障害発生を防ぐために

～長期間の受託システム開発を通して発生した障害内容から考察する～

（株）九州地区農協オンラインセンター

■ 執筆者Ｐｒｏｆｉｌｅ ■

2007 年　（株）九州地区農協オンラインセンター入社
業務部 IT 班・開発担当

佐藤　拓也

■ 論文要旨 ■

システムの障害を分析し検討することで、障害を減らすための効果的な対策を打ち出すことができる。本論文では、大規模な業務システム開発を通して発生した障害原因を分析し、対策を打ち出した。障害の原因は「不注意」、「調査検討の不足」で全体の 50%を占めていることが調査により分かったため、その障害を防ぐことで、大幅に品質改善ができると判断した。業務ごと、工程ごとの詳細な分析・検討から以下の７つの事項が効果的な対策であると結論付ける。

(1)各業務の優先度付け、作業の優先度付けの実施。(2)仕様書作成者による検証実施。(3)パッケージソフトは要件定義完了後に決定。カスタマイズ工数が大きい場合導入見送りを検討。(4)仕様書・テスト仕様書のレベル統一化基準作成。(5)画面項目、テーブル項目に対するチェック内容を記載するレベルまで落としたテスト仕様書の作成。(6)調査方法の妥当性チェック実施。(7)仕変発生分プログラムの顧客による検証実施。

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

１．１　当社の概要

九州地区の７県 JA、信連の信用事業に関するシステム会社として、昭和 52 年 10 月 1 日に設立された。主に以下の事業を行っている。

(1)　九州７県 JA・信連の信用事業に関する情報処理システムの開発、保守、運用
(2)　ネットワークの設計・管理・運用・監視
(3)　計算事務の受託
(4)　情報提供サービス業務の受託
(5)　ソフトウェアの開発・販売
(6)　施設の賃貸並びに受託管理

１．２　受託業務システム開発の経緯

今回分析対象とする業務システムは以前から受託していたが、旧システムは当社にて開発したシステムではなく、またドキュメント類も少なかったため、メンテナンスや運用の面で非常に不都合が多かった。また、使い勝手の面やサーバの性能面の不安、ワークフローやグループウェア等の導入もあり新たなシステム開発を行うという運びになった。

開発においては、基本的に旧システムにドキュメント類があまり存在しなかったこと、大幅な仕様変更はないという前提のもとで、プログラムから仕様書を起こし開発を行っていくということで決定し作業を行った。

１．３　受託業務システムの基本情報と特徴

受託業務システムのサーバ構成、開発言語、データベース、業務分類、利用者数等は以下のとおり。

(1)　サーバ構成・ネットワーク構成を**図１**に示す。

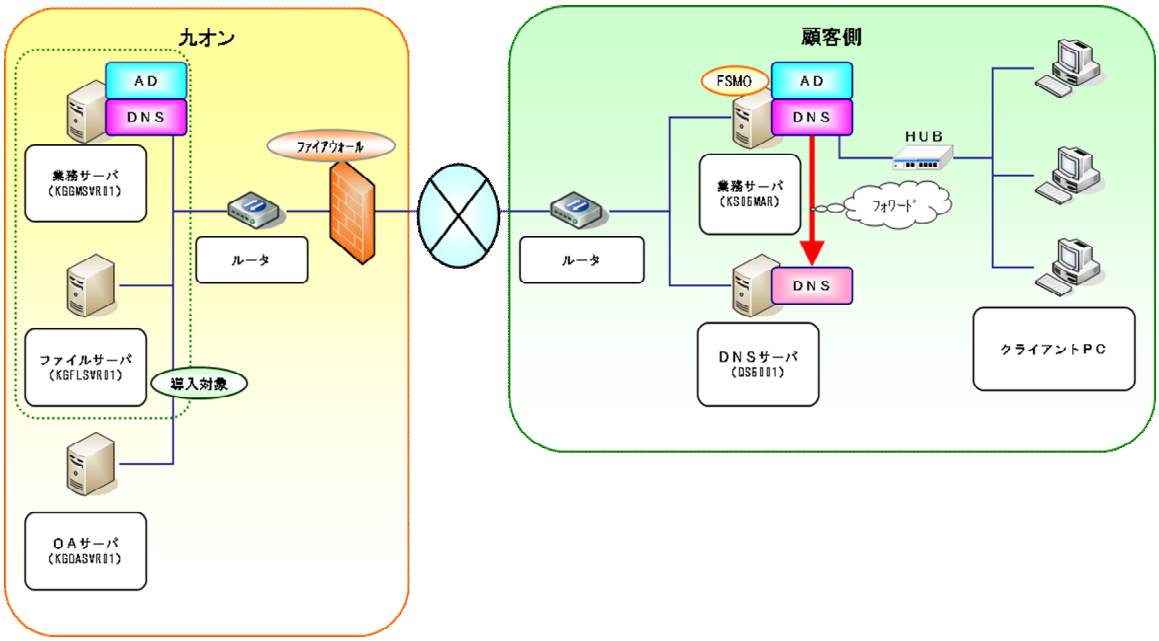

図１．ネットワーク構成図

(2)　サーバ OS　　：Windows Server 2003
(3)　クライアント OS　：Windows2000
(4)　開発言語　　：VB6.0（ワークフローのみ Java）
(5)　データベース　　：SQL Server 2005
(6)　業務分類と開発規模を**表１、表２**に示す。

業務名	VBP 数	画面数
公庫業務	41	52
農業改良資金	15	43
県公金収納	11	13
店舗人事情報	24	30
現金輸送授受管理	10	15
貸出金残高管理	10	10
有価証券	37	65
斡旋品管理	8	11
文書管理	3	9
国庫金年金	4	4
貯金奨励金	3	3
為替受発信管理	9	13
事業計画策定	2	11
農協残高試算表	4	7
経理業務	64	85
サーバ処理	13	0
合併移管処理	4	9
合計	262	380

表１．業務分類と開発規模（VB）

業務名	クラス数	メソッド数	書類数
ワークフロー （カスタマイズ対象数）	22	150	23

表２．業務分類と開発規模（Java）

(7)　クライアント数：124
(8)　当システムの特徴
・ VB で作成した経理業務システムと、WEB のワークフローシステムとの連携により出張や予算の管理などが効率よく行えるようになっている。
・ グループウェア、ワークフロー、ファイルサーバ等も含め顧客のすべての業務を受託し、運用している。（ファイルサーバのみ来年度稼動予定）

１．４　本論文の目的

問題を振り返り対策を立てるということは、よりよいシステム作りには必ず必要である。実例を探してみると、障害の内容がどの工程に起因するかということや、どういった箇所に発生したかということに対して統計を取っている資料は多くみられたが、その根本的な

原因の部分に言及して記述したものは少なかった。そのため、実際に稼働後に起きた障害とその原因、それから導かれた対策という生きた開発参考書を作りたい、ということが本論文の目的である。

７月をもって、新受託業務システムが完全本稼動を迎えた。様々な取引パターンやエラー発生を想定してシステム開発を行い、時間をかけてテストを行ってきた。顧客からの反応としては、大きな障害が少ないため概ね成功と思っていただけている。私としても開発規模からすると障害の件数は少なかったと感じているが、数十件の障害を発生させてしまったため、完璧なシステムということになるとまだまだ程遠いといえる。そこで発生した障害を整理することにより、その発生原因＝問題点を究明し、今後改善する必要がある点を洗い出し、次回以降の開発に活かせるようにしたいと考えている。

２．開発体制等

２．１　開発体制とスケジュール

２．１．１　開発体制

受託業務システムの開発体制を図２に示す。

図２．開発体制図

人数が少ないことから、一人の担当者で複数の業務・工程を担当して作業を行った。

２．１．２　開発スケジュール（概要）

大まかな開発スケジュールを図３に示す。

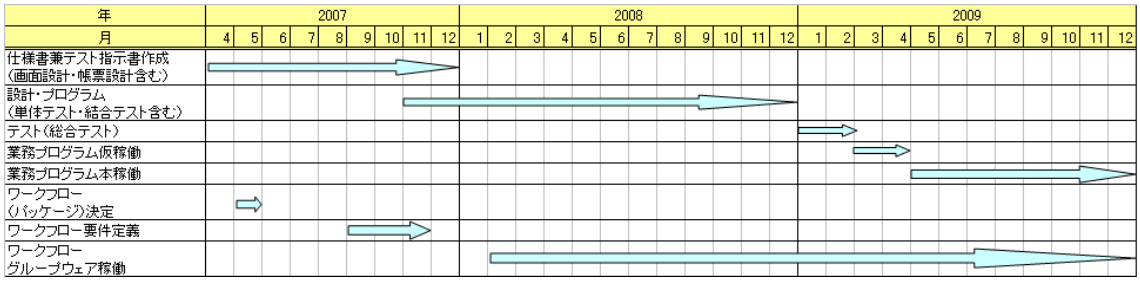

図３．開発スケジュール（概要）

２．２　開発における工夫

開発期間は２年間と長期間ではあったが、人員が少ない上に規模が非常に大きかった。そのため通常の開発と比べ一部効率化を行うことによって対応を行い、また標準化・自動

化により作業の簡略化を図った。その他、作業の平準化及び成果物の客観的チェックという観点から作業分担を考案した。具体的には以下に挙げることを行った。

(1)　作業工程の見直し

一般的な作業工程に対し**図４**のとおり工程の統合、簡略化等を行うことにより効率化を図った。

図４．作業工程

具体的には以下の点について工程を変更している。

- 仕様書を「仕様書兼テスト指示書」とし、仕様書作成作業とテスト指示書作成作業をまとめた。テスト項目を見て仕様が分かりプログラミングまで行えるレベルで作成した。そのため、プログラム構造設計を省いた。
- プログラミング、単体テストを１人で行う体制をとった。
- 結合テストを必要としないプログラムも多かったこと、プログラミングと単体テストを１人で行ったこと等に対して品質を落とさないために、仕様書作成者によるプログラム検証を設けた。

(2)　標準化、自動化を進めた部分

- VSS によるソース管理を行い、ソースのバージョン管理を自動・かつ確実に行えるようにした。
- SQLServer のユーザ定義関数を使用し、SQL 文作成の効率化、ミスの軽減を図った。

３．開発作業における障害の分類と集計結果

障害の発生源を特定するために、まず障害を「１．業務別」「２．発生工程別」それぞれについて分類を行う。今回の分析に利用したデータは、稼働から３カ月に発生した障害であり、全体の件数としては 43 件となっている。

３．１　業務別分類

業務別の分類結果を以下の**図５**に示す。

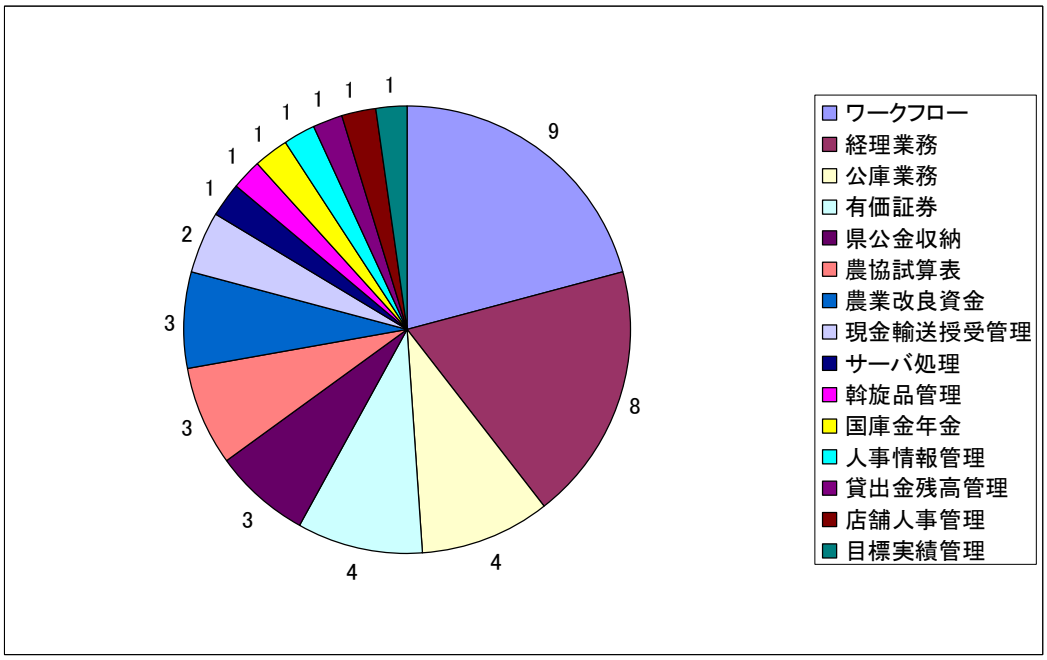

図５．障害発生（業務別）

ワークフローと経理業務、公庫業務、有価証券で全体の過半数を占めていることが分かる。上位の４業務は開発規模も上位の４業務であるため、開発規模と障害発生数は関連性があると言える。その他、２件以上の障害が発生している業務としては、農業改良資金、農協試算表、現金輸送授受管理、県公金収納がある。また、**表１、表２**に記載されていて**図５**に表れていない業務については、障害は発生していない。

３．２　発生工程別分類

発生工程別の分類結果を以下の**図６**に示す。

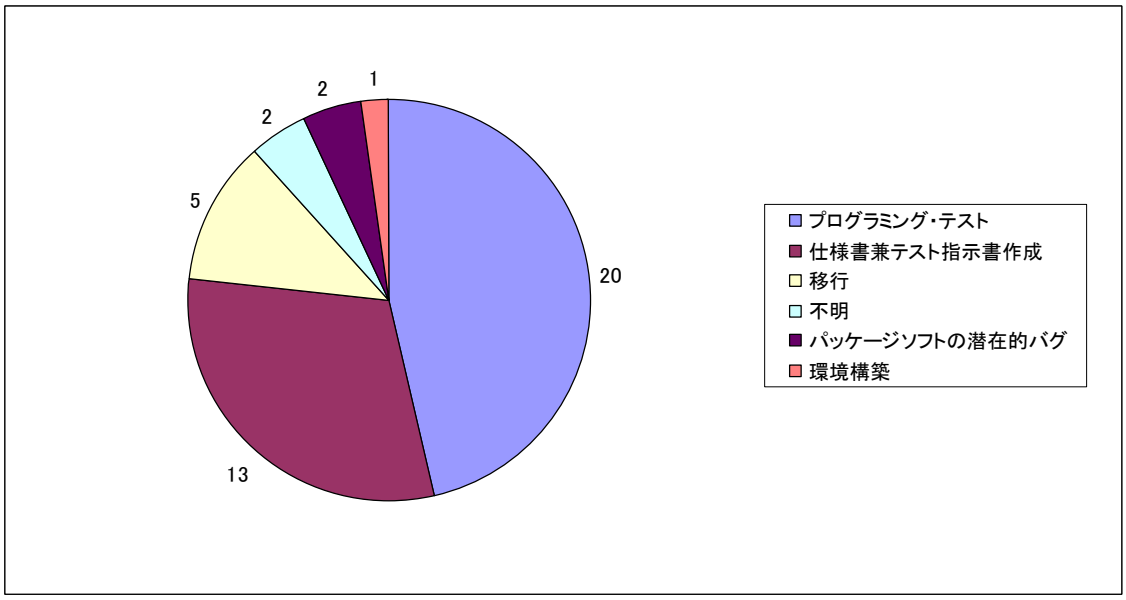

図６．障害発生（工程別）

仕様書兼テスト指示書作成、プログラミング・テストで全体の 75%となっており、ほとんどの障害はこの工程の作業に起因して起きているといえる。

４．障害の発生原因と対策

３章でまとめた障害の発生と、実際の作業方法の関連について考察していく。障害の原因については[1]畑村洋太郎「失敗学のすすめ」の 10 の失敗原因を元に**表３**のとおり分類した。（一部変更・追加あり）

No	障害原因
1	不注意
2	調査・検討の不足（調査・検討の掘り下げ方が足りなかった）
3	手順の不順守（決まった手順があったにも関わらずその手順を守らなかった）
4	環境の相違・変化（開発環境では問題なかったが、本番の環境で障害になった）
5	無知（個人の知識不足による）
6	未知（現在もはっきりとした理由が分からない）
7	認識相違（顧客との認識が異なっていた）
8	パッケージソフトの不具合（不可避）

表３．障害原因一覧

４．１　業務別分類結果と対策

特に経理業務とワークフローに障害の発生が集中していることが、「**図５．障害発生（業務別）**」から分かる。そのため、経理業務とワークフローの障害について、掘り下げて考察していく。

４．１．１　経理業務の障害発生原因と対策

経理業務については、旧システムと仕様は変わらないということ、また、非常にプログラム数、画面数が多い業務であるにも関わらず時間的に厳しいという理由から、仕様書は作成せず、仕様書＝旧システム、ということで作業を進めた。障害原因に関しては「不注意」、「調査・検討の不足」が多い。（**図７**参照）また、実際に障害の詳細を見てみると３件がプログラム解析時の仕様の取り違えによって発生している。そのことからも、やはり旧システムのプログラムを解析しながら製造作業を進めるというやり方は非常に困難であるということが分かる。

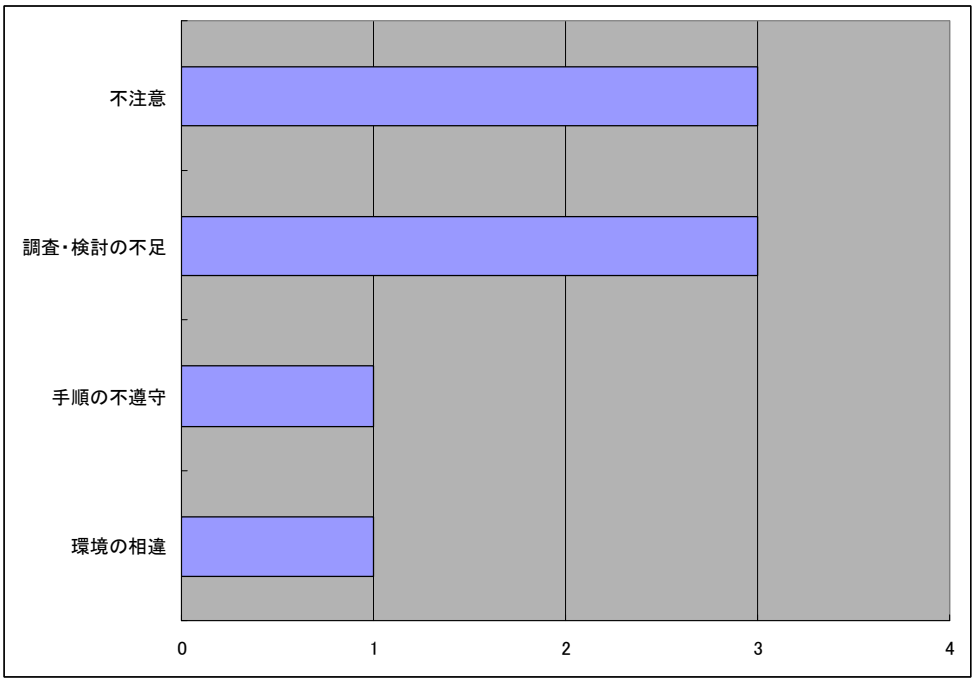

図７．経理業務の障害発生原因

更にここから掘り下げて原因と対策を考える。

(1) 原因１

開発の経緯から仕様書なしが問題だったということは明らかだが、ではなぜ「仕様書なし」で作業を進めるに至ったか、ということが重要な問題となってくる。仕様書なしと決めた一番大きな理由は、時間的制約があったためである。経理業務は他の業務とも一部関わってくるコアな業務であるにも関わらず、開発を最後に回したというところで時間的制約が生まれてしまった。本プロジェクトでは、経理業務に全員で集中して取り掛かるために、まず他の業務をすべて片付けてから、という考えで進めた。しかし、業務の重要度を考え優先順位をつけてから作業分担を考え、それから開発作業に取り掛かればこのような障害は起きなかった可能性は高い。というのも、重要度の高い業務の場合、時間のかかる検討事項が生じる可能性があり、またトラブル発生時のインパクトが大きくなるからである。実際、開発している途中で発生した検討事項に対して顧客からの長時間の回答待ちが発生したことや、経理業務の影響で更に他の業務に影響を与えたような修正も開発中にいくつか発生したことからもそのことが分かる。また、上記のようなトラブルは事前に予測できるにもかかわらず、その影響を事前に読み切れず、時間がないからという理由で「仕様書なし」で作業を進めると決定したことも大きな問題であると言える。

(2) 対策１

上記の原因を考えると、最初に経験のあるリーダーを含めた会議を行い、業務の優先度をつけ、それに応じてスケジュールや作業分担、作業の優先度を決めるということが有効な対策といえる。

(3) 原因２

仕様書を作成せずに作業を行ったため、仕様書作成者による検証を行えなかったということも経理業務で障害が多くなった大きな理由としてあげられる。仕様書作成者による検

証が入らない状態で総合テストを行ったため、その中で多くのバグを潰していくということになった。本来なら総合テストまでに潰しておくべきバグが総合テストで多発したことで、総合テストの進捗が大幅に落ち、それに伴い品質が大幅に落ちたことは間違いないと考える。一般的に、バグの処理工数は、後工程に進むほど跳ね上がっていくと言われている。（上流で作成したものを利用し下流工程に進むために、上流でバグが存在すると、そのバグは雪だるま式に膨れ上がることになってしまう）つまり、できるだけ早い工程でバグを潰すことがシステムの安定、また開発の効率化につながるといえる。

(4) 対策２

上記の原因に対して、仕様書及びテスト仕様書を作成し、単体テストをテスト仕様書通りに確実に行う、仕様書作成者によるプログラムの検証を行うというミス発見の取りこぼしを確実に減らす仕組みがあれば品質を大幅に上げることができる。

４．１．２　ワークフローの障害発生原因と対策

ワークフローは旧システムには存在しなかった新システムであり、他の業務と異なりパッケージソフトを使用した WEB システムである。この業務での不具合は他の業務のものと異なり、パッケージソフトの潜在的バグであったり、パッケージソフトの仕様に対する理解不足が原因であるものが多かった。（**図８**参照）

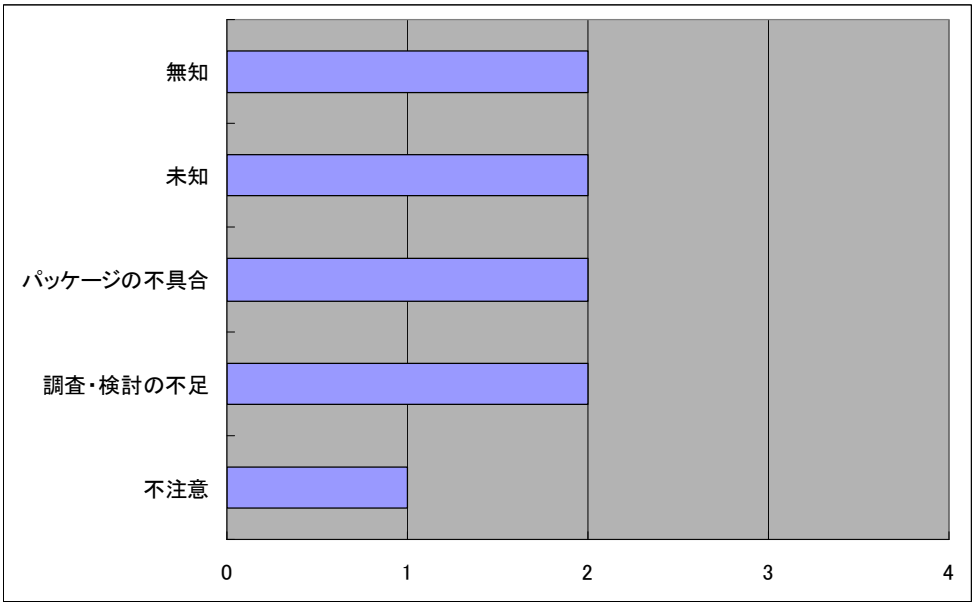

図８．ワークフローの障害発生原因

(1) 原因

本プロジェクトにおいて、ワークフローのカスタマイズはメーカーではなく社内で行い、また１人のメンバーで行っていた。しかしこの点に関しては、開発体制の問題というよりむしろパッケージ導入の経緯にあると考える。通常パッケージソフトは仕様がしっかり固まった段階でそれに合うソフトを探し、どうしても必要な部分のみカスタマイズしてもらうということが望ましい。しかし、今回は要件定義を正式に行う前にパッケージの選定を先に行った。（「**図３．開発スケジュール（概要）**」参照）そのため、要件定義の段階で当初予想していた仕様から大幅に変更が発生し、パッケージソフトの仕様と合わなくな

り、それに対してカスタマイズを重ねた。その結果として障害が発生したと考えられる。

(2) 対策

上記の原因から、パッケージソフトの導入を以下のポイントに留意して行えばこの問題は回避できると考えられる。

① 機能要件を確実に顧客と詰め、その要件に合うソフトを探す。

② 複数のソフトを顧客に使用していただき、その上で導入を決定する。

③ 導入前にカスタマイズポイントを明確にし、必要な追加費用を割り出す。

４．２　発生工程別分類結果と対策

「**図６．障害発生（工程別）**」から、「プログラミング・テスト」と「仕様書作成」が障害発生の起きる主な工程だということが分かる。そのため、その２つの工程に対して掘り下げて考察する。

４．２．１　プログラミング・テスト時の障害発生原因と対策

プログラミング・テスト時の障害発生原因は圧倒的に「不注意」によるものが多い。次に「調査・検討の不足」、「環境相違」という順になっている。（**図９**参照）

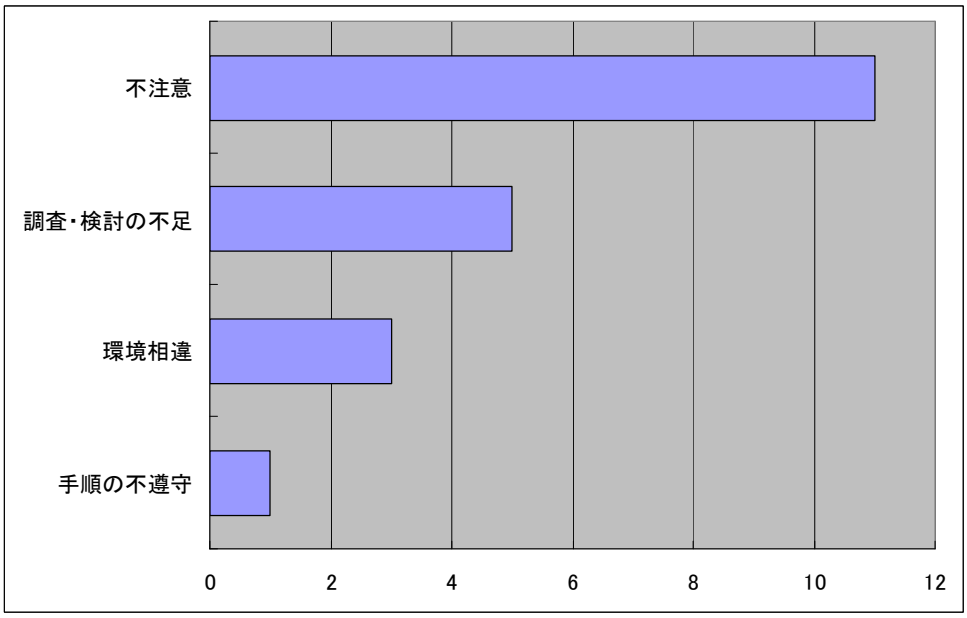

図９．プログラミング・テスト時の障害発生原因

(1) 原因１

プログラミング・テストの作業は仕様書を元に行い、ソースレベルの設計書（プログラム構造設計書）は作成しなかった。そのため、複雑なロジックを必要とするプログラムで「不注意」が原因の障害が多く発生したと考えられる。

(2) 対策１

「不注意」による障害はプログラム構造設計書が存在すれば減らすことはできたかもしれないと考える。しかし、プログラム構造設計書は必要かという点に関しては、私は必ずしも必要ではないと考えている。というのも、プログラム構造設計書を作成する、つまりプログラムの条件分岐やループなどについて設計書として落とすとなると、メンテナンス

に膨大な時間がかかる上に、効率よいプログラムを組む際の制約になってしまいかねないと考えるからである。また、プログラム構造設計書作成時の「不注意」によるミスもコーディング時と同様に発生する可能性もある。そこで、開発効率を落とさずにプログラミング・テストの障害を減らすための対策として、仕様書、テスト仕様書を高いレベルで統一化することを提案する。設計書を作成しなくてもコーディングできるレベル、つまり画面項目に設定するデータやテーブルに入れるデータを記述するレベルまで落とした仕様書を全業務において作成することで、ロジックの制約をつけずにかつコーディングまで確実に行うことができる。仕様書作成に時間はかかるが、後のプログラム構造設計書作成を省けることや仕様に即したプログラミングを行えるなどのメリットがあるため、効果は大きいと考える。また、この対策は本プロジェクトにおいて実行していたが、統一基準が明確ではなく、業務によってばらつきがあったため障害が発生したと考えている。本プロジェクトで障害が発生したため有効ではない、という判断はしていない。

(3) 原因２

全体の基準として共通のテスト項目は作成されていたが、個々の項目についてマックス値のチェック、０件チェック等が個別のテスト仕様書に記述されていなかった。「不注意」による障害の中の数件はやはり単純なオーバーフローや決済データの項目長を間違っていた、データの件数によって行えない処理がある等であったことを考えると、テスト仕様書の作成レベルも障害の原因であると考えられる。

(4) 対策２

上記の原因から、テスト仕様書の中に画面・テーブルの項目単位でテスト項目を設定し、それを元にテストを行うことで、障害を間違いなくなく減らすことができると考える。テスト仕様書の作成に大きい労力を使うことになるが、テスト作業の効率化につながることを考えると作業工数的にも問題はないと考える。

４．２．２　仕様書作成時の障害発生原因と対策

仕様書作成時の障害発生原因は「調査検討の不足」が一番多い。また、顧客との「認識相違」による障害も４件と多くなっている。（図 10 参照）

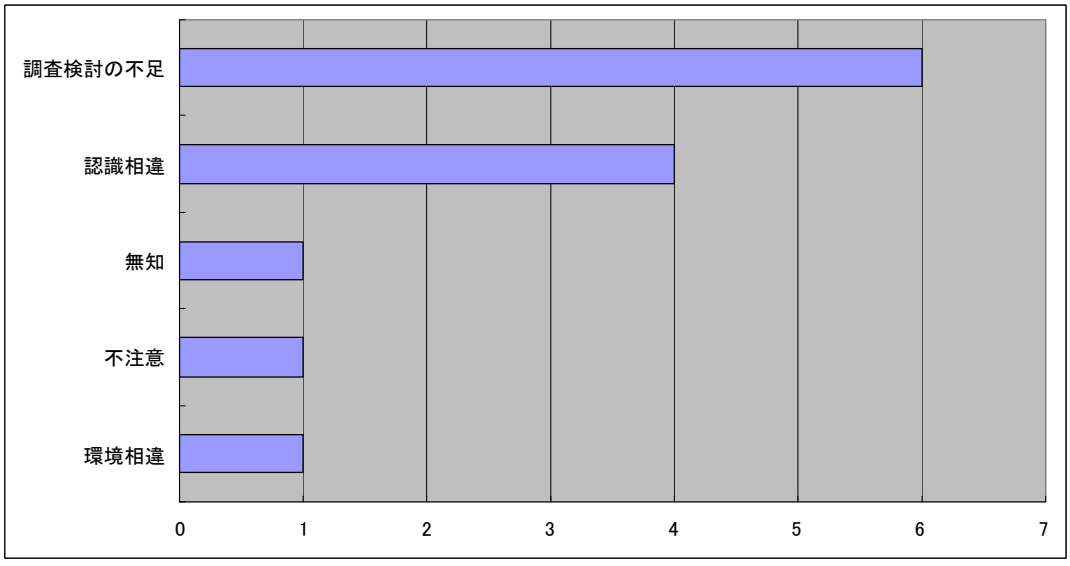

図 10．仕様書作成時の障害発生原因

今回、１番多い原因であった「調査検討の不足」、次に多かった「認識相違」について具体的な障害内容と対策を考察する。

(1) 原因１

「調査検討の不足」について、具体的な障害内容を見てみる。具体的には、調査漏れが４件、実データの調査不足が１件、仕様変更ミス１件となっている。調査漏れに関して、調査は１度しか行わないことが多いため、その調査の妥当性をチェックする仕組みが必要となると考える。

(2) 対策１

影響調査を行う際は調査資料を作成し、更にその業務に関わっている人全員にその資料を見てもらうことにより、影響がないかどうかの確認だけではなく、調査方法が妥当かどうかということのチェックもできる。例えば、あるテーブルの変更が発生した時は、以下のような手順で整理することが効果的である。

①　そのテーブルを使っているプログラムをまず GREP をかけてすべて洗い出す。
②　洗い出されたプログラム内でそのテーブルがどのように使われているかを整理する。
③　実際に変更しても問題がない動きをするということを確認する。
④　①～③のことを資料にまとめ、メンバーに確認してもらう。

それぞれの手順の効果は、「①：影響の可能性のある範囲をチェック」「②：影響の理論上のチェック」「③：実際の影響チェック」「④：妥当性チェック」となる。影響調査漏れは、実際に修正した部分そのもののチェックではないため非常に起こりやすいミスであるが、障害が起きた時に、大きく被害の広がる可能性のある部分でもある。そのため、上記のようなチェックを行う体制作りと影響調査の手順書を準備することで、間違いのない調査を行えるようにする必要がある。

(3) 原因２

「認識相違」について具体的な障害内容を見てみる。具体的には顧客からの回答の取り間違いが２件、口頭でのやりとりによる間違いが１件、顧客内の内部通知ミスが１件となっている。これらについては、単純な顧客とのやり取りの方法の問題や、やり取りの不足が原因といえる。

(4) 対策２

顧客とのやり取りは必ずドキュメントに残しそれに基づいて作業を行う、仕様書レベルでの変更があった際は、仕様書作成による確認はもちろんのこと、出来上がったプログラムを直接顧客に動かしていただき、お互いの認識が間違っていないことを確認するということで「認識相違」による障害は回避できる。具体的には、QA 表の中に顧客による成果物確認終了をチェックできる項目を作ることが、顧客の意識も高めることができるため効果的である。

４．３　全体を通しての障害発生原因と対策

ここまでは、障害発生の業務別・工程別に分けてどのような原因が多いか、またその原因と作業内容にはどのような関係がありどのような対策が必要かを記述してきた。ここで、全体の障害発生原因について**図 11** に示す。

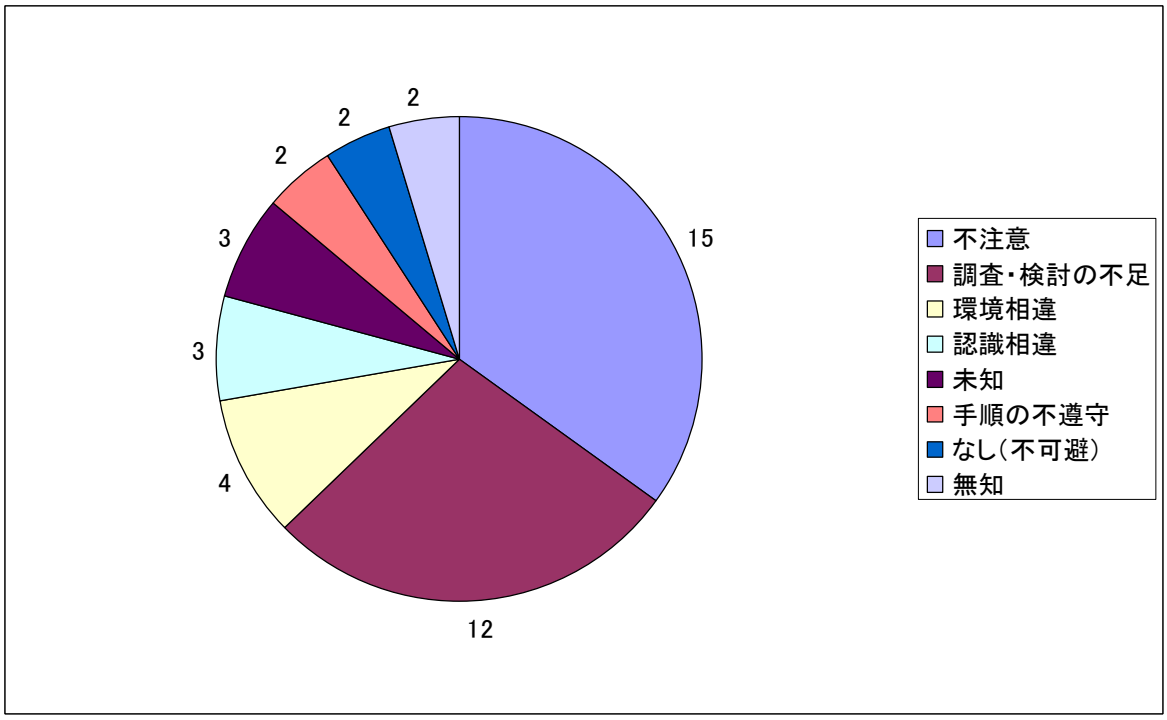

図 11. 障害発生原因

前項までに具体的な発生原因と対策を述べた。対策については特に原因の中で多かった「不注意」による原因の障害と「調査・検討の不足」が原因の障害を中心に記述した。やはり全体の件数から見ても、業務別・工程別で整理した結果と同様「不注意」と「調査・検討の不足」の障害が圧倒的に多く過半数を占めている。つまりこの２つの原因を確実に潰すと単純計算で障害も半分に減らすことができるということになる。前項までに、「不注意」と「調査・検討の不足」への対策は記述しているが、その内容を徹底することが障害をなくすことに大きくつながるということが、全体を通してもいえることが分かる。

５．障害発生を防ぐために

５．１　障害発生を防ぐ７つの事項

前章までで障害を防ぐために何が必要だったかということを様々な角度から検証してきた。それらをまとめると、以下の７つの事項について確実に守って開発することで大半の障害は防げるという結論に至った。

(1) スケジュール作成及び作業分担、作業方針決定の前に経験のあるリーダーやメンバーを交えた会議を行い、各業務の優先度付けや作業の優先度付け、作業方針の妥当性を検討する。

(2) 仕様書作成とそれに基づいた仕様書作成者の検証は必ず行い、バグ発見の取りこぼしを防ぐ。

(3) パッケージソフト導入に際しては、要件を明確にした上で決定する。カスタマイズが大きくなる場合はパッケージソフトのメリットはない。

(4) 仕様書・テスト仕様書を項目設定内容まで記載するというレベルで統一し、安定した

成果物が出来上がるようにする。

(5) テスト仕様書には、画面項目、テーブル項目に対するテスト内容レベルまで記載する。

(6) 影響調査が発生した場合は、調査内容のチェックだけではなく、調査方法の妥当性もチェックする。

(7) 仕様変更があった箇所、また複雑な箇所は、顧客に実際にプログラムを動かして確認してもらう。そのために、QA 表の中に、顧客による成果物確認の要否、成果物確認の終了が分かる項目を追加しておく。

この７つの事項は一つのプロジェクトの統計結果から導き出した内容であるため、元となるデータ量としては少なく、また偏りがあることは間違いない。そのため、必ずしも汎用的であるとはいえないかもしれない。しかし、以下の点から、今回まとめた内容が他プロジェクトでも活きてくることは間違いないと結論付ける。

① 開発規模、開発期間が非常に大きかったこと

② 一般的にも今回まとめた障害に近い障害が多数報告されていること
具体的には２００８年７月の東証の障害のような単純な設定ミスをテストでも見抜けなかった事例や、２００２年４月のみずほ銀行統合の際、方針決定が遅くなり、調査やテストの不足により大規模障害が発生、さらに二次障害も発生したという事例から、今回と規模の違いはあるが、一般的にも同様の障害が発生していることが分かる。

③ 過去にも同じような障害が起きていること
今回開発を行ったシステムの旧システムにおいて、「プログラムを変えた際同時に定数を変える必要があったが抜けていた」、「仕様変更の際、修正すべき帳表の変更が漏れていた」等、不注意や調査検討の不足による障害が多く発生していることから、旧システムにおいても同様の障害が多数発生していたことが分かる。

この７つの事項を開発作業開始前に開発基準の中に入れていただくことで、多くの障害を防ぐことができ、安定したシステム作りにつながるはずである。

６．作業工程の工夫は効率化につながったか

６．１　本プロジェクトの作業工程有効性の検証

本プロジェクトは、「**図４．作業工程**」で示す通り、やや変則的な作業工程を取っている。そのため、本プロジェクトで採用した作業工程は効率化、安定したシステム作りにつながったかという有効性を述べておく必要がある。

障害発生原因の統計データから、本プロジェクトの作業工程をきちんと踏んだ業務での障害は少なく、障害が多かった経理業務とワークフローはこの作業工程をきちんと踏んでいないために障害が多かったと判断できる。また、「不注意」によるバグの障害発生が多かった点については、テスト仕様書の作成レベル基準引き上げと標準化により減らすことができると判断したので、工程による問題ではないと考えられる。本プロジェクトで行った作業工程は、効率化と安定の観点を両方とも備えた作業工程であり、プログラミングやテストをそれぞれ個別に別会社に外注するようなプロジェクトでなければ非常に効果的な作業工程だと考える。

７．終わりに

５章で提言した７つの事項については、一般的な感覚として必要だと認識されている内容であり、また斬新なアイデアでなければ難しい内容でもない。しかし、開発作業において削られたりおろそかにされたりしていることが多く、そのため失敗が後を絶たないのである。つまり、それが失敗につながりやすい部分であることが分かっている以上確実に守る必要がある内容であるといえる。本論文は、５章でも記述した通り多少データの不足や偏りがある統計を元に記述している。しかし、実際の障害に基づいて対策を記述しているため、きっと他の開発の中でも活かすことができる活きた論文になっているのではないかと思う。そして、本論文が障害に悩まされているプロジェクトに少しでもヒントを与えるものになればと思っている。

また、今後のシステム開発においても今回の分析を活かし、システム開発時及び開発後には常々問題を振り返り、更に効果的な品質向上策を検討・実行していきたい。

参考文献

［１］畑村洋太郎：“失敗学のすすめ”，講談社文庫