アプリケーション・フレームワークを活用したソフトウェア開発ついて

（株）トウ・ソリューションズ

■ 執筆者Ｐｒｏｆｉｌｅ ■

松田　康

1994 年　(株)中島董商店　入社
2005 年　(株)トウ・ソリューションズ　出向
現在　システムソリューション部所属
ソリューション課　受託開発担当

■ 論文要旨 ■

ユーザー要望やＩＴ環境の変化に対しての機能変更や拡張が繰返されることによりソフトウェアの保守性が低下し、その結果、ソフトウェア改修作業に想定以上の工数がかかったり、また思いもしない箇所で不具合が発生したりする。

このような問題を解決する手段として「アプリケーション・フレームワーク」を採用することを決めた。フレームワークで提供されていない機能については社内開発を行い、また開発技術がフレームワークに特化しないように注意しながら、標準化した開発手順と組合せて生産性と品質及び保守性の向上を目指した。

フレームワーク導入当初、多くの開発者がその手法に慣れていないということもあり困惑気味であったが、経験を積むにしたがいフレームワークの利点を活かせるようになってきた。しかし、その一方でフレームワーク上でしか開発が出来ないという開発者も現れ、総合的なスキルの低下という課題も現れるようになった。

■ 論文目次 ■

１．はじめに ･･《 3》
１．１　当社の概要

２．フレームワーク採用の背景 ･･････････････････････････････････････《 3》
２．１　開発現場の現状
２．２　フレームワーク採用にあたっての有効性と課題

３．フレームワークを利用しての開発事例 ･････････････････････････《 8》
３．１　鶏卵業界向けの物流管理システムの構築
３．２　導入当初の開発の混乱

４．フレームワーク採用の評価 ･･････････････････････････････････････《 11》

５．課題と今後の取り組み ･･《 12》
５．１　課題
５．２　今後の取り組み

■ 図表一覧 ■

図１　フレームワークイメージ････････････････････････････････････《 4》
図２　Sturts　MVCデザインイメージ･･････････････････････････････････《 4》
図３　Struts ＋ 自社開発フレームワークのイメージ･････････････････････《 5》
図４　業務コンポーネント構成････････････････････････････････････《 6》
図５　SQLフレームワークのシステムイメージ･･････････････････････････《 7》
図６　鶏卵業界向け物流管理システム　概略図･･････････････････････《 8》

表１　変更内容一覧･･･《 11》

１．はじめに

１．１　当社の概要

従来、キユーピー株式会社情報企画部がグループ企業の情報企画並びに活用推進に取り組み、株式会社中島董商店ＩＴセンターがシステム設計、開発、運用を担当してきた。ただ、これからの情報部門には、情報企画からシステム構築、運用、活用支援までの一貫した、且つ生産、販売、管理、物流を網羅した機能が求められると判断し、キユーピー株式会社情報企画部と株式会社中島董商店ＩＴセンターの機能を有機的に統合すると同時に、ケイ・システム株式会社（グループのシェアードセンター）のグループコードセンターを併せて、情報新会社「株式会社トウ・ソリューションズ」として、平成17年４月に設立された。

また、グループで培ったノウハウ、資産を活かして、ＩＴの事業化に取り組んでいる。食品業界でのシステム受託開発、パッケージ化を推進することで、業務の効率化と「安全・安心」にシステム面で貢献したいと考えている。

２．フレームワーク採用の背景

２．１　開発現場の現状

ＩＴに依存する傾向がますます高まる昨今において、ユーザ要件もそれに比例して複雑化、多様化してきた。加えて、そのスピードは加速傾向にあり、開発期間の短縮化は免れなくなってきている。既存システムでの機能変更や拡張については、特に厳しい開発期間が要求される。

あるお客様の物流管理システムにおいて、「在庫数量の引当て」機能の引当てタイミングを変更したいという要望があった。このシステムは新規導入してから既に８年が経過しており、この間に何回か機能変更や拡張が行われていたのだが、システムとしては安定稼動していた。今回の変更要望に対して、ユーザの思惑としては「大した修正ではない」との想定であったが、実際に開発の工数を見積もったところ想定以上の工数と工期が必要となった。更に、設計書などのドキュメントはシステムが新規開発された時から更新されていなかったため、ソースコードから仕様を読み解く必要があり、不確定な要素もテスト範囲に含めることでテスト工数も大きくなった。この開発方針については、お客様のご理解を得たことから必要な工数と工期にて開発を進められ、大きな問題もなく無事完了することができた。しかし、こういうケースは稀で、多くは改修作業を短期間で終了させるために、保守性（修正のしやすさに関する性能）を犠牲にした開発が行われる。そして、改修前に比べてソフトウェアの保守性が更に低下し、その結果、次の改修作業では更に想定以上の工数がかかったり、思いもしない箇所で不具合が発生したり負の連鎖に入り込んでしまう。

ソフトウェアの保守性は他の品質特性、例えば機能性や使用性などに比べ、軽視される傾向にある。ユーザにとっては運用しているときの品質が大半を占めていて、保守性の低下が直接的に見えないことが大きな要因だと思われる。しかし、保守性の低下という負の連鎖に一度でも入り込むとなかなか抜け出せなくなる。保守性を低下させないためにも命名規約やコーディング・ルールを整備し、ソースコード検査を強化してきた。しかし、それについても徹底されることはなく、システム改修作業の緊急性からルールが無視され、

そのことも黙認されてきた。

　そんな状況下で採用を決めたのがアプリケーション・フレームワークである。

　再利用可能な汎用的な機能が部品として提供されていて、具体的なロジックを然るべき箇所にプログラミングすることができる。このことから極端に逸脱したソースコードがなくなり、ある程度ルール化されたソースコードによって開発されるため保守性が向上し、再利用を目的とした部品化によって生産性の向上が見込めた。

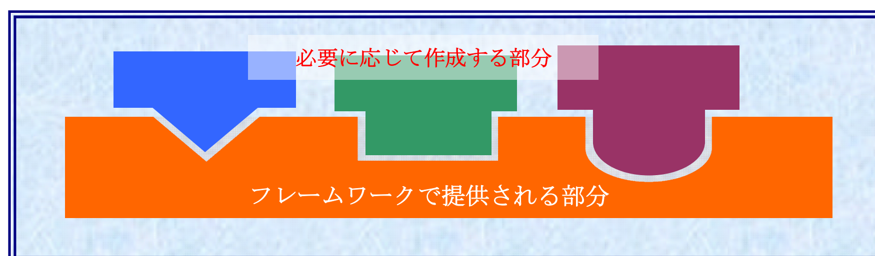

図１　フレームワークイメージ

２．２　フレームワーク採用にあたっての有効性と課題

　アプリケーション・フレームワークとしての実績と Java ベースの Web アプリケーションという点からフレームワークには Jakarta プロジェクトのオープンソース「Struts」を採用する方向で検証を始めた。

　「Struts」はＭＶＣ（Model-View-Controller）デザインで設計されていて、View 部分では画面表示を簡素化するカスタムタグライブラリが、Controller 部分では設定ファイルを基本とした入出力データの取扱や画面遷移、例外処理といった機能が用意されている。また、Model 部分に「アクション」クラスという処理内容をプログラミングできるようになっている。

図２　Struts　MVC デザインイメージ

　「Struts」について、採用にあたって問題がないかどうかを確認するために、３人のメンバーがサンプル・アプリケーションを作成して検証を行った。こうして具体的にプログ

ラミングすることで前述の「アクション」クラスに以下の４点で当社にとって不都合な問題があると判断した。

① ビジネス・ロジックと制御や表示ロジックとの混在
例えば、ビジネス・ロジックと画面制御のためのフラグ項目記述が同じ箇所に記述するなど、ＭＶＣデザインが徹底されないプログラミングが可能であった。

② データベース接続とコネクションの扱い
データベース接続が各々の「アクション」クラスでプログラミングされていたためイベントのたびに接続処理が発生して余計な接続が増えていった。

③ トランザクション処理の記述方法とその箇所
例外時のトランザクション処理で、データベース更新をロールバックする記述の方法やプログラミング箇所が「アクション」クラスごとにバラバラであった。

④ SQL 文の記述方法やその箇所
SQL 文が各々の「アクション」クラスに記述され、同じ SQL 文であってもイベントごとに SQL 文が記述されていた。

これらの問題を解決するために、EJB(Enterprise Java Bean)も検討にあがったが、ベンダーごとに設定が異なっているということと、手軽さを求めるということで、EJB の採用は見送られ、社内で自社開発することとなった。

自社開発する範囲としては、前述の「アクション」クラスから呼出される Model 部分（SQL でのデータベースアクセスを含む業務コンポーネント）とし、「アクション」クラスから利用できる構成とした。

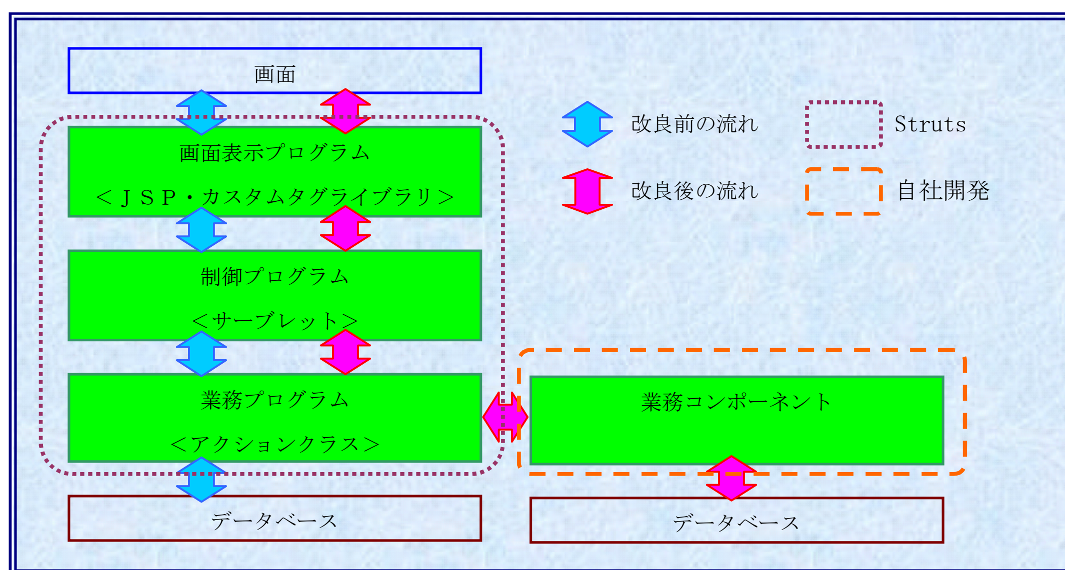

図３　Struts ＋ 自社開発フレームワークのイメージ

また、前述の問題については具体的には以下のような対応を行った。

（１） ビジネス・ロジックと制御や表示ロジックとの混在について
ビジネス・ロジックをその業務機能別にまとめ、業務コンポーネントとして扱え

るようなクラス構成とした。それによって、コンポーネントとしての再利用性が高まり、会社ごとに異なるビジネス・ロジックを取り外し部品として、それごとに扱うことが可能となる。コンポーネントの生成は設定ファイルで行えるようにして、そこのクラス記述を変更することでビジネス・ロジックの取替えが可能となるようにした。

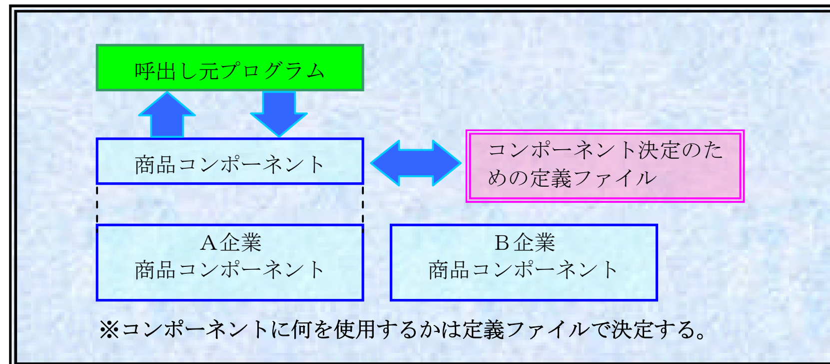

図４　業務コンポーネント構成

（２）　データベース接続とコネクションの扱い

データベースへのコネクションを自動的に取得できる構成とした。接続文字列などの管理は設定ファイルに記述し、プログラミングする際にはデータベースへのコネクションを意識しないでも自動的に取得可能となった。また、コネクションの再利用について、コネクション・プールを備えることで無駄なコネクションを作成することを禁じた。コネクション・プールを標準装備しているアプリケーション・サーバもあるが、Model 部分のフレームワークに組込む必要があったために自社開発を行った。

（３）　トランザクション処理のタイミング

トランザクション処理の開始と終了時にデータベースへのコミットやロールバックをプログラミングするが、その際に発生する不具合を防ぐためにトランザクションを意識させない構成とした。例外やエラーが発生した際には自動的にロールバックを行い、正常に終了した場合にはコミットされる。この機能からトランザクションを意識しなくてもよいプログラミングを可能とした。

（４）　SQL 文の記述方法や記述場所

「アクション」クラス内にソースコードと SQL 文が混在しないように、SQL 文を外部ファイルにする構成を取った。これは Java ソースコード内に SQL 文を埋め込んでしまうことで可読性の低下につながると判断したためで、SQL 文のみ記述したファイルを用意することで可読性の向上を目指した。更に、機能ごとに複数の SQL 文を１ファイルにまとめることで、どこに何の SQL 文が記述されているか把握しやすくした。その結果、データのレイアウトに変更があった場合でも関連の SQL ファイルを確認するだけでよくなる。また、SQL ファイルから雛形 SQL 文を基に SQL 文を動的に組立てられるようにした。それまでは条件文や更新する値

に対して、プログラムの各々の箇所で SQL 組立てロジックをプログラミングしていたが、SQL 文記述に一定のルールを採用し、それに基づいて SQL を組立てが可能となった。それによって、SQL 文組立て時の不具合や複雑さから開放されることになった。更にデータベースから取得した検索結果をデータオブジェクトにセットする機能も追加した。これによって、今まで何度も繰返し同じようなプログラミングが行われていたが、XML の定義ファイルを利用して検索結果のカラムとデータオブジェクトの属性名とでマッピングすることから、生産性向上と可読性向上を実現した。

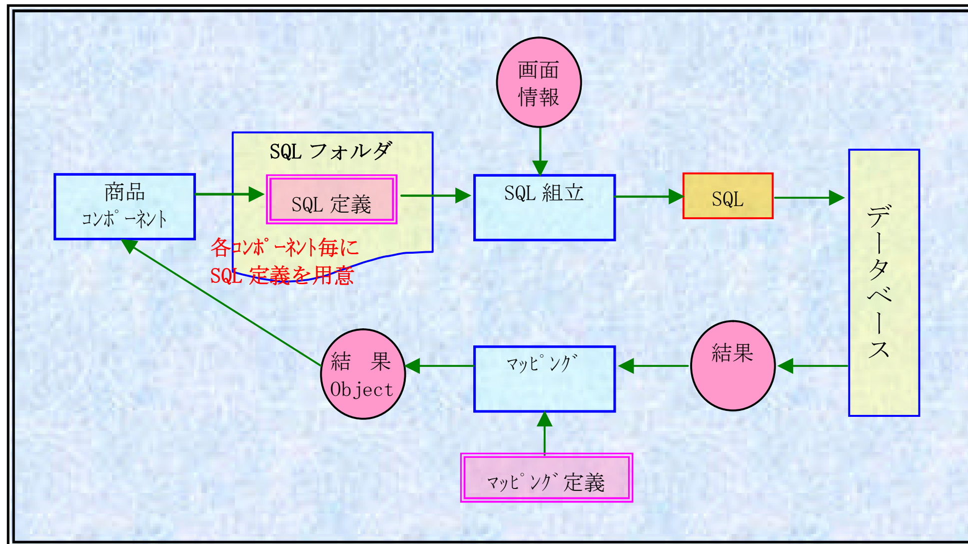

図５　SQL フレームワークのシステムイメージ

３．フレームワークを利用しての開発事例

３．１　鶏卵業界向けの物流管理システムの構築

「Struts」と Moedl 部分の自社開発フレームワークを利用して、本格的な開発に着手したのが「鶏卵企業向けの物流管理システム」である。このシステムは相場制の取り入れや取引先別単価計算式の設定などの鶏卵業界の慣習に対応しており、受注・仕入・生産・出荷の各業務における作業の手間やミスを軽減し、業務の効率化を支援することを目的としている。

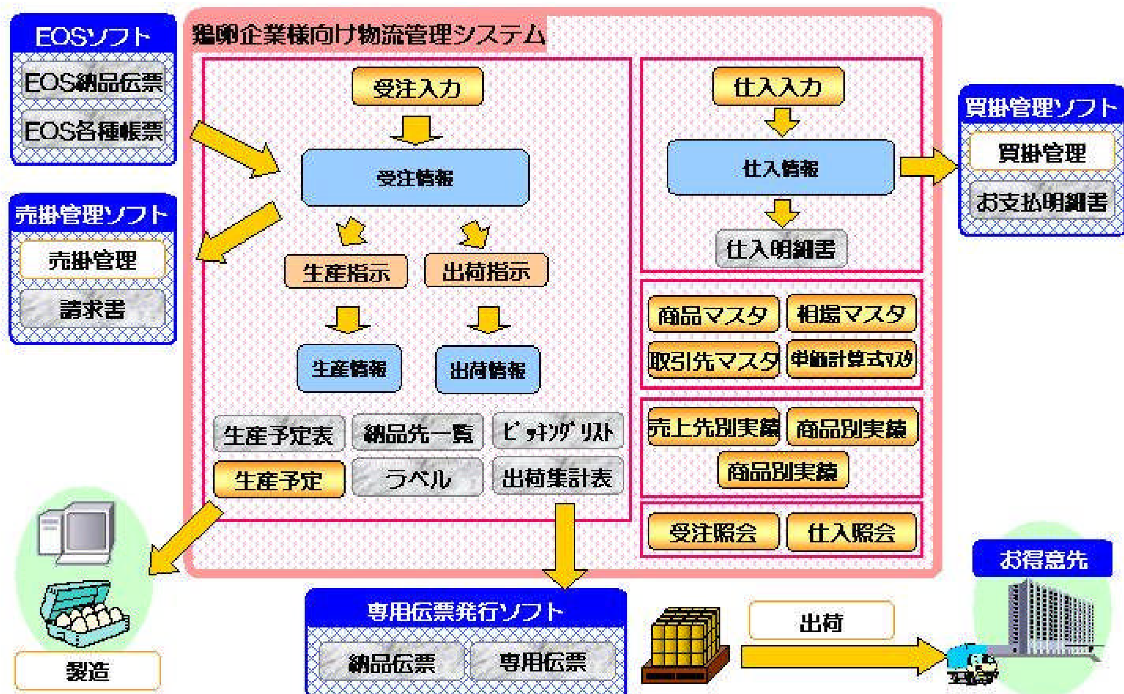

図６　鶏卵業界向け物流管理システム　概略図

当システムの構築にあたっては、他の鶏卵製造販売会社にも適用できるように、ビジネス・ロジックの変更や追加が想定される箇所については業務コンポーネントを作成して、変更や追加拡張が容易となるように設計を行った。例えば、鶏卵業界特有の機能として、以下のようなものがあるが、これらの仕様は取替え可能なコンポーネントとして設計を行った。

（１）　単価決定機能

鶏卵相場に対して各社独自の単価決定のロジックをもつ。

（２）　受注調整機能

鶏卵農場から仕入れる鶏卵のサイズや数量は日々異なるが、得意先に安定した供

給を行うために受注数の割当てを調整する必要がある。

（３） ピッキング機能

出荷する際に各配送コースごとにピッキング作業を行うが、コースごとに最適な荷を積載させるため、ピッキングリストには、タマゴ特有の什器（箱・ロール・パンコンなど）を効率よく行えるように調整する必要がある。

（４） 原卵仕入機能

原卵の仕入れについては、卵のサイズごとに仕入単価が設定されているが、サイズが混在したまま一括で行う。サイズの比率を予め決定しておき、合計数量からその比率で各々のサイズの数量を算出する。

このように、雛形システムとしての保守性を向上させ、短納期で機能を変更・追加が実現できるようにした。会社特有の業務ロジックを部品として取替可能とした。

３．２　導入当初の開発の混乱

フレームワークを理解している開発者が設計しているということもあって、設計工程までは大きな問題もなく順調に進んでいた。しかし、プログラミング工程に入って、進捗が思うように進まなくなってきた。開発者がフレームワークに戸惑い、当初予定していた生産性を保てなくなってきたのである。

原因を調査した結果、下記の問題が明確になった。

（1） 定義ファイルが増加

画面遷移や例外処理、SQL 文の組立て、OR マッピング、メッセージ内容、画面表示文字など、各々の定義ファイルに記述することで機能を実現しているが、定義の記述ミスという単純なミスが開発者を悩ませていた。定義の記述ミスに気づくのに時間を費やしていたのだ。慣れていないということもあったのだが、記述ミスで出力されるエラーの表示内容が貧弱で直接不具合箇所に結びつかなかった。そのため、1 時間以上調査して記述ミスに気づくケースも珍しくはなかった。また、これまではプログラムをコンパイルする時にエラーが出力され確認できていたことも、プログラムを実行する時までエラーに気付かないという状況も生産性を悪化させる一因となっていた。対策として、定義ファイルの記述エラーの出力メッセージをより具体的にして分かりやすくした。それによって、定義ミスに気づきやすくなり、このことで時間を多大に費やすことが少なくなった。

（2） ＭＶＣデザインの不慣れ

ＭＶＣデザインを用いてのプログラミングを徹底するためのルールを採用した。しかし、ルールの啓蒙や認識が完全に行われなかったため、完成したプログラムを修正することもしばしばあった。例えば、売上金額「1234500」について、仕様で「千円単位で表示する。小数点以下は四捨五入。」という場合には、ＭＶＣの View に当たる部分をプログラミングすべきだが、SQL 文で 1000 で割って、

SQL 関数を使って四捨五入するようなプログラムになっていた。表示(View)に関することは表示(View)に任せるという考え方から外れていたため、修正を行わなければならなかった。ＭＶＣデザインを理解できていれば防げると判断したため、ＭＶＣデザインの教育とルールの啓蒙を行った。その結果、こういったケースで修正を必要とすることがほとんどなくなった。

（３） マニュアルの不備

自社開発フレームワークのマニュアルを作成したが、開発者にとって有効なマニュアルにはなっていなかった。標準的な手順についてはマニュアルに記載されていたが、例外的な問題となっている事項が記載されていなかった。よって、ＦＡＱや事例別の開発手順を記載して実践的なマニュアルを作成した。それでも補えない事項については、フレームワークを熟知している開発者が他の開発者をサポートすることで開発の生産性低下を抑えた。

（４） リファクタリング

保守性に対して高いレベルを目指していたこともあり、コード・インスペクションやリファクタリングにも時間を費やした。コード・インスペクションについては検査ツールを利用して、指摘された不都合なソースコードに対して修正を行った。リファクタリングについてはプログラミング上級者のソースコードチェックやペアプログラミングにおいて、細部までリファクタリングを行った。コード・インスペクションやリファクタリングは作業範囲として計画していたが、慣れないこともあり、計画していたよりも１．５倍から２倍の工数を費やした。

以上のことから、「慣れていない」ことが大きな要因ではあるが、保守性を向上させるためにはプログラム工程での生産性が低下する可能性があることが分かった。しかし、最初の開発において保守性が高く維持できれば、この後の修正や追加開発のときに修正や追加作業が容易となり、生産性の向上につながると信じてプログラミング工程を進めていった。そして、プログラム工程の終盤では当初ほどの混乱もなくなり、全てのテストに合格し、システム導入も無事終えた。「鶏卵企業向け物流管理システム」として、第２、第３の鶏卵企業に水平展開できるようなレベルで保守性を維持できていると思われる。その評価は、次に当システムを適用する際にプロジェクトによって計られる。

４． フレームワーク採用の評価

「鶏卵業界向けの物流管理システム」の雛形システムに２社目の適用を行い、フレームワークを使用したソフトウェアの保守性について評価を行った。１社目とのギャップは４１箇所であった。代表的な修正パターンにおいて保守性を検証した。４１箇所の内訳は以下のとおりである。

	修正内容の分類	修正個数	実績工数（人日）	対応
1	画面･帳票のレイアウト、文字列の変更	8	1.1	レイアウトファイルの修正や文字列定義ファイルの変更で対応
2	ビジネス・ロジックの変更	12	4.6	業務コンポーネントの置き換えで対応
3	マスタ属性の追加・変更	8	6.3	主に SQL ファイルの変更で対応
4	その他	13	9.5	個々に適した方法で対応

表１　変更内容一覧

（１）　画面・帳票のレイアウト、文字列の変更

レイアウトファイル(JSP)の修正や文字列の定義ファイルの変更で対応可能である。同一文字列であれば一箇所の変更で一括した修正が可能となり修正漏れなどは無かった。

（２）　ビジネス・ロジックの変更

業務コンポーネントの置き換えで対応。例えば、単価計算ロジックの場合、単価計算コンポーネントを新規に作成して、それを呼出し定義に加えた。修正であるが既存プログラムに手を加える必要がなく、クラスの新規追加によって対応が可能であった。既存のプログラムへの修正ではなかったので、テスト範囲も限定することができた。

（３）　マスタ属性の追加・変更

マスタに関連するコンポーネントの SQL ファイルを変更することで対応ができた。マスタへの追加・参照・更新・削除処理の記述が一箇所で限定的であったため、マスタ属性の追加や変更も当該コンポーネント内での修正で対応できた。これについてもテスト範囲を絞ることができた。

（４）　その他

変更・追加を想定していない箇所では、個々に対応を行った。しかし、フレームワークとＭＶＣデザインのルールに則っていることから、修正箇所の調査には時間を取られなかった。その分、保守性を低下させないプログラミングに注意を払った。

「鶏卵業界向けの物流管理システム」をはじめ、現在まで複数のシステム構築でフレームワークを利用した開発を行ってきた。これらすべての開発において、プログラミングの

構成が同じである。画面の作り、ログの出力、ビジネス・ロジック、例外表示、入力チェックなど、同じ構成を取っている。よって、どこにどういうロジックがプログラミングされているかが分かりやすい。例えば、在庫の引当方法について、在庫コンポーネント内の引当機能でプログラミングされていて、その箇所を見つけるのに時間がかからない。修正範囲も限定的である。

フレームワーク開発とＭＶＣデザインに則ったプログラミングにおいて、開発メンバーが同じ目線で開発が可能となり、保守性が向上しているといえる。また、修正箇所の特定についても、迷うことが少なく確実性を持って行うことができている。それにより余計なリスクを組み込む必要もなく、工数見積もりに大きな隔たりが出ることがなくなった。これは開発側にとっても、またユーザ側にとっても喜ばしいことである。

５．課題と今後の取り組み

５．１　課題

フレームワークを利用した開発が定着して、ソフトウェアの保守性は向上したといえる。ルールに従わざるを得ない開発状況において、開発者は記述すべき箇所に画面のレイアウトやデータベースからのデータ取得、ビジネスロジックを記述すればよく、そこにだけ注力すればよくなった。煩わしいログ出力やトランザクション管理、データベース・アクセス、例外処理などのシステム機能の実装から解放され、頻繁に使用される機能については部品化されて、使い勝手がよいものとして提供される。しかし、こういう開発環境において、喜ばしいことばかりではない。

開発者のスキルの低下が目に付くようになった。というのも、本来は意識すべき前述のようなシステム機能要件を意識しなくてもよくなった。その結果、フレームワークを利用した開発では力を発揮できるのだが、フレームワークを利用しない開発では力を十分に発揮できない。

共通部品や自社開発フレームワークを作成してきたベテラン開発者とそれを使用する開発者の間でその差が大きくなっている。「トランザクションとは？」「データベースにアクセスするには？」といった基本的な知識が欠落してしまっている。

これらの教育を行うことで知識としては身に付くのだが、経験がないので応用力が生まれてこない。問題があった場合に、解決の糸口をつかめない。これらの問題に対しては、経験を積ませるべく、フレームワークに関する開発を経験できるようにして、スキルアップを進めている。今後、人を育てながら、フレームワークも育てていく必要がある。

５．２　今後の取り組み

今後の取り組みとして、シン・クライアントをリッチ・クライアントに移行することを検討中である。リッチ・クライアントに何を採用するかは決まっていないが、選択するリッチ・クライアントによってはフレームワークもそれに合わせて成長させる必要がある。自社開発を行った Model 部分のフレームワークについては、クライアント側が変わったとしても引き続き利用可能となる。こういった点においてもＭＶＣデザインを採用していて正解だったと考える。