ソフトウェアプロダクトラインのエンタプライズ・システムへの適用と評価



中村　伸裕[†1†3]　谷本　收[†2]　楠本　真二[†1]

ソフトウェアプロダクトラインは，ソフトウェアを共通性と可変性が事前に整理された再利用資産から開発する手法で，派生ソフトウェアを効率よく開発できる．組込ソフトウェア開発での事例は多いが，エンタプライズ・システム開発には適用が難しいことが指摘されている．本論文ではエンタプライズ・システムへのソフトウェアプロダクトラインの適用の試みについて述べる．ソフトウェア資産としてビジネスロジックではなく画面部品を中心に構築することにより各アプリケーションで開発するソースコード量を削減し，プログラム開発のコスト削減を実現した．また，操作性の高い部品を提供することで利用者の満足度を高めるとともに開発者に部品利用の動機付けを行った．その結果，ソフトウェア資産は 10 年以上の期間にわたり全開発プロジェクトで再利用されており，組織全体で開発したソースコード量は大幅に減少し，開発コストも低減することができた．

Application and evaluation of Software Product Line for enterprise systems

Nobuhiro NAKAMURA[†1†3], Osamu TANIMOTO[†2], and Shinji KUSUMOTO[†1]

Abstract

Software Product Line (SPL) method is one of the reuse technologies that software is developed by adding specific features to a common set of core assets in a prescribed way. Though there are many case studies that applied SPL to embedded software developments, it has been pointed out the difficulties of applying it to enterprise software developments. This paper describes a successful study introducing SPL to enterprise software developments. We constructed mainly GUI components, instead of business logic components, as software assets. By using the components, we have reduced the amount of source code newly developed and attained development cost reduction. We also motivated developers to use the components by providing high operable components for users. As the results of introducing SPL, the software assets have been using for over a decade, and we could greatly reduce the amount of source code size and the development cost.

1. はじめに

ソフトウェアの大規模化と複雑化に伴い，高品質なソフトウェアを短期間に効率よく開発することが求められている．再利用はこれを実現する一つの有効な手法であると考えられ多くの研究が行われてきた [1]．再利用の形態はサブルーチン，モジュール，オブジェクト，コンポーネントと

【脚注】
† 1　大阪大学　Osaka University
† 2　住友電工情報システム株式会社
Sumitomo Electric Information Systems Co., Ltd.
† 3　住友電気工業株式会社
Sumitomo Electric Industries, Ltd.

進化し，2000 年代にはソフトウェアプロダクトライン (以下，SPL)[2][3] の考え方が広まっている．

SPL の 2 つの特徴は (a) ソフトウェア資産を構築するドメイン開発とソフトウェア資産を活用するアプリケーション開発の 2 つのプロセスを分離する，(b) ソフトウェア資産のどの部分が共通でどの部分が可変なのかを明示的に示すことである．SPL は従来のアドホックな再利用ではなく計画的な再利用を実現することが目的となっている．SPL は組込み系の分野では多くの事例が報告 [4][5] されているが，エンタプライズ・システムの事例報告は少ない [6]．

住友電気工業の情報システム部門 (以降，当組織) は社内で利用する生産管理，在庫管理，販売管理，購買管理，物流管理，会計，人事などの事務処理システムの開発を主な業務としており，継続的に品質，コスト，納期の改善に取り組んでいる．本論文では 1999 年から開始した，SPL に基づくソフトウェア部品の再利用により社内の業務システム開発で作成するソースコード量を削減することで開発コストを低減させる取り組みと，2003 年から 2012 年までの 10 年間の実績評価について報告する．ソフトウェア部品へは業務ロジックを対象とするアプローチと画面部品を中心とするアプローチが考えられたが，これまでの経験から後者の方が開発するソースコード量の削減に効果があると考えた．業務システムで利用する画面はそれぞれ表示するデータ項目が異なり，データ長や入力方法が異なるため簡単に部品化することができない．この問題を解決するためにデータ項目をオブジェクト化する項目オブジェクトの考案し画面部品を開発した．また，操作性の高い画面部品を提供することで利用者の満足度を高め，開発者の再利用に対する心理的な抵抗を解消した．さらに演習を中心とした 3 日間のトレーニング・コースを定期開催することで全社展開することができた．これらの取り組みの結果，IPA/SEC が発行している文献 [7] に掲載されている生産性と比較して，3 ～ 5 倍の高い生産性と利用者の高い満足度が実現できている．

2. ソフトウェアプロダクトライン (SPL)

ソフトウェアの再利用は品質，コスト，納期の改善策として期待されてきた．初期段階ではサブルーチンが作成され，モジュール，オブジェクト，コンポーネントと進化し [8]，2000 年頃から計画的な再利用を目的として SPL の考え方が広がっている．従来の製品ごとの開発プロセスと SPL の開発プロセスとの違いは，ソフトウェア資産を開発するドメイン開発とソフトウェア資産を活用して個々のアプリケーション（システム）を開発するアプリケーション開発（以下，ＡＰ開発) の 2 つのプロセスが体系化されていることである．ドメイン開発の主要プロセスは，(1a) ドメイン要求開発，(1b) ドメイン設計，(1c) ドメイン実現，(1d) ドメイン試験であり，これらのプロセスから複数アプリケーションの共通的な要求，アーキテクチャ，コンポーネント，試験に関する成果物といったソフトウェア資産が構築される．ＡＰ開発は，(2a) アプリケーション要求開発，(2b) アプリケーション設計，(2c) アプリケーション実現，(2d) アプリケーション試験のプロセスで構成され，各プロセスがソフトウェア資産を活用する．また，上記 8 プロセスがうまく計画管理できるよう，個々のアプリケーションのリリース計画などを管理する製品管理プロセスがあり，合計 9 プロセスで構成されている．各プロセスの内容は 6 章，7 章で実施内容と共に説明する．なお，本稿では業務システム開発がＡＰ開発に相当する．

3. SPL の導入目的

1999 年にオブジェクト指向言語 Java をサーバーサイドで利用する技術が登場したことをことによりソフトウェア部品の開発環境が容易に入手できるようになった．当組織では以前から開発コスト削減の手段としてオブジェクト指向言語によるソフトウェアの再利用を検討しており，1999 年にソフトウェア部品の開発を進めることになった．当時は個々の開発チームで共通部品を開発し再利用を進めていたが局所的な再利用のため大きな成果は得られていなかった．ソフトウェア資産を構築し再利用によるコスト削減効果を得るためには各開発プロジェクトが開発する成果物量を削減し，開発工数を削減する必要がある．また，ソフトウェア資産の開発投資に対する効果を最大化するためには開発したソフトウェア資産は全プロジェクトに展開し，長期間利用する必要がある．このような全社的な再利用を進めるためには SPL で示されているようにドメイン開発チームとＡＰ開発チームを分離し，ソフトウェア資産の開発にも要求定義から試験のプロセスを実施する必要があると考えた．

SPL 導入の目的は，(a) ＡＰ開発プロジェクトが開発する成果物量を削減できるソフトウェア資産を構築，(b) ソフトウェア資産を長期間，全社展開することでコスト削減効果を最大化することである．

4. SPL 導入の課題

4.1. エンタプライズ・システムへの SPL 適用の課題

文献 [6],[9] ではエンタプライズ・システム開発における SPL 適用の課題が述べられている．ここでは当組織の状況を説明する．

(1-1) 移り変わる実装技術と非機能要件の高度化

ソフトウェア資産を構築する上で資産が永続的に利用できることは重要な要素である．当組織では OS，ミドルウェアに OSS を積極的に採用することで，ベンダーの事業戦略（事業撤退を含む）の影響を受けにくい環境を整えている．また，開発言語は複数のベンダーが提供している Java を採用しており，長期間の利用が期待できる状態となっている．また，操作性やリアルタイム性，24 時間稼働などの非機能要件は製造業ということもあり，大きな問題となっていない．全社員が利用する勤務管理システムでは利用者が数千名，同時利用は数百名の規模であり，事業部ごとに開発するシステムでは同時利用者は 100 名以下であることが多い．

従って，一般の PC サーバーの能力で処理可能な負荷である．

(1-2) RDBMS への依存度

一般に，SQL による RDBMS の処理がボトルネックになることが指摘されている．当組織では同時利用者数が少ないことに加え，サーバー能力が不足気味であった 1990 年代から，正規化されたテーブル構造を実装した上で十分な応答速度を確保するチューニング技術を蓄積しているため，SQL をそのまま利用しても問題が発生していない．

(1-3) 個別案件主体とレガシーシステムの存在

案件単位で利用する技術やコスト，納期の制約があり，SPL 適用の障害になることが示されている．当組織では新技術を評価し，社内展開する部署が設置されているため，案件単位で個別に技術を選定することがなく，SPL を導入しやすい環境となっている．

(1-4) 工数削減できるソフトウェア資産の構築

文献 [9] の事例では SPL の取り組みが初期段階で充分な結果が得られていない．よりコスト削減効果があるソフトウェア資産の構築が求められている．

4.2. SPL 導入の組込み系との共通課題

文献 [10] では組込み系，エンタプライズ系共通の課題として以下のものが示されている．

(2-1) 品質管理のためのソフトウェア構成管理

開発したソフトウェア資産はＡＰ開発で利用されたあとも機能拡張や欠陥除去が行われ，複数バージョンの資産が開発される．欠陥除去は配付されたすべてのバージョンに対して実施されるべきであるが，構成管理をうまく行わないと局所的にしか欠陥除去が行われない可能性がある．

(2-2) 再利用に対する人間的側面

NIH(Not Invented Here) は自分達が開発したものでないと再利用したがらない傾向を表す言葉であるが，NIH 症候群を考慮したプロジェクト運営が必要である。

5. SPL 推進方針

4. で示した (1-1), (1-2), (1-3) の課題は既に解消していたため，(1-4), (2-1), (2-2) の課題に取り組んだ．

5.1. 開発量が削減できるソフトウェア資産の開発

ソフトウェアの再利用により開発コストを削減するためにはＡＰ開発において開発するソースコード量が削減できるソフトウェア資産が必要となる．文献 [6][9] の例ではユーザーインターフェースではなく，ビジネスロジックに焦点を当てている．一方，我々の Web システム構築の経験ではビジネスロジックよりも画面出力に関するソースコードの方が多いことがわかっている．我々は画面部品を開発することで (1-4) の課題を克服することにした．しかし，ユーザーインターフェースである画面は利用者の要求により多くのバリエーションがあり，部品化が難しい．また，部品化により画面デザインの自由度が低下すると，操作性の悪化により利用者の満足度が低下する恐れもある．画面部品の開発にあたっては操作性のよい画面が作成できる機能を持たせ，部品化により利用者の満足度を向上させることにした．

5.2. ソフトウェア資産の展開

(2-1),(2-2) はソフトウェア資産展開の課題と位置付けた．開発したソフトウェア資産は全社に展開し，長期間利用することでコスト削減効果を最大化したい．また，ソフトウェア資産の欠陥除去などの保守コストも抑制したい．機能拡張により基本アーキテクチャが変更すると欠陥除去のための調査コストやソースコードの修正コストが増加することが予想されるため，ドメイン要求分析を実施しすることで安定したアーキテクチャを構築することにした．その上で (2-1) の品質管理の課題をドメイン開発チーム内の構成管理で解決する体制を検討する．また，(2-2) の人的側面の課題は開発者が積極的に使いたいと思うよう，利用者の満足度を高められる操作性の高い画面部品を提供することで解決することにした．さらにトレーニングなどの支援体制を整備することでソフトウェア資産が容易に利用できる環境を整えることにした．

6. ドメイン開発の実践

6.1. ドメイン要求開発

ドメイン要求開発ではソフトウェア部品を抽出し，安定したアーキテクチャを構築するために想定されているアプリケーションの固定化できる共通点と個別に変更できる可変点を抽出する．

(1) 共通要件の抽出

想定するドメインは企業内の事務処理システムであり，生産管理，在庫管理，販売管理，購買管理，物流管理，会計，人事といった幅広い業務を支援するソフトウェアである．図 1 に事務処理システムの構造を示す．サーバー内にデータベースを構築し，正規化されたテーブル構造が実装されている．端末には Web ブラウザがインストールされており，サーバー内のプログラムにアクセスすることで業務を行う．プログラムは注文登録や出荷指示など，１つの業務に対応したもので複数の画面で構成されている．１つのシステムが対応する業務は 30 ～ 50 種類程度のものが多い．

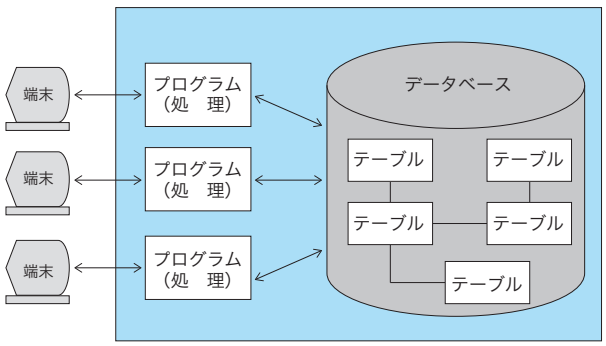

図１　事務処理システムの構造

上記構造を前提とし，以下の共通的な要件が存在している．

(R1) エンドユーザーはブラウザでシステムにアクセスし，業務で利用する画面を表示する．

(R2) エンドユーザーは画面を操作して，データベースにデータを登録，照会，変更，削除する．

(R3) その際，システムは入力された値のエラーチェックを行う．

(R4) システムは必要に応じて，データベースのデータを加工してブラウザに表示する．

(R5) システムは必要に応じて，入力されたデータを加工してデータベースに登録，変更する．

(R6) システムは処理結果を端末に表示する．

(2) 画面の共通点と可変点の抽出

図２，図３に業務で使用する画面の例を示す．図２は商品を登録する画面である．図３は商品の注文を登録する画面である．この２つの画面の共通点は，(a) タイトル，(b) メニュー，(c)1 つ以上のデータ入力ブロック (図３では (c-1), (c-2))，(d) 登録ボタンが配置されていることである．可変点は，共通点の中に多く含まれており，以下のものが抽出できる．

- タイトルに表示されている文字が “ 商品登録 ” と “ 注文受付 ” で異なる．
- データ入力項目は図２では１ブロック，図３では２ブロックで構成されている．
- データ入力ブロックに含まれるデータ項目（商品コード，受注番号等）が異なる．

部品化の課題となるのはデータ入力項目であり，可変点は以下の通りである．

- データ項目の名称
- データ入力領域の桁数
- データ入力の方法（TEXT, CHECKBOX 等）

(3) 画面遷移の共通点と可変点の抽出

画面遷移についても再利用を行うため，共通点と可変点を抽出した．基本的な画面遷移を図４に示す．利用者がアプリケーションのメニューからプログラムを選択すると，プログラム内のメニューから (a) 登録入力画面または (d) 検索条件入力画面を表示することができる．その後は矢印で示された流れで画面を進めることができ，登録，照会，変更，削除の業務が行えるようになっている．この画面遷移で可変点は，(h) 変更確認画面，(k) 削除確認画面で，アプリケーション要求に応じて省略することができる．

6.2. ドメイン設計

ドメイン設計ではＡＰ開発で使用するアーキテクチャを設計する．今回のソフトウェア資産は Web システムを前提としており，端末とサーバーとの通信が毎回切断されるという条件の中で可変点に対応できる構造にする必要がある．図５に利用者が画面に入力したデータをデータベース

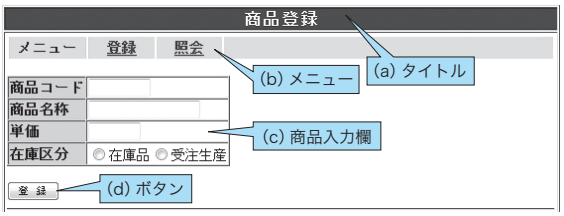

図２．商品登録画面の例

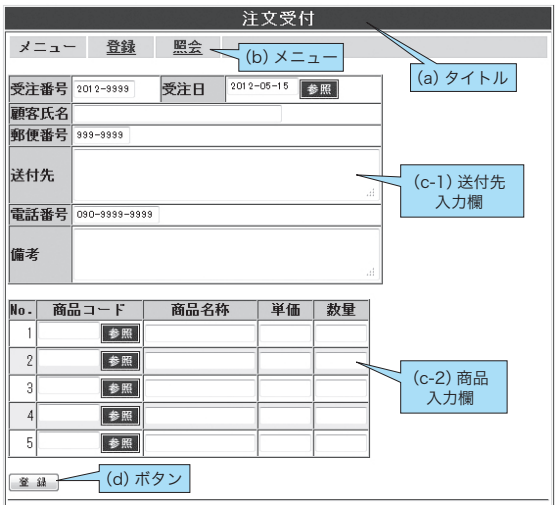

図３．注文受付画面の例

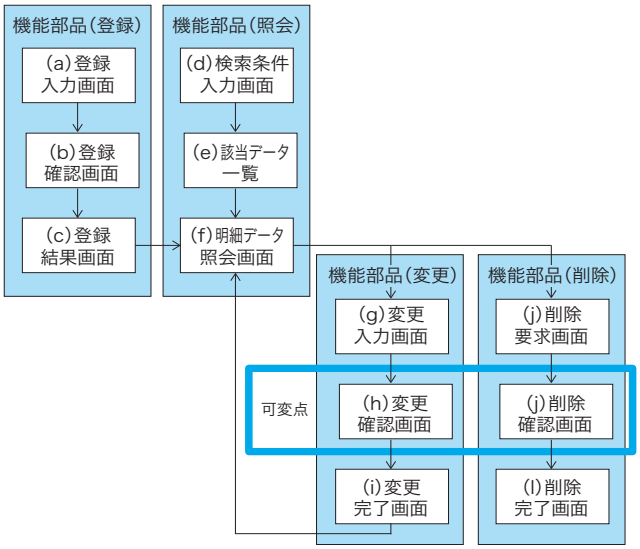

図４．画面遷移の共通化

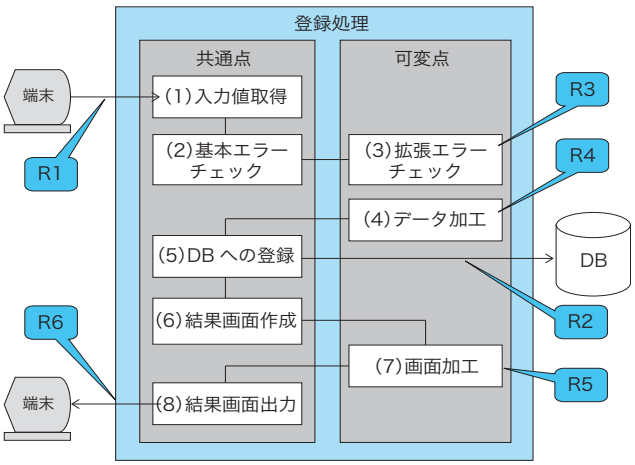

図５．データをデータベースに登録する処理の流れ

```
<form action="…">
<table>
<tr><th> 商品コード </th>
<td><input type="text" name="prod_cd" size="8"></td></tr>
<tr><th> 数量 </th>
<td><input type="text" name="order_qty" size="5"></td></tr>
</table>
<input type="submit" value=" 登 録 ">
</form>
```

図 6．受注品目 , 受注数量入力画面の HTML

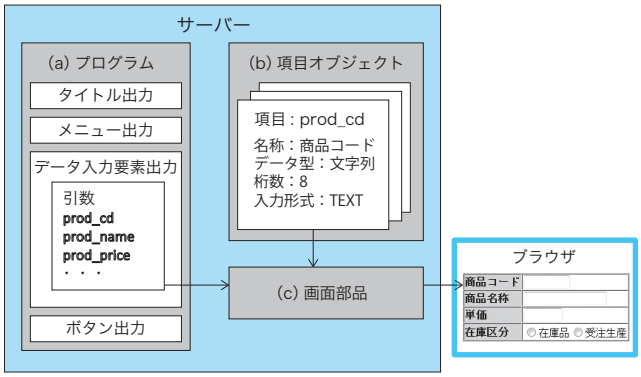

図 7．画面部品の構成

に登録する際の処理の流れを示す．この流れは 6.1 の要件 (R1) から (R6) に対応している．端末の登録ボタンを押すと，サーバー内の登録処理が起動される．まず，共通部の (1) 入力値取得は端末から送信されたデータをデータ項目ごとに分解し，変数に保管する．次に，(2) 基本エラーチェックでは変数に格納されたデータが予め定義されたデータ型（文字型，数値型，日付型等）に合っているか，日付の場合は閏年を考慮して存在する日かなどのエラーチェックを実施する．その後，アプリケーションが追加のエラーチェックを要求されているのであれば，(3) 拡張エラーチェックを実行し，要求に対応する．この様なアーキテクチャを作成し，(R1),(R2),(R6) を共通点とし，(R3),(R4),(R5) は可変点として処理が追加できるようにした．

6.3. ドメイン実現

ドメイン実現では，ソフトウェア資産の詳細設計と実装を行う．エンタプライズ・システムでのソフトウェア資産実装の課題は対象となる事業部や対象業務の違いで使用しているデータ項目が異なることである．画面はデータベースの項目を表示するため，同じ注文入力の画面でも事業部ごとにデータ項目が異なり，再利用が難しくなる．同様にデータベースとの入出力を行うデータ項目も異なるためそのままではソフトウェア資産の再利用ができない．

6.3.1. 画面に関する可変点抽出と抽象化

ブラウザにデータの入力画面を表示するための HTML を図 6 に示す．この画面では，受注品目と受注数量の入力項目のブロックが 1 つずつ表示される．この HTML を出力する再利用可能なプログラムを考える場合，下線を引いた部分が可変点となり，残りの部分は共通点である．従って，共通点の方が可変点より多く，再利用の効果が期待できる．ＡＰ開発者がソフトウェア資産に追加する情報は，画面に表示し利用者が認識するためのデータラベル，プログラム内で処理するためのデータ識別子（当組織のルールでは英数字），桁数の 3 種類が必要となり，具体的には “ 商品コード , prod_cd, 8 桁 ”, “ 数量 , order_qty, 5 桁 ” の 6 項目設定が必要である．このような単一の単純な画面では画面表示機能の部品化は簡単である．しかし，我々の組織で標準的なシステムでは 200 以上の画面があり，それぞれ平均 5 つのデータ項目が使用されているとすれば合計 3000 件 (5 項目× 3 種類× 200 画面) の定義が必要となる．また，実際のシステムの画面はもう少し複雑で，図 2 に示したように RADIO ボタンで選択できたり，プルダウン形式で値を選択できたりするものもある．その選択肢も画面ごとに設定する必要があり，再利用のための作業が増加する．

画面表示の機能を部品化するためには，ＡＰ開発チームの作業量を削減する必要がある．図 7 にこの課題を解決するための仕組みを示す．商品コードなどのデータ項目は，データベース設計時に項目名称や実装用の識別子，桁数などが設計されているため，ＡＰ開発チームは，各画面を設計・実装する際にデータベースの設計情報を参照している．図 7 の (b) 項目オブジェクトはデータベース設計で設計された設計情報をサーバー上のメモリにオブジェクトとして実装したものである．例えば，商品コードは商品登録画面や注文入力画面でも通常同じデータラベル，同じ桁数となるため画面ごとに設定する必要はない．図 7 で今回作成する (a) プログラムには画面に出力したいデータ項目の実装用識別子 prod_cd を設定する．利用者がブラウザに画面を表示する際，prod_cd をキーとしてメモリ中の項目オブジェクトを検索し，項目オブジェクトからデータラベル，入力桁数を取り出すことで画面を表示することができる．更に，項目オブジェクトの入力形式に RADIO ボタンを指定し，選択肢を登録しておけば，利用者が RADIO ボタンで入力できる画面を出力することができる．項目オブジェクトに必要な基本情報はデータベースの設計情報から自動生成可能であるため，ＡＰ開発者は必要に応じて追加情報を登録するだけでよい．また，プログラム開発では，データラベルや桁数を気にすることなく，表示したいデータ項目の実装用識別子を指定するだけで画面部品の再利用が簡単にできる．図 8 に項目オブジェクトを利用する主な画面部品を示す．部品は入力画面と表示画面に分類でき，更にデータを 1 件表示するものと複数件のデータをリスト表示するものに分類できるため，合計 4 種類できる．その他，マトリックス形式で表示する部品も存在するが，ごく一部の機能でのみ利用されている．

項目オブジェクトの考案により画面部品を実現することができた．業務で利用する画面はこれらの部品を組み合わせることで開発することができるが，これらの部品を組み合わせて図 3 に示したような画面全体を中間部品として開発できるようになった．さらにこの中間部品を使って図 4 の画面遷移や図 5 の共通点の部分も部品化できるようにな

りＡＰ開発チームが作成するソースコード量の削減が期待できるソフトウェア資産が構築できた．

6.3.2.　画面部品の高機能化

画面部品の開発は開発工数削減が期待できる一方で画面デザインの制約となる．制約が強ければ利用者の満足度を低下させたり，開発者が再利用に対してネガティブな印象を持ったりすることになる．再利用を全社展開するためには使い勝手の良い画面を従来よりも少ない工数で開発できるようにする必要がある．

Web ブラウザは本来閲覧用のソフトウェアであり，データ登録作業を効率的に行えるように設計されていない．そのためＡＰ開発者は JavaScript などでプログラムを作成して操作性を改善する必要がある．今回開発した画面部品では図９に示すようにカーソルキーやリターンキーで項目移動ができるようになっている．このような機能はマルチブラウザ対応にする必要があり，セキュリティ面の配慮も必要となる．また，ブラウザのバージョンアップにも継続的に対応する必要がある．このような継続的に保守が必要な高機能部品をドメイン開発チームが品質保証して提供することで，ＡＰ開発チームに対してソフトウェア資産を活用した方が低コストでよいシステムが開発できるという動機づけを行っている．

6.3.3.　構成管理

開発したソフトウェア資産はソースコードでＡＰ開発チームに提供し，ＡＰ開発チームでカスタマイズする方法とドメイン開発チームが機能追加してバイナリ形式のライブラリで提供する方法が考えられる．我々は後者を採用した．その理由は欠陥除去すべきバージョンの特定作業が容易で品質保証の点で優れており、新機能を全社展開するのにも好都合であるからである．一方，複数のＡＰ開発が並行して進行している状況ではドメイン開発チームはタイムリーに新機能の提供や欠陥除去が行える高い開発能力が要求されるという制約がある．ドメイン開発チームはリリースしたソフトウェア資産の各バージョンを一元管理し，欠陥が発見された場合，各ＡＰ開発チームが利用しているバージョンのソフトウェア資産に対して欠陥除去を行い，新しいライブラリを提供する．各ＡＰ開発者はライブラリ間の互換性を心配することなく開発作業を継続することができる．

6.4.　ドメイン試験

ドメイン試験では，ドメイン開発で作成した成果物の試験を行う．また，ドメイン試験で開発した成果物をＡＰ開発チームに提供する．当組織でのドメイン試験の課題は品質保証であった．ソフトウェア資産はＡＰ開発チームのニーズに合わせて，短い周期で新機能のリリースを行っている．この際，他の機能への悪影響があれば，ＡＰ開発チームの開発効率を悪化させる恐れや，本番稼働中のシステムのトラブルを引き起こす危険があり，高い品質管理が求められる．日々増加するソフトウェア資産を人手でテストすることはできないため，自動テストツールを活用したテストを

(1) データ入力部品（1 件）

受注番号	2012-9999	受注日	2012-05-15 参照
顧客氏名			
郵便番号	999-9999		
送付先			
電話番号	090-9999-9999		
備考			

(2) データ入力部品（複数件）

No.	商品コード	商品名称	単価	数量
1	参照			
2	参照			
3	参照			

(3) データ表示部品（1 件）

受注番号	2012-9999	受注日	2012-05-15
顧客氏名	住友　太郎		
郵便番号	999-9999		
送付先	大阪市中央区北浜1-1		
電話番号	090-9999-9999		
備考			

(4) データ表示部品（複数件）

No.	商品コード	商品名称	単価	数量
1	LD10L	LED電球 電球色 850lm	3,000	2
2	LD10N	LED電球 昼白色 1000lm	3,500	3

図８．主な画面部品

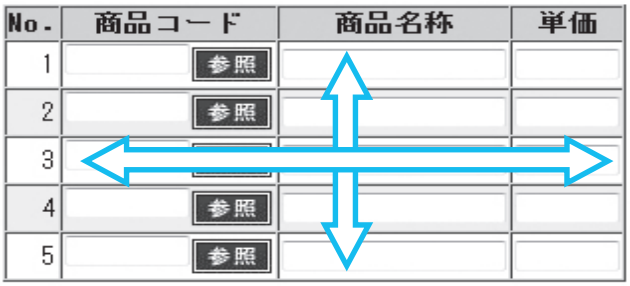

図９．カーソル操作

行っている．作成したテストシナリオは約 1000 種類あり，テスト項目数は約 10000 である．ＡＰ開発チームへリリースする際，一晩かけて自動テストすることで互換性が確保されていることを確認し，不具合流出を防止している．なお，ＡＰ開発チームへの試験成果物の提供はできていない．

7.　アプリケーション開発の実践

7.1.　トレーニング

演習を中心とした３日間のトレーニング・コースを開発した．定員は 10 名で月１回の定期開催に加え，プロジェクトの状況に応じて追加開催した．新入社員は入社後の新人研修で全員がトレーニングを受け，研修課題のプログラムを数本開発する．現在は開発者全員がトレーニングを受けた状態で開発を行っており，定期開催のトレーニングは協力会社から新たに開発に加わる開発者向けに行われている．

7.2. アプリケーション要求開発

アプリケーション要求開発では，ソフトウェア資産を活用し，個別アプリケーションの要求を開発する．事務処理システムでは，ユーザーインターフェースである画面・帳票や，データベース更新時の計算ロジックなどを明確にし，利用部門と合意する．ソフトウェアの再利用を効果的に行うためにはこの段階で再利用可能なソフトウェアで実現可能な仕様になっている必要がある．今回の取り組みでは画面部品が再利用の対象となっているため，画面設計が再利用の度合いを決めるポイントとなる．ソフトウェア資産を活用した実装経験がある開発者が要求開発を行うのが望ましいが，最初の３年間は実装経験のない開発者が要件定義を行うケースも多いため，ドメイン開発チームのメンバーが設計された画面をレビューし，再利用ができるよう指導していた．現在では要求開発の担当者が入社後に実装を経験しているケースが多く，望ましい状態に近づいている．

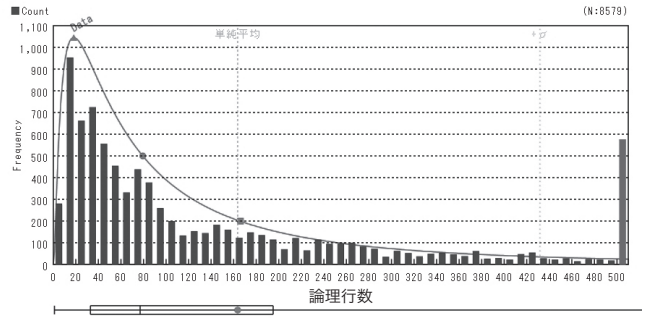

図 10. 作成したプログラムのライン数の分布

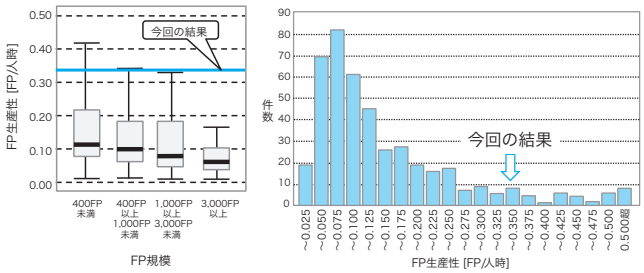

図 11. 生産性のベンチマーク結果との比較

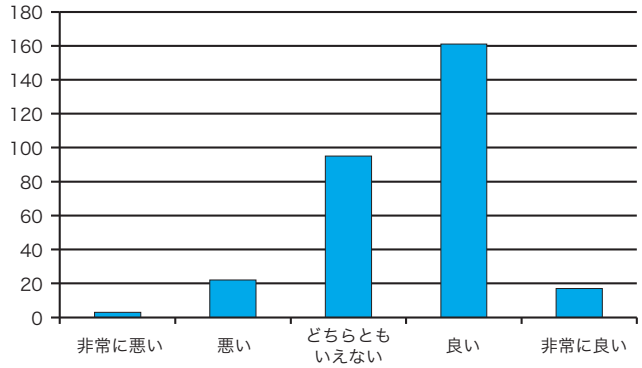

図 12. 操作性に関するアンケート結果

7.3. アプリケーション実現

アプリケーション実現では，アプリケーションの詳細設計と実装を行う．当組織ではソフトウェア資産の可変点の設定は XML 形式のファイルで行い，ソフトウェア資産で実現できない機能は Java でコーディングする．作成したコードはプラグイン方式で既存のソフトウェア資産から呼び出され，要求にあった機能が実現できるようになっている．また，ソフトウェア部品は体系化されており，開発者はネーミングルールにより必要な部品が簡単に特定できるようになっている．

8. 評価

ここでは 5 章の SPL 推進方針で示した課題と解決策の評価を行う．ソフトウェア部品の開発は 1999 年より行っているが 2003 年に部品化の範囲を拡大したため評価対象は 2003 ～ 2012 年の 10 年間で開発した約 300 の業務システムとする．

8.1. 構築したソフトウェア資産の評価

今回の取り組みでは再利用性の高いソフトウェア部品を構築することで，ＡＰ開発チームが開発するソースコードの量を削減し，開発コストを削減することが目的であった．

(1) 開発コード量削減の評価

開発したソフトウェア部品によるコードの削減量を評価するため，これまでにソフトウェア資産を再利用して開発したプログラム 13,174 本を調査した．その結果，4,595 本 (34.9%) のプログラムは，既存の可変点に対する設定のみで実現できていた．残りの 8,579 本 (65.1%) は可変点に対して追加コーディングを行っている．追加コーディングを行ったソースコードのライン数の分布を図 10 に示す．ライン数は空白行やコメントを取り除いた論理行数である．平均値は 164 行であり，中央値は 77 行であった．文献 [11] のデータを利用して，機能当たりのライン数を計算すると 2084 行となり，ソフトウェア資産の再利用によりコード量が約 92% 削減されており，狙い通りのソフトウェア部品が構築できたと考えられる．

(2) 開発生産性の評価

コスト削減の成果を評価するため，文献 [7] のデータと比較する．今回のソフトウェア資産を活用した開発での開発生産性は 0.33FP[※1]/ 人時であった．文献 [7] に掲載されている FP 生産性に今回の結果を加えたものを図 11 に示す．今回開発したシステムの規模は 1500 ～ 5000FP のものが多く，左側の箱ひげ図で見ると一般的な開発の中央値に比べ３～５倍程度の生産性が実現できている．右側の分布を見ても比較的高い生産性が達成できていることが分かる．ただし，生産性は要求される品質に応じて再利用とは無関

【脚注】

※１　FP: ファンクションポイント [12]

係にテスト工数が増加することも考えられるため，この評価結果は参考値と考えるべきである．

(3) 利用者満足度の評価

今回の取り組みでは操作性の高い画面が実現できる高機能な画面部品を提供することで NIH 症候群を回避することが 1 つの対策であった．部品機能の強化による利用者の満足度を評価するためにシステム稼働してから 3 ヶ月後に実施している利用者向けアンケート調査の結果を集計した．その結果を図 12 に示す．有効回答は 298 件であった．縦軸は回答数である．“ 良い ”,“ 非常に良い ” という回答が 188 件 (63%) あり，利用者のニーズに応えられていると考えられる．

8.2. ソフトウェア資産展開の評価

開発したソフトウェア資産は 10 年間社内の全開発プロジェクトで再利用されており，当初の狙い通り展開できている．ソフトウェア資産展開の課題は品質保証のための構成管理と NIH 症候群の回避であった．発見された欠陥の除去はドメイン開発チームが一括して行っているため特に問題は発生してない．また，ＡＰ開発チームが独自に開発すると手間がかかる高機能部品を提供することで NIH 症候群も回避でき積極的にソフトウェア資産を活用する取り組みが進んでいる．

9. 考察

9.1. ソフトウェア資産構築に関する考察

今回調査した 13,174 本のプログラムでは 318 個のソフトウェア部品が延べ 28,444 回再利用されている．再利用された部品の利用頻度を図 13 に示す．横軸は部品を示し，縦軸に利用割合と累積値を示している．最も利用回数の多かった機能部品は 1,929 回使用され，全体の 6.78% を占めていた．上位 15 個の部品で 50.4%，34 個で 70.5%，55 個で 80.3%，120 個で 90.0%，200 個で 92.9%，318 個で 93.8% であった．200 位以下の部品の使用率は 0.02% 以下と低い値であった．当組織では部品が必要になる前に網羅的に部品を提供する戦略をとったが投資効率の観点では開発するソフトウェア資産の範囲は利用頻度の高いものに絞り込む戦略も考えられる．

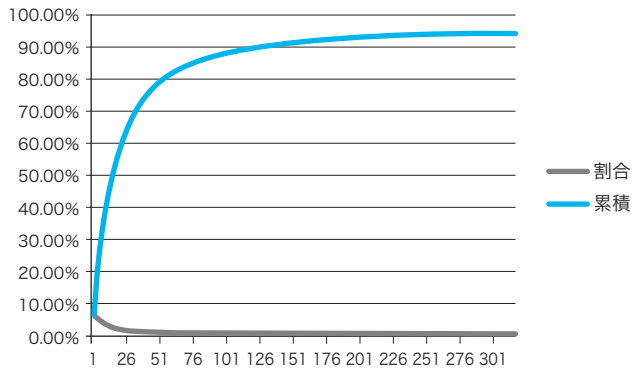

図 13. 部品の利用頻度

9.2. 継続的なソフトウェア資産の機能拡張

SPL ではドメイン開発チームとＡＰ開発チームを分離し，協業することが望ましいとされている [2]．当組織では操作性の高い部品が提供できたこともあり，各ＡＰ開発チームは汎用的な部品を独自に開発せず，ドメイン開発チームに依頼するようになった．要件定義や外部設計の段階からドメイン開発チームに機能追加の相談があり，ＡＰ開発チームと協業しながらソフトウェア資産の拡張と他プロジェクトへの展開ができる体制になっている．開発者にとって魅力あるソフトウェア部品の提供がソフトウェア資産の拡張・展開サイクルがうまく回る重要な要因だと考えられる。

10. まとめ

本論文では SPL のエンタプライズ・システムへの適用事例を示した．ビジネスロジックではなく画面部品を開発することにより開発する成果物量を削減することで開発コストの削減が実現できている．また，画面部品の機能を強化することで利用者の満足度も向上することができた．ただし，今回の取り組みではＡＰ開発のプログラム開発工程のみが対象であった，今後の課題は設計や試験工程に対するソフトウェア資産を構築し，更にコスト削減を図ることである．

【参考文献】

[1] William B. Frakes and Kyo Kang, “Software Reuse Research: Status and Future”, IEEE Transactions on Software Engineering, Vol.31, No.7, pp. 529-536, Jul 2005.

[2] Software Engineering Institute, "A Framework for Software Product Line Practice, Version 5.0”, http://www.sei.cmu.edu/productlines/frame_report/index.html, 参照 2013.4.1.

[3] Klaus Pohl, Guenter Boeckle, Frank J. van der Linden, "Software Product Line Engineering: Foundations, Principles and Techniques", Springer, 2005. (邦訳 " ソフトウェアプロダクトラインエンジニアリング―ソフトウェア製品系列開発の基礎と概念から技法まで ")

[4] Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig, “Defining a strategy to introduce software product line using the existing embedded systems”, Proc. of 6th ACM & IEEE International Conference on Embedded Software, pp.63-72, Oct 2006.

[5] Kentaro Yoshimura, Fumio Narisawa, Koji Hashimoto, and Tohru Kikuno, “Factor analysis based approach for detecting product line variability from change history”, Proc. of MSR 2008, pp.11-18, May 2008.

[6] 石田 裕三 , " エンタープライズ・システムにおけるソフトウェアプロダクトラインの適用 ", 情報処理 , Vol.50, No.4, pp.303-310, Aprl 2009.

[7] 情報処理推進機構 (IPA) ソフトウェア・エンジニアリング・センター (SEC), “ ソウトウェア開発データ白書 2010-2011, p.234,” 情報処理推進機構 , 東京 , 2012.

[8] Linda Northrop, "Software Product Lines: Reuse That Makes Business Sense", ASWEC2006, http://www.sei.cmu.edu/library/assets/ASWEC2006.pdf, 参照 2013.4.1

[9] Yuzo ISHIDA, “Challenge for the SPL Approach in Enterprise Software Development,” NRI Information Technology Report 2007, Vol.8, pp.1-11, 2007.

[10] 野中 誠 , " ソフトウェアプロダクトライン開発のマネジメント：課題と技法 ", 情報処理 , Vol.50, No.4, pp.289-294, Aprl 2009.

[11] 日本情報システム・ユーザー協会 , “ ユーザー企業ソフトウェアメトリックス調査 2011”, 日本情報システム・ユーザー協会 , 2011.

[12] A. J. Albrecht, "Function point analysis", Encyclopedia of Software Engineering, Vol.1, pp.518–524, 1994.