エンタープライズ・システム開発における
品質・コスト・納期の改善に関する研究

提出先　　大阪大学大学院情報科学研究科
提出年月　2014 年 1 月

中村　伸裕

内容梗概

近年，多くの企業で社内業務プロセスがシステム化され，業務効率化や業務品質確保の基盤として必要不可欠なものになっている．このようなエンタープライズ・システムの開発においてコスト削減，品質向上，納期改善は継続的に取り組むべきテーマである．現在の日本の状況は品質が向上しているものの開発コストは増加し，納期も長期化の傾向にある．また，経営的な視点ではシステム構築やシステムの安定稼働については比較的高い評価であるもののビジネスモデルやビジネスプロセスの改革に関してはほとんどが満足しておらず，評価が低い．今後，利用部門の潜在的なニーズを引き出してシステム提案できる手法が必要であると考えられる．今後のシステム開発においては，(1) システム品質を維持したまま開発コストを削減できる開発手法，(2) システム利用部門の潜在的なニーズに合致した付加価値の高いシステムが構築できる開発手法が求められている．

本研究では(1)の課題を解決するためにソフトウェアプロダクトラインの適用によりソフトウェア資産を再利用してコスト削減できるかを評価すると共に統計的品質管理の適用により検証コストの増加を抑制しながら品質を制御できるか評価した．更に(2)の課題を解決するためにアジャイル手法の評価を行った．

先ず，システム開発コスト削減の手段としてソフトウェアプロダクトラインをエンタープライズ・システムに適用した．ソフトウェアプロダクトラインは，製品系列を持つソフトウェアに対してソフトウェアの再利用が効率的に実現できるように共通部分を開発するドメイン開発チームと個々の製品を開発するアプリケーション開発チームが協力して開発する手法であり，組込みソフトウェア等，類似の仕様で多数のソフトウェアを生産しなければならない場合に有効な手法である．組込みソフトウェア開発では適用が進んでいるものの，エンタープライズ・システムでは同じ生産管理システムであっても工場によって製造方法や製品，管理項目（品番，導電率，配合比率等）が異なるため再利用可能なソフトウェア資産の構築が難しく，適用が進んでいない．そこで，製造方法といった業務プロセスの共通点を抽出するのではなく，画面出力等の内部処理の共通点をソフトウェア資産にすることによりソフトウェアプロダクトラインをエンタープライズ・システムに適用した．更に，どの程度作成するソースコード量の削減ができ，どの程度コスト削減できるか評価した．具体的には企業内の販売管理，生産管理，資材管理，経理，人事といった業務全体を 1 つの製品系列と見なし，主に画面出力，画面遷移，データベースの入出力部分を再利用部品として構築し，13,174 機能をソフトウェア部品の再利用で開発した．その結果，開発するソースコード量は約 82%削減でき，開発生産性は業界の標準値に比べて 3～5 倍になった．

次に，品質管理は管理図を使った統計的品質管理手法を適用した．統計的品質管理は戦後，デミング博士が日本の製造業に広めた手法であり，工程のバラツキを管理図等を使って安定状態にあるか監視する手法である．生産設備を使って製造する工場では工程が安定しやすいがソフトウェア開発は人手で行う工程が多いため工程のバラツキが大きく統計的品質管理の適用が難しい．しかし，ソフトウェアのプロセス改善を進め，開発プロセスをうまく定義することでバラツキを減らすことができる．更にソフトウェアプロダクトラインで開発したソフトウェア資産の再利用により開発プロセスの個人差が少なくなり統計的品質管理の適用が可能になった．開発コストの増加を抑制しながら品質を向上させるためには，欠陥の検出を強化するのではなく，欠陥を作り込む設計や製造工程を管理する必要がある．しかし，作り込まれた欠陥は直接測定することができず，レビューやテストで検出された欠陥から計算する必要がある．

例えば, ある機能の外部仕様書に含まれる欠陥は外部仕様書のピアレビューで多くが検出されるがプログラム仕様書のヒアレビュー，コーディングの最中，コードインスペクション，単体テスト，統合テストといったすべての下流工程で検出される機会がある．作込欠陥数を管理する為には，各工程で検出された欠陥を原因工程別に分類し，集計する必要がある．また，あるソースコードの欠陥は, コードインスペクションで多く検出すれば単体テストで検出できる欠陥数は減少する．このように欠陥の検出プロセスはそれぞれ独立して管理するだけでは不十分で複数の欠陥検出工程の状況を統合して監視する必要がある．これらの機能をツール化し，開発現場で評価した結果，管理図等の手法を活用することで再レビュー，再テストの必要性が適切に判断できるようになり，コスト増加を抑制しながら品質の制御ができることがわかった．

最後に，システムの付加価値向上のために，アジャイル手法の中で最もよく利用されている Scrum の試行を行った．アジャイルソフトウェアの 12 の原則に ”顧客満足を最優先し，価値のあるソフトウェアを早く継続的に提供します” という項目が存在するものの価値あるソフトウェアを提供できる仕組みは明確に示されていない．試行の結果，複数の開発者が 1 つの機能を設計することで複数の設計案が提案され，設計案の選択の際，利用者の潜在的なニーズを引き出し評価していることがわかった．またプロセス改善の評価モデル(CMMI)で評価すると要件管理，技術解，妥当性確認，決定分析と分析，プロジェクト管理等のプロセスが改善することがわかった．インタビューの結果，開発者のモチベーションが高いことがわかり従来の開発手法より付加価値の高いシステムが開発できる状態であることを確認した．

以上の研究により，ソフトウェア資産の再利用によりコスト削減が実現でき，統計的品質管理の導入により開発コストを増加させずに品質を向上できることを示した. さらにこれらの手法に Scrum を適用することで付加価値の向上が期待できることを示した．

関連発表論文

Ⅰ．学会論文誌等掲載論文

(1) 中村伸裕，高橋覚，楠本真二, "管理図を用いた欠陥管理システムによる品質改善とその効果, " 電子情報通信学会論文誌 D, Vol. J96-D, No.10, pp. 2214-2225, Oct. 2013

(2) 中村伸裕，服部悦子，永田菜生，楠本真二: “システム価値向上を目的とした Scrum の試行・評価,” SEC journal, No.34, Vol.9, No.3, pp.110-117, Sep. 2013

(3) 中村伸裕, 谷本收, 楠本真二, "ソフトウェアプロダクトラインのエンタープライズ・システムへの適用と評価," SEC journal, No.35, Vol.9, No.4, pp.190-197, Jan. 2014 (採録決定)

Ⅱ．研究集会等発表論文（査読付き）

(1) N. Nakamura, S. Takahashi, S. Kusumoto and K. Nakatsuka, "Approach to Introducing a Statistical Quality Control," IWSM-MENSURA 2011, pp.297-301, Nara, Japan, Nov. 2011.

謝　辞

本研究全般に関し，常日頃より適切なご指導を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻　楠本真二教授に心から感謝申し上げます．

本論文を執筆するに当たり有益なご教示，ご助言を賜りました大阪大学大学院情報科学研究科 井上克郎教授，増澤利光教授に深く感謝致します．また，本研究のベースとなるソフトウェアエンジニアリング全般に関してご指導頂きました岡野浩三准教授，井垣宏特任准教授，肥後芳樹助教に心から感謝します．

また，本研究を大阪大学大学院情報科学研究科で進める機会を与えて頂いた住友電気工業 岩佐洋司前部長（現ミライト情報システム社長），奈良橋三郎部長，業務上の配慮を頂いた住友電工情報システム 白井清志社長，難波明人前理事，西川満取締役，高橋和人部長に深く感謝致します．

ソフトウェアプロダクトラインの研究にあたっては多大なご協力とご指導を頂きました住友電工情報システム 谷本收部長並びに関係者のみなさんに心から感謝申しあげます．また，ソフトウェアプロダクトラインの技術的な指導を頂きました日立ソリューションズ 遠藤潔氏，SRA 林好一氏に心から感謝致します．

統計的品質管理の研究にあたっては管理図の適用評価にご尽力頂いた住友電工情報システム 高橋覚部長，住友電気工業 中塚康介氏をはじめとした多数の関係者のみなさんに深く感謝申したげます．また，管理図のソフトウェアへの適用に関してご指導頂いたソフトバレット 平昌寿氏，乗松プロセス工房 乗松聡氏，富士フイルムソフトウエア 小室睦氏に心から感謝致します．

Scrum の研究に関して実践，評価にご協力頂いた住友電工情報システム 服部悦子氏，住友電気工業 永田菜生氏他，開発チームのメンバーに心から感謝申しあげます．

また，本研究のバックグラウンドになる知識の多くは日本情報システム・ユーザー協会(JUAS)のシステム開発保守 QCD 研究プロジェクトで得たものであり，事務局である JUAS 細川泰秀氏，山田信祐氏，森未知氏はじめ参加メンバーの皆さんに感謝します．特に貴重なアドバイスを頂いた情報処理推進機構 菊島靖弘氏，新谷勝利氏，ジャスティック 太田忠雄氏，JUAS 玉置彰宏氏，日立製作所 初田賢司氏，東京証券取引所 古川正伸氏，NKSJ システムズ 駒井昭則氏，日本経営データ・センター 奥保正氏，東芝インフォメーションシステムズ 北村秀生氏，アイエックスナレッジ 田中一夫氏，リクルートテクノロジーズ 森下哲成氏，大日本印刷 佐藤正人氏，キャノンマーケティングジャパン 原田豊氏，キリンビジネスシステムズ竹中裕二氏に深く感謝します．また，関西で活発な議論をさせてもらった関電システムソリューションズ 川嶋博文氏，ニッセイ情報テクノロジー 内藤康夫氏，中電シーティーアイ 諸岡隆司氏，コベルコシステムズ小山隼人氏に心から感謝します．

最後に長期間にわたる論文作成を支えてもらった家族に感謝します．

目　次

図一覧

表一覧

第1章 はじめに

1.1. 本研究の背景・目的

近年，企業における事務処理の多くがシステム化されており，システムは企業活動に必要不可欠なものになっている．企業のＩＴ投資は金融業では売上の約 5%であり，日本全体では約 1.2%ある[JUAS2013a]．売上高 1 兆円の企業では年間約 120 億円の投資を行っている．システム開発における品質向上，コスト削減，納期短縮は継続的に改善すべき課題であるが，まず，日本および世界のシステム開発の現状を確認する．

(1) 品質状況

2000 年以降表 1 に示すような国民生活に影響を与えるシステム障害が発生している．経済産業省は情報システムの障害発生が社会的影響を及ぼし，日々深刻化している状況を受け，信頼性を高めるための指針として，2006 年に「情報システムの信頼性向上に関するガイドライン」を策定し，2009 年に第２版[KEISAN2009]を発行している．この様な背景によりソフトウェアの品質が企業経営の問題と認識され，コストが増加しても信頼性を優先するシステム開発が増えてきた．2008 年に行われた三菱東京 UFJ 銀行のシステム統合は総費用が 3300 億円という大規模なものであった．この様な背景の中で各企業およびベンダーの改善努力により品質は向上している．図 1 は開発したシステムを顧客に納入した後に発見された不具合密度を示した物であり，縦軸は開発工数(人月)あたり検出欠陥数を示している．総データ件数は 571 件である．データ収集を開始した 2005 年以降，改善傾向にあることがわかる．

表 1. 国民生活に影響を与えたシステム障害の例

発生年	システム障害内容
2002 年	第一勧業，富士，日本興業の３銀行のシステムをみずほ銀行として一本化するシステム統合で ATM の障害や公共料金の自動引き落としが遅延
2005 年	みずほ証券が東京証券取引所に対して行った誤発注の取り消しがソフトウェアの不具合により実施できず約 400 億円の被害
2006 年	JR 東日本で Suica の利用者が改札を通れなくなる
2007 年	全日空国内予約システムの不具合により 130 便が欠航し，4 万人以上の利用者に影響を与えた
2007 年	東京周辺の JR や私鉄など 16 社合わせて 662 の駅で自動改札機が作動しない．自動改札のゲートを開放して対応．

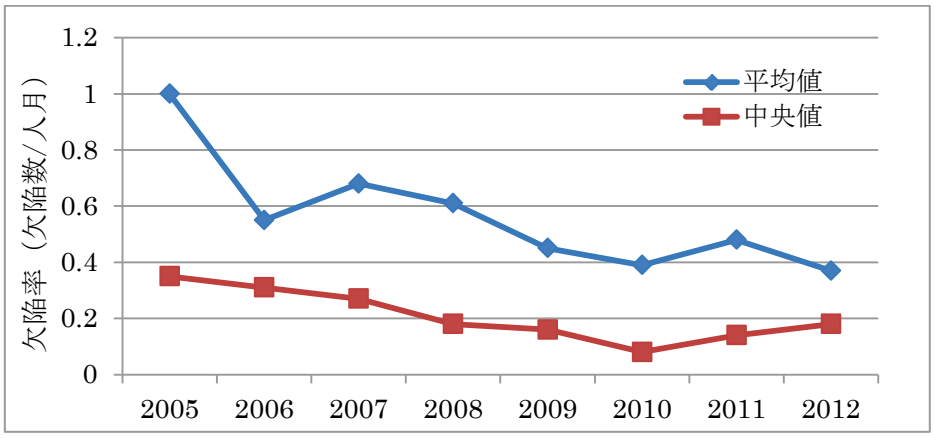

図 1. システム品質の経年変化 [JUAS2013b]

(2) 開発コストの状況

KLOC 当たりの開発コストの平均値を図 2 に示す. 開発コストは増加傾向にあることがわかる．品質が改善傾向にあることから，システムの欠陥を減らす為にテスト項目を増やす等の対策を実施した結果，開発コストが増加しているものと考えられる．これらの状況から品質を維持しながらコスト削減を実現できる手法の確立を 1 つの課題とした.

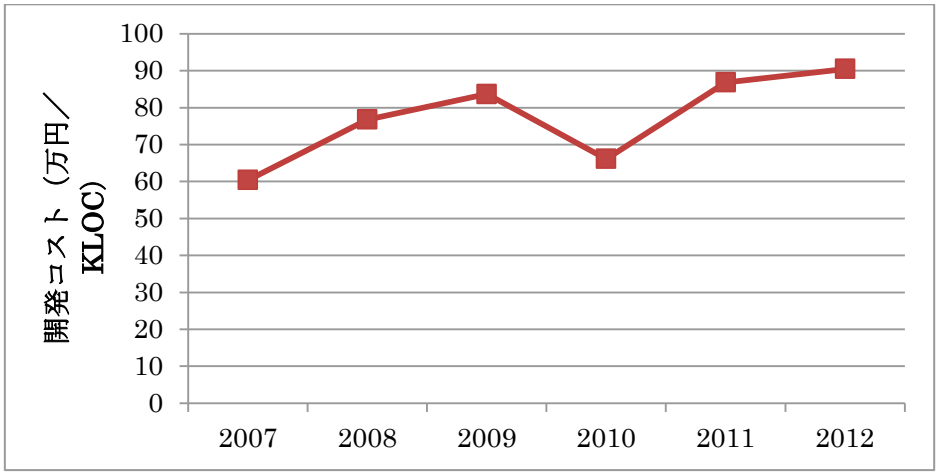

図 2. KLOC 当たりの開発コスト [JUAS2013b]

(3) システム開発工期（納期）の状況

システムの開発工期は開発工数と深い関係にあり COCOMO(Constructive Cost Model)[BOEHM1981]法では式 1 で示される.

$$\text{全体工期} = a \times \sqrt[b]{\text{全体工数}} \qquad \textbf{式 1}$$

文献[JUAS2013b]では b=3 で固定し，a の値の経年変化を測定している．その値を図 3 に示す．係数 a が増加していることから工期が長期化していることがわかる．ただし，納期はビジネス上の要件（法律の改定，企業の合併，新工場の建設等）や企業内の優先順位，開発時期による投入可能な開発者人数のばらつき等により大きく変動するため，開発技術だけで解決できる問題ではない．係数 a の改善は本研究の課題とはせず，単純に工数削減による納期短縮を目指すことにした.

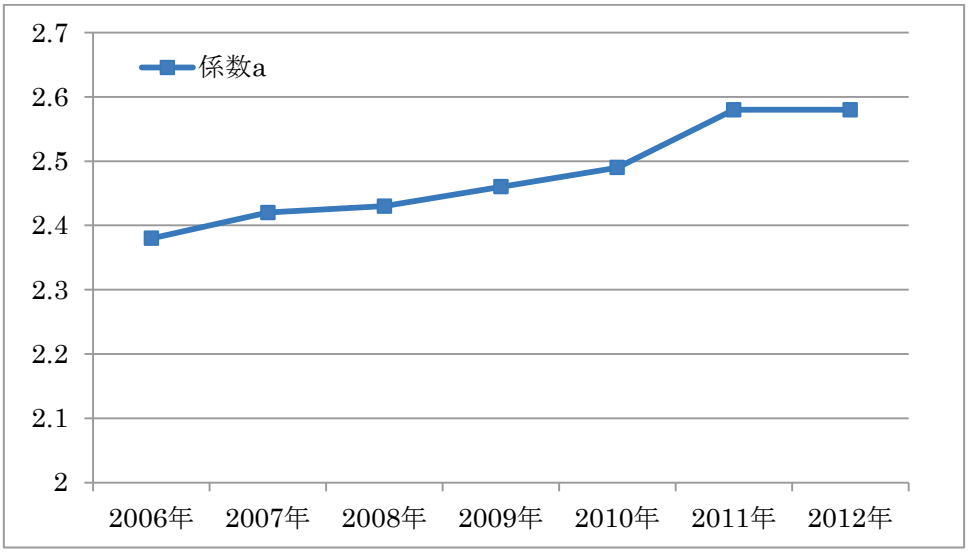

図 3. 工期係数 a の推移

(4) 経営企画部門の評価

1075 社の経営企画部門に対して実施したアンケート[JUAS2013a]によると，IT 投資で改善したい経営課題は業務プロセスの効率化（省力化，業務コスト削減）が 52%で 1 位になっており，現在でも業務の効率化を目としたシステム開発ニーズは高い状態にある．業務効率化を目的としたシステム開発投資は通常，企業内の投資回収期間等の基準をクリアする必要がある．開発コストの削減は投資回収期間の短縮や従来，投資基準に合わない業務のシステム化が可能になるといったメリットがあり引き続き改善ニーズがあると考えられる．また，品質に関してはシステムの安定稼働に対する期待に応えられていると回答した割合は 58%であり，満足している企業が多い．一方，ビジネスモデルの変革への期待に応えられているという回答は 3.1%であり，ビジネスプロセスの改革については 7.1%であった．ビジネスモデルの変革という課題はビジネスモデルの構築，業務設計といったシステム開発前の工程によるところが大きく，3.1%という満足率から考えると現在のシステム開発部隊の能力ですぐにこの課題を解決することは難しいと考えられる．従来のシステム開発では与えられた問題を解決するロジカルシンキングといった思考が重視されたが，ビジネスモデルの改革は問題を定義するところから始めるためクリエイティブシンキング[MATSUBAYASHI1981]のような思考が要求される．ビジネスモデルという上位の課題に取り組む前段階としてシステム設計のフェーズで創造力を発揮し，利用部門の潜在的なニーズを引き出し，提案できる能力を身に付けることが必要であると考えられる．利用部門の潜在的ニーズに適合した付加価値の高いシステムが開発できる開発手法を確立することをもう一つの課題とした．

(5) 海外の状況

文献[CAPERS2013]に国別のソフトウェアをリリースした後のFunction Point(FP[1])当たりの欠陥密度が示されており，上位 10 の国を表 2 に示す．日本は 1 位であり，高品質であることがわかる．

[1] FP: ソフトウェアの規模を測定する手法の1つであるファンクションポイント法で測定した指標の単位[ALBRECHT1994]

表 2. リリース後の FP 当たりの欠陥密度

	Country	Delivered Defects
1	Japan	0.29
2	India	0.34
3	Denmark	0.38
4	Canada	0.39
5	South Korea	0.39
6	Switzerland	0.40
7	United Kingdom	0.40
8	Israel	0.41
9	United States	0.47
10	France	0.49

文献[STANDISH2013]には開発プロジェクトの推移が示されており，表 3 に成功プロジェクトの割合の経年変化を示す．Failed はプロジェクトのキャンセルや本番化しなかったプロジェクトの割合であり，Challenged は予算超過，納期未達，機能の一部が開発できなかったプロジェクトの割合である．成功率は改善されているものの 39%という比較的低い値となっている．

表 4 に Challenged に分類されたプロジェクトの要素を示している．TIME は納期未達，COST は予算超過の割合を示している．コスト超過のプロジェクトは 2012 年に 59%と最高値になっており，開発予算に対して開発生産性が低い状態にあることがわかり，開発コスト削減や納期短縮の改善ニーズは海外でも続いていると考えられる．

表 3. 成功プロジェクトの割合の推移

年	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

表 4. 失敗プロジェクトの納期未達，予算超過の推移

年	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%

(6) 日本と海外の開発スタイルの相違

日本のシステム開発はアウトソーシングが中心であることもあり，大半がウォーターフォール型となっている[JUAS2013b]．しかし，海外ではアジャイル開発が普及[VERSION2001]しており，アジャイル開発がビジネスの競争力強化に貢献している[IPA2012]．

本研究の目的は，以上のデータから得られた下記２つの課題に対する解決策を示すことであり，以下の開発手法の確立を目指す．

(a) 要求水準に合った品質を確保した上で開発コストを削減できる開発手法の確立

(b) システムの付加価値向上が向上できる開発手法の確立

1.2. 本研究の概要

本研究では，まずソフトウェア資産の再利用によってシステム開発コストを削減し，次に適切なコストで品質を確保する管理手法を確立することを目指す．最後にこれらの手法に要求引き出しの効果が期待できるアジャイル手法を組み合わせることで利用部門の潜在ニーズを満たした付加価値の高いシステム開発手法の確立を目指す．図 4 に目的と実現手段の関係を示す．まずソフトウェア資産の再利用により開発する成果物量を削減し，開発工数を削減するとともに作業の増加に合わせて増加すると考えられる欠陥の作込量も削減する．次に欠陥検出プロセスの最適化を行い，欠陥検出工数の削減を図り，コスト削減と品質確保の両立を図る．また，開発プロセスに異常が発生して通常より多くの欠陥が発生している場合は欠陥予防を行い，欠陥の作込量を制御する．これらの施策により１つのゴールである，開発コスト削減と工数削減に伴う納期短縮と目標とする欠陥密度を実現する．

次に高速・高品質でソフトウェアを開発できるプロセスを利用して短期間のサイクルで開発を繰り返すアジャイル手法を実施し，ソフトウェアの魅力品質の向上を図る．アジャイル開発で低コスト・高品質の開発プロセスが必要となる理由は以下の２つである．

(a) 短い期間で開発を繰り返すことで利用者からの明示的な要求がより多く収集できる，

(b) 欠陥修正の付加価値を生まない工数が少なく，利用者の潜在的なニーズを引き出す時間が確保できる．

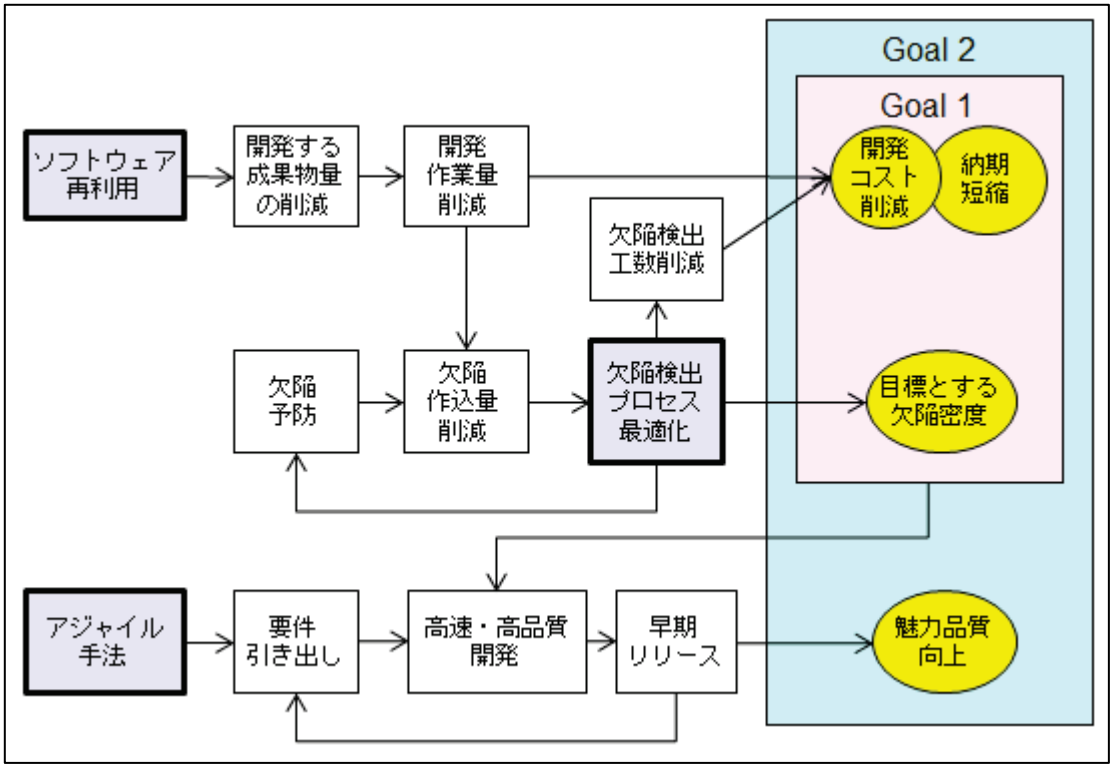

図 4. 本研究の目的と実現手段

1.3. 本論文の構成について

本論文ではまず第２章でソフトウェア再利用，ソフトウェア品質管理，アジャイルに関する既存技術を確認し，試行する手法を説明する．第３章では手法を適用する組織の特徴を説明する．第４章ではソフトウェアプロダクトラインを適用した試行開発の結果を報告し，ソフトウェアプロダクトラインが開発量の削減と開発工数削減に効果があることを示す．第５章ではソフトウェアプロダクトラインの開発に統計的品質管理手法の１つである管理図を適用したプロジェクトの試行結果を示し，管理図を用いて効果的に開発コストと品質の制御ができることを示す．第６章では Scrum の導入による付加価値の向上について述べ，第７章で総括する．

第2章 既存技術

2.1. ソフトウェア再利用

2.1.1. ソフトウェア再利用の形態

ソフトウェアの再利用はシステム開発の品質，コスト，納期の改善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［WILLIAM2006］．ここではサブルーチンにより再利用からソフトウェアプロダクトラインが提案されるまでの再利用の形態を確認しておく．

(1) サブルーチンによる再利用

ソースコード中の共通部分を切り出し，再利用する．特別なスキルは必要なく多くの開発組織で利用されている．再利用の計画がなくても開発中に随時，再利用可能なサブルーチンを開発することができる．当組織の実績ではサブルーチンによるコード作成量の削減率は２割以下であった．

(2) モジュールによる再利用

サブルーチンはボトムアップのアプローチであるが，モジュールはトップダウンで一連の機能を分割したものである．一般的にはサブルーチンとデータ構造の集合体となる．サブルーチンによる再利用に比べ，事前の計画，設計が必要となるが，再利用範囲の拡大が期待できる．

(3) オブジェクト指向開発による再利用

データと処理をカプセル化したもの．継承の仕組みによりカスタマイズすることができることが特徴．ソフトウェア部品は実世界をモデル化して識別されたクラスが基本となる．

(4) コンポーネント指向開発による再利用

モジュールを汎用化したもので，インターフェースによりカスタマイズが可能である．オブジェクト指向開発による再利用は既存のソフトウェア部品を派生させて再利用するが，コンポーネント指向開発では既に開発されているコンポーネントを組み合わせて新しいソフトウェアが開発できることを目的としている．

(5) ソフトウェアプロダクトライン

(1)～(4) の技術は再利用されるソフトウェア部品に着目しているが，ソフトウェアプロダクトラインはソフトウェア資産の構築とソフトウェア資産の再利用プロセスを改善してより効率的なソフトウェア部品の再利用を実現させるものである．

本研究では最新技術であるソフトウェアプロダクトラインの試行と評価を行うが，ソフトウェア資産の構築では(1)から(4)の技術が利用されることになる．

2.1.2. ソフトウェアプロダクトラインについて

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[LINDA2006]，2000 年頃から計画的な再利用を目的としてソフトウェアプロダクトラインの考え方が広がっている．従来の製品毎の開発プロセスとソフトウェアプロダクトライン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. 図 5[KLAUS2005]にソフトウェアプロダクトラインの概要を示す.

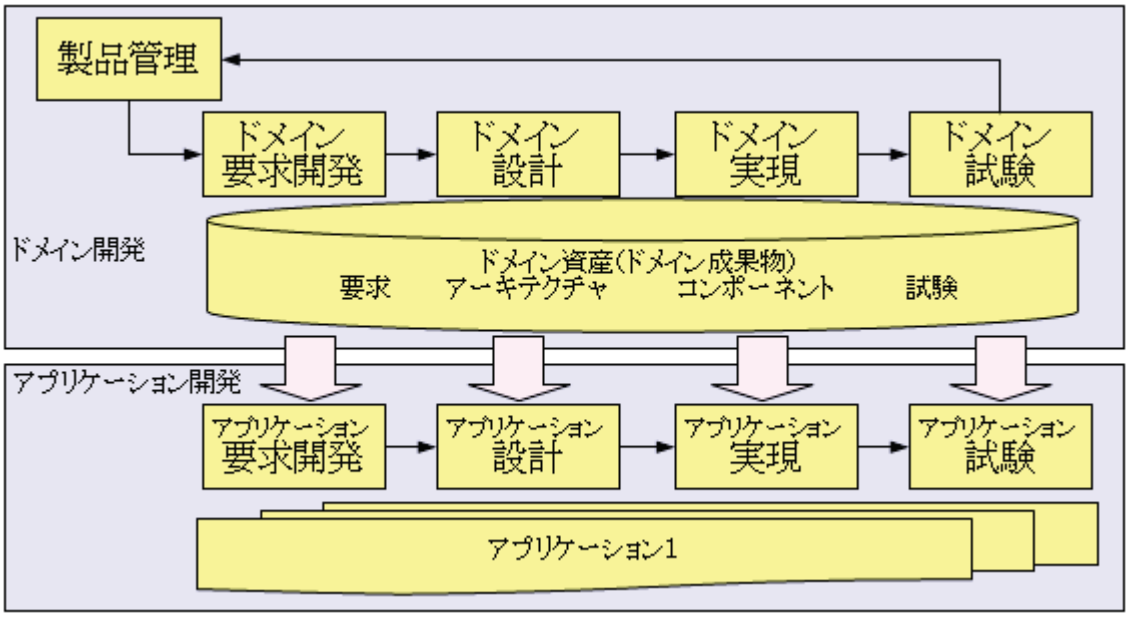

図 5. ソフトウェアプロダクトライン概要［KLAUS2005]

ドメイン資産を構築するドメイン開発は以下の５つのプロセスで構成される.

(1a) 製品管理：主にソフトウェア製品系列のマーケティングを扱い，製品のロードマップを出力する.

(1b) ドメイン要求開発：製品ロードマップを入力とし，製品系列の共通要求及び可変要求を抽出する.

(1c) ドメイン設計：製品系列の共通要求及び可変要求から製品系列アプリケーションの高位の構造を示す参照アーキテクチャを構築し，可変性に対応する仕組みを設計する.

(1d) ドメイン実現：再利用可能な詳細設計と実装を行う.

(1e) ドメイン試験：再利用可能なコンポーネントの妥当性確認と検証を行う.

ドメイン資産を活用するＡＰ開発は以下の４つのプロセスで構成される.

(2a) アプリケーション要求開発：個々の製品の要求仕様を開発し，ドメイン成果物との差分を明確にする.

(2b) アプリケーション設計：参照アーキテクチャを利用し，アプリケーションのアーキテクチャを設計する.

(2c) アプリケーション実現：再利用可能なコンポーネントを利用し，アプリケーションを実装する.

(2d) アプリケーション試験：アプリケーションの仕様に照らして妥当性確認と検証を行う.

2.1.3. ソフトウェアプロダクトライン実践の課題

ソフトウェアプロダクトラインはエレベータの制御ソフトといった具体的なドメインへの適用はイメージしやすい．既に存在している用途別のエレベータの共通点と相違点が比較的簡単に抽出できると考えられる．今回は業務システムという比較的広い範囲のソフトウェアを対象とするため, 共通点と相違点をどう抽出するかという課題がある. 例えば製品の出荷システム，決算業務を行う経理システム，社員の人事考課を管理する人事システムといった業務システムに対して共通点と相違点を明確にする必要がある．さらにコスト削減の為にはドメイン資産が幅広く再利用される必要があり，有用性，可用性の高いドメイン資産を構築する必要がある．

2.2. ソフトウェア品質管理

ソフトウェアの欠陥はレビューやテストで検出する．レビューやテストにはさまざまな手法が存在するが，本研究は欠陥数を 0 に近づけるのが目的ではなく，目標とした欠陥密度に欠陥数を制御することが目的である為，再レビュー，追加テスト等の判断ができる品質管理の手法を検討する．

2.2.1. ソフトウェア品質管理の手法

テストやレビューの完了判断に利用できるよく知られる手法として以下のものがある．

(1) コードカバレッジを基準としたテストの実施

ホワイトボックステストでソースコード中の命令がどの程度網羅されたかを示す指標である．コードカバレッジは，通常 3 種類の基準を用いて測定される．

・命令網羅(C0)：コード中のすべての命令が実行された際，100%となる．
・分岐網羅(C1)：コード中のすべての分岐が実行された際，100%となる．
・条件網羅(C2)：コード中のすべての分岐の組み合わせが実行された際，100%となる．

テストの基準を C0 から C1, C2 とあげれば残存欠陥を検出できる確率が高まり，信頼性の向上が期待できるが，テスト工数が増加するためシステム開発コストの増加や納期の長期化が課題となる．

(2) 信頼度成長曲線の利用によるテストの終了判断

信頼度成長曲線は縦軸に検出欠陥数，横軸に実施したテスト項目数または日付をとった折れ線グラフで時間と共に右上方向にプロットが追加される．一般的にはテスト項目数の増加に従って検出量が減る為，作り込まれた欠陥数に収束していく．図 6 に示すようなゴンペルツ曲線等を使って近似曲線を求めることで残存欠陥数を予想することができ，テストを完了するか，継続するかの判断に利用することができる．一般的には単体テストではなく，プログラムの開発が終了した後に行う統合テストで利用されるケースが多い．この手法は欠陥がシステム全体に均一に分布する場合は有効であるが，欠陥が偏在している場合，テストの順序によっては成長曲線が一端収束傾向を示した後，急激に増加することがあり，制約となっている．

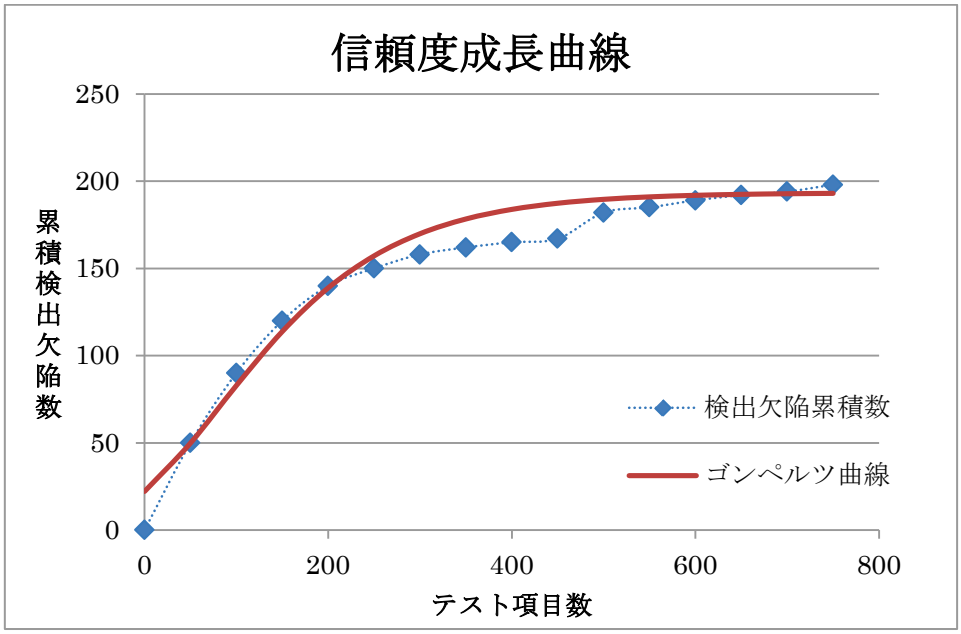

図 6.信頼度成長曲線のサンプル

(3) 管理図

シューハート管理図(JIS Z 9021)は，品質や製造工程が統計的に安定な状態にあるか判断するグラフのことである．日々の工程管理で利用できるため，品質の平準化を実現するために有効である．ソフトウェア開発の利用に関してはプロセス改善モデルである CMMI[MARY2009]ではレベル４で管理図等の統計的手法を使った管理を要求しており，成熟した組織で利用されている．設計，開発プロセスの進行に合わせて監視できるため，欠陥の再発防止策が実施できる．

本研究では管理図の試行と評価を行う．信頼性成長曲線は通常，設計・実装が終わった後の統合テストで利用されるため開発コスト削減の効果が期待できない．一方，管理図は設計・実装中に利用できるため欠陥予防によるコスト削減の効果が期待できる．

2.2.2. 管理図について

管理図は品質や工程が統計的に安定しているかどうか判断するために利用するグラフである．図 7 に長さ，重量，時間といった計量値に利用される X 管理図の例を示す．縦軸が管理指標を表す．過去のデータから平均値を求めて，中心線(CL)を設定する．また，標準偏差（σ）を計算し，CL+３σを上方管理限界(UCL)，CL－3σを下方管理限界(LCL)とする．さらに CL と UCL,LCL の間を３分割する．横軸は管理対象のデータを発生順に並べる．JIS 規格 JIS Z 9021 では異常判定の８つの判定基準が示されているが，そのうち最もわかりやすい判断基準は(a)プロットした点が管理限界を超えるもので，他に(b)連続する３点中２点が３分割した領域 A にあるといったものが示されている．

一方，文献[STEPHEN2002]ではソフトウェアへの管理図の適用は正規の統計的プロセス管理(SPC)やプロセス能力に使うのではなく，むしろ一貫性と安定性を改善するためのツールとして用いることが有効であることが示されている．

管理図は管理対象の種類によりいくつかの種類が用意されているが，欠陥数の管理は計数値を扱う u 管理図の利用が基本となる．u 管理図の例を図 8 に示す．縦軸はプログラムの欠陥密度（規模あたりの欠陥数）を示している．横軸はプログラムを表す（図 8 では，10 個のプログラム 1～10，テスト実施順に並べている）．各プログラムは登録，照会，変更，削除の機能を持ち，複数のソースファイルで構成されているため，各点は総ライン数に対する欠陥の加重平均となっている．中央線(CL)は組織全体の実績から求めた基準値である．u 管理図の管理限界は中心線(CL)と対象物の規模を使って式２で求めることができる[JIS2007][STEPHEN2002].

$$\mathbf{UCL} = \mathbf{CL} + \mathbf{3} * \sqrt{\boldsymbol{CL}/\textit{規模}} \qquad \text{式 2}$$

規模に応じて階段状の管理限界になっている. 規模の大きいものほど管理限界の幅が狭くなる. 更に，中央線と管理限界の 2/3 の位置に補助線を描いている．この例では下方管理限界(LCL)は計算上 0 以下となるが，欠陥数がマイナスになることはないので LCL は描画していない.

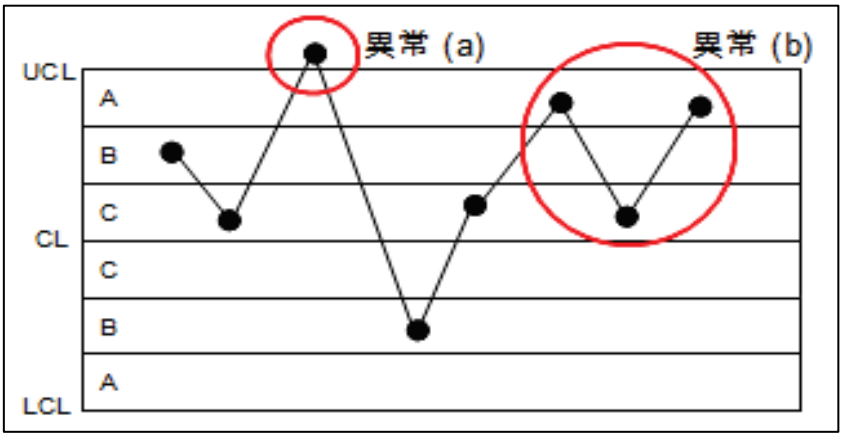

図 7. X 管理図のサンプル

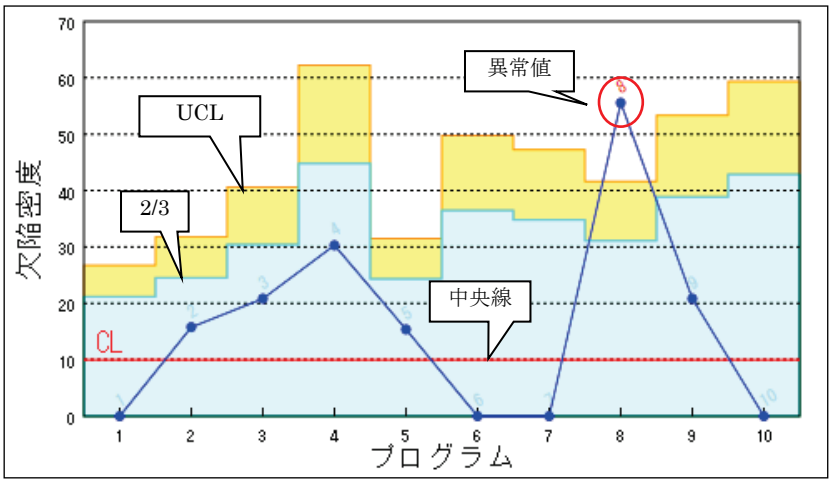

図 8. 検出欠陥密度の u 管理図

2.2.3. 管理図導入の課題

管理図は多くの工業製品で工程の管理に利用されている．しかし，公開されているソフトウェア開発への適用事例が少なく，実践上のノウハウが得られにくい．実践を通じて明らかにしていく必要がある．また，ソフトウェア技術者の多くは統計的な手法に精通していないため，開発現場に普及させる為にはツールの提供が必要と考えられる．

2.3. アジャイルソフトウェア開発

2.3.1. アジャイルソフトウェア開発について

アジャイルソフトウェア開発は迅速かつ適応的にソフトウェア開発を行う軽量な開発手法群の総称である．Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas による以下のアジャイルソフトウェア開発宣言がアジャイルソフトウェア開発の定義であると考えられる．

アジャイルソフトウェア開発宣言

私たちは，ソフトウェア開発の実践あるいは実践を手助けする活動を通じて，よりよい開発方法を見つけだそうとしている．この活動を通して，私たちは以下の価値に至った．

プロセスやツールよりも個人と対話を，
包括的なドキュメントよりも動くソフトウェアを，
契約交渉よりも顧客との協調を，
計画に従うことよりも変化への対応を，

価値とする．すなわち，左記のことがらに価値があることを認めながらも，私たちは右記のことがらにより価値をおく．

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

© 2001, 上記の著者たち
この宣言は，この注意書きも含めた形で全文を含めることを条件に自由にコピーしてよい．

また，アジャイルソフトウェア開発宣言の背後にある原則として表 5 に示すアジャイルソフトウェアの 12 の原則がある．比較的歴史の長いアジャイルソフトウェア開発手法として，Scrum, Crystal Clear, XP(eXtreme Programming), Adaptive Software Development, FDD(Feature Driven Development), Dynamic System Development Method(DSMM)等がある．文献[VERSION2001]にアジャイルソフトウェア開発手法の利用状況の調査結果が示されており，Scrum および Scrum と XP を組み合わせた利用が全体の 74%を占めている．本研究では利用実績の多い Scrum の評価を行う．

表 5. アジャイルソフトウェアの 12 の原則

1	顧客満足を最優先し，価値のあるソフトウェアを早く継続的に提供します.
2	要求の変更はたとえ開発の後期であっても歓迎します. 変化を味方につけることによって，お客様の競争力を引き上げます.
3	動くソフトウェアを，2-3 週間から 2-3 ヶ月というできるだけ短い時間間隔でリリースします.
4	ビジネス側の人と開発者は，プロジェクトを通して日々一緒に働かなければなりません.
5	意欲に満ちた人々を集めてプロジェクトを構成します. 環境と支援を与え仕事が無事終わるまで彼らを信頼します.
6	情報を伝えるもっとも効率的で効果的な方法はフェイス・トゥ・フェイスで話をすることです.
7	動くソフトウェアこそが進捗の最も重要な尺度です.
8	アジャイル･プロセスは持続可能な開発を促進します. 一定のペースを継続的に維持できるようにしなければなりません.
9	技術的卓越性と優れた設計に対する不断の注意が機敏さを高めます.
10	シンプルさ（ムダなく作れる量を最大限にすること）が本質です.
11	最良のアーキテクチャ・要求・設計は，自己組織的なチームから生み出されます.
12	チームがもっと効率を高めることができるかを定期的に振り返り，それに基づいて自分たちのやり方を最適に調整します.

2.3.2. Scrum について

Scrum は竹内 弘高氏，野中 郁次郎氏が日米の製造業の設計・開発を調査した文献[TAKEUCHI1986] が起源となっており，Ken Schwaber 氏，Jeff Sutherland 氏がソフトウェア開発へ適用したものである．以下，文献[KEN2011a]を基に Scrum の概要を説明する．

Scrum は，複雑で変化の激しい問題に対応するためのフレームワークであり，可能な限り価値の高いプロダクトを生産的かつ創造的にリリースするためのものであり，役割，成果物，イベントが定義されている．

(1) 役割

ソフトウェア開発に携わる人について，次の(R1)～(R3)の３つの役割が定義されている．(R1) プロダクトオーナー：ソフトウェアの開発順序を決める権限を持ち，価値を最大化する責任を持つ要求者である．(R2) 開発チーム：ソフトウェアを開発チームであり，通常 5～9 名で構成される．(R3) スクラムマスター：開発チームに Scrum を正しく理解させ，開発チームが成果を上げるために支援や奉仕を行う．一般的なプロジェクトマネージャーとは役割が異なる．

(2) 成果物

次の(P1)～(P3)の３種類の成果物が定義されている．(P1) プロダクトバックログ：プロダクトオーナーが作成するソフトウェア要件の一覧である．要件には優先順位が示されている．(P2) スプリントバックログ：次回リリースする機能をプロダクトバックログから選択したもので開発チームが作成する．(P3) インクリメント：スプリントバックログの機能を既存の（前回リリースした）ソフトウェアに追加実装したもので，動作して，かつ，リリース可能なものである．

(3) イベント

次の(E1)～(E5)の５種類のイベントが定義されている．図 9 に Scrum の１サイクルを示す．(E1) スプリント計画ミーティング：２つのパートに分かれており，Part 1 ではプロダクトオーナーが作成したプロダクトバックログを元に何をすべきか理解する．Part 2 では今回の開発で

作成すべき機能とそのための作業を計画する．(E2) スプリント: 計画した機能を開発する．スプリントの期間は通常２週間から１ヶ月とされている．(E3) デイリースクラム: 毎日行う 15 分以内のミーティングで進捗の評価と次に行うタスクの調整を行う．(E4) スプリントレビュー: 開発チームが開発した機能のデモをプロダクトオーナーに対して行う．プロダクトオーナーはデモを確認して，完了しているかどうかの判断を行う．(E5) スプリントレトロスペクティブ: 人・関係・プロセス・ツールの観点から今回のスプリントを点検し，改善計画を作成する．

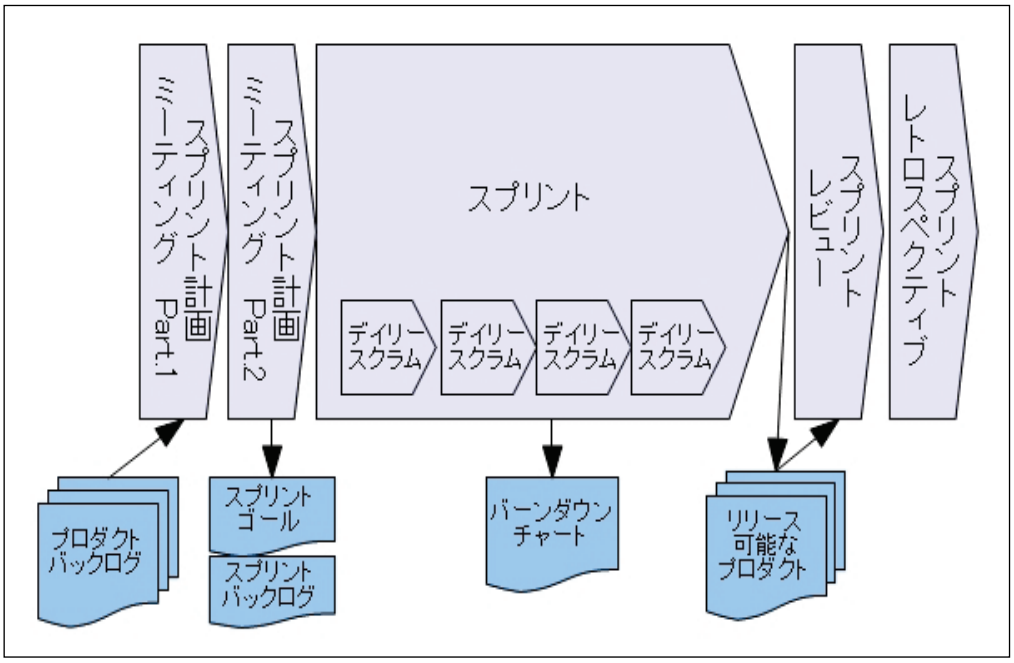

図 9. Scrum のイベントと成果物

2.3.3. 課題

アジャイルソフトウェアの 12 の原則に ”顧客満足を最優先し，価値のあるソフトウェアを早く継続的に提供します” という項目が存在するものの価値あるソフトウェアを提供できる仕組みは明確に示されていない．付加価値の高いソフトウェアを開発するためにはさまざまなアイデアが創出され，提案されることが必要であり，更にそのためには開発者のモチベーションが高く，設計スキルも向上する必要がある．Scrum にこのような効果があるか評価する必要がある．

第3章 適用組織について

ここではソフトウェアプロダクトライン，統計的品質管理，Scrum の適用を行った住友電気工業株式会社の情報システム部門の特徴を説明する．

(1) 体制

情報システム部門は事業部単位ではなくコーポレート部門に組織されており，１つの組織で全社横断的に生産管理，在庫管理，販売管理，購買管理，物流管理，会計，人事等のといった社内の業務プロセスを支援するエンタープライズ・システムの開発・保守・運用を行っている．システムの開発は情報システム部が企画，要件定義を担当し，設計・開発・保守は情報子会社である住友電工情報システム株式会社が担当しており，開発に関与する技術者は合計約 200 名である．また，新技術の導入や標準化は親会社の情報システム部が担当している．

(2) 開発技術

システム開発の標準化を重視している組織であり，開発ツール，開発プロセスは全社で統一している．オープンソースソフトウェアの開発ツールを積極的に採用しており，Java, Tomcat, Eclipse, PostgreSQL 等を全てのシステム開発で利用している．

(3) 方法論

設計手法としてデータ中心設計を採用し，全システムで ER 図(Entity Relationship Diagram)を作成している．データベース設計の属人性を極力排除し，正規化されたデータベースを実装することが原則となっている．高品質のデータベース設計がシステム開発の生産性向上，品質の基盤となっている．

(4) ソフトウェア開発の改善活動

ソフトウェア開発の品質，納期，コストの改善は継続的に行っており，これまで表 6 に示すような取り組みを行っている．特に CMMI [MARY2009] [WATTS1991]のモデルを使ったプロセス改善は継続的に実施しており, CMMI の評定資格を持っている技術者が 50 名在籍している. なお, CMMI を使った評定は米国, 中国をはじめ 88 カ国で実施されており, 評定結果は5段階で示される. レベル1はレベル 2 に未達の状態を示している. 状態が定義されているレベル 2～5 の割合はそれぞれ 24.3%, 63.6%, 1.7%, 6.3%であり, 多くの組織がレベル 3 に留まっている. [CMMI2013]

表 6. 品質・コスト・納期改善の取り組み

年	取り組み内容
1994	データ中心設計の導入
1999	Java による自社フレームワークを開発
2001	CMM によるプロセス改善を開始
2003	CMM レベル３達成
2003	ソフトウェアプロダクトライン　試行開始
2007	統計的品質管理　試行開始
2011	CMMI レベル５達成

(5) ハードウェア環境

1997 年以降ＰＣサーバーを採用し，コスト削減を図っている．現在は社内にクラウドサーバーを設置し，ハードウェアを共用しているが，Intel ベースの CPU を利用した Linux サーバー上でシステムが稼働している．

(6) ツール開発

システム開発を支援する開発ツールは必要に応じて自社開発することも多く，1989 年から 10 年間はソースコードジェネレータを自社開発して利用していた．また，2005 年にシステム開発で作成された仕様書等のドキュメントを一元管理する文書管理ツールを自社開発している．その他，サーバーの異常監視を行う運用支援ツールも自社開発して運用している．

第4章 ソフトウェアプロダクトライン適用によるソフトウェア再利用の試行と評価

4.1. 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良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［WILLIAM2006］．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)[SEC2009][KLAUS2005]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 は組み込み系の分野では多くの事例が報告[YOSHIMURA2006][YOSHIMURA2008]されているが，エンタープライズ・システムの事例報告は少ない[ISHIDA2009]．

住友電気工業の情報システム部門(以降，当組織)は第 3 章で述べたとおりエンタープライズ・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[IPA2010]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．

4.2. 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．

4.3. エンタープライズ・システムへの SPL 適用の課題

文献[ISHIDA2009],[ISHIDA2007]ではエンタープライズ・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 時間稼働等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，事業部毎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[ISHIDA2007]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．

4.4. 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．

4.4.1. 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[ISHIDA2009][ISHIDA2007]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.

4.4.2. 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等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等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.

4.4.3. SPL 導入の組み込み系との共通課題

文献[NONAKA2009]では組み込み系，エンター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開発で利用された後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.

4.5. ドメイン開発の実践

4.5.1. 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エンタープライズ・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 10 にエンタープライズ・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等，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.

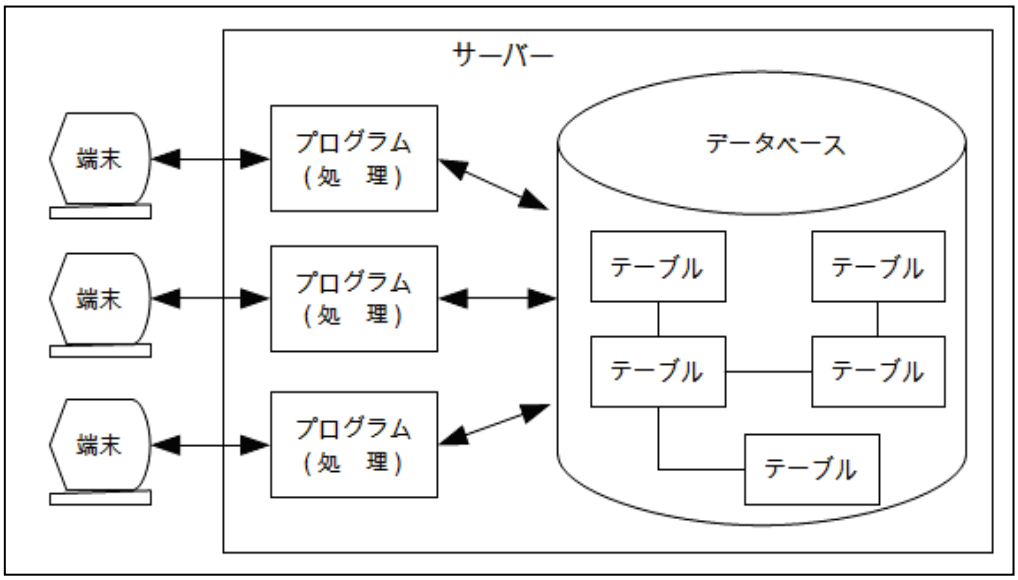

図 10. エンタープライズ・システムの構造

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 11，図 12 に業務で使用する画面の例を示す．図 11 は商品を登録する画面である．図 12 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 12 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 11 では１ブロック，図 12 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）

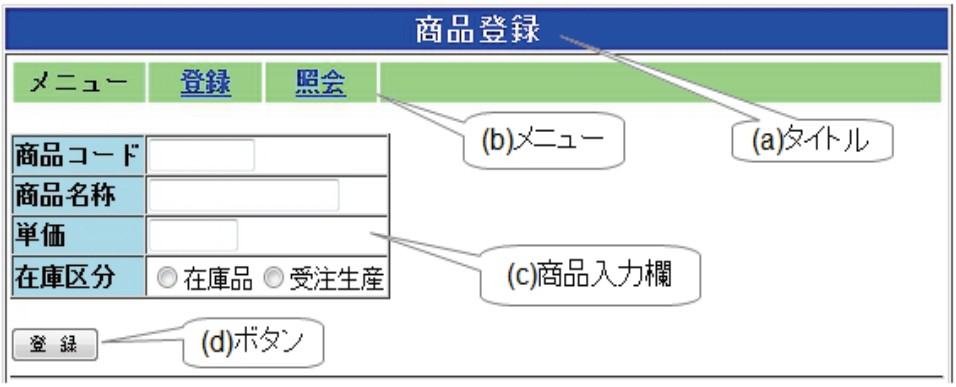

図 11. 商品登録画面の例

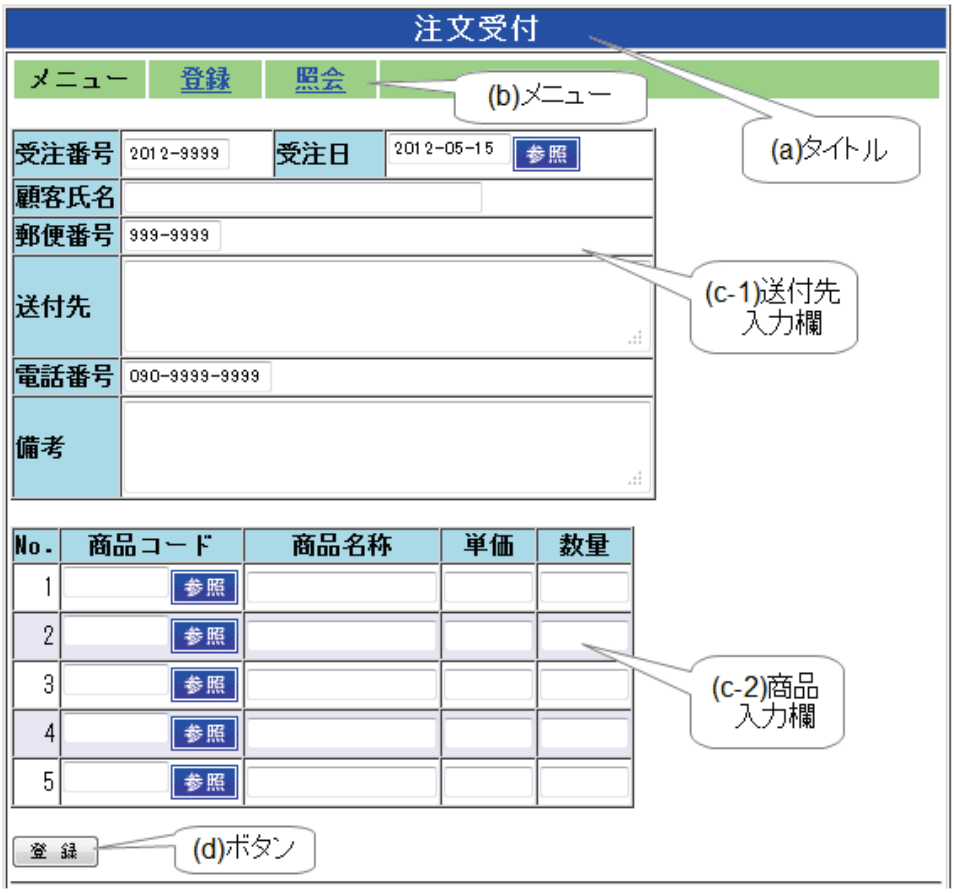

図 12. 注文受付画面の例

(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 13 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．

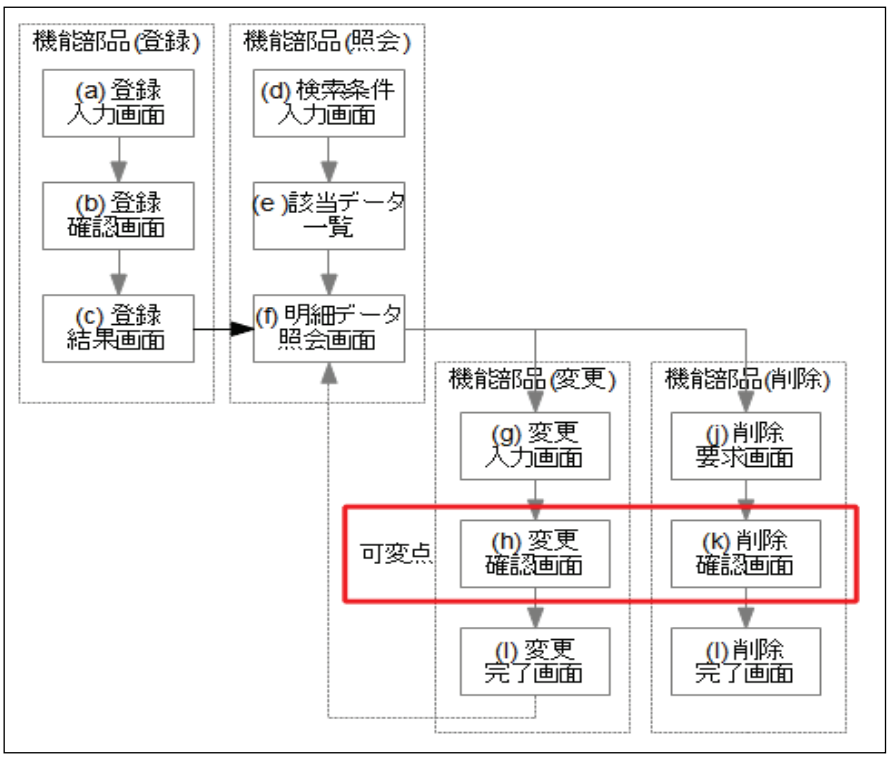

図 13. 画面遷移の共通化

4.5.2. 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 14 に利用者が画面に入力したデータをデータベース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毎に分解し，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等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．

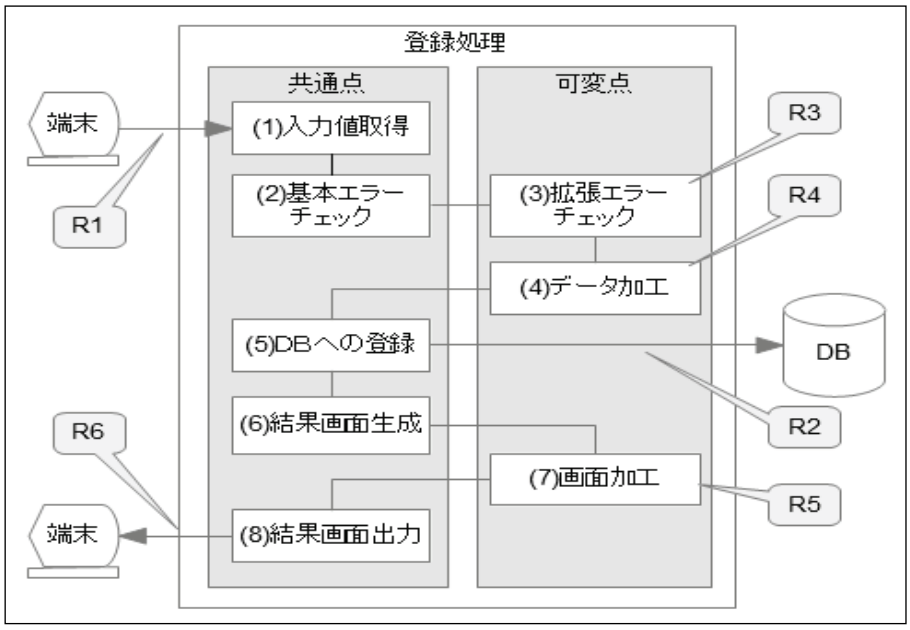

図 14. データをデータベースに登録する処理の流れ

4.5.3. 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．エンタープライズ・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毎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．

4.5.3.1. 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 15 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 11 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．その選択肢も画面毎に設定する必要があり，再利用のための作業が増加する．

```
<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 15. 受注品目,受注数量入力画面の HTML

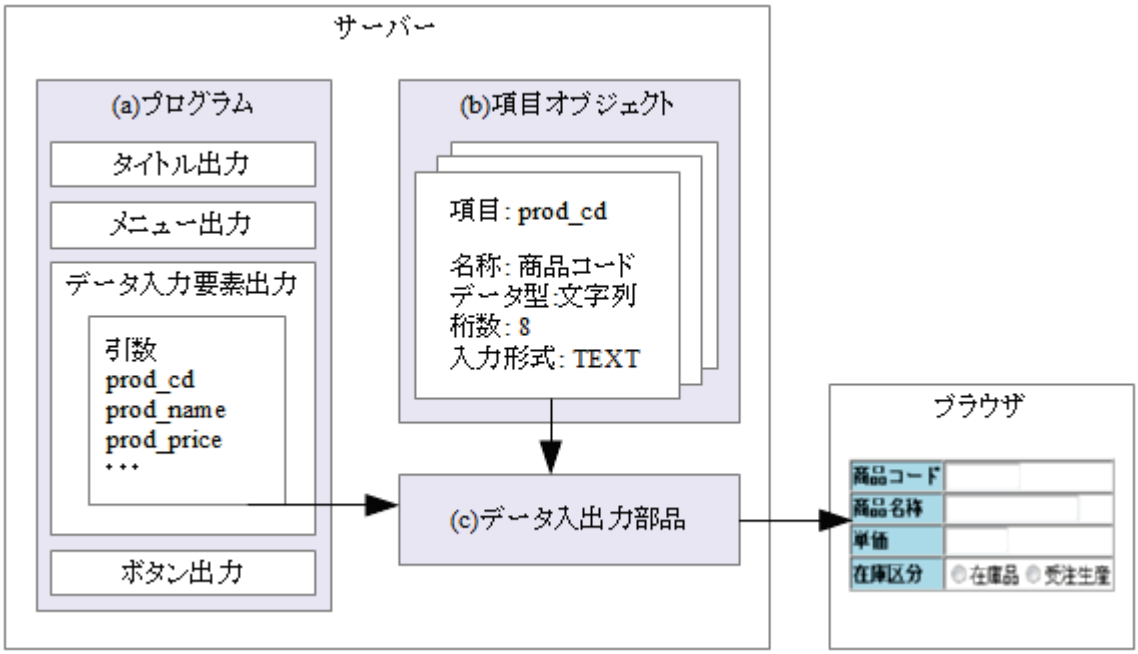

図 16. 画面部品の構成

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 16 にこの課題を解決するための仕組みを示す．商品コード等のデータ項目は，データベース設計時に項目名称や実装用の識別子，桁数等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 16 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毎に設定する必要はない. 図 16 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 17 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 12 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 13 の画面遷移や図 14 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．

図 17. 主な画面部品

4.5.3.2. 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 等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 18 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．

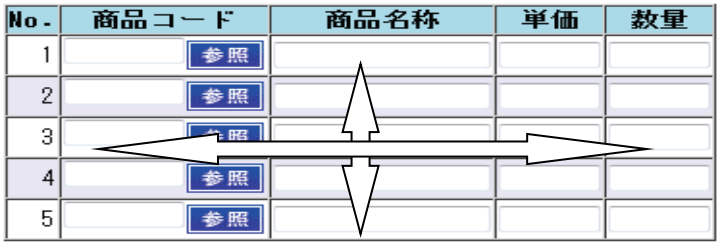

図 18. カーソル操作

4.5.3.3. 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．

4.5.4. 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．

4.6. アプリケーション開発の実践

4.6.1. 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．

4.6.2. 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．エンタープライズ・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等を明確にし，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．

4.6.3. 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．

4.7. 評価

ここでは 4.4 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．

4.7.1. 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 19 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[JUAS2011]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．

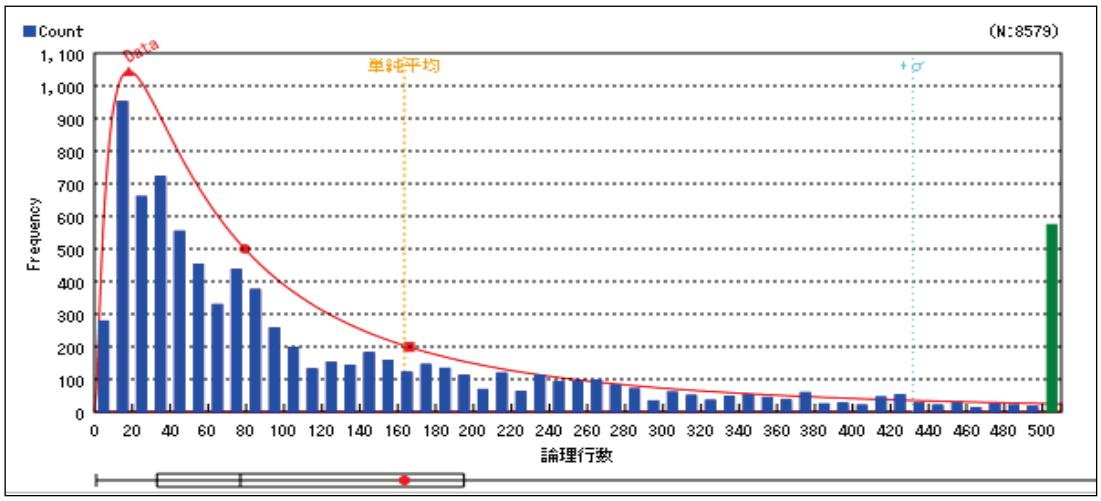

図 19. 作成したプログラムのライン数の分布

(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[IPA2010]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[2]/人時であった．文献[IPA2010]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 20 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わかる．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係にテスト工数が増加することも考えられる為，この評価結果は参考値と考えるべきである．

[2] FP: ファンクションポイント[ALBRECHT1994]

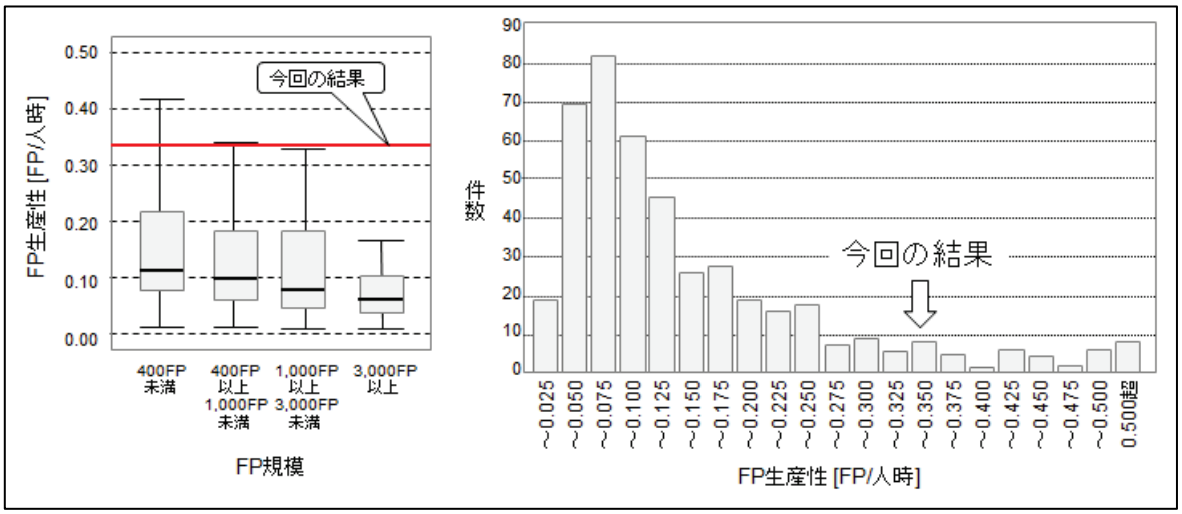

図 20. 生産性のベンチマーク結果との比較

(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 21 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.

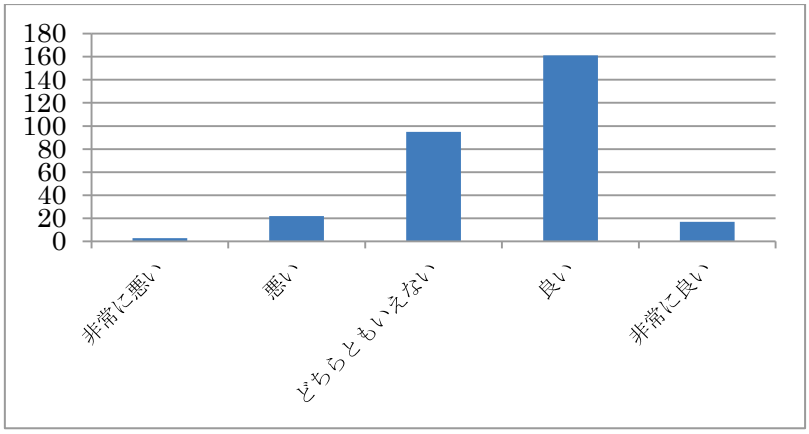

図 21. 操作性に関するアンケート結果

4.7.2. 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．

4.8. 考察

4.8.1. 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 22 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．

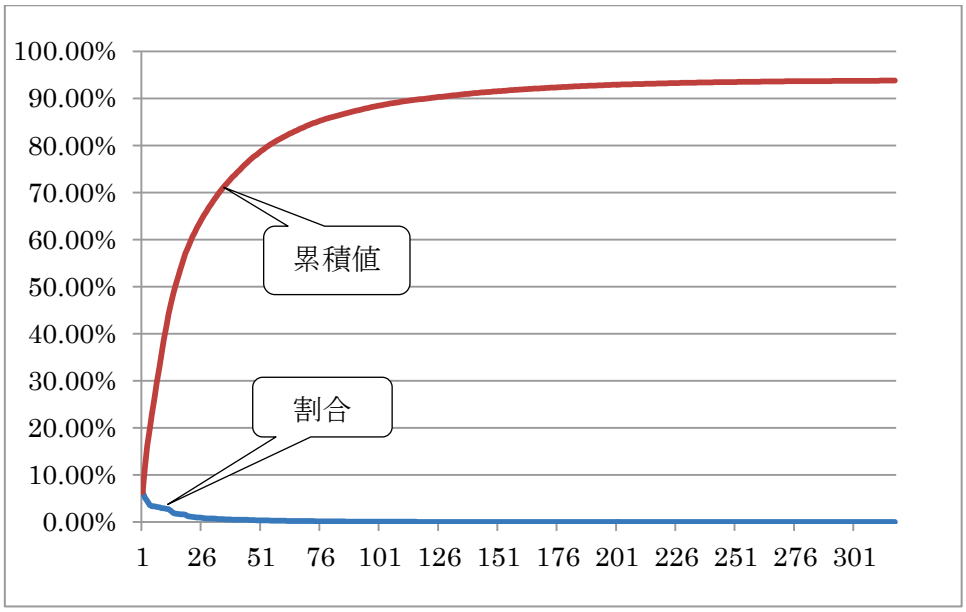

図 22. 部品の利用頻度

4.8.2. 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[SEC2009]．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．

4.9. まとめ

本章では画面および画面遷移の処理を項目オブジェクトと呼ばれる抽象化技術を利用して部品化することで広範囲に利用できるソフトウェア資産が構築でき，エンタープライズ・システムの開発においても SPL が効果的に適用できることを示した．ビジネスロジックではなく画面出力を部品化することで開発する成果物量を削減し，開発コストの削減が実現できている．また，画面部品の機能強化により利用者の満足度を向上することができている．今回開発したソフトウェア資産は住友電工以外の組織にも外販しており，開発コスト低減の効果が確認できている．なお，本章で示した事例は画面が少なく，計算ロジックが多いソフトウェアでは効果が期待できず別のアプローチが必要となる．

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．要件定義，設計，テストの領域でソフトウェア資産を構築し，再利用することで更なる開発コスト削減が今後の課題である．

第5章 管理図を利用した効率的な欠陥管理手法の評価

5.1. はじめに

近年，ソフトウェアの利用は社会インフラとなる金融・交通等のシステムや企業内システム等に広範囲に広がっており，システム障害が国民生活や企業活動に支障をきたすこともある．日本では証券取引所のシステム障害等により経済産業省が『情報システムの信頼性向上に関するガイドライン』[KEISAN2009]を発行し，情報システムの品質改善の取り組みを後押している．文献[JUAS2013b]によればシステムの品質は毎年向上し，納品後に顧客よって発見された欠陥の密度は中央値で 0.20 件/開発工数(人月)まで改善されている．また，文献[CUSUMANO2003]によればインド，米国，欧州にくらべ日本の欠陥密度が 1/10 以下と極めて高い品質を実現していることがわかる．一方で，テスト強化により高品質を実現している企業はそのためのコストを負担しており，全体としてのコスト削減が課題となっている．

住友電気工業ではシステムの品質向上と投資効率の両立をめざし，コスト増加を抑制しながら品質改善を目指してきている．品質改善でよく利用される方法はテストを重視し（例えば，テスト項目を増やすことで網羅性を上げ），より多くの欠陥を納品前に検出する方法である．しかし，この方法ではテスト設計，テスト実施のコスト増加が避けられない．そこで以下の２つの施策を実施することにした．

(a) 作込む欠陥数を削減し，修正工数の削減を図る．
(b) 作込まれた欠陥をできるだけ早期に効率よく検出する仕組みを確立し，修正工数の増加を押さえる（例えば，設計の欠陥がソースコードに残された場合，欠陥場所の特定コストが増加するため）．

これらを実現するためにはプロジェクト進行中に欠陥の作込と検出の状況を把握し，迅速に対策が実施できるようにする必要がある．組織全体でこれらの施策を実施するには欠陥管理のシステム化が不可欠であると考え，レビューやテストの結果を登録することで，欠陥の作込や検出の状況がリアルタイムに把握できるシステムを構築した．プロジェクトマネージャーが，毎週，欠陥作込・検出状況を監視し，問題発生後すぐに対策を実施することで欠陥の再発防止ができるようになることが期待される．欠陥管理システムの導入後，導入前に比べて本番稼働後３ヶ月間に検出される不具合は 1/2 以下となった．本論文では開発した欠陥管理システムの機能とその適用効果について報告する．

以降，5.2.では，本取り組みの背景について説明する．5.3.では，開発した欠陥管理システムについて述べ，5.4.では組織内での展開について紹介する．5.5.で適用結果について考察し，6.でまとめと今後の課題について述べる．

5.2. 取り組みの背景と方針

5.2.1. 組織の状況

今回対象とする組織は住友電気工業のグループ企業を対象としたエンタープライズ・システムの開発を主な業務とする組織である．システム規模は 1000～3000FP[ALBRECHT1994]のものが多い．開発者の人数は約 400 名である（但し，システム開発の負荷により協力会社の人数が増減する）．当組織では新規開発が多く，既存システムに対する保守開発に比べ約 1.5 倍となっている．技術的な特徴を次の(1)～(5)にまとめる．

(1) 標準化の徹底: システム開発で使用する OS，ミドルウェア，フレームワーク，RDBMS 等の開発環境をはじめ，各種仕様書やソースコードの書式を統一している．開発者はどのプ

ロジェクトでも同じスキルで開発できるため，プロジェクトの需要にあわせて開発者を割り当てることができる．一方，プロジェクトマネージャーから見れば，チームメンバーが毎回入れ替わるため，事前に開発者の能力を把握できないケースもある．

(2) データ中心設計手法の導入: 1995 年からデータ中心設計を導入し，全プロジェクトで ER 図(Entity Relationship Diagram)を作成している．プログラムは受注，出荷指示，出荷といった業務毎に開発し，データベースをインターフェースとして独立して動作する．そのため，プログラムは独立して動作する単位を１本として管理している．一部の共通部品を除いてプログラム間の呼び出しは行わない．

(3) 組立型システム開発の実施: 数百種類のソフトウェア部品を組み合わせてシステム開発を行うことにより文献[IPA2010]のデータと比較して２～３倍の高い生産性を実現している．再利用率が高いためプログラム１本あたりのソースコード(Java) の論理行数は中央値 77, 平均 164 ステップと小規模である．部品を利用しない場合，2KStep 程度と想定される．

(4) プロセス改善: 2003 年に CMM 成熟度レベル３を達成し, 更に上位のレベルを目指していた．しかし, CMMI[MARY2009] 成熟度レベル４で求められている管理図等を利用した統計的品質管理の経験やノウハウを持った開発者は存在しない．

(5) 開発工程: 主な開発工程は，要件定義，外部設計，データベース設計，プログラム設計，プログラム開発，統合テスト，システムテスト，本番稼働である．

5.2.2. 欠陥管理システム構築の背景

当組織では比較的高い生産性を実現していたが，品質のバラツキが大きく，利用部門から低い評価を受けるシステムもあった．そこで 2006 年から品質改善の取り組みを開始した．品質改善の代表的な方法はテストの網羅性を高め欠陥の検出率を高めるものであるが，テスト設計およびテスト実施の工数増加による開発費の増加が避けられない．一方，一般に，品質を上げれば修正コストが減り，品質向上とコスト削減は両立することはよく知られている[YAMADA1993]. そこで，コスト増加を抑制しながら品質改善を実現することを目指し, CMMI モデルを教科書として，他社の取り組みも参考にしながら具体的な施策を検討した．

5.2.3. 欠陥管理に関する先行事例

日本電気(株)は「品質会計」[HONDA2011]と呼ばれる技法により 1985 年から 1990 年代にかけ１年間で発生する出荷後バグ件数を 1/20 に削減している. この手法は「バグを作り込まない．作り込んだバグは素早く摘出する」，テスト工程では「作り込んだバグは全て摘出してから出荷する」という２つの考え方に基づいている．具体的には，「回帰型バグ予測モデル」と呼ばれるモデルを使って各工程で摘出するバグの件数を予想し，実績を管理している．品質改善のポイントは検出した欠陥を作り込み視点での分析を行うことで, 再発防止を促していることである．この手法では，レビューによる早期品質確保によりテスト段階でのバグ摘出が減少するため全体工数が増加することはないと報告されている．

日本アイビーエム(株)は「混入欠陥予測モデル」と呼ばれるモデルを利用し，品質管理を行っている[HIYAMA2011]．ソフトウェアの欠陥は混入された工程で検出されるとは限らず，欠陥の種類によって検出されている工程が異なることに着目し，欠陥の混入，検出の関係をモデル化している．各プロジェクトはプロジェクトの途中で既に検出されている欠陥をモデルの入力とし，次工程の欠陥検出目標を求める．

上記２つの事例では欠陥の検出だけではなく，欠陥の作込工程に着目しており，作込工程の

対策を実施することが品質改善とコスト増加抑制の両立のポイントであると考えられる. また，２つの事例とも過去の実績データからモデルを構築し，作込欠陥数の予測を実施している．しかし，我々の組織では欠陥を作り込む要因に関するデータが収集できていないため，これらの手法にすぐに取り組める状況ではなかった.

CMMI 成熟度レベル４では品質予測モデルと共に統計的なプロセスの監視を求めている．レベル４を達成した組織ではＱＣ７つ道具の１つである管理図(JIS Z 9021)[JIS2007]を利用しているケースが多い[FLORENCE2001][VIJAYA2010]．管理図は管理指標の種類（計数値，計量値）や管理対象のサイズの種類（一定，可変）等によりいくつかの種類が用意されている．ソフトウェアの欠陥の管理には基本的に u 管理図を利用する（図 8）．しかし，u 管理図は実績値のバラツキに関係なく，対象物の規模により管理限界が固定されるという性質があるため別の種類の管理図を利用している例もある．文献[AISO2011] では，成果物単位ではなく，１日毎にテスト結果を集計して欠陥密度の X-R 管理図を利用している．取り組みの効果としてシステムテストでの修正工数 52%削減が報告されている.

また，文献[SHIGEMOTO2006] では，大量のソースコードの品質や進捗を短期間で俯瞰的に評価する技法が示されている. 欠陥が混入されやすい箇所に関するノウハウや経験則を蓄積し，欠陥混入の疑いが強い箇所を検出するツール開発している．このツールにより大量のソースコードの中から目視調査する対象を絞り込むことにより，短期間で品質評価が行えるようにしている．欠陥混入の疑いが強い箇所を重点的にレビュー，テストするという考え方は，欠陥検出の効率化に役立つものである．しかし，実践するためにはノウハウや経験則の蓄積が課題となる.

5.2.4. 品質改善の基本方針

組織の実力と他社での取り組みを参考にし，品質改善とコスト増加抑制を両立させるために以下の方針で品質改善を進めた.

(1) 作込欠陥の削減と安定化

欠陥の作込量を削減すれば従来発生していた修正工数の削減が実現できる．また，検出プロセスから見れば欠陥密度が安定し，一定範囲に収まっていることが望ましい．管理図を使って作込欠陥密度を監視し，欠陥の作り込みが多い場合に再発防止策を実施できるようにする.

(2) 欠陥検出効率の改善

管理図やその他の統計情報を使って欠陥の検出状況を監視できるようにし，検出欠陥密度が少ないものを対象に追加レビュー，追加テストが効率的に実施できるようにする.

(3) システム化による効率化

管理図の作成や教育コストを削減するためできる限りシステム化を行い，簡単に欠陥管理が実施できるようにする.

5.2.5. 作込欠陥と検出欠陥の関係

作込欠陥と検出欠陥の関係を表 7 で説明する．この表は縦軸に成果物，横軸に欠陥の検出工程を示している．実際のプロジェクトではより多くの種類のレビューやテストの工程が実施されるが単純化している．この表の PG 設計の列はプログラム設計レビューで検出した欠陥を示しており，外部仕様書の欠陥１件，プログラム仕様書（以下，ＰＧ仕様書）の欠陥３件を検出している．検出欠陥数は合計４件である．一方，PG 仕様書に作り込まれた欠陥は PG 設計，単体テストの工程でそれぞれ 3 件，2 件検出している．この時点で検出されている作込件数は 5

件となる．工程が進むにつれて未検出の欠陥が検出されるため，作込欠陥数は時間とともに増加する．当組織では，これまでの実績データに基づいて，本番稼働後３ヶ月の時点でほぼ全ての欠陥が摘出されたと見なし，作込欠陥数を確定するルールとした．

表 7. 検出欠陥数と作込欠陥数

成果物 \ 検出工程	外部設計	ＰＧ設計	単体テスト	作込欠陥数
外部仕様書	3	**1**	0	4
ＰＧ仕様書	0	**3**	**2**	**5**
ソースコード	0	0	6	6
検出欠陥数	3	**4**	8	-

5.2.6. ツールの選定

管理図を作成するためのツールは多数存在し，表計算ソフトでも作成することができる． しかし，以下の理由により自社開発することにした．

(1) 組織全体の品質実績管理

表計算ソフト等のパソコン単体で動作するソフトウェアを利用して管理図を作成する場合，管理図および管理図作成の元になっている品質データが個人のパソコンに保存される可能性が高く，全社の基準値を作成する際，各プロジェクトからデータを収集するのに手間がかかる．また，管理図の利用状況が品質管理部門から見えにくくなり，プロジェクト管理上の問題を検出できない．

(2) 協力会社への対応

管理図による品質管理は協力会社の開発者も実施する必要があり，有償のソフトウェアを利用する場合，費用負担の問題が発生する．

(3) 作込欠陥の管理

検出欠陥の管理はレビューやテストの結果から容易に作成できる．しかし，作込欠陥の管理は表 7 に示したように複数の工程にまたがって集計する必要があり，下流工程で作成される仕様書，ソースコードとの構成を管理する必要がある．作図の前のデータ集計の効率化が必要となる．

5.3. 欠陥管理システムの構築

5.3.1. システム概要

システム開発で作成される仕様書等のドキュメントは全て自社開発の文書管理システムで管理している．欠陥管理システムではレビュー対象となる文書のデータ（規模等）が必要となるため，本文書管理システムに欠陥管理機能を追加する形で構築することにした．

本文書管理システムは Linux サーバーを利用して構築されており，文書閲覧者は一般的なブラウザを使って文書を閲覧することができる．文書作成者のＰＣには専用ツールがインストールされており，簡単に文書が作成できるようになっている．本文書管理システムの主な機能は以下の通りである．

(1) 文書作成管理: 外部仕様書, プログラム仕様書等の文書の作成計画（設計計画）が立案でき，計画に従って文書を作成することができる．また，文書の作成状況を確認する進捗管理表

が出力できる．

(2) Web 文書生成: HTML タグを記述しなくもブラウザで閲覧できる HTML 形式の文書を簡単に作成できる．また，画像やスプレッドシート等のバイナリファイル，ソースコード等が添付できる．

(3) 文書の履歴管理: 各文書はリビジョン管理されており，過去の文書も閲覧することができる．例えば，外部仕様書の場合，初版(Rev.001)を作成して利用者に提案する．通常，新たな要求が追加されるため第 2 版(Rev.002)を作成し，再度利用者に提案する．利用者と合意できれば Rev.002 は正式文書として発行される．更に利用者から新たな要求が発生した場合，Rev.003 を作成し，利用者に提案する．この段階では Rev.003 は作成中のドラフトであり，正式文書は Rev.002 であるため，一般閲覧者には Rev.002 が表示される．改訂履歴の一覧画面からはすべてのリビジョンを閲覧することができる．

(4) 関連文書管理: 文書作成（設計）時に入力として使用した文書（以降，先行文書と呼ぶ）を登録することができる．例えば，プログラム仕様書の先行文書は外部仕様書となる．ブラウザによる閲覧ではプログラム仕様書と外部仕様書の双方にリンクが設定され，簡単に閲覧できる．

(5) 文書属性管理: 文書 ID，文書タイトル，作成者，作成日，作成部署，検索用カテゴリ，備考等の文書属性が登録できる．

(6) 自動測定: 作成した文書の文字数や添付されているスプレッドシートの行数，添付されているソースコードの文字数，行数，コメント文の文字数，IF 文の数等を自動測定し，文書の属性として保管できる．

今回開発した欠陥管理システムの機能概要を図 23 に示す．欠陥管理機能の基本的な流れはレビュー者がレビューの結果を品質ＤＢ (Database) に登録し，登録されたデータをニーズに合わせてさまざまなグラフを表示することである．また，開発プロセスの異常を判断するためには基準値が必要となるが品質集計部門が毎年 1 年間の実績データをもとに基準値を登録している．本システムは Microsoft 社の Visual Basic で開発しており，文書管理機能と欠陥管理機能を含めた総ライン数は約 100KStep である．以下，グラフ化機能について詳細に述べる．

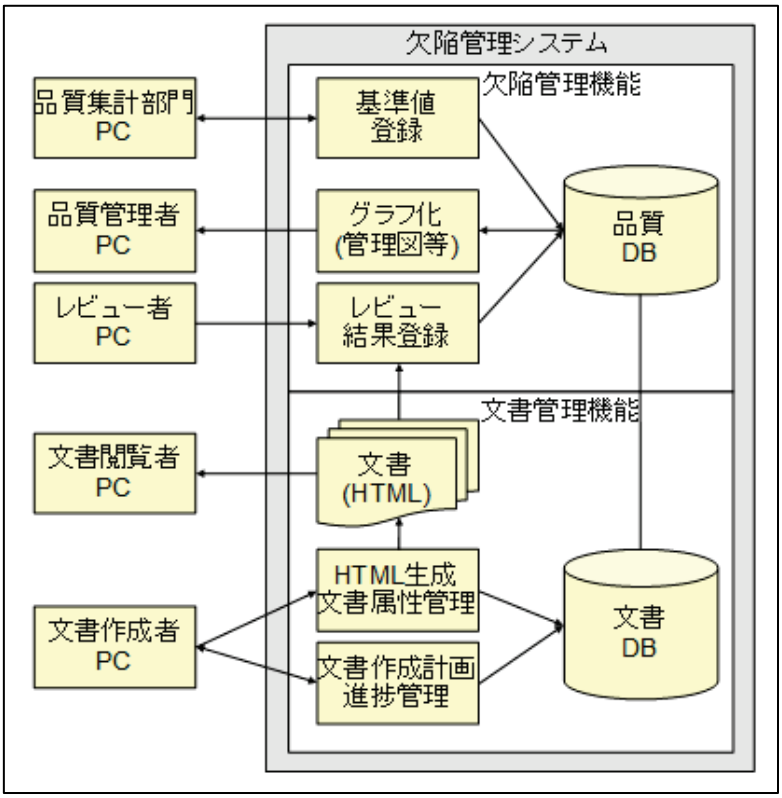

図 23. 欠陥管理システム概要

5.3.2. 欠陥検出プロセスの管理図

欠陥検出プロセスはレビューとテストに大別できる．どちらも投入された工数と検出した欠陥数を監視する．

(1) レビュー時間密度（テスト時間密度）

レビュー時間の監視を行う為に成果物規模に対するレビュー時間密度をランチャートで表示する．ランチャートはデータを発生順にプロットしたものであり，管理図との違いは中央線と管理限界が設定されていないことである．将来的には最適値を求め，管理図で管理することが望ましい．当組織では，検出された欠陥が少ない場合にレビュー時間密度を確認して再レビューするかどうかの判断に利用している．図 24 は縦軸に成果物規模当たりのレビュー時間をとり，横軸にレビュー対象をレビュー順にプロットしたものである．１件目と２件目はレビュー時間密度が小さく，レビュー時間が不足している可能性が高いと判断できる．

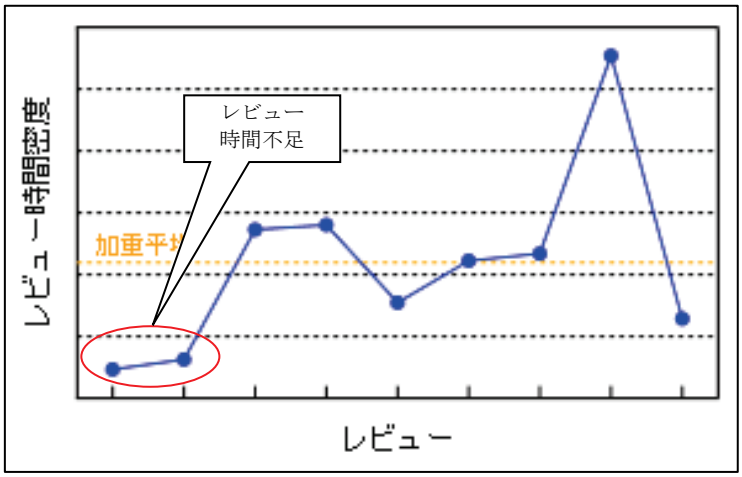

図 24. レビュー時間密度のランチャート

(2) 検出欠陥密度

検出した欠陥は図 8 に示した u 管理図を使って管理する．しかし，試行プロジェクトに適用したところ改善点が２つあることが判明した．１つはレビュー不足により欠陥の検出量が少ないと判断した場合，再レビューを実施することになるが，再レビューの結果，十分欠陥が検出されたかどうか判断できないことである．もう１つは，ソースコードに対してコードレビューと単体テストの２種類の検出プロセスが実行された場合，コードレビューでより多くの欠陥を検出すると単体テストでは通常より少ない欠陥しか検出できないという問題である．この場合，再テストを実施しても更に欠陥が検出できる可能性が少ない．

前者の問題は，成果物単位で検出された欠陥を集計して管理図を出力することで解決する．レビューの結果の合計値が管理限界の範囲に収まれば問題なしと判断できる．

後者の問題は，成果物単位で全てのレビュー，テストで検出された欠陥の合計値で管理図を出力することで解決する．例えば，単体テストの欠陥密度が低くても，コードレビューの欠陥密度と合計し，その値が管理限界の範囲に入れば問題ないと判断できる．

文献[VIJAYA2010]では u 管理図の代わりに X 管理図等を利用していたが，試行の結果，当組織ではソースコードの規模が中央値 77 ステップと小さいこともあり規模に応じて管理限界が変化する u 管理図の方が適切に異常値を確認できることがわかった．例えば，中心線が 10 件/KStep である場合，77 ステップのプログラムは欠陥が 1 件あるだけで 13.0 件/KStep となり，中心線を超える．更に小さいプログラムでは X 管理図の管理限界に収まらなくなる．

5.3.3. 欠陥作込プロセスの管理図

表 7 に示したように検出欠陥にはさまざまな工程で作り込まれた欠陥が含まれているため，作込欠陥を把握するには原因となった成果物毎に欠陥を振り分ける必要がある．１つの方法は欠陥発生の原因となっている成果物およびリビジョンを直接指定して入力する方法であるが，入力ミス防止と入力の負荷を軽減するため文書管理システムの構成管理機能を活用することにした．図 25 に構成管理の仕組みを示す．図中の四角形は１つの文書を示している．例えば 10 種類の業務を支援するシステムでは，外部仕様書，プログラム仕様書，プログラムがそれぞれ 10 種類作成されるが，図 25 では１つの業務に対する文書のみを表示している．外部設計では要件定義書 A を入力に対象業務毎に外部仕様書 a.1 のリビジョン Rev.001 を作成する．外部仕

様書のレビューを実施すると Rev.001 の文書は変更不可の状態となる．レビュー指摘に対応して外部仕様書を修正すると Rev.002 が作成される．プログラム設計では外部仕様書 a.1 の Rev.002 を入力としてプログラム仕様書 PG01 Rev.001 を作成する．さらにプログラム開発の工程ではプログラム仕様書を入力としてプログラム PG01 Rev.001 を作成する．このプログラムの単体テストを実施し，プログラム仕様書の欠陥が見つかればプログラム仕様書を修正し，Rev.002 を作成し，プログラムも修正し，Rev.002 を作成する．プログラム PG01 Rev.002 をレビューし，問題が外部仕様書の欠陥であることがわかれば，作込成果物の種類として”外部仕様書”を選択するだけで自動的に対象文書が “a.1 Rev.002” であることが判断できる．このような仕組みにより成果物毎の欠陥数を自動計算し，作込管理図を自動出力するようにした．

また，改善を促進するために図 26 に示すような作込欠陥密度の分布図を出力するようにした．この図は横軸が作込欠陥密度，縦軸が件数を示す．棒グラフに加えて(a) 組織基準分布：毎年，過去 1 年間の実績値を対数正規分布の近似曲線で示したもの，(b) 改善目標分布：毎年改訂される組織目標を達成するために工程毎に意図的に定めた目標分布，(c) プロジェクト実績：該当プロジェクトの実績値から求めた対数正規分布の近似曲線を示している．棒グラフだけでは該当プロジェクトの品質評価が難しいが，近似曲線を示すことで評価できるようになる．分布のピークが高く，幅が狭いほど設計・製造プロセスの品質が安定していることを示す．

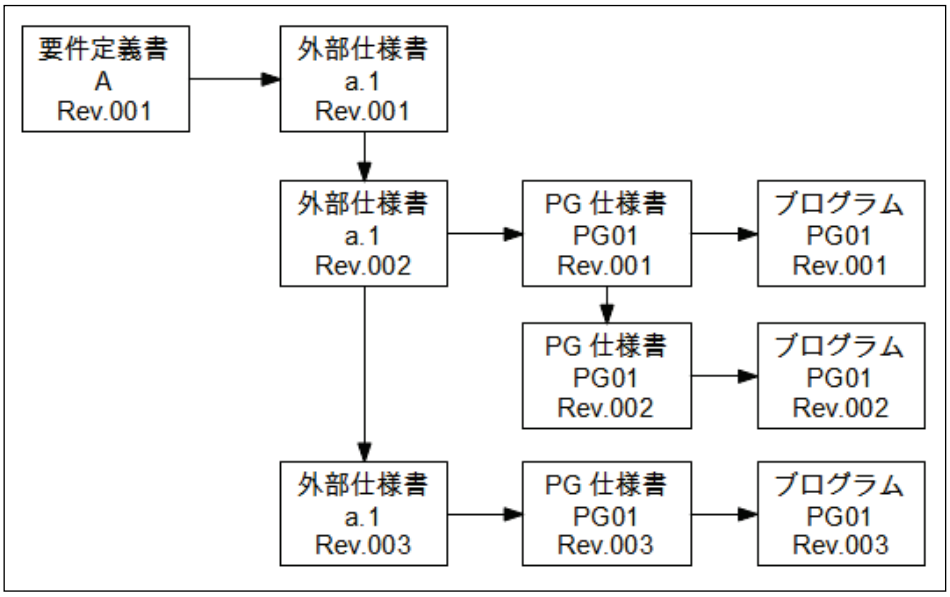

図 25. 文書の構成管理

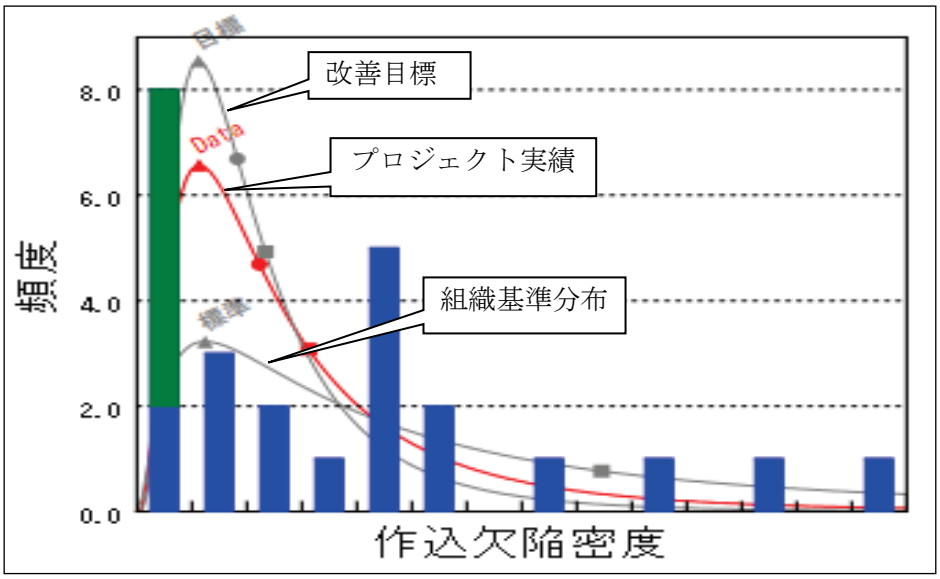

図 26. 作込欠陥密度の分布

5.3.4. IF 文と欠陥数の相関図

文献[SHIGEMOTO2006]に示されているように欠陥が残っている可能性の高い部分を収集したデータで示すことができれば低コストで効率よく欠陥を除去できる可能性が高い．これまでに示した管理図は，規模と欠陥が比例することを前提としているが，一般的にはプログラムの複雑さと作込欠陥数の相関も高いと考えられる．当社で過去のデータを分析した結果，欠陥がよく制御されたプロジェクトではソースコード中に含まれる IF 文の数と欠陥数との間に相関係数 0.7 程度の相関があることがわかった．この結果から，各プロジェクトで作成した IF 文の数と検出欠陥数の相関図を自動出力することにした．図 27 にサンプルを示す．この図は文献[WESTERN1961]に示されている回帰線の周囲に管理限界を設ける方法に基づき作成している．横軸は IF 文の数を，縦軸は欠陥数である．対象プログラムについてプロットされている．補助線は組織の基準値を示している．この範囲から下側に外れた点は複雑なプログラムにもかかわらず欠陥数が少ないことを示しており，再テストを検討する必要がある．また，上側に外れた場合，簡単なプログラムにもかかわらず欠陥が多いため，プログラム開発の指導を検討する必要がある．

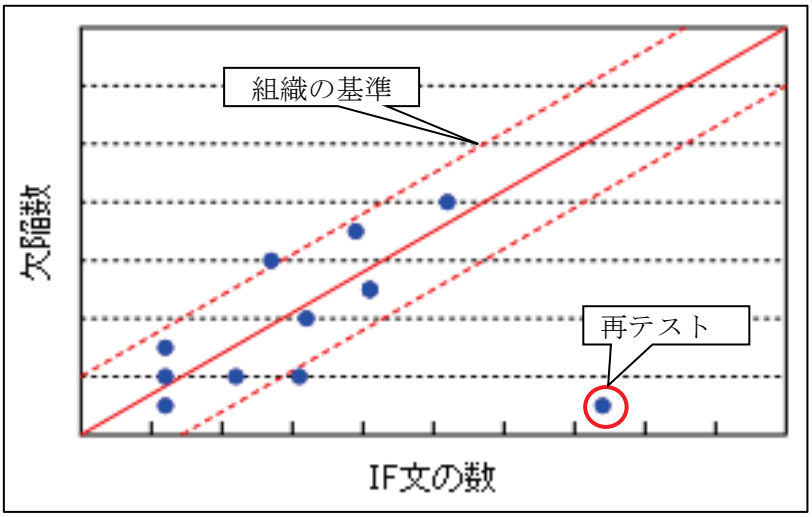

図 27. IF 文の数と欠陥の相関図

5.3.5. 欠陥フロー図

これまでに示した品質に関するグラフは開発進行中に問題を発見し，是正するためのものである．これらに加え，次回プロジェクトに向けた改善計画が立案できるよう図 28 に示す欠陥フロー図と名付けたグラフを出力する．横軸は工程を示している．実際にはより多くの工程があるが説明のため簡略化している．縦軸は欠陥数を示す．上向きの矢印は該当工程で作り込まれた欠陥の数を示し，下向き矢印は検出された欠陥の数を示す（数値は省略している）．作込欠陥数が多い工程は次プロジェクトでの改善対象となる．工程をまたぐ横線は次工程への流出欠陥を示す．自工程保証の観点から少ない程良い．

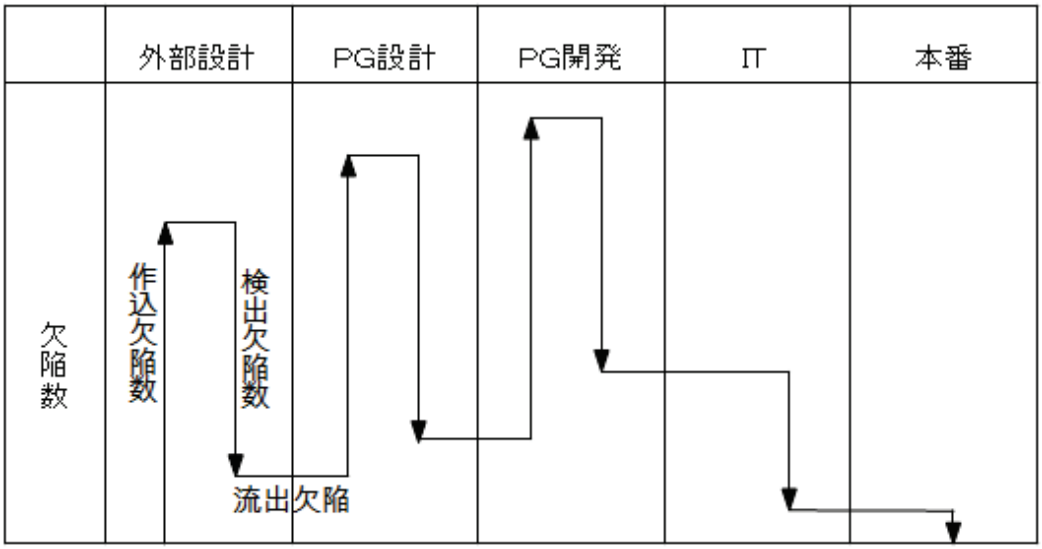

図 28. 欠陥フロー図

5.4. 欠陥管理システムの組織展開

5.4.1. システム導入教育

開発した欠陥管理システムを組織全体で活用できるよう，2007 年末から 2009 年 3 月にかけて 3 時間の演習付きセミナーを合計 20 回開催した．各セミナーの受講者は 5～15 名で，合計約 200 名が受講した．

5.4.2. 展開の支援と監視

多忙な開発者が使い方がわからない等の理由で欠陥管理システムを活用しない事態を防ぐため，問合せ窓口を設置し，すぐに不明点を解消できる体制を整えた．また，管理図が作成された際，自動的に窓口の担当者にメールで通知する機能をシステムに追加し，品質データの入力方法等に間違いがないか確認するようにした．使用方法に問題があれば窓口から開発者に連絡し，積極的に活用方法の指導を行った．

5.4.3. プロジェクト完了報告会の改善

当組織では，従来から本番稼働後 3 ヶ月のタイミングでプロジェクト完了報告を行っている．ツールの展開に合わせて，欠陥フロー図および管理図を使って品質管理の報告を行うように変更した．

5.5. 成果

5.5.1. 品質改善の実績

図 29 は年度別に本番稼働後 3 ヶ月で検出された各プロジェクトの欠陥密度(件/KFP)を箱ひげ図で表したものである．年度別の平均値も点で示している．縦軸は 2005 年の中央値を 100 とした相対値であるが，箱部分と中央値の変化がわかりやすいように縦軸の最大値を 800 にしている．800 を超える外れ値は 2005 年度 2 件，2006 年度 1 件，2008 年度 1 件発生している．なお，2012 年度のデータは 2012 年 11 月まで 7 ヶ月分の途中集計となっている．

2007 年に欠陥管理システムを開発し，2 つのプロジェクトで適用評価し，運用に問題がないことを確認した．2008 年度は教育を組織全体に広めた年である．実際の適用は，教育を受けた後，新たに開始する新規開発プロジェクトとなる．結果は稼働日を基準に年度を判断しているため，欠陥管理システムを利用して 2008 年度中に稼働したシステムは少なく効果が出ていない．2009 年度のプロジェクトはほぼ全てのシステム開発で欠陥管理システムが利用されている．結果として 2009 年度から 2011 年度の中央値は 2005 年度の中央値対比で 22～46%に改善されている．また，箱の幅は 16～31%となり，ばらつきも改善されている．欠陥管理システムの効果を判定するために 2005～2008 年の欠陥密度と 2009 年以降の欠陥密度に対して平均値に有意な差があるか有意水準 5%で検定[3]を行った．その結果，2009 年以降の平均値が改善していることが確認できた．

[3] 分布が対数正規分布であることが予想できたため，まずコルモゴロフ・スミノフ検定で2つの集合の log 値が正規分布であることを確認した．次に F 検定で等分散であることを確認し，両側検定の t 検定で有意な差があることを確認した．

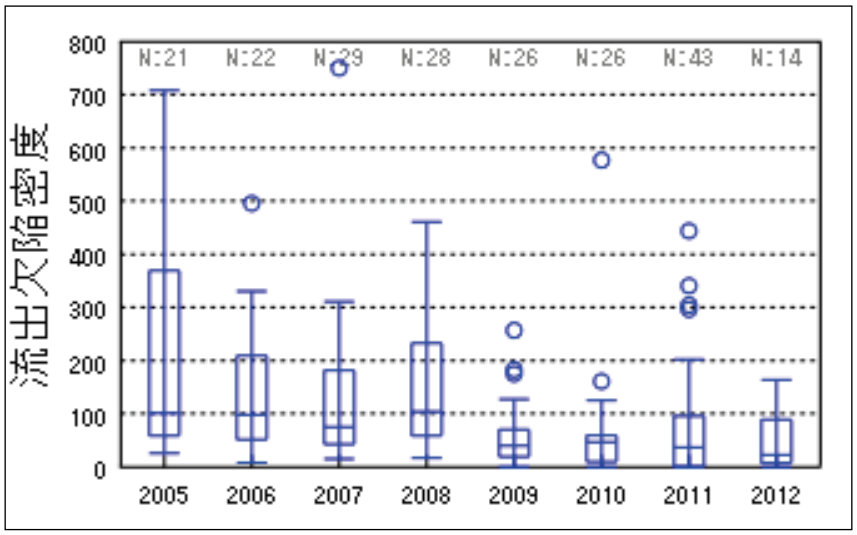

図 29. 年度別 流出欠陥密度の箱ひげ図

5.5.2. 欠陥検出プロセスの実績

(1) プログラム設計 ピアレビュー

図 30 は 2007～2012 年に欠陥管理システムを利用したプログラム設計ピアレビューの結果を示したものである．縦軸は，品質改善の効果が現れた 2009 年のプログラム仕様書毎の初回のレビューの検出欠陥密度の平均値を 100 とした相対値で，初回レビューのみの検出欠陥密度（加重平均）と再レビューの結果を合計した平均検出欠陥密度をプロットしている．更に再レビューを実施したプログラム仕様書の割合を再レビュー率(%)として示している．全体的に再レビューにより初回の 2 割程度の欠陥が検出できている．2010 年は再レビュー率が低下しており，再レビューによる欠陥検出数も減少している．残った欠陥は後述する単体テストで検出されたと考えられる．また，2008 年の初回および再レビュー後の検出欠陥密度が低いのはレビュー能力の低さによるものであると考えられる．

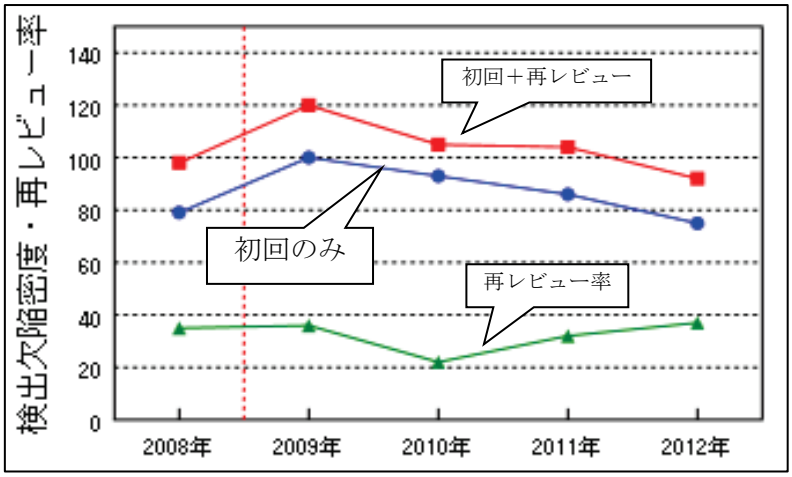

図 30. ＰＧ設計レビューの検出欠陥密度

表 8 はピアレビューの効率を示したものである．回数は，同一仕様書に対する何回目のレビューかを示している．2 回目のデータに着目すると 41%の仕様書が 2 回目のレビューを実施して

おり，合計 805 件の欠陥を検出している．検出有率はレビューで 1 件以上の欠陥を検出できた割合であり，43%となっている．検出効率は 1 回目の検出効率(件/時間)を 100 とした相対値で示しており，126 となっている．1 回目に比べて 26%高い効率で欠陥が検出できている．工数比は 1 回目の総レビュー時間を 100 とした相対値で 23%の工数を使っている．2〜5 回目の検出効率は 1 回目に比べて高く，欠陥が残っている可能性の高い仕様書を選択的にレビューできていると考えられる．また，2〜5 回目の工数合計は 1 回目の 34%であった．

表 8. ピアレビューの効率(2012 年)

回数	実施回数	実施率	検出数	検出有率	検出効率	工数比
1	1571	100%	3738	62%	100	100
2	650	41%	805	43%	126	23
3	283	18%	296	35%	136	7
4	119	8%	126	41%	168	3
5	52	3%	60	46%	251	1

(2) プログラム開発 単体テスト

図 31 は図 30 と同じ書式で 2008〜2012 年に実施した単体テストの結果を示している．2009, 2012 年の初回テストの検出密度は前年よりも 2 割以上少なく，基準量の欠陥を摘出するために再テスト率が増加したと考えられ，欠陥管理システムの仕組みが有効に働いたことがわかる．表 9 は表 8 と同じ書式で 2012 年の再テストの効率を示したものである．プログラム仕様書のピアレビュー同様，2〜5 回目の検出効率は 1 回目に比べて高く，欠陥が残っている可能性の高いプログラムを選択的にテストできていると考えられる．また，2〜5 回目の工数合計は 1 回目の 31%であった．しかし，早期欠陥検出により統合テストの工数が減少したため，システム開発全体の工数増加にはつながっていない．

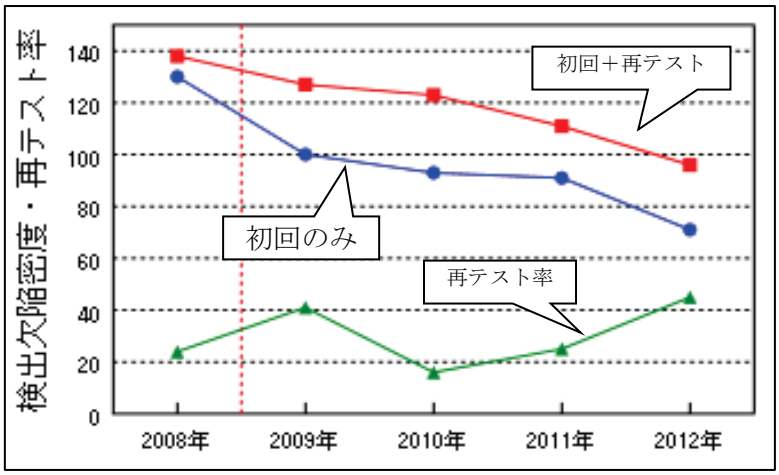

図 31. 単体テストの検出欠陥密度

表 9. 単体テストの効率(2012 年)

回数	実施回数	実施率	検出数	検出有率	検出効率	工数比
1	1497	100%	2912	62%	100	100
2	666	45%	680	47%	147	24
3	196	13%	185	47%	224	4
4	69	5%	78	58%	206	2
5	32	2%	46	69%	247	1

5.5.3. 作込欠陥密度の評価

欠陥管理システムを利用していないシステムの作込欠陥密度が測定できていない為，システム化の効果は判断できないが，欠陥管理システムを利用したプロジェクトのプログラム開発の作込欠陥密度を図 32 に示す．縦軸は平均値を 100 とした相対値で，横軸はプロジェクトを示している．また，標準偏差(σ)を使って±1σの範囲を補助線で示している．プロジェクト毎にバラツキが大きく組織全体で設計・開発プロセスが安定しているわけではない．全体の平均値(100)以下の多くのプロジェクトは作込欠陥の削減を実施しているがそのノウハウが他のプロジェクトに横展開されておらず，効果がプロジェクト内に留まっている．ノウハウの共有が進めば組織全体でさらに作込欠陥密度を低減できると考えられる．また，作込欠陥密度が高いプロジェクトでは機能的に問題がなくても保守性の観点から改善点を指摘しているものも多く，バラツキの要因となっている．2012 年のプロジェクトは平均値を下回るものが増えており，今後の改善が期待できる．

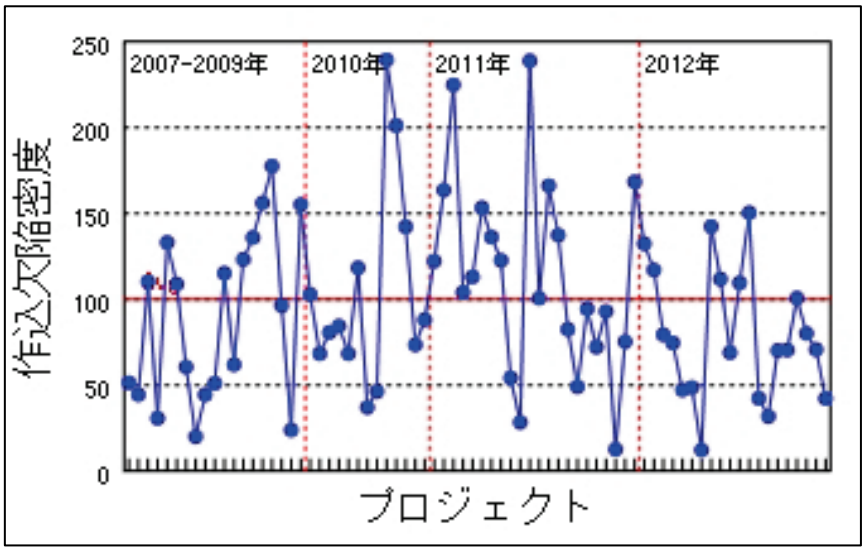

図 32. プログラム開発の作込欠陥密度

5.5.4. 品質管理プロセスの変化

(1) 改善計画の立案

図 33 は実際に作成された欠陥フロー図の一部（外部設計～単体テスト）を拡大表示したものである．プロジェクトの実績値と組織の基準値が示されている．プログラム設計，プログラム開発の作込欠陥数は組織の基準値に比べ, それぞれ 21%, 10% 少ない値となっている. また,

外部設計を含めて全ての工程で流出欠陥が組織の基準値より少なく，うまく品質が制御できていることがわかる．しかし，外部設計では組織の基準値の 1.5 倍の欠陥を作り込んでおり，次回プロジェクトの改善課題となる．このようにシステム開発を開始する際，前回プロジェクトの結果を客観的に評価し，改善計画を立案することができるようになった．

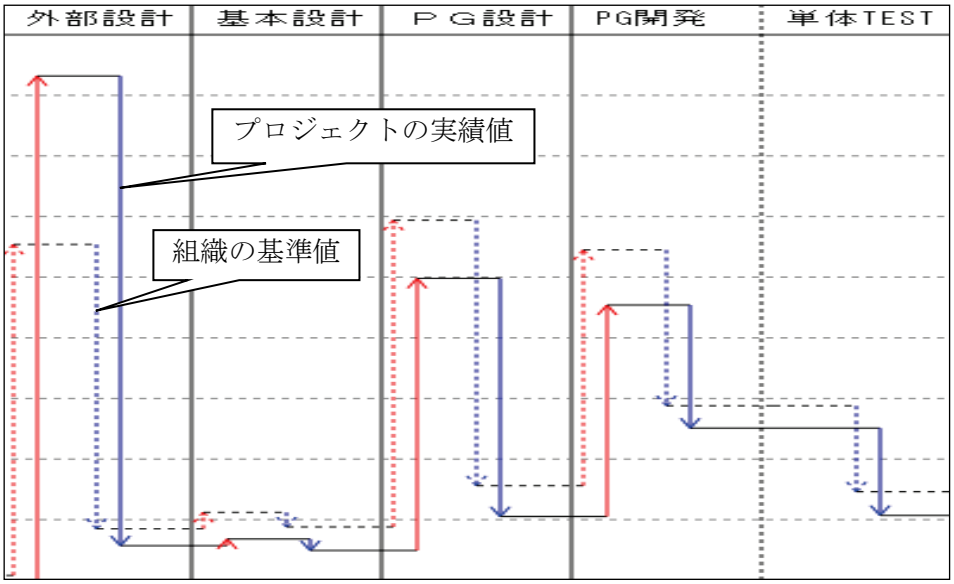

図 33. 欠陥フロー図のサンプル

(2) レビュー計画の策定

作込欠陥の削減を行うためには，設計，製造の後，できるだけ早く検証を行い，問題があれば再発防止策を実施する必要がある．しかし，フェーズの最後のほうでまとめてコードレビューや単体テストを行っているプロジェクトもあった．そこで，既存の文書管理システムの進捗管理機能にレビュー，テストの計画，実績の管理機能を追加し，欠陥の再発防止対策がより確実に実施できるようにした．この結果，レビューがより確実に実施できるようになり，リアルタイムに品質の状況が把握できるようになった．

(3) プロセスの監視と制御

図 34 は管理図の異常点に対して迅速に欠陥の再発防止策を実施してないプロジェクトの管理図のサンプルである．20 本中 5 本のプログラム（図中の丸）が異常値となっており，検出された欠陥の合計値は組織の基準値に比べて 1.5 倍であった．欠陥管理システム導入前はこのような状況であったと考えられる．一方，図 35 は，毎朝の連絡会等で再発防止策を行ったプロジェクトの管理図である．６番目に開発されたプログラム（図中の丸）は共通部品の利用法が適切でなく多くの欠陥を出しているが利用方法の指導を行うことで以降のプロクラムは品質が安定している．図 35 のように前半で異常値が発生し，後半安定するプロジェクトが多いことから，欠陥管理システムにより多くのプロジェクトで管理図による品質の制御が実施できるようになったと考えられる．

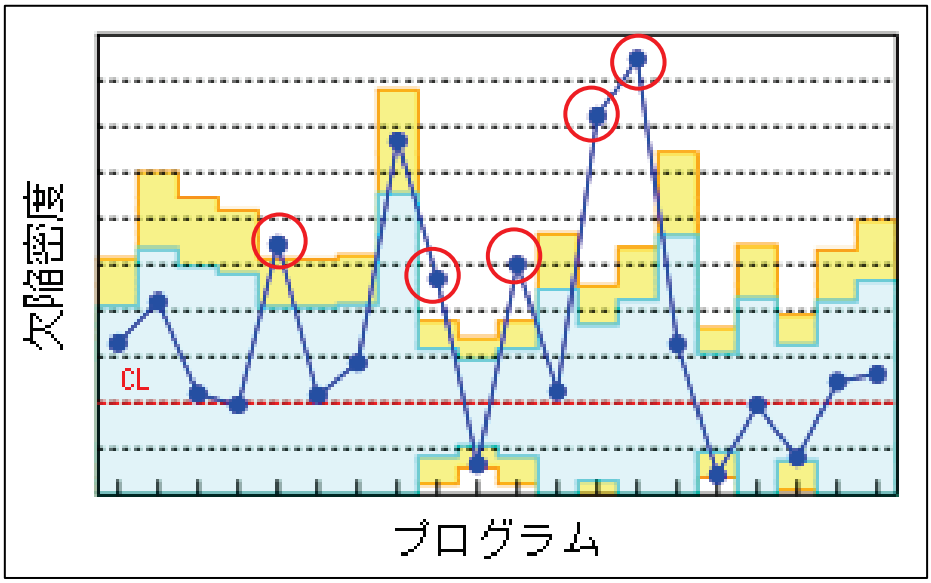

図 34. 品質制御なしプロジェクトの u 管理図

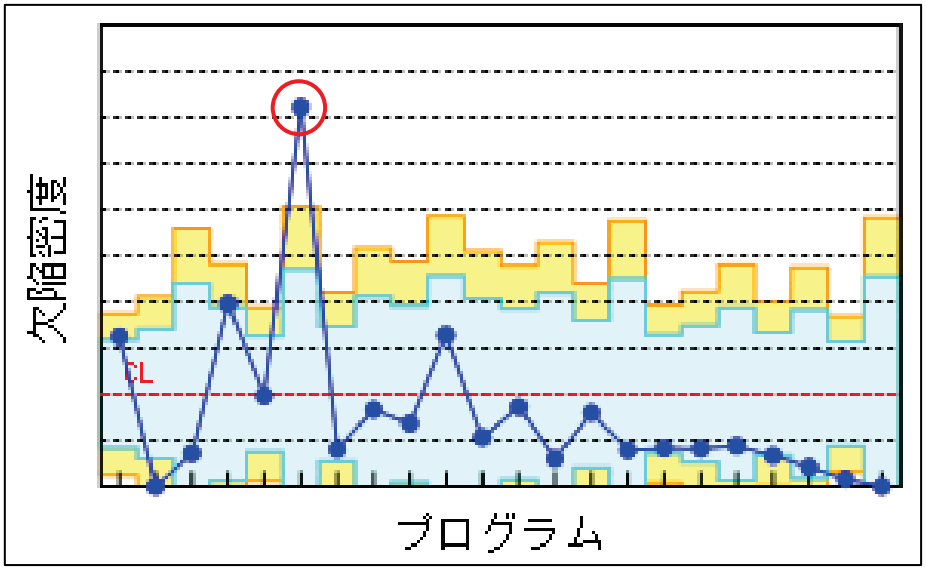

図 35. 品質制御されたプロジェクトの u 管理図

5.6. まとめ

本章ではCMMI レベル３の組織がコスト増加を抑制しながら組織全体の品質改善を実施した事例を示した．人手で行うプロセスに統計的品質管理を適用するためには設計，実装，レビュー，テストのプロセスが定義され，実践する担当者の個人差が抑制されていることが前提となる．そのためソフトウェア開発の成熟度が CMMI レベル３相当の組織でしか管理図の効果は得られない可能性が高い．管理図の作成は管理図の作成ツールがあれば難しくはない．しかし，開発現場でうまく活用するには単に異常判断だけではなく，再レビューを実施すべきかどうかの判断材料としても使える必要がある．成果物毎の欠陥数の合計値を使った u 管理図を出力することでこの要求を満たすことができ, 効率的に再レビューの実施判断ができるようになった．さらに構成管理機能を応用して簡単な入力で作込欠陥数が計算できるようにし，欠陥の発生状況がわかるようにした．早期欠陥検出や欠陥再発防止策の実施により成果物の修正工数が削減でき，結果的に増加した品質管理工数を相殺できている．以上の取り組みの結果，従来と同じ開発費で欠陥が 1/2 以下の品質の高いシステムが提供できるようになった．

今回の取り組みでは欠陥管理システムを構築し，外部設計，プログラム設計，プログラム開発の工程に対する統計的品質管理を実施してきた．しかし，外部仕様書の変更は設計上の欠陥によるものと利用部門からの追加要求によるものがあるが両者の境界が明確でなく，後者は利用部門の特性によりバラツキが大きいことがわかった．そのため組織全体として基準値を定めることができず，管理図による管理がうまく機能しなかった．今後改善する必要がある．

なお，当組織の一部のグループは品質予測モデルを構築[NAKAMURA2011]し，2011 年 6 月に CMMI レベル５を達成している. 管理図を活用した品質管理の仕組みが CMMI のモデルに整合していることが確認できている．

第6章 Scrum 適用による付加価値の向上の試行と評価

6.1. はじめに

日本情報システム・ユーザー協会(JUAS)の『IT 動向調査』[JUAS2013a]によれば経営層のＩＴ部門への期待は，システム構築や安定稼働については高い割合で応えられているものの，ビジネスモデルやビジネスプロセスの変革については十分応えられていないことが示されている．この期待に応える一般的なアプローチは要件開発プロセスの改善であるが，その一つの手段として，システムの利用部門と密度の高いコミュニケーションをとるアジャイル型開発が考えられる．

アジャイルの基本コンセプトは 2001 年に行われた Kent Beck らによるアジャイルソフトウェア開発宣言[KENT2001]で示されている．それから 10 年以上経過し，アジャイル型のシステム開発は欧米を中心に普及してきている[VERSION2001][IPA2012]．アジャイル開発手法の中でも Scrum[KEN2001] を利用した開発は多く報告されており[VERSION2001]，Scrum と XP を組み合わせて利用しているケースも多い．Scrum は中小規模のソフトウェアを 5～9 名のチームが１ヶ所に集まって開発を行うのが基本であるが，より大規模なソフトウェア開発やグローバルに分散した拠点での開発への適用可能性に関する報告も行われている[CAIVANO2011]．

また，プロセス改善の視点では Scrum と CMMI（プロセス改善モデル）を効果的に組み合わせる研究[DIAZ2009][JAKOBSEN2009] や CMMI Level 5 を達成したプロセスに Scrum を組み合わせることで生産性を向上させた事例が報告されている[SUTHERLAND2007]．

一般的に Scrum 導入の効果として (1)事業目的により整合した機能のリリース, (2)開発者のモチベーションの向上，(3)短期間のリリースサイクルによる利用者の利便性向上，(4)生産性の向上等が報告されている．

一方，住友電気工業の情報システム部門ではソフトウェア部品の再利用を進める等してソフトウェアの開発生産性を改善し，プロセス改善によりソフトウェアに含まれる欠陥を低減してきた．結果として，2011 年には CMMI Level 5 を達成した．残された課題の一つが，システムの価値の向上である．今回，高い生産性と品質を実現する既存プロセスと Scrum を組み合わせることで，より価値の高いシステムが構築できるかどうかを試行・評価した．その結果，従来のウォーターフォール型の開発に比べ，Scrum はより価値が高いシステムを構築できる要素を持っていることがわかった．本論文では試行から得られた(a)設計品質向上，(b)ソフトウェアプロセスの改善，(c)開発者のモチベーション向上，(d)教育効果の結果と考察を示す．

6.2. Scrum 導入の背景と狙い

住友電気工業の情報システム部門では第 3 章で述べたとおり主として自社で利用するエンタープライズ・システムを開発しており，システム開発の品質・納期・コストの改善も継続的に実施している．技術面ではデータ中心設計の採用により上流工程の品質を高め，自社開発フレームワーク(楽々Framework II[SIS2012]) によるソフトウェア部品の再利用により高い開発生産性を実現している．また，CMMI を活用したプロセス改善も進めており，2011 年にレベル５を達成している．品質管理では管理図(JIS Z 9021)を使ったプロセスの監視と制御を行っている．今後の課題の一つとして，利用部門の経営層や利用者にとってより価値の高いシステムを提供することがあげられている．

経営者にとってのシステム価値は投資対効果等の指標で示すことができる．エンタープライ

ズ・システムの構築では業務設計が投資対効果を決める重要な工程であるため，今回の試行とは別に改善活動を継続している．

利用者にとっての価値は業務効率が大きな要素である．当組織のエンタープライズ・システムは(a)業務活動を記録する，(b)担当者へ業務上の指示を伝達する，(c)業務上の意思決定に必要な情報を提供する，(d)業務管理に必要な情報を集計する機能を持つ．(a),(b)については，操作性や理解性が業務効率に関係する．(c),(d)については人の判断が伴うため担当者の経験や知識によって判断に必要な情報が異なることも多く，より多くの担当者が正しい意思決定を行える情報提供がシステムの価値となる．利用者のニーズとシステムが提供している機能のギャップは改善要望として現れてくるため，改善要望の少ないシステムがより価値の高いシステムといえる．

しかし，当組織では外部仕様書を利用部門と合意し，合意された外部仕様書通りにシステムを開発することが基本的な考え方になっており，外部仕様書確定後にシステム開発に参加する開発者はより価値の高い機能を提供しようというモチベーションは低い．また，一部の外部設計担当者は利用部門と合意することに気を取られ多様な利用者の考慮が不十分であることもある．今回の試行では Scrum が設計プロセスやモチベーションの観点で価値向上の要素をもっているかどうか評価する．主な評価項目は以下の通りである．

(a) 設計プロセスの改善効果
(b) 開発者の価値向上に対するモチベーション向上
(c) 設計プロセスの教育効果の有無

また，アジャイル開発に対する不安要素である下記の点も評価する．

(d) リリース時に含まれる欠陥の増減

6.3. Scrum 試行の準備

(1) パイロットシステムの選定

パイロットシステムは失敗するリスクも考え，情報システム部門内部で利用するシステムとし，当時，開発が計画されていたタスク管理システムを対象とした．タスク管理システムはエンドユーザーからの問合せ，開発・保守部隊への技術支援，計画的な技術調査，改善活動等のタスクを管理するシステムであり，従来システム化されていなかった業務の効率化を目的としている．

(2) 試行体制

スクラムガイドによれば開発チームは５～９名とされている．開発チームは標準的な７名とした．そのうちプロジェクトマネージャーの経験のある１名がスクラムマスターを兼任した．なお，開発チームには新人が２名含まれており，開発生産性は組織の平均値よりも低い状態であった．また，別の２名は時短勤務者であり会議開催の時間帯の制約があった．開発チームとは別に情報システム部門からプロダクトオーナー１名が参加し，さらに Scrum の評価，ツール提供等の支援を行う改善推進部門の担当者が１名参加した．試行期間は３ヶ月に設定した．

(3) スプリント期間の設定

スプリント期間は２週間から１ヶ月とれされており，開発するソフトウェアの特性やチームの実力に合わせて設定することが求められている．プロダクトオーナーの視点ではスプリント期間は短い程，利用者により早いタイミングで機能が提供でき，また，開発する機能の軌道修

正がしやすくなる．一方，開発者の視点では期間が長い程，進捗を調整する余裕ができ，スプリントゴールの達成確率が増加する．今回はスプリントをなるべく多く実施できるように 2 週間に設定した．

(4) 作成する成果物作成の変更

当組織では外部仕様書，プログラム仕様書，ソースコード，統合テスト仕様書を成果物として作成している．Scrum では開発チーム全員が要求を聞き，外部仕様決定に関わるため，開発すべきプログラムの機能を理解することになる．そのため，機能の詳細を記述したプログラム仕様書の必要性は低く，作成しないこととした．外部仕様書は保守のために従来通り残すことにしたが，Scrum では開発中でも仕様変更が高い頻度で発生するため，プログラム開発後に，最新の状態に修正することとした．また，統合テスト仕様書もプログラム開発後に作成することとした．

(5) ミドルウェア・OS

ミドルウェア・OS は，組織の標準に従っている．具体的には，自社製フレームワーク(楽々 Framework II[SIS2012]), Java, Tomcat, PostgreSQL，OS は Linux を使用した．

(6) 教育

Scrum の経験者は社内にいなかったが，文献[KEN2011a]やインターネット上の資料を参考にしながら，半日の説明会を実施した．その後，全員が文献[KEN2011a]を読み，具体的な進め方を開発チームメンバーで議論した．社外の教育は受けなかった．

(7) オフィス

話がしやすいよう隣向かいの 8 席を確保し，一ヶ所で開発した．席の横と後ろに金属製のキャビネットがあり，チャート等を張ることができる環境であった．

(8) ツール

Scrum では進捗管理に残作業の工数の推移を示すバーンダウンチャートを使用するのが一般的である．図 36 にバーンダウンチャートの例を示す．縦軸にタスクの残工数，横軸に日付をとり，計画と実績をプロットしていく．実績が 0 になることを”着地”と呼ぶ．最終日に残工数が 0 になっているのが望ましい．今回の開発では残工数がわかるスプレッドシートを作成した．バーンダウンチャート自体はスプレッドシートが示す数値をもとに手書きでプロットし，キャビネットに貼り付けた．

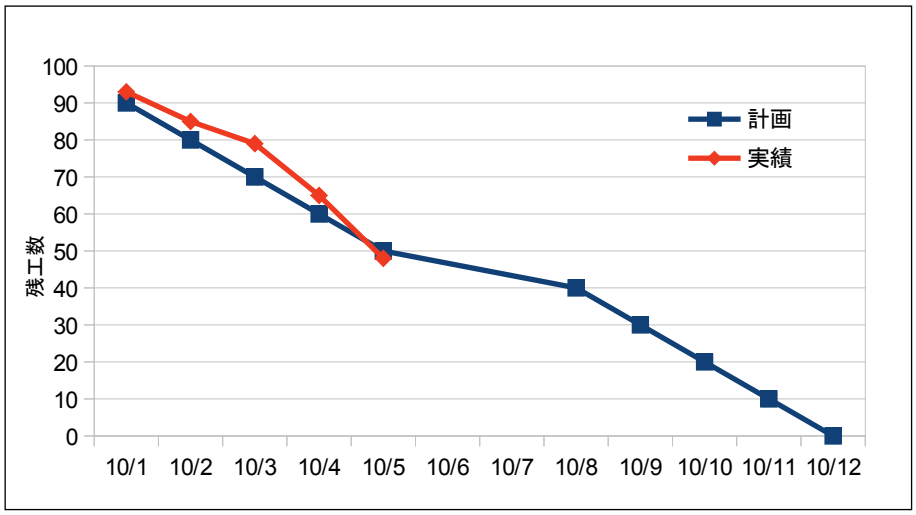

図 36. バーンダウンチャートの例

6.4. Scrum 試行

6.4.1. プロダクトバックログの作成

プロダクトバックログはプロダクトオーナーが作成する．文書作成はウォーターフォール型の開発で標準として採用されている自社開発の文書管理ツールを使用した．本文書管理ツールは文書の属性が定義でき，文書一覧で優先順位，開発の状況(完了/未着手等)が把握できるようになっている．プロダクトバックログは，業務上の施策毎に１つの文書として作成した．主な記述内容は，目的，業務フロー図，各業務での利用方法，想定される機能の概要，影響が予想される機能である．画面イメージや詳細な機能の記述はなく，開発チームが検討する．開発中に出てくる利用者の要望にあわせてプロダクトバックログに新たな文書を追加したり，優先順位の見直しをしたりする．初回のスプリントは Scrum の経験者がいないこともあり，重要度が高く実装が簡単なものの優先順位を最上位に設定した．

6.4.2. スプリント計画ミーティング Part.1

スプリント計画ミーティング Part.1 の目的は開発チームが何を作るか正しく理解することであり，プロダクトオーナーが次回スプリントで開発して欲しい機能を中心にプロダクトバックログの説明を行った．優先順位に変更があった場合は，変更内容の説明も行う．

6.4.3. スプリント計画ミーティング Part.2

Part2 では，今回のスプリントで実装する機能を仮決めし，タスクを分解し，計画を策定する．最終的には２週間の期間で開発できる作業量に合わせて実装する機能を確定し，プロダクトオーナーに連絡する．スクラムガイドによれば Part2 のタイムボックスは 2 時間であるが，実際にはこの作業の前に２～３日必要であった．作業量の見積もりを行うためにはプロダクトバックログに示されている要求を外部仕様（画面イメージを含む）に変換する必要があり，この作業に時間がかかっていた．Scrum では既存システムの変更が主として想定されている[KEN2011b]ため，システム化の範囲を拡張する新機能が中心となるシステム開発では乖離が発生するものと考えられる．スプリントバックログの計画の粒度はウォーターフォール型のプログラム開発とほぼ同じ粒度であった．

6.4.4. スプリント（開発）

プログラム開発に関するプロセスで計画的に変更したものはプログラム仕様書作成の廃止であったが，ソースコードのチームメンバーによるコードレビュー，単体テストは従来通り実施した．Scrum の実践で変化した点は，単体テストの際にも外部仕様の改善の相談が行われていることである．仕様通り動作するかといった検証(Verification)だけではなく，最終利用者にとって適した仕様になっているかといった妥当性確認(Validation)の観点が含まれている点が大きな改善点であった．

6.4.5. 品質管理

通常，品質管理は管理図を使用してプロセスの異常を検出する方法を採用しており，Scrum でも同様の方法を利用しようとしていた．しかし，Scrum では要件単位の開発になるため，別の目的で新規開発した機能に今回のスプリントで機能追加するケースも多い．１つのプログラムが複数のスプリントで段階的に機能追加されることになる．そのため従来使用していたソースコードのライン数に対する欠陥数の指標が適切ではなく，プロダクトバックログ毎の開発工数に対する欠陥数を新たな欠陥密度の指標として管理図を使用することにした．過去の開発実績から工数ベースの欠陥密度の分布（プロセス実績ベースライン）を作成すると従来のライン数ベースの欠陥密度と同じように管理図による管理ができることがわかった．

図 37 に工数ベースの u 管理図の例を示す．縦軸は欠陥数/開発工数で計算される欠陥密度であり，横軸は開発順にプロダクトバックログの番号を示している．プロットした点が規模によって変化する階段状の管理限界を超えた場合，対策を検討する必要がある．

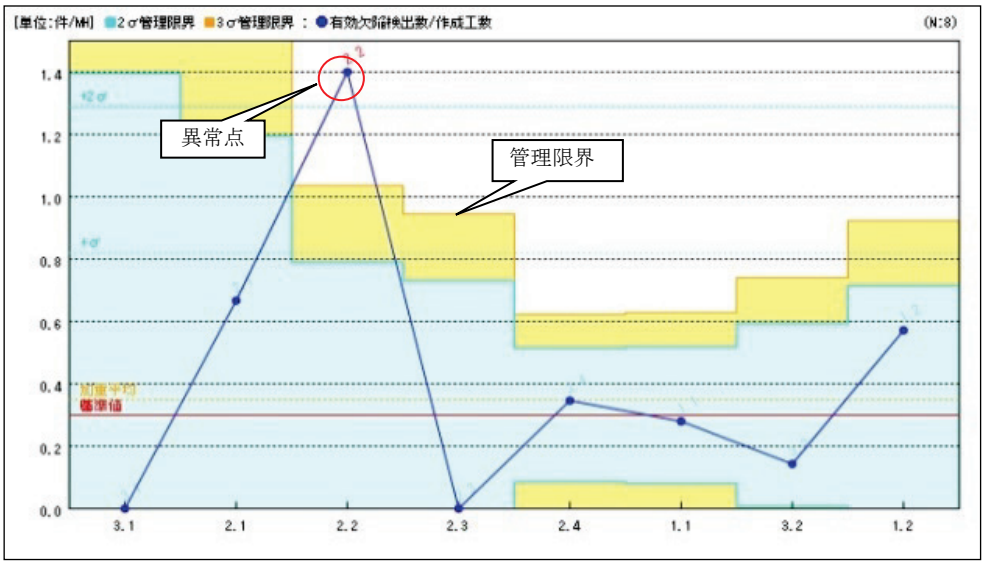

図 37. 工数ベースの u 管理図の例

6.4.6. デイリースクラム

(1) 開発状況の共有

今回のチームは勤務時間の制約があった為，午後 1 時からデイリースクラムを行った．より

正確に実績を把握する為にデイリースクラムの直前にバーンダウンチャートを作成した．タスク計画は１日単位（その日の終業時刻）で立案したが，実績集計は昼間に行ったため半日分の差異が発生する．バーンダウンチャートの実績のプロットを半日分左にプロットすることで，計画と実績の差をより直感的にわかるようにし，開発チーム全員で進捗状況の認識を共有した．

(2) タスクのコントロール

Scrum 開始当初は計画見直しの観点が弱く，効果的に運用できなかった．しかし，第３スプリントからはプロジェクト計画から遅れているタスクを抽出して印刷し，担当者に作業状況や問題点を確認することにした．作業遅れの状況がより明確にわかり，バーンダウンチャートが着地できるよう，作業の進んでいる開発者が遅れている開発者のタスクを引き取ったり，支援したりして自発的に協力できるようになった．

6.4.7. スプリントレビュー

スプリントレビューでは，作成した機能のデモを行い，プロダクトオーナーがゴール達成の判断を行った．今回の試行では，関係部署の３人の課長に対するプロジェクトの状況報告の場としても利用し，欠陥の発生状況や残予算の状況等を共有した．参加者の都合により次回スプリントレビューの日程が決まった為，実質的にスプリントレビューの日付がスプリントの期間を決定することになった．

6.4.8. スプリントレトロスペクティブ

スプリントレトロスペクティブは振り返りの場であり，継続すべき良かった点(Keep)，問題(Problem)，対策(Try) を開発チーム全員で抽出する KPT と呼ばれる手法[ALISTAIR2001]を使って実施した．開発チームの関心事の１つは最終日にバーンダウンチャートが着地することである．着地できなかった場合は対策を検討した．問題意識の共有とプロセス改善の目的の共有ができ，Scrum を効率的に実施するために必要なプロセスであることがわかった．以下，改善例を示す．

(1) 計画漏れタスクの削減

１回目のスプリントでは計画漏れのタスクが多く発生し，バーンダウンチャートがなかなか下がらないという現象が発生した．計画した作業は予定通り消化しているものの，必要なタスクの計画漏れが明らかになり，残作業の合計時間が計画通り減らなかった．２回目のスプリントではこれらのタスクを計画に盛り込むことで計画漏れタスクは大幅に減少した．

(2) 仕様の検討時間

２回目のスプリントでタスクの計画漏れは大幅に減少したものの，納期までに全てのタスクを完了させることが出来なかった．原因を分析した結果，システム価値を高めるための仕様改善の相談時間に全体の約２割の工数が使用されていることが判明した．この時間は設計品質を向上させるためのもので削減すべきものではない．アドホックに行われるため計画が難しく，持ち時間の８割でタスクの立案をすることにした．その結果，３回目のスプリントで初めて期間内に全てのタスクを完了させることができた．

6.4.9. ヒアリング調査

Scrum による開発を評価のために開発者全員に対して個別にヒアリング調査を行った．調査内容はウォーターフォール型開発との比較に関する 6 項目の質問と自由な感想である．ヒアリングの所要時間は 15 分程度であった．

6.5. Scrum の評価と考察

6.5.1. 試行結果

表 10 に今回開発したシステムの改善要望とほぼ同一メンバーで前回開発したシステムの改善要望を示す．今回の規模は前回の規模の 1.1 倍であった．改善要望の合計件数は 88 件から 14 件に減少している．改善要望を操作性，理解性，機能性の観点で分類した．操作性はより少ない操作で業務を完了させる為の要望である．理解性は，操作方法や表示されているデータ，メッセージ等の意味が正確に伝わらない点の改善要望である．機能性は必要な機能がない，必要なデータが表示されていない問題に対する改善要望である．すべてが Scrum の効果とは言い切れないが，理解性，機能性に関する改善効果があると考えられる．なお，Scrum 以外の要因としては開発者が時間と共にシステムの利用実態をより詳細に理解する，開発技術のスキル向上により使い勝手のよいインターフェースが実装できるようになる，といったことが考えられる．

表 10. 改善要望の件数

	合計	操作性	理解性	機能性
前回	88	27	27	34
今回	14	8	4	2
削減率	84%	70%	85%	94%

6.5.2. 価値の最大化

アジャイル宣言では最初に顧客価値を最優先することが示されている．ここでは，今回の試行でどのように価値が向上できたのか考察する．

(1) スプリントの効果

従来の開発では，１つの要求もしくは対象業務に対して１人のＳＥが外部仕様書を作成し，チーム内でピアレビューを実施していた．しかし，Scrum ではスプリント開始時から全開発者が参加するため，スプリントの初めに行う外部設計を全員で行うことになる（プログラマもコーディングの作業が出来ないため）．今回は，プロダクトオーナーの要求を全員で聞き，その後３～４名の２つのチームに分かれ，ホワイトボードを使って画面イメージや機能を検討した．従来，１機能に対する設計工数は１名で約 10 人時であったが，今回は 3 名で合計約 14 人時に増加している．更にプログラム開発に着手した後も仕様の見直しに約 5 人時の工数が使われていた．合計約 19 人時，1.9 倍の工数が投入されており，価値向上の要因になっていると考えられる．なお，設計開始から設計終了までの期間は従来の 1/2 であった．

また, 当組織では１つ要求に対する設計を一人で行っていた為, 通常設計案は１つであった．Scrum では前述のとおり１つ要求に対する設計を 3～4 名のチームで行う．異なる価値観を持つ複数の技術者が設計を行う為，複数の案が出てきてどちらが利用者にとって良い案かを議論するケースが増えている．解の統合，選択が行われることでより画面に表示すべきデータの抽

出，必要な機能の抽出，メッセージやデータラベルのわかりやすさ，操作性といった点で設計品質が向上していると考えられる．

しかし，スプリント毎に開発チームの能力が向上したため，前半のスプリントに比べ，後半のスプリントの方がより操作性，機能性，理解性の点で優れたユーザーインターフェースが設計・実装できている．前半のスプリントで開発した機能は後半に比べやや設計品質が劣っており，システム全体からみれば一貫性の確保が不十分であった．

(2) 段階リリースの効果

今回のシステムは従来システム化されていない業務を対象としていたため，システム企画段階ではより効果を高める為の積極的なアイデアはあまり出なかった．しかし，システムを自分自身の業務に適用し，自分達の実データが画面に表示されると具体的な要求が出てくるようになった. 要求を明確にできない状態でプロダクトバックログに 20 件の要求を書き出していたが，結果として 15 件が実装された．残り 5 件分の工数は新たな 3 件の要求の実現と開発した機能の改善に使われた．段階リリースにより有効性の低い要求が廃棄され，より価値のある機能に置き換わった．

6.5.3. Scrum によるプロセス改善効果の考察

今回の試行で特に変化があった開発プロセスを CMMI のプロセスエリアに分解して報告する．

(1) 要件開発(Requirement Development)

従来の開発では要件開発はシステム全体の効果と開発費を見積もることが重要な課題であり，個々の要件を深掘りすることは多くなかった．しかし，今回の試行では複数人で設計を行ったことから設計案の選択の際，何のためにこの機能が必要なのか，この要求の本質は何かといった議論が行われ，潜在的な要求がより多く引き出された．

(2) 技術解(Technical Solution)

従来の開発では，解の選択は個人の頭の中で行われており，解の選択が行われた記録が残ることが少なかった．Scrum では前述の通り，解の選択が頻繁に行われるようになり，設計プロセスの品質が向上した．

(3) 決定分析と解決(Decision Analysis and Resolution)

決定分析と解決は重要な決定（案の選択）を行うプロセスである．事前に評価項目とその重み，評価方法等を設定し，案の選択を行う．当組織ではこのような決定プロセスの記録があまり残っていない状況であった．今回の試行ではこのプロセスを完全に実施する機会はなかったが，案の選択プロセスが頻繁に実施されるようになったため，制度化，定着化が比較的容易に行えると考えられる．

(4) 妥当性確認(Validation)

従来の開発では単体テスト，統合テストは仕様に対する不整合を確認する検証(Verification)の観点のみであったが，今回の開発では単体テスト実施時も作成したプログラムが利用者にとって使い勝手がよいものかといった妥当性確認の観点が入っている．要求者の声を直接聞き，仕様検討の経緯を知っている開発者がプログラムを作成しているためあらゆるフェーズで妥当

性確認ができるようになっている．

(5) プロジェクト計画(Project Planning)

Scrum の開発チームはバーンダウンチャートの着地が１つの目標になっており，計画精度向上の動機づけが行われる．２週間毎に計画作業が発生するため，成熟度の向上を加速させる効果がある．

(6) プロジェクトの監視と制御(Project Monitoring and Control)

プロジェクト計画同様バーンダウンチャートの着地が１つの目標となっているため，チームの多くのメンバー計画に対する進捗の把握に関心が高く，またチーム全体で進捗の遅れを取り戻そうという意識があり，成熟度を向上させる効果がある．

(7) プロセスと成果物の品質保証(Process and Product Quality Assurance)

当組織では通常，プロジェクト管理グループによるプロセスと成果物の評価がプロジェクト毎に月１回の頻度で行われている．今回プロジェクトは試行ということもありこの評価は行わなかった．今後実施する場合，以下の点が課題となる．(a) スプリントの期間が２週間と短いため評価のタイミングが難しい．(b) レトロスペクティブで毎回プロセスが改善されるため，定められた手順に従ってシステム開発が行われているかの評価が難しくなる．

6.5.4. モチベーション

(1) ヒアリング調査結果

Scrum を実践した７名の開発者に対して行ったヒアリング調査の結果を図 38 に示す．６つの質問に５段階で答えてもらい，平均値をプロットしている．いずれの質問に対しても Scrum の方がウォーターフォール型より良い回答が得られた．また，ヒアリングでは，”一人で悩んでいるよりも気軽に相談できるのがよい”，”従来担当できなかった外部設計に参加できたのがよい” といった感想が得られた．

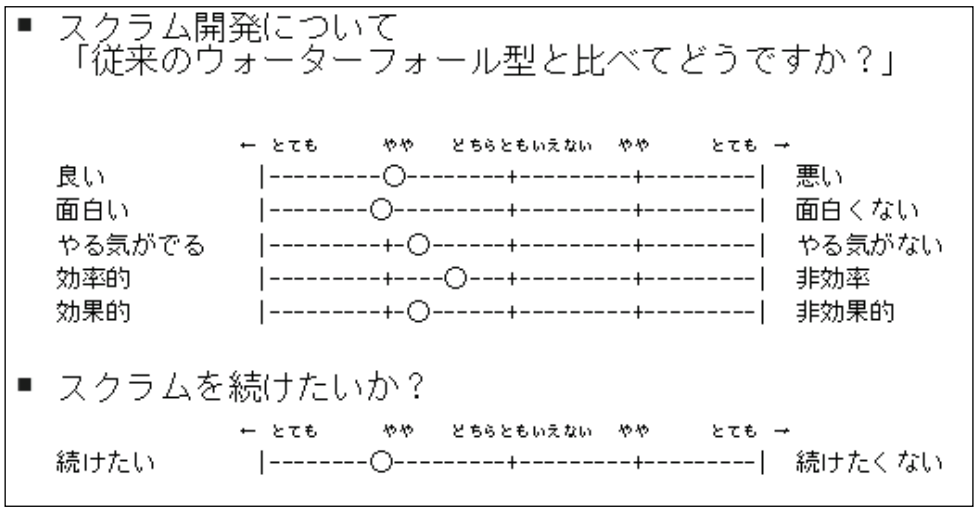

図 38. Scrum 実践者へのヒアリング結果

外部からの観察では，(a)バーンダウンチャートを着地させる，(b)約束した機能をより良い機能で提供するといったチームメンバー全員で共有した目標がチームワークを高め，モチベーションアップに寄与しているように感じられた．また，ウォーターフォール型の開発では初期に進捗が遅れると，プロジェクト終了まで進捗遅れのプレッシャーが続くが，Scrum ではスプリントが終わる度に再計画となり，こういったストレスから解放される．ヒアリングでは ”休みが取りやすかった” といった意見があった．

(2)楽しさに関する考察

大林伸安氏の著書”仕事が楽しくなる！２５のルール”[OOBAYSHI2009]に仕事が楽しくなる以下の５つのキーワードを使って楽しさに関する考察を行う．５つのキーワードと Scrum の要素を対応させてみると以下の様になり，Scrum 自体が楽しくなる要素を含んでいることがわかる．

(a) 「ありがとう」と言ってもらえる
・スプリントレビューで自分が開発したものの評価
を直接聞くことができる
(b) 「なぜ，なんのために」かがわかっている
・プロダクトオーナーから要求を直接聞き，理解する
・レトロスペクティブで問題を共有し，プロセスを改善する
(c) 「ゴール」が見える
・開発する機能（スプリントゴール）をチーム全員で決める
・バーンダウンチャートでゴールが見える化される
(d) 「昨日より今日が」が前進している
・レトロスペクティブにより継続的に改善活動が行われる
・今まで担当していなかった新人が外部設計に参画
(e) 「おめでとう」が言い合える
・バーンダウンチャート着地の喜びを共有

6.5.5. 教育効果

(1) 要件開発･外部設計の能力

ウォーターフォール型の開発では若い開発者はプログラム開発フェーズからプロジェクトに参加することが多く，外部設計の場を体験する機会が少ない．一方，Scrum では全員が設計に関与する．新人も積極的に外部設計に参加していた．スプリントを繰り返すことで基本スキルが習得できると考えられる．

(2) プロジェクト管理能力

今回の試行では，スプリント計画策定時に計画の達成可能性がより強く意識され，スプリントを繰り返すことで計画能力が向上していることが確認できた．開発中は，バーンダウンチャートが着地できるかといった観点から現状を評価し，問題があればすぐに対応策が実施されている．このような環境で経験を積むことで特に中堅社員の監視・制御能力が改善した．また，２週間のサイクルで計画, 監視, 制御が繰り返されることも成熟度を上げる要因になっている．

開発者からは「30 分の時間が貴重に感じ，限られた時間で何をすれば価値の最大化ができる

か考えるようになった．仕事の進め方が変わった.」 という意見もあり，品質・納期・コストに対するより高い意識がうかがえる．

(3) プロセス改善能力

試行の結果，短い開発期間で開発計画通りシステム開発するためには単に開発生産性の平均値を上げるだけではなく，開発プロセスを安定させ，ばらつきを小さくする必要があることがわかった．例えば２ヶ月の開発期間で計画通りシステム開発できるチームであっても２週間の単位に分解すれば生産性が平均よりも高い期間と低い期間が存在する．この差が大きければ，スプリント計画通り開発できるスプリントとそうでないスプリントが発生することになる．Scrum でスプリント期間終了までに計画した全作業を完了させる努力は，プロセスのばらつきを少なくし，安定化させる能力を身に付けさせる効果がある．

6.5.6. 品質（リリース時に含まれる欠陥）の評価

Scrum で開発した機能の欠陥でリリース以降に発生した欠陥は 12 件であった．ただし，第４スプリントで欠陥が 6 件発生しており，その他のスプリントでは 0 件または 1 件であった．第４スプリントは通常の開発で利用しない特殊な外部システムとの連携があり，インターフェース上の問題が発生した．第４スプリントを含めた欠陥密度は組織の基準値の 115%であった．第４スプリントの特殊要因を除いた 6 件では 67%であった．後者の方が今後の開発を予想する値として適切であると考えられる．

欠陥密度が従来より 33%低くなった要因は，開発者が設計段階から参加していることにあると考えられる．プログラムの欠陥は，(a)作成すべきプログラムの仕様を誤って理解する，(b)理解した仕様を誤ってコーディングする，の２つの要因に分類することができる．前者は本人が実施する単体テストで見つけることができないため，次工程に流出する可能性が高い．しかし，Scrum で要件を聞くところから開発者が参加しているため，このミスを起こす可能性が低くなると考えられる．

6.6. まとめ

今回の試行だけで Scrum の効果を評価することは危険であるが，我々の組織では CMMI レベル５に適合するウォーターフォール型の開発プロセスに比べ，より価値の高いソフトウェアを提供できる可能性が高く，開発者のモチベーションも高くなることがわかった．更に，従来のプロセス資産と Scrum を組み合わせることで，要件開発，外部設計，妥当性確認，決定分析と解決，プロジェクト計画，プロジェクトの監視と制御のプロセスが改善できることがわかった．なお，今回の試行では自社開発のフレームワーク(楽々Framework II) を使用し，既存部品の組み合わせでプログラムを開発しているため，IPA/SEC が公開している開発生産性[IPA2010]に比べ２倍以上の開発生産性を実現している．そのため，２週間のスプリントでの開発が可能になっている．一般的な開発では３～４週間のスプリントが適切と思われるが，今回の評価に対する影響は少ないと考えられる．

今後の課題は外部設計の期間の確保である．2 週間のスプリントでは外部設計に３日間程しか割り当てることが出来ない．設計では通勤途中に良いアイデアが出てくることもあり，投入工数が同じでも期間が長い方がよい設計ができる可能性が高い．より設計品質を上げる為には１つ前のスプリントで次のスプリントで機能を考え始める等の工夫が必要である．

第7章 むすび

7.1. まとめ

本研究では，高付加価値，高品質のエンタープライズ・システムを低コスト，短納期で開発できる手法を確立することを目的とし，エンタープライズ・システムにはあまり適用事例が報告されていない既存技術をエンタープライズ・システムに適用し，評価した．

まず，低コスト，短納期を実現するためにソフトウェア資産の再利用を評価した．ソフトウェア資産の構築ではビジネスロジックではなく画面部品を中心に部品化することで再利用性の高い部品を実現することができている．さらにソフトウェアプロダクトラインのプロセスを導入することで組織全体でのソフトウェア資産の展開が実現でき，コーディングの開発量は組織全体で約 92%削減することができた．また，開発生産性も文献[IPA2010]に示されているデータに比べ３～５倍となり，低コストで開発できていることを確認した．エンタープライズ・システムを開発している企業内の情報システム部門では開発ツールが開発プロジェクト間で統一できていない等の理由によりソフトウェアプロダクトラインの実践が難しいことも考えられるが今回開発したソフトウェア資産は外販されており，開発ニーズに合わせてソフトウェア資産の改善を行うこともある．複数の企業間でドメイン開発とＡＰ開発を実践し，開発コスト削減を実現している例とみることもでき，ソフトウェア資産を適切に構築できればソフトウェアプロダクトラインが有効であることを示している．

次に高品質と低コストの要件を両立させるために統計的品質管理の手法を試行した．エンタープライズ・システムに対する品質要求は社会インフラとなるソフトウェアに比べて低いものも多く，欠陥０件を目標とするのではなく，一定レベルの欠陥密度であれば許容できる．そこで管理図を使って開発プロセスの異常を検出し，問題が発生した場合のみテスト密度を上げてより多くの欠陥を検出することを目指した．試行の結果，管理図を使って再テストの必要性を判断することで効率的に欠陥が検出でき，流出する欠陥の密度が制御できることを確認した．その結果，テスト工数も削減でき開発量の削減に加えてより低コスト，短納期で高品質のソフトウェアが開発できることが確認できた．ただし，統計的品質管理のソフトウェアへの適用は設計，製造，レビュー，テスト等のプロセスが定義され，実施する担当者の個人差を減らせていることが前提になっている．そのためソフトウェア開発の成熟度が CMMI レベル３相当の組織が得られる効果であると考えられる．

最後にソフトウェアの付加価値を向上させるために上記２つの技術に Agile の手法の１つである Scrum を組み合わせたシステム開発を試行した．その結果，複数の設計者が複数の設計案を提案し，設計案の選択時に潜在的なニーズを深掘りするため，より付加価値の高い設計ができることを確認した．さらに Scrum の開発プロセスに開発者のモチベーションを向上させる仕組みがあることも確認できた．

これらの研究の結果，当初目的としていた (a) 要求水準に合った品質を確保した上で開発コストを削減できる開発手法の確立，(b) システムの付加価値向上が向上できる開発手法が確立できるようになった．

7.2. 今後の研究方針

本研究では図 39[KLAUS2005] に示すソフトウェアプロダクトラインのプロセスの内，ドメイン実現とアプリケーション実現の効果が中心であった．要求開発，設計，試験のプロセスを改善することでさらにコスト削減，品質向上が期待できる．具体的には以下の研究を推進したい．

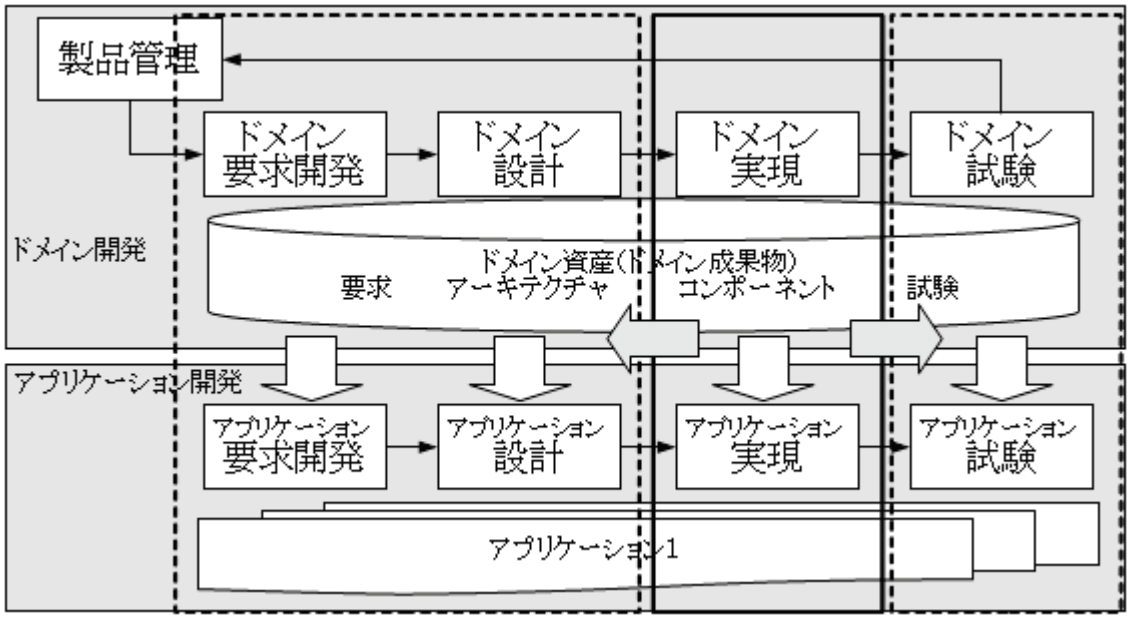

図 39. 今後の研究課題

(1) ソフトウェア設計とソフトウェア実装の統合

実装用のソフトウェア資産が構築されている場合，設計は既存資産の活用を意識して行うことになる．既存のソフトウェア資産を活用するための設計情報は予め決まっているため，設計結果を自然言語ではなく，コンピュータが理解可能な状態で格納できれば，設計情報からコードへの変換を不要にすることができる．今回の試行では高い再利用率が得られたことから，多くの機能は設計情報をそのまま実装で利用できる可能性が高い．設計ツールを開発することで設計者は人間が理解しやすい画面で設計し，設計結果をコンピュータが理解可能な形式で設計データベースに格納することができる．実際にツールを開発して効果を検証したい．

(2) ソフトウェア自動テストの効率化

今回の試行では試験に関するソフトウェア資産はテスト設計基準書といったプロセス定義書のみであった．高品質と低コストを同時に達成できる可能性がある技術として自動テストが考えられる．しかし，組み込み系ソフトに多く見られる API をもつプログラムでは JUnit といったテストツールで自動テストが比較的容易に開発できるものの，業務システムでは多く画面インターフェースとデータベース・インターフェースを持っており，JUnit のような API を呼び出すタイプの自動テストの導入には工夫が必要なる．しかし，実装用のソフトウェア部品が整備されていればテスト項目も一般化することができ，ソフトウェア部品側にテスト支援の機能を組み込むことで現在のテスト工数よりも少ない工数で自動テストが実現できる可能性がある．さらに設計データからテストに必要なテストデータを自動生成できればテスト準備の工数も削

減することができる．今後，自動テストが低コストで実現できるソフトウェア資産とテストに必要なテストデータの自動生成に関する研究を行い，更なる品質向上とコスト削減を実現したい．

(3) 長期的な課題

現在，ソフトウェア開発の多くの工程が人手で行われている．利用部門からの要求に従って設計が行われ，仕様書が作成される．また，仕様書に従ってソースコードが作成される．さらに作成した仕様書やソースコードに対してレビュー，テストといった検証作業が人手で行われている．本研究では，エンタープライズ・システムの開発においてソースコードの９割以上が再利用可能であることがわかっている．設計作業の標準化が進めば設計の一部自動化，仕様変更の際の影響分析の一部自動化が可能ではないかと思われる．近年，自動車の運転も自動化が進みつつあり，ソフトウェア開発の設計，開発，検証も自動化を進め，さらにコスト，納期，品質の改善を進めると共に，削減した時間を活用してより付加価値の高いシステム提案ができるようにしたい．

参考文献

第１章の参考文献

[JUAS2013a] 日本情報システム・ユーザー協会(JUAS),“2012 年度版企業 IT 動向調査 2013”, JUAS, 2013.

[KEISAN2009] 経済産業省, “情報システムの信頼性向上に関するガイドライン第2版,” 経済産業省, http://www.meti.go.jp/committee/materials2/downloadfiles/g90722a07j.pdf, 2009.

[JUAS2013b] 日本情報システム・ユーザー協会(JUAS), “ソフトウェアメトリックス調査 2013,” JUAS, 2013.

[BOEHM1981] Barry W Boehm, ″Software Engineering Economics″, Prentice Hall, 1981.

[MATSUBAYASHI1981] 松林 博文, クリエイティブシンキング,” ダイヤモンド社, 2003.

[CAPERS2013] Capers Jones, “SOFTWARE QUALITY IN 2013: A SURVEY OF THE STATE OF THE ART,” the keynote at the Pacific Northwest Quality conference, October 2013.

[ALBRECHT1994] A. J. Albrecht, ″Function point analysis″, Encyclopedia of Software Engineering, Vol.1, pp.518-524, 1994.

[STANDISH2013] THE STANDISH GROUP,”CHAOS MANUFEST 2013,” http://versionone.com/assets/img/files/ChaosManifesto2013.pdf, 2013, 参照 2013-11-12.

[VERSION2001] Version One, “2011 State of Agile Development Survey Results,” http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf, 参照 2012-10-01.

[IPA2012] IPA/SEC,“非ウォーターフォール型開発の普及要因と適用領域の拡大に関する調査報告書”, http://sec.ipa.go.jp/reports/20120611.html, 参照 2012-08-16.

第２章の参考文献

[WILLIAM2006] 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.

[LINDA2006] 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.

[KLAUS2005] 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″)

[MARY2009] メアリー・ベス・クリシス, マイク・コンラド, サンディ・シュラム, “CMMI 標準教本 第 2 版,” 日経 BP, 2009.

[STEPHEN2002] Stephen H. Kan, ″Metrics and Models in Software Quality Engineering,″ Addison-Wesley Professional, 2002, (邦訳)古山 恒夫, 富野 寿, ″ソフトウェア品質工学の尺度とモデル″, pp.116-121, 構造計画研究所, 2004

[JIS2007] 日本規格協会, “JIS ハンドブック 57 品質管理,” pp.1100-1122, 日本規格協会, 2007.

[VERSION2001] Version One, “2011 State of Agile Development Survey Results,” http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf, 参照 2012-10-01.

[TAKEUCHI1986] H.Takeuchi and I.Nonaka, ″The New New Product Development Game,″ Harvard Business Review, January-February, 1986.

[KEN2011a] Ken Schwaber and Jeff Sutherland, “スクラムガイド – スクラム完全ガイド:ゲームのルール,” http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20JA.pdf, 2011, 参照 2012-08-16.
[MARY2009] メアリー・ベス・クリシス, マイク・コンラド, サンディ・シュラム, “CMMI 標準教本 第 2 版,” 日経 BP, 2009.
[WATTS1991] Watts S. Humphrey, ″ソフトウェアプロセス成熟度の改善,″ 日科技連出版社, 1991.
[CMMI2013] CMMI Institute, “Maturity Profile Reports,” http://cmmiinstitute.com/assets/presentations/2013SepCMMI.pdf, Sep 2013,参照 2013-12-01

第3章の参考文献

[MARY2009] メアリー・ベス・クリシス, マイク・コンラド, サンディ・シュラム, “CMMI 標準教本 第 2 版,” 日経 BP, 2009.
[WATTS1991] Watts S. Humphrey, ″ソフトウェアプロセス成熟度の改善,″ 日科技連出版社, 1991.
[CMMI2013] CMMI Institute, “Maturity Profile Reports,” http://cmmiinstitute.com/assets/presentations/2013SepCMMI.pdf, Sep 2013,参照 2013-12-01

第4章の参考文献

[WILLIAM2006] 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.
[SEC2009] 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.
[KLAUS2005] 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″)
[YOSHIMURA2006] 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.
[YOSHIMURA2008] 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.
[ISHIDA2009] 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, Apr. 2009.
[IPA2010] 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, 2010.
[ISHIDA2007] Yuzo ISHIDA, “Challenge for the SPL Approach in Enterprise Software Development,” NRI Information Technology Report 2007, Vol.8, pp.1-11, 2007.
[NONAKA2009] 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, Apr. 2009.

[JUAS2011] 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”, 2011.
[ALBRECHT1994] A. J. Albrecht, ″Function point analysis,″ Encyclopedia of Software Engineering, Vol.1, pp.518-524, 1994.
[SEC2009] 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.

第5章の参考文献

[KEISAN2009] 経済産業省, “情報システムの信頼性向上に関するガイドライン第2版,” 経済産業省, http://www.meti.go.jp/committee/materials2/downloadfiles/g90722a07j.pdf, 2009.
[JUAS2013b] 日本情報システム・ユーザー協会, “ソフトウェアメトリックス調査 2013,” JUAS, 2013.
[CUSUMANO2003] M. Cusumano, A. MacCormack, C.F. Kemerer, W. Crandall, ″Software Development Worldwide: The State of the Practice,″ IEEE Software, Nov/Dec 2003, pp.28-34, 2003.
[ALBRECHT1994] A. J. Albrecht, ″Function point analysis,″ Encyclopedia of Software Engineering, Vol.1, pp.518-524, 1994.
[IPA2010] 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, 2010.
[MARY2009] メアリー・ベス・クリシス, マイク・コンラド, サンディ・シュラム, “CMMI 標準教本 第 2 版,” 日経 BP, 2009.
[YAMADA1993] 山田茂, 高橋宗雄, ”ソフトウェアマネジメントモデル入門-ソフトウェア品質の可視化と評価法,” 共立出版, 1993.
[HONDA2011] 誉田 直美, 山田 茂, “高品質ソフトウェア開発を実現する品質会計技法,” プロジェクトマネジメント学会誌, vol.13, no.5, pp.9-14, 2011.
[HIYAMA2011] 梶山 昌之,合田 英二, 千野 智子, “ソフトウェア開発プロジェクトの計数管理フレームワークによる定量的管理,” プロジェクトマネジメント学会誌, Vol.13, No.5, pp.3-8, 2011.
[JIS2007] 日本規格協会, “JIS ハンドブック 57 品質管理,” pp.1100-1122, 日本規格協会, 2007.
[FLORENCE2001] A. Florence, “CMM Level 4 Quantitative Analysis and Defect Prevention,” CROSSTALK The Journal of Defense Software Engineering, Feb. 2001.
[VIJAYA2010] G. Vijaya, S. Arumugam, ″Monitoring the Stability of the Processes in Defined Level Software Companies Using Control Charts with Three Sigma Limits,″ WSEAS Transactions on Information Science and Applications archive, vol.7, Issue 9, pp.1200-1209, Sep. 2010.
[AISO2011] 相磯 正司, 湯浅 耕季, 鈴木 圭一, ″ソフトウェア開発におむる統計的プロジェクト管理手法の導入と実践,″ FUJIFILM RESEARCH & DEVELOPMENT, No.56, pp.31-34, 2011.
[SHIGEMOTO2006] 繁本 将憲, “ソースコード品質の俯瞰的評価技法 : ソースコード品質評価への Quality Inspection 技法の適用,” プロジェクトマネジメント学会誌, vol.8, no.3, pp.26-31, 2006.
[WESTERN1961] Western Electric Company, ”統計的品質管理ハンドブック,” 住友電気工業株式会社訳 東京河北印刷, pp,225-226, 1961.
[NAKAMURA2011] N. Nakamura, S. Takatashi, S. Kusumoto and K. Nakatsuka, ″Approach to Introducing a Statistical Quality Control,″ IWSM-MENSURA 2011, pp.297-301, Nara, Japan, Nov. 2011.

第6章の参考文献

[JUAS2013a] 日本情報システム・ユーザー協会(JUAS),“2012 年度版企業 IT 動向調査 2013,” JUAS, 2013.

[KENT2001]Kent Beck , “Manifesto for Agile Software Development,” http://www.agilemanifesto.org, 2001, 参照 2012-10-01.

[VERSION2001] Version One, “2011 State of Agile Development Survey Results,” http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf, 参照 2012-10-01.

[IPA2012] IPA/SEC,“非ウォーターフォール型開発の普及要因と適用領域の拡大に関する調査報告書,” http://sec.ipa.go.jp/reports/20120611.html, 参照 2012-08-16.

[KEN2001] Ken Schwaber, Mike Beedle, ″Agile Software Development with Scrum,″ Prentice Hall, 2001, (和訳 ″アジャイルソフトウェア開発スクラム″).

[CAIVANO2011] D. Caivano et al., “Scrum Practices in Global Software Development:A Research Framework,“ PROFES 2011, LNCS 6759, pp.88-102, 2011.

[DIAZ2009] J.Diaz, J.Garbajosa and J.A.Calvo-Manzano, “Mapping CMMI Level 2 to Scrum Practices: An Experience Report,” Springer Berlin Heidelberg, vol. 42, no. 42, pp. 93-104, 2009.

[JAKOBSEN2009] C.R.Jakobsen and J.Sutherland, “Scrum and CMMI Going from Good to Great,” in 2009 Agile Conference, pp.333-337, 2009.

[SUTHERLAND2007] Sutherland, J., C. Jacobson, et al. “Scrum and CMMI Level5: A Magic Potion for Code Warriors!,“ Agile 2007, Washington, D.C., IEEE., 2007.

[SIS2012] 住友電工情報システム(株), “楽々Framework II 製品紹介,” http://www.sei-info.co.jp/products/products_fw_top.html, 参照 2012-10-20.

[KEN2011a] Ken Schwaber and Jeff Sutherland, “スクラムガイド – スクラム完全ガイド:ゲームのルール,” http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20JA.pdf, 2011, 参照 2012-08-16.

[KEN2011b] Ken Schwaber, “SCRUM Development Process,” http://www.jeffsutherland.org/oopsla/schwapub.pdf, p.3, 2011, 参照 2012-08-29.

[ALISTAIR2001] Alistair Cockburn, ″Agile Software Development,″ Addison-Wesley Professional,2001,(和訳“アジャイルソフトウェア開発”,P.256).

[OOBAYSHI2009] 大林伸安, “仕事が楽しくなる！25のルール,” ダイヤモンド社, 2009.

第7章の参考文献

[IPA2010] 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, 2010.

[KLAUS2005] 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″)