大規模金融システム開発における品質管理事例

金融ビジネスユニット
金融第四センター長
川真田 一幾

1．はじめに

本稿では、銀行向け「オフバランスシステム」再開発時における品質管理事例について報告する。

対象は、東京、ロンドン、ニューヨーク3拠点で行われるデリバティブ取引を管理するための大規模なシステムである。現行のC／S（クライアント／サーバー）システムは稼働後10年以上が経過し、端末／サーバー機器の保守期限到来などもあって、システム更改されることになった。新システムは、東京にサーバーを一極集中し、各拠点ユーザーはWeb画面経由でサーバーにアクセスするWeb系システムとして設計、2005年4月に開発着手した。途中、詳細設計～実装工程でスケジュール遅延と品質問題が発生したため、IT（Integration Test＝結合テスト）工程を7ヵ月延長してこれに対応することとなった。最終的に1,800人月を超える大規模システム開発となったが、2007年1月に東京、2月にロンドンおよびニューヨーク向けリリースを、それぞれほぼパーフェクトな形で終え、その後もほとんど障害なく安定稼働を続けている。

品質問題の発生について自体はもちろん厳しく反省しなければならない。しかし、その原因の分析や、リカバリとしてどのようにバグを除去しシステムの品質向上を実現したかについても、同様の失策を防ぐためには必要と考え、記事としてまとめることにした。すべてのアクションが予定どおりうまくいったわけではないが、本稿における品質管理の考え方が、今後のプロジェクトにおいてお客様に満足いただけるシステム品質確保のための参考となれば幸甚である。

2．品質管理のフレームワーク

2.1　品質管理の狙い

テスト工程における品質管理の目的を一言で表すと、「検出した障害の定量・定性分析を通してソフトウェア品質の状態を評価し、必要な是正対策を施すことによって、当該テスト工程の目的（スコープ）をすべて達成すること」といえるのではないだろうか。

作業管理の観点には、そのテスト工程で予定したテストをすべて終え、検出したすべての障害の対応が完了すればよい。一方、品質管理の観点では、検出した障害の数量や内容（重要度、発生箇所、事由、検出フェーズ）が妥当なものであるかどうかをよく吟味し、原因を明確にしたうえで対処し、そのテスト工程を完了するに足る品質を確保したと評価、宣言することが必要になる。

そのため、当プロジェクトではこうした狙いを達成するために、次の3層の活動に分けて品質管理を実施することとした。

・個別障害の品質管理
・機能別の品質管理
・システム全体の品質管理

2.2　3層の品質管理

(1) 個別障害の品質管理

どのプロジェクトでも、テストで検出した障害（バグ）ひとつひとつについて、その直接的な原因を解明し、設計書やソース等を修正し、再テストを行うというアクションはとっているはずである。当プロジェクトでは、こうした個別障害の対応を行う際に、根本的な原因まで深掘りして同根の残存障害を取り除くことに努めた。

(2) 機能別の品質管理

当システムは、約100の機能によって構成されている。システム全体の品質状態を可視化するためにも、まずは100の機能単位に分けて、それぞれの品質状態を分析・評価することとした。テスト工程の切れ目（UT：Unit Test＝単体テスト、IT1＝結合テスト1など）では、テストケースと検出した障害の定量・定性分析を行い、次工程への進行に問題がないことを検証することとした。この結果は機能別品質評価報告書にまとめ顧客に提示した。

(3) システム全体の品質管理

個別障害の品質管理や機能別の品質管理を通じて、システム全体の品質状態がどう推移しているのかを明らかにするために、日々発生する障害を、数だけでなく内容に至るまで様々な切り口で分類・集計し、システム全体の品質状態を可視化するようにした。6ヵ月のIT期間中、毎月末のタイミングで実施し、障害傾向や機能別偏向、障害の収束状況などを分析・評価した。これによって、必要に応じて追加テストや実装・設計見直しといった品質強化タスクを組み込み、翌月にその効果もあわせて分析・評価するという品質管理のPDCAサイクルが確立することになった。

以下、品質管理の3層それぞれについて詳細を説明する。

3．個別障害の品質管理

3.1 目的

個別障害の品質管理は、「1つの障害検出で当該1つの障害を修正するだけでなく、同根の残存障害をすべて検出し対応すること」を目的とする。その結果として、各テスト工程のスコープを完全に満たしたことを担保することが最終目的である。ただ単に見つけた障害を修正するだけでは、障害管理の域を出ず、個別障害の品質管理とはいえない。

3.2 障害検出時のアクション

障害を検出した際の基本的なアクションは、下記のとおりである。

1) 障害発生の事由、原因、対応方法、重要度レベル、検出妥当フェーズなどを明らかにして記録する。
2) 障害発生箇所の対応に加え、同じ直接原因による類似の残存障害もすべて検出し対応する。
3) 根本原因の解明によって、同根を持つ残存障害がほかにないことを調査確認し、あればすべて対応する。
4) 当該テスト工程以前に検出すべき障害だった場合は、前工程のテスト漏れを検証し追加テストなどを実施して残存障害を根絶させる。

1)、2) は主に直接原因への対応であり、3)、4) は根本原因への対応である。

なお、障害記録は品質分析の元データとなるため、正確かつ迅速に行う必要がある。そこで、早い段階で障害記録票のフォーマットを統一し、PMO（Project Management Office）チームが一括管理できるようツール化を推進した。障害記録票には、障害個々の事由、原因（直接原因、根本原因）、修正対象、ステータスなどに加え、重要度、発生箇所、検出妥当フェーズなどの障害分析用情報もあわせて保有した（P.49表1、P.50表2）。

UT工程では、機能ごとに一覧表形式で記録したが、IT以降はExcelで作成した単票に記録した。IT以降は複数のメンバーが同じ機能の障害を検出することもあり、また、障害の内容分析も深くなるため単票形式が便利であった。顧客担当者がテストし障害検出した場合も、同一フォーマットの障害記録票を起票し送付してもらうことによって、障害総数を漏らさず把握するよう努めた。

3.3 直接原因への対応について

直接原因とは、設計不良やソースロジックの不具合のように障害発生を引き起こすきっかけになったもので、通常は原因解明～対策確定ののち、設計書／ソースの修正、確認テスト、回帰テスト等を経て対応完了となる。このとき重要なのは、原因解明～対策確定の段階で、他機能において残存するかもしれない類似不良をすべてあぶりだすことである。当システムでは障害記録票に、影響範囲や類似対応の必要の有無を記載する欄を設け、対応内容だけでなく、調査範囲はどこまでとしたか、なぜその範囲でよいとしたか、調査ポイントは何かといった理由・根拠を明記するようにした。こうすることによって、他機能を含めて同様の類似障害がないことを、理路整然と証明できるようにした。

3.4 根本原因への対応について

障害の背景には必ず根本的な問題が潜んでいる。「なぜこの障害が発生したのか」という根本原因を徹底的に追究し、再発防止策を講じるとともに、同根による残存障害がほかにないことを追加レビューやテストを通じて検証・証明した。個別障害の品質管理では、この根本原因への対応が最も重要なアクションであると考える。

根本原因への対応については、「なぜそのような障害を埋め込んでしまったのか」、「なぜそれまで検出できなかったのか（見逃したのか）」、という2つの方向から原因追究を行うことを基本とした。

(1) 埋め込み（設計～実装工程）の原因分析と対応のポイント
- 遡及フェーズ：障害を埋め込んだ設計／製造工程はどこかを特定する。
- 根本原因：人、技術、プロセスなどの観点でなぜそのような不具合を埋め込んでしまったのかについて根本的な原因（技術者のスキル不足、レビュー方法の不備、

表1　結合テスト障害記録票

テストフェーズ	IT2	IT2テスト分類		ケースNO	HO2-175
B票番号	IT2_nnnn	障害が発生した機能	XXX取引解約処理		
B票番号（枝番）		障害概要	xxx取引の解約処理で決済ステータスが不正な値に更新される		
発生日	2006/8/23				

発生ﾗｳﾝﾄﾞ区分	拠点	東京本部	優先度	緊急	起票者	社名	CAC	受付窓口	社名	
	現/新	新	調査期限	2006/8/23		担当者	J		担当者	
	ﾀｽｸID		完了期限	2006/8/25		再鑑者			再鑑者	
関連B票番号			重要度	中	検出妥当フェーズ		UT	添付有無		

事象	障害事象の説明 xxx取引を過去日付で解約したところ、決済済明細の決済ステータスが、"zzz"に更新された。 本来は、"YYY"に更新されるべき。

直接原因調査	社名	CAC	開始日	2006/8/23	類似調査	社名	CAC	開始日	2006/8/23
	担当者	T	終了日	2006/8/23		担当者	T	終了日	2006/8/23
	再鑑者	K	再鑑終了日	2006/8/23		再鑑者	K	再鑑終了日	2006/8/23
対策（設計書）	修正要否	修正不要			対策（PGM/ﾃｽﾄ）	修正要否	修正要		
	社名		開始日			社名	CAC	開始日	2006/8/23
	担当者		終了日			担当者	T	終了日	2006/8/24
	再鑑者		再鑑終了日			再鑑者	K	再鑑終了日	2006/8/24
						CL数		回帰CL数	
IT1テスト	社名	CAC	開始日	2006/8/24	IT2テスト	社名	CAC	開始日	2006/8/24
	担当者	T	終了日	2006/8/24		担当者	J	終了日	2006/8/24
	再鑑者	K	再鑑終了日	2006/8/24		再鑑者		再鑑終了日	
	CL数	17	回帰CL数	16		CL数	2	回帰CL数	2

横展開要否	要	対応担当	Dチーム	ステータス	完了	完了日付	2006/8/24
リリース予定日	2006/8/25	汎用1		非障害扱い	障害	ｽﾃｰﾀｽ内容	

直接原因	PGM種別	画面	PGM ID	OPEnnnnn	原因区分	11プログラミング誤り（仕様誤解）
	カテゴリ	10オンライン			障害発生箇所	08画面コールバック
	内容	①直接原因内容 ・設計書、ソースの不具合箇所と内容				

対策	対策種別	本対応
	内容	①直接原因対策 ・設計書、ソースの修正箇所と修正方法

根本原因	PGM種別	画面	PGM ID	OPEnnnnn	原因区分	2.UT不備・テスト漏れ
	カテゴリ	10オンライン			障害発生箇所	08画面コールバック
	内容 対策	①根本原因内容 ・埋込（なぜ不正ロジックを埋め込んでしまったのか） ・検出（なぜこれまでのテストで検出できなかったのか） ②根本原因対策				

影響範囲	①横展開調査のポイント、調査範囲 ②その範囲に絞った理由・根拠 （必要に応じ「影響調査シート」にも記載）
類似対応必要性等	①類似対応が必要な機能（横展開調査の結果） ②類似対応の内容、結果
特記事項	

表2　障害区分表

直接原因区分	障害発生箇所	重要度レベル	
01顧客提示仕様誤り	01DB更新処理	重大	当システムの機能として成立しない
02要件定義誤り	02DB検索処理		レベルの障害。異常終了、または
03基本設計誤り	03テーブル項目定義		その要因を作るもの
04詳細設計誤り（含む補足説明）	04プログラム項目定義		
11プログラミング誤り（仕様誤解）	05プログラム条件判定	高	他機能へ影響を及ぼすことが
12プログラミング誤り（共通 機能誤使用）	06入力項目関連チェック		明白なもの
13プログラミング誤り（DB項目誤使用）	07入力項目単体チェック		
19プログラミング誤り（ｹｱﾚｽﾐｽ他）	08画面コールバック	中	不正な処理　（DB更新、帳票出力）
21テスト誤り（テストケース設定誤り）	09画面編集		他機能への影響が無いと判断できるもの
22テスト誤り（テスト運用誤り・オペミス）	10計表編集		
23テスト誤り（使用データ誤り）	11計算処理	低	表示上の問題など軽微な障害
29テスト誤り（その他テスト誤り）	12I/F・パラメータ設定（画面）		
31環境設定誤り	13I/F・パラメータ設定（関数・部品）		
32H/W、ﾈｯﾄﾜｰｸ障害	14I/F・パラメータ設定（JNI）		
33S/W(OS、ﾐﾄﾞﾙ)障害	15I/F・パラメータ設定（その他）		
34リリース誤り	21共通機能（画面）		
41現行潜在バグ	22共通機能（関数・部品）		
99その他（原因を記入）	23共通機能（JNI）		
	24共通機能（その他）		
	31環境・H/W・N/W・S/W		
	32運用・LIB		
	99その他（発生箇所を記入）		

コミュニケーションロスによる仕様の誤解など）を追究する。

・類似不良箇所：根本原因と同根の残存障害の調査を行う。必要に応じて埋め込みフェーズまで遡及し、他機能に残存しているかもしれない障害をあぶりだす。

・対応：根本原因に対する再発防止策をたて、残存障害の検出と除去を行う。

(2) 検出（テスト工程）の妥当性と対応のポイント

これは、本来、その障害を検出すべきテスト工程（検出妥当フェーズ）がどこであったかによって、前工程、当工程だがテストスコープ外、当工程でかつテストスコープに合致、の3ケースがあり、それぞれによって、以下のようにとるべき対応が違ってくる。

a. 検出妥当フェーズが前工程だった場合

当該前工程まで立ち戻り、テスト検出漏れの原因を解明し対応を実施する。

・根本原因：なぜ前工程で検出できなかったのかについて、前工程のテスト網羅性や十分性に立ち戻り原因を追究する（必要なテストの切り口・観点の漏れ、テストケース設定漏れ、テストデータのバリエーション不足、テスト結果検証時の不具合見落としなど）を追究する。

・類似不良箇所：同根の残存障害の調査を行う。

・対応：根本原因に対する再発防止策をたて、残存障害の検出と除去を行う。

b. 検出妥当フェーズは当工程だがテストスコープ外だった場合

当工程のテスト仕様を見直し、テストの切り口・観点漏れの検証と追加テストを行う。

・根本原因：なぜテストスコープの漏れが発生したのかについて、当工程のテスト網羅性、十分性に立ち戻り原因を追究する。

・類似不良箇所：ほかに漏れている切り口・観点がないか調査する。

・根本対策：根本原因に対する再発防止策をたて、残存障害の検出と除去を行う。

c. 検出妥当フェーズが当工程でテストスコープにも合致している場合

これは妥当な障害検出であり、前工程の遡及や当工程のテスト仕様見直しは不要。

4．機能別の品質管理

4.1　目的

当システムは、4つの業務（サブシステム）に分かれており、約100の機能（画面）によって構成されている。これらの機能ひとつひとつが、画面入力～DB更新・検索～画面（帳票）出力という一連の処理を有しており、詳細設計や結合テストの管理単位となった。そのため、システム品質状態の可視化にあたり、まずはこの100の機能に分けて、ひとつひとつの品質状態を把握することにした。そして、テスト工程の切れ目（UT、IT1）において、テストケースと検出した障害の定量・定性分析を機能ごとに品質評価報告書としてまとめ、次工程への進行に問題がないことを証明することとした。

4.2　機能別品質評価の内容

機能別の品質評価では、当該機能に関する「テストケース」と「検出した障害」について、定量的な見地と定性的な見地の両面から分析・評価した。なお、定量分析の基礎数値としては、テスト前にあらかじめ定めておいた規模（ステップ数）あたりの目標テストケース数と予想障害数を利用した。これら基礎数値は、当システムの保守を通じて得た実績値や、アーキテクトグループ保有の他プロジェクト情報などをもとに算出した。

(1) テストケースの品質評価

テストケースについては、当該テスト工程の目的を満足するための切り口、観点として漏れはないか、テストケースやデータバリエーションは必要十分かといった網羅性、十分性について分析・評価した。さらに目標ケース数と実績ケース数について比較分析を行い、両者の乖離がある場合は、その理由について明らかにすることで、テストケースとしての信頼性を担保することとした。

(2) 検出した障害の品質評価

障害についても、予想障害数と実績障害数との乖離がある場合は、その理由・根拠をきちんと分析するようにした。予想数に反して実績数が少ない場合は、プログラム品質が高いという見方だけではなく、ケース設定やデータバリエーションの問題でそもそも障害の検出が不完全なのではないかということを疑い分析した。また、障害の内容や種類についても当該テスト工程として、また、当該機能として妥当なものかどうか、発生箇所、原因区分、重要度などについて傾向や偏向を分析・評価し、当該テスト工程で検出すべき障害がほかにないということを根拠づけて説明するようにした。

4.3 機能別品質評価報告書

前述の品質評価の結果をまとめ、100の機能ごとに品質評価報告書を作成した。この報告書の構成は、下記のとおりである。

1) メトリクス情報：ステップ数、テストケース数・障害数の予測数と実績数、内容・種類別のテストケース数
2) テスト概要：テスト目的、スコープ、ポイント、環境、制約条件
3) テストケース分析：ケース数の定量分析、ケース内容の網羅性・十分性に関する定性分析
4) 障害分析：検出した障害数の定量分析、障害内容の定性分析（テスト目的との整合性、検出工程の妥当性など）
5) 結果：結論として当該テスト工程で確保すべき品質に到達したかどうか、次工程に問題なく進行できるかどうかの評価

4.4 機能別品質評価の効果

この機能別品質評価報告書は、主にシステムの4業務のリーダーと主要設計者が作成を担当した。レビューはPMO要員と顧客品質管理担当者が行い、テストの網羅性・十分性の根拠に加え、類似不良の横展開調査や根本原因への対応が妥当だったかどうかについて徹底的に吟味した。矛盾があるものや対応が不足していると思われるものについては、追加テストや設計書・ソースの追加レビューといった確認作業を都度実施してさらなる品質向上を目指した。

当プロジェクトの品質管理の基盤が、この機能別品質評価にあったといっても過言ではない。100の機能について品質状態を横並びで比較検討できたため、低品質の機能に関する傾向や特徴の把握が容易となり、効果的な品質向上策の実施が可能となった。なお、検出障害にどう対応したかではなく、なぜその障害が発生し、なぜほかに障害がないと言えるのかという観点での掘り下げは、将来に向けた欠陥予防につながる重要な考え方であり、PM、リーダー、PMOといった管理者はもちろん、SE、テスト担当者の全員が身につけるべきものだと考える。

5．システム全体の品質管理

5.1 目的

テストや品質管理の活動を通じて、日々、システムの品質状態は変化し続けている。この品質状態の推移をシステム全体の視点で捉え可視化することによって、テストが計画どおりに進捗し、品質管理のPDCAが予定どおりに回っていることを確認することが、システム全体の品質管理の第一の目的である。そのため、IT期間中の毎月末時に、システム全体および4業務の単位で、障害内容や収束状況を分析・評価し新たな品質強化策とともに報告書にまとめ、顧客および社内関係者に提示した。最終目的は、工程の切れ目において、システム全体が当該テスト工程の目的をまっとうし、問題なく次工程に移行できる品質を確保したことを立証することである。

5.2 品質可視化の具体例

品質状態を可視化するために使用したメトリクスの基本項目は「障害数」であるが、必要に応じて「規模（ステップ数）」と「テストケース数」を組み合わせて活用した。定量的な分析・評価のためには数多くのメトリクス項目が必要だという考え方もあるが、まずは基本となるこの3要素をしっかりと計測・把握することが先決だと考える。以下、品質可視化の具体例について説明する。

(1) 障害総数

品質可視化の最も基本となるものが、図1「障害数推移グラフ」である。

テスト工程の進捗（時間の流れ）とともに、障害発生数がどのように推移したのかを見ることで、大まかな品質状態の変化を知ることができる。通常、テスト開始期～中盤期は進捗と共に障害検出も増えていき、テスト終盤になると障害検出が徐々に減り収束に向かう。こうした通常の推移と異なる動きとなる場合は、何らかの問題があるはずとして障害内容の詳細分析を進め、システム全体の品質状態の評価と、テスト計画やリソース配分の見直しを行った。

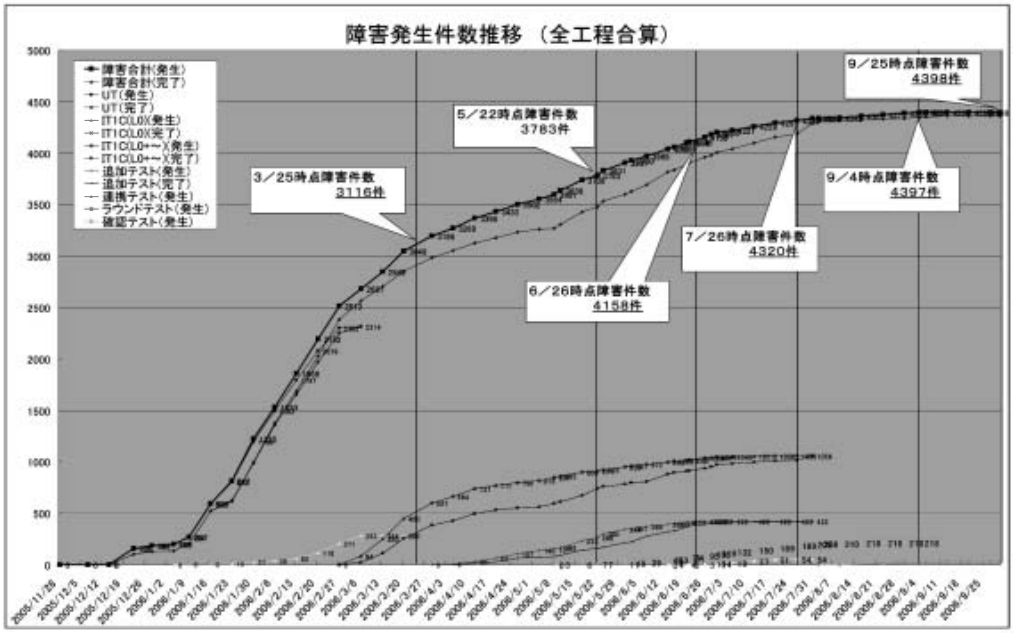


図1　障害数推移グラフ

(2) 重要度別の障害数

障害事象を重要度レベルに応じて4段階（重大、高、中、低）に分け、このレベル別の障害数の推移を月別にあらわしたものが、図2「重要度別障害数推移グラフ」である。また、全体の障害数に占めるレベル別の障害数割合を月別推移であらわしたものが、図3「重要度別障害構成比推移グラフ」である。

テスト開始期は重大レベルをはじめ、他レベル障害も数多いが、徐々に重大レベルは減り、低レベル障害が散見されるようになる。この2つのグラフから障害内容の変化が把握でき、システム品質の向上を質の面から確認することができるようになった。

(3) 発生箇所／原因区分別の障害数

障害の発生箇所や原因については、それぞれ障害記録の時点でいくつかの種類に区分しており、これを定期的にグラフ化して内容の変化を分析・評価したものが、図4「発生箇所原因区分別グラフ」である。この結果、全体傾向として画面（表示項目）編集に関する障害収束が鈍いとか、業務や機能によっては入力チェックの仕様不正が多いといった特徴的な障害傾向が把握でき、追加テストやレビューといった品質強化策を無駄なく効果的に実施することができるようになった。

(4) 障害密度

品質向上の度合いを把握するため、障害総数の推移だけでなく、規模（ステップ数）やテストケース数を母数とした障害密度（障害発生率）の推移をグラフ化した（P.54 図5）。

実装規模割合の障害密度（1,000ステップあたりの障害数）は、業務・機能別に一覧化し、比較検討することによって品質の低い業務や機能を特定することが容易となった。テストケース割合の障害密度（100テストケースあたりの障害数）は、時系列にグラフ化することで障害収束の

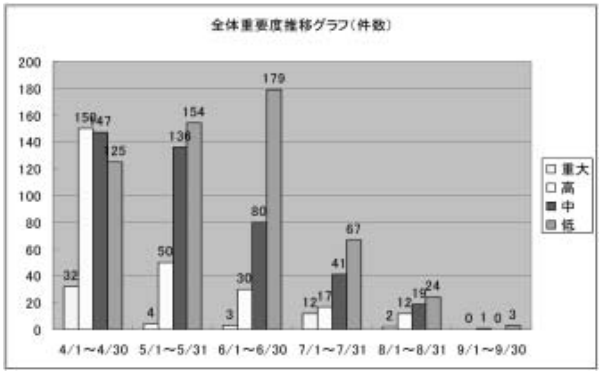


図2　重要度別障害数推移グラフ

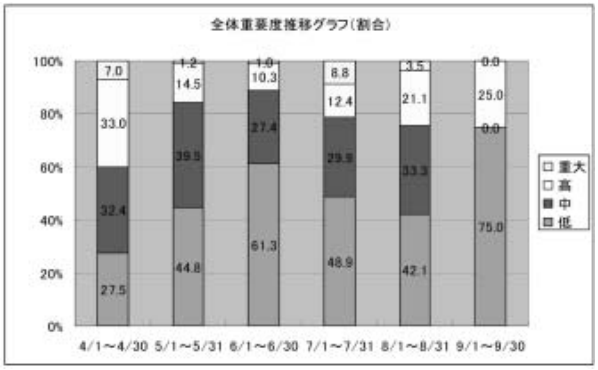


図3　重要度別障害構成比推移グラフ

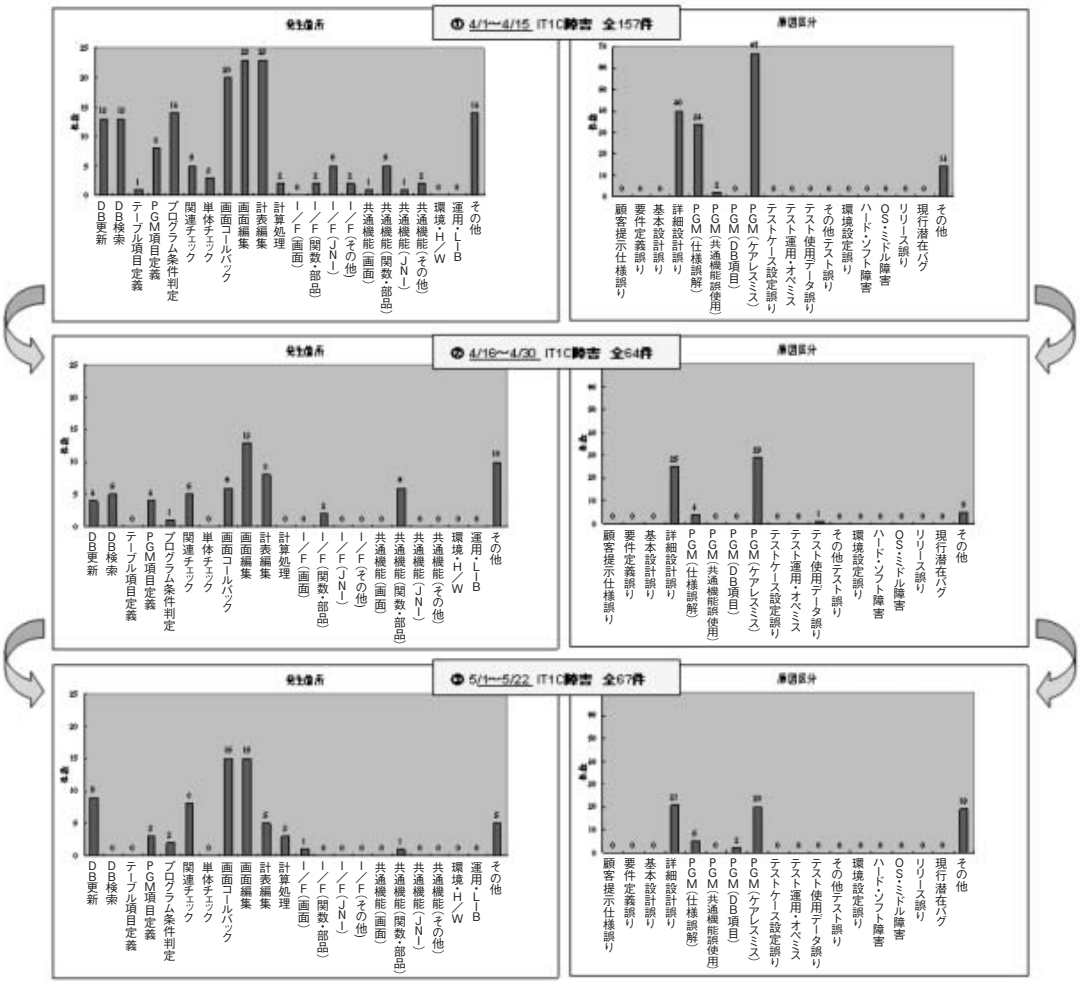


図4　発生箇所原因区分別グラフ

過程を可視化でき、品質向上を知らしめるのに役立った。

5.3　障害収束（テスト完了）の基準

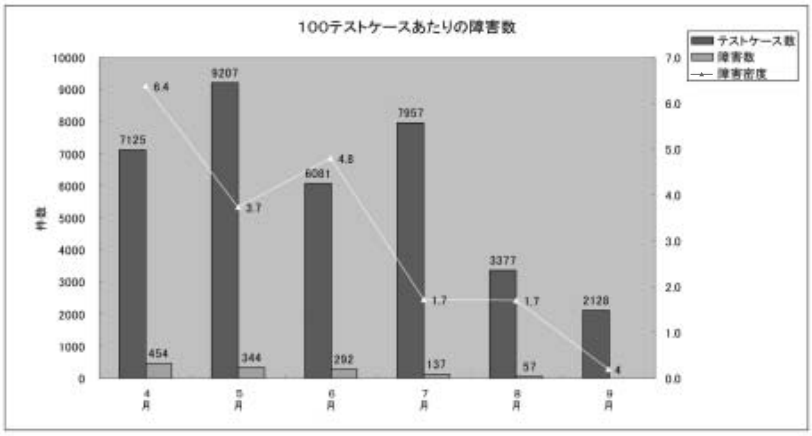
いくつかの切り口、観点でシステムの品質状態を可視化し、全体の品質管理を進めたが、最終的にすべての機能・業務・全体として残存障害の検出が進み、十分収束したことを宣言するために統括PMOと協議して次のように定量基準（IT2完了基準）を定めた。

「テストケースの後半10%で検出した障害数が、障害総数の1%以内であれば、障害が十分収束したものと判断する」

そのため、予定したテストをすべて完了した機能であっても、この基準をクリアできていなければ、さらに新たなデータバリエーション等を追加してテストを繰り返した。結果としてすべての機能がこの基準をクリアし、実感をもって品質向上を確認できただけでなく、顧客に対しても強い説得力をもってIT2完了の宣言をすることができた。

6．質のよいシステムとは

これまで述べてきた品質管理は、システムからバグ（設計不良やコード不良によるシステム障害）を取り除くことを目的としたものである。当然ながら、ユーザーの利用に耐えないほど障害が頻発するシステムは論外だが、バグの少ないシステムが即ちお客様にとって「質のよいシステム」かというと、必ずしもそうではないことを我々は認識すべきである。システムによっては障害除去に時間をかけるよりも早期のサービスインを優先したり、使い勝手や維持・保守性の向上のために多くのリソースをかけたりすることもある。お客様に質のよいシステムだと評価してもらうには、バグが少ないことは必要条件ではあるが、それ以外の十分条件も揃うことが必要だ。こうした十分条件が何であるかはそのシステムの利用目的によってさまざまである。戦略系システムであれば速いサイクルの戦略見直しにも即座に対応できるシステム構造の柔軟性かもしれない。業務系システムではさらなる効率化を実現するための操作性や



テスト種類		4月	5月	6月	7月	8月	9月	合計
IT1(CAC)	テストケース数	7125	8826	1917	1519	20	0	19407
	障害数	454	333	185	50	0	0	1022
	障害密度(/100cl)	6.4	3.8	9.7	3.3	0	0	5.3
IT1(顧客)	テストケース数	0	381	4164	6438	1078	0	12061
	障害数	0	11	107	87	13	0	218
	障害密度(/100cl)	0	2.9	2.6	1.4	1.2	0	1.8
IT2	テストケース数	0	0	0	0	858	104	962
(ﾗｳﾝﾄﾞ)	障害数	0	0	0	0	22	1	23
	障害密度(/100cl)	0	0	0	0	2.6	1.0	2.4
IT2	テストケース数	0	0	0	0	1421	2024	3445
(確認)	障害数	0	0	0	0	22	3	25
	障害密度(/100cl)	0	0	0	0	1.5	0	0.7
合計	テストケース数	7125	9207	6081	7957	3377	2128	35875
	障害数	454	344	292	137	57	4	1288
	障害密度(/100cl)	6.4	3.7	4.8	1.7	1.7	0.2	3.6
	障害密度(/Kstp)	1.0	0.8	0.7	0	0	0	3.0

図5　障害収束グラフ

大量データの処理速度かもしれない。

当システムは、非常に多くの入力項目や画面遷移があり、更改前のシステムのユーザーからは画面操作性の向上を強く要望されていた。さらに以前のC／Sでは現地（東京、ロンドン、ニューヨーク）にサーバーを設置していたが、今回の更改によって東京に設置したサーバーに海外からWebアクセスされることになり、画面の操作性だけでなく、応答速度の維持・向上も重要な課題となった。そのため、業務系のテストと並行して、画面操作性の改善タスクとパフォーマンス検証・改善等の非機能テストには十分なリソースを配分した。結果としてバグが少ないだけでなく、操作性や応答速度に優れたユーザー満足度の高いシステムを構築することができ、リリース後のお客様評価も上々のものとなった。

システムの利用目的をお客様の視点で捉えたとき、そもそもこのシステムに何を期待し何がやりたかったのかが見えてくるはずである。その目的を達成しなければ、たとえバグのないシステムを納品しても、決して質のよいシステムだという評価はいただけない。通常、システムの本来の目的はお客様の中では明確になっているものだが、プロジェクトが設計～実装～テストと進むうちに、プロジェクトメンバーの間ではシステムを無事リリースすることが目的となってしまいがちだ。バグの除去に相応のリソースを配分することは当然だが、お客様が真の満足を得るため、全工程を通じて常にシステムの本来の目的を見失わないようにすることが品質管理の本質ではないかと考える。

7．終わりに

本稿では、品質問題が発生した後のIT工程における品質管理手法を中心に説明した。しかし、「品質は作りこむもの」と言われるように、本来、要件定義～設計～実装といったモノ作り工程における品質保証アクションが最重要であることは言うまでもない。実装は設計を満足し、設計は要件定義を満足し、要件定義はお客様の要求仕様を満足していること。こうした基本的なポイントをしっかりレビューしておくことが、システムの品質向上には必要不可欠である。また、こうしたレビューを意味あるものにするには、そもそも要件定義や設計の工程で何をしなければならないのか、成果物に何を記すべきなのかといったソフトウェアプロセスについて、工学的な理解ができていることが前提条件になる。こうした前提条件がそろってはじめて、本稿で述べた品質管理の活動が意味を成すということを忘れてはならない。