キリンビジネスシステム様での運用生産性向上への取り組み

経営統括本部　人事部　人事グループ
（キリンビジネスシステム株式会社出向中）
桐山 俊也

1．はじめに

キリンビジネスシステム株式会社様（以下KBS）は、今年、運用生産性を向上させる新たな取り組みを開始した。従来からの、主に運用フェーズに焦点を当てた施策に加えて、今回新たに、運用の生産性向上のため、開発フェーズに焦点を当てるというアプローチを取っている。つまり、運用効率を低下させる要因の多くが、システムの仕様や設計に由来している点に注目し、設計・開発段階でそうした運用生産性の阻害要因が埋め込まれるリスクを顕在化させようという取り組みである。

2．背景

これまでにもKBSは、開発の生産性をはじめとする、さまざまな社内の生産性向上策に取り組んできている。今回取り上げている「運用生産性」をテーマとする主な施策に限ってみても、非常に多種多様である。具体的に挙げると、表1のような施策が実行に移されている。

いずれの施策も一定の成果を上げており、継続的な施策はどれも現在も実施中である。ただ残念ながら、各部門内の個別努力、単発・単年実施、予備検討や準備レベルにとどまる施策も少なくない。運用を担う各部が個別努力や工夫で効率を上げてゆく活動はもちろん不可欠である。だが、担当業務、システム、ユーザー業務などによって、運用の実態はきわめて多様である。部門・業務・システムの違いを凌駕するような業務統一や標準化は、現実的でないばかりか、無理に適用しようとすれば、場合によっては効率低下をもたらしかねない。そういった事情が、抜本的な運用業務改善を難しくしている1つの理由である。

表1　KBSにおける、これまでの運用生産性に関する取り組み

目　的	施　策
1. 運用業務の改善（無駄の排除）	・業務の棚卸しとマンダラを使った現状分析　など
2. 運用業務の指標づくりと実績把握	・運用生産性指標の策定 （FP：Function Pointベースの独自指標） ・インシデント* 情報のデータベース化 ・実績把握・計測のための作業体系（作業コード）整備　など ＊障害・改修・ユーザー（システム利用者）からの依頼・その他イベント対応などの諸事象
3. サービスレベルの明確化	・運用のサービスレベル規定およびユーザーとの合意 （SLA契約）
4. 標準化・ガイドライン	・ITILベースの運用業務ガイドライン策定 ・各種運用業務のマニュアル化　など
5. ナレッジ化	・ノウハウの形式知化、各種の情報共有　など
6. 担当者のスキルアップ	・情報共有、教育研修、その他の人材育成策　など

しかし、KBSは“だから現状で仕方なし”とはせず、さらに運用生産性を向上させようと、この数年は、戦略的な課題として取り上げ、取り組みを模索していた。その時、新たな取り組みへのトリガーになったのは、1つの視点の切り替えである。

すなわち、

「運用工数を減らそう⇒運用業務の効率を上げよう」

という、従来の施策の主流だった視点から、

「元を断とう⇒運用工数を増大させる要因を排除しよう」

という視点への切り替えである。

運用現場で起きている問題・現象の原因を探っていくと、必ずしも運用フェーズに閉じない課題が多い。ところが、従来の多くの施策が、最下流の運用フェーズを対象に、まさに運用業務そのものの効率を何とかしようとする試みだった。運用フェーズでいくら工夫・改善を積み重ねても、システム開発時に運用の生産性を阻害させる要因を次々と作りこんでいれば、蛇口を閉めずに水を汲み出すようなものである。――ここに、未着手の改善ポイントがあると考えられた。

経営環境が変化するにつれて、「短期開発・スピード実装」の傾向が強まってきている。短期間で開発する必要に迫られると、開発担当者は、例外やレアケースへの対処を省略するようになる。また、低コストで開発しなければならないというプレッシャーが強いと、システム利用者が直接意識しない管理機能や安全対策など、非業務機能を簡略化しようとするようになる。その結果、複雑な処理や、実装に手間のかかる機能は「運用で逃げる」対応が増えていく。だがこれは、明らかに問題の先送り――開発から運用への負荷の付け替えである。

保守性、汎用性、障害対策、安全管理のための機能などが省かれると、技術者が手厚くサポートしなければ動かないような、「裸のシステム」ができ上がる。実装されなかった機能は、運用フェーズに移ってから人手で対処することになり、例外やレアケースは手作業で処理しなければならなくなる。運用工数は増大し、システムライフサイクル全体で見れば、システムコストはどんどん高くなってしまう。

例えば、あるシステムについて、システムの利用者から操作や機能についての問合せが非常に多い場合は、入力画面のユーザビリティが低かったり、マニュアルや導入トレーニングが不十分であったりする可能性が疑われる。あるいは、頻繁に機能修正や仕様変更が発生するシステムの場合、現状分析や要件定義における検討が不十分であった可能性が疑われる。このような、開発フェーズで発生する要因が、運用フェーズの生産性にどのように影響するかを関連付け、対処すべき「開発～テスト」の工程を整理したものが、図1である。

KBSでは、KSM（Kirin Standard Methodology）という開発方法論が標準適用されている。この方法論は、設計フェーズとテストフェーズをV字型に関係付けた、いわゆる“V字モデル”を全体構成の基底に用いている。これになぞらえ、設計フェーズ⇔テストフェーズ⇔運用フェーズ

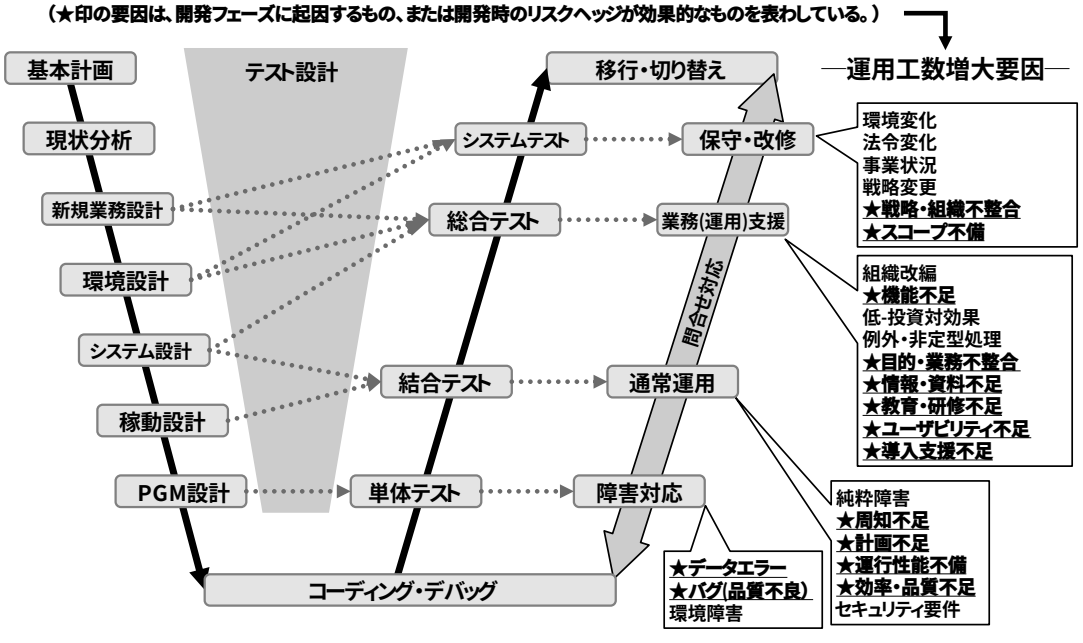

図1　設計・開発から運用フェーズへの影響関連（Wモデル）

の相関をとり、運用の負荷を高める要因が、どのようなフェーズで発生するかを整理したのである。KBSではこれを“Wモデル”と呼んでいる。

ひと昔前――といってもほんの数年前までだが、システム利用者の利便性や操作性に関する配慮の行き届いた「重厚長大型」システムの開発姿勢に対して、社内から疑問の声があがった時期がある。利用者の要望を何でもそのままシステムに実装していたのでは、過剰装備である、と。それは正しかった。しかし、今度は逆に、ユーザビリティを簡略化するだけでなく、本来は必要な、保守・運用のための仕様まで削ってしまうプロジェクトが現われるようになった。

「短期開発・スピード実装」は時代の必然的な流れであり、逆行するわけにはいかない。しかしだからといって、運用効率を落とす要因が、設計・開発時に埋め込まれている事態をみすみす看過するわけにもいかない。どちらに振れすぎることも、避けなければならないのである。

KSM方法論では、「業務要件⇔非業務要件」、「開発コスト⇔運用コスト」などのバランスを適切に計り、どちらにも偏らないよう留意すべきことが謳われている。しかし、PM（プロジェクトマネジャー）の認識を新たにし、バランス重視の認識を定着させるには、方法論のようなリファレンスよりも、もっとインパクトを与える仕組みが必要とされていた。そこで採用したのが、チェックリストによる定期的なチェックを通して、リスクを顕在化させるという手段である。

3．チェックシステムの導入

実際の仕組みは、リスク・チェックツール「SQET」(Software Quality Examination Tool：販売＝株式会社ソフトウェア・サイエンス）を用いてシステム的に構築されている。SQETは、質問に答える形で開発工程の実施状況や作成ドキュメントの記述レベルをチェックすると、後続工程への影響（リスクやその程度）を数値的に可視化できるツールである（図2参照）。

これは、設計開発時の作業や成果物が、運用に与えるインパクトを顕在化させたい、というKBSのコンセプトにマッチしていた。そこで、KSM方法論が規定するシステム開発の全体工程をSQET上に再現し、その各工程ごとに、運用生産性の阻害要因をシステムに埋め込まないためのチェック項目を配置することにした。

目標は、評価点やグラフを通して、開発プロジェクトがどのようなリスクをどの程度抱えているかを共通指標で評価し、顕在化させることである。例えば、開発コストに関するリスクだけが高く、ユーザビリティや運用効率などにリスクがまったく表われていなければ、まさに過剰装備に偏っている可能性が高い。逆に、コストや納期に関するリスクが全く見られないのに、ユーザビリティや運用効率のリスクが大きく表われていれば、必要な支援機能・管理機能まで削ってしまっている可能性がある。

開発の各工程でこうしたリスクチェックを定期的に行なえば、「開発⇔運用」や「業務⇔非業務」のバランスが、PMの判断に依存せず、全社的な共通指標で計れる。さらに、それらを部門・全社で蓄積すれば、リスク度合い（＝SQETの評価）と運用生産性（＝運用コスト）の関係を数値的に結び付ける基礎数値が得られ、適正バランスを実測値に基づいて評価・判断できるようになる。そうすれば、将来的には、システム利用者を巻き込んだ共通認識の形成も可能となっていく。

「運用」はその実、システム開発よりもはるかに多様で広範な業務の総称である。これをまるごととらえて、一気

図2　セルフチェックシステムのイメージ

表2　チェックリストに取り入れたリスクの視点

分　類	チェック項目
業務目的達成	導入効果、合目的性、機能要件
システム品質（Q）	信頼性、正確性、効率性（時間・資源）、非機能要件
開発コスト（C）	開発コスト、開発効率、開発ボリューム
スケジュール（D）	日程・納期、調達・準備、コミュニケーション（仕様決定）
サービスレベル	ユーザビリティ、ユーザサービス
運用・保守効率	保守性、運用コスト、標準化（移植性）
安全性・遵法性	コンプライアンス、内部統制、セキュリティ、BCP

に生産性を向上させるような“魔法”は、ほとんどありえない。今回の取り組みも、決して“魔法”ではない。むしろ、効果を見るまでに長い時間を要する漢方薬的体質改善のような試みである。しかし、派手ではないが、こうした地道な対処をPDSサイクル（Plan-Do-Seeサイクル）で回していく方が、現実には着実な成果に結びつくと考えられた。

4．チェックリストの作成

ただし、大きな問題があった。運用生産性のためのチェックリストをまともに作成してしまうと、開発担当のPMは、「まさにそのためだけ」にSQETを起動し、チェック作業を行なわなければならない。それでなくても、PMは普段から多くの管理シートを作成しなくてはならない。そこへさらに運用生産性のためのチェック作業を追加して、負荷を上乗せするわけにはいかなかった。

そこで、運用生産性の向上こそ当初の目的ではあったが、いわゆるQCD（Quality：システム品質、Cost：開発コスト、Delivery：スケジュール）をはじめ、表2にあるような開発プロジェクト全般のリスク事項を網羅して、チェックリストを汎用化する方針とした。

このチェックリストによって、社内で運用していたいくつかの管理シートを代替・集約し、何種類ものチェックや管理シート作成をしなくてもすむようにした。また、主要な標準やガイドの重点ポイントをチェックリストに盛り込んで、PMが様々な方法論やガイドラインの分厚い資料と首っ引きにならなくても、押さえるべき主要ポイントをSQETから得られるようにした。さらに、PMならば誰でも当たり前に実施しているはずの事項や、実施しなくても影響の少ないと思われる事項のチェックを省略して、必要十分な項目数に絞り込み、できるだけPMに負荷を掛けないような工夫を行なった。

なお、チェックリストには、表3に示すような社内標準などを取り入れた。

ところで、前掲のようなリスクは何も、新規開発プロジェクトだけで生じるものではない。運用フェーズにおいても、保守と称して日常的に改修や拡張が行なわれている。また元来、運用効率を上げるために、運用フェーズで、運用担当者が、日々の業務において継続的に対処していくべき事項は少なくない。したがって、運用担当のPMも、チェックリストを利用すべきであり、チェックリストには、開発フェーズだけでなく、運用フェーズにおいても日常的に業務チェックが行なえるような配慮がなされた。

表3　チェックリストに取り入れた社内標準その他

- 開発方法論／KSM（Kirin Standard Methodology）
- パッケージシステム導入標準
- プロジェクトマネジメント標準
- セキュリティ基準
- 開発成果物受入基準
- 運用プロセスガイド
- 標準RFP（Request for Proposal）ガイド

ほか（過去1年分のプロジェクト終了報告、成果・事例報告、など）

これまでの各種施策で、運用の生産性向上に結びつく各種のノウハウや課題は、ある程度蓄積されていた。そこで、過去数年の施策を振り返り、得られた成果、知見、工夫やノウハウを棚卸しして、エッセンスを抽出し、運用そのものの改善に生かせるよう、チェック項目に盛り込んだ。

これによって、開発担当のPMやリーダー、保守運用担当者、ユーザー委託業務の担当者、問合せ窓口担当者など、スタッフ部門以外のほぼすべての業務範囲をカバーし、運用の効率化を全社レベルのPDSサイクルで回せるようになった。

チェックリストは、工程や項目ごとに多段階で重み付けされており、下流工程への影響が大きい事項は、より重い位置づけになっている。「運用生産性への影響ばかりでなく、品質、セキュリティ、コスト――どれも重要」などという全部主義は極力廃し（ありがちなことだが）、できる

だけ重み付けの分布が正規形に近くなるよう調整されている。加えて、それぞれのリスクに関連するチェック項目をできるだけ平均的に取り上げて、特定リスクに関するチェック項目の数だけが突出しないように調整されている。

こうして、もともとは運用生産性向上をねらい、設計開発時に生じる阻害要因を排除する目的で立案されたチェックリストだが、最終的には、KBSの開発業務・運用業務の全般に関するリスクチェックができるものとなった。チェック項目は370余、システム開発・保守関連が約230、運用関連が約40、その他が約100という内訳である。一方、KSMの工程定義をリファインおよび拡張してSQET上に展開した開発〜運用業務の全体は、大きく6フェーズに分類され、大工程で28、もっとも詳細なレベルで109工程になる。この数値からわかるように、チェック項目数は、平均すれば1工程あたり3〜4項目であり、大工程の場合でも13〜15項目程度である。

もちろん、必ずしもこれらすべてのチェック事項が常に必要なわけではない。システムや対象のユーザー業務、PMの担当するフェーズによって、チェックすべき項目は異なる。すべてのPMが一律に同じチェックを行なう必要はない。それゆえ、開発規模の別や、「手組み開発⇔パッケージベース」の別、運用保守の形態別（保守管理、業務運用、運行監視、ユーザーサポート）などのパターンをPMが選べば、自動的に関連するチェック項目が取捨選択されるようになっている。さらに、必要に応じて、特に気になるフェーズ・工程だけをチョイスしてチェックする——という限定的な使い方もできるようにしてある。

このように、必要なリスクチェックを網羅的に盛り込みながらも、一方で、PMのチェック負担をできるだけ軽くする工夫が行なわれた。たとえば、開発プロジェクトのPMは、工程を次に進めるタイミングでチェックを行なえば、1回当たり15分程度で、終えようとしている工程のリスクを確認し、またはこれから着手しようとしている工程のリスクを事前に俯瞰できるのである。

5．セルフチェックによる改善アプローチ

こうしてでき上がったのは、IT（情報技術）業務全般において、リスクに関する「気づき」の機会をもたらす、セルフチェックの仕組みである。開発プロジェクトにおいては、PMまたはリーダーが、プロジェクトの進行に合わせて、各工程をチェックしてゆく。既に運用フェーズに移っているシステムの場合、運用担当者は、担当システムまたは部門の運用保守業務に関して、定期的にセルフチェックを行なう。各部の品質担当者は、既に終了しているプロジェクトの「振り返り」「事後評価」も可能である。ユーザーは、開発のスピードアップや開発コストの抑制を優先しすぎた場合のリスクをビジュアルに認識して、開発・運用担当者と共通の指標にもとづいて判断できる。

「気づき」のためのセルフチェックには、合格・不合格があるわけではないので、チェック結果にかかわらず、ツールが示すリスクを念頭において開発や運用業務を行なうならば、先に進んでよいとしている。ただしそれは、PMが勝手に判断してよいという意味ではない。KBSでは、品質管理機構（プロセスを含む）が各部門・全社レベルで、それぞれ常に動いている。この品質管理機構は、プロジェクトの状況把握と判断を行なう。セルフチェックの結果に表れたリスクの内容を把握し、許容するかどうかは、そうした然るべき意思決定レベルで行なわれる。

チェックの評価点や、リスクに対する感応度は、種々の実プロジェクトでシミュレーションを行なって調整した。平均的な成功プロジェクトで85前後（リスクが15あるという意味）、一般的なプロジェクトだと70前後になるようチューニングしてある。リスクをゼロにすることは基本的に難しく、また現実には、ほとんどありえない。コスト／納期の制約、ユーザーの（リスクテイクを承知の上での）要望、技術・環境・人材等々…リスクを承知の上で先に進まざるを得ない状況や理由は、いくらでもある。また、チェックされている対処や配慮をすべて実施すると、逆に対応工数が激増してしまいかねない（全方位的な対処はせず、バランスを計っているかどうか自体も、チェック項目になっている）。極端に運用を度外視していると、運用へのインパクトが顕著に表れるが、逆も然り、過度に運用を重視しすぎると、開発コストの著しい高騰を招く。したがって、あくまでも、リスクのバランスを「意識」することがポイントなのである。

6．おわりに

チェックリストを使って、リスクを顕在化させること、下流工程への影響を可視化すること、時系列を追って評価を蓄積・共有すること、システムや業務に応じてチェック項目をパターン選択できるようにすること——こうした仕組みの1つひとつは、どれも突飛な発想ではない。“Wモデル”の着想からチェックリストを実装するまでは、SQETというツールを得て実現が加速できた。

だが実は、（どんな仕組みであれ、）いかに普及・定着させて、PDSサイクルを回していくか——という課題の方が、大きく難しい。今回は、あえて運用生産性という当初目的をはみ出してチェックリストを汎用化し、適用範囲を広げて利用機会・応用機会を増やした。PMの負荷を上げないよう、リストに様々な工夫を施した。その結果、チェックリストは何倍にもなったが、社内の開発・運用リスクを測

る共通基盤ができ上がった。

本施策はまだ開始段階であり、本格的な展開や成果は今後を待たなければならない。もちろん、数多くの課題が残されている。今後、開発・運用現場でチェックリストによるPDSサイクルを回すと同時に、得られた評価データに基づくチェックリスト自体のチューニング（実測値と指標との比較検証）、今回は盛り込めなかった各種の指標や標準の取り込みなども、PDSサイクルを回しながら継続していく。それによって、よりいっそう、システム開発・運用の生産性向上を図り、かつ有効なリスク指標を確立することを目指している。