15-B-9

パッケージソフトウェア開発プロセス改善による品質向上と生産性向上 [1]

1. 概要

本事例は、株式会社富士通マーケティング（以降、富士通マーケティング）が開発する統合ビジネスソリューションおよび業務管理ソフトウェアのパッケージソフトウェア開発における、「プロセス改善、品質強化」事例である。

バージョンアップを繰り返すパッケージソフトウェア開発、かつソフトウェアプロダクトの分業開発において、各社間で設計手法・設計書が不統一であり、テスト品質が不安定であった。また、開発規模が大きいにも係らず開発スピードが求められるため、障害が増加する問題があった。さらに、ソースプログラムについては、記述が複雑なため影響範囲が不明であり、重複プログラムに起因する修正漏れやインターフェースの不整合などの問題があった。

これらの課題を解決するため、以下に示す 3 つの対応を実施した。

- 開発プロセスの順序を入れ替えて、テスト設計をプログラミングの前に行うというテスト駆動開発に近い手法を採用し（図 15-B-9-1）、設計手法や設計書を統一した。
- タイムボックスという開発手法を採用し、すべての作業にタイムボックスを設定し繰り返し開発した。タイムボックスの手法は、「時間は動かさない」と宣言することであり開発遅延をなくすやり方である。
- ソースプログラムのリファクタリングで、保守をする上で手に負えないソースプログラムを分かりやすく保守しやすいソースプログラムに洗練させた。

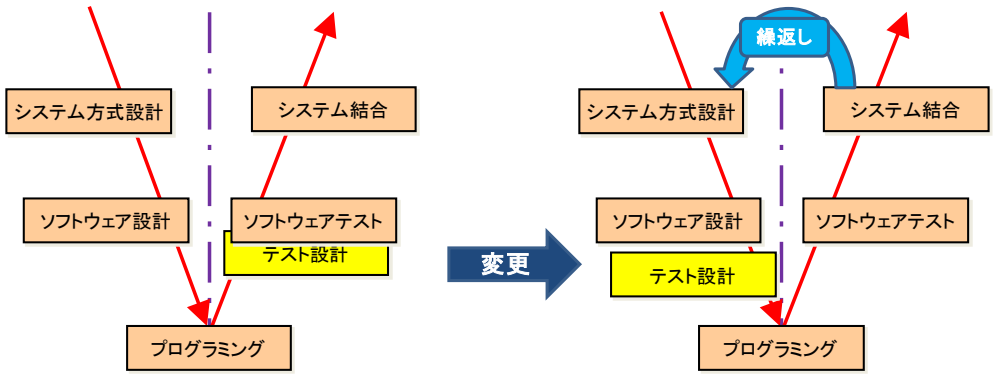

図 15-B-9-1　テスト設計の前倒し

[1] 事例提供: 株式会社富士通マーケティング　GLOVIA 事業本部　松浦 豪一 氏

2. 開発プロセスの課題

富士通マーケティングでのシステム開発は、開発工程により分業されている。要件定義、システム要件定義、システム方式設計、システムテスト、運用テストは富士通マーケティング（設計担当）で担当し、ソフトウェア設計、プログラミング、ソフトウェアテスト、システム結合は子会社または協力会社（開発担当）が担当している。

従来の開発手法では、開発担当社間などで設計手法が統一されておらず、設計書に重複記述があることから保守しにくい設計書になっており､保守コストの増加要因にもなっていた。また、テストにおいて、最小値・最大値・境界値・異常値などに関してテストが不足し品質面で不安があった。

開発担当の設計者は単純にチェック仕様を考える。開発担当のプログラマは単純に仕様を順序どおりに実装する。開発担当のテスト者は組み合わせを洗い出してテストを実行する。この方法でテストを実行すると、異常のパターンが網羅的に作られておらず、そのため障害を検出することができない｡プログラムを作成してから､テストケースの洗い出しをすると、実装方法に依存したものになる。そのため、実装が正しいことは確認されるが、実装が正しくないことを確認するのはむずかしくなり、障害の検出は困難になる。分業化、分断化することによりフィードバックがなくなり、障害検出は後プロセスに回される。このような設計障害は、いくら設計書を書いてもなくならない。

過去一年間の開発データを調査し開発規模で整理すると表 15-B-9-1 のようになる。表 15-B-9-1 から判明したことは、開発規模が大きいにも係らず開発スピードが求められるため障害が増加する問題も多くなっている。これはレビュー不足やテスト不足が要因と考えられる。

表 15-B-9-1　開発規模と障害検出率

開発規模	スピード（KS※／人月）	障害検出率（件／KS）
50KS 未満	1.0～1.5	0
50KS 以上 300KS 未満	2.0～3.0	5
300KS 以上	0.5～1.0	10～20

※ KS … Kilo Step（1,000 行）

また、ソースプログラムについては、機能追加のための改変を行おうとすると、記述が複雑なため、影響範囲が調査しにくいことや、プログラムが重複しているため修正漏れが発生したり、インターフェースが擦り合わせされていなかったりする問題があった。

以上述べたように、従来の開発手法の問題、開発期間の問題、ソースプログラムの問題の各問題が、開発の遅れや障害の増加の原因になっていることが判明した。

上記の問題を解決するため表 15-B-9-2 に示すように問題ごとに対応策を採用することとした。

表 15-B-9-2　問題ごとの対応策

問題	対応策
開発手法の問題	テスト設計方法の見直し
開発期間の問題	タイムボックス手法の導入
ソースプログラムの問題	リファクタリング

3. 実施した技術や手法の詳細

3.1. テスト設計

(1)　要因分析法によるテスト設計

テスト設計とは、ソフトウェアテストのテスト項目の洗い出し方法であり、本事例では要因分析法によるテスト設計を行う（図 15-B-9-2)。ソフトウェアの外的要因を洗い出し、それが取りうる因子を洗い出し、要因と因子の組み合わせパターンからテスト項目を洗い出す方法である。

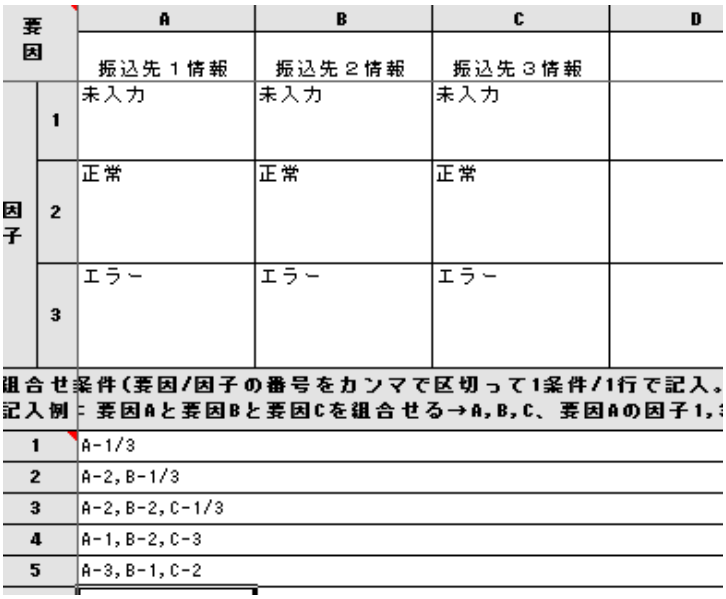

図 15-B-9-2　要因分析法

(2)　段階的試行

テスト設計の手法を定着させるために、最初は設計担当の分担を多くし、開発担当は少ない分担で開始した。外部仕様、システム設計書、システム結合テスト仕様書を渡して、プログラミングからを開発担当に開発させる。慣れてきたら、システム結合テスト仕様書の作成を開発担当に任せた。こういう手順を１か月単位で繰り返し行い、開発担当の担当範囲を増やしていった。また、レビューについては、少なくとも週に

2、3 回程度実施した。レビューをすることにより、文書化されていない情報が共有されるようになった。

(3)　顧客視点での設計

顧客視点で価値がある設計になっているかを常に問いかけていった。具体的には、以下を実施した。

①　設計時の成果物を統一し、併せて製造の前にテスト設計を実施
⇒設計品質を一定レベルに保つことに寄与

②　テスト設計は入出力条件を明確化しながらテスト仕様の作成とテストコードを作成
⇒後工程でのバグ対応時の再検証、リファクタリングなどに再活用

③　組み合わせパターンを考えて設計
⇒テストのしやすさを考慮して作成

④　重複する設計書を削除
⇒価値ある設計書を作成

3.2. タイムボックス

タイムボックスは、タイムボックス内の標準工程と標準時間を設定し、各工程で作業状況をチェックし分析が行えるものである。これにより、無理な作業や作業自体の異常の検知が早期にできるようになる。

設計からテストに至るすべての作業にタイムボックスを設定し 1.5 か月単位とした。精度を高めるため作業単位を小さくし、タスクについては 2 時間以下とした。

また、開発のリズムに合わせてタイムボックスを設定し繰り返し開発を行った。設計 1 週間、製造 1 週間を 3 回繰り返し、2 週間のテストを実施した。この 1.5 か月の開発を何回か繰り返し開発を行った。繰り返し開発時においても、3.1（3）でふれた顧客視点での設計を常に心がけた。タイムボックスの開発スパンを図 15-B-9-3 に示す。

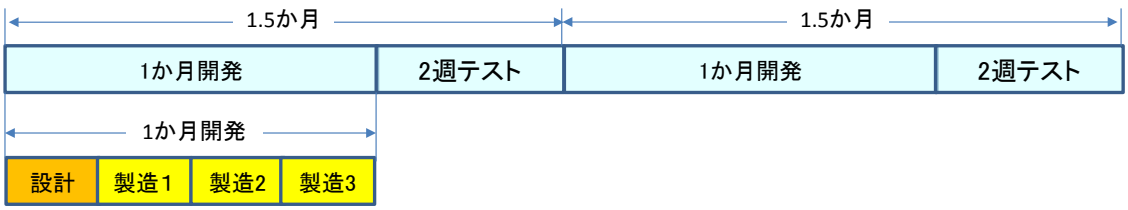

図 15-B-9-3　タイムボックスの開発スパン

3.3. リファクタリング

(1)　リファクタリングの方法

ソースプログラムの構造に依存しないテスト仕様とテストコードを用意した。これは、3.1 (3) ② で作成したものを基本的には使用する。リファクタリングするたびに、このテスト仕様とテストコードを使いリファクタリングの影響範囲を確認していった。

これで IT[2]レベルのリグレッションテスト（回帰テスト）が可能になった。

リファクタリングする順番は優先度の高い機能から実施するようにした。機能追加の高いと思われるものから優先度を高め、目的と範囲を限定してリファクタリングを実行した。

アーキテクチャレベルの改造が必要なものは時間をかけてリファクタリングした。リファクタリング時はソースプログラムだけでなくテストコード（テスト仕様）もリファクタリングの対象とした。

(2)　リファクタリングの例

リファクタリングする前のプログラム、テストパターンがたくさんできてしまう悪いプログラムの例を図 15-B-9-4 に示す。プログラムは洗練されておらず、テストパターンは 12 と多くなる。

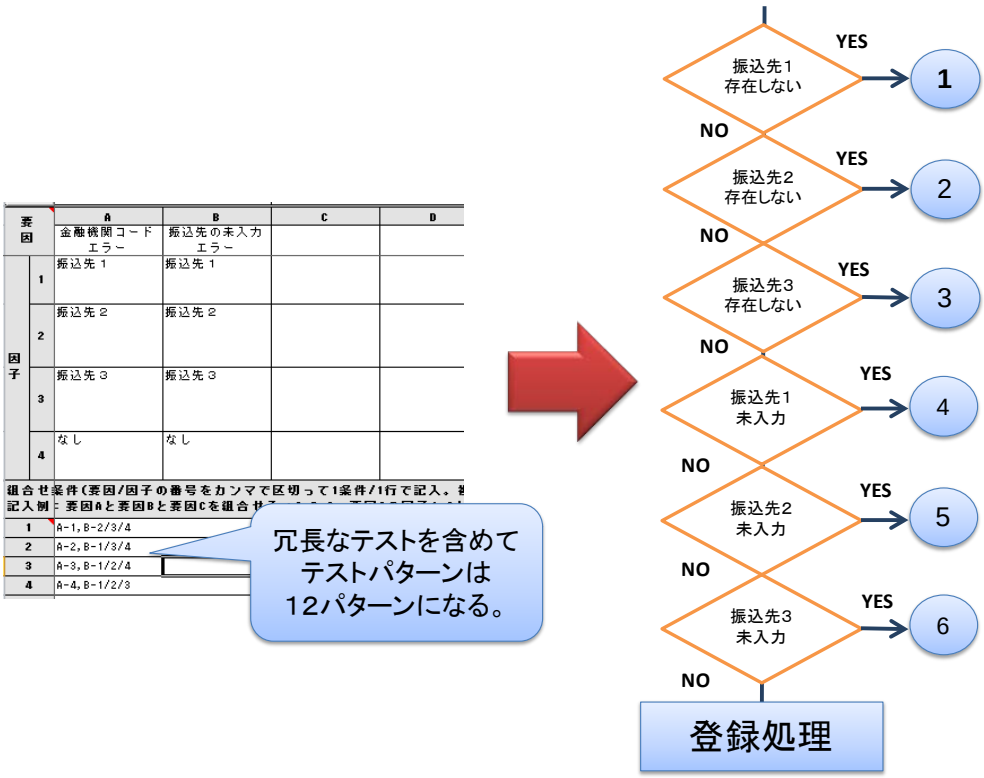

図 15-B-9-4　リファクタリング前

上図のプログラムをリファクタリングした例を図 15-B-9-5 に示す。外的要因を洗い出し、組み合わせパターンを設計した上で、その条件で全てテストできるようにプログラミングする。その結果、データ構造と処理構造が変更になりプログラムがリファクタリングされる。図 15-B-9-4 に比べ、プログラムは洗練され、テストパターンは 12 から 5 パターンに減少しているが必要十分なテストパターンを考慮しているので

[2] Integration Test

品質上問題ない。このようにリファクタリングした結果、プログラムは分かりやすくなり、保守しやすくなった。このことは、今後の派生開発などでも品質を十分に担保できるという効果に繋がる。

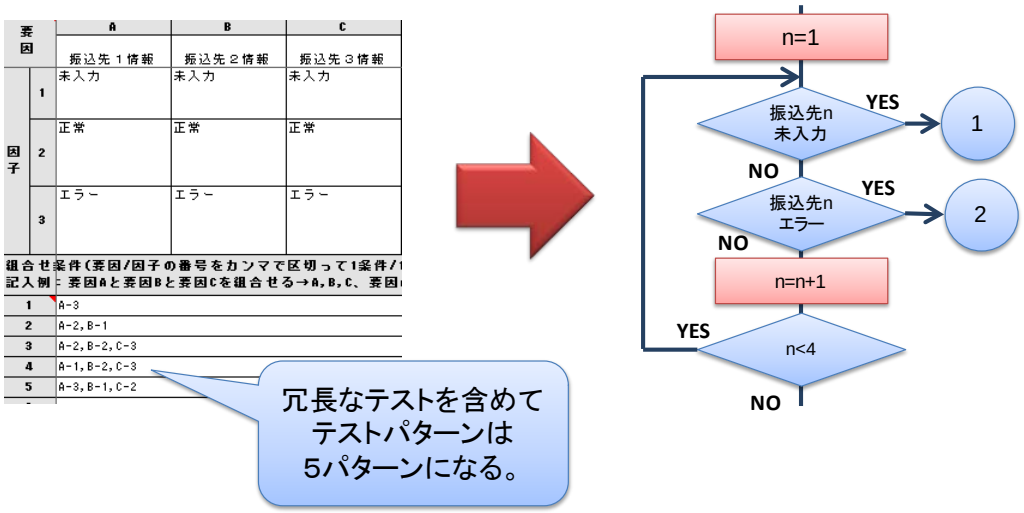

図 15-B-9-5　リファクタリング後

(3)　リファクタリングの実施トリガ

リファクタリングの実施トリガは一般的に、テストコードやソースコードの構造が悪いと判断したときに行うが、本事例では、機能追加時とアーキテクチャの変更時に実施している。

1)　機能追加時のリファクタリング（機能追加をしやすくするために修正する）

以下の手順でリファクタリングする。

1-1）従来機能を保証するテストを用意する。

1-2）ソースプログラムの構造を修正する。

（手続き型の順次処理で書かれていたものをループ処理に変更するなど）

1-3）1-1）で用意したテストを実施して、レベルダウンがないことを確認する。

1-4）機能追加する。

1-5）機能追加分のテストと 1-1）で作ったテストを実施する。

2)　アーキテクチャ変更時のリファクタリング

以下の手順でリファクタリングする。

2-1）従来機能を保証するテストを用意する。

2-2）従来ソースは残し、別ソースで、アーキテクチャの追加をする。

2-3）2-1）で用意したテストを実施して、レベルダウンがないことを確認する。

2-4）機能追加分のテストと 2-1）で作ったテストを実施する。

2-5）繰り返し実施した結果、旧機能と新機能が同等レベルになったら、新機能と旧機能を置き換える。

4. 適用の効果測定方法とその結果

(1) テスト設計

テスト設計手法の採用前後の障害検出率を表 15-B-9-3 に示す。開発中の障害検出率は 4%の 13.0 件／KS にわずかに上昇したが、出荷後の障害検出率は逆に 44%減の 0.17 件／KS に大幅に減少した。障害が前倒し検出されたものと思われる。障害を早期に検出し、出荷後に障害を出さないという観点から、この手法は有効と判断した。

表 15-B-9-3 テスト設計手法の採用前後の障害検出率

	障害検出率（件／KS）	増減率	出荷後障害検出率（件／KS）	増減率
手法採用前	12.8	－	0.30	－
手法採用後	13.0	＋4%	0.17	－44%

(2) タイムボックス

タイムボックス手法の採用前後の障害検出率を表 15-B-9-4 に示す。開発中の障害検出率は 37%減の 8.9 件／KS に減少し、かつ、出荷後の障害検出率は 63%減の 0.11 件／KS に大幅に減少した。作業の定型化や繰り返し開発したため減少したと思われる。タイムボックスの徹底により、設計不足・テスト設計不足などの異常を検知し、再設計や再テスト設計を行うことで品質を高めることができた。この手法も有効と判断した。

表 15-B-9-4 タイムボックス手法の採用前後の障害検出率

	障害検出率（件／KS）	増減率	出荷後障害検出率（件／KS）	増減率
手法採用前	12.8	－	0.30	－
手法採用後	8.9	－37%	0.11	－63%

設計不足・テスト設計不足などの異常検知の例を図 15-B-9-6 に示す。設計を 5 人日でタイムボックスが計画されていたものが 10 人日にかかったとすれば、開発規模の見積りが甘かったのか、設計の仕方が悪かったのか、など、早期に異常が検知でき速やかな対処が実施できることになる。また、設計は計画通り 5 人日で完了したが製造が 5 人日で完了しない場合は、設計品質に問題があったり、製造担当者のスキルが低かったり、など、異常を早期に検知し対応が速やかに実施できることが可能になる。

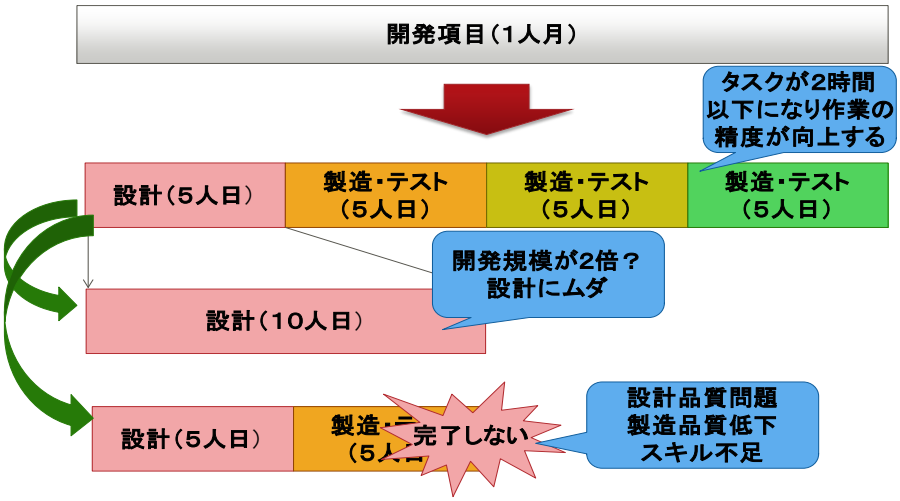

図 15-B-9-6　異常検知の例

(3)　リファクタリング

リファクタリング手法の採用前後の障害検出率を表 15-B-9-5 に示す。過去の通常の開発で作り込んだ障害はリファクタリング時には障害を作り込んでおらず高品質を達成することができるようになった。

表 15-B-9-5　リファクタリング手法の採用前後の障害検出率

	V1 作込障害	V2 作込障害	V3 作込障害
新手法未適用	0.04 件／KS	0.02 件／KS	－
テスト設計	0.00 件／KS	0.00 件／KS	0.03 件／KS
リファクタリング	0.00 件／KS	0.00 件／KS	0.00 件／KS

リファクタリングによる生産性の評価を表 15-B-9-6 に示す。表 15-B-9-6 のスピードの欄のように生産性を規模で表すことが多いが、冗長なプログラミングがなされた場合、生産性が高くなり不合理な指標になる。そこで、価値単位の生産性を表 15-B-9-6 の pt 当たりの工数や pt 当たりの規模で表した。一つの価値を得るための開発規模はリファクタリングしたものが最小で、かつ工数も最小である。これは生産性が高いことを示している。

表 15-B-9-6　リファクタリングによる生産性評価

	スピード	pt[3]当たりの工数	pt 当たりの規模
新手法未適用	1.2 KS／人月	0.29 人月／pt	0.35KS／pt
テスト設計	1.7 KS／人月	0.19 人月／pt	0.32 KS／pt
リファクタリング	1.1 KS／人月	0.19 人月／pt	0.20 KS／pt

[3] pt はポイントで、価値のことであり、機能と言い換えることもできる。1pt は 1 人週程度である。

このように、手法適用により検証時の障害件数が減少した。また、障害が出ると、分析して、その結果をテストパターン、テスト仕様、テストコードに反映させているので、次回の派生開発において開発の効率化と高品質化に大きく寄与している。

5. 現場適用での課題と対策

(1) 「守れるルール」ではなく、「実践によって改善が進む」ルールを作ることが重要である。タイムボックスや要因分析などの実践を通してルール改善を進めてきた。

(2) 新手法の導入時にも教育と実践を繰り返した。開発項目を段階的に開発担当に渡しOJT で教育を実施した。技法や基礎的技術（例：規定値・境界値・例外値の相違）を理解していない開発担当が多いのが現状である。そのような開発担当員がプロジェクトに配属になったときはプロジェクトで教育してきた。配属前の共通教育に技法や基礎的技術の追加も必要になる。

(3) リスクや課題の抽出を繰返し行い、一緒に解決するマネジメントを実施した。進捗は、成果物や発生した問題をみて判断することとした。

(4) 一気に新手法を適用せず、段階的に適用していくことで展開を図った。プロジェクトごとにプロセスは若干の違いがあるため、人の入れ替えなどをしながら改善している。

6. その他の改善策と今後の課題

システム設計を実施するときに、文章で記載する設計書が多くなっている。

このことが、要因分析法での要因の洗い出しやプログラムの実装方法の検討に工数がかかったり、設計と開発の齟齬を生み出しやすくしている。この点を改善するために、要因分析内容をもとに、状態遷移図をあわせて記載するように改善している。このことにより、更なる改善が望める。

また、本取り組みは、新規プロジェクトに水平展開している。

7. まとめ

テスト設計などの開発工程を変更することで、出荷後の障害を減少させ、品質を向上させることができた。また、開発時の障害対応のコストも減少させることができている。

タイムボックスの採用については早期に異常検知できるメリットがある。

リファクタリングを実施することで、作込み障害を減少させソースプログラムの品質が向上した。今後の派生開発においては生産性を向上させることが期待できる。

参考文献

[1] 株式会社富士通マーケティング　松浦 豪一　ソフトウェアテストシンポジウム 2014 テスト設計のタイミングと手法の変更による、品質向上と生産性向上－順序を入れ替え、繰り返し考えるー

[2] 株式会社富士通マーケティング　松浦 豪一　SPI Japan 2013 パッケージ開発プロセス改善による品質向上と生産性向上～品質データからのアジャイルに関する考察～

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（IPA/SEC）