SEC BOOKS

実務に活かすIT化の原理原則17ヶ条

～プロジェクトを成功に導く超上流の勘どころ～

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

SEC BOOKS
実務に活かすIT化の原理原則17ヶ条
独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター 編

はじめに

「ユーザとベンダの想いは相反する」。

『超上流から攻めるIT化の原理原則17ヶ条』は、この条文から始まります。システム開発で発生する問題の多くは、ユーザとベンダをはじめとする様々な関係者が違った想いを持ち、開発が進むにつれて、はじめてそれに気付くことに起因しています。

『超上流から攻めるIT化の原理原則17ヶ条』(以下、『原理原則17ヶ条』)は、システム開発を成功させるために、要件定義フェーズのような超上流工程において、発注者と受注者が守るべき基本的な考え方と行動規範を格言のように短く表現したものです。

本書は、原理原則17ヶ条の理解を深め、より広く活用されることを目的に、以下を収録しています。

第1章　超上流から攻めるIT化の原理原則17ヶ条

　『超上流から攻めるIT化の原理原則17ヶ条』の本文です。

第2章　原理原則17ヶ条の理解を深める事例集

　原理原則17ヶ条の条文毎に、関連する事例の概要を紹介します。

第3章　失敗から学ぶ原理原則17ヶ条

　システム開発で失敗した事例を詳細に紹介し、どの場面で原理原則17ヶ条のどの条文を活用すればよかったのかを明らかにします。

第4章　成功を支える原理原則17ヶ条

　成功裏に終わったシステム開発の事例において、原理原則17ヶ条の精神がどのように活かされているかを紹介します。

第5章　実務に活かす原理原則17ヶ条

　原理原則17ヶ条を企業の社員研修やシステム開発の実務で活用している事例を紹介します。

第6章　実践！　原理原則17ヶ条

プロジェクト当初から原理原則17ヶ条を積極的に実践し成功につながった事例を紹介します。

参考資料　上流工程品質確保における発注者責任の実現

第6章で紹介した原理原則17ヶ条の適用事例を理解するために、プロジェクトの内容と取組みを参考につけています。

付録　原理原則17ヶ条と共通フレーム2007

原理原則17ヶ条を実務で活かすために、それぞれの条文に関連する『共通フレーム2007』のアクティビティやタスクを列挙するとともに、実施する際の留意点を一覧表にしたものです。

本書では情報システムの取得者と供給者について“ユーザ”と“ベンダ”、あるいは“発注者”と“受注者”のような表現を用いていますが、ユーザ内の業務部門と情報システム（情シ）部門（図0-1の青矢印）、あるいは一次ベンダと二次ベンダ（図0-1の緑矢印）のように読み替えることで、多くの場で応用できます。

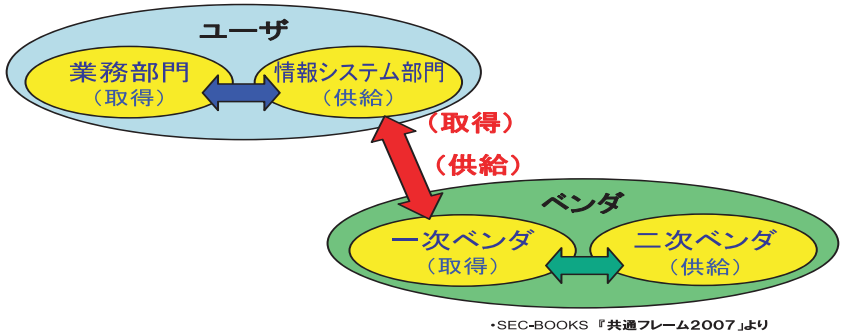


図0-1　取得プロセスと供給プロセスに関するプロセス呼出し関係

原理原則17ヶ条は、多くの企業の事例や専門家の知見をまとめた実務的・実践的な優れた指針です。本書が、システム開発を成功に導くための一助となれば幸いです。

＜プロセス共有化ＷＧ＞

目　次

第１章　超上流から攻めるIT化の原理原則17ヶ条

〈原理原則17ヶ条のねらい〉

1. 産業界の共通認識として
2. プロジェクトを成功に導く羅針盤として
3. IT化における定石として
4. ユーザ、ベンダの役割分担の規範として
5. いつでも立ち戻れる原点として

原理原則17ヶ条の説明

本17ヶ条は、開発プロセス共有化部会において検討してきた､「超上流」フェーズを発注側、受注側の双方がうまく進めるためのポイントをまとめたものです。「経営者が参画する要求品質の確保（第2版）」、「超上流から攻めるIT化の事例検索システム」と密接に関係しますが、本17ヶ条は、重要なポイントを短い言葉でまとめることにより、常に活用していただくことを主眼においています。

重要と考えられる17の原理原則について、基本的な考え方と行動規範にまとめています。

原理原則は､「超上流」において必要とされる事柄を、格言のように短く表現したものです。

基本的な考え方は、原理原則を理解しやすくするため、原理原則の基になる考え方を説明したものです。

行動規範は、原理原則に基づいて、受注者・発注者のそれぞれが具体的にどのように行動すべきかを示したものです。

本17ヶ条は、理解しやすさ、覚えやすさを重視して、重要なポイントを17に絞っていますが、活用する側の状況、特性などにより、17ヶ条以外に必要なポイント(18ヶ条以降)があり得ると考えています。そのようなポイントの追加などにより、それぞれのソフトウェア開発プロジェクトにおいて有用な形で活用されることを、作成者一同、望むものです。

原理原則17ヶ条の利用方法

本17ヶ条は、ソフトウェア開発の現場で広く利用してもらうことを目的としています。その利用（追加／削除などを含めて）は、原則自由とします。

ただし、以下の内容を守ることを前提とします。

(1) 17ヶ条は原文をそのまま使用する。

(2) 17ヶ条を使用・参照する場合出典を明記する。

・出典：IPA－SEC「超上流から攻めるIT化の原理原則17ヶ条」

(3) 17ヶ条の順序の変更は自由とする。

・ただし、使用・参照の出典を明記すると共に本来の順序について明示する。

・出典：IPA－SEC「超上流から攻めるIT化の原理原則17ヶ条（第n条）」

・削除などにより17ヶ条の一部ポイントを表記しない場合には、その旨を明示する。

・出典・変更の明記などは、脚注、欄外など、同一ページに明記する。

ただし、全体が17ヶ条関連である場合には巻末などに、まとめて明記してもよい。

原理原則17ヶ条

原理原則[１]　ユーザとベンダの想いは相反する

原理原則[２]　取り決めは合意と承認によって成り立つ

原理原則[３]　プロジェクトの成否を左右する要件確定の先送りは厳禁である

原理原則[4]　ステークホルダ間の合意を得ないまま、次工程に入らない

原理原則[５]　多段階の見積りは双方のリスクを低減する

原理原則[６]　システム化実現の費用はソフトウェア開発だけではない

原理原則[７]　ライフサイクルコストを重視する

原理原則[８]　システム化の方針・狙いの周知徹底が成功の鍵となる

原理原則[９]　要件定義は発注者の責任である

原理原則[10]　要件定義書はバイブルであり、事あらばここへ立ち返るもの

原理原則[11]　優れた要件定義書とはシステム開発を精緻にあらわしたもの

原理原則[12]　表現されない要件はシステムとして実現されない

原理原則[13]　数値化されない要件は人によって基準が異なる

原理原則[14]　「今と同じ」という要件定義はありえない

原理原則[15]　要件定義は「使える」業務システムを定義すること

原理原則[16]　機能要求は膨張する。コスト、納期が抑制する

原理原則[17]　要件定義は説明責任を伴う

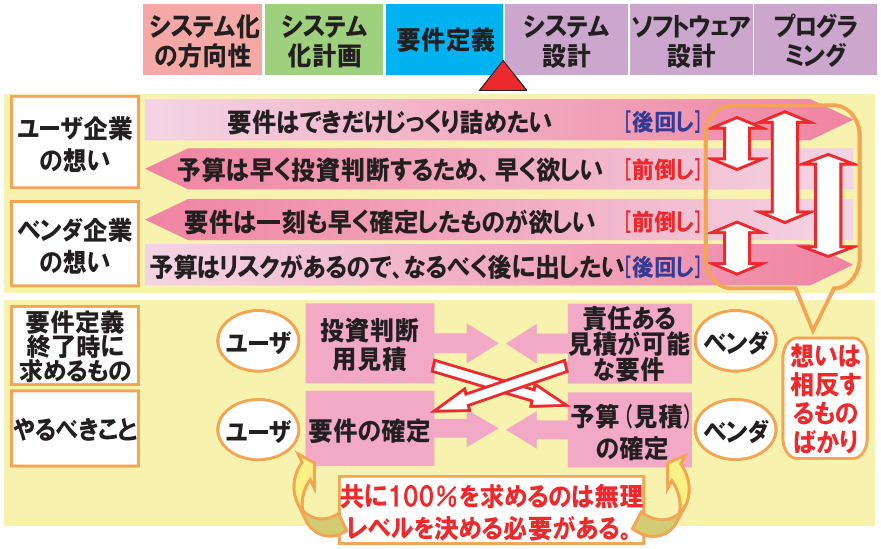
原理原則[1]

ユーザとベンダの想いは相反する

基本的な考え方

ITシステムの企画・開発の現場では、ユーザ企業とベンダ企業の相反する想いがあります。例えば、ユーザ企業は、要件はできるだけじっくり詰めたいし、予算は早期の投資判断を求められるので最終費用を早く確定してほしいとの想いがあります。他方のベンダ企業の想いはまったくその逆です。これがお互いにとってそもそもの不幸の始まりとなります。

「発注内容が固まらないうちに開発作業が開始される」といったことや、その結果としての「赤字案件の発生」といった問題もここに起因しています。



ユーザ企業・ベンダ企業の相反する想い

さらに、発注側業務部門も、システム開発のプロセス、システム開発のために必要な情報、例えば、プログラムの作成や変更には、コスト・期間・工数が掛かること、品質確保にも時間と労力が必要なことを知らなくてはなりません。

開発規模（工数）に見合った、最低限の工期を確保できなければ顧客満足を満たす開発はできません。受注者には開発規模に見合った工期を主張することが求められます。

行動規範

・発注者・受注者は、お互いの責任、義務、想いを知る。
・発注者は受注者との役割分担を明確にし、プロジェクトに積極的に参画する。
・発注側業務部門も、“システム開発”を理解する。

原理原則[2]
取り決めは合意と承認によって成り立つ

基本的な考え方

全ての取り決めは、承認ルールを明確にして、実施しなければなりません。誰が承認するか、どのように承認するかを明確にするとともに、承認の証跡を残すことです。

証拠のない口約束のように、決まったと了解していることが、それ以降の都合で無責任に変更となり、残念な思いをする、ということはよくあります。

決め事は可能な限り文章に残し、承認ルール（主体と方法）の確認をして、信頼度を高めなければいけません。

承認は合意に基づいていることが必要です。

この場合、受注者は、専門用語や業界用語の多用を避け、発注者が理解しやすく合意を得やすい提案を心がけるべきでしょう。

行動規範

・発注者・受注者は、合意プロセスと承認ルールを明確にし、それに基づいて行動する。
・受注者は、良否判断を仰ぎやすい提案を心がける。

原理原則[3]
プロジェクトの成否を左右する要件確定の先送りは厳禁である

基本的な考え方

要件定義はシステム開発全体の成否を左右する重要な工程です。曖昧な要件のまま開発が始まると、プロジェクトが失敗するリスクが大きくなります。

特に、システムの出来を左右する要件に高いリスクを抱えたまま、プロジェクトを進めることは危険です。あせってベンダに開発を依頼しても、先に進めず、かえって時間・コストがムダになることもあります。

解決の目処が立つまでは、先に進まない勇気も必要です。

行動規範

・発注者は、未確定要件の先送りは厳禁であり、現工程を延ばしてでも確定させる。
・受注者は、主要要件の実現の目処がたたないままプロジェクトを進めない。
・発注者・受注者は、未確定要件によるリスクを早期に低減する施策を打つ。

原理原則[4]
ステークホルダ間の合意を得ないまま、次工程に入らない

基本的な考え方

システム開発に係るステークホルダが増えてきています。

システムの対象となる範囲が広がった結果、システム開発に関係する部門も増え、利用者がマーケットに広がったことで稼働してからのサービス担当もステークホルダに加わりました。

また、企業間接続によるサービスの提供進展により、相手企業のシステム部門もまたステークホルダとして担当営業などとともにコミュニケーションの対象となりました。システムの運用をアウトソースしている場合は、アウトソーサもステークホルダに入るでしょう。

プロジェクトを起こした業務企画担当者は、プロジェクト責任者として、これらステークホルダの方針、意見、課題などについて、漏れなく綿密に把握し、できることとできないことをIT担当者、ベンダとともに切り分け、業務要件として取りまとめていく責任を果たす必要があります。

ステークホルダもまた、システムの供給側に立つ場合は、積極的にシステム開発要件の策定に参加し、利用者ニーズを確実に把握して、正確にシステム機能に反映していくことが必要です。したがって、ステークホルダの果たす役割は、きわめて重要なものになっています。

行動規範

・発注者は、ステークホルダの合意確認を自らの仕事と心得る。
・受注者は、合意を得ないまま開発に入ると、要件定義自体がひっくり返るおそれがあると心得る。
・受注者は、合意確認の作業支援はできるが請負（責任）はできないことを明示する。
・双方は、ステークホルダが誰か、漏れはないかを確認する。

原理原則[5]

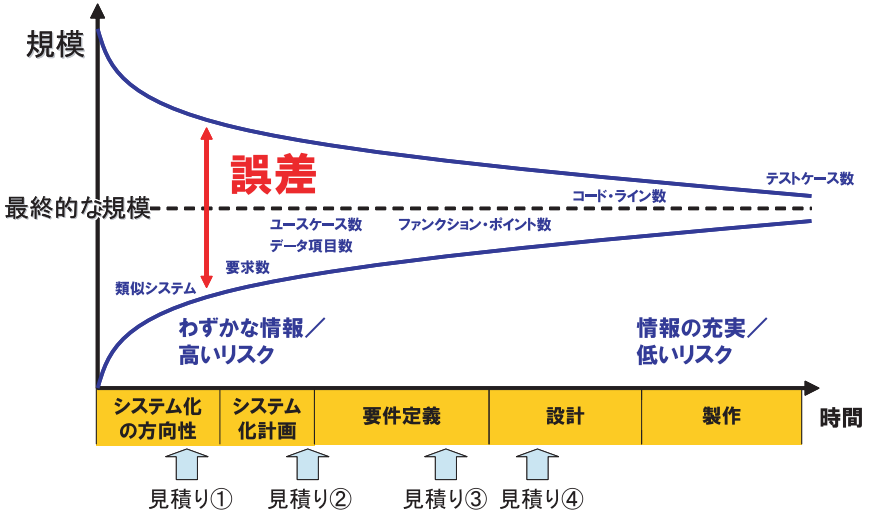
多段階の見積りは双方のリスクを低減する

基本的な考え方

開発プロセスのどの段階での要求仕様かによって、その固まり度合いや、見積り対象の深さなどに違いが出てきます。超上流での見積り内容は、仮試算、試算、概算レベルであり、システム設計に入って、確定となります。

にもかかわらず、あいまいさがある段階での見積りが最後まで開発側（情報システム部門、ベンダ）の束縛になってプロジェクト成功の阻害要因になっている現状があります。

不確定要素が多い中での見積りをプロジェクトの目標値として設定すべきではありません。



（注）文献：Barry Boehm 著の“Software Engineering Economics（Prentice-Hall社）”の図に基づきSEC作成

見積り時期とリスク

あいまいさがある段階の見積りを、はっきりした段階で見積り直せるルールづくりなどがプロジェクト成功の鍵となります。

要件の不確定さやプロジェクトの特性・リスクに応じて、適切な契約方式（多段階契約、インセンティブ付契約など）を選択することにより、発注者・受注者の双方にメリットが生まれます。

また、非機能要件に対する見積り技術が確立していないため、発注側が一方的に要求を提示しても、要件定義段階では、受注側で保証できないものもあります。

多段階とは、受注先をその都度変えるということではなく、固まり具合に応じて見積り精度をあげていこうということです。

行動規範

・発注者は、事業リスクを低減するためにも、多段階見積りを活用する。
・受注者は、見積りリスク回避のため多段階契約を活用する。
・受注者は、要件定義段階では非機能要件に保証できないものがあることを説明する。

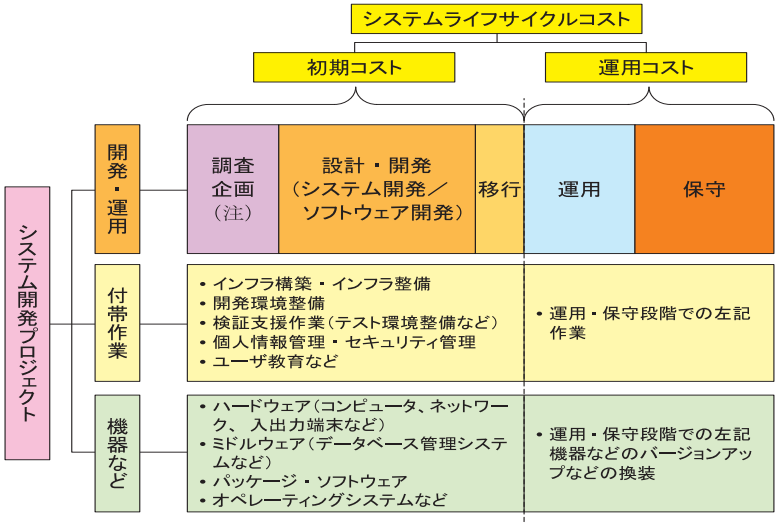
原理原則[6]
システム化実現の費用はソフトウェア開発だけではない

基本的な考え方

見積り範囲がソフトウェア開発のことだけを指しているのか、インフラ整備（システム基盤整備）などのような付帯作業も対象にしているかなど、スコープを明確にしていくことが大切です。

特に、教育や旧システムからの移行にかかる費用などは見落とさないようにしましょう。

また、連携する周辺システムとのインタフェースのための費用や、新システムの変更内容をユーザに周知してもらうための費用なども考慮しておく必要があります。



（注）システム化の方向性から要件定義

システム開発プロジェクトの構成要素

発注者は、何をお願いし、何を自分で行うのか、一方、受注者は自分の提供する作業やサービスはどの範囲なのかをお互いに明確にしておくことが重要です。

開発から運用・保守への引き継ぎをスムーズに行うことがシステムの安定稼働には重要です。そのため、要件定義の段階から、システム稼働後の運用・保守を見据えた計画・体制作りを行うことが必要となります。

行動規範

・発注者は、依頼する範囲、内容を漏れなく洗い出し、提示する。
・受注者は、見積りに含まれる内容と根拠を明確化する。
・発注者は、運用・保守も見据えた計画・体制を作る。

原理原則[7]
ライフサイクルコストを重視する

基本的な考え方

開発コスト､運用･保守コストのバランスを考えなければなりません。大切なことはライフサイクルコストを意識することです。

業務パッケージを採用する場合は、カスタマイズを前提とすることは避けましょう。カスタマイズ費用の予想外な増加を招いたり、パッケージのバージョンアップ時にそのまま適用できないなどの問題が発生する可能性があります。

プロセス主導、データ主導、オブジェクト指向には、それぞれに適した業務があります。対象業務と扱う情報を分析し、それに合った開発技法を適用すべきです。

ミドルウェアも含めた製品の採用にあたっては、実績、信頼性を十分評価し、保守性・運用性を高めることも考慮して、採用すべきです。

例えば、運用性・保守性を高めるポイントとして以下があります。

- メンテナンスフリー
- 拡張性の容易さ確保
- モニタリング・トレーサビリティの確保
- 障害発生時の調査、リカバリーが容易な設計
- OS・ハードウェアのバージョンアップ対応

行動規範

・発注者は、システムのライフサイクルにわたって投資対効果(ROI)を算定する。
・発注者は、業務パッケージを採用する場合は、カスタマイズを前提としない。
・受注者は、対象システムの特性をよく見きわめて開発技法・環境・ツールを適用する。
・受注者は、運用性・保守性を高める提案をする。

原理原則[8]
システム化の方針･狙いの周知徹底が成功の鍵となる

基本的な考え方

情報システムを入手しようと思う者は、その目的を明確にしなければなりません。さらに、システム化の方向性を示すとともに、関係者で共有しておくことが肝要です。

超上流のフェーズで、システム化の方針･狙いを浸透させておかないと、各人が勝手気ままに要件を考えるため、仕様の統一に時間がかかり、最初の構築だけでなく、その後の維持・保守においても費用と時間が増大することになります。

システム化の目的はコンピュータやプログラムではなく、事業目標を達成するための情報システムの構築なのです。

行動規範

・発注者は、情報システム構築の目的を明確にする。
・発注者は、情報システム構築の方針・狙いをステークホルダに周知徹底する。
・受注者は、方針・狙いを理解して、情報システムを構築する。

原理原則[9]
要件定義は発注者の責任である

基本的な考え方

要件定義とは、どのようなシステム、何ができるシステムを作りたいのかを定義することです。それはあくまでも発注者の仕事であり、発注者の責任で行うものです。要件定義があいまいであったり、検討不足のまま、受注者に開発を依頼した場合、その結果として、コスト増、納期遅れ、品質低下を発生させるおそれがあります。その責任を受注者に負わせることはできません。

要件定義作業は発注者の業務部門とIT部門が二人三脚で進めます。また発注者によっては、人的資源、経験、スキルなどの問題で、独自で実施できない場合もあります。このような場合、受注者をうまく活用し、不足しているシステム知識を補うことが有効であり、受注者に一部委託し、支援を受けることもあります。その上で受注者は発注者の側に立った支援を提供します。ただし、受注者が支援する場合であっても、要件定義で作成した成果物に対する責任は発注者にあります。

行動規範

・発注者は、「我々が要件を決め、責任を持つ」という意識を社内に浸透させる。
・発注者は、業務部門とIT部門が、二人三脚で要件定義を進める。
・発注者は、要件定義段階で受注者をうまく活用する。
・受注者は、発注者の側に立った支援を提供する。

原理原則[10]
要件定義書はバイブルであり、事あらばここへ立ち返るもの

基本的な考え方

ベンダ企業を含むステークホルダ間の合意のベースとなるのは常に要件定義書です。設計工程以降よりも、むしろ、要件定義の合意形成時点での吟味が重要です。「決定先送り型」の要件定義では、あいまいな海図に基づく航海のようなもので、早晩プロジェクトが破綻します。

ステークホルダ間の合意は、名目的な合意ではなく、実質的な合意であることが不可欠です。そのかわり、一旦きちんと決めれば、それに沿った運用をすればよく、最初に苦労するだけの価値はあります。

行動規範

・発注者は、安易に変更できない「重み」を認識して要件定義書を提示する。
・受注者は、安易に回避できない「責任」を認識して要件定義書を受託する。
・受注者・発注者とも、以降の変更はすべて要件定義書をベースとして議論する。

原理原則[11]
優れた要件定義書とはシステム開発を精緻にあらわしたもの

基本的な考え方

要件定義工程では、業務要件を整理・把握し、その実現のためのシステム機能要件をしっかり固めます。あわせて性能、信頼性、セキュリティ、移行・運用方法などの非機能要件、既存システム接続要件、プロジェクト特有の制約条件も洗い出します。また、将来の方針を見込んで稼働環境を定めることが大切です。流行に流されず、ルールを定めることです。

業務担当部門とIT 部門とが協力し合って、決めるべきことをきちんと決めることです。

それがシステム開発工程以降のコスト超過を最小限に抑えるとともに、開発工期の確約、要求品質の確保にもつながります。

結果として、システム開発の契約は基本設計、開発と多段階になるとしても、発注者としては、要件定義後にシステム総費用を把握し予算化するため、すべてを漏れなく洗い出す必要があります。

行動規範

・発注者は、機能要件、非機能要件などを漏れなく洗い出す。
・受注者は、特に非機能要件の定義で専門家としての支援をする。
・双方の協力で、システム開発の総費用を固める。

原理原則[12]
表現されない要件はシステムとして実現されない

基本的な考え方

この原則は、建築における施工主と工事業者の関係にあるように、発注と受注における常識です。しかし、情報システム開発においては往々にしてこの原則が成立しない場合があり、「行間を読め」、「言わなくても常識」、「言った言わない」など表現されない要件が、両者のトラブルの原因になります。

行動規範

・発注者は、文書・モックアップなどの手段を講じて、要件を表現しつくす努力をする。
・受注者は、行間を読むのではなく、きっちり確認をとって進める。

原理原則[13]

数値化されない要件は人によって基準が異なる

基本的な考え方

要件定義では、定量化できるものは、極力、数値化します。数えられないものは定義できません。「大きい、小さい、速い」だけでは、人によって「ものさし」が異なります。

例えば、障害発生時の復旧については、「すぐに」、「速やかに」といった表現は避け、想定障害を明記した上で「5秒以内に復旧」、とか「1分以内」、「翌日オンライン開始まで」ということを、双方が協力して定義します。

また、数値化されていても誤りはあります。例えば、使用する単位が違えば結果は大きく変わります。単位まで含めて確認し、決めなければなりません。

行動規範

・発注者・受注者は、協力して、定量化できる要件は極力数値化する。

原理原則[14]

「今と同じ」という要件定義はありえない

基本的な考え方

「今のシステムと同じでよい」という要件定義は、トラブルの元です。

「同じ」という言い方が正しく伝わるのは、具体的なプログラム、コード体系、テーブルなどそのとき存在する個別の形を持ったものについてです。実現機能レベルで同じと言う言葉を乱発しないようにしたいものです。

「今と同じ」でも要件定義は必要です。そもそも同じでよいなら再構築する必要はありません。よくないから再構築するというところから発想したいものです。

現行システムの調査をする場合は、システムの機能を洗い上げ、新システムの実像を明確にするだけでは不十分です。現行システムをどう使っているか、という点から調査をしなければなりません。例えば、データの再利用、アウトプットの二次加工、客先提供などの使われ方について調べて把握しないと、新システムの機能は不十分なものになってしまう可能性があります。

受注者は、発注者の「今と同じ」という要件を、そのまま受け入れてはいけません。

「そもそも今の要件はどうなっているのか」を問い直し、場合によっては具体的な要件にまで導くことも必要です。

行動規範

・発注者は、現行システムと同じ機能の実現であっても、要件定義を実施する。
・発注者は、既存機能だけを見て要件とするのではなく、使われ方まで十分調査し、要件とする。
・受注者は、「今と同じ」要件を具体的な要件まで問い直す。

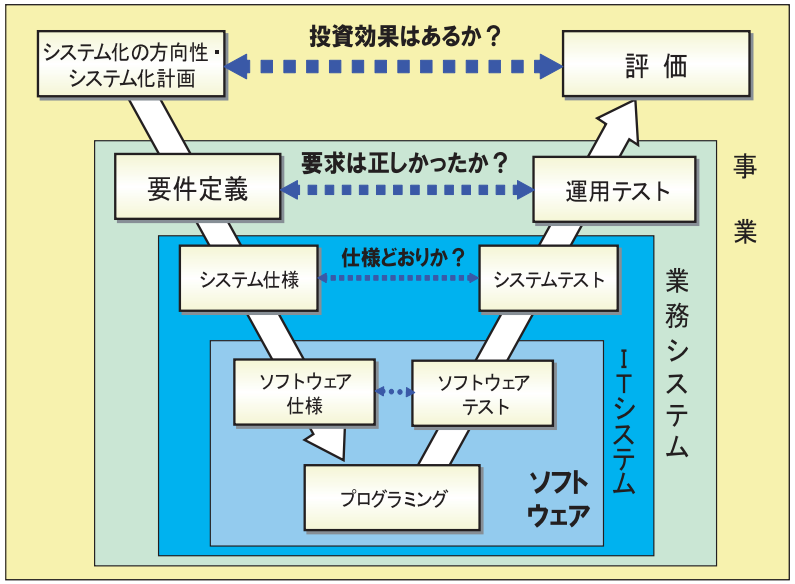
原理原則[15]
要件定義は「使える」業務システムを定義すること

基本的な考え方

要件定義は、業務にとって「使える」、「役に立つ」、「運用できる」システムを定義することです。

発注者は、それまでのやり方にとらわれることなく、むだな業務や非効率な手順を客観的に評価し、新業務をゼロベースで再設計することが大切です。ともすると、業務の複雑な部分を複雑なままシステムに置き換えようとするので、そうならないように注意しなければなりません。

また、ビジネスニーズからシステムの実現機能に落とし込んだ後、その



要件定義・仕様とテストの関係

機能が本来のビジネス要求を満たしているものか、立ち戻って検証することが重要です。IT化が自己目的化して、何のために実現したかったのかを見失うこともよくあります。これに運用テスト段階で気が付くのでは悲劇です。業務要件があいまいであると判明した場合には、常に業務部門と調整し、システムの合目的性を高いレベルに保つことが必要です。

そして、定義した新たな業務、新たなシステムが運用できるのかどうかの検証も重要となってきます。

要件定義の場に参加して、議論が横道にそれたり、枝葉末節に陥らないように助言するのは受注者の役割です。また、受注者は、要件として定義したものが、システム化計画で想定したコストや期間と比べて過剰なものや、逆にあまりに多くの費用を要さずとも実現可能な要件は勇気を持って変更を進言しなくてはなりません。

行動規範

・発注者は、常にビジネス要求の視点から、システム要件の妥当性を検証する。
・発注者は、シンプルな業務設計を心がける。
・発注者は、運用要件を要件定義の中で定義する。
・受注者は、オーバースペックを是正し、コストショートを進言する。

原理原則[16]

要求機能は膨張する。コスト、納期が抑制する

基本的な考え方

限られた「期間」と「コスト」の中で必要な「機能を実現」して、初めて評価されるということを忘れてはいけません。システムの開発においては、実現する機能とコストと納期のバランスが重要ですが、このバランスを保つのは非常に困難なことでもあります。プロジェクトメンバーはともするとシステム開発に没頭し、本来同時に達成すべきコストと納期をおろそかにしがちです。自分の家を建てるときには予算や引っ越しの時期との折り合いをつけるのに、システム開発では自分の懐が痛まないので、どうしても、機能＞コスト・納期の関係になりがちです。また、多くの場合、システム開発のコストは実現する機能ではなく、工数に比例しますから、どのくらいの作業が残っているのかをきちんと把握しながら、機能との折り合いをつけて作業を進める必要があります。このバランス感覚をプロジェクトメンバー全員が持っていなければ意味がありません。

新規ビジネスほど不確定要素が多く、ビジネス環境や事業戦略の大きな変化が予想されるため、システムも初期投資をおさえて、段階的に大きくしていくことを考えなければなりません。プロジェクトの背景や目的に応じたシステム化の範囲を検討し、「ついでにこの範囲も」という考え方は本来の目的を見失うので絶対に避けましょう。

要件の検討は、工程の進度に応じ、自由に発想する段階と、現実的にMUST要件に絞り込む段階を使い分ける必要があり、要件がある程度の粒度に詳細化された段階で、優先順位付け、コスト評価･リスク評価を行い、予算や期間などプロジェクトの制約の中で絞り込む必要があります。

納期が守れないと単にペナルティなどの経済的損失だけでは済まず、社会的信用すら失われることもあります。その場合は、段階的システム稼働の提案など、確実に実現できる機能に絞り込むことも必要となります。

行動規範

・発注者は、必要最低限のシステム構築からスタートする。
・発注者は、要求を抽出する段階と、要件として絞り込む段階を分ける。
・発注者は、要件の優先順位付けをする。
・受注者は、納期限界を超える開発量と判断したら、段階的稼働を提案する。

第２章　原理原則17ヶ条の理解を深める事例集

本章では、原理原則17ヶ条の趣旨を守らなかったために失敗した事例や、守って成功した事例を簡潔に紹介します。事例は、SEC BOOKS『ITプロジェクトの「見える化」上流工程編』及び『ITプロジェクトの「見える化」中流工程編』からの引用と、SECプロセス共有化WGで収集したものです。

これらの事例や解説を通じて、原理原則17ヶ条の条文レベルの理解から具体的な事例に結びつけた理解へと深めることができ、様々な場面での応用が可能になります。

原理原則17ヶ条が「リスクを未然に防ぐ、あるいは最小化するための考え方や方法」を示しているのに対して、『見える化』の本では「リスクを受容する判断（見切り）の仕方や、受容したリスクが顕在化した場合の対処の方法」を示していると言えます。『見える化』の本の事例集では、実際のプロジェクトを推進する上での実践的視点として、“本来の見切りの考え方”や“（リスクの顕在化として）とらえるべき兆候”など、一歩踏み込んだ提案も記載されています。

実案件に携わるプロジェクトマネージャ（PM）などの実務者は、原理原則を理解した上で、『見える化』の本を合わせて参照されると、よりリアルなプロジェクト運営に役立つと思います。

なお、本書では、原理原則17ヶ条に沿った視点で事例を説明するため、前述の『見える化』の事例を一部改変・省略しています。

原理原則[17]

要件定義は説明責任を伴う

基本的な考え方

システム開発における万全なる準備は、正確な要件という情報の次工程に向けての伝達です。自分が次工程に伝える必要のある情報について、要件確定責任だけでなく説明責任を負う必要があります。

システム開発の受託側から見た原則は「受託した要件として、書いてあるものは実現させる。書かれていないものは作らない。」ことです。システムは決めたとおりに作られ、決めたことに理解の誤りがない限り、正しい結果を生み出します。もちろん、プロジェクトのスタート地点で、すべてを誤りなく責任をもって確定することはできません。決め事も、それに基づいてのシステム構築も、人間である限り、見込み違い、思い込み、決め付け、聞き違い、聞き漏れはなくなりません。システム構築は、そういったことにより発生した「誤り、漏れ」を解消していく過程ともいえます。

「要件の行間を読め」ということを要求してはいけません。基本的には当たりまえの前提や例外処理であっても漏れなく伝達する必要があります。

また、同じ言葉を聞いても頭に浮かぶものが異なるのが人間です。発注者、受注者双方が説明責任を果たすことが、多様化した要求と、複雑化したシステム開発において品質を確保する重要なポイントとなることは間違いありません。

行動規範

・発注者は、受注者に要件を正しく説明する。
・受注者は、要件を理解して、理解した内容を発注者に確認する。

表2-1　事例の出典一覧

原理原則	本書上の事例ID	出　　所
[1]	1-1	『ITプロジェクトの「見える化」上流工程編』　付録3の事例3
	1-2	プロセス共有化WGで収集した事例
	1-3	プロセス共有化編で収集した事例
[2]	2-1	『ITプロジェクトの「見える化」上流工程編』　付録3の事例1
	2-2	『ITプロジェクトの「見える化」上流工程編』　付録3の事例8
[3]	3-1	プロセス共有化WGで収集した事例
[4]	4-1	『ITプロジェクトの「見える化」上流工程編』　付録3の事例2
	4-2	プロセス共有化WGで収集した事例
[5]	5-1	プロセス共有化WGで収集した事例
[6]	6-1	プロセス共有化WGで収集した事例
[7]	7-1	プロセス共有化WGで収集した事例
[8]	8-1	プロセス共有化WGで収集した事例
[9]	9-1	『ITプロジェクトの「見える化」上流工程編』　付録3の事例49
	9-2	プロセス共有化WGで収集した事例
[10]	10-1	プロセス共有化WGで収集した事例
	10-2	プロセス共有化WG収集した事例
[11]	11-1	『ITプロジェクトの「見える化」中流工程編』　付録3の事例36
[12]	12-1	『ITプロジェクトの「見える化」上流工程編』　付録3の事例16
[13]	13-1	プロセス共有化WGで収集した事例
[14]	14-1	プロセス共有化WGで収集した事例
	14-2	プロセス共有化WGで収集した事例
[15]	15-1	プロセス共有化WGで収集した事例
[16]	16-1	プロセス共有化WGで収集した事例
[17]	17-1	プロセス共有化WGで収集した事例

原理原則[1]
ユーザとベンダの想いは相反する

ポイント

システムを発注するユーザの想いとシステムを構築するベンダの想いは異なることが多い。双方の認識の違いを共有した上で、システム化の範囲や要件、開発実施計画について合意しておく必要がある。

事例1-1は、納期が確定している中でも、新しいサービス要件や仕様は慎重に詰めたいという発注者の想いと、納期を守るために開発時間を確保したいという受注者の想いとが異なっていたために起きたトラブル事例である。

事例1-2は、タイトなスケジュールの中で、その限られた時間をなるべく業務のイメージ固めや要件定義に使いたいユーザと、開発やテストに早く着手したいベンダの想いが相反することによって生じたトラブル事例である。

事例1-1、1-2とも、発注者と受注者の双方の想いが異なることを踏まえ、双方の事情を加味・理解した上で、QCDの優先順位とゴールの共有化を図ることで、想いの違いを吸収することが必要であった。

事例1-3は、できるだけ開発コストを抑制したい発注者と、できるだけ要件追加リスクを避けたい受注者の想いの違いが招いたトラブル事例である。受注者は、顧客の予算に見合った現実的な外部設計仕様を提案する必要があり、発注者もそれに歩み寄る努力が必要であった。

関連する「見える化」事例（上流工程編）

1-1 新サービスと非合理的な納期

金融機関で創立○○年の記念日に新サービスを開始することが最優先課題だった。新サービスについてはマスコミへの発表もされていたため納期厳守が絶対であった。ところが、新サービスということで実現方式も二転三転し、手戻りが発生。カットオーバー前に突貫作業でとりあえず動くレベルの物を納入。このため、稼働後は突貫作業の影響で品質の問題が多発した。

失敗の要因

新しいサービスということで、実現方式がなかなか決まらなかった。そのため、非機能要件やセキュリティポリシーについてもなかなか決まらず、方式設計が曖昧なままであった。納期まで時間がないこともあって、開発に進めそうなところから着手していかざるを得なかった。
上流工程の段階で要件が確定しない場合は、最悪のシナリオを考えて、実現しなければならない最低限の機能と品質について早めに合意を取るなどの対応が必要であった。

関連するプロセス共有化WG事例

1-2 要件定義未完のまま次工程に進み、工期遅延、品質不良多発

流通系の会社で、バックオフィス系システムの全面入替を希望し年度内稼動が必須であったため、タイトではあったが、ベンダと合意したスケジュールを組んだ。要件定義の途中で、ベンダから開発期間が厳しいので早く開発に着手したいと聞いて、未確定要件はあったが、次の設計工程の冒頭で早期に決めることとして、既定要件分の開発に踏み切った。しかし、未確定要件の定義は目論見より時間がかかり、先行して設計に入っていた既定の要件にも影響し、設計工程は遅延した。そのため、すべての後続工程が遅れ、稼動直前までプロジェクトは連日連夜の残業で遅れを取り戻そうとしたが、テストが不十分なまま稼動せざるを得ず、フォローに走り回ることとなった。

失敗の要因

納期など誰から見てもわかりやすいものは、ユーザ・ベンダに認識の違いはない。しかし、その限られた時間をなるべく業務のイメージ固めや要件定義に使いたいユーザと、開発やテストに使いたいベンダとでは、双方の想いが反する。したがって、明確にゴールを決めて時間を区切らないとズレが生じかねない。相手の認識も同じだろうと、要件未確定のまま、要件定義工程を切り上げてしまったことが問題である。

関連するプロセス共有化WG事例

1-3 要件定義の完成度が低く、システム化を断念

あるユーザ企業では、要件定義終了時を重要なマイルストーンととらえ、次工程への移行可否をステアリングコミッティで諮る社内ルールになっている。ベンダからの見積りを受け、社内的には、システム化の予算がここで一旦確定する。しかし、その後の作業で追加要件が多く生じたことから、外部設計完了時にベンダから提示された開発以降の見積り額は、要件変更のリスクを非常に高く積んだ結果、要件定義時と大きな乖離が生じ、ステアリングコミッティ開催時のシステム化の費用対効果の想定が崩れ、投資そのものの意義が薄れて開発を断念してしまった。

失敗の要因

重要な経営判断を下すタイミングで、正しい判断をするのに必要な情報が揃っていなかった。これは、要件も見積りも詰めが不十分なまま、要件定義を終えてしまったことに起因している。

原理原則[2]

取り決めは合意と承認によって成り立つ

ポイント

システム開発において問題となりやすいことは、曖昧な合意のままに行われる作業の先行着手である。発注者と受注者の双方が承認ルールを定め、これから行われる作業を合意しルールに従って承認することが極めて重要である。取り決めが必要となるのは、実現するシステムの範囲や内容、開発の実施計画・体制など、システムの品質、コスト、納期に係る事項である。合意と承認後は、取り決めに従って双方が行動する。取り決めた内容に変更を加えたい場合は、新たな合意と承認を目指して「契約の変更管理プロセス」(*1) に従い行動する。

事例2-1は、顧客と契約せずに開発に着手したため、顧客側の職制が変更になった（前任者は退職）ことにより、今までの話がご破算になったトラブル事例である。

費用と作業内容については最低限、文書で取り交わしておく等、作業の妥当性を顧客側の新担当者に説得できるようにしておく必要があった。

事例2-2は、提出した基本設計書の承認を得ずに詳細設計に着手した後に、顧客から基本設計の変更を要求され、詳細設計をやり直すことを余儀なくされために要員を追加することになったトラブル事例である。

仕様変更に柔軟なシステム設計を工夫するとともに、基本設計書の顧客承認と仕様変更ルールの合意をしておくことが望ましい。

（*1）共通フレーム2007　第3部　1.3参照

関連する「見える化」事例（上流工程編）

2-1 顧客側担当者が異動になり口約束事が反故に！

顧客側の課長レベルの担当者と費用及び作業内容について内々に話をつけ仕事を進めた。顧客側の職制が変更になったが（前任者は退職）、このことについての引き継ぎが全くなかったため、今までの話がご破算になった。

失敗の要因

顧客との契約を締結せずに開発に着手した。
費用と作業内容については最低限、文書で取り交わしておく。契約締結前に作業を実施するに至った経緯についても記録しておき、たとえ後付けでも作業の妥当性を説明できるようにしておく必要がある。

関連する「見える化」事例（上流工程編）

2-2 基本設計の承認なしで詳細設計に着手

顧客がビルを移転するとともに古いシステムを廃棄することになっていたため、新ビルでの新システム稼働が絶対条件であり、なんとしてでもサービスインスケジュールに間に合わせる必要があった。要件定義工程が大幅に遅れたため、基本設計を顧客に提示したが、承認を得ないまま、詳細設計まで進めた。途中、顧客から基本設計の変更を要請され、詳細設計をやり直すという二度手間になり、遅れたスケジュールを取り戻すため、要員を追加することとなってしまった。

失敗の要因

要件定義に時間がかかったが、その分、システムの実装の姿が見えていたつもりだった。顧客側もだいぶ焦っており、もともと承認されるまでに時間のかかる顧客だったので、提出した基本設計書の承認を得ずに詳細設計に着手した。
仕様の固まり具合を見定めて、システム単位、機能単位、データ単位などのいくつかの視点で、仕様変更に柔軟なシステム設計をする工夫をする必要がある。その見極めを行い、仕様変更をして欲しくないところを説明できるようにしておくこと。それでも仕様変更が必要な場合は、契約の変更管理を実施することが大切である。

原理原則[3]
プロジェクトの成否を左右する要件確定の先送りは厳禁である

ポイント

要件定義を曖昧なまま先送りすることはシステム開発において非常に大きなリスクとなる。リスクが顕在化した場合のインパクトは工程が進むほど増大し、発注者・受注者の許容範囲を超えることすらある。万が一、要件定義において未確定部分があるときは、どこが未確定部分かを双方が認識し、未確定であるためのリスク（コスト・期間）についてしっかり合意しておく必要がある。

事例3-1は、顧客の意思決定の責任が分散しており、しかも全体をまとめるキーパーソンが決まっていないため、要件提示が細切れでかつ曖昧となり、システムの出来を左右する要件の確定を先送りにしたまま設計に入ってしまった。そのため、手戻り多発、品質不良、工期遅延に陥ってしまったトラブル事例である。

関連するプロセス共有化WG事例

3-1　業務に精通した要員がアサインできず要件が曖昧のまま次工程に突入

A社の全社BPRプロジェクトの一環として、パッケージを基本とし、パッケージがカバーしていない機能をスクラッチ開発で実現する案件。詳細設計フェーズ終了時点でも、インタフェース機能や帳票の詳細仕様は未確定のままとなっていた。何とか未確定部分の仕様を決めて、結合テストまでこぎつけたが、以降のテストで不具合が多発し、スケジュールに追われ、低品質のものをリリースし、障害対応に多くの精力を費やすという負のループから抜けられず、大幅な工期遅延と費用増大を招いた。

失敗の要因

当該案件に必要なスキル、特に顧客側の業務要件に精通した人的リソースが不足していた。また、ベンダ側のプロジェクトマネージャは、掛け持ちしていた他案件のトラブル対応で負荷が増大した。このことが影響し、詳細な詰めは基本設計フェーズで行えばよいとの判断から、要件の詰めが甘いまま次の工程へ進んでいった。基本設計に入ってからも数少ないキーパーソンへの負荷集中を招き、機能横断的なコーディネート、設計不備等のチェック、要件拡大に対する適切な対応がタイムリーにできなかった。
結果として、上流工程の要件、仕様の詰めが甘かったことのツケが下流工程に影響し、不採算プロジェクトへの道を歩む結果となってしまった。

原理原則[4]

ステークホルダ間の合意を得ないまま、次工程に入らない

ポイント

ステークホルダ間の合意を得ないまま次工程に進んでしまうと、後工程での手戻り発生リスクが大きくなり、工期遅延、予算オーバーといった大きなトラブルに発展しかねない。特にステークホルダ（経営層、関連部門等）を十分に把握し、要件・仕様等についてシステム開発の早い段階で合意を得ておくことは発注者の責任である。また受注者も、発注者のステークホルダ間の合意形成に向け最大限の努力をすべきである。

事例4-1は、要件検討の場にユーザ部門の参画がなく、情報システム部門主導で仕様が決められていった。エンドユーザの合意・承認を得ないまま開発を行ったことにより、ユーザ部門の利用段階において、操作性や仕様に大きな問題が発見され、大幅な手戻りが発生し、予定どおりの本番稼動ができなかったトラブル事例である。

工程（マイルストーン）と先行着手のリスクについて、受注者は発注者と事前に協議し、リスク発生時の責任分担・役割分担について合意を得ておくとともに、プロジェクトの進捗に対応した体制をとってもらう必要があった。

特に上流工程での要件決定に際しては、情報システム部門だけでなくユーザ部門からも承認を得ることが必要である。

事例4-2は、要件定義段階でシステムの基本事項について情報部門と制度所管部門の部門長同士で合意し、さらに経営レベルの決裁を受けて確定させ、システム開発を成功させたものである。意思統一が不調の場合、エスカレーションの手続きを踏むことが組織的合意形成として有効である。

関連する「見える化」事例（上流工程編）

4-1 本当のユーザを見誤ってしまう

情報システム部門からの要請によるシステム構築であり、情報システム部門と契約し、開発を進めていた。発注者である情報システム部門主導でのシステム仕様作成を進めていた。いざ実装してユーザ部門での試使用が始まると、操作性や仕様に関しての問題が相次ぎ、仕様変更対応に追われ、予定どおりの本番稼働ができなくなった。

失敗の要因

ユーザ部門からの旧システムに対する問題提示や新システムへの要求などが上がっていたが、あまり議題に取り上げることなく情報システム部門主導で仕様決めを行った。本来はユーザ部門からも仕様検討者が参加するべきではあるが、顧客企業の情報システム部門ということで、特に仕様に関しては問題ないだろうと思っていた。
一般的にエンドユーザであるユーザ部門から「これでは業務が回らない」と新システムに対する文句が出ると、情報システム部門は対応せざるを得ないことが多い。したがって上流工程においては、仕様の決定に際し、情報システム部門だけでなくユーザ部門からも承認を得るようにする必要がある。

関連するプロセス共有化WG事例

4-2 基本事項を経営レベルで確定

自治体の総務事務システム構築プロジェクト。旅費請求や出退勤情報を職員が発生源で入力するシステムを導入すると同時に、部門に配属されている庶務担当者を組織共通の総務事務センターに集約した。プロジェクトを進めるに当たっては、事務フローだけでなく手続きの制度も改正する必要があった。要件定義段階でこれらを確定し、システム開発を開始した後は、大きな変更もなくサービスインすることができた。

成功の要因

プロジェクトチームを情報部門に設置し、制度を所管する人事部門と密に連携しながら、事務フローと制度を検討した。担当レベルで意見が一致しなかった事項は、双方の最終決定者臨席の会議で確定させ、結果を最終責任者に報告して決裁を受けた。システムの機能に大きな影響を及ぼす事務フローや制度といった基本事項がシステム開発前に確定していたため、開発段階でベースラインがぶれることがなかった。

原理原則[5]

多段階の見積りは双方のリスクを低減する

ポイント

見積りの精度は要件や仕様が詳細であるかどうかに大きく左右されるため、初期段階での見積りは多くの場合、ある程度のブレが発生しうる概算額とせざるを得ない。しかし、ユーザとベンダが早い段階で概算見積りに合意しておくことにより、契約段階の見積りが想定と大きく異なることを防ぐことができる。これは､ユーザにとっては予算の上振れリスクの低減、ベンダにとっては受注額の下振れリスクの低減になる。また、要件や仕様が曖昧な段階での概算見積りを、それらがはっきりした段階で見積りをやり直すルールについても双方で合意しておくことが重要である。

事例5-1は、基本設計以降を多段階で契約し、プロジェクトリスクを低減している成功事例である。

関連するプロセス共有化WG事例

5 - 1　要件定義の明確化と基本設計以降の多段階契約

建設業Ａ社では、発注者の要件定義工程の成果物である提案書（RFP）に、機能要件に加え非機能要件も精緻に記述し、特に構築システムに付随する移行要件、展開要件、システムテスト、運用テスト要件等も盛り込んでいる。このため、RFPに対するベンダ提案書の品質及びプロジェクト全体費用に関する概算見積りの精度が格段に向上した。
この概算見積りでプロジェクト着手の稟議を取り、プロジェクトマネージャはプロジェクト総予算を抑えながら、基本設計、開発、テスト、移行支援等を多段階で契約している。

成功の要因

要件定義の内容が発注者側で十分に検討され、総費用の網羅性、精度が向上し、さらに基本設計以降も多段階契約を実施しているため、各工程（各契約）でのインプットとアウトプットも明確になり、双方のリスクが低減されるようになった。最終的にはプロジェクト全体を通して、QCDの管理レベルの向上が実現できている。

原理原則[6]

システム化実現の費用はソフトウェア開発だけではない

ポイント

システム化実現のための費用は、ソフトウェアの開発のみならず、ハードウェア取得、データ移行、教育・研修など多岐にわたる。また、開発対象のシステムだけでなく、連携先のシステムの改造や接続テストなどの費用も必要となる。

「使えるシステム」とするために、ソフトウェア開発以外に発生する作業や満たすべきパフォーマンスを定め、予算や開発コスト・期間を計画することが重要である。

事例6-1は、要件定義後の投資額の抜け・漏れを防止するためのセルフチェックリストを整備し、後工程での予算追加申請件数を減少させる施策である。

関連するプロセス共有化WG事例

6 - 1　レビュー用セルフチェックリストの活用

A社では、要件定義後、開発着手の可否を判断する「投資ゲート」を制度化している。制度化当初は、投資額検討の抜け・漏れが開発着手後に発覚し、追加予算申請が必要となる事例があった。そのため、セルフチェックリストに抜け・漏れを起こしやすい費用項目の欄を設けることで要件定義時の検討を促すこととした。

明示した項目は、要件について「運用・移行・教育の各要件の詰まり具合は十分か」、見積り根拠について「移行準備・作業費、他システム変更作業費、業務システム教育費・システム操作マニュアル作成費、ハードウェア等についても考慮しているか」、「開発一時費用だけでなく、稼動後の運用・保守費用も検討しているか」、「運用体制・維持管理費用は明確か」といったものである。また「投資ゲート」の視点から、これらの費用を踏まえて「開発一時費用、及びシステムライフサイクル期間の運用・保守費用を含めて、費用対効果を評価したか」を確認する。

成功の要因

要件定義時のセルフチェックリストを整備し、抜け漏れしやすい項目を明示したことで、要件定義時の検討を促した。また、費用算定の必要上、どの範囲をベンダに発注し、どの部分を社内で手当てするのかなどが明らかになるような工夫をしており、後工程での予算追加申請件数が減少した。

件名				記入日　　年　月　日
記入者				

項目	項目区分	レビュー項目	自己評価・自己認識 ・自己所見・コメント ・記載されているドキュメント
要件の確定度合い		:	
	運用要件	運用サービス要件を明確化していますか（サービス提供時間、ヘルプデスク受付時間、等）。 運用委託先の運用基準と、システムが要求する運用要件は、整合していますか。	
	移行要件	業務およびシステムの移行時期・移行対象（業務・データ）・移行方法を具体的に計画しましたか。 並行本番を実施する場合、新旧システムの比較・確認手段が決められていますか。	
	教育要件	システムを効果的に活用していくための取り組みを実施あるいは計画していますか。 （業務改革やシステム化の目的を達成するための、ユーザーへの意識付けや教育、利用促進のための組織的な支援、等）	
		:	
見積りの妥当性	見積り根拠	見積りは、過去の類似システムの実績を参考にして作成しましたか。 移行作業費、他システム変更作業費、業務システム教育費・システム操作マニュアル作成費、ハードウェア等についても考慮されている 成果物のチェックと検収に関わる工数を見込んでいますか。 社内の調整、会議、レビュー、承認手続き、管理業務等のための工数を見込んでいますか（特に大規模案件）。 維持管理用ドキュメントの作成工数を見込んでいますか（業務取扱書、、維持管理用資料、等）。	

図2-1　レビュー用セルフチェックリストへの組み込み例

原理原則[7]

ライフサイクルコストを重視する

ポイント

情報システムのコストを算定するにあたっては、開発だけでなく運用を含めたライフサイクル全体のコストを重視する必要がある。システムのライフサイクルでは、定常的な運用・保守だけでなく、業務の見直し、あるいは法令・制度変更に伴う改修や、パッケージ、ミドルウェア、OSのバージョンアップ、ハードウェアの更新でも費用が発生する。

事例7-1は、パッケージのカスタマイズ部分が、パッケージのバージョンアップ時にそのまま適用できずに再構築が必要になったというトラブル事例である。

業務パッケージを採用した場合、カスタマイズを施すと、バージョンアップ時に予想以上のコストがかかることも考えられ、カスタマイズを前提としないことが原則である。

関連するプロセス共有化WG事例

7-1 パッケージのバージョンアップでカスタマイズ部分の再構築が必要となった

社内人事給与業務のレガシーシステムの再構築時に、人事給与パッケージの適用を決断し、要件定義、Fit & Gap分析（*1）もしっかりやり、パッケージ費用に対する50%比率のカスタマイズで、新システムを開発・導入し、スクラッチ開発に比べ、開発費用、工期の短縮に成功した。しかし、4年後の機器更新時に、パッケージのバージョンアップも必要になり、検討したところ、パッケージのミドルウェアがすべて一新されたことが判明した。そのため、カスタマイズ部分の全面再構築が必要になり、カスタマイズ部分については、当初の開発費用と同等の費用がかかった。

失敗の要因

当初は、パッケージベース開発なので、カスタマイズ部分の保守、バージョンアップ時のリスクを考慮し、可能な限りカスタマイズは行わず、パッケージに業務を合わせるという方針でプロジェクトを開始したが、ユーザ要求を抑えられず、カスタマイズ比率があがってしまった。

（*1）パッケージを適用する際に、要件定義との合致状況及び相違事項などを分析すること。

原理原則[8]

システム化の方針・狙いの周知徹底が成功の鍵となる

ポイント

情報システムを取得するにあたっては、少なくとも要件定義の段階でシステム化の方針や狙いをすべてのステークホルダで共有していないと、不適切な設計を誘発してしまう恐れがある。

事例8-1は、システム全体の実現イメージが描けないまま、個々の機能レベルの議論に終始してしまい、結果的に現場部門とシステム化の方針・狙いの共有が果たせず、先に進めなくなったというトラブル事例である。

システムの実現イメージをしっかり描ききり、システム化の方針・狙いとともにステークホルダの理解を得ることが成功の鍵となる。

関連するプロセス共有化WG事例

8-1 システム全体の実現イメージが描けない

ERPパッケージの適用を前提としたシステム化計画の案件で、IT部門主導で作業を推進した。体制としては、CEO/CIOの下に、現場代表、エンドユーザも参画して、IT部門主導で作業を推進した。大規模にもかかわらず、リリースまで１年という短期開発であったため、パッケージの適用を前提とし、現状の問題点とパッケージでの解決策を現場の代表と一緒に検討するというスタイルで要件定義を実施したが、パッケージを前提とした業務の全体像が描けず、個々の機能レベルの議論に終始してしまい、CEO/CIOから要件定義完了の承認を得ることができなかった。

失敗の要因

現場の反発を恐れたため、現行業務とパッケージの差異を気にしすぎ、システム化の方針、狙いの周知徹底が十分でなかった。現場の意識変革を得られないまま、詳細だけを追った作業に終始した。
本来はパッケージ導入により、実現イメージが現行業務と大きく異なる部分、あるいは機能が後退する場合もあるはずである。全体最適のメリットを訴えて、これら負の部分を相殺してみせる必要があるが、そのような手順を踏めなかった。

原理原則[9]

要件定義は発注者の責任である

ポイント

要件定義は、発注者がどのようなシステムを作るか、何ができるシステムを導入するかを定義することである。受注者が支援を提供することはあっても、あくまで発注者の仕事として、発注者の責任で行うものである。

事例9-1は、発注者側の業務知識不足や要件決定力が乏しいことにより、発注者の責任が果たせずに招いたトラブル事例である。発注者としての責任を全うするには十分な体制を組むことが第一の条件であり、要件定義には業務部門の経験者や有識者の参画が必須である。それが難しい場合は、外部のコンサルタントや受注企業の専門家の支援を依頼し、業務部門の経験者の参画を得るといった運営上の工夫を行うことも重要である。

事例9-2は、発注者責任を明確にすることにより、要件定義等の上流工程での品質の作り込みに取り組み、難易度の高いシステム開発を成功裡に完遂した成功事例である。

関連する「見える化」事例（上流工程編）

9-1　顧客の体制が弱いときは、仕様追加、変更が多発する可能性が大

システム化範囲が広い業務システムの開発を受注し、スパイラル型で作業を進めた。発注者側のとりまとめ担当者が業務部門の要望を仕切ることができず、業務内容の定義がきちんとできないことが懸念された。4回のイテレーションを回すスパイラル型で開発を進めることによって、顧客要件をきちんと抽出でき、システム構築後の手戻りもなくすことができると考えたが、各イテレーション(*1)毎に追加要求や仕様変更が多発し、規模がどんどん膨れた。それに伴い、進捗も遅れだし、品質にも問題が出始めた。さらに、規模の増大に伴ってテスト規模も大きくなり、結果的に工数が大幅超過となってしまった。

失敗の要因

それぞれのイテレーション毎に追加要求や仕様変更が多発し、規模がどんどん膨れたことにより、進捗が遅れ、品質にも問題が出始めた。その結果、テスト項目も大幅に増え、工数が大幅超過となってしまった。

関連するプロセス共有化WG事例

9-2　非機能要件を含む詳細な要件定義書の作成等、上流工程で発注者側が実施すべきタスクの強化により、ミッションクリティカルシステムが順調に稼働

極めてレベルの高い信頼性と高速性が要求されるシステム開発に当たって、発注者側は、上流工程での品質が重要との認識の下、RFP段階から詳細な要件を明確にした。要件定義書の作成に当たっては詳細化に努めるとともに、要件定義書の階層構造化や要件の抜け・齟齬等を削減するためのW字モデル(*2)の採用など、数々の新たな取り組みを実施した。さらに、従来の開発ではベンダに委託していた外部設計についても発注者側の責任で実施した。また、非機能要件の確実な抽出のために、発注者側が第三者（専門家）による設計書チェックを実施するなど、机上での検証を徹底し、後工程での問題の顕在化による手戻りの回避に努めた。
開発工程に入ってからは、変更管理を厳格に行い、QCDの確保に努めた。

成功の要因

最新のアーキテクチャを採用した難易度の高いシステム開発であったものの、計画どおりにカットオーバーし、安定稼働している。特に性能要件については、要求水準を大幅に上回る実績が達成でき、多くのシステム利用者からは、新システムが提供する高いサービスレベルに満足できるとの評価を得ている。

（*1）反復型開発プロセスにおいて、終了条件を満たすまでの一つの処理を実行すること。

（*2）反復型開発モデルにおいて、V字モデルを繰り返して採用するモデルのことをいう。

原理原則[10]
要件定義書はバイブルであり、事あらばここへ立ち返るもの

ポイント

要件定義書を作らないということは、発注者側がシステムに何を期待し、何を実現し、いつまでに開発を終了させるかなどを受注者側に明示せずに、システム開発に取り掛かるようなものである。目的、費用、工程などを明確にせずに開発に着手することは、早晩、プロジェクトに破綻をきたすことは明白である。

ステークホルダ間の合意は要件定義書により合意し、以降は、安易に変更できない「重み」のあることを認識して対処する必要がある。

事例10-1は、問題があれば、開発時だけでなく保守時にも、要件定義書に立ち返って対処している成功事例である。

事例10-2は、システムテストケース及びデータを、業務要件定義書から起こすのではなく、ベンダ自身が作成したシステム要件定義書から起こしたため発生したトラブル事例である。

基本動作としてシステムテストケース、データは処理の目的となった要件定義書から作成しなければならない。

関連するプロセス共有化WG事例

10-1　開発だけでなく保守においても要件定義書に立ち返って対処

建設会社Ａ社では、要件定義書をベースにした提案書（RFP）により、概算費用を把握しベンダを選定している。基本設計工程以降も、要件定義書を変更管理のベースラインとして進めている。
したがって、機能要件の増減や運用テスト時の検証も、要件定義書をもとに判断している。これにより、開発の各工程において、成果物に対する責任が明確になった。
また、保守時においても、業務要件レベルの変更を伴う機能追加・変更では、要件定義書の中の業務フローと業務機能説明書の改訂を義務付けている。

成功の要因

開発時だけでなく保守時にも、要件定義書に立ち返るため、業務フローや業務機能説明書の役割や位置付け、重要度が明確になり、要件定義の精度が向上する。

関連するプロセス共有化WG事例

10-2　システムテストケースを要件定義書から起こさなかったことによる本番障害発覚

金融系事業会社Ｂ社の業務システム。請求書に記載する請求金額について、特約等にかかる金額が加算されない不具合がサービスイン後に発生した。システムテストで検出されるレベルの不具合だった。

失敗の要因

この会社では、業務処理条件を、業務要件定義書（基本要件定義書）として作成し、それをベンダがシステム要件定義書（概要設計定義書及び詳細設計定義書）に落とし、ソフトウェア作成につなげている。システムテストケース及びデータを作る際、業務要件定義書から起こすのではなく、ベンダ自身が作成したシステム要件定義書から起こしたため、システムテストケース及びデータの要件把握不足を発見できなかった。

原理原則[11]
優れた要件定義書とはシステム開発を精緻にあらわしたもの

ポイント

目的とする業務要件を実現するために、機能要件はもちろんのこと、性能、移行、運用などの非機能要件を漏れなく洗い出して、精緻に定義する必要がある。

特に、ライフサイクル全体にわたる運用要件など、プロジェクト特有の制約条件などを考慮する必要がある。

事例11-1は、日常の運用要件は設計されていたものの、年次処理における運用要件が抜け落ちていたために重大障害を招いてしまったものである。言い換えれば、要件定義書にシステムライフサイクル全体にわたる運用要件を精緻に表していないことが生んだトラブル事例である。

ベンダとしては、年度又は年度末をまたがる処理などのシステムのライフサイクルを通した運用についても、仕様が曖昧であれば、顧客に提案するべきであった。

関連する「見える化」事例（中流工程編）

11- 1　仕様通りに開発したが運用障害で事業が長時間停止

端末でのマンマシンインタフェースの操作性要件や通常日の運用仕様は、顧客側で具体的に決めてもらえた。しかし、システムのライフサイクルを通した運用については曖昧であった。カットオーバー後、年度をまたがる時に、本来行うべきファイル切替・再設定などの自動処理機能がシステムとして設計されておらず、また運用者の手動手順も用意されていなかったことから、ファイルの上書きなど重大障害が発生した。運用手順マニュアルにはないデータ・リカバリが必要となり、サービス再開まで長時間かかった。顧客は事業機会を逸失する事態となったので、ベンダが責められた。

失敗の要因

要件定義書に従い、マンマシンインタフェースの操作性要件などを忠実に設計に反映したのだから、それで十分だと思った。
ベンダとしては、月末処理、年末処理・年度末処理、閏年処理などの場合には、システムのライフサイクルを通じた運用についても仕様が曖昧であれば、顧客に提案するべきであった。

原理原則[12]
表現されない要件はシステムとして実現されない

ポイント

要件定義書で表現されていない要件はシステムとして実現されない。また、ベンダは要件定義書に表現されていないものを推測してシステムに実装してはならない。システム開発の原則は､「書いてあることはやらなくてはいけない。書いてないことはやってはいけない。疑問に思ったら確認して、記述して実施せよ。」である。

事例12-1は、現場での運用実態まで詳細に調査しないまま要件定義を行ったことにより、事業所独自のローカル仕様を要件定義に反映できずにシステム更改を実施したことによるトラブル事例である。

要件定義では、運用要件などの非機能要件も忘れずに検討対象とする必要がある。

関連する「見える化」事例（上流工程編）

12- 1　運用テストに入ってから現場固有の業務要件が多数発覚

給与システムの更改プロジェクトで、運用テストに入った段階で、事業所独自のローカル仕様があることが判明した。ローカル処理は当面、手作業でカバーすることにした。そのため、1年間にわたって、顧客側の運用コストが大幅にかかってしまった。
さらに、事業所によって処理方法の一部が異なっていることも判明した。

失敗の要因

事業所独自のローカル仕様があることが運用テストで判明したが、他の事業所でのローカル仕様の有無を調査せずに、システム更改を実施した。
事業所ごとのローカル仕様がどれだけ存在するのかを把握し、もしその量が多い場合は、段階移行か、リリースの延期をするなどの検討を顧客と実施する必要があった。

原理原則[13]
数値化されない要件は人によって基準が異なる

ポイント

要件定義では、定量化できるものは必ず定量的に記述しておく必要がある。特に、非機能要件については､「すぐに」「速やかに」といった抽象的な表現は避け、定量的な数値化された表現を用いなければならない。品質要件、技術要件、運用・操作要件等は、数値化した表現が可能であり、抽象的な表現では、発注者・受注者の解釈・物差しが異なり、トラブルの原因になる。

非機能要件については、以下が参考となる。

・『非機能要求仕様定義ガイドライン』
（検収フェーズのモデル取引・整備報告書
UVC（User Vender Collaboration）研究プロジェクトⅡ報告書2008）
（経済産業省 情報処理振興課、株式会社NTTデータ経営研究所、
社団法人 日本情報システム・ユーザー協会（JUAS））

事例13-1は、ユーザからの「業務上支障のないレスポンス」という定性的な要求をそのまま受け入れるのではなく、数値によって定量的に要件化したことによる成功事例である。

関連するプロセス共有化WG事例

13- 1　性能要件の数値化

業務データを集計・分析するデータウェアハウス・システムの開発プロジェクトで、顧客が作成したRFPには､性能に関する要件が定義されていなかった｡改めて要件定義を行い､性能に関する要件を確認すると､「業務上支障のないレスポンスで」という回答であった｡さらに具体的な数値の提示を求めたところ､「一つの操作に要するレスポンスタイムは x 秒以内」という定量的な数値を得られた。

成功の要因

顧客の要求は「業務上支障のないレスポンス」というものであったが、ある人にとっては業務上問題がないレスポンスタイムであっても、他の人にとってはそうでないこともある。また、数値は記述されていてもレスポンスの定義を明確にしなかったために、紛争に至った事例もある。
「一つの操作に要するレスポンスタイムはx秒以内」という具合に、性能要件を条件と数値で定義することにより、適格性確認テストで要件を満たしていることが確認できる。

原理原則[14]

「今と同じ」という要件定義はありえない

ポイント

既存システムと同じ機能、すなわち「今と同じ」システムを構築する場合でも、新たに要件定義を行わなければならない。既存システムがどのように使われているかを調査し、改めて必要なシステム機能を洗い出すとともに、新システムの実像を明確にした上で、要件定義書に落とし込む必要がある。「今と同じ」機能を部分的に実現する場合も同様である。

事例14-1は、陳腐化に伴うシステムの再構築において、旧システムを踏襲する機能も含めて、きちんと本来の要件定義を進めたことにより、精度の高いアウトプットを作成することができた成功事例である。

事例14-2は、プロジェクトを円滑に進めたいがために、「旧システムの仕様どおりのものは、その旨をベンダに伝えればよい」という方針（ベンダも確認済み）で進めたことにより発生したトラブル事例である。

たとえ、「今と同じ」システムを構築する場合でも、新たに要件定義を行う必要があった。

関連するプロセス共有化WG事例

14-1　再構築案件でも新たに要件定義を実施

陳腐化に伴い、マスタ、受注、請求入金関連システムを再構築することになった。システム化計画の中では、約4割の機能は“何も手を加える必要がない”という結論に達していた。本プロジェクトでは、開発規定に則り、要件定義に入る前に、必要な成果物や記述レベルの確認をステークホルダ間で行ったが、“何も手を加える必要がない”機能も、区別することなく要件定義の対象にすることを確認した。必要な機能だけでなく、廃止する機能を明確にするとともに、業務フローをゼロベースで見直し、業務のシンプル化も図った。また、保守運用フェーズのことを考え、旧システムでは対象ドキュメントにしていなかったCRUD図[*1]（情報分析図）も新たに作成している。

成功の要因

この企業は、過去に実施した再構築プロジェクトで、“何も手を加える必要がない”機能の検討を疎かにし、失敗した経験があった。本プロジェクトでは、その経験を活かし、必要なすべての機能について要件定義をしっかりと行ったことが高く評価できる。初期の検討段階ではユーザ要望がなかったとしても、要件定義を進めていく中で、業務フローの見直し、あるいは新規・修正機能などから変更要求が出てくる可能性を加味する必要がある。

関連するプロセス共有化WG事例

14- 2 「旧システム仕様」を設計書に十分に反映しなかった事による品質不良

システム化計画～要件定義の遅延に伴い、基本設計以降の工程が遅れていた。設計が完了したものから随時レビューし、設計書に修正を反映するという流れになってしまった上に、修正規模が予想以上に大きくなり、設計書の修正と再レビューが徹底できなかった。一方、「旧システムの仕様どおり」の機能は、ソースコードを含めた調査を行い、その結果を設計書に反映することになっていたが、遅延に伴う調査の遅れと設計書修正の混乱の中、反映すべき情報が錯綜し、他機能との整合性の確認も疎かになった。結果、設計以降の工程の更なる遅延を招くとともに、テスト段階での品質が悪化した。

失敗の要因

この企業は、過去に実施した再構築プロジェクトで、変更がない機能の要件定義を疎かにし、プロジェクトが失敗した経験があったにもかかわらず、それがまったく活かされなかった。「旧システムの仕様」は、設計書に反映しないという方針ではなかったが、プロジェクトが遅れ始めていたことが明白だったことを考えると、更なる遅れによる混乱と、それによって生じるリスクを考慮し、別の方策を採るべきであった。

（＊1）データの生成、削除、更新などを表形式で表した「情報分析図」

原理原則[15]
要件定義は「使える」業務システムを定義すること

ポイント

業務部門、システム部門、運用部門が一致協力して、新システムの実現を目指して、業務運用が成立するかどうかを検証しつつ、目的とするビジネス要求を満足するための要件を確定する必要がある。

事例15-1は、要件定義工程でシステム部門とユーザ部門が共同で、業務要件定義作業を行ったことにより、基本設計工程以降における業務機能レベルでの手戻りや変更が少なくなった成功事例である。

関連するプロセス共有化WG事例

15- 1 要件定義作業をユーザ部門と共同で実施

建設会社Ａでは、要件定義工程でシステム部門とユーザ部門が共同で、業務要件定義作業を行っている。成果物としては、システム化後の業務フローと業務機能説明書に加え、画面イメージも作成している。レビュー時には、画面イメージを確認しながら、業務フロー及び業務機能の検討を行っているため、システムカットオーバー後の新業務の具体的な運用イメージがつかみやすくなった。
このような形で要件定義工程を実施して以来、基本設計工程以降での業務機能レベルでの手戻りや変更が少なくなった。

成功の要因

要件定義工程へのユーザ部門の参画と、レビュー時にシステム化後の業務フローや業務機能を画面イメージを併用して確認するため、運用を想定した使える業務システムの要件定義が実現できている。

原理原則[16]

要求機能は膨張する。コスト、納期が抑制する

ポイント

システム開発は限られた「期間」と「コスト」の中で、業務に必要な「機能を実現」して初めて評価されるものであり、ステークホルダ全員がこれらのバランスを常に念頭に置く必要がある。

事例16-1は、既存システムのリプレースに当たって、ユーザ部門からの新規機能要件の内容が不明確であったので、段階的稼働を選択した。これにより、開発途上での要求機能の膨張を回避した成功事例である。

関連するプロセス共有化WG事例

16- 1　2段階稼働により、要件不確定リスクを回避

Ａ社では、基幹業務システムのリプレースに合わせて、社内支援システムの同時リプレースを計画していた。リプレースに当たっては、社内ユーザ部門からは、様々な新規機能の追加要望が寄せられていたが、基幹業務システムの要求機能の策定と並行して検討が進められたことから、社内支援システムへの追加要望の具体化が進まなかった。
このため、同社では社内支援システムの開発遅延が、基幹業務システムのリプレース計画に影響を与えないようにするため、社内支援システムの段階的稼働を決定した。

成功の要因

社内支援システムの開発に当たっては、基幹業務システムと連動する既存システムからの継承機能と要件が確定していた新規機能を、基幹業務システムのリプレースに合わせて、優先的に稼働させた。各ユーザ部門では、二段階開発となったことにより、追加要望の具体化にかかる作業時間を確保できたので、第二段階の開発に入ってからの要求変更等の発生を最小限に抑えることができ、コスト面でも当初の予算からの若干の増加に止めることができた。また、新規機能の利用開始時期は、当初計画より遅れたものの、ユーザ部門の満足度は高い。

原理原則[17]

要件定義は説明責任を伴う

ポイント

確定した要件を、次の基本設計工程に正確に伝えるために、要件定義者は基本設計者に対して、要件定義書を渡すだけではなく、その内容を正しく説明する責任がある。

「要件の行間を読め」といった要求は論外であり、当たり前の前提や例外処理であっても漏れなく伝達する必要がある。品質確保の重要なポイントは、発注者、受注者双方が説明責任を果たすことにある。

事例17-1は、大規模、短納期、特殊な業務という難しい条件を的確に判断した発注側IT部門リーダが、業務部門を説得して詳細な仕様書を作り、それをベンダに徹底的に説明することによって、品質、予算、納期を守った成功事例である。

関連するプロセス共有化WG事例

17- 1　発注者側の手厚い説明と設計書合同レビューにより難しい開発をクリア

スクラッチ開発の大規模社会システムの割に納期が厳しく、１年半で稼働させなければならなかった上に、業務は科学技術系の特殊仕様で、一般的な知識は全く通用しない状況であった。顧客側のIT部門リーダが、この状況を的確に判断し、業務部門から詳細な業務仕様をベンダ側に対して提示、説明させた。ベンダ側も徹底的に内容を確認した上で設計書を作成し、合同レビューを重ねることで、納期どおり、予算どおりに開発を完了し、運用開始後、数年を経た今日まで大きな問題は発生しておらず、安定稼働を続けている。
顧客IT部門のリーダが「レガシーシステム刷新による、運用コスト削減」という開発目的を明確に掲げ、この目的に直接関係しないエンドユーザ要求を抑えて、開発規模の増大を最小限に止めた。

成功の要因

「本業務システムは特殊かつ国民の日常生活に密着したものであり、システム仕様に漏れがあっては、運用開始後に重大な社会問題を引き起こしかねない」という顧客IT部門リーダの強い想いの下、顧客が基本設計書にも匹敵する詳細な仕様書を作成した。
設計作業開始に先立って顧客が仕様書内容をベンダに説明するとともに、ベンダが作成した設計書を合同でレビューし、レビュー会回数は半年で200回以上に及んだ。

第３章　失敗から学ぶ原理原則17ヶ条

原理原則17ヶ条は、システム開発の実務ではどのような場面で必要となるのか。本章では、ユーザあるいはベンダにとって失敗に終わったシステム開発事例を題材に、17ヶ条を守るべきだった場面を明らかにする。

以下の文章で、下線部は17ヶ条を守るべきだった場面を示し、その後ろの番号(#)は関連する原理原則の条文番号を示す。

3.1　ユーザ事例[１]　原材料輸入システムの新規構築

（１）経　緯

ユーザA社は、原材料の輸入に係る一連の業務をIT化するプロジェクトを立ち上げた。工期1年3ヶ月、総予算1億円の中規模プロジェクトである。

A社では、各部に予算と権限が与えられ、それぞれの目的に応じてシステムを作る体制に変わったところだった。そのため、このシステム開発は調達部が主管し、A社と関係の深いベンダB社に発注した。調達部はIT化の経験がなく、B社は当該業務の経験がなかった。従来の体制ならば、A社内のIT部門が両者の調整役となるが、社内の分権化促進のため、このプロジェクトにIT部門はタッチしないという方針でスタートした。

原材料輸入の取引相手は、15ヶ国を超える。契約ごとの輸入計画、税関手続き、検収、為替レートの変動に対処する決済などがシステム化の対象業務である。多様な処理を含むが、それほど複雑な業務システムではないと想定された。

しかし、このプロジェクトは、結論からいえば、大きな失敗に終わった。サービスインは二度延期され、開発期間は2年半も延び、開発コストも当初予算より大幅に超過した。超過したコストは、A社及びB社の双方が負担するという形でようやく決着したのである。

（２）"相手はプロ"が生んだ悲劇

失敗の原因は上流工程のミス、この一言に尽きる。

調達部はRFPに「**要件定義以降を一括請負**(#5)とし、**要件定義はベンダ中心**(#9)に行うこと」という条件を入れ、Ｂ社は推測可能な範囲内で業務を想定し、調達部に質問することとした。しかし、この調達部の考えは「ITは、よく分からないから任せるよ。**聞かれたら答える**(#17)から。」という考えであり、双方協力して新規業務に対する課題、問題点を発見していこうという姿勢に欠けていた。調達部は“IT化のプロに任せれば要件定義はうまくいく。請け負っているのだから当然だろう”と思い込み、Ｂ社も“業務のプロが言うことに抜けはない。**言われたものを作れば大丈夫**”(#12)と思い込んだ。このように、**双方が大きく相手に依存**(#1)した結果、多くの要件が見過ごされていった。

（３）上流工程の失敗が尾を引いた

開発スタートから7ヶ月後、要件定義から次フェーズ（基本設計）に移行するに当たってIT部門によるレビューを実施し、「**関係者間で認識が共有されているかどうかやや不安**(#8)を感じる」と指摘された。が、その指摘に基づいて**要件定義が見直されることはなく**(#4)、Ｂ社は基本設計、さらには詳細設計へと突入した。その間、**調達部はほとんど開発にタッチすることなく**(#15)、のちに「ユーザ空白の期間」とまで呼ばれることになった。

見逃された大きな要件の1つは、**過年度分データ移行の見落とし**(#6)である。

原材料価格は相手国との交渉で決まるが、その価格交渉自体が決着するまで数ヶ月～数年かかる。そのため、いったん仮価格で支払いし、交渉が決着してから遡及計算する。これは決算・財務への影響があるため、経理システムとの連携でも考慮しなければならない。したがって、このシステムの開発では、**過年度分のデータの扱いが大きな意味をもっていた**(#11)のである。

だが、これはＡ社とＢ社の共通認識となっていなかった。ユーザである調達部にとっては、「遡及計算がある＝過年度分のデータ移行は、説明するまでもない常識」。Ｂ社にとっては「決算は単年度で行うのが常識」。双方の常識がまったく食い違っていたことが、最初の開発が終了し、データ移行･切替の間際に発覚したのである。要件定義の大きな落とし穴だった。

これ以外にも要件の抜け漏れが、総合テストで多数見つかり、稼動の目処が立たない事態となった。プロジェクトは結局、調達部側の増員、**要件定義の徹底的な洗い直し**(#16)、Ｂ社のプロジェクトマネジャの交代・補佐の追加などの対策を講じて再スタートし、1年後に何とか稼動にこぎつけた。「上流工程の大きな失敗は挽回するのに多大な時間とコストを要する」「要件定義は発注者側の責任である」など、このプロジェクトの失敗から得た教訓は大きく、Ａ社におけるシステム開発の行動規範となっている。

3.2　ユーザ事例［２］　特約店業務のIT化

（１）経　緯

衣料販売会社Ｃ社は、商品の受発注、商品情報の紹介、各種情報の提供等、特約店の業務全般をIT化するプロジェクトを立ち上げた。工期1年、総予算1億円の中規模プロジェクトで、ITベンダＤ社がシステム開発を請け負った。

Ｃ社のIT部門は、基幹システムの開発･運用業務を主体とした体制となっていた。新システムの開発は現行業務に割り込む形で作業を行うことになり、十分な体制は構築されなかった。しかし、Ｄ社は、Ｃ社の基幹システム、業務システムの開発・運用も請け負っているため、業務内容・システム構成をよく把握しており、Ｃ社の体制の脆弱さは問題視せず、プロジェクトをスタートした。

（２）一括請負契約のリスク

要件定義及びRFPの作成はＢ社が実施したが、体制の問題もあり、**要求**

が十分に網羅されたものではなかった[#9]。Ｄ社は、RFPの内容に不安を感じ、Ｄ社も参加した要件定義の再実施、システム開発の多段階契約を主張した。しかし、Ｃ社は、要件定義とシステム開発契約の分割やシステム開発の多段階契約は年度をまたがってしまうため、予算計画の都合上難しいと、一括契約を主張し、譲らない。結局、Ｄ社は、長年Ｃ社のシステムを請け負ってきた経験と知識で不安要素はカバーできるだろうと判断し、**一括請負で契約**[#5]した。

プロジェクトでは、Ｄ社が中心となってシステム要件定義を進めた。Ｄ社は、長年の付き合いによって業務内容を把握しているとはいえ、Ｃ社の業務すべてを把握しているわけではなく、また、特約店の利用者と直接接点があるわけでもなかった。システム要件定義段階での開発規模はＤ社の想定範囲だったが、詳細設計に入ると、現場の利用者、システムの運用者など関係部門からの要求が一気に噴き出し、システムの開発規模が急激に膨れ上がった。

Ｄ社は、システムの開発規模が想定を大幅に上回ったため、Ｃ社に契約内容の変更、もしくは開発規模の縮小を要求したが、既に一括請負で契約を結んでいるため、契約の変更は認められなかった。

Ｄ社は、多段階契約を諦めて一括契約したことを後悔したが、あとの祭りで、このプロジェクトは不採算覚悟で進められた。

（３）システム運用要件の見落とし

その後、プロジェクトは開発段階に進み、テストからシステム運用の作業についてＣ社とＤ社で確認会議が実施された。

ここで、新たな問題が発覚した。「マニュアル作成」「教育」という作業項目に対する認識がＣ社とＤ社で異なっていた。Ｄ社は、システム運用者向けのマニュアル作成、教育を想定していたが、Ｃ社は、システム運用者のみならず、利用者向けのマニュアル、教育も含むと主張した。

問題の原因は、**システム開発の要件にばかり気を取られ、システム運用**

に入るための要件を十分に定義できていない[#6]ことにあった。結局、Ｄ社は利用者向けのマニュアルを作成し、Ｃ社は特約店向けの講習会を実施するという折衷案となったが、Ｃ社、Ｄ社とも想定外の作業で大きな負担を強いられた。

最終的に2ヶ月遅れでシステムが完成しサービスインしたが、Ｃ社にとってはサービスインが遅れ、Ｄ社にとっては不採算という結果になってしまった。

3.3　ユーザ事例［３］　基盤システムの再構築

（１）経　緯

金融商品を取り扱うＥ社で、基幹システムを全面的に再構築する計画が持ち上がった。現行のメインフレーム中心のシステムをJava中心のクライアントサーバー型システムに刷新するという計画である。ITベンダＦ社は受注に向けた提案を行うことになった。既に別のコンサルタント会社Ｇ社が、Ｅ社と共同で「開発仕様書」を作成していた。そのため、Ｆ社はこの「開発仕様書」を読み解いて提案書を作成し、最終的にＦ社が受注してシステム開発を担当することになった。

（２）見積り時の失敗

要件定義では「開発仕様書」の内容をもとにして、現行システムの設計書やユーザマニュアルを参照し、要件定義書を作成しようとした。現行システムの仕様は当初予想より複雑であり、要件定義書の内容に細部まで反映させることができなかった。Ｆ社も、Ｇ社も、現行システムの仕様の知識が乏しく、突っ込んだ検証ができなかったことが原因と考えられる。見積りは、開発仕様書をベースとした提案時の内容を踏襲する形になり、**外部設計以降の請負契約につながった**[#5]。これが、結果的には、開発コストのみならず、仕様の不整合や漏れを招く一因になった。

（３）サブシステム間の仕様不整合の問題

外部設計の段階ではあまり遅れが目立なかったが、詳細設計フェーズに入ると、サブシステムのレベルで進捗に差異が生じ、その後の開発フェーズでも、サブシステム間の進捗の差異が解消できなかった。サブシステム間結合テストでは、仕様不整合によるテストのやり直しが多発するようになり、何とかこぎつけたシステムテストの途中で完全にストップしてしまった。全体スケジュールの遅延を恐れて、設計仕様が固まらないうちに開発に着手するなど、**五月雨式に開発に入っていったことがサブシステム間の仕様不整合の原因となった**(#4)。もともと開発期間は約2年間の予定だったが、それを1年以上遅らせることになった。

（４）追加案件への対応

このプロジェクトでは、開発期間中に、さらに追加案件の検討も進められていた。プロジェクトとしては必要最低限の仕様でシステム稼動を行う計画だったが、**設計漏れ等が発見されることもあり、Ｅ社の要求で少しずつ要件を追加することがあった**(#16)。また、最初に稼動するシステム設計を担当したメンバーが、第二期、第三期稼動システムの要件検討のために、開発チームから抜けることもあった。そのため、設計漏れや仕様不整合が判明しても、設計者とは異なるメンバーが原因を調査することになり、そのワークロードが作業の進捗に影響を及ぼした。

（５）結　果

システムテストがストップした時点で、Ｆ社はプロジェクトのトップマネジメントの交代を行い、開発システムの品質やプロジェクト管理体制の見直しを行った。その結果をＥ社に説明した上で、新たなスケジュールで開発を再開した。結果として、システムを無事稼動させることはできたが、Ｅ社にとってもＦ社にとっても大きな損失をもたらすものとなったことは否めない。

このように、システム開発の超上流工程には落とし穴が多数あり、そこに落ちると、考え違い・思い違いが次から次へと波及していき、気付いたときには取り返しのつかない事態となってしまう。

この落とし穴を回避して、プロジェクトの成功というゴールに無事に到達するためには、原理原則17ヶ条を理解し、その行動規範に沿ってユーザ・ベンダ双方が協力し、経営者や現場ユーザも含めたコミットメントを得ながら、確実な要件定義を行うことが重要であることが理解できると思う。

第４章　成功を支える原理原則17ヶ条

「システム開発の成功事例には、「原理原則17ヶ条」に基づいた取り組みが含まれていることが多い。失敗事例と同じく、ユーザー企業の視点で探ってみる。ただし、原理原則17ヶ条のうち、どの項目が、どのようにしてシステム開発の成功につながったのか、本章ではあえて個別項目との関係は示していない。読者自身で見出してもらえれば幸いである。

「原理原則17ヶ条」は、その全項目がシステム開発を成功させるために必須というわけではない。システム開発に臨む企業や組織が、直面するプロジェクトの特性や過去の経験に応じて使い分けることが大切である。自らの長所を活かし短所をカバーする際のヒントとしても活用できる。また、17ヶ条で挙げた以外にも、システム開発を成功させる上で重要なポイントはあり得る。ここで取り上げる成功事例を参考に、読者自身が「18条以降」を見出すのも有意義であろう。

4.1　手戻りを徹底排除、開発スピード５割増

開発スピードを5割高め、開発コストを3割減らす。小売り業Ａ社は、店舗の発注システムを全面再構築する際、このような納期と予算の目標を掲げて臨んだ。経営者から「これらの条件をクリアできる見込みが立たなければ、ゴーサインは出さない」と言われたからだ。

Ａ社の発注システムは開発規模が750万ステップに及ぶ大規模なものだ。開発に費やせる期間は、構想･企画フェーズを含めて4年強。過去のプロジェクトと比べると、開発スピードを5割以上高めなければ、間に合わない計算だった。今までと同じ仕事のやり方では、間違いなく納期遅れをまねく。

そこでＡ社のIT部門は、システム開発の仕事の進め方を全面的に見直すところから着手した。Ａ社は、システムの設計から結合テストまでを、複数のITベンダーにアウトソーシングしている。開発スピードの5割向上とコストの3割削減を実現するには、これらITベンダーの協力が欠かせない。そこで、まず主要なITベンダーを巻き込んで、プロジェクトの進め方を議論した。

その結果、「手戻りを徹底して排除するしか方法はない」という結論に達した。この結論を導くのは、それほど難しいことではない。問題は、どうやってそれを実現するかだ。Ａ社は検討の末、二つの工夫を考え出した。

一つ目はシステム開発の仕事の標準化だ。新システムの計画立案、要件定義、設計、テスト、稼働という、プロジェクトの最初から最後までを対象に、どの段階でどんな作業をするのか、誰が参加するのか、ステークホルダ（利害関係者）の合意をどのタイミングで取るのか、成果物をいつ作り、その承認をいつ誰がするのか、といったことを、すべて明確にしようと考えた。二つ目は、Ａ社が要件定義の段階で、システム化の要件をしっかりと固めることだ。できる限り詳細なレベルまで、システムの機能を確定することで、ITベンダーが設計以降の作業を進めやすいようにした。

（１）ステークホルダ間の合意と承認を重視

一つ目の工夫である仕事の標準化では、Ａ社独自のシステム開発手順を改めて定義した。プロジェクトの各段階で、検討作業に参加するステークホルダと、実施すべき作業、作業を終えたと判断するための条件、各作業の承認プロセスを明確化。さらに、要件定義書や設計書といったプロジェクトの成果物を、どの段階で作成するかを決めた。併せて、成果物の作成ガイドラインもそろえた。これらにより、システム開発から属人性をできる限り排除することを目指した。開発手順を固めても、属人性を完全に排除することはできないが、「この作業は、まだ完了レベルに達していない」といった問題を一定の基準に基づいてチェックできるようにはなる。

ステークホルダが、稼働直前になって「こんなはずではなかった」「そう決まっていたとは知らなかった」と言い出すことがないよう、各作業においてどんな取り決めをしたのかについて、ステークホルダとの合意や承認

をしっかり得ることも強く意識した。要件定義、設計といった各フェーズの完了段階でステークホルダの合意を得るのはもちろん、例えば要件定義を「システム化要件検討」「新業務の整理」「要件定義書の確認」など五つの作業に分けるなど各フェーズを細分化し、それぞれについてステークホルダの合意を取ってから、次の作業に進むことにした。A社が新たに定めた開発手順は、システム化の検討から稼働まで、約30の作業に分かれる。

A社が担当する要件定義やテストの工程だけでなく、ITベンダーが担う設計や実装、テストなどの工程についても、作業の進め方を見直した。

（２）要件は先送りでなく前倒しで固める

二つ目の工夫は、システムの要件を早い段階で徹底して固めることである。A社は要件定義の段階で、システムに盛り込む機能の検討・確定はもちろん、画面レイアウトをどうするかなど、かなり詳細まで踏み込んで議論した。要件確定を先送りしないという、システム開発の基本を守るのはもちろん、それだけにとどまらず、むしろ先に決められる仕様があれば、設計フェーズを待たずに前倒しで盛り込んでいくようにした。要件確定の先送り、要件定義の漏れや誤解による手戻りをできる限り減らすためだ。

例えば、店舗の従業員が商品を発注する際に使う発注端末の画面については、要件定義が完了するまでに、ボタン配置などのレイアウト、さらには画面遷移のフローまで確定した。いくら早く仕様を決めても、設計以降のフェーズになって、A社が画面レイアウトを変更してしまったら、ITベンダーは画面を再設計しなければならない。そこでA社は、一度決めた画面レイアウトは最後まで変えない前提で、画面の遷移や画面の数まで踏み込んで議論した。

例えば画面レイアウトでは、ドット単位でボタンの大きさや配置位置などを指定した。こうすることで、ITベンダーが画面を設計・実装する際、ボタンの表示位置をぴったり重ねて作りやすいように配慮した。ITベンダーの技術者が、ボタンをどの位置に配置すればいいか迷ったとき、要件定義書に立ち返れば、必ず疑問が解決するようにした。

テストの段階で、ボタンの位置が要件定義書から1ドットでもずれていたら「不具合」として扱うなど、要件定義の遵守を徹底した。要件定義書は、システム開発を精緻に表したものであるという考えに基づいての行動である。これは極端な例ではあるが、ここまで細部にこだわらないと、システム全体の品質は維持できないとA社は考えた。

結果的に、A社は発注システムだけで、1,000頁以上の要件定義書を書き上げた。なかには、現行システムと変わらない要件があったが、それらについても、「現行と同じ」とはせず、具体的な内容を明記することにした。A社は、現行システムの開発を任せたITベンダーに、新システムの開発も発注した。このITベンダーにはA社の現行システムのノウハウが蓄積されている。にもかかわらず、「現行と同じ」という表現は避けた。現行システムの担当者がそっくりそのまま新システムのプロジェクトに参加するとは限らないからだ。要件の誤解や漏れを防ぐために、たとえ発注先が同じ会社であっても、「現行と同じ」という記述は許さない方針を掲げた。

（３）「使える」システムを目指し、非機能要件を数値で明記

要件定義にあたっては、要件定義書をできるだけ詳細に記述するように努めた。要件として記述していない機能は、システムとして実現されないからだ。機能要件はもちろん、システムの性能や信頼性などに関する「非機能要件」の記述においても、この方針を貫いた。例えば、店舗の従業員が過去の販売データを検索する機能について、「データベースから1,000件程度のデータを抽出・加工し、結果をパソコンの画面に表示するような標準的な検索処理を、5秒程度でできるようにすること」といった具合である。

このとき、「ストレスなく検索できる」「すぐさま結果を表示する」といった定性的な表現でなく、「5秒」というように数値で要件を表すことを強く意識した。「ストレスなく」という表現に対する解釈は人によって異なるからだ。要件は数値化するなどしないと、細かく定義したことにはならな

い。

さらに、検索処理については、「全国の店舗から何件のアクセスが同時に押し寄せた場合」といった前提条件も明確にした。従業員が検索機能を使う時間帯は、発注の締め切り時間の直前などに集中する。要件定義で、こうした負荷の集中まで考慮することが、「使えるシステム」を生み出すことにつながると考えたからだ。

負荷の集中を考慮しないと、肝心なときに検索に時間がかかってしまうことになりかねない。使いにくいシステムに仕上がってしまう可能性があるというわけだ。そもそも、販売データの検索機能は、店舗の従業員が精度の高い発注をできるようにするためのものである。その機能が使いにくいとみなされて、あまり使われなくなってしまったら、新システムを開発する意味がなくなってしまう。

（４）開発費以外のコストも削減

手戻りを防ぐ二つの工夫により、システム開発のスピードを5割向上させるメドはついた。これにより、システム開発費用を下げられる見通しは立った。だが、システム開発にかかるコスト全体を3割減らすには、これだけでは足りない。ハードウェアやミドルウェアなどのライセンス費用、保守料、システム運用費といったライフサイクルコスト全体を見直さなければならない。

A社は小売業として国内最大規模の店舗網とビジネス規模を誇る。そのスケールメリットを、商品や原材料の調達と同様、ITの分野でもフルに活用した。最たる例が、店舗で従業員が発注操作に使う端末の調達だ。A社が端末の購入台数を見積もると、店舗数などから算出して2万台超となることが分かった。これだけでも大量購入のメリットを得られるが、A社はさらにグループ会社の店舗にも発注端末の導入を決断、購入台数を5万台まで増やした。このような工夫によって、システム化の費用を3割減らすことに成功した。

システム化の費用の中には、システムの設置にかかる費用も含まれている。A社はこの設置費用についても、コスト削減を果たした。システムの設置作業はITベンダーが担当する。そこでA社はこのITベンダーと共に、システムの設置に関する一連の作業を効率化するための方法を考えた。まず、ITベンダーの工場から製品を出荷する前に、旧システムから新システムにデータを移行する専用アプリケーションをあらかじめインストールしておいた。電源を入れるだけでアプリケーションが動き、特に問題がなければほぼ自動的に新旧システム間でのデータ移行が完了するようにした。さらに、店舗での初期設定作業、動作確認作業については、ITベンダーの特定の担当者が何度も繰り返し担うことにした。担当者の店舗での作業効率を高めるためだ。これらにより、以前のシステム刷新の際は二人がかりだった店舗での設置作業を、一人でこなせるようにした。結果的に、システム設置費用をかつての半分近くまで削減できた。

A社は、システム再構築の条件として、開発スピード5割増、コスト3割減という納期と予算の目標を、経営者とIT部門がコミットしてからプロジェクトに着手した。このことは、要件がどんどん膨らむような事態に歯止めをかけることにもつながった。

4.2　入念な計画とプロジェクト管理が奏功

金融業B社は、経営統合に伴うシステム統合プロジェクトを、大きなトラブルもなく、予算内で納期どおりに完遂した。B社のシステム統合は、約3年にわたる大規模プロジェクトである。このプロジェクトが成功した理由は、B社がプロジェクトを始める前に綿密な計画・準備をして臨んだことと、プロジェクト開始後のプロジェクトマネジメントを徹底したことにある。

（１）リスクを事前に洗い出し、対策を検討

システム統合プロジェクトの陣頭指揮を執ったIT部門長は、まずプロジ

ェクトマネジメント・オフィス（PMO）や品質・リスク管理、要員確保、コスト管理といった役割を担うチームを設けた。IT部門長の右腕として、ヒト・モノ・カネを管理する「コントロールタワー」の役割を果たす組織である。これらの組織の要員は、直接はシステム開発に携わらないが、特に大規模プロジェクトでは、このような管理・推進組織が欠かせないと考えた。

管理・推進組織を発足させると、B社はプロジェクトの途中で直面する可能性のあるリスクの洗い出しに取り掛かった。リスクについては、サブシステム毎に、システムの特性やデータ量、接続先システム数などを踏まえて洗い出した。ITベンダーの技術者を確保できるか、社外システムとの接続試験の予定が組めるか、といったプロジェクト全体に共通するリスクも挙げた。

いずれのリスクについても、単に列挙するのではなく、数値に換算して影響の大きさを定量的につかむことを目指した。定量化したリスクはプロジェクトの進捗に応じて解決状況を確認した。

洗い出したリスクについて、事前にリスクの解決手段を考え、具体的な実行計画を作成した。そこから「切り替え判定基準」のような各工程の完了基準を設定し、すべてのリスクを解決したかどうかを確認できるようにした。

リスクの解決策を考える際は、自らの努力だけでは解決できないような、外部に依存するリスクに特に注意した。最たる例がITベンダーの技術者の確保だ。技術者を確保できるかどうかは、開発会社の受注状況に依存する。10人や20人ならなんとかなるかもしれないが、このプロジェクトではITベンダーの技術者だけで1,000人単位の要員が参加した。B社は早くから要員確保をリスクととらえ先手を打った。東京近郊でメドがつかなければ地方都市にも出向き、いつごろどのようなスキルを持った技術者が何人ぐらい必要といった具体的な要望をあらかじめ開発会社に伝えた。

（２）「あいまい」を徹底排除

B社は要件定義などの際には、解釈があいまいになりそうな不確かな記述を徹底的に排除するという方針を定めた。例えば、新旧システムのデータの関連性を定義する際には、新旧システムのすべてのデータの意味や使われ方、及び双方の関連を、仕様書などに明確に記述した。新システムのこのデータは旧システムのどれに当たるといったことが一目で分かる一覧表を作ったわけだ。これを使って、データを受け渡す旧システムの担当者と、データを受け取る新システムの担当者が共に、両方のシステムのデータ項目を徹底的に理解できるようにした。

一覧表には、各項目の意味や取り得る値を説明する詳細資料を紐付けて管理した。「取引金額」という名称の項目は元本だけなのか利息だけなのか、それとも元本と利息の合計なのか、といったことが分かるようにした。「あの項目がこの値のときはこの項目はこうなる」といった複数の項目に関連するルールも盛り込んだ。

新旧データの関連付けに気を配ったのは、新旧システムでデータベースの構造やデータの持ち方が変わったからだ。右から左へ単純にデータ項目を移せばよいというわけにはいかない。例えば外貨貸付に関連するデータ項目を外為関連のデータベースに保有しているか、融資のデータベースに格納してあるかといった違いである。

新旧システムによる機能の違いについても、データの関連性と同様に一覧表を作り、あいまいな点を排除する方針で要件定義を進めた。このとき、特に注意したことは、新旧システムで機能に変更がない部分についても要件定義書に明確に記述する、ということである。というのも、二つのシステムを一つに統合する場合、「旧システムの仕様」は、少なくとも二つ存在する。旧システムが二つあるからだ。この状態で、もし仮に「現行どおり」という表現を使ってしまうと、二つの旧システムのうちどちらを指すかによって、意味が全く違ってくる。「現行どおり」という記述は、要件を定義

したことにはならないというシステム開発の基本に従って、Ｂ社は要件定義を進めたといえる。

（３）要件や仕様を説明する「ブリッジSE」を配置

Ｂ社は、要件定義書にできる限り詳細な記述を盛り込む工夫を凝らした上で、さらに、要件に対する誤解や理解不足をなくす対策を講じた。最たる例が、遠隔地にあるITベンダーの拠点を訪ね、要件や仕様について説明したり質問に答えたりする「ブリッジSE」を配置したことである。

ブリッジSEは、各拠点の技術者にシステムの仕様を理解してもらうための、Ｂ社とITベンダーとの橋渡し役だ。ブリッジSEは、遠隔拠点の技術者が設計仕様を正しく理解しているか確認したり、ドキュメントに記載のない設計思想を補足したりして、仕様や設計を明確にしていった。各拠点の技術者が問題を発見したり、あるいはほかの拠点の技術者に確認したい事柄が見つかったりしたときは、ブリッジSE同士が連携を取って、問題や質問・回答を共有できるようにした。

Ｂ社がブリッジSEを配置したのは、システム統合の発注者として、ITベンダーに要件の詳細を説明する責任があると考えていたからだ。ITベンダーのすべての技術者が、Ｂ社の業務やシステム開発方針を十分に理解していると考えるのは、現実的ではない。ITベンダーの技術者は自社の業務を知らないかもしれないと考えて、ブリッジSEを設けるなどの対策を講じることが、プロジェクトの成功につながった。

（４）研修で切り替え当日の事務作業の混乱を防ぐ

要件定義に労力をかけて高品質なシステムを作っても、そのシステムを店舗などの社員が使いこなせなければ、システム統合は失敗である。プロジェクトの最終的な成否は、店舗や事務センター、コールセンターなど現場の社員が、新システムを使って新しい事務手順に沿って業務をこなし、顧客にサービスを提供できるかどうかによって決まる。

Ｂ社は、現場の社員が切り替え当日から新システムを使いこなせるように、いくつかの手を打った。例えば、現場のシステム操作を支援する専門チームを店舗に派遣した。新システムを熟知した要員を集め、3人1組の支援チームを結成、切り替え対象店舗に常駐要員として送り込み、現場の事務作業を支援した。支援チームのメンバーは、「この作業がこう変わる」といった具合に、変更点を具体的にアドバイスできるように、事前に新旧システムの操作訓練を受けてから現地に向かった。

Ｂ社は新システムの操作研修も抜かりなく実施した。研修は業務内容に応じて複数のカリキュラムを用意。パート職員などを含む全社員に、職種に応じたカリキュラムの受講を義務付けた。受講してくださいと伝えるだけでなく、一人ひとりの受講状況をツールで管理できるようにした。Ｂ社の店舗網は日本全国に及ぶため、研修用の施設を全国に32カ所設け、どの店舗からも1時間以内で行けるようにした。講習を受ける現場の社員の負荷を少しでも減らすためだ。

研修専門の講師だけで、百人以上を養成してもいる。さらに、新システムの操作訓練のためのシステムを各店舗に導入するなど、現場の社員が新システムの操作訓練に励むことができる仕組みを用意した。

これらの研修・操作支援策には、それなりの費用がかかるが、それはシステム統合を実現するためには必要不可欠なものである。システム統合に必要な費用は、ソフトウェアの開発費やハードウェアの購入費、ソフトウェアのライセンス費用だけではない。Ｂ社は、研修にかかる費用についても、プロジェクトの開始段階で見積もり、予算を確保した。研修場所の手配、支援チームの派遣などについても、早い段階から準備に取り掛かった。

研修策を用意する前提として、Ｂ社のIT部門は、店舗をはじめとする現場の社員をシステム統合プロジェクトに積極的に参加させることに気を配った。システム統合の成功には現場の社員の参加が欠かせないとの考えに基づき、現場の社員に積極的に協力を仰いだ。この点については、IT部門だけでなくＢ社の経営者が率先して、「システム統合の成功にはIT部門と

現場部門の相互協力が不可欠。両者のバランスが取れずに責任のなすり付け合いが始まるとプロジェクトは必ず失敗する」と経営会議などで繰り返し強調し、全社一丸の体制を築いていった。

（５）システム化の方針を全社に周知徹底

徹底した計画と管理の一方で、Ｂ社はプロジェクトに携わる関係者の士気向上策も重視した。「歴史に残るプロジェクトをやり遂げるという強い意志を持って臨んでほしい」。プロジェクトが始まる直前、IT部門長はＢ社のIT部員と関連会社の技術者を前に、自らの思いを伝えた。プロジェクトを成功させるには、共通の目標を掲げ全員が心を一つにする必要があると考えた。

プロジェクトがスタートしてからは、経営者が全社員に対して、システム統合の重要性や目的、基本方針をことある毎に発信した。特にシステム統合は全社プロジェクトである。全社員がその目的を共有できるかどうかが、プロジェクトの成否に直結する。Ｂ社のプロジェクトでは、IT部員、現場の社員、ITベンダーの技術者など、システム統合プロジェクトに携わった関係者全員が目標を共有することができた。そのことが、システム統合の成功につながった。

4.3　「超上流」強化、予算オーバーを減らす

続いては、ある製造業Ｃ社が、システムの設計・開発に着手する前の「超上流」フェーズを強化したケースである。システム開発の発注者としての責任を果たすため、自社におけるシステム化のプロセスを見直し、超上流フェーズに十分時間を費やすことができるようにした。この取り組み自体は、先の二つの事例のような個別のシステム開発プロジェクトとはやや性質が異なるが、システム開発を成功させるためには重要なアプローチである。

システム開発の予算オーバーを減らせ。そもそものきっかけは、Ｃ社のIT部門が、経営者にこう命じられたことだった。この企業では、システム開発にかかる費用の実績が、平均して予算を20％上回っていた。

（１）上流工程の内製化を決断

経営者の期待は、システム開発における予算見積りの精度を上げることだった。この期待に応えるため、Ｃ社のIT部門は、システム化の構想･企画検討や要件定義といった超上流フェーズの作業を、社外のコンサルタント会社などに依頼することをやめ、自前で取り組むことにした。要件定義は発注者の責任であるという基本原則を再認識するところから始めたのである。

自前で取り組むことにした最大の目的は、超上流フェーズに十分な時間を割けるようにすることだった。Ｃ社におけるシステム開発プロセスは、まず「構想・企画」フェーズで、システム化の目的や対象範囲、予算などについて、IT部門と利用部門が議論する。これらが確定したら、要件定義フェーズに進み、IT部門と利用部門が要件を決める。ここまでが超上流フェーズである。要件が確定したら、設計、開発、テストと進める。

予算が超過したプロジェクトについて原因を調べてみると、超上流フェーズに十分な時間を費やしていなかったことが分かった。ボトルネックになっていたのは、超上流フェーズにおける一部の作業を、社外のコンサルタント会社に委託していたことだった。Ｃ社では、設計・開発などはもちろん、業務分析や要件定義などの作業についても、社外に発注する際は、経営会議で承認を得る必要があった。この会議向けに稟議書を作り、会議にかけて承認をもらうまでに最大で3カ月かかっていた。

ビジネスのスピードを考えると、例えば構想・企画フェーズであればどんなに時間がかかったとしても6カ月で終わらせる必要があった。ところが、この6カ月のうち稟議や承認に3カ月かかるので、構想・企画には実質3カ月しか取れないプロジェクトが少なくなかった。システム化の対象範囲の検討や予算の見積りは、システム開発において極めて重要な作業であ

る。ここに十分な時間を確保できず、一部の検討項目について先送りをしたり、ステークホルダの合意を得られないまま次の要件定義フェーズに進んだりしたことがあった。それらが見積り精度の悪化をまねき、結果的に予算オーバーにつながっていた。

要件定義フェーズについても、構想・企画フェーズと似た問題が発生していた。社外に委託する作業の承認プロセスに時間を取られ、一部の要件が未確定のまま、あるいはステークホルダの合意が得られないまま、設計フェーズに進むことがあった。

そこでＣ社は、承認期間を不要とすることで、超上流フェーズ全体の期間は延ばさずに、計画の立案あるいは投資金額の見積り、IT投資効果の分析、要件定義、といった作業にかける時間を増やそうと考えた。これを実現するための施策が、現状分析や業務検討などの「内製化」だ。構想・企画や要件定義の作業を社外に委託せず、自社でこなすことにしたわけだ。承認プロセスを不要とすることで、検討項目の先送りをなくし、ステークホルダから合意を得るプロセスにも十分に時間をかけられるようにした。結果的に、超上流工程の内製化によって、Ｃ社はシステム開発プロジェクトの見積り精度を向上させ、予算超過プロジェクトの数や予算超過額を大幅に減らすことができた。

超上流工程の内製化に際しては、業務コンサルタントを25人前後、自前で育成した。これらのメンバーが社外コンサルタントの代役として、売り上げ拡大や納期短縮、生産コスト削減といった利用部門のビジネス目標を達成するためにITをどう生かせるか、関連する利用部門の担当者と一緒に議論した。

Ｃ社はシステム開発をIT部門と利用部門の共同プロジェクトと位置付けている。例えばプロジェクトの契約時に、利用部門とIT部門の責任者が契約書にサインし、お互いの協力を誓う。利用部門はシステムを使って想定したビジネス効果を出すことに責任を持ち、IT部門はIT投資効果がより大きくなるように努める。システム開発におけるステークホルダとの合意を重視した行動といえる。IT部門と利用部門が自動車の両輪のように同期を取りながらビジネス価値の向上を実現するのが、Ｃ社におけるシステム開発の最終的な狙いである。

（２）ITコストの内訳管理システムを構築

Ｃ社は、予算超過を防ぐ取り組みに先駆け、ITコストの内部構造を可視化するプロジェクトに挑んだ。これも経営者の要請に基づく。Ｃ社のIT部門が経営者から受けたミッションは、予算超過を防ぐことのほかにもう1つあった。それがITコストの可視化である。Ｃ社の経営者は当時、IT部門について、「仕事の内容だけでなくコスト構造も不透明で、何をやっているのか分かりにくい」という不満を抱いていた。

Ｃ社のIT部門の責任者であるCIO（最高情報責任者）は、まずコスト構造を明らかにするところから着手しようと考えた。いきなり予算超過を減らそうとしても、コスト構造が分からなければ、見積りの精度が上がるはずはない。

ところが、コストの見える化は思うように進まない。IT部門の責任者が、個別システムを担当するIT部員に聞いても、なかなか数字が出てこなかった。新システムを開発するプロジェクトのコスト構造については、なんとか把握できたが、稼働後の運用保守費については、特に不透明だった。システムのライフサイクルにかかるコストを管理していなかったのだ。

そこでＣ社のIT部門では、CIOが主導する形で、システムのライフサイクルコストを可視化するプロジェクトをスタートさせた。コスト把握のための専用システムを構築、このシステムを使って、開発費はもちろん運用保守費についても、予算と実績をプロジェクト単位で管理できるようにした。

システム全体のライフサイクルコストが見えるようになると、次はITコストの細分化に挑んだ。開発費ならアプリケーション開発にかかった人件

費、ソフトウェアの購入費、ハードウェアの購入費といった具合に、内訳を押さえていく。「システム開発一式いくら」という大雑把な契約も少なからずあり、これまた作業は難航した。開発費に出精値引きはあっても、保守費用は定価ベースで決まっているなどの問題も見つかった。それでも、ある程度の案件について内訳を明らかにできた。

システムのライフサイクルコストを重視して、それらを可視化するという一連の取り組みと専用システムによって、C社はシステム単位あるいは利用部門単位でシステムのライフサイクルコストをすぐにはじき出せるようになった。これが、先の予算オーバーの削減につながった。

ITコストを可視化し、超上流フェーズに費やす時間を増やしたからといって、プロジェクトを必ず成功できる保証はない。だが一連の取り組みを通じて、C社がプロジェクトの成功率向上に向けた取り組みのスタート・ラインに立てたことは間違いない。

（注1）第4章については、日経コンピュータ2006年5月29日号、2007年5月28日号、2009年3月4日号をもとに編集しました。

（注2）第4章は、（注1）のとおり、原文をもとにして編集しているため、以下に示す用語についてはもとの表記のままにしています。
“ユーザー”、“ベンダー”、“あいまい”、“ほかに”、“と共に”

第５章　実務に活かす原理原則17ヶ条

原理原則17ヶ条が、システム開発プロジェクトを成功に導くための勘どころであることは、第2章～第4章の事例でご理解いただけたことと思う。

17ヶ条を実務の場で活かすには、システム開発に携わるすべての人々の心に17ヶ条を刻み込み、さらに実務の手順の中に17ヶ条の精神を埋め込む必要がある。

本章では、発注者で研修に活用している事例、受注者で研修に活用している事例、及び発注者のプロジェクトレビューにおいて実践している事例を紹介する。

5.1　要求獲得の改善意識の醸成教育への活用

本節では、自動車部品製造業A社における原理原則17ヶ条の活用事例として、「要求獲得の改善意識の醸成教育への活用」を紹介する。各項は、当該企業のプロセス改善担当者による活用報告を、その承諾を得て紹介するものである。

（1）活用の背景 ～エンジニアの改善意識の醸成教育～

良いソフトウェアを作るためには、早く正確に要求事項を掴み、無駄な仕事の発生を抑え、無理のない適切な工期を確保することが必要になる。そのためには、顧客との連携の仕組みを顧客と協議し、構築し、実行することが必要となる。また、協力会社への説明不十分や仕様提出遅れなどに起因して発生するソフトウェアの品質と納期への支障を回避するために、協力会社との連携の仕組みを協力会社と協議し、構築し、実行することも必要である。

しかしながら、顧客の指示どおりにソフトウェアを開発することに追われている状況では、現場において改善意識を持って、顧客や協力会社との連携の仕組みを見直し、再構築することは難しい。これでは、いつまでた

っても悪循環から抜け出せない。

そこで、改善意識を「受け身から攻めの姿勢」に転じてもらうために、ソフトウェア開発のエンジニアを対象とし、以下の視点から「要求獲得（要求管理）に対する改善意識を醸成するための教育」を実施できないか模索していた。

①ソフトウェア開発がうまくいくためには、顧客の要件が正確に早く決まることが最も重要である。

②そのためには、受注者側の仕事のやり方（プロセス）を改善するだけではなく、発注者側の仕事のやり方（プロセス）までも踏み込まないといけないケースが多い。例えば、適度な大きさに作業を分割する、適度な間隔（例えば、仕様変更が頻繁に発生する場合に、変更依頼と次の変更依頼の間にある程度の間隔をあける必要がある。）をとる、質の良い要件を受ける、といった仕事の受け方に改善することが考えられる。

③発注者側の仕事のやり方を改善してもらうためには、発注者側のメリットも考えた「Win-Winの関係」が成立する提案を行い、地道に根気よく、発注者側としっかりとした議論やコミュニケーションを図ることが大切である。

④協力会社にソフトウェア開発を発注している場合、自分たちも発注者側の立場であることを認識し、受注者側とのしっかりとした双方向コミュニケーションを取ることも大切である。

そんな折りに、あるシンポジウムで配布されていた『原理原則17ヶ条』と出会った。そのときの第一印象は「これだ！よくぞ、うまくまとめてくれた！」と琴線に触れる思いであった。過去からよく言われてきたことを、顧客や協力会社との間の役割分担の規範（あり方）として、うまく17ヶ条にまとめていると感動したのである。そこで、これを活用しない手はないということで、先ほど述べた教育への活用に取り組み、有用な結果を確認することができたので、その概要を報告する次第である。

これ以降、『原理原則17ヶ条』を活用した3つの事例を紹介する。

（２）活用事例１ ～「IT化の原理原則17ヶ条」の折りたたみ版の作成と周知～

まず、最初に『原理原則17ヶ条』を現場のエンジニアに読んでもらうことにした。そこで、図5-1に示すように、本の内容をもとにしてA4版両面（全2頁）に圧縮・編集し直し、かつ、その本の見方などの解説も入れて、『原理原則17ヶ条』の折りたたみ版（以下、“折りたたみ版”と呼ぶ）を作成した。これは、四つ折りにして本などに挟み、手軽に持ち歩いていつでも読めることを狙ったものである。また、発注者／受注者欄を使って、チェックリストとしても活用できるように編集した。

この“折りたたみ版”は、自部署のエンジニアだけではなく、社外の知人や社外講演の際に出席者にも配るなどしたが、大変好評であった。

図5-1　「超上流から攻めるIT化の原理原則17ヶ条」の本の折りたたみ版

（３）活用事例２ ～要求獲得の改善意識の醸成教育への活用～

次に実施したのは、「考えてもらう教育」への原理原則17ヶ条の活用である（図5-2）。

教育名	要求管理（獲得）の改善意識の醸成教育				
目　的	① 担当業務における要件定義（管理）の現状を、テキストの「17ヶ条の原理原則と行動規範」に照らし合わせ、今後の自分たちの要求管理の改善活動につなげる ② 人の意見を聞き、自分の考えを深める。また、 決められた時間で所定の成果を出す経験を積む ③ 自分の意見を述べる、人の意見を聞く、質問をするといった基本的コミニュケーションスキルの向上を図る				
講義形式	ワークショップ形式	時 間	2.0 H	その他	事前課題付き
テキスト	IPA／SEC 発刊 「超上流から攻めるIT化の原理原則17ヶ条」 サブテキスト：　IPA／SEC 「経営者が参画する要求品質の確保 ～超上流から攻めるIT化の勘どころ～」第2版				
事前課題	①「 超上流から攻めるIT化の原理原則17ヶ条」の冊子及び折りたたみ版を読む ② EXCELシート「超上流から攻めるIT化の原理原則17ヶ条実施度チェックシート」に、各原理原則／行動規範に対する自分の理解度、重要度、実施度を記入する ③ 同じく、重要と思う原理原則を5つ選定し（TOP5）、順位と選定理由を記入する ④ 上記②③の回答結果を、講師まで送付する				
教育本番	① オリエンテーション（15分）、② 個人検討（10分）、③ グループ討議（45分）、④ 全体討議（45分）、⑤ まとめ（5分）				
フォローアップ	① 要求管理に関する講演会の実施 ・要求管理及び超上流の重要性を経験豊かな講師により語ってもらう ② 教育受講後の実務への適用、意識の変化の追跡アンケートの実施				

図5-2　「考えてもらう教育」のカリキュラム（抜粋）

このカリキュラムに従って、9ヶ月間に、今までに3部署、延べ約60名に対して教育を実施した。3回の教育及びフォローアップの講演会については受講後のアンケート結果から、どれも比較的に良い評価であった。「IT化の原理原則17ヶ条」そのものの評価も高いものであった。

なお、活用事例3とも関連するが、教育受講に際して事前課題（図5-3）を行うよう受講対象者に指示している。

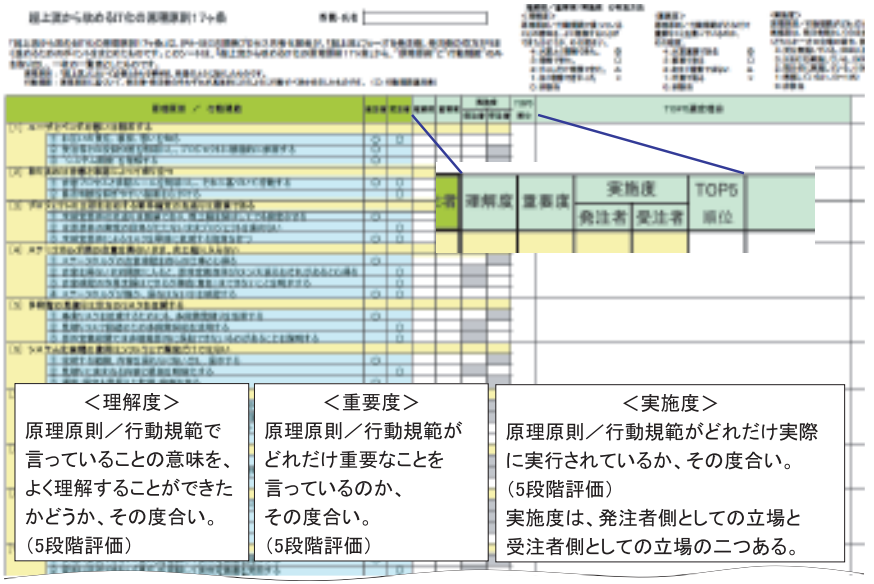


図5-3　事前課題 ～原理原則17 ヶ条の理解度、重要度、実施度、TOP5の選定～（抜粋）

ここでは、参考までに、図5-4のアンケート結果をもとに、そこから読み取れることを以下に示す。これらの結果から、今回実施したカリキュラムは大変有効であったものと評価している。

①教育受講後、「自分の意識に変化があったという人」が、88％であった。

②教育受講後、「IT化の原理原則17ヶ条」を「意識的に活用している人」は、83％に上る。

③意識的に活用している人のうち、「具体的な効果が出たと思った人」は49％、「具体的な効果まではいかないが何らかの良い変化への兆候を感じる又は、今後効果が期待できると思った人」は29％であった。

④特に、以下に示すような感想が多く目立った。

(a)自分のいる立場が明確に認識できるようになったことにより、相手の立場で考えることができ、それがさらに、相手の立場、思いの理解につながり、相手への気遣い、心配りができるようになった。

(b)発注者側の立場として、受注者との円滑なコミュニケーション、明確な指示、十分な説明責任を果たすことが、受注者／発注者双方が幸福になる道であることに気づいた。

(c)「IT化の原理原則17ヶ条」を引用して、ソフトウェア開発の留意点としている人が多い。

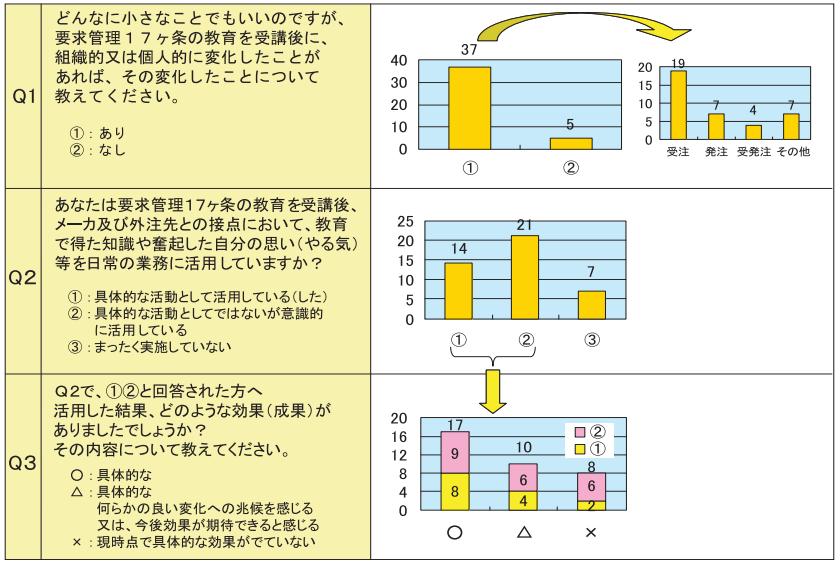


図5-4　教育受講後の意識の変化、業務への適用状況調査アンケート結果（抜粋）

（４）活用事例３ ～要求管理プロセスの強み／弱みを表すメトリクスとしての活用～

活用事例1でも述べたが、“折りたたみ版”は、それ自体がチェックリストになっている。しかし、単なるチェックリストとしてではなく、「IT化の原理原則17ヶ条」自体がプロジェクトの強みや弱みを表すメトリクスとして活用できないかどうかという狙いでも作成した。つまり、経験別やプロジェクト特性（ビジネスモデルの特徴など）によって、「IT化の原理原則17ヶ条」の各項目をどれだけ意識しているのか、どの程度実際に実施できているのかを「見える化」できるのではないかと考えたからである。

そこで、活用事例2における教育の実施の際に、以下の3つの仕掛けと工夫を行った。

（A）「IT化の原理原則17ヶ条」の理解度、重要度、実施度プロファイルの作成

1つ目の仕掛けは、対象プロジェクトにおいて、「IT化の原理原則17ヶ条」の各原理原則の理解度（4：大変よく理解できた、3：理解できた、2：少し理解できた、1：全く理解できなかった、0：非該当）、重要度、実施度をみることにより、対象プロジェクトの要求管理プロセスの強み／弱みを見つけることである。

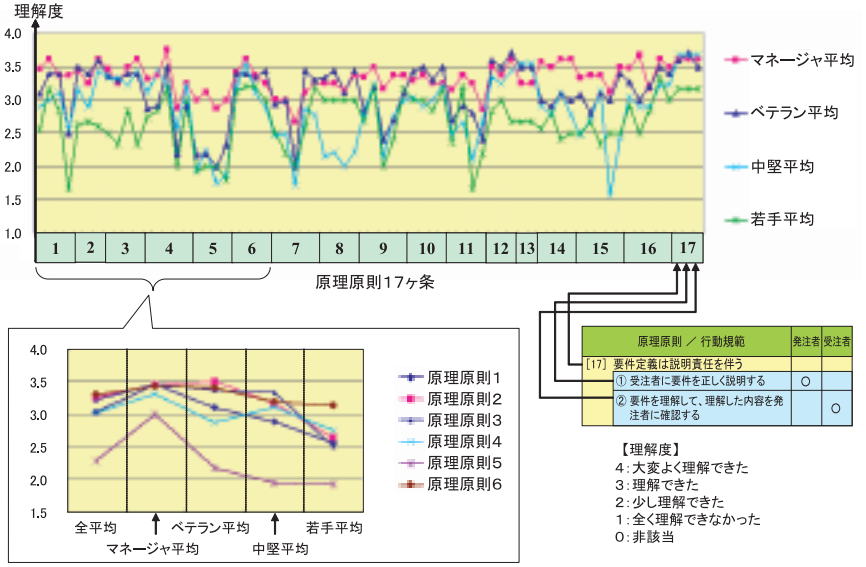


図5-5　事前課題の回答例 ー理解度の場合

そこで、教育受講者に事前課題で記入してもらった各原理原則の理解度、重要度、実施度を集計・グラフ化（プロファイルとして作成）した。例えば、グラフ図5-5は、理解度のプロファイルである。このプロファイルから

は、原理原則［5］、［7］、［9］、［11］、［14］、［15］が低い傾向を示していること、また、原理原則［5］「多段階の見積りは双方のリスクを低減する」のように、全体的にマネージャ層では理解度が高いが、ベテラン、中堅、若手と経験が少なくなるほど理解度が低くなることがわかる。

（B）教育本番ワークショップの際のグループ編成の経験（職位）別化

2つ目の仕掛けは、教育本番ワークショップの際のグループ編成を、マネージャ、ベテラン、中堅、若手の4階層に分けることにより、業務経験（職位）別にグループ討議結果に差が識別できるようにしたことである。

しかしながら、事後分析の結果、業務経験（職位）に対しては、細かいところで選択の多少のバラツキはみられるものの、特に大きな差はないことがわかった。

また、事前課題で選択されたTOP5の分布とグループ討議で選択されたTOP5の分布でも、大きな差はないことがわかった。これは今回教育を実施した3つのプロジェクト（すべて部が違う）は、もともと同じ一つの部から分化した部署であり、対象製品もほぼ同じドメインであることから、今回は差異が小さかったものと考えている。

なお、TOP5として非常によく選択された原理原則を、以下に列挙する（“P”は「原理原則」の意味である）。

P2　：取り決めは合意と承認によって成り立つ
P3　：プロジェクトの成否を左右する要件確定の先送りは厳禁である
P8　：システム化の方針・狙いの周知徹底が成功の鍵となる
P10：要件定義書はバイブルであり、事あらばここへ立ち返るもの
P16：機能要求は膨張する。コスト、納期が抑制する
P17：要件定義は説明責任を伴う

これらを見ると、「発注先との間で抱えている要求に関する問題」に集中していることがよくわかる。実際、これらは、現場の担当者が直面している、特に切実な課題なのであろう。

また、TOP5にP5、P6、P7があまり選択されなかったことから、コストに関する関心（意識）が比較的低いということがわかった。

（C）「IT化の原理原則17ヶ条」の各原理原則間の相関図の作成

3つ目の仕掛けは、原理原則が17ヶ条もあると、それぞれの原理原則の間で関連が深いもの（従属性があるもの）や浅いもの（比較的独立しているもの）があると考え、A社では、図5-6のように分類してみた。

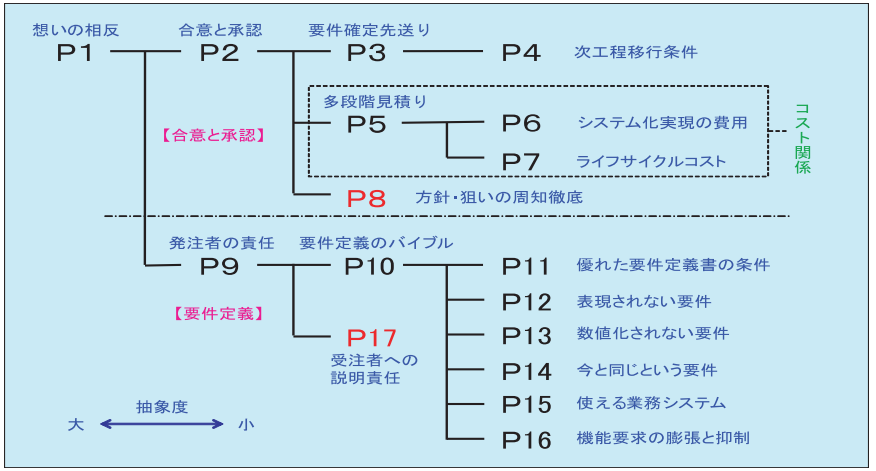


図5-6　原理原則の相関図

ルート（根源）は、発注側と受注側との間の想いの相反にあると考え、ここにすべてのことが集約されるとして、残りの原理原則を、

・“合意と承認”のグループ：発注者と受注者が仕事の進め方について取り決めておくべき留意点
・“要件定義”のグループ：その取り決めに従って実際に仕事を進めていく上で果たすべき役割と責任、及び要件の明確化のために押さえるべき留意点

という2つのグループに分け、関係を展開してみた。図では、左にいくほど抽象度が高く、右にいくほど具体性が高くなるようにし、4層のツリー構造

とした。

また、P5、P6、P7をコスト関係（“コストの三兄弟”と呼んでいる）としてグルーピングした。

図5-7(左)は、(3)項で説明した「教育受講後の意識の変化、業務への適用状況調査アンケート結果」から、どの原理原則に注目して活用したのかを集計したものである。図5-7(右)は、図5-6の上に(4)項(B)で述べたTOP5のグループ討議結果を、選択された数の多さに応じて色分けしたものである（図中の◯の中の数字は図5-7(左)の値)。この図から、17ヶ条の原理原則の構造と、原理原則に対する強み／弱みを一緒に鳥瞰することができる。

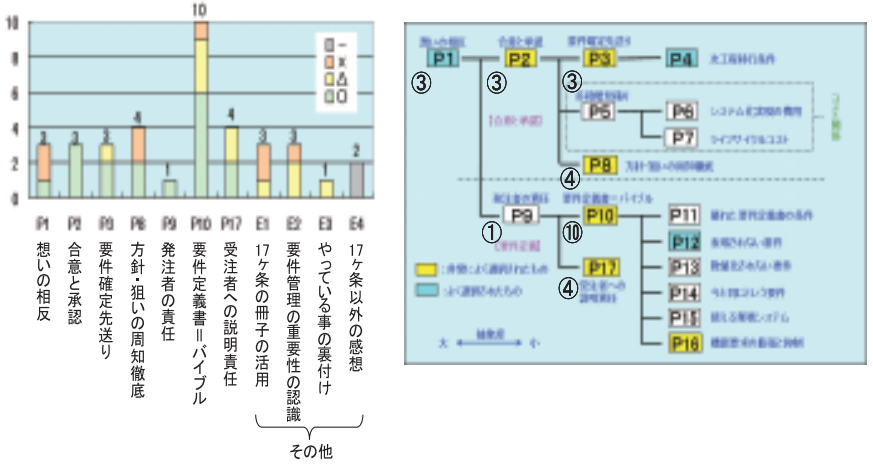


図5-7　原理原則の相関図に、事前課題のTOP5をプロットしたもの

（５）まとめ ～パートナーとしての原理原則17ヶ条～

「IT化の原理原則17ヶ条」を活用した教育により、ソフトウェア開発のエンジニアが心がけることとして、

①発注者と協力した改善活動の大切さ（発注者への積極的な働きかけ）

②発注者の立場に立って、受注者への責任を果たすことの大切さ

③自律的にできることから、日々改善を行うことの大切さ

などの認識を深めることができた。これにより、「IT化の原理原則17ヶ条」が、「考えてもらう教育」として要求獲得の改善意識の醸成教育に活用できることを実証することができた。

また、弱点が「コストに係る要求管理の押さえ」にあることがわかるなど、「IT化の原理原則17ヶ条」が要求管理に対するチェックシートとして使えることもわかった。

最後に、参考として、この活動報告を提供していただいたプロセス改善担当者の感想を提供する。

「私は、『原理原則17ヶ条』と、その“折りたたみ版”(図5-1）を常に持ち歩いている。通勤、出張、会議などにおけるちょっとした空き時間に眺めている。眺めるたびに、新たな様々な考えが浮かんでくる。そこから、抱えている課題の解決のヒントが見つかることもある。今では、頼もしい友である。」

5.2 「気づき」と「体験」共有のための社内研修への活用

本節では、情報通信機器製造業Ｂ社における原理原則17ヶ条の活用事例「『気づき』と『体験』共有のための社内研修への活用」を紹介する。各項は、当該企業のSE共通技術部門担当者による活用報告を、当該企業の承諾を得て行うものである。

（１）活用の背景

Ｂ社では、2006年度から超上流（特に要件定義）工程の品質向上策の一環として、「ビジネス・アーキテクト」(*1)(以下「BA」と略）の育成に力を入れてきた。BA育成のための研修体系は、6週間にわたる集合研修（通称「BA研修」)で以下のような理論や技術を身につけるよう構成されている。

①基礎理論：ソフトウェア工学、情報システム工学などのエンジニアリング的視点や知識を習得する。

②ヒューマン系スキル：現場に対するフィールドワーク、インタビューなどにより、事実／実態を正しく把握する能力を習得する。
③プロセス系スキル：対象業務そのものや、その業務が抱える問題／目的などをモデルとして構造化する能力、設計モデル作成の基本的な作法を学ぶ。
④プロマネ系スキル：リーダーシップやファシリテーションなどの人間系プロマネ能力を高める。
⑤ケース演習：日常業務を想定したケース・スタディによってチームで考え抜く能力を高める。

このうち、主に「⑤ケース演習」の一環として、上流工程の重要性を再認識するため、また他人の失敗体験を分析して教訓を共有するため、原理原則17ヶ条を活用した研修を実施している。

（２）研修の概要

①事前課題

まず、演習実施の2週間ほど前に、原理原則17ヶ条の小冊子と「自分の経験したプロジェクトでの成功・失敗例のうち、17ヶ条に関連したものを抽出してくる」という事前課題が与えられる。

受講者は、自分の体験した成功・失敗例が17ヶ条のどれに該当するかを考えるが、その前に、原理原則17ヶ条の趣旨を十分に理解する必要があるため、知らず知らずのうちに小冊子を熟読することになる。これが、研修企画者の第一の狙いである。

そして、受講者は演習前日までに、図5-8のようなフォーマット（アンケート形式）に事前課題の作成を行い当日に臨む。

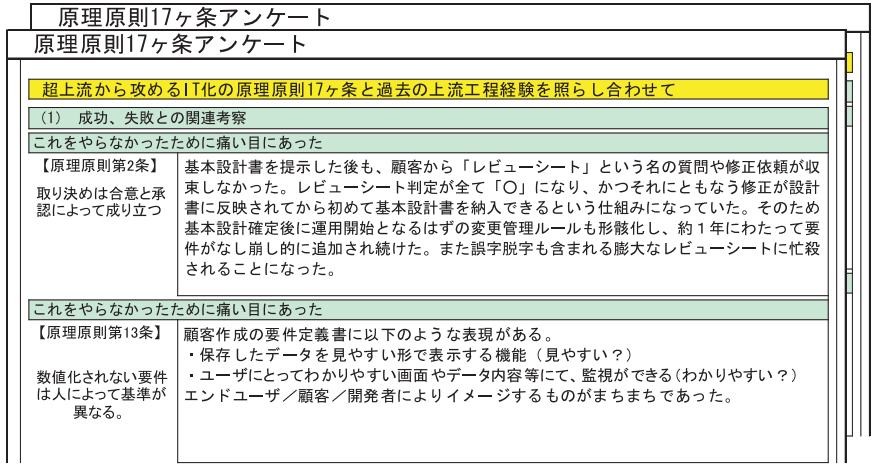


図5-8　事前課題：原理原則17ヶ条アンケート

②グループ演習

当日は、まず4～5名のグループに分かれ、各自作成してきた事前課題を発表する。その際、アンケート結果だけではわかりにくいプロジェクトの状況等も補って説明し、情報共有する。

次いで、グループ内で最も興味深い失敗事例を選出し、討議をして以下の成果物を作成する。

1）プロジェクト（ユーザ）名
2）商談（プロジェクト）の経緯
　（ポイントを簡潔に）
3）失敗の現象→QCD [*2]への影響
4）原因（第ｎ条）
　・事例の当事者が挙げた原因
　・討議の課程で発見された原因
5）防止策
　（こうしておけば失敗しなかったかも知れない）

6）標語「気をつけよう　×××××××　△△△△△」
（図5-9の研修風景（ホワイトボード）参照）

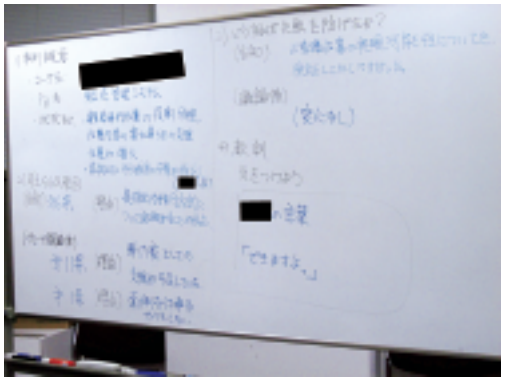


図5-9　研修風景（ホワイトボード）

このとき、4）の「討議の過程で発見された原因」や 5）の「防止策」を導き出す過程で、事例の当事者が新たな「気づき」を得ることが研修企画者の第二の狙いである。また、事例の当事者以外が類似状況に遭遇した際のヒント、教訓を得ることが第三の狙いである。そして、その中の最大の教訓を端的な標語で表現することによって、その印象をより深く心に刻むことも狙っている。

③代表者による成果発表

グループ演習が終わると、クラス全体で各グループの代表者の成果発表を聞く。ただし、発表者は事例の当事者以外が行うこと、発表順はその場で決めることなど、最後まで緊張感を保つ工夫も施している。

成果発表後、質疑応答の時間を設け、最後に講師が講評を行って終了となる。発表グループにとっては、グループ内では出なかったような原因の指摘や防止策の提案などがあり、さらに新たな「気づき」が得られる。また、発表グループ以外にとっては、ヒント、教訓の共有化はもちろん、他グループの検討過程・結果とベンチマークすることによって、自分たちの討議の質の検証にもなる（図5-10の研修風景（成果発表）参照）。

図5-10　研修風景（成果発表）

（３）研修から見えてきたもの

ところで、研修を積み重ねることで見えてきたことがある。それは、図5-11の事前課題集計結果に見られるように、17ヶ条の中でも条文によって、事例の抽出にいくつかの傾向があるということである。

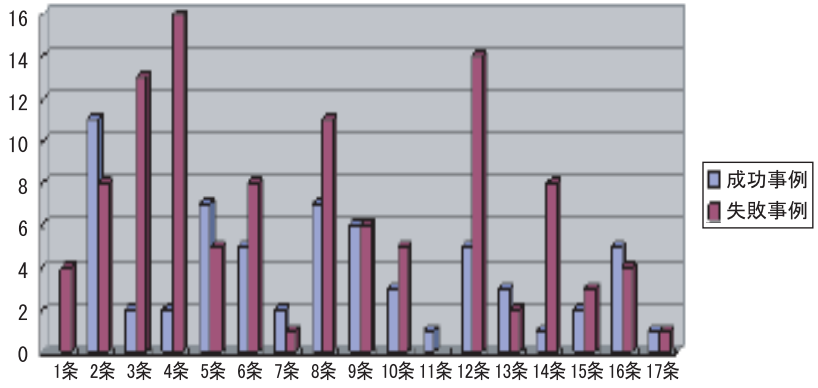


図5-11　事前課題集計結果

①全体的に成功事例より失敗事例が多い（37:63の割合）が、原理原則[2]、[5]、[16]は、ある程度の件数がある割に成功事例の方が多い。

■原理原則[2]　取り決めは合意と承認によって成り立つ
■原理原則[5]　多段階の見積りは双方のリスクを軽減する
■原理原則[16]　機能要求は膨張する。コスト、納期が抑制する

これらはユーザ、ベンダ間の「契約」「取り決め」といった「公式プロトコル」に関することであり、穿った見方をすれば、何度も痛い目に遭っているから用心していると見ることもできる。

②①とは逆に、ある程度の件数がある中で失敗事例の割合が圧倒的に高いのが、原理原則[3]、[4]、[12]、[14]である。

■原理原則[3]　プロジェクトの成否を左右する要件確定の先送りは厳禁である
■原理原則[4]　ステークホルダの合意を得ないまま、次工程に入らない
■原理原則[12]　表現されない要件はシステムとして実現されない
■原理原則[14]　「今と同じ」という要件定義はありえない

これらは「合意」というキーワードで括れる条文であり、①との対照で「公式プロトコル」のレベルではない「実質的な」合意がいかに難しいかを端的に表していると思われる。

③100件以上の事例が集まった中で、極めて少ない事例しかないのが、原理原則[7]、[11]、[17]（それぞれ、3、1、2件）である。

■原理原則[7]　ライフサイクルコストを重視する
■原理原則[11]　優れた要件定義とはシステム開発を精緻にあらわしたもの
■原理原則[17]　要件定義は説明責任を伴う

これらは、一見、当たり前のことを言っているように見えるため、わざわざ事例として取り上げるのは難しかったのかも知れない。ただし、条文の真意は奥深いものがあるので、ぜひ再読してほしいと考える。

（４）まとめ

この他にもＢ社では、BA研修と同様に、成功・失敗体験の教訓を現場に活かすための大規模な発表会である「プロジェクト事例研修会」や「生産革新事例研修会」などの取り組みを行っている。2005年度に社内の営業、SE全員に17ヶ条の小冊子を配布したこともあって、最近はこれらのプレゼンの中でも「17ヶ条で言えば、××条を意識していなかったための失敗と言えるでしょう」といったフレーズが発表者から自然に出てくるようになるほど浸透してきていると感じている。これら「超上流」のマインドと、これらを実現するためのエンジニアリング系、ヒューマン系などの技術をさらに磨いて、システムによる経営、業務の価値実現を目指していきたい。

（＊1）ビジネス・アーキテクト
お客様顧客の真の要求を抽出・分析し、本質をとらえたビジネス要件定義を行い、次にビジネス要件をシステム要件に展開し、システム仕様として具体化することのできる人材。

（＊2）QCD
「Quality（品質）」「Cost（費用）」「Delivery（納期）」の頭文字。

5.3　プロジェクトレビューにおける実践

原理原則17ヶ条を実務で活かすためには、その精神を理解するだけでなく、日々の開発、保守業務の中で実践できる仕組みとすることが重要である。

プロジェクトレビューは原理原則17ヶ条全般を具現化する手法であり、特に、「原理原則[2] 取り決めは合意と承認によって成り立つ」、「原理原則[4] ステークホルダ間の合意を得ないまま、次工程に入らない」等を実現するために有効である。

プロジェクトレビューは、開発品質を向上させたい大手企業などを中心に、システム開発管理手法として制度化され実施されている。

本章では、プロジェクトレビュー制度の概要を説明すると共に、完成度の

高い事例として大手建設会社のプロジェクトレビュースキームを紹介する。

5.3.1　プロジェクトレビューの概要 (*1)

（１）開発工程に合わせた公式プロジェクトレビューの設定

プロジェクトレビューはシステム開発工程に合わせて設定するのが基本である。エンタープライズ系といわれるシステム開発の場合、その開発工程は大きく次の①～⑥の流れで行われる。

①「起案される」(構想・稟議決裁工程)
②「システム化対象業務と、いつまでに実現するかを決める」(システム開発計画工程)
③「何をどう作るかを決める」(システム要件定義工程)
④「設計・構築する」(システム構築・プログラミング工程)
⑤「検証する」(システムテスト工程)
⑥「利用者に提供する」(本番移行工程)

（２）システム開発の各工程は、次の工程のために行われる

システム開発におけるそれぞれの工程は、次工程のために行われる。つまり、前の工程のアウトプットが次工程作業のインプットとなり、次工程作業の品質を決定付けることになるため、当然のことながら、次工程への説明責任を前工程の担当者が負うことになる。

システム開発においては、次工程の作業に必要な事項を充足し、その内容を次工程に説明する姿勢で進めることが成功の鍵である。次工程に進むプロジェクトレビューの場で、両工程のステークホルダが参加して成果物の品質を押さえることが極めて大切である。

（３）工程が変わるところにプロジェクトレビュー

各工程の作業を進めると、既決内容ではうまくいかないことがたくさん起こってくる。漏れや変更、勘違い、了解違いも発覚する。加えて、権限者の考えが変わるなどということも起こりうる。次工程に進む節目で、こういった既決内容の変更について取り扱いを決め、ステークホルダや責任者の了解、承認も取り付けなければならない。

このように、一つの工程が終了する節目は大変重要なポイントになっている。プロジェクトを次工程に進めるにあたっては、作業やアウトプットの品質、解決しなければならない課題や企て、未決事項やリスクや問題を明らかにして管理しなければならない。

（４）問題、課題は把握しにくい

日々、刻々プロジェクトの状況は変わる。リアルタイムでセンサーが働いて、さまざまなアラームが出て、はじめて、管理者が助け舟を出せる。ありとあらゆる課題、問題がリアルタイムにつかめ、即座に評価され、対処策が承認され、手当てがなされるようになれば、「止まらない、戻らない、膨らまない」プロジェクトが実現できる。しかし、システム開発は人間が作業し、人間が管理する部分が多いだけに、すべてをデジタルな数値で見える化することは難しい。

企画担当者、開発担当者、ステークホルダ等に期待される役割と責任をプロジェクトレビューの場で、タイムリーに明らかにし共有することで、担当者の抱え込み、ステークホルダの決め込み、責任者の失念等を排除し、目標と現実との差分から問題・課題を明らかにして、工程毎に期待されるフォーメーションとアクションを実現することが可能になる。

（５）臨機応変のプロジェクトレビュー

プロジェクトの健全性をタイムリーに確保するためには、工程毎に設けるプロジェクトレビューとは別に、状況や内容に応じて臨時のプロジェクトレビューを召集することも必要になる。システム開発プロセス規定に、どういう場合に臨時のプロジェクトレビューを開催するか、危機管理の一環として決めておくことが肝要である。

（６）プロジェクトレビューを制度化し、継続的に見直す

システム開発は人手に頼った仕事である。人間でなければできない価値

を生み出す仕事でもある。システムができなければ何も実現されない。システム開発は、ビジネスに命を与える仕事といってもよい。

プロジェクトレビューは、システム開発に携わるすべての人たちの役割と責任を明確化し、担当者の能力を引き出し、人にゆだねられた業務の水準を引き上げ、最高の価値を実現する仕組みであり、システム開発の業務として制度化しておく必要がある。

一方、制度を作ってルールとしただけでは時間のムダである。また、レビューばかりしていたら仕事にならない。道路の信号機も設置しすぎれば、渋滞の原因になる。適切に設置されていても、時間によっては点滅に変えて、交通がさらに円滑になるように、その時、その場所に合わせて運用しなければならない。

プロジェクトレビューを制度化したら、何より不断の見直しが必要である。始めてみるとわかってくることがある。やってみるとよりよいやり方が見えてくる。要件定義と一緒である。

（７）警備８　捜査２

しばらく前の日経新聞に、堺屋太一さんが日本の警察のあるべき姿は「警備8、捜査2」であると書かれていた(*2)。「警備をしっかりすれば犯罪が減少し、事件が起きてから捜査に追い回されて犯罪対策が後手、後手になることはなくなるでしょう。」といった主旨のご意見であった。

これはシステム開発にも当てはまる。問題は、大きくなる前に、早め早めに芽のうちに摘み取る。これが開発業務の仕組みとなっていることが大切である。プロジェクトレビューでシステム開発の警備8、捜査2を実現していただきたい。

（＊1）「実務で役立つプロジェクトレビュー」(菊島靖弘著、翔泳社刊）より
（＊2）「警備8、捜査2」 日経新聞2003年９月１日掲載の「領空侵犯」より

5.3.2　大手建設会社におけるプロジェクトレビュー

ここに紹介するプロジェクトレビュースキームは、超上流の企画立案工程から、運用工程までもスコープに入れた網羅的、体系的なものであり、完成度の高い模範的な内容となっている。

（１）プロジェクトレビュー一覧

大手建設会社におけるプロジェクトレビューの全体像は、次の一覧表のとおりである。大きな特徴は、各工程で「業務主管部門」と「情報システム部門」の役割と責任を定めている点である。

システム開発の目的は、ビジネスの遂行に必要となる業務を実現することである。そのために、両部門はそれぞれの役割を担い、責任を果たさなければならない。レビュー対象事項を、それぞれの立場、視点から吟味し、確認し、権限を明確にした上で承認しながら進めるという、極めて合理的で洗練されたプロジェクトレビューによって、開発品質を向上させている。

◎:承認　○:確認

	工程	PR名称	目的	承認者(確認者)						
				業務所管部門		情報システム部門				
				部門長	開発責任者	部門長	開発責任者	運用責任者	保守責任者	予算管理責任者
1	システム化計画	システム化計画レビュー	・システム化計画の確認	○	○	○	○			
2	要件定義	要件定義レビュー	・要件定義内容(RFP)の承認		◎		◎	◎	◎	
		システム開発計画レビュー	・システム開発計画の承認	◎		◎		◎	◎	○
3	基本設計	基本設計レビュー	・基本設計内容の承認		◎		◎			
4	詳細設計									
5	プログラム作成 単体・結合テスト	システムテスト計画レビュー	・結合テスト結果の確認 ・システムテスト計画の確認		○		○	○		
6	システムテスト	運用テスト計画レビュー	・システムテスト結果の承認 ・運用テスト計画の承認		◎		◎	◎		
7	運用テスト	カットオーバーレビュー	・運用テスト結果、移行・展開・教育計画の承認 ・カットオーバーの承認	◎		◎		◎	◎	

図5-12　プロジェクトレビュー一覧

（2）プロジェクトレビューの流れ

以上のように制度化された大手建設会社のプロジェクトレビューは、ベンダーへの提案、社内稟議、サービスイン前後の移行・展開も含め、次の図のとおりに運用されている。

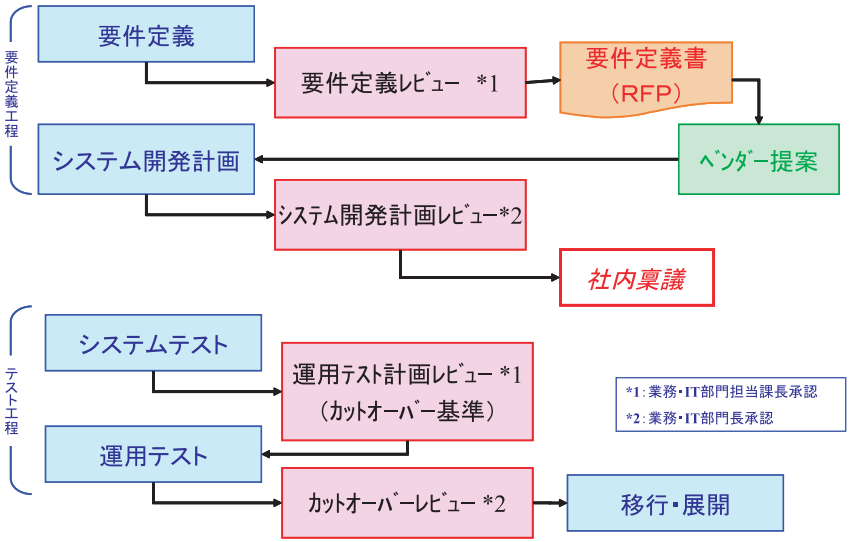


図5-13　主要プロジェクトレビューの流れ

第６章　実践！　原理原則17ヶ条
～東京証券取引所arrowheadプロジェクトでの取組み～*1

株式会社東京証券取引所（以下、東証）は2010年1月4日、新・株式売買システム「arrowhead」を稼働させた。株式売買システムは社会インフラ的存在であり、非常に大規模なシステムである。その性格上、高い信頼性と共に、近年では注文受付通知をミリ秒単位で行う高速処理が求められる。それらの要件を実現すべく同社IT開発部はarrowheadの開発に全力を注ぎ、要求通りの品質とスピードを実現した。また、arrowhead開発プロジェクトを成功に導いた裏には、IPA/SECがまとめた「経営者が参画する要求品質の確保～超上流から攻めるIT化の勘どころ～第2版」の存在があった。

（1）「IT化の原理原則17ヶ条」を行動規範に

株式売買システムに求められるのは、高い信頼性と可用性、高速性である。arrowheadの開発は、データを三重化して高信頼性を確保すると共に、99.999%以上の可用性（5年間で10分程度の停止時間）と、注文受付通知のレスポンスを10ミリ秒以下で処理する（稼働したarrowheadは平均2ミリ秒という高速処理を達成）等、高い目標を掲げて2007年初頭にスタートした。

陣頭指揮を執ったのは、株式会社NTTデータで大規模システムの開発に豊富な経験を持つ鈴木義伯常務取締役（当時、現専務取締役）。2006年2月に東証のCIOとして招聘された鈴木氏は、IT開発部のメンバに「経営者が参画する要求品質の確保～超上流から攻めるIT化の勘どころ～第2版」（以下『超上流』と表記）を配った。「超上流」は、システム開発プロセスにおける要件定義の重要性に着目して、システム開発を成功に導く意図の下に作成されたものである。その内容は、巻末付録「超上流から攻めるIT化の

＊1　SEC journal誌21号（2010年6月）のSEC成果活用事例紹介記事をそのまま掲載させていただいた。

原理原則17ヶ条」(以下、『原理原則』）に簡潔にまとめられている。

鈴木CIOは、3年間強にわたる開発プロジェクトにおいて『超上流』の考え方をプロジェクトチームが実践すべく指示を繰り返した。IT開発部マネージャーの田倉聡史氏は、「鈴木常務の指示は『超上流』に書かれていることに即したものであり、arrowheadの開発は『超上流』をバイブルとして位置付け、『超上流』に書かれている多くのことが実践されています」と振り返る。

（２）発注者が要件定義に責任を持つことを貫く

以下、「arrowhead」のプロジェクトにおいて、『超上流』がどのように活用されたのかを見ていく。

arrowheadの開発にあたって鈴木CIOが強調したのは、「要件定義は発注者である東証が責任を持つ」ということである。それは、「要件定義は発注者の責任である」という原理原則第9条の実行にほかならない。

一般に要件定義書は、どのようなシステムを実現したいかを発注者が作成するものだが、洩れや誤解の無い要件定義書の作成は困難であり、あいまいな記述が存在することが多い。要件定義書の記述があいまいでは（いわゆる「行間がある」)、開発工程の下流に位置するテスト段階になってから、「思い描いていたものと違う」ことが表面化し、情報システムの品質や納期に影響をもたらすこととなる。

従来、東証のIT開発部が作成していた要件定義書も、やはり「行間が多い」記述内容だった（田倉氏)。しかし今回、東証は受注者に対して「『要件の行間を読め』ということを要求してはいけません」(原理原則第17条の本文）という原則を実践したのである。arrowheadの要件定義書をはじめとするドキュメントの記述は詳細にわたった（図6-1)。その結果、プロジェクトチームが作成したドキュメントの総量は、ベンダを選定するために作成したRFP（提案依頼書）が1,500頁、機能要件や非機能要件を記述した要件定義書が1,800頁、更に外部設計書を含めて合計4,000頁にとなった。

「プロジェクトの成否は要求定義書で決まるものです。非機能要件を含めてきちんと書くことは難しいものでしたが、要件定義書の作成は発注者の責任で行う方針の下、自分たちで書ききりました」と田倉氏は語る。また、すべての要件変更の管理は鈴木CIOのチェック、承認を経てから行った。「は」を「を」に変更するといった文言の修正も例外ではない。要求変更に関する書類のすべてに目を通す作業は膨大なものである。それを行ったのは東証でも初めてのことだった。原理原則第10条には、「要件定義書はバイブルであり、事あらばここへ立ち返るもの」とある。発注者も受注者も幾度となく立ち返るバイブルだからこそ、その記述内容には厳密さが求められる。だからこそ、要求変更にはCIO自らがチェックしたのである。

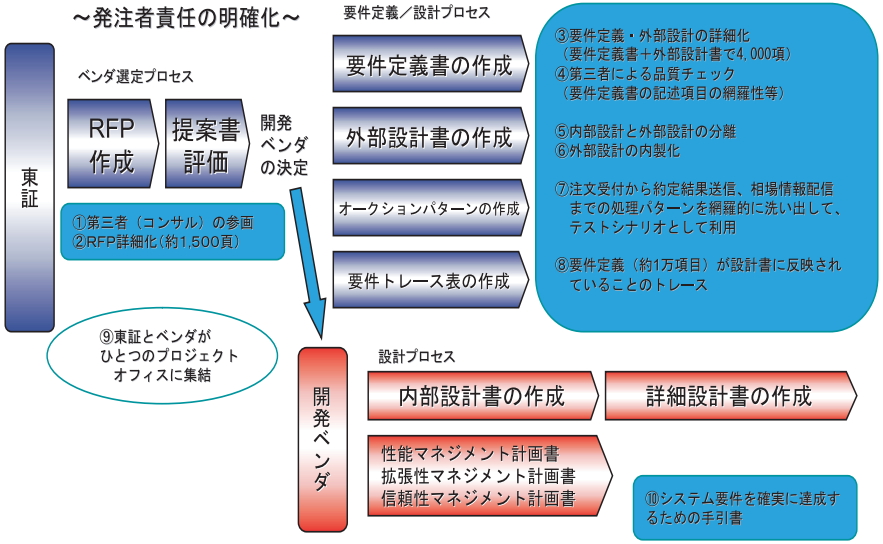


図6-1　次世代システムにおけるプロセス改善の取組み（発注者責任の明確化）

（３）ステークホルダとの合意確認に大きく配慮

ITベンダや社内ユーザ、社外ユーザ等ステークホルダと合意確認をすることは、情報システムの開発を成功させるための重要なポイントである。原理原則第4条は「ステークホルダ間の合意を得ないまま、次工程に入らない」とある。arrowhead開発プロジェクトチームは、ステークホルダとの合意確認をとても大切にした。

例えば、arrowheadプロジェクトでは、東証とベンダの開発拠点を1つのビルに集約し、互いに行き来しやすくすると共に工程の節目ごとに両社の社長を交えたミーティングを実施する等、緊密にコミュニケーションを図る体制を整えた。また、社内の業務部門である株式部の社員にもプロジェクトチームのメンバとして加わってもらった。従来は、IT開発部で要件定義に取り組み、要件定義書が出来上がった段階で業務部門に提示していたが、今回は業務部門が正式に要件定義書を確認する前に、業務部門と一体となって要件定義に当たったのである。それは、「業務担当部門とIT部門とが協力し合って、決めるべきことをきちんと決める」（原理原則11条の本文）の実践にほかならない。

更に、外部設計書や定義書が出来た段階（arrowhead稼働開始3年前）でarrowheadと接続する社外ユーザ（証券会社）に設計書を開示したり、結合テストの段階で証券会社にパイロットユーザとしてテストに参加してもらったこともステークホルダと合意確認を取るという原理原則第4条を踏まえたものだ。

（４）不具合を残さないフィードバック型Ｖ字モデルの実践

最後に、今回のプロジェクトにおいてIT開発部が実践したフィードバック型Ｖ字モデルと呼ばれる開発モデルに触れておきたい。この開発モデルは、各工程において前工程までに作り込まれた不具合を即時に分析して、①現在の工程で対処すべき事象が発生したら次の工程に進まずにその工程で対策を考える、②不具合が生じた原因が上流工程に起因するものだと判明したら上流工程に戻る、ことを指している。不具合を後の工程に残さないことで、情報システムの品質確保と稼働時期の厳守が達成出来るからだ。

フィードバック型開発モデルを実施するにあたってIT開発部は、要件定義書に記述した要件定義が基本設計及び詳細設計書等に反映されているかどうかを確認するために、1万項目に及ぶ要件トレース表を作成し、要件トレース表と設計書とを突き合わせる作業を行った。その作業で不具合が見つかった場合、設計書の問題であれば設計書を修正し、要件定義にあいまい性があったことが不具合の要因であれば要件定義書を修正した。この取り組みは、原理原則第3条「プロジェクトの成否を左右する要求確定の先送りは厳禁である」の実践である。

要件トレース表の作成と同時にIT開発部は、テストケースの作成を並行して行った。これは、「W字モデル」と呼ばれる開発プロセスである。テスト工程に入る前の上流工程においてテストケースを作成することによって、要件の洩れや矛盾を発見することが出来、修正工数を削減することにつながる。「必ずしもすべての機能について行う必要はありません。arrowheadプロジェクトではシステムの中枢処理である、売り注文と買い注文をマッチングさせる処理を対象に実行しました」と田倉氏は語る。東証のIT開発部は、開発スケジュールと人的リソースを考慮しながら、要件定義に発注者が責任を持つという『超上流』の基本を貫いた。それがarrowheadの開発を成功に導いた最大の要因といっても過言では無いだろう。

参考資料
上流工程での品質確保における発注者責任の実現
（東京証券取引所arrowheadプロジェクトの事例）*2

株式会社東京証券取引所　IT開発部マネージャー
田倉　聡史

システム開発プロジェクトの成否を分ける重要なカギは上流工程の品質である。

上流工程での品質作込みにおいて発注者の果たすべき役割は大きい。

本稿では、東京証券取引所（以下「東証」）のarrowheadプロジェクトにおける発注者側の品質確保の取り組みについて紹介する。

1. arrowheadプロジェクトの概要

arrowheadは2010年1月4日に稼働した東証の新・株式売買システムの呼称であり、①証券会社からの注文の受付、②受け付けた注文のマッチング、③注文受付結果、マッチング結果の証券会社への通知、④注文情報及びマッチング結果情報の情報ベンダ等への配信、⑤東証社員による注文のリアルタイム監視のためのデータ生成、等の処理を行っている。

稼働時の証券会社からの注文データ処理容量は4,600万件／日、注文受付にかかる処理時間は1件当たり平均2ミリ秒となっており、世界最高水準の処理性能となっている。

2. 上流工程完璧主義

arrowheadの開発プロジェクトにおいて、全メンバに貫かれた最も重要な考え方を顕著に表した言葉がある。

それが「上流工程完璧主義」である。

「上流工程完璧主義」の大前提となるのは、東証が作成する要件定義書及び外部設計書（接続先システムとの接続仕様書、画面や台帳の設計書等）がプロジェクトのバイブルであり、要件定義書等に明確な記載がない事項は実現されないという考え方である。

arrowheadプロジェクトにおいて東証が作成した要件定義書等はA4版で約4,000頁となっており、これは旧システム構築時の約3倍のボリュームであった。

arrowheadプロジェクトでは、こうして作られた要件定義書等をバイブルとし、前工程のバグ等を後続する次の工程で確実に刈り取り、下流工程に悪さを残さないと言う方針で以下のとおりの取り組みを行った。

・要件の妥当性確認
　＜実施工程＞　要件定義工程
・要件と設計書のトレース
　＜実施工程＞　基本設計工程～詳細設計工程
・ウォークスルーレビュー
　＜実施工程＞　詳細設計工程
・要件変更管理
　＜実施工程＞　基本設計工程～受入テスト工程
・リアルタイム品質管理
　＜実施工程＞　基本設計～受入テスト工程
・ベンダ側テストケースの要件充足確認（2回目のトレース）
　＜実施工程＞　システムテスト工程

これらの取り組みは、バグをV字モデルの反対側に位置するテスト工程で摘み取るのではなく、バグを作り込んだ工程の次に続く工程で積極的に

＊2　第6章で紹介した原理原則17ヶ条の適用事例を理解するために、SEC journal誌21号（2010年6月）のSEC成果活用事例紹介記事をそのまま掲載させていただいた。
　なお、当システムはIT Japan Award 2010の経済産業大臣賞（グランプリ）を、2010年7月に受賞している。

検出することを目的に行っている。東証ではこうしたプロセスの進め方を「フィードバック型Ｖ字モデル」(図1）と定義している。

以下では、各取り組みの具体的な内容について説明を行うこととする。

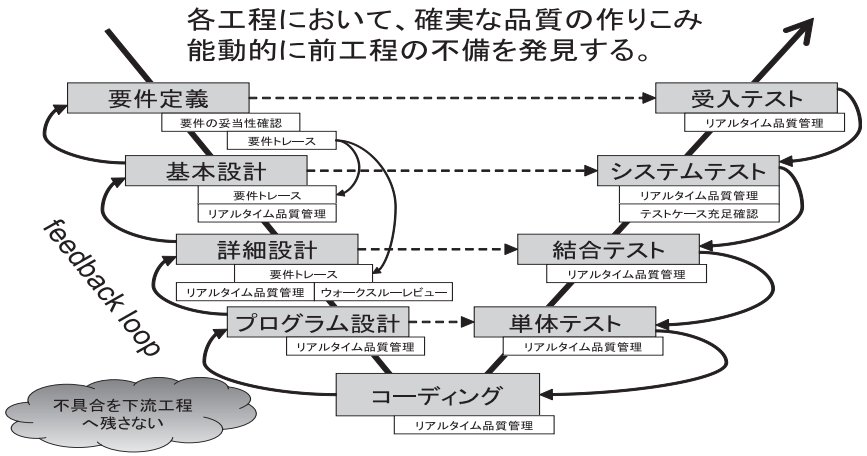


図1　フィードバック型Ｖ字モデル

3. 要件の妥当性確認

プロジェクトのバイブルを東証が作成する要件定義書及び外部設計書とする以上、これらのドキュメントに記載されている内容が妥当性のあるものになっていなければならない。

すなわち、東証が実現を要求する要件が漏れ無く記載されており（網羅性の確保)、かつ、実現可能な要件が記載されている（フィージビリティの確保）ことが必要となる。

要件の網羅性を確保するために、東証では、arrowheadの要件定義書及び外部設計書の記載項目と旧システム（arrowheadが稼働するまで現行機として動いていた売買システム）の設計書及び派生売買システムの設計書の記載項目との突き合わせチェックを行った。

また、フィージビリティ確保の観点から、非機能要件について外部機関（開発委託先とは異なるベンダ）にチェックを依頼し、実現可否の確認を行っている。とくに性能要件については世界最高水準の処理性能を目標としたことから、開発着手時に実在している技術及び実現可能と考えられる技術で実現が可能か見極める必要があり、発注者責任として、第三者の見解を確認することとした。

4. 要件トレース

プロジェクトのバイブルは東証が定める要件ではあるが、実際のシステムはベンダが作成する設計書に基づいて構築される。このため、要件が抜け漏れ無く設計書に反映されているかを確認する必要がある。東証では設計書のレビューを行う際、事前に要件の主要要素を約1万項目抜き出して、一覧表形式に取りまとめ、すべての項目について、設計書に対応する記述が行われていることを確認した。要件の実現は必ずしもアプリケーションで行われるとは限らないことから、要件要素の記述確認においては、ミドルウェアやOSの機能にまで確認範囲が及ぶこととなった。

また、東証の要件の中でも最も重要かつ複雑な処理であるマッチングの機能については、注文板の「状態」と注文・取消、売買停止、立会終了等の「トリガ」との組み合わせで要件をパターン分けした資料を要件定義書の補足資料として作成した。本資料により具体的なケーススタディを行うことで、ベンダ側の要件理解を促すと共に、東証側でも要件の網羅性確認と後工程で行うテストのケース設定に役立てている。前者の網羅性確認はフィードバック型Ｖ字モデルに基づく活動であり、一方のテストケース設定の基資料作りを上流工程で行っているという点では、W字モデルに類したプロセスの実践とも言える。

5. ウォークスルーレビュー

arrowheadのような大規模プロジェクトでは、ベンダ側の開発が幾つものチームに分かれて行われることとなる。この際に重要となるのが各チーム間の設計の整合性確保及び抜け漏れ防止である。

このため、arrowheadプロジェクトでは主要な業務処理について、東証とベンダ共同でウォークスルーレビューを行った。ウォークスルーレビューは3段階に分けて実施しており、第一段階では個別のチームが記載した処理内で、おのおの、正しく共通部品やフレームワークを使用しているかを確認した。第二段階ではチームをまたがる処理間で電文ヘッダーの定義やファイルの呼び出し方等の妥当性をチェックし電文疎通の確認を実施し、続く第三段階では上位処理と下位処理の間で受け渡される電文内の項目突合せによる業務疎通確認を行った。

6. 要件変更管理

要件定義書及び外部設計書はプロジェクトのバイブルであり、これが例えばコーディングやテストの工程で変更されるということは、すなわち当該工程からプロジェクトの最上流まで手戻りが起こることを意味する。

このため、arrowheadプロジェクトでは安易あるいは不要な要件変更が行われることを防止するためのセルフコントロールシステムとして、要件確定後の要件変更については、システム担当の常務取締役（CIO）のチェックを受け、承認が得られた案件についてのみ、ベンダに要件変更依頼を提示する仕組みとした（図2）。

要件変更管理に経営者自らが深く関与することにより、プロジェクト下流での不必要な要件変更が抑制された。また、前述したフィードバック型V字モデルの実践の効果と合わせて、東証が要件変更の必要性を発見した案件総数のうち83%を製造工程着手前に検出することが出来た。

要件が明確に記載され、確定した状態となっていることがすべての出発点であることから、arrowheadプロジェクトでは、要件の記載が読み手によって誤解が生じ得る表現となっていた場合には、仮に東証から見れば「常識的に判断すれば間違える筈がない」という類の誤謬であったとしても、要件の不備として取り扱うルールとした。

また、いわゆる「てにをは」レベルの修正であっても、正規の手続きを踏まなければ要件変更は行えないこととした。arrowheadプロジェクトでは、結合テスト工程での要件変更が10%弱あったが、これらの大半はこうした「てにをは」レベルであり、プログラム修正を伴うような要件修正はほとんど下流工程に持ち越されなかった。

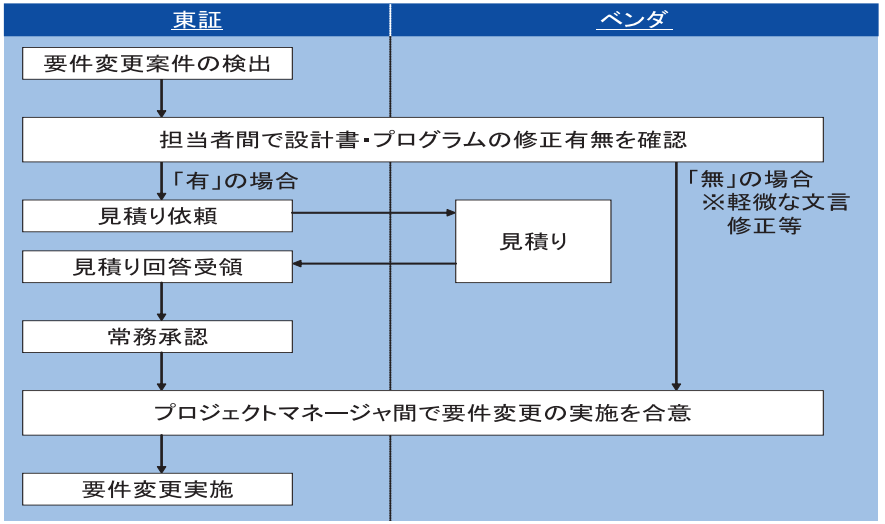


図2　要件変更の実施フロー（東証が要件変更の必要性を検出した場合のフロー）

7. リアルタイム品質管理

前章まで、上流工程での品質の作込みにかかる取り組みを紹介したが、製造以降の下流工程における品質向上策も確実に実施する必要がある。

下流工程での効果的な品質向上策適用のために、arrowheadプロジェクトでは、納品前にベンダ側で実施するテストでの障害発生状況を、東証及びベンダ間で日々共有することとした。

両社で工程別にあらかじめ合意、設定したアラームライン（許容下限値）に障害発生件数が近づいた際には、パレート分析による障害発生傾向の分析や発生した障害の内容（業務継続への影響度）の分析を行い、必要な品質強化策を実施する仕組みとした（図3）。

強化策は必ずしもベンダだけが行うものではなく、状況によっては、ベンダ側の品質強化策と並行して東証側でも要件定義の記述の詳細化・明確化を実施することとした。

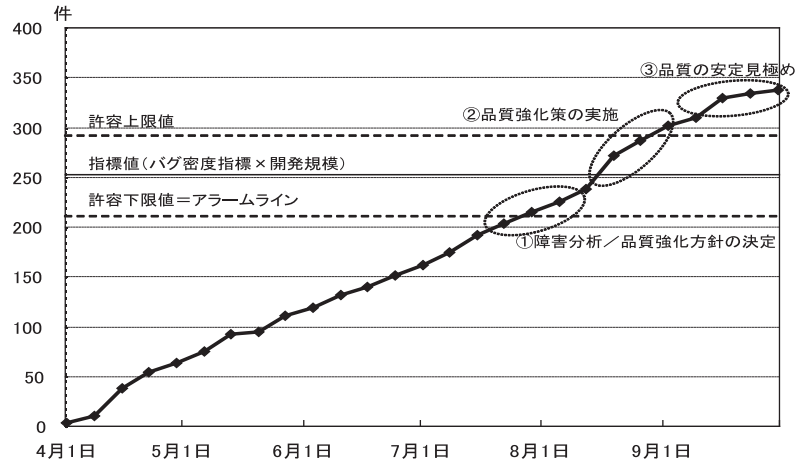


図3　リアルタイム品質管理

8. ベンダ側テストケースの要件充足確認

下流工程における品質向上策のもう一つの取り組みがベンダ側で実施するテストケースの要件充足確認である。これは上流工程で実施した設計書の要件トレースと対をなす活動で、設定されたテストケースが要件の主要要素を満たしているかを確認するものである。テストケースに不足があれば、追加のケースを設定するようベンダに要請した。

9. その他の工夫

9. 1　エンドユーザとのワーキング

要件定義書及び外部設計書を東証が取りまとめる際に、arrowheadのエンドユーザとなる証券会社とワーキングを開催し、仕様の調整及び合意形成を行うプロセスを組み込んだ。

本ワーキングはシステムの稼動まで継続的に開催し、エンドユーザを交えて実施するテストの内容・日程調整やエンドユーザ側でのシステム対応の進捗状況確認なども行った。

エンドユーザとの合意形成の仕組みを持つことは、手戻りの少ない円滑なプロジェクト運営の実現の重要な一要素である。

9. 2　プレジデント・レビューの開催

arrowheadプロジェクトでは東証及びベンダの両社長が出席し、プロジェクトの進捗状況や課題、リスクの認識を共有し、必要に応じた対応策の実施を決定するための会議を「プレジデント・レビュー」と称して、工程の区切りやプロジェクトレベルの課題発生時に開催した。

トップマネジメントがプロジェクトの状況を把握し、意思決定をタイムリーに行ったことは、arrowheadプロジェクト成功の重要な一要因となっている。

9. 3　開発拠点の統合

arrowheadプロジェクトでは東証とベンダの開発拠点を一個所に集約した。これにより東証とベンダ間のコミュニケーションが密になり、認識ギャップの低減等に寄与した。

9. 4　ツールの共同利用

テストの進捗状況や障害の情報について、一元管理するツールを東証側で用意し、ベンダと共同利用した。発注者側とベンダの状況認識や各種対策要否等にかかる見解相違は保有している情報の差異が原因となるケースも多々あるが、ツールの共同利用によるリアルタイムでの情報共有は有効な解決手段となり得る。

9. 5　リスク管理

arrowheadではリスク管理に予定／実績管理及びPDCAサイクルの考え方を導入した。

まず、検出されたリスクごとに発生確率を7段階（3～0、0.5刻み）、影響度を3段階（3～1、1刻み）に分類評価し、発生確率と影響度の掛け合わせ

によりリスクスコアを決める。次に各リスクのリスクスコアが、プロジェクトで計画されている、どの作業ないしイベントを実行することにより低減するのか計画を立てる。

プロジェクトの進捗と共に、計画通りにリスクスコアの低減度合いをチェックし、計画との差異が生じた場合には、必要に応じて打ち手を用意することとした（図4）。

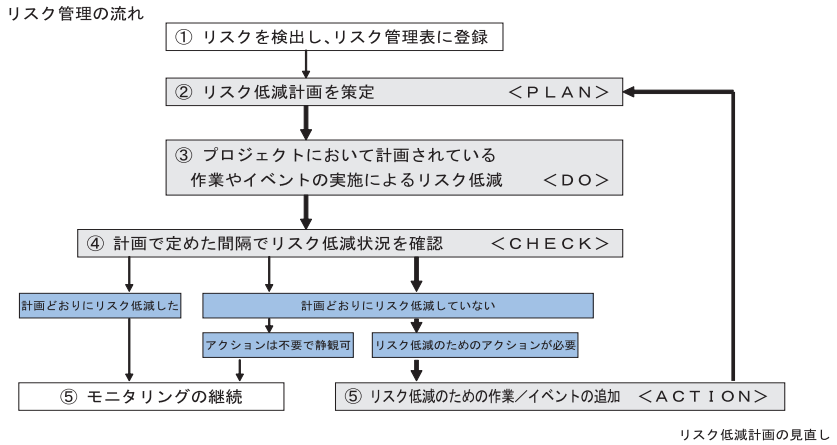


図4　リスク管理のフロー

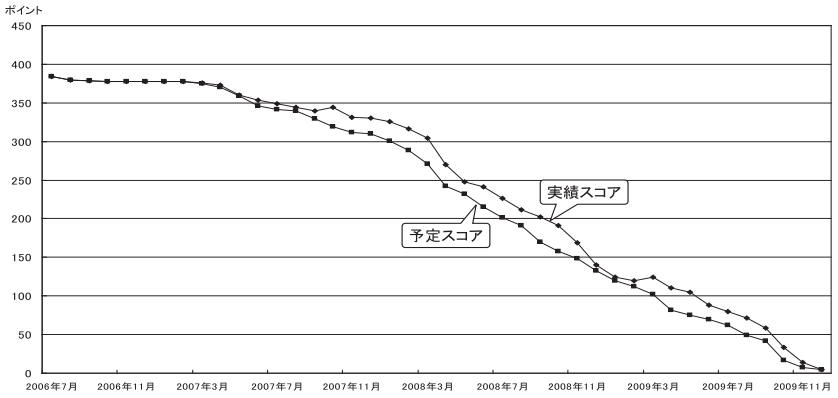


図5　トータルリスクスコアの遷移図

また、各リスクの予定リスクスコアの総和と実績リスクスコアの総和を時系列で比較することにより、プロジェクト全体でのリスクの状況を概念的に把握することに役立てており、当該データは前述の「プレジデント・レビュー」においても確認を行った（図5）。

9. 6　移行統制

システムの本番移行をスムースに行うためには、開発対象となったシステムにおけるデータ等の移行や接続先システムと合同の移行訓練のみならず、システムを利用する社内業務部門等や外部ユーザにおけるタスクの洗出しと管理を行い、本番稼働までにすべての準備を完了させる必要がある。

このため、arrowheadプロジェクトでは専務取締役（COO）を責任者とする「全体移行統制本部」を稼働4カ月前から立ち上げ、社内全部署にarrowheadが稼働することに伴うタスクの洗出しと各タスクの実施スケジュールの提示を依頼し、スケジュール通りに各タスクが進捗しているか管理を行った。遅延が見られるタスクについては原因、対応策の要否と対応を行う場合のスケジュール等を確かめ、全タスクについて完了確認を行った。

また、外部ユーザに対しては、東証とのシステム間接続仕様の変更点をリスト化し、いつまでにどの事項の対応を完了しておいていただきたいという目安を数段階で提示した。これをベースに定期的なアンケートを実施し、外部ユーザ側での対応状況を確認、対応が遅れているユーザについては個別に対応見込みの確認を行った。

プロジェクト全体での会議体は、図6の通りとなっている。

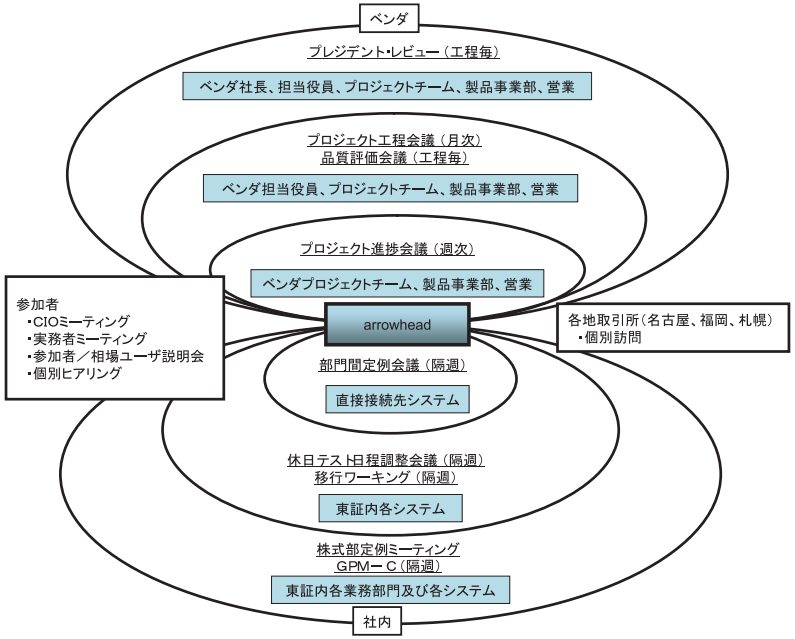


図6　arrowheadプロジェクトの会議体

10. おわりに

本稿では東証のarrowheadプロジェクトにおける発注者側での品質確保の取り組みについて紹介した。これらの取り組みは発注者側に相応の負担が生じるものであり、あらゆるプロジェクトに普遍的に適用できるものではない。

しかしながら、品質の問題は最終的には発注者の身に降りかかるのであるから、自身が引き受けるリスクを減少させるために、事前にベンダとの役割分担を明確にした上で、発注者責任を果たすよう取り組むべきであろう。

付録　原理原則17ヶ条と共通フレーム2007

原理原則17ヶ条はシステム開発における重要なポイントを列挙したものですが、字面を読んだだけではその精神を実際の場面で活かすことはできません。実際の場面で活かすには、原理原則17ヶ条に掲げられた行動規範を適切なタイミングで正しく実行する必要があります。

このタイミングを捉えるため、原理原則17ヶ条の行動規範を実行するのに最も適した共通フレーム2007のプロセス・アクティビティやタスクと、留意すべき点を付表1（巻末折込み）に整理しています。SEC BOOKS「共通フレーム2007 第2版」の第3部には関連する原理原則17ヶ条がガイダンスに挙げられていますが、本付録は、その逆引き的な関係にあります。

この付表1を用いることで、例えば、共通フレーム2007を各社の開発標準にテーラリングする際の参考にしたり、開発標準に注記したりすることで、システム開発の現場が原理原則17ヶ条の行動規範を理解し、実行するための手がかりにすることができます。

この付表1を活用し、システム開発の現場において原理原則17ヶ条にまとめられた発注者・受注者の心構えや行動規範が適切なタイミングで実践され、発注者・受注者が協力し合いシステム開発が成功裏に終わることを願ってやみません。

参考文献

① SEC BOOKS「経営者が参画する要求品質の確保 ～超上流から攻めるIT化の勘どころ～ 第2版」､独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター、オーム社、2005年4月

② SEC BOOKS「共通フレーム2007 第2版」、独立行政法人 情報処理推進機構 ソフトウェア･エンジニアリング･センター、オーム社、2009年10月

③ SEC BOOKS「ITプロジェクトの「見える化」 上流工程編」、独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター、日経BP社、2008年5月

④ SEC BOOKS「ITプロジェクトの「見える化」 中流工程編」、独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター、日経BP社、2008年10月

⑤ 日経コンピュータ2006年5月29日号、2007年5月28日号、2009年3月4日号、日経BP社

⑥「実務で役立つプロジェクトレビュー」、菊島靖弘著、翔泳社、2006年2月

編著者（敬称略）

独立行政法人 情報処理推進機構（IPA）
ソフトウェア・エンジニアリング・センター（SEC）
エンタプライズ系プロジェクト ビジネス・プロセス改善領域
プロセス共有化WG

主　査：**村上　憲稔**　富士通株式会社
副主査：**菊島　靖弘**　東京海上日動システムズ株式会社（株式会社アイネス）
委　員：**荒生　知之**　株式会社野村総合研究所
石川　貞裕　株式会社日立製作所
尾股　達也　社団法人情報サービス産業協会（JISA）
加藤　光明　伊藤忠テクノソリューションズ株式会社
清田　辰巳　株式会社東京証券取引所
角田　千晴　社団法人日本情報システム・ユーザー協会（JUAS）
寺田　尚弘　清水建設株式会社
内藤　裕史　日本アイ・ビー・エム株式会社
加藤　淳一　日本電気株式会社
黒田　英嗣　株式会社みずほコーポレート銀行
橋本　恵二　東京国際大学
端山　　毅　株式会社NTTデータ
福田二三雄　新日鉄ソリューションズ株式会社
松下　邦彦　新潟県総務管理部
室谷　　隆　TIS株式会社
山口　一郎　東京ガス株式会社（株式会社ティージー情報ネットワーク）
芳仲　　宏　東京地方裁判所
若杉　賢治　富士通株式会社
倉持　俊之　IPA／SEC（TIS株式会社）
新谷　勝利　IPA／SEC
長谷部　武　IPA／SEC（株式会社CSKシステムズ）
森下　哲成　IPA／SEC
山下　博之　IPA／SEC（株式会社NTTデータアイ）
山形　　薫　IPA／SEC（三菱電機株式会社）
WG委員以外の作成協力者：
足立　久美　株式会社デンソー
大和田尚孝　日経BP社
遠地　卓二　富士通株式会社
田倉　聡史　株式会社東京証券取引所
事務局支援：**小久保岩生**　株式会社三菱総合研究所

編者紹介

独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター
2004年10月に独立行政法人 情報処理推進機構 （IPA） 内に設立されたソフトウェア・エンジニアリング・センター（SEC）は、エンタプライズ系ソフトウェアと組込みソフトウェアの開発力強化に取り組むとともに、その成果を実践・検証するための実践ソフトウェア開発プロジェクトを産学官の枠組みを越えて展開している。
［所在地］　〒113-6591 東京都文京区本駒込2-28-8
文京グリーンコート センターオフィス
電話 03-5978-7543、 FAX 03-5978-7517
http://sec.ipa.go.jp/index.html

●お問い合せについて

本書又は掲載事例に関するお問い合せは、IPAホームページの「お問い合わせ」からお願いします。「一般の問い合せ」のフォームに必須事項をご記入の上、返信してください。

ご記入の際には、お問い合せ内容とともに、冊子名、頁番号等をご記入いただくようお願いします。

SEC BOOKS
実務に活かすIT化の原理原則17ヶ条
～プロジェクトを成功に導く超上流の勘どころ～

平成22年10月12日　第1版第1刷発行

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

発行所　独立行政法人 情報処理推進機構
所在地　〒113-6591
東京都文京区本駒込2-28-8
文京グリーンコート センターオフィス
電話　03-5978-7501（代表）
ＵＲＬ　http://sec.ipa.go.jp/index.html

付表1. 原理原則17ヶ条と共通フレーム2007

超上流から攻めるIT化の原理原則17ヶ条		共通フレーム2007	
原　理　原　則	行動規範	該当箇所	活用上の留意点
［1］ ユーザとベンダの想いは相反する			
	発注者・受注者は、お互いの責任、義務、想いを知る。	**1.1 取得プロセス** **1.2 供給プロセス** **1.3 契約の変更管理プロセス** **1.4 企画プロセス** **1.5 要件定義プロセス**	特に「1.1.2.2 対象プロセスの決定と修整」や「1.2.2.1 提案書の用意」にあるように共通フレーム2007のタスクを利用して、相互理解可能な提案依頼書・提案書を作成する。「3.1 管理プロセス」や「2.8.2 問題解決」を利用して齟齬を無くす手段を相互に確立することも有用である。
	発注者は受注者との役割分担を明確にし、プロジェクトに積極的に参画する。		「1.1.3.5 責任分担の決定」や「3.1.2.1 実施プロセス実行計画の策定」を用いてタスクとその責任分担を明確化する。積極的参加を促すために「2.6 共同レビュープロセス」を活用するとよい。
	発注者側業務部門も、“システム開発”を理解する。		特に「1.4 企画プロセス」及び「1.5 要件定義プロセス」では、発注者側業務部門が主体的に参画し、業務のイメージと“使えるシステム”のイメージを固める。プロジェクトのフェーズや内容に合わせて「2.6 共同レビュープロセス」を用いて発注者側業務部門も積極的にレビュー参加することで相互理解を深める。要件定義後に生じた変更については「1.3.2 契約の変更要求」「1.3.4 協議の実施と合意の形成」を利用して合意形成を図りながら進めていくことが重要である。
［2］ 取り決めは合意と承認によって成り立つ			
	発注者・受注者は、合意プロセスと承認ルールを明確にし、それに基づいて行動する。	**1.1 取得プロセス** **1.2 供給プロセス** **1.3.3 影響の調査分析** **1.3.4 協議の実施と合意の形成** **1.4.3 システム化計画の立案** **1.5.3 利害関係者要件の確認**	合意を得る場は色々あるが、契約に係わるもの、要件に係わるものは特に合意形成と承認が重要である。問題が発生した場合に速やかに解決するためのルールを「2.8.2 問題解決」又は「2.6 共同レビュープロセス」を用いて定めておく。
	受注者は、良否判断を仰ぎやすい提案を心がける。		
［3］ プロジェクトの成否を左右する要件確定の先送りは厳禁である			
	発注者は、未確定要件の先送りは厳禁であり、現工程を延ばしてでも確定させる。	**1.5.2 利害関係者要件の定義** **1.5.3 利害関係者要件の確認**	実質的な要件は、「1.5.2 利害関係者要件の定義」で定義し、「1.5.3 利害関係者要件の確認」で確定される。これらのアクティビティで、業務的未定事項や技術的リスクを無くし、要員・納期・コストの正確な見積もりが可能となるようにする。どうしても未定事項や技術的リスク（例えば、新技術利用が前提など）が残る場合は、「3.1 管理プロセス」を用いて課題又はリスクを明示的に管理対象とし、早期解決あるいはリスク顕在化時の代替手段の検討を行うことが重要である。
	受注者は、主要要件の実現の目処がたたないままプロジェクトを進めない。		
	発注者・受注者は、未確定要件によるリスクを早期に低減する施策を打つ。		
［4］ ステークホルダ間の合意を得ないまま、次工程に入らない			
	発注者は、ステークホルダの合意確認を自らの仕事と心得る。	**1.3.4 協議の実施と合意の形成** **1.4.3 システム化計画の立案** **1.5.3 利害関係者要件の確認**	「1.4 企画プロセス」「1.5 要件定義プロセス」では、最終的に関係者の合意を得ることが成果の一つである。「1.3 契約の変更管理プロセス」でも同様である。「関係者の合意」は、企画プロセスでは経営を含めた組織全体の総意とするために、要件定義プロセスでは以降のベースラインを決定するために、契約の変更管理プロセスではベースラインの変更を決定するために、重要なステップである。関係者の合意を得るため「2.6 共同レビュープロセス」を利用するが、レビューの主催は発注者側であることが合意形成に有効であり、明示的に役割を定めるべきである。いずれの場合も、合意を得るステークホルダの選定の誤りはトラブルの元となるので慎重に行う必要がある。
	受注者は、合意を得ないまま開発に入ると、要件定義自体がひっくり返るおそれがあると心得る。		
	受注者は、合意確認の作業支援はできるが請負（責任）はできないことを明示する。		
	双方は、ステークホルダが誰か、漏れはないかを確認する。		
［5］ 多段階の見積りは双方のリスクを低減する			
	発注者は、事業リスクを低減するためにも、多段階見積りを活用する。	**1.1 取得プロセス** **1.2 供給プロセス**	共通フレーム2007は、契約形態から独立している。多段階見積りに関しては、経済産業省の「情報システム・モデル取引・契約書」を参照するとよい。
	受注者は、見積りリスク回避のため多段階契約を活用する。		
	受注者は、 要件定義段階では非機能要件に保証できないものがあることを説明する。		「1.5.2.5 非機能要件の定義」で定められる非機能要件（特にシステムパフォーマンスなど）には、「1.6.3 システム方式設計」又は「1.6.5 ソフトウェア方式設計」に大きく依存することがあるため、見積り条件に明示し発注者に説明するなど留意が必要である。
［6］ システム化実現の費用はソフトウェア開発だけではない			
	発注者は、依頼する範囲、内容を漏れなく洗い出し、提示する。	**1.4.3 システム化計画の立案** **1.5.2 利害関係者要件の定義**	企画立案時は「1.4.3 システム化計画の立案」、要件定義時は「1.5.2 利害関係者要件の定義」において、ソフトウェア開発以外の費用を織り込む必要がある。発生する作業や費用の洗い出し及び特定には、 共通フレーム2007又はその修整を用いることで漏れと重複をなくすことができ、関係者間の誤解のリスクを低減できる。
	受注者は、見積りに含まれる内容と根拠を明確化する。		
	発注者は、運用・保守も見据えた計画・体制を作る。		
［7］ ライフサイクルコストを重視する			
	発注者は、システムのライフサイクルにわたって投資対効果（ROI）を算定する。	**1.4.2 システム化構想の立案** **1.4.3 システム化計画の立案** **1.5.3 利害関係者要件の確認**	企画段階では、ライフサイクルコストの見積りと合わせて、コスト以外のメリットについても目論見を極力数値目標化しておく。このことは、運用段階でコストやコスト以外のメリットの実績評価を行うために必要である。実績評価は、稼動後、数年は継続して実施することが望ましい。
	発注者は、業務パッケージを採用する場合は、カスタマイズを前提としない。		カスタマイズは、開発コストのみならず運用後の保守コストの増大にもつながるため、慎重に検討すべきである。
	受注者は、対象システムの特性をよく見きわめて開発技法・環境・ツールを適用する。		共通フレーム2007は、開発技法・環境・ツールから独立している。ただし、「3.2.1.1 環境の定義」として必要な環境を定義するタスクがあるので、その際に対象システムの特性を十分考慮することが必要である。
	受注者は、運用性・保守性を高める提案をする。		「1.7 運用プロセス」「1.8 保守プロセス」からコスト検討に関係するタスクを洗い出し、必要に応じて「3.4 人的資源プロセス」「3.5 資産管理プロセス」「3.6 再利用施策管理プロセス」「3.7 ドメイン技術プロセス」を考慮し全体コストの検討を行うとよい。なお、「3. 組織に関するライフサイクルプロセス」は、発注者側のスコープにも関係するため、発注者も交えて最適化を検討すべきである。
［8］ システム化の方針・狙いの周知徹底が成功の鍵となる			
	発注者は、情報システム構築の目的を明確にする。	**1.4 企画プロセス** **1.5 要件定義プロセス** **1.7 運用プロセス**	「1.4 企画プロセス」において、システム化の目的を経営・事業との関係においてシステム化構想・システム化計画に織り込むことが重要である。
	発注者は、情報システム構築の方針・狙いをステークホルダに周知徹底する。		システム化の目的の周知徹底は、企画段階においては「1.4.2.8 システム化構想の文書化と承認」「1.4.3.17 システム化計画の作成と承認」を用いて経営・事業主体に対し周知し、要件定義段階では「1.5.2 利害関係者要件の定義」を通じて業務・現場の代表・受注者に対し周知する。また、開発以降は「1.7.1.8 業務運用に係る作業手順の確立」や「1.7.5 利用者教育」を通じて個別現場へ周知する。
	受注者は、方針・狙いを理解して、情報システムを構築する。		

付表1. 原理原則17ヶ条と共通フレーム2007

超上流から攻めるIT化の原理原則17ヶ条		共通フレーム2007	
原　理　原　則	行動規範	該当箇所	活用上の留意点
[9] 要件定義は発注者の責任である			
	発注者は、「我々が要件を決め、責任を持つ」という意識を社内に浸透させる。	**1.1 取得プロセス** **1.2 供給プロセス** **1.5 要件定義プロセス**	要件定義は、システム開発のためではなく、新業務の定義のために実施するという心構えを発注者・受注者双方が共有することが重要である。そのために「1.1.3.5 責任分担の決定」や「3.1.2.1 実施プロセス実行計画の策定」を用いて要件定義の責任分担・実行計画を定め、関係者で共有する。発注者は、受注者の役割とした作業について、「1.1.4 供給者の監視」により「2.6 共同レビュープロセス」を用いて必ず確認と承認を行うと同時に、「1.1.4.2 供給者への協力」に従い、必要な情報の提供・未決定事項の解決に協力する。要件定義は、発注者の新業務定義が目的であるため、受注者は要件定義における「1.2 供給プロセス」の実施において目的を達成するよう支援する。
	発注者は、業務部門とIT部門が、二人三脚で要件定義を進める。		
	発注者は、要件定義段階で受注者をうまく活用する。		
	受注者は、発注者の側に立った支援を提供する。		
[10] 要件定義書はバイブルであり、事あらばここへ立ち返るもの			
	発注者は、安易に変更できない「重み」を認識して要件定義書を提示する。	**1.1 取得プロセス** **1.2 供給プロセス** **1.3 契約の変更管理プロセス** **1.5 要件定義プロセス**	「1.5 要件定義プロセス」で策定された要件定義は、「1.1 取得プロセス」「1.2 供給プロセス」により発注者・受注者のベースラインとなる。ベースラインの変更となる要件の変更は、「1.3 契約の変更管理プロセス」で管理する。なお、金銭的関係を含まなくても、要件定義書の提示と受託は、契約締結であることに留意すべきである。
	受注者は、安易に回避できない「責任」を認識して要件定義書を受託する。		
	受注者・発注者とも、以降の変更はすべて要件定義書をベースとして議論する。		
[11] 優れた要件定義書とはシステム開発を精緻にあらわしたもの			
	発注者は、機能要件、非機能要件などを漏れなく洗い出す。	**1.5.2 利害関係者要件の定義**	「1.5.2 利害関係者要件の定義」に従って、できるだけ精緻に要件定義を実施する。要件定義が精緻であれば、無駄な開発やテスト、手戻りを排除でき、トータルコストの抑制になる。精緻さの基準には、例えば「未決定事項がなくなるまで」「積算見積りができるようになるまで」などがある。 新業務が成立するための直接的機能要件だけでなく、機能要件を実現するための非機能要件についても十分に洗い出し、定義しておくことが必要である。非機能要件は業務ピーク・締め日等の情報、システム運用の実務・経験、開発者の経験・ノウハウを総合的に検討する必要があるため、関係者の協力が必須である。また、経済的実現可能性の検討が可能となるよう、要件が見通せる段階から総費用見積もりを開始し、目標との整合を図りながら、要件定義を進めることが合意形成の近道である。
	受注者は、特に非機能要件の定義で専門家としての支援をする。		
	双方の協力で、システム開発の総費用を固める。		
[12] 表現されない要件はシステムとして実現されない			
	発注者は、文書・モックアップなどの手段を講じて、要件を表現しつくす努力をする。	**1.5.2 利害関係者要件の定義** **1.7 運用プロセス**	「1.5.2 利害関係者要件の定義」において、システム機能は、目に見えにくく動作を伴うため、様々な手段を用いて見える化し、確認していくことが重要である。曖昧さを排除するため画面と画面遷移のモデル作成や機能の数値化は有効な手段である。なお、「1.7.1 プロセス開始の準備」において発生する要件もある（例えばシステム運用のツール要件やシステム障害時の業務運用と復旧後のリカバリー要件など）ことに注意する。
	受注者は、行間を読むのではなく、きっちり確認をとって進める。		
[13] 数値化されない要件は人によって基準が異なる			
	発注者・受注者は、協力して、定量化できる要件は極力数値化する。		
[14] 「今と同じ」という要件定義はありえない			
	発注者は、現行システムと同じ機能の定義であっても、要件定義を実施する。	**1.5.2 利害関係者要件の定義**	“今と同じ”であれば、システム開発の必要性を疑う必要がある。同じ部分がある場合は、現行システムとの同一性ではなく、現行業務が新システムで同じとなるかを確認する。
	発注者は、既存機能だけを見て要件とするのではなく、使われ方まで十分調査し、要件とする。		
	受注者は、「今と同じ」要件を具体要件まで問い直す。		
[15] 要件定義は「使える」業務システムを定義すること			
	受注者は、常にビジネス要求の視点から、システム要件の妥当性を検証する。	**1.4.3 システム化計画の立案** **1.5.2 利害関係者要件の定義**	「1.4.3 システム化計画の立案」により、新業務における新システムの果たすべき役割が明確になり、その役割を実現するために「1.5.2 利害関係者要件の定義」により「使える」システムを具体的に定義する。要件定義では、システム開発が主眼とならないよう、業務要件から常に出発することが基本である。複雑な業務は複雑なシステムとなり開発コスト増・運用コスト増に直結するため、「1.5.2.2 業務要件の定義」では業務がシンプルとなるよう、心がけることが重要である。
	発注者は、シンプルな業務設計を心がける。		
	発注者は、運用要件を要件定義の中で定義する。		
	受注者は、オーバースペックを是正し、コストショートを進言する。		
[16] 機能要求は膨張する。コスト、納期が抑制する			
	発注者は、必要最低限のシステム構築からスタートする。	**1.1 取得プロセス** **1.2 供給プロセス** **1.3 契約の変更管理プロセス** **1.5.2 利害関係者要件の定義**	「1.5.2 利害関係者要件の定義」において、発注者側の要求が膨張しないよう、受注者側はコスト・納期目標を発注者に適宜示し、実務的に合意を得やすいように要件定義を進めることが必要である。また、要件定義完了後に変更が生じた場合は「1.3 契約の変更管理プロセス」によりコスト・納期への影響を見極め合意を図る。 いずれにせよ、双方が納得するためには要件定義を文書化し、ベースラインとして契約に明示することが重要である。
	発注者は、要求を抽出する段階と、要件として絞り込む段階を分ける。		
	発注者は、要件の優先順位付けをする。		
	受注者は、納期限界を超える開発量と判断したら、段階的稼動を提案する。		
[17] 要件定義は説明責任を伴う			
	発注者は、受注者に要件を正しく説明する。		
	受注者は、要件を理解して、理解した内容を発注者に確認する。		

ISBN978-4-9905363-2-9
C3055 ¥477E

9784990536329

19230550047 71

定価500円(本体477円+税)

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

SEC BOOKS

実務に活かすIT化の原理原則17ヶ条

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

SEC－TN10－001