情報処理学会論文誌 Vol.55 No.8 1796–1806 (Aug. 2014)

Architecture Domain Matrix手法による医用分析装置統合システムソフトウェアの開発

兒玉 隆一郎[1,a)] 島袋 潤[2] 高木 由充[1] 小泉 忍[2] 田野 俊一[3]

受付日 2013年10月31日, 採録日 2014年5月17日

概要：本論文では，医用検査分野で使用される異種の分析装置を統合する医用分析装置統合システムのソフトウェア（Analyzer Integration Management Software，以下 AIMS と略す）においてソフトウェア・プロダクトラインに基づくソフトウェア開発手法を提案する．AIMS に新たな分析装置を接続するためには，分析装置に特有な管理が必要となるため，AIMS 内に広範囲な改造が発生し開発計画が難しくなる．そこで AIMS のアーキテクチャ要素をさらに検査室の業務フロー要素単位に分解して，コア資産とアプリケーションを区分け解析する Architecture Domain Matrix（ADM）手法を考案した．この手法により見積り時点でのコア資産開発コストを統制し，加えて，改造部位を業務フロー要素ごとにまとめると Work Breakdown Structure（WBS）が作成でき，アーキテクチャ要素ごとに集計すると改造量に見合った開発チーム編成に役立つので開発プロセスの生産性向上が期待できる．本開発手法を実プロジェクトに適用したところ，組込みソフトウェアにおいて 3 機種接続を 1.5 年で完了させることができ我々の過去の開発に比べ 2.5 倍の生産性を実現することができた．

キーワード：プロダクトライン，再利用技術，ソフトウェアアーキテクチャ，組込みソフトウェア開発手法

The Development of Clinical Analyzer Integrated System Software by Architecture Domain Matrix Method

Ryuichiro Kodama[1,a)] Jun Shimabukuro[2] Yoshimitsu Takagi[1]
Shinobu Koizumi[2] Shun'ichi Tano[3]

Received: October 31, 2013, Accepted: May 17, 2014

Abstract: This paper proposes the software development method based on Software Product Line approach employed for Analyzer Integration Management Software (AIMS) to systemize heterogeneous clinical analyzers. It is difficult to make a development plan to connect a new analyzer to AIMS because an analyzer requires its own particular management and various portion of AIMS software should be changed to implement the new management. To solve this problem, we devised the method called Architecture Domain Matrix (ADM) method in which each architecture component is further decomposed into clinical operation flow elements and core asset of software is extracted from those elements. This method controls development cost of core asset in a cost estimate phase and enhances productivity of software development because Work Breakdown Structure (WBS) can be generated by collecting all change specifications for each operational flow element and a development team suitable for change can be designed by adding up all changes for each architecture component. After applying this method to a real project, we integrated embedded software of three different analyzers in one year and a half and achieved 2.5 times embedded software productivity compared with our previous methods.

Keywords: product line technologies, software reuse, software architecture, embedded software development methods

[1] 株式会社日立ハイテクノロジーズ
Hitachi High-Technologies Corporation, Hitachinaka, Ibaraki 312–8504, Japan
[2] 株式会社日立製作所
Hitachi, Ltd., Yokohama, Kanagawa 244–0817, Japan
[3] 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] kodama-ryuichiro@naka.hitachi-hitec.com

1. はじめに

医療機関の検査部門では多様な分析装置が使われている．人から採取された血液などを検査する，検体検査と呼ばれる部門でも，分析装置による自動化が進んでいる．自動化は，検査の迅速化やコスト低減ならびに検査品質の向上に貢献している．

高度な自動化の例として，複数の分析装置を搬送路で結合するシステム化 [1] がある．システム化を実現するにあたって，検体データと分析装置と搬送路を集中管理するシステムソフト（Analyzer Integration Management Software，以下 AIMS）が必要となるが，その開発には多大な開発期間と開発コストがかかる．このようなソフトは組込みソフトに該当するが，代表的な組込みソフトウェアである車載ソフトや携帯電話ソフトはプログラム規模が数百万行以上といわれる [2]．AIMS も桁数において肩をならべるほど規模が大きく開発は困難をきわめる．

開発量を軽減するために，装置を接続するための共通プラットフォームが求められる．このような共通プラットフォームを更新・維持していく開発手法としてはソフトウェア・プロダクトライン（Software Product Line，以下 SPL）[3], [4] が知られている．この手法は，ソフトウェア開発のコア資産を管理することにより，コア資産を利用するアプリケーションの開発量を軽減し生産性を向上させる．

コア資産である共通プラットフォームを構築するために，複数の製品に分散したコードを分析する手法が報告されている [5]．実績ある既存ソフトウェア製品群を複数解析して，共通性・可変性を抽出している．AIMS においても過去に更新されたソフトウェア製品群が存在する．しかし，開発が年のオーダで長期にわたる，いわば少品種非量産系の製品であるため，共通性を見い出すサンプルが少ない．したがって一番最近に開発された単一の製品を基礎に共通プラットフォームを構築せざるを得ない．

また，既存ソフトウェアから共通プラットフォームを構築するにあたってはリファクタリング手法が活用できる [6]．この手法は外部仕様を変更せず再利用性や保守性を向上させるためにソースコードを改変する．しかし，この活用にあたってはどの程度改変すべきかを統制する手法を明らかにする必要がある．

本研究は計画された複数製品を開発する前に単一の既存製品のソースコードから抽出すべきコア資産を見積もる手法を述べる．見積り後は複数製品とコア資産の開発が並行し交錯するため，開発計画面で支援する必要がある．

そこで本研究は，計画された新規複数製品を，単一製品のソースコードを基礎にして開発するにあたり，以下 3 点の課題を解決する．

（課題 1）見積り時点でのコア資産コスト統制
（課題 2）全体開発量の見積り精度の向上
（課題 3）コア資産開発を含む開発計画の困難さの低減

これらに対応する施策として，まず（課題 1）については，開発予定の複数製品を可変性の範囲とみなし，複数製品間を共通化するコストと個別に開発するコストのバランスを全体計画の中で図る．

（課題 2）については，リファクタリングの解析粒度をアーキテクチャ要素とドメイン要素からなるマトリクスを導入し，ソースコードをこのマトリクス上に分類する．分類されたソースコードは機能（アーキテクチャ要素）だけでなく，その要求（ドメイン要素）からもコア資産の在り方を検討することにより，コア資産の共通性の範囲を限定する．

（課題 3）については，前述したマトリクスからコア資産開発の重みをアーキテクチャ要素単位に判定してコア資産開発者を割り当てる．複数製品が個別に実現されるにあたりコア資産開発者を含めて開発者全員で設計をレビューしテストできるツールとして，マトリクスから Work Breakdown Structure（以下，WBS）を導出する．

以上のようにマトリクスの分析を中心とした開発上流工程の強化により単一製品から SPL を導入して共通プラットフォームを構築すると同時に複数製品を開発することが可能となる．

以下，2 章で AIMS における分析装置接続の課題，3 章で上述したマトリクス分析の手法について述べる．4 章で適用結果，5 章で結果の考察，6 章で他の研究との比較を行い，7 章でまとめを述べる．

2. 検体検査の自動化と課題

2.1 検体検査業務

患者から採取した血液などの検体は試験管に採取される．検体に依頼された検査項目は医事会計を管理するホストコンピュータに入力され，関連付けされたバーコードが試験管に貼付される．試験管は検査室に運ばれ分析装置に設置されると，分析装置がバーコードを読み取り，先のホストコンピュータに問い合わせることにより検査項目を認識し，依頼された検査項目を分析できる．分析が終了し分析装置のモニタで検査技師が検査結果の適正を確認すると検査結果が報告書に印刷される．この報告書が医師に届き診断に利用される．

1 つの検体を複数の分析装置で分析する必要のある施設では複数の分析装置を搬送路で結合したシステムを導入している．検体は適切な搬送路を経由して分析装置に自動搬送される．搬送された先ではバーコードにより検査項目がホストコンピュータから取得され分析装置で分析処理が行われる．複数の分析装置から出力される検査結果は検体ごとにまとめられ，検査技師によってデータの適正を確認された後，検体ごとに報告書に印刷される．このように検体を自動搬送する搬送路と分析装置を結合するシステムが医用分析装置統合システムである．

2.2 AIMS

医用分析装置統合システムにおいては全体を統合する計算機が必要となる．この計算機は，複数の分析装置と搬送路からなるシステムを 1 つの大きな分析装置として機能させる．すなわち，ミクロには検体の搬送先分析装置を決め

その分析装置に検査項目を指示し分析後の検査結果を収集する一方で，マクロには検体の検査項目を入力して検査結果を出力する．また個々の分析装置が分析に用いる試薬の管理も行う．この計算機のソフトウェアが AIMS である．

AIMS は新たな分析装置を接続するたびに改造される．この改造は，新しい分析装置が開発されるたびに発生し，そのたびに AIMS は進化していく．しかも，上位互換性の要請から進化した AIMS は過去に接続した分析装置と接続できなければならない．進化に耐えうるプラットフォームが必要とされる．

2.3 AIMS の改造

AIMS のアーキテクチャ要素は，① UI（User Interface：画面・帳票），② DB（データベース），③ 検体管理，④ 装置通信，⑤ 分析装置，⑥ 外部通信の 6 つに分割できる．③ 検体管理は検体の検査項目が決定してから検査結果が全部出揃い報告書が出力されるまで系内の検体を追跡管理し必要に応じて分析装置と通信を行うコンポーネントである．⑥ 外部通信は前述したホストコンピュータと通信を行うコンポーネントである．

AIMS に新たな分析装置を接続するためには，①〜⑥ のすべてを変更する必要がある．新たな分析装置と通信を行うために ③ 検体管理，④ 装置通信の変更は必須である．また，分析装置が独自に持っている検査機構構成の状態をモニタするため，また，試薬を管理するために ① 画面・帳票，② データベースの改造が必要となる．さらに，この分析装置固有の機能をホストコンピュータから利用するためには ⑥ 外部通信の変更を余儀なくされる．

我々の課題は，このような変更が連鎖する環境において，目前の開発計画にある分析装置の要求に合わせて全体開発コストが統制されるような見積り手段を確立することにある．

3. ADM 手法の提案

3.1 SPL の適用

SPL のエンジニアリングは，ドメインエンジニアリング（または Core Asset Development），アプリケーションエンジニアリング（または Product Development），マネジメントの 3 活動で定義されている [7]．

この SPL の 3 活動に基づいた開発の流れを**図 1** に記す．ドメインエンジニアリングでは，将来いつごろにどのような新製品を顧客に提供すべきかといった事業戦略が入力となり，分析装置のドメインにおいて維持すべきコア資産を出力する．アプリケーションエンジニアリングでは，コア資産を有効に適用しながら事業戦略に基づき顧客満足を最大限に提供できる製品を開発する．両エンジニアリングにわたり一貫した経営資源の配置などのマネジメントが欠かせない．本論文は，ドメインエンジニアリングおよびアプリケーションエンジニアリングにまたがる開発手法を提案している．

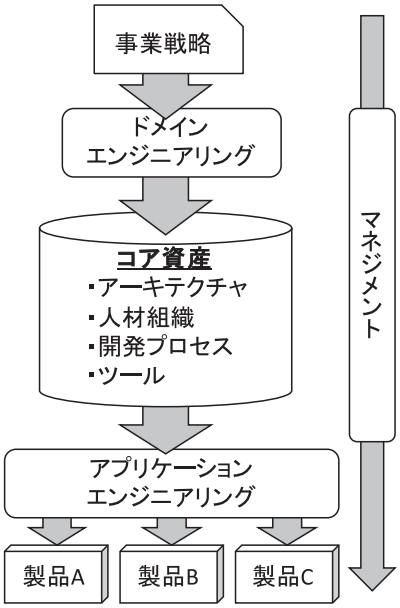


図 1 SPL の開発プロセス
Fig. 1 Development process in SPL.

SPL のアプローチには，製品とは独立にコア資産を開発する Proactive 型と 1 つの製品を雛型にしてコア資産を形成しながら次の製品を開発する Reactive 型がある [8]．本論文では少品種非量産系の製品を対象にしているため Reactive 型を採用している．少品種非量産系における Proactive 型の問題点を以下にあげる．

① コア資産の設計範囲があいまい：長期開発で少品種非量産のためコア資産の基礎となるソースコードが少なく，また直近の開発以外はソースコードが古く参考になりにくい．

② 必要以上の汎用性：参考が少ないため必要とされる以上の汎用性を求める可能性がある．

以上の理由により，少品種非量産系の AIMS 開発においてコア資産は Reactive 型で構築される．

(1) 目標とするコア資産の姿

SPL の目的は，製品シリーズにおいて開発すべき製品が N 個あった場合，個別に N 個の製品開発を行うよりも開発期間，開発コスト，品質の面で優位な開発手法を提供することにある．そのためにコア資産を設け，N 個の製品開発負担を軽減する．では，そのようなコア資産はどのような形が求められるであろうか．

ここで，A，B，C の 3 製品をコア資産なしで独立に開発することを考える．この場合 3 つの製品開発が独立に行われるので，本当は共有すべきツールや設計やコードなどが重複して作られることになる．このように個別に製品を開発するコストを個別開発コストと呼ぶことにする．次に，コア資産を形成して 3 製品を開発することを考える．する

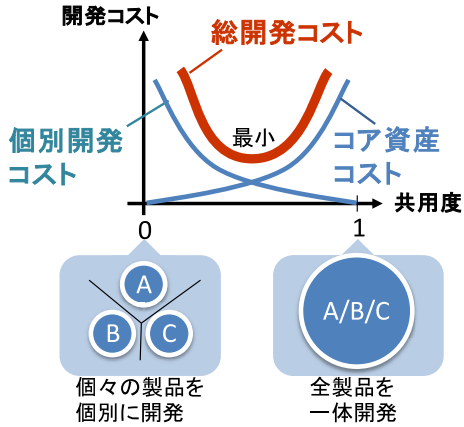


図 2 製品 A，B，C 開発のコスト見積り
Fig. 2 Development cost estimate for product A, B, and C.

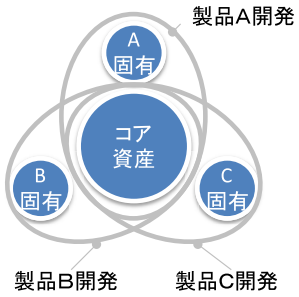


図 3 コア資産と製品の関係
Fig. 3 Relationship between core asset and products.

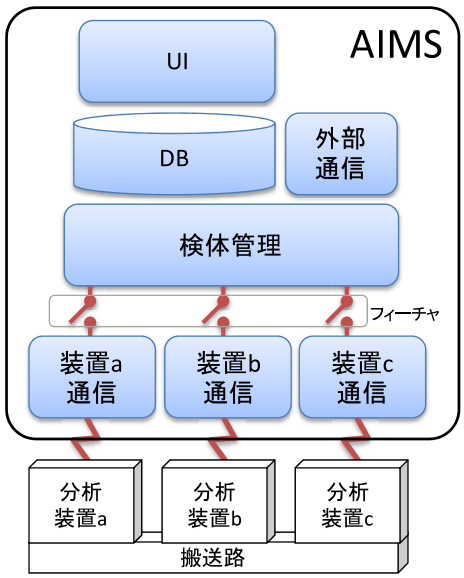


図 4 AIMS のアーキテクチャ
Fig. 4 Architecture of AIMS.

とコア資産によって重複を減少させることができる．コア資産をどんどん大きくしていくと極限において 3 製品一体のコア資産となる．3 製品のコードが共通一体となりパラメータを調整する範囲の構成管理で 3 製品が実現できるかもしれない．このパラメータのうち，ユーザ視点のシステム特性がフィーチャ [9] である．この究極のコア資産最大化によっても新たなコストが発生する．3 製品の外部仕様の差が大きい場合，各アーキテクチャ層で参照されるパラメータの数および論理的な組合せは膨大になる．たとえば，AIMS では装置通信の通信手順の違い，特殊な検査項目の有無によるデータの有無，保守操作画面の違いなどが組み合わさって製品が作られ，組合せに応じて設計が複雑となる．コア資産の開発にかかるコストをコア資産コストと呼ぶことにする．

個別開発コストとコア資産コストの関係を図 **2** に示す．3 製品のソフトウェアが共用されている割合（図中，共用度）がゼロから 1 に変化する間，個別開発コストは減少するがコア資産コストが増える．個別開発コストとコア資産コストを加えると総開発コストになり，総開発コストはコア資産比率がゼロと 1 の間の値をとるどこかで最小となる．ここが目標とするコア資産の規模となる．つまり共用度を上げてもコア資産コストが大きいようでは共用のメリットを享受できないので，バランスの検討が必要になる．

我々が望む理想形を描くならば，コア資産と製品 A，B，C 固有のアプリケーションの関係は図 **3** のようになる．コスト最小となるコア資産が製品 A，B，C のアプリケーションソフトと組み合わさって製品が開発される．このような理想の開発構成を見積り時点で統制する手法が求められる．

(2) AIMS におけるコア資産

上述の製品開発を AIMS に当てはめるため，製品 A，B，C をそれぞれ分析装置 a，b，c が接続された AIMS とする．上位互換性を求められるため，開発後の AIMS は，システム内に分析装置 a，b，c の全部または一部のいずれも接続可能にする必要がある．そのため AIMS はどの分析装置がどのような搬送経路で接続されているかをフィーチャとしてもつ．システム立ち上げ時に AIMS はシステムに接続される装置を認識して必要なソフトを立ち上げる（図 **4**）．

装置構成に依存する部分が通信にとどまれば，これで統合は終了となる．しかし，前述したように分析装置の接続は AIMS のソフト全体に影響を与える．したがって，コア資産とアプリケーションはシステム全体に散在する．この状況でコア資産を見積もれば，全体開発量の見積り精度が大きく揺らぎ，開発リスクが多大となる．全体開発量の見積り精度の向上が望まれる．

3.2 ADM 手法

本論文ではコア資産を抽出する手法として Architecture Domain Matrix 手法（以下 ADM と呼ぶ）を提案する．ADM はドメイン知識要素，ならびに統合システムのアーキテクチャ構成要素の 2 要素からなるマトリクスである（図 **5**）．

ADM のセルに対応してソースコードを割り当て，セル単位にソースコードを分析し，コア資産にするか，または，

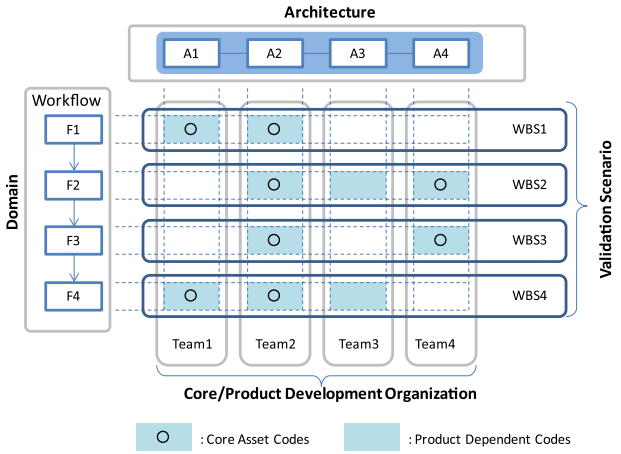


図 5 Architecture Domain Matrix
Fig. 5 Architecture Domain Matrix.

分析装置固有の改造にするかを判定し，変更量の見積りを行う．ソースコードはドメイン要素とアーキテクチャ要素の対に位置づけられるため，業務知識からくる要求の範囲，機能実現の範囲が明確になる．

ADM 手法の狙いは，見積り時点でのコア資産コストを統制し，その見積り精度を向上させることにある．そこでこの 2 要素でソースコードの改造範囲を特定して見積り精度を向上させる．開発が予定された，要求仕様が確定している複数製品において，既存製品のソースコードを前述 2 要素で囲まれた範囲で解析し，共通なコア資産にすべきか製品個別に開発すべきかを判定する．このような統制をマトリクスの全セルに対して行い，複数製品間を共通化するコストと個別に開発するコストのバランスを全体計画の中で図る．

ADM のアーキテクチャ要素の粒度は，1 人のリーダーと開発メンバからなる機能チームで取りまとめられる範囲と規定する．この規定により，ADM のアーキテクチャ要素単位（図 5 における列単位）に変更を眺めると，どのようなスキルを持った人材がどのような変更を行うべきかが明らかになる．コア資産を含んでいる場合にはコア資産開発要員をアーキテクチャ要素に割りつける．

また，ADM のドメイン要素の粒度はひとまとまりにシステムテストができる範囲と規定する．ADM のドメイン要素単位（図 5 における行単位）に変更を眺めると，ドメイン知識要素を実現するために必要な一連のアーキテクチャ要素変更が分かる．変更箇所と実現されるドメイン要素をまとめることで WBS が作成され，これにより変更仕様レビューの充実が図られる．加えて，どのアーキテクチャ要素の変更が最終テストに関連するかが明らかになる．

以上のようなコア資産開発要員の配置，およびコア資産開発者を含めたレビューの支援によりコア資産を含む開発計画の困難さを低減することができる．

本研究ではドメイン知識として業務フロー要素を用いて

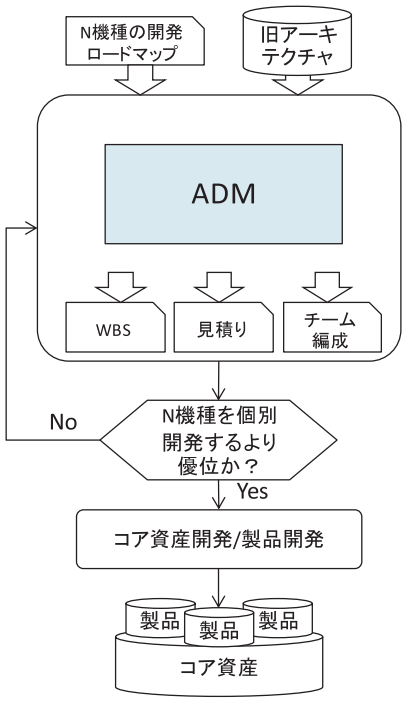


図 6 ADM 手法による開発の流れ
Fig. 6 Development workflow of ADM method.

いる．ここで業務フローとは分析装置を立ち上げ，検査実行前の準備，検査の実行，保守，分析装置の終了処理まで，1 日分の業務の流れである．報告 [10] では医用装置のドメイン知識として業務フローを活用している．医用装置では医療業務フローを自動化することに注力しているため，業務フローがソースコード解析のための安定的なドメイン知識になる．また，業務フローは独立な業務工程から構成されている．アーキテクチャ要素はシステムの基本入出力に対応した独立な層構造になっている．アーキテクチャが独立な層構造であれば，ADM の 2 要素の独立性が確保でき安定したソースコード解析ツールとなる．ADM 手法は，長期間にわたり変更の可能性が少ないアーキテクチャ構成要素とドメイン知識要素が存在するような製品であり 2 要素が独立であることを前提としている．

3.3 ADM 手法の手順

図 6 を用いて ADM 手法の手順を説明する．入力は，過去に稼働実績のある AIMS（図中旧アーキテクチャ）と開発ロードマップである．開発ロードマップにはこれから接続される N 機種と出荷時期が具体的に記述される．

N 機種それぞれについて行列セルのソースコードを分析して，N 機種個別にプログラムを作成すべきか，それとも N 機種共通のコア資産とすべきかを検討し ADM 手法を実行する．

以上の検討結果として，コストおよび開発期間が N 機種を個別開発するより優位かを判定し優位であれば開発に着手する．優位でなければ，さらに開発量を軽減する策を検

討する．策は共通範囲（コア資産）を括りだすこと，または，既存の製品プログラムを機能重複覚悟で流用することの選択が考えられる．これによりすでに述べた個別開発コストとコア資産コストとの調和を図る（図 2）．

ドメイン要素 i，アーキテクチャ要素 j における改造コスト $C_{i,j}$ は以下の式となる．

$$C_{i,j} = \min(CC_{i,j}, CD_{i,j}, CA_{i,j})$$

ここで，$CC_{i,j}$ はコア資産に仕上げるコスト，$CD_{i,j}$ はコア資産とアプリケーションを分離するコスト，$CA_{i,j}$ は純粋アプリケーションを開発するコストである．3 つのコストのいずれが低いかは一義的に決まらない．開発体制にも依存し，どれくらい深くソースコードを解析するかにも依存する．コア資産に仕上げるためには複数の分析装置で共用できる確証が必要であり，それはアーキテクトの意見あるいはソースコードの見通しの良さにかかっている．ADM 手法では，アーキテクチャとドメインにより切り出された部位が他と独立であることを前提としているため $CC_{i,j}$ の積み上げにより全体のコスト統制が図られる．

開発に入る前に，コア資産と決定されたソースコードはコア資産を取り扱う責任者のみが改変できるようパスワード付きの構成管理下に置かれる．このコア資産を変更するにあたっては，一部の製品だけが開発量減の利益を得て他の製品が法外な開発量増の不利益となることのないようコア資産チームと製品チームとの共同レビューを行って決定する．

以下に ADM 手法の具体的な手順を述べる．

（Step 1）開発計画の設定

実際に稼働する品質の確定したシステムを選定する．これは図 6 の旧アーキテクチャに相当しベースプロダクトと呼ぶ．加えて並行開発する N 機種の開発計画を確定させる．ベースプロダクトとの相似性から N 機種は絞られる．

（Step 2）ベースプロダクトの解析

図 5 に相当する表をベースプロダクトについて作成する．まず，ベースプロダクトのソースコードを分類して，アーキテクチャ要素とドメイン要素の行列セルに対応させる．次にベースプロダクトソースコードを解析して，行列セルごとにソースコードを ① コア資産として今後共通に使われるプログラム，② コア資産を含むプログラムでコア資産としてソースコードを分離されるべきプログラム，③ アプリケーションとして製品個々に任されるプログラム，の 3 種類に分類する．便法として ①，②，③ を c (core)，d (dual)，a (application) と記号化し表内に記す．②，③が N 機種の可変部位を吸収する変更部位になる．

（Step 3）N 機種の見積り

図 5 に相当する表を N 機種について作成する．N 個の表の中身（c，d，a）はベースプロダクトの表と同じであるが，c，d，a の 3 種類に分類されたそれぞれのソースコードにおいて，改造・追加が必要なものにはダッシュ（’）を付ける．ダッシュの付いたコードの意味は以下のとおりである．

①c’：コア資産として共通に使われるために改造する．
②d’：コア資産として共通に使われる要素を含んでいるため，その共通要素を抽出するために改造する．
③a’：個別機種対応のソースコードではあるが，他のコア資産と接続するために改造する．

ベースプロダクトの表にもダッシュが付く可能性がある．このダッシュは，ベースプロダクト内のコア資産が N 製品を吸収するにあたり変更が発生するということを意味する．表内には c，d，a，c’，d’，a’ が記され，このうち c 以外は改造・追加になるので，これらから N 機種の全体開発工数を見積もる．この時点で c のコア資産が少なすぎると，共通化の効果が薄く全体開発工数は大きくなる．一方，c’，d’，a’ が多すぎるとコア資産の存在によってかえって工数が増えることになる．c を大きくするか c’，d’，a’ を小さくするかの可能性がある場合には（**Step 2**）に戻り，コア資産を再計画する．コア資産の増減が全体開発工数の削減に寄与しなくなったならば，最終結果として全体開発工数を受け入れる．

（Step 4）チーム編成設計

チームはアーキテクチャ要素単位に形成される．ADM の表を縦方向（アーキテクチャ要素単位）にたどって開発工数を加算し，その変更量と難易度に応じて現状人材のスキルを照らし合わせ人材を配置する．アーキテクチャ要素内にコア資産が存在する場合コア資産開発者を割り当てる．

（Step 5）WBS 設計

ADM を行単位に眺め，業務フロー要素ごとに変更点を集める．これらを業務フロー要素の実現のための改造一覧，つまり WBS にまとめ以下のように活用する．

① アーキテクチャ要素の検証が終わった後，システム全体として妥当性をテストするときの拠り所とする．
② アーキテクチャ要素間のインタフェースを明確にして各アーキテクチャ要素を変更するチーム間でレビューをガイドする．

4. ADM 手法の適用

4.1 対象プロジェクト

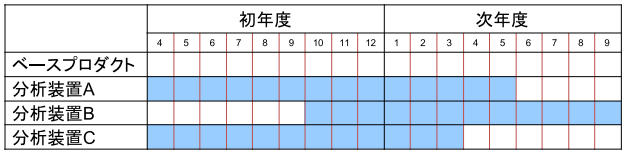
ADM 手法を適用した実プロジェクトについて述べる．事業戦略から与えられた開発課題は新たな 3 機種の分析装置を 1.5 年で結合するものであった．具体的な月単位の開発ロードマップを**表 1** に記す．初年度 3 月までに完了する製品をベースプロダクトとした．網掛け部位が開発工程である．

分析装置 A，B の 2 本の開発は，それぞれ設計に始まり実装・テスト・QA 完了までの工程となり，C は QA を含まない装置性能検証の工程である．ここで，QA とは組込

表 2 P，A，B．C 製品の ADM
Table 2 ADM of P, A, B, and C products.

	ドメイン		アーキテクチャ																							
	大分類	中分類	UI				DB				検体管理				装置通信				分析装置				外部通信			
1	STARTUP（SU）		c’	c	c	c	d	d	d	d’	c’	c	c	c	d	d	d	d’	a	a	a	a				
2	全体管理(GM)	GM1	d	d	d	d	d	d	d	d	a	a	a	a	d	d	d	d	a	a	a	a				
3		GM2	d’	d’	d’	d	d	d	d	d	a	a	a	a	d	d	d	d	a	a	a	a				
4		GM3	d’				d	d	d	d	a’	a	a	a	d	d	d	d	a	a	a	a				
5		GM4	d	d	d	d																				
6	試薬準備(RP)	RP1	d	d’	d’	d’	d	d	d’	d’	a	a	a’	a	d	d	d	d	a	a	a	a				
7		RP2	d	d	d	d	d	d	d	d	a	a	a	a	d	d	d	d	a	a	a	a				
8		RP3	d	d	d	d’	d’	d	d	d’	a’	a	a	a’	d	d	d	d’	a’	a	a	a’				
9		RP4	d	d	d	d	d	d	d’	d	a	a	a’	a	d	d	d	d	a	a	a	a				
10	測定（M）	M1					d	d’	d’	d’	c	c’	c’	c’	d	d’	d’	d’	a	a’	a’	a’				
11		M2					d	d’	d’	d’	c	c’	c’	c’	d	d’	d’	d’	a	a’	a’	a’				
12		M3					d’	d’	d’	d’	c’	c’	c’	c’	d	d’	d’	d’	a’	a’	a’	a’				
13		M4	d	d’	d’	d’	d	d	d’	d’																
14	キャリブレーション（C）	C1	d	d	d	d’	d	d	d	d’	c	c	c	c’									c	c	c	c
15		C2	d’	d	d’	d’	d’	d	d’	d	c’	c	c’	c’									c	c	c	c
16	精度管理(QC)	QC1	d	d	d	d’	d	d	d	d’	c	c	c	c’									c	c	c	c
17		QC2	d	d	d’	d					c	c	c’	c’									c	c	c’	c’
18		QC3	d	d	d	d																	c	c	c	c
19	検査業務(I)	I1	d’	d	d	d	d’	d	d	d													c	c	c	c
20		I2	d	d	d	d	d	d	d	d													c	c	c	c
21		I3	d’	d	d	d’	d	d	d	d’	c	c	c’	c’									c	c	c	c
22		I4	d’	d’	d	d’	d	d	d’	d	c	c	c’	c									c	c	c	c
23		I5	d	d	d	d’					c	c	c	c	d	d	d	d	a	a	a	a’				
24		I6									c	c	c	c	d	d	d	d	a	a	a	a				
25	SHUTDOWN（SD）		c	c	c	c	c	c	c	c	c	c	c	c	d	d	d	d	a	a	a	a				
		分析装置	P	A	B	C	P	A	B	C	P	A	B	C	P	A	B	C	P	A	B	C	P	A	B	C

表 1 ADM 手法適用対象プロジェクトの計画
Table 1 Plan of the projects that ADM was applied to.



みソフトウェアの検査であり，これを終えるとシステムの妥当性を検証する網羅的なシステムバリデーションが社内および社外で実施される．このシステムバリデーションが完了して初めて医療機関に製品としてシステムが提供される．B のスタートが遅れているのは分析装置ハードウェアの開発と同期させた結果である．

4.2 ベースプロダクトの解析と 3 機種の見積り

表 2 にベースプロダクトと分析装置 A，B，C の 3 機種を加えた ADM の分析最終結果を示す．行には AIMS の業務フロー要素の大分類と中分類をならべ，列にはアーキテクチャ要素をならべている．最下段に分析装置の区分けが記されている．P はベースプロダクトを示す．表内にはソースコードを分類して前章の（**Step 3**）で説明したダッシュ付記号 c’，d’，a’ を付けている．

（**Step 2**）から（**Step 3**）の分析の途上，コア資産を積極的に形成する方向に統制した例として検体管理，その逆に消極的に統制した例として装置通信があった．検体管理はシステムをモデリングする部位であり抽象化が可能と判断しコア資産の幅を広げた．装置通信では分析装置 A，B，C の通信形式の違いが大きく，違いをすべて吸収する構成容易なコア資産を作るには工数が大きすぎるため最低限のコア資産形成とした．

最終的に，表中行数にあたる 25 のドメイン領域中 6 領域において（たとえばドメインの No.2 や 5 など）変更は発生しないものの，残りの 19 ドメイン（76%）には変更が必要と判断された．このダッシュは，ベースプロダクト内のコア資産が 3 製品を吸収するにあたり変更が発生するということを意味する．事業計画における現実的な変更要求である 3 機種を頼りに，これらが最適に接続できるために再利用性をいかに高められるかを追求している．

表 3 開発コスト見積り
Table 3 Development cost estimate.

ベースプロダクト	本プロジェクト				
	コア資産	A接続	B接続	C接続	計
100	11.3	18.9	34.0	52.8	117.0

4.3 見積り結果

表 3 にベースプロダクトに要した開発工数を 100 としたときの本プロジェクト見積りを示す．

ベースプロダクトの開発実績に比べて，見積りでは 17%増しで 3 機種開発の見通しを得た．前述したように ADM 手法では個別開発コストとコア資産コストとの調和を図るという設計行為が含まれているため，17%増しに収める設計ができたといえる．ここで，C 接続の開発工数が他に比べて大きいが，これは分析装置自体に開発要素があったためである．見積りは本プロジェクトの開始可否を判定する試金石であり，この結果は本プロジェクトの成功率を高めた．

表 4 チーム編成設計の結果概要
Table 4 Summary of development team design result.

	UI	DB	検体管理	装置通信	分析装置	外部通信
コア資産開発者	○	○	○	○		○
分析装置単位にチーム編成	○		○		○	

No.	要件名称	要件内容
	装置C用保守ツール	装置C固有の保守ツールを実行する。

関連プロジェクト				要求元	優先度	採否	採否コメント
P	A	B	C				
			○				

アーキテクチャ要素	詳細要素	新規・改造内容	改造量
外部通信			
UI	System	パラメータ設定及び実行の画面を追加。	
	Overview	保守の実行状況の表示。	
DB		保守パラメータ登録。	
検体管理	制御	保守に使用する検体制御	
	状態処理	保守の状態モニタリング	
装置通信		保守の指令とパラメータ転送。	
分析装置	保守管理	保守の実行制御と実行状況報告。	

図 7 ADM 手法から導出される WBS 例
Fig. 7 WBS illustration to be introduced by ADM.

また，業務フロー要素という要求仕様とアーキテクチャ要素という実現機能が明確になりソースコードの自由度が限定されたことも見積り精度を向上させている．

4.4 チーム編成設計

表 4 にチーム編成設計の結果概要を示す．アーキテクチャ要素単位にコア資産開発者を割り当てるか，分析装置単位にチーム編成をするかを検討した（表中の○）．コア資産の開発には各アーキテクチャ要素で技術的にリードできる人材を最低 1 名あてた．

DB，装置通信，外部通信については開発量が比較的少ないこととその標準的な性格から，分析装置単位のチーム編成は行わずコア資産開発者のみを配置した．分析装置はすべてコア資産を含まない分析装置依存のコードであるため，分析装置単位のチーム編成とした．

このようにマトリクス状に組織配置を検討することにより，技術的な難易度と具体的な人のスキルとを調和させることが可能となり，開発のリスク軽減が図られた．

4.5 WBS 設計

図 7 は本プロジェクトで作成した WBS 仕様書の一例を簡略化して示している．要件名称および内容に，業務フロー要素が書かれ，下段のアーキテクチャ要素ごとに変更内容が書かれている．この例では保守ツールの実行のために，画面，DB，検体管理，装置通信，分析装置の変更が連携することが記される．この WBS を元に，アーキテクチャ要素担当者は設計書を書き，ここに上がっている関係者が集まりレビューを行い実装に入る．各要素は設計書に基づき試験され，後に組み合わせた時点で，システムテスト「装置 C 固有の保守ツールを実行する」によって最終テストを終える．

このように，ADM 手法では WBS を導出できるため，アーキテクチャにおける全体と部分の掌握が容易であり，最終試験のあり方が明示されるため，要素開発が完結することができ，開発効率向上への貢献が大きい．

表 5 プロジェクトの実績進度
Table 5 Actual progress of the projects.

	初年度									次年度									
	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11
ベースプロダクト	■	■																		
分析装置A			■	■	■	■	■	■	■	■	■									
分析装置B									■	■	■	■	■	■	■	■				
分析装置C			■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■

5. ADM 手法適用結果に対する考察

ADM 手法を適用して得られた以上のような見積りや計画に基づき開発が行われた．その結果を**表 5** に示す．

実際には，本プロジェクトに先立つベースプロダクトの開発が 2 カ月遅延したため，本プロジェクトの開始は 2 カ月遅延した．しかし，プロジェクト A においては 3 カ月，プロジェクト B においては 2 カ月前倒しで完了することができた．プロジェクト C については分析装置の機能追加が行われたため 2 カ月の遅延となったものの，全体として，1.5 年で 3 機種の開発を完了させ計画に適合したコスト統制ができた．過去の自社内実績では SPL を使わない開発で 2 機種開発に 2.5 年をかけている．この 2 機種開発ともにその規模は，本論文における分析装置 A，B，C を平均した規模に比してほぼ対等といえる．したがって，年当りの開発機種数は 0.8（機種/年）から 2.0（機種/年）と改善し，2.5 倍の生産性を得たといえる．

開発のゴールを達成することができた理由として，見積り精度が向上したこと，および，再利用コードが増えて改造量を抑えられたことがあげられる．3 機種を開発完了するための見積り工数は表 3 により，ベースプロダクトの開発工数に比べ 1.17 倍であった．ベースプロダクト開発は SPL を用いない 1 機種開発であったため，その工数は 2.5（年）/2（機種）×（全人員）= 1.25（年 · 全人員）となる．これを 1.17 倍した 1.46（年 · 全人員）が 3 機種開発の見積り工数である．これに対して実績工数は，ベースプロダクト開発と本プロジェクトとの間で開発人員に差を設けていないことを考慮すると，1.5（年 · 全人員）であった．見積り誤差は +2.7%であり実用的な見積り値が得られたといえる．

再利用コードについては，コア資産の中核をなす UI · DB · 検体管理の総ソースコード行数におけるコア資産の割合を**表 6** に示す．コア資産以外の行数は分析装置固有のソースコードとなる．見積りに対して実績は A，B の接

表 6 UI・DB・検体管理におけるコア資産の割合
Table 6 Ratio of core asset in UI, DB, and sample manager.

	A接続	B接続	C接続
見積り	95.4	93.7	76.0
実績	96.5	98.1	60.8

続において差が少なく，全体開発を揺らぎなく行う程度に見積り精度は高い．また，A，B の接続では見積り以上のコア資産率を実績で出しており，開発の前倒しに貢献している．これはベースプロダクトの改造の中でより高く均衡ある共用度が実現できたためである．一方，分析装置 C では見積りに比べてコア資産率の実績が下がっている．これは分析装置固有の機能追加によるためで，実際進捗の遅れが出ている．性能検証のための装置には仕様の揺らぎがあり実際見積り以上の期間がかかっている．しかし，分析装置 A，B の開発前倒しがこの遅れをカバーする形となり，また，60%以上のコア資産率はコア資産の有用性を示している．

また，ADM 手法で生成された WBS が開発の中で有効に活用された．特にアーキテクチャ単位に編成されたチームはそれぞれの専門領域において独立性を維持しながら，WBS を通じてレビュー，システムテストで全体の整合性をチェックすることができた．この点でコア資産を含む開発計画の困難さを低減できたといえる．

なお，ここで抽出されたコア資産は，その後 4 年間で新分析装置 5 機種，新搬送路 2 機種接続に活用されている．これは抽出されたコア資産が実効的であったことを示す．

ADM 手法は，長期間にわたり変更の可能性が少ないアーキテクチャ構成要素とドメイン知識要素が存在するような製品開発を前提にしている．医用機器向けの規制により業務が規律されている医用製品においては比較的安定した業務フローがドメイン知識要素として利用できる．したがって，規制適用が求められる製品においてはアーキテクチャ構成要素が安定していれば，ADM 手法の適用可能性は高く，本事例はそれを例示している．

6. 関連する研究との比較

SPL の成功事例は実践的なフレームワークにまとめられ公表されている [7]．ドメイン理解はそのフレームワークの 1 つであり，関連する手法として Feature Oriented Domain Analysis（FODA）が紹介されている [9]．これは，製品間で共通な特徴であるフィーチャを抽出し，その組合せをモデル化し，個別製品を開発するベースにしている．フィーチャが多数になるとその組合せは膨大になりフィーチャ設定の工数は増大する．そこでフィーチャの論理的組合せを合理化する手法が提案されている [11]．しかし，フィーチャの組合せ数の削減であり，フィーチャ数の削減にまでは踏み込んでいない．また，Schmid [12] はフィーチャ設計にあたってフィーチャを取捨選択する手法を提案している．フィーチャを使うことによるリスクと利益，およびビジネス目標に合致するかを分析し，使用の有無を判定している．しかし，フィーチャ作成コストと利用利益の比較にとどまり，製品固有アプリケーションを作成するコストの検討を含んでいない．SPL を用いたときの一般的な経済性は損益分岐点により説明されている [13]．すなわち，SPL におけるコア資産開発費を固定費とし，それを使った製品が一定数以上開発されれば固定費を回収できるとしている．しかし，コア資産を作りすぎずに開発全体を統制する手法については報告がない．以上のように 1 章の（課題 1）に対応する製品個別開発を配慮したコア資産統制は行われていない．

その他既存ソースコードからプロダクトラインを構築する手法では Stoermer ら [14] が提案する MAP がある．アーキテクチャに関する情報を抽出しプロダクトライン化を評価している．しかし，ドメイン知識が評価に含まれず構造の要件に検討余地がある．また，設計書や取扱説明書や専門家などからリバースエンジニアリング情報を得てコア資産を構築する手法も提案されている [15]．情報源にアーキテクチャ要素が含まれているが，それ以上の粒度における解析は述べられていない．アーキテクチャとドメインの両面から再利用性を検討している研究に Koziolek らの研究 [16] がある．これはアーキテクトとの面談から抽象レベルの高いアーキテクチャを記述して再構築に活用している．以上いずれの文献もアーキテクチャとドメインの両面をソースコード解析に活用しておらず，したがって，1 章の（課題 2）のコア資産見積り精度面について言及がない．

コア資産開発と製品開発の全体開発において [17] が WBS の生成を将来の課題としてあげており，1 章の（課題 3）に対応する具体的解は書かれていない．

以上，アーキテクチャ要素とドメイン要素の両面の知識を使っている点では類似例があるが，本研究はこの知識を複数製品ではなく単一製品のソースコード解析に活用してコア資産を抽出している点に特徴がある．単一の製品であっても両知識を用いることにより適切なコア資産の開発計画を導出できる．

7. おわりに

本論文では，アーキテクチャ要素をドメイン要素単位に分解してコア資産とアプリケーションの区分け解析を行う ADM（Architecture Domain Matrix）手法について述べた．この手法は AIMS（Analyzer Integration Management Software）のような多岐にわたるソフト改造に対して生産効率を向上させる．本方式は，現実の分析装置仕様に適応したコア資産とアプリケーションの開発量を見積もっている．また，見積りに対応しアーキテクチャのスキルに対応したチーム編成を導出し，WBS を導出して開発要素が最

終的に貢献する業務要素を明らかにして設計からテストまでの開発プロセスを支援する点に特徴がある．

ADM 手法を実プロジェクトに適用したところ，旧製品 1 機種をインプットとして 3 機種の同時開発を実現し，手法の有効性が示された．その後 4 年にわたり新分析装置や新搬送路の接続に活用され，抽出されたコア資産の有効性も示された．ADM 手法で採用した構造は中長期的に改変すべきときを迎えるので，構造改革のタイミングを決定することが次なる課題である．

今後の取り組みとしては，他のドメインへの応用とともに，要素のバリデーションが全体のバリデーションにつながるようなアーキテクチャの研究が考えられる．そのようなアーキテクチャにおけるコア資産の形を研究したい．

謝辞 本研究に関連して開発に多大な貢献をした竹辺靖昭氏および技術者達に敬意を表するとともに，強力に支援していただいた日立オートモティブシステムズ（株）の森清三事業部長付に心より感謝いたします．また，貴重なご意見をいただいた（株）日立製作所日立研究所の吉村健太郎主任研究員に感謝いたします．

参考文献

[1] 三巻 弘，兒玉隆一郎，栗山裕之：臨床検査の効率的な運用に適したモジュール組合せ方式の血液自動分析装置，日立評論，Vol.79, No.10, pp.757–762 (1997).
[2] 経済産業省：IT 化の進展と我が国産業の競争力について，経済産業省（オンライン），入手先 ⟨http://www.meti.go.jp/committee/materials/downloadfiles/g70124b06j.pdf⟩（参照 2013-09-08）.
[3] Software Engineering Institute, Carnegie Mellon University: Software Product Lines | Overview (online), available from ⟨http://www.sei.cmu.edu/productlines/⟩ (accessed 2013-09-08).
[4] 吉村健太郎，菊野 亨：5 組込みシステムにおけるソフトウェアプロダクトラインの導入（〈特集〉ソフトウェア再利用の新しい波—広がりを見せるプロダクトライン型ソフトウェア開発），情報処理，Vol.50, No.4, pp.295–302 (2009).
[5] 吉村健太郎，ダルマリンガム・ガネサン，ディルク・ムーティック：プロダクトライン導入に向けたレガシーソフトウェアの共通性・可変性分析法，情報処理学会論文誌，Vol.46, No.8, pp.2482–2491 (2005).
[6] Fowler, M.: Refactoring: Improving the Design of Existing Code, Addition Wesley (1999).
[7] Software Engineering Institute, Carnegie Mellon University: A Framework for Software Product Line Practice, Version 5.0 (online), available from ⟨http://www.sei.cmu.edu/productlines/frame_report/index.html⟩ (accessed 2013-09-08).
[8] Frakes, W.B. and Kang, K.C.: Software reuse research: Status and future, *IEEE Trans. Software Engineering*, Vol.31, No.7, pp.529–536 (2005).
[9] Kang, K., Cohen, S., Hess, J., et al.: Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-021, ADA235785) (online), Software Engineering Institute Carnegie Mellon University, available from ⟨http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm⟩ (accessed 2013-09-08).
[10] Hofman, P., Pohley, T., Bermann, A., et al.: Domain Specific Feature Modeling for Software Product Lines, *Proc. 16th International Software Product Line Conference* (*SPLC*), pp.229–238 (2012).
[11] 吉村健太郎，成沢文雄，菊野 学：製品リリース履歴における論理的結合集合に基づいた横断フィーチャ分析法，情報処理学会論文誌，Vol.50, No.11, pp.2654–2664 (2009).
[12] Schmid, K.: A comprehensive product line scoping approach and its validation, *Proc. 24th International Conference on Software Engineering*, *ICSE '02*, pp.593–603, ACM (2002).
[13] Northrop, L.: Software product lines essentials (online), Software Engineering Institute, Carnegie Mellon University (2008), available from ⟨http://www.sei.cmu.edu/library/assets/spl-essentials.pdf⟩ (accessed 2014-02-04).
[14] Stoermer, C. and O'Brien, L.: MAP — Mining architectures for product line evaluations, *Proc. Working IEEE/IFIP Conference Software Architecture 2001*, pp.35–44 (2001).
[15] Knodel, J., John, I., Ganesan, D., et al.: Asset Recovery and Their Incorporation into Product Lines, *WCRE '05: Proc. 12th Working Conference on Reverse Engineering*, pp.120–129, IEEE Computer Society (2005).
[16] Koziolek, H., Coldschmidt, T., de Gooijer, T., et al.: Experiences from Identifying Software Reuse Opportunities by Domain Analysis, *Proc. 17th International Software Product Line Conference* (*SPLC*), pp.208–217 (2013).
[17] Jones, L.G. and Bergey, J.K.: Exploring Acquisition Strategies for Adopting a Software Product Line, No. NPS-AM-10-041, CARNEGIE-MELLON UNIV COLORADO SPRINGS CO (2010).

兒玉 隆一郎 （正会員）

1957 年生．1981 年東京大学工学部計数工学科卒業．同年（株）日立製作所入社．1989 年米国ロチェスター工科大学大学院修士課程修了．2001 年（株）日立ハイテクノロジーズ入社．組込みソフトウェアの研究開発に従事．

島袋 潤 （正会員）

1965 年生．1988 年大阪大学基礎工学部情報工学科卒業．1990 年同大学大学院博士前期課程修了．同年日立製作所入社．以来一貫して，ソフトウェア開発技術の研究開発および実用化に従事．

高木 由充

1967 年生．1989 年東京電機大学理工学部経営工学科卒業．同年（株）日立製作所入社．2001 年（株）日立ハイテクノロジーズ入社．臨床検査向け自動分析装置の開発に従事．（一社）日本臨床検査機器・試薬・システム振興協会（JACLaS）理事長．

小泉 忍

1956 年生．1980 年東京工業大学工学部制御工学科卒業．1982 年同大学大学院総合理工学研究科システム科学専攻修士課程修了．同年（株）日立製作所システム開発研究所入社．言語処理系の研究開発，ソフトウェアエンジニアリングの研究開発等に従事後，社内組込みソフト開発の高度化を推進．

田野 俊一 （正会員）

1958 年生．1981 年東京工業大学工学部制御工学科卒業．1983 年同大学大学院総合理工学研究科システム科学専攻修士課程修了．同年（株）日立製作所システム開発研究所入社．1990～1991 年カーネギメロン大学客員研究員．1991～1995 年国際ファジィ工学研究所．1996 年電気通信大学大学院情報システム学研究科助教授．2000～2001 年マサチューセッツ工科大学客員研究員．2002 年電気通信大学教授．現在に至る．博士（工学）．主として，人工知能，知識工学，自然言語理解，あいまい理論，知的ユーザインタフェースの研究に従事．人工知能学会，日本ファジィ学会，言語処理学会，AAAI，IEEE，ACM 各会員．