての完了予定日と完了実績日を記録しよう。

モジュール進捗管理表は”見える化”の最たるものだ。ぜひやってみよう。

機能モジュール進捗管理表(サンプル)

| 業務グループ | プログラム名称 | EXE,DLL,OCX等名称 | 開発Vol(Kbyte) 見込 | 開発Vol(Kbyte) 実績 | 担当者 | 開発状況 | コーディング(C) 完了(予定日) | コーディング(C) 完了(実績日) | 単体評価(U) 完了(予定日) | 単体評価(U) 完了(実績日) | 結合評価(K) 完了(予定日) | 結合評価(K) 完了(実績日) | 不具合現件数 | 開発現状説明・処理日程 9月30日 | 開発現状説明・処理日程 10月10日 | 開発現状説明・処理日程 10月30日 | 開発現状説明・処理日程 備考 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | XXX受信プロセス | xxxxx.exe | 50 | 60 | A, B | C | 6/30 | 6/30 | 8/15 | 8/15 | 8/30 | 8/30 | | △△△機能 | | | △△△抜の機能処理 |
| | XXX送信プロセス | xxxxx.exe | 45 | 40 | A, B | C | 6/30 | 6/30 | 8/15 | 8/15 | 8/30 | 8/30 | | △△△機能 | | | □□□のパフォーマンス向上 |
| | 通信プロセス | □□.exe | 25 | 30 | A | U | 6/30 | 6/30 | 7/15 | 7/15 | 7/30 | 7/30 | | | | | |
| | リカバリー応答プロセス | □□RECV.exe | 35 | 30 | A, B | U | 6/30 | 6/30 | 7/15 | 7/15 | 7/30 | 7/30 | | | ○○○処理 | | スケジュールのスライド(3日)検討 |
| | △△受信プロセス | △△RECV.exe | 25 | 20 | A | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |
| | △△送受信プロセス | △△SEND.exe | 20 | 30 | A | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |
| | △△定期問合せ | △△INQ.exe | 30 | 10 | B | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |
| | FTPプロセス | □□FTP.exe | 20 | 30 | C | K | 5/30 | 5/30 | 6/15 | 6/15 | 6/30 | 6/30 | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 最小モジュール単位までブレークダウンする | 量的(数値)の見込み/実績を(行) | 担当者の明確化 | 工程別進捗を予定/実績日程にて行い、色別にでも直感いやすくする | 品質状況の把握を(行) | 現件処理日程を明確にする | 標準設定値による問題点のフォロー |

［図表14．モジュール進捗管理表］

### （8） 修正影響度表

修正影響度表はバージョンアップ開発時によく用いられる表だ。 今回追加あるいは修正したソフトウェアが他のソフトウェアのどの部分にどの程度の影響を与えるのかを図示したものだ。

例示したものは簡単なサンプルなので自分なりにもっと見える化を図った明細な影響度表を作ってみよう。

XXサーバ XX様 カスタム対応 修正影響度表

◎：対象(*新規/変更) △：関連

| 大分類 | 中分類 | No. | 小分類 | | システム | | 修正範囲 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 業務効目 | 修正概要 | XX | | GUI | 印字 | ｻｰﾋﾞｽ | DB構造 | ﾊﾟﾗﾒｰﾀ | その他 |
| 1.画面系 | 1．ﾒﾆｭｰ関連 | | | | | | | | | | | |
| | | 1 | 業務選択メニュー | | | | | | | | | |
| | | 2 | H/T運用メニュー | | | | | | | | | |
| | | 3 | POP/プライスカード運用 | | | | | | | | | |
| | | 4 | 精算日報業務 | | | | | | | | | |
| | | 5 | 棚卸業務 | | | | | | | | | |
| | | 6 | マスタ運用 | | | | | | | | | |
| | | | | | | | | | | | | |
| | 2．H/T運用 | | | | | | | | | | | |
| | | 1 | H/Tモニタリスト | 2．1次改修完成 | ◎ | | ◎ | ◎ | | ◎ | ◎ | |
| | | 2 | 返品伝票出力 | | | | | | | | | |
| | | 3 | 他店舗発注確認 | | | | | | | | | |
| | | 4 | 衣料売変リスト | 2．1次改修完成 | ◎ | | ◎ | ◎ | | ◎ | | |
| | | 5 | 衣料返品仕入伝票出力 | 2．1次改修完成 | ◎ | | ◎ | ◎ | | ◎ | | |
| | | | | | | | | | | | | |

［図表15．修正影響度表］

いわゆるスパゲッティプログラムでは修正影響度表を作るのは至難の業だけど、修正部分だけに絞り込めばなんとか作成できるだろう。キミにチャレンジしてほしい。

### （9） 評価チェックリスト

評価チェックリストの最も原始的なものは詳細仕様書を流用したもので、チェック該当部にアンダーラインおよびチェック番号を記入したものだ。

ある程度まではこれでも役に立つが仕様書が記述していない部分はそのまま評価チェックもれにつながりやすい。

完成度の高いチェックリストは通常操作の機能チェックのみならず異常操作におけるチェック、過去の類似不具合のチェック、システムの弱点チェックなどの項目を網羅した方がいいだろう。

システムの弱点箇所はたとえば、排他処理機能、非同期制御機能、タイミングずれ、リトライ処理機能、タイマー値動作、設定データ間の矛盾、メモリーリーク、データ長異常、伝文長異常、カウンター・ポインターなどのミニマム・マックス制御などの部分に多く潜んでいるものだ。

評価チェックリストの内容は各開発組織のノウハウのかたまりだ。 ノウハウは組織の価値そのものなのだ。 ノウハウは品質を上げ、開発者を成長させ、利益を生み出すのだ。

過去の失敗をチェックリストに取り込もう。

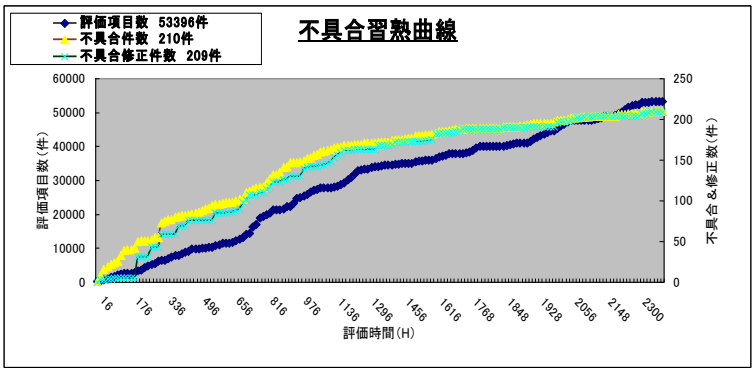
このリストは評価側の人間だけで作成するのは難しい。開発リーダーはチームの仲間としてこのリスト作成に協力する義務がある。

### （10） 不具合管理票

結合テストや総合テストおよび市場で発見された不具合を管理する一品一葉の票だ。 この票は不具合の発生状況、原因、対策方法などが記述されるが特に原因についてはどの工程に起因したものか、どのモジュールで発生したものか、不具合発生の要因および真因は何かについて記述されてなければ後でこの不具合に学ぶ手がかりを全く失うことになってしまい、何度も同じような不具合を再発させてしまうことになってしまう。

### （11） 不具合習熟管理表

縦軸にバグ発生数およびバグ修正数、横軸に評価時間をプロットした表で、不具合発見数のカーブが習熟する時期を見極めるために使用されるものだ。 評価の終盤でチェック件数を意図的に減らしたり、テスト自体を止めてしまえば習熟曲線はあたかも習熟したかのようなカーブになる。こんなインチキは厳罰なのだ。




［図表16. 不具合習熟管理表］

### （12） リリースノート

開発したソフトウェアを評価チームにリリースするときにリリースの内容についての注意や評価条件などを書いたものだ。 評価工程においては開発チームが評価チームにソフトウェアをリリースする際に使用され、まだ内在する不具合や機能テストの制限事項などを示したもので効率的な評価作業の実施には必須となる。ダメな開発チームはそのダメさ加減を知られたくないために意図的にこのリリースノートを渡さないことがある。

リリースノートは次工程である評価工程の人たちに対する最低限の礼儀なのだ。礼儀知らずの開発チームにならないようにしよう。

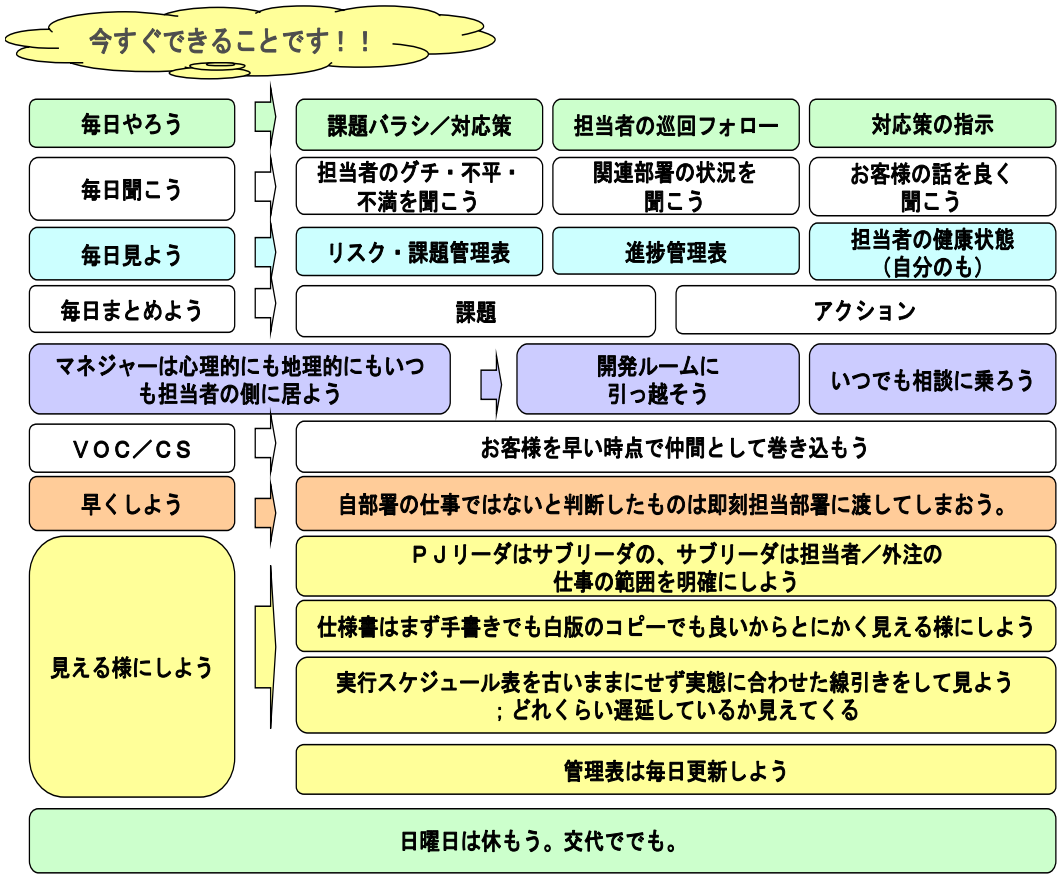
### （13） マンパワー管理表（人員配置・工数・コスト管理表）

ソフトウェア開発においてマンパワー管理の成否は製品の品質を左右するだけではなくその開発コストに大きな影響を与えてしまうのだ。 マンパワーの管理は週次および月次ごとの見直しにおいて増強・削減を適時繰り返し、業務の適正負荷の保持および予算の適正消化の把握を継続的に実行しよう。

## さいごに

これまでいろいろ話を続けてきたがプロジェクトリーダーに最後に言っておきたいことがある。
プロジェクトリーダーはプロジェクトの遂行にあたって”絶対に犠牲者は出さない”という強い信念をもっていてほしい。またその信念を貫徹するための強く賢い行動をとってほしい。
プロジェクトの成功指標としてよくQCDの3つがあげられるが”人間”のことを忘れてもらっちゃ困る。犠牲者だらけの死屍累々の焼け野原じゃダメなのだ。 ”夏草やつわものどもが夢の跡”じゃまずいのだ。
プロジェクトにおけるプライオリティは ①人 ②品質 ③利益 の順番なのだ。プライオリティは、それを失ったら困る順番で決まるものだ。あなたの会社ではこの順番はどうなっているのだろうか？
メンバー（人）の健全性を確保できたプロジェクトは結果として製品の品質が確保され利益を生み出すことは経験則で分かっていることだ。逆に多くの人的犠牲を伴ったプロジェクトで品質と利益を確保したケースは見たことがない。

犠牲者を出さずにプロジェクトを成功させるためには以下のことを毎日実行しよう。




［図表17. 毎日やろう］

**添付資料1．アジャイルソフトウェア開発宣言**

（引用　http://agilemanifesto.org/iso/ja/manifesto.html）

我々は、ソフトウェアの開発の実行により、そしてまた他の人々がそれを実行するのを援助することによりソフトウェア開発のよりよい方法を明らかにしていこうとしている。 この活動を通して、我々は次に示した価値に至った。

**（4つの価値）**

価値1．　プロセスやツールの価値も認めるが、個人とその相互作用を、
価値2．　包括的なドキュメンテーションの価値も認めるが、動作するソフトウェアを、
価値3．　契約交渉の価値も認めるが、顧客との協調を、
価値4．　計画に従うことの価値も認めるが、変化への対応を、
価値とする。左記の項目における価値を認めつつも、右記の項目に更なる価値を認めるものである。

**（12の原則）** 私たちは以下の原則に従う;

原則1．アジャイルの最優先事項は顧客を満足させることであり、これは価値あるソフトウェアの早期かつ継続した提供を通して実現される。
原則2．要求変更を歓迎する。たとえそれが開発の後期であったとしても。アジャイルは顧客の競争力を高めるために日々の変化に対応する。
原則3．動作するソフトウェアを数週間から数ヶ月の単位で、より短い時間軸にて繰り返しリリースする。
原則4．ビジネス側の人々と開発者たちはプロジェクトを通じて毎日共に作業しなければならない。
原則5．意欲のある人々でプロジェクトを構築する。彼らが必要とする環境とサポートを与え、そして仕事が完了するまで彼らを信頼する。
原則6．開発チームに対してあるいは開発チーム内において最も効率的かつ効果的な情報の伝達方法はフェイス・トゥ・フェイスの会話である。
原則7．動作するソフトウェアが、進捗を測る最も重要な尺度である。
原則8．アジャイルプロセスは持続的な開発を促進する。スポンサー、開発者、そしてユーザはたえず一定のペースが維持できるようにしなければならない。
原則9．技術的な優位性および良好な設計に対する継続的に払われる注意は俊敏さを増強する。
原則10．単純化、すなわち不要な仕事量の削減の最大化の技術こそが、アジャイルの本質である。
原則11．最良のアーキテクチャ・要求・設計は自己組織化チームから生まれる。
原則12．アジャイルチームは定期的に、より効果的な方法につき振り返りを行い、それに従ってその行動を変更し調節していく。

**添付資料2.　米国におけるアジャイル導入率調査結果**（和訳：筆者による）
（引用：http://www.ambysoft.com/surveys/agileFebruary2008.html）

調査結果についてすでに下記のように報告がなされている。 この調査は 2008 年 2 月に実施され、642件の回答が得られた。調査は the Dr. Dobb's Journal 編集者の Jon Erickson のブログにてアナウンスされたものである。

**調査結果**

調査結果の概要は Dr. Dobb's Journal の 2008 年 6 月号の" Has Agile Peaked?"に掲載されており、調査結果は下記の通りである：

1. 回答の69%は彼らの組織では 1 つあるいはそれ以上のアジャイルプロジェクトを実施していると回答している。15%のアジャイル未実施組織では1年以内に実施すると回答している。

2. 開発業者たちの61%は彼らの組織はアジャイルを実行していると思っており、そのマネジメントの78%も同様に思っている。明らかに開発業者たちは徐々に認めつつある。

3. アジャイル実行組織の82%はパイロットプロジェクト段階を過ぎている。

4. 回答者は圧倒的に、アジャイルチームが高品質製造であり、高生産性であり、利害関係者の非常な満足を得ていると回答している。

5. 同一場所チームのアジャイル成功率は82%であり、近接場所チームでは72%であり、分散場所チームでは60%であった。

6. アジャイルチームの84%は 4 週間あるいはそれ以下のイテレーション期間であり、2 週間のイテレーション期間が最も一般的である。

7. アジャイルチームにおけるコストは平均的に低いが、回答者の23%は平均的に高いコストがかかったと信じている。回答者の40%はコストは変わらなかったと答え、37%はコストは低いと答えた。

8. 同一場所アジャイルプロジェクトは、非同一場所プロジェクトに比べて成功率が高いし、それぞれの場合においてオフショアリングを含んでいるプロジェクトよりは含んでいない方が成功率は高い。

**参考文献・出典**


[Agile Scrum]　p39、p72
「アジャイルソフトウェア開発スクラム」、ケン・シュエイバー／マイク・ビードル
共著、ピアソン・エデュケーション社刊、2003 年

[CHAOS]　p54
プロジェクトの成功要因調査（米国）、2003/3/25、Latest Standish Group
CHAOS Report

[Drucker]　p73
プロフェッショナルの条件、ピーター・ドラッカー著、ﾀﾞｲﾔﾓﾝﾄﾞ社刊

[http://shippai.jst.go.jp/fkd/Search]　p55
失敗知識データベース、科学技術振興機構（JST）

[Kinoshita]　p26、p27
「ソフトウエアの法則」、木下恂著[1995]、中央公論社刊

[Takeuchi_Nonaka_1986]　p46
竹内弘高・野中郁次郎[1986]、「新たな新製品開発競争」
ﾊｰﾊﾞｰﾄﾞ・ﾋﾞｼﾞﾈｽﾚﾋﾞｭｰ 1・2 月号

[Shimizu]　p25
「派生開発を成功させるプロセス改善の技術と極意」、清水吉男著[2007]、
技術評論社刊

[Squse]　p59
【SQUSE 通信　Vol. 80】チームプレイの魅力　2009/6/17　Vol.80
http://www.squse.co.jp/info/log/eid153.html

[朝日 1998]　p67
朝日新聞社記事 1998/1/18　責任について

[朝日 2005]　p84
朝日新聞記事 2005 年　仕事考

[朝日 2008]　p68
朝日新聞記事　2008/4/16　世界は「見方」で出来ている

[朝日 2010.2.1]　p52
朝日新聞記事　2010/2/1　確実にキャッチされる球を

[朝日 2010.2.9]　p48
朝日新聞記事　2010/2/9　指示待ち　脱却なるか

[朝日 2010.7.5]　p59
朝日新聞記事　2010/7/5 オシムの目

[朝日 2010.7.26]　p75
朝日新聞記事　2010/7/26　金々節

[失敗の本質]　p10
「失敗の本質」、野中郁次郎共著[1984]、ﾀﾞｲﾔﾓﾝﾄﾞ社刊

[スウェーデン式アイデア・ブック]　p56、p57
フレドリック・ヘレーン著[2005]、ダイヤモンド社刊

[日経BP2009]　p64
日経BP社調査、2009 年 2 月 2 日、対象プロジェクト814件

[ピータ・グレン] p68、69、70
「それは私の担当ではありません」、ピータ・グレン著[1992]、ダイヤモンド社刊
[プレジデント 2004] p71
プレジデント誌記事 2004. 11. 1号、P50、御立尚資
[プレジデント 2007] p48
プレジデント紙記事 2007.6.4 P. 62 渋井真帆
[ワインバーグ] p75
「コンサルタントの道具箱 勇気と自信がもてる16の秘密」
G. M. ワインバーグ著、伊豆原 弓訳[2003]、日経BP社刊

**引用**

イラスト: いらすとや http://www.irasutoya.com/

**図表索引**



**「ティーブレイク」索引**

**索引**

**【さ】**