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

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[††]

本論文では，レガシーシステムにソフトウェア・プロダクトライン（SPL）を導入するための可変性・共通性分析手法を提案する．既存システムに基づいてコア資産を開発するためには，実装ソースコードの解析・再利用が有効である．提案手法では，レガシーシステム間の共通性を評価するために，異なるソフトウェア間でコードクローンによる解析を行う．また，共通性・可変性を評価するために，SPL の観点から見てコードクローンをいくつかのタイプに分類する．さらに，参照アーキテクチャを利用して階層的な分析を実施する．提案手法によって，実装されているソースコードに基づいてレガシーシステム間の共通性・可変性を効率的に評価することができる．また，本手法を用いて自動車用エンジン制御システム間の共通性・可変性評価を行った．

A Method to Assess Commonality and Variability of Existing Systems into a Product Line

Kentaro Yoshimura,[†,☆] Dharmalingam Ganesan[††] and Dirk Muthig[††]

This paper describes a method to assess commonality/variability of existing systems into a software product line (SPL). For developing core assets from the existing systems, analyzing and reusing the implemented source code are effective method. In order to assess the commonality, we identify code clones between different systems. In the assessment of commonality and variability, we classify the clones into categories from the view point of SPL variability. We also apply hierarchical decomposition assessment of systems. By using our method, we can assess commonality and variability between existing systems from the view point of implementation. We examine our method with a case study to engine management systems for vehicles.

1. はじめに

自動車，産業用機器，家電製品等，広い分野で，組み込みソフトウェアが用いられている．組み込みソフトウェアは，しばしば特定のビジネス領域に対する単一の製品として開発され，その後ビジネスの成功とともに改良，派生が加えられ複数の製品群として開発されるようになる．

たとえば自動車用エンジン制御システムは，エンジン仕様の違い，自動車メーカごとの独自要求仕様，さらに地域ごとに異なる法規制に対応するために製品バリエーションが非常に多い．また，ソフトウェア規模・複雑度の増大，開発コスト・期間の削減，信頼性の向上等の要求によって，バリエーションを個別に開発することはきわめて困難であり，組み込みソフトウェアの再利用性向上が重要な課題になっている．

しかし，組み込みシステムにおけるソフトウェア再利用手法は，既存の類似製品を流用して新規機能を開発・統合する差分開発が主流である．すなわち，新規製品の要求仕様に類似したレガシーソフトウェアを流用して設計仕様書，ソースコード，テストケース等を変更し，新たなソフトウェアを開発している．また，組み込みソフトウェアはリソースの制約から個々の製品に特化して開発・最適化されることが多く，製品群を横断したソフトウェア再利用が困難であるという問題があった．

近年，製品群を横断したソフトウェア再利用のための手法として，ソフトウェア・プロダクトライン

† Hitachi Europe SAS, France
†† Fraunhofer Institute for Experimental Software Engineering, Germany
☆ 現在，株式会社日立製作所日立研究所
Presently with Hitachi Research Laboratory, Hitachi, Ltd., Japan

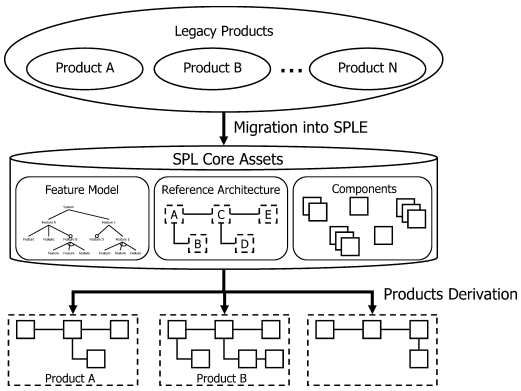

図 1 SPL の概念図
Fig. 1 SPL overview.

(Software Product Line，以下，SPL）が注目されている[1)~9)]．**図 1** にレガシーソフトウェアに基づく SPL 型開発の概要を示す．既存のソフトウェア資産（ソースコード，設計仕様書，アーキテクチャ等）を入力として，SPL のコア資産を開発する．コア資産には，製品間の共通性・可変性の分析結果，製品群の共通フレームワークとなる参照アーキテクチャ，ソフトウェア部品等が含まれる．SPL 型開発の成功事例として携帯電話や自動車制御システム等が報告されており[10)~12)]，同一の組織が類似の仕様で多数の製品を開発する事例に有効とされている．

SPL 型開発では，要求分析により製品群の共通性および可変性を分析する必要があるとされるが，この作業は容易ではない．そのため，自動化が可能なソースコードを用いた共通性・可変性分析支援が必要不可欠である．たとえばエンジン制御システムは 50 万 LOC を超える大規模ソフトウェアであり，当然要求仕様書の規模も大きい．さらに自動車メーカごとに仕様書の形式が異なるだけでなく，使用している自然言語（日本語，英語）が異なるために要求分析の自動化が難しいという課題がある．モデルベース開発[13)] の導入も始まっているが部分的であり，モデルレベルでの分析は限定的なものとなる．

また，自動車制御等のセーフティ・クリティカル・システムでは信頼性が重視されるため，実績のある既存ソースコードを再利用したいという要求がきわめて強い．新規に開発する場合に比べて，ソースコードを再利用した場合の信頼性は高くなるといわれている[14),15)]．そこで，開発済みのレガシーソフトウェアの製品バリエーション間における共通性・可変性を分析し，SPL におけるコア資産としてリファクタリングすることが望ましい[16)~18)]．

そこで我々の研究の目的は，レガシーソフトウェアへの SPL 技術導入のための共通性・可変性分析法を確立することである．特に，参照アーキテクチャと，後述するクローン分析メトリクスを用いて階層的・効率的にレガシーソフトウェアの共通性・可変性を分析することを目的とする．

上記目的を達成するための第 1 の課題は，複数の製品に分散したコードの共通部分を効率的に発見する分析プロセスの開発である．我々は，対象ドメインの製品群が有する参照アーキテクチャに着目し，システムレベル，サブシステムレベル，コンポーネントレベルでのレガシーソフトウェア間の実装の共通度を階層的に分析する．これにより，製品間で実装の共通性が高い領域と，可変性が高い領域とが効率的に分析できるようになる．

第 2 の課題は，複数のレガシーソフトウェア間における共通性分析手法の開発である．ソフトウェアの共通性を分析する手法として，ソースコード中の重複したコード列を検出するコードクローン分析手法がある．しかし，提案されているコードクローン分析手法は単一のソフトウェア内でのコード列重複の検出を目的としたものであり，複数のソフトウェア群にまたがる共通部分を検出するという我々の目的には十分でない．そこで我々は，複数のソフトウェアにまたがるソースコードの共通性を分析する新たなメトリクスを開発した．これにより，レガシーソフトウェア間の実装の共通性および可変性を定量的に分析できる．

第 3 の課題は，SPL 導入支援に適した共通性・可変性の分類方法の開発である．上記手法に基づいて製品間の共通性を分析できるが，ソフトウェア・プロダクトライン導入のためには，分析したコンポーネントを共通部とするか，可変部とするかを判断する必要がある．そこで我々は，上記メトリクスに基づいたソフトウェアの共通性・可変性の分類法を導入する．本分類法により，SPL 導入に向けたソフトウェアの共通性・可変性を効率的に判断できる．

以上により，実績のあるレガシーソフトウェアを重視した SPL の導入に適した，ソフトウェア共通性・可変性の分析が可能になる．

また，提案手法の適用対象は関数集合として定義されるプログラムソースコード集合とし，同一組織内で開発されたソフトウェアであることを前提とする．BAPO[19)] 等の SPL 成熟度評価モデルで示されるように，SPL 技術の導入は同一組織内から開始される．SPL 導入の効果的な支援を行うためには，同一組織内で開発されたソフトウェアを前提とした手法提案が有効であるためである．

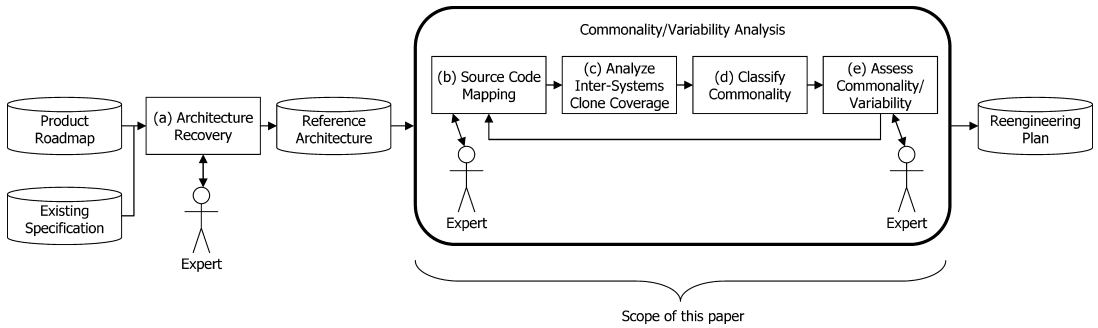

図 2 分析アプローチ全体図
Fig. 2 Overview of assessment approach.

以下本論文では，まず 2 章で，本論文のアプローチについて述べる．次に 3 章で，提案するレガシーソフトウェア共通性・可変性分析法について述べる．そして 4 章で，本手法の自動車エンジン制御用組み込みソフトウェアへの適用実験を行い，本手法の有用性を確認する．また 5 章で関連する研究との比較を，6 章で今後の課題を示す．

2. 本論文のアプローチ

図 2 に本研究のアプローチを示す．適用対象は，関数集合として定義されるプログラムソースコード集合であり，同一組織内で開発された製品群であることを前提とする．

本アプローチでは，まず初めに対象製品群の参照アーキテクチャの設計を行う．製品ロードマップとレガシーシステムの仕様書に基づいて，製品群で共通利用する参照アーキテクチャを設計する（図 2 (a)）．この設計はドメインの専門知識を持った専門家によって実施する．なお，この時点では参照アーキテクチャの可変性は設計しない．

次に，レガシーソフトウェアの共通性・可変性の分析を行う．本論文では，この共通性・可変性分析法について提案する．まず初めに，レガシーソフトウェアを構成する関数を参照アーキテクチャ内のコンポーネントへとマッピングして，関数の集合を定義する (b)．続いて，レガシーソフトウェアの関数における製品間コードクローンを分析する (c)．その後，本論文で導入する製品間クローンカバレッジに基づいて，関数ごとの共通性・可変性を分類する (d)．分類結果に基づいてコンポーネント内でのクローン分布状況を測定し，専門家によってコンポーネントの共通部品化の判断が行われる (e)．該当コンポーネントがさらに複数のサブシステムにより構成されると判断される場合には，分割を詳細化して本プロセスを再実行する．

提案手法の結果として，レガシーシステムに対する再構築計画が導かれる．SPL の実際の導入では，リスク軽減を目的として段階的な導入が行われることが多い．そこでレガシーシステムの関数群に対し，外部への振舞いを保ったままでのリファクタリングの方針が提案される．

3. 共通性・可変性の分析

3.1 共通性・可変性分析の階層的マッピング

一般に 1 つの組み込み制御ソフトウェアは複数の機能の組合せからなり，参照アーキテクチャも機能ごとに階層化される[13)]．そこで，共通性・可変性の分析も機能単位にモジュール化し，全体・部分関係に従って階層化する．すなわち，システムの全体構成における共通性・可変性，システムを構成する 1 つのサブシステムの共通性・可変性，サブシステムを構成するコンポーネントの共通性・可変性をそれぞれ段階的に分析する．

図 3 に，エンジン制御システムの参照アーキテクチャと共通性・可変性分析との対応例を示す．エンジン制御システムは，入出力機能や制御アプリケーション等のサブシステムからなる．また，制御アプリケーションは，アイドル回転数制御やクルーズ・コントロール制御等のコンポーネントから構成される．

まず初めに，システム全体が単一のモジュールから構成されていると仮定し，レガシーソフトウェア製品間の共通性・可変性を分析する．次に，参照アーキテクチャのサブシステムに対してレガシーソフトウェアの関数をマッピングし，製品間でのサブシステムどうしの共通性・可変性を分析する．その後，システムの規模や分析結果に応じて階層的に共通性・可変性分析を詳細化し，共通化可能なサブシステムと可変部分となるサブシステムとを分類する．さらにサブシステムの分析結果に応じて，詳細な分析をコンポーネントに

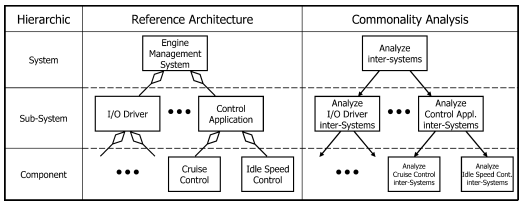

図 3 階層的な共通性・可変性分析
Fig. 3 Commonality and variability analysis using decomposition hierarchy.

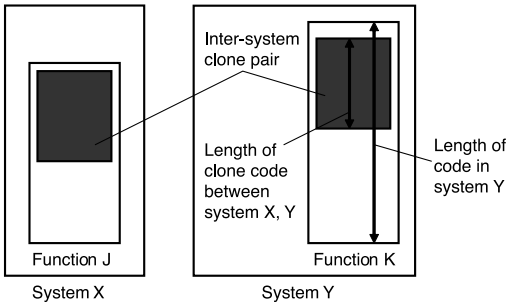

図 4 システム間コードクローン
Fig. 4 Inter-system code clone.

対して実施する．

3.2 システム間コードクローン

一般的に，コードクローンとは単一のシステム内の重複したコード列のことを指す[20]．しかし，我々の目的は製品群を構成している関数の製品間を横断した共通性を分析することであり，従来のコードクローンの定義では不十分である．そこで我々は，システム間コードクローンの概念を導入する．

図 4 に製品間コードクローンの概念を示す．互いに等しい 2 つのコード列が異なるシステムの関数に存在する場合，このコード列をシステム間コードクローンとして扱う．

次に，検出されたコードクローンに基づいて各関数の共通性・可変性を評価するためのメトリクスとして製品間クローンカバレッジを定義する．J，K を異なる製品における関数とし，製品間クローンカバレッジは次のように定義する．

$$CloneCoverage(J,K) = \frac{\text{Lines of Cloned Code from J to K}}{\text{Lines of Code in K}}$$

製品間クローンカバレッジとは，ある製品 Y の関数 K に含まれる行のうち，異なる製品 X に含まれる関数 J に含まれる行と一致する行の割合を示す．製品 X から製品 Y へと関数を流用した場合には製品間クローンカバレッジは 100%となり，関数内部のコードを完全に書き換えた場合には 0%となる．

```
int foo(int j){
  if(j<0)
    return j;
  else
    return j++;
}
```

System X

```
int foo(int j){
  if(j<0)
    return j;
  else
    return j++;
}
```

System Y

図 5 Type 1 コードクローンの例
Fig. 5 An example of a clone Type 1.

```
int foo(int j){
  if(j<0)
    return j;
  else
    return j++;
}
```

System X

```
int foo(int j){
  if(j<=0 && j>=-5)
    return j;
  else
    return j++;
}
```

System Y

図 6 Type 2 コードクローンの例
Fig. 6 An example of a clone Type 2.

3.3 製品間クローン関数の分類

さらにレガシーソフトウェアの共通性・可変性の判断を支援するため，製品間クローンカバレッジに基づいてコードクローンを次の 4 つに分類する．

- Type 1
  関数のインタフェース（関数名，引数，戻り値）が同一であり，かつクローンカバレッジが 100%である製品間クローンペアを Type 1 とする．**図 5** に Type 1 クローンの例を示す．Type 1 クローンは，SPL における共通部分であると分類される．
- Type 2
  関数のインタフェースは同一だが，ソースコードの一部が変更されているクローンペアを Type 2 とする．**図 6** に Type 2 クローンの例を示す．Type 2 クローンは，製品特有の要求を満たすために変更された部分であり，SPL における共通部分と可変部分が混在した関数であると分類される．
- Type 3
  関数のインタフェースが同一だが，ソースコードの大部分が異なっているクローンペアを Type 3 とする．**図 7** に Type 3 クローンの例を示す．実際には，Type 2 クローンと Type 3 クローンとは，製品間クローンカバレッジの閾値により判定する．Type 3 クローンは，SPL における可変部分であると分類される．
- Type 4
  関数のインタフェースは異なっているが，実装されているコードには重複部分があるクローンペアを Type 4 とする．**図 8** に Type 4 クローンの例を示す．実際の開発では，製品間で関数をコピー

し変更を加えた後に関数名を変更することがある．このようなケースを発見するために，Type 4 クローンを用いる．

上記の製品間クローンカバレッジを用いたクローンペアの分類により，レガシーソフトウェア間での共通性と可変性との分類を容易に行うことができる．たとえば，Type 1 クローンを共通部分に，Type 3 クローンを可変部分としてドメイン分析や参照アーキテクチャでの可変部設計を支援する．

3.4 クローン分布

3.1 節において関数をマッピングしたシステム，サブシステム，コンポーネント等，関数集合としての共通性・可変性を評価するために，クローンの分布状況を調査する．

図 9 にクローン分布の概念を示す．対象とする関数集合に含まれる全関数の総行数を分母とする．次に，3.3 節においてそれぞれの各 Type に分類された関数ごとの合計行数を，各 Type の分布とする．この分布状況を用いて，SPL の観点から見た関数集合の性質を評価する．

```
int foo(int j){
  if(j<0)
    return j;
  else
    return j++;
}
```

System X

```
int foo(int j){
  return j--;
}
```

System Y

図 7 Type 3 コードクローンの例
Fig. 7 An example of a clone Type 3.

```
int foo(int j){
  if(j<0)
    return j;
  else
    return j++;
}
```

System X

```
int bar(int j){
  if(j<0)
    return j;
  else
    return j+2;
}
```

System Y

図 8 Type 4 コードクローンの例
Fig. 8 An example of a clone Type 4.

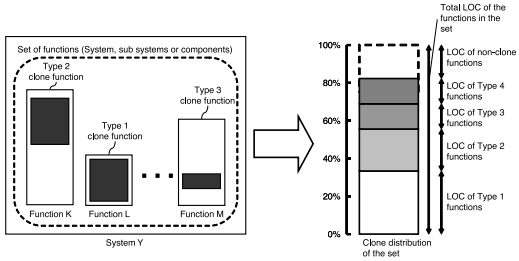

図 9 クローン分布の例
Fig. 9 An example of clone distribution.

4. 適用実験

4.1 実験概要

実際の組み込み制御ソフトウェアに対して本方式を適用し，共通性・可変性を分析した．分析対象のソフトウェアは自動車用エンジン制御システムであり，ソフトウェア規模は 50 万 LOC 以上である．

本実験では特に可変部分が多いと考えられる 2 つのエンジン制御システムを選定して実験対象とした．実験対象となる 2 つのエンジン制御システムは，自動車メーカ，エンジン形式および出荷地域（法規制）がすべて異なっている．また，コードクローン分析には独自に開発したパーサおよび市販ツール “CloneFinder” を用いた．

4.2 実験結果

4.2.1 システムレベル

まず初めに，製品 A および製品 B におけるシステムレベルでの共通性・可変性を分析した．システムレベルでの分析の目的は，共通性・可変性をおおまかに調べることである．もし，製品 A および製品 B の分析の結果，Type 1 クローンが大部分を占めていれば，少なくともこの 2 機種にはほとんど可変性がなく，共通コンポーネントで構成できると考えられる．逆に，製品間クローンがほとんど発見できなかった場合は，アーキテクチャ内に可変部が多くなると考えられる．

なお，本実験における Type 2 クローンと Type 3 クローンとの製品間クローンカバレッジ閾値は 20%とした．この閾値は，エンジン制御ドメインにおいて共通性が失われる製品間クローンカバレッジを，専門家の判断によって決定した．

図 10 に，システムレベルでのクローン分布分析結果を示す．製品 A から製品 B への Type 1 クローンの行数が，製品 B のソースコード行数の 9%を占める．同様に，Type 2 クローンは 19%，Type 3 クローンは 15%，Type 4 クローンは 2%を占めた．クローンとして分類されなかった関数の行数は，製品 B のソースコード行数の 55%を占めた．

上記の結果より，製品 A と製品 B の実装には共通部分と可変部分とが混在していることが分かる．より

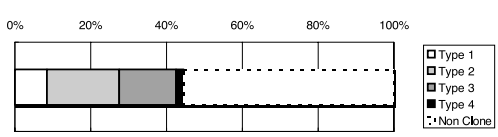

図 10 システムレベルでのクローン分布
Fig. 10 Clone distribution on system level.

詳細に製品の共通性・可変性を分析するため，サブシステムレベルの分析を実施する．

4.2.2 サブシステムレベル

本項では，サブシステムレベルでの共通性・可変性分析結果を示す．サブシステムレベルでの分析により，エンジン制御システムを構成する機能ごとの共通性・可変性を知ることができる．

サブシステムレベルでの分析にあたり，参照アーキテクチャへのソースコードのマッピングを行った．**図 11** に本実験で用いた参照アーキテクチャを示す．たとえば電圧計測等，エンジン制御システムでの入出力に関する関数群は図 11 での I/O サブシステムにマッピングされる．本実験では参照アーキテクチャは独自に開発せず，車両制御ソフトウェア標準仕様として議論されている AUTOSAR [21),22)] 仕様を簡略化したものを用いた．これは，将来的に車両制御ソフトウェアを AUTOSAR に移行させる可能性があることが理由である．

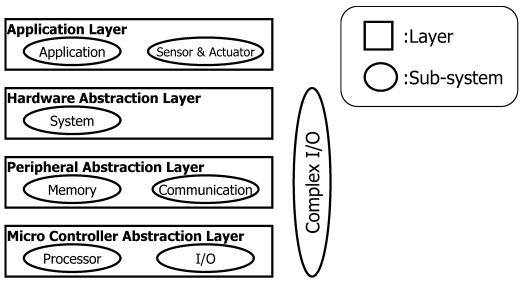

図 11 エンジン制御システムの参照アーキテクチャ
Fig. 11 Reference architecture of engine management systems.

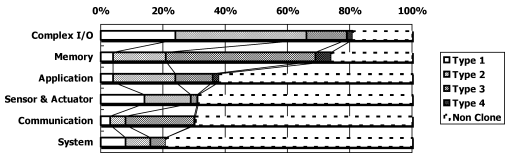

図 12 サブシステムレベルでのクローン分布
Fig. 12 Clone distribution on sub-system level.

図 12 に，サブシステムレベルでのクローン分布を示す．共通性・可変性分析の一例として，コンプレックス I/O（Complex I/O），メモリドライバ（Memory）および制御アプリケーション（Application）サブシステムの共通性・可変性および SPL 化手順を分析する．なお，コンプレックス I/O は文字どおり，クランクセンサ入力処理や燃料噴射タイミング制御等の複雑な入出力処理を扱うサブシステムである．

- メモリドライバ（Memory）
  メモリドライバは，約 5%の Type 1 クローンを持ち，約 50%の Type 3 クローンで構成されていることが分かる．これは，メモリドライバの主機能が外部不揮発メモリの制御であり，操作手順が実装されるハードウェアによって異なるためであると考えられる．その一方，レガシーシステムはある共通したフレームワークを用いて実装されており，サブシステムを構成する関数の内部が異なる実装であると推測される．このため，メモリドライバは既存のフレームワークおよび関数を用いて SPL 化が行え，かつ多くの関数はサブシステム内部の可変部分となってメモリごとに対応すると考えられる．
- コンプレックス I/O（Complex I/O）
  コンプレックス I/O は 25%の Type 1 クローンにより構成されている．この結果は，コンプレックス I/O の共通度が高いであろうという事前の予測とよく一致した．コンプレックス I/O はその複雑さから，なるべく既存のコードに基づいて開発が進められている部分である．しかし，コンプレックス I/O は 35%の Type 2 クローンも含んでいる．ここから，SPL 導入にあたっては Type 2 クローンの実装状況をよく吟味し，共通部分となる Type 1 クローンとサブシステム内の可変部分となる Type 3 クローンへのリファクタリングが必要であると考えられる．
- 制御アプリケーション（Application）
  制御アプリケーションの分析状況は事前の予想とは異なる結果となった．制御アプリケーションは，ハードウェアへの依存性が低いサブシステムのため 30%から 40%を Type 1 クローンが占めると予想されていたが，実際はわずか 5%のみが Type 1 クローンに分類された．そこで制御アプリケーションの実装状況をより詳細に調査するため，次項において分割を詳細化して共通性の分析を行う．

4.2.3 コンポーネントレベル

続いて，Application サブシステムを構成するコンポーネントの共通性・可変性を分析した．**図 13** に，コンポーネントレベルでのクローン分布を示す．

以下，各コンポーネントごとの共通性・可変性分析の一例を示す．

- アイドル回転数制御機能（Idle Speed Control）
  アイドル回転数制御機能は，エンジン制御システムにおける基本機能の 1 つである．図 13 に示す分析結果から，Type 1 クローンと Type 2 クローンの合計が 50%近くに達していることが分かる．

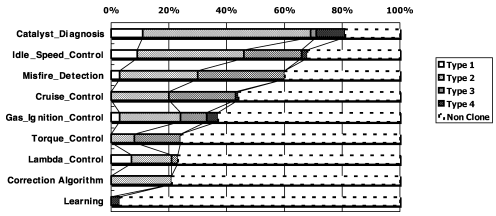

図 13 コンポーネントレベルでのクローン分布
Fig. 13 Clone distribution on component level.

この結果から，レガシーシステム間で相当部分の共通性があることが推測できる．この場合，製品間の共通部分を再利用・統合して，共通コア資産を開発する．

- 学習機能（Learning）
  図 13 に示す分析結果から，対象製品間ではクローンが数%しか発見されず，実装コードにはほとんど共通性がないことが分かる．これは，学習機能は顧客や対応エンジンに依存した機能が多いためであると考えられる．このような場合は，学習機能は製品ごとにコンポーネント全体を変更する可変部分として，SPL 化する．
- 燃料噴射量演算機能（Gas Injection Control）
  燃料噴射量演算機能は，エンジン制御システムにおいて基本となる制御機能の 1 つである．しかし，図 13 に示すように実装の共通性は低い．要求機能の共通性は高いにもかかわらず共通性が低い理由として，燃費改善や環境規制への対応等，つねに機能の拡張が行われていることが考えられる．このような場合には，要求の分析を行ったうえで，共通要求機能に対しては最新の実装ソフトウェアをもとに共通コンポーネントへとリファクタリングを行う．

4.3 実験結果に対する考察

既存の自動車用エンジン制御システムに対して提案手法を適用して，レガシーソフトウェアの共通性・可変性分析を行い，提案手法の評価を行った．階層的な分析により，製品全体の共通性から詳細部分の共通性へと効率良く分析が行えることを確認した．また，製品間クローンカバレッジに基づいてレガシーソフトウェア間のクローンを分析することにより，製品間の共通性・可変性を発見できることを確認した．さらに，製品間クローンカバレッジにタイプ別分類を導入することによって共通性・可変性分析を支援できることを確認し，本手法の有効性を確認した．

また，エンジン制御システム群は機能要求の観点から見るとかなりの部分が重複しているにもかかわらず，実際のソースコードが重複している部分はわずか 28%（Type 1，Type 2 クローン合計）しか測定されなかった．このことから，同一組織において開発される同一ドメインのソフトウェアであっても，エンジンの型式，顧客，対応地域等の違いによって可変部分が多く発生することが分かった．

5. 関連する研究との比較

組み込みシステムを対象としたプロダクトラインの事例研究がある[10)~12)]．しかし，これらの研究では，機能要求，バリエーション管理およびアーキテクチャ設計を対象としており，実績あるレガシーシステムを活用するという観点から見て十分ではない．また，Kang らはフィーチャ指向に基づいて，レガシーシステムをプロダクトライン化する手法を提案している[23)]．彼らの研究の主な目的は，プロダクトライン導入のためのアーキテクチャの改良である．単一のシステムのソースコードにおける関数間の依存関係から実装されているアーキテクチャを回復させ，SPL の観点からアーキテクチャを改良している．しかしながら複数のバリエーションを有するレガシーソフトウェアのリファクタリングには十分でない．さらに，Knodel ら[24)] および Arciniegas ら[25)] は，既存システムの SPL への統合を目的として，既存システムに実装されたアーキテクチャを復元して参照アーキテクチャと比較する手法を提案している．しかし，これはコンポーネント間のインタフェースの検討にとどまり，実装されているソースコードの再利用を支援するには不十分である．

これに対して我々の研究は，製品間クローンカバレッジを用いてレガシーソフトウェアの共通性・可変性を分析するという特徴がある．本手法は，従来多大な工数を必要としていた共通性分析フェーズを自動化することが可能であり，特に既存のソースコードの信頼性を重視する組み込み制御システムに対する SPL 導入段階でのデータマイニング手法として有効である．

クローン分析によるソフトウェア・システムのリファクタリング分析の研究がある[26),27)]．しかし，これは単一製品内の重複クローンを分析するものであり，複数製品間での可変性を分析するという目的に対しては適用できない．Balazinska らはクローンを複数のタイプに分類してリエンジニアリングの可能性を分析している[28)]．しかし，同一製品でのバージョン間のクローンを分析するものであり，異なる製品間の可変性を分析するものではない．また，Clements ら[29)] および Krueger ら[30)] は，既存 SPL 内のバリエーションポイント間でのクローンを分析し，アーキテクチャの改

善を行っている．本手法により，コンポーネント間のコードの重複から共通部を抽出し，効果的な SPL のメンテナンスが可能になる．しかし，この手法は成立済みのプロダクトラインへの適用を前提としており，レガシーソフトウェアへの SPL 導入を支援するものではない．また，山本らは異なるシステム間の類似度メトリクスを定義している[31]．しかし，システム全体での類似度の定義にとどまっており，システムを構成するコンポーネントごとの共通性・可変性を分析するには不十分である．

これに対して我々の手法は，異なるレガシーソフトウェアの共通部と可変部とを階層的に分析できるという特徴がある．本手法により大規模・多品種のソフトウェア製品群をシステマチックに分析可能となり，特に組み込み制御システムのように同一製品群内に多数のバリエーションを有するソフトウェアに対して有効である．

6. 今後の課題

本論文では，SPL の導入支援を目的として既存のソースコードに潜在的に存在する共通性・可変性を分析する手法を提案したが，参照アーキテクチャはドメイン知識を持った専門家により開発されたものを用いた．実装されている関数間の依存関係の解析手法と組み合わせることで，レガシーソフトウェアの SPL 導入における参照アーキテクチャ設計を効率化できる可能性がある．

提案手法は，参照アーキテクチャへのマッピング，クローン判定の閾値の決定，階層的なクローン分布の測定結果をふまえた共通性・可変性の決定について，専門家による作業に依存している．特に，クローン分布測定結果の解釈には対象ドメインの知識が必要であり，専門家の十分な支援が受けられない場合に，階層的な分析をどこまで深く行うかの判断が困難となる．そのため，対象ドメイン別の事例収集およびクローン分布のパターン分類が必要である．

本論文では共通性・可変性の分析に着目して提案を行ったが，分析結果に基づいた効率的な統合手法も必要不可欠である．特に，Type 2 クローンのように共通部分と可変部分とが混在している関数を多く持つサブシステムおよびコンポーネントへの手法提案が必要である．

また，我々は組み込み制御システム向けのモデルベース開発環境を開発している[13),32)]．本論文で提案した手法を適用してレガシーシステムの SPL 化を進めるとともに，モデルベース開発環境への統合を進めたいと考えている．さらに，モデルベース開発環境においてシステム統合およびコンポーネントの統合バリエーション管理およびを行う機能が必要であると考えており，今後研究を進めていきたい．

7. おわりに

本論文では，SPL 導入に向けたレガシーソフトウェアの共通性・可変性分析手法を提案した．レガシーシステムに基づいてコア資産を開発するにはソースコード解析が有効である．本論文では，異なるレガシーソフトウェア間でコードクローン解析を行うとともに，参照アーキテクチャを利用して階層的な解析手法を提案した．提案手法により，実装されているレガシーソフトウェア間での共通部と可変部との分類を効率的に行うことができ，信頼性の高い既存のソースコードを活用した SPL 導入を支援できる．また，提案手法を実際の自動車用エンジン制御システムへ適用し，共通性・可変性の効率的な分析に役立てられる可能性があることを確認した．

今後の課題は，本手法により分析したレガシーシステムの SPL 化とモデルベース開発環境への統合である．

謝辞 本研究を進めるにあたり多大なるご支援をいただいた武蔵工業大学の横山孝典教授，（株）日立製作所オートモティブシステムグループの白石隆部長，根本守主任技師，三宅淳司主任技師，日立研究所の野木利治部長，守田雄一郎主任研究員，成沢文雄研究員，橋本幸司研究員に心より感謝いたします．

参 考 文 献

1) Bayer, J., et al.: A methodology to Develop Software Product Line, *Proc. 5th ACM SIGSOFT Symposium on Software Reusability* (*SSR'99*) (1999).
2) Bockle, G., Clements, P., McGregor, J., Muthig, D. and Schmid, K.: Calculating ROI for Software Product Lines, *IEEE Software*, Vol.21, No.3 (2004).
3) Clements, P. and Northrop, L.M.: *Software Product Lines: Practices and Patterns*, Addison-Wesley (2001).
4) Pohl, K., Bockle, G. and Linden, F.V.D.: *Software Product Line Engineering: Foundations, Principles And Techniques*, Springer-Verlag New York Inc. (2005).
5) Linden, F.: Software Product Families in Europe: The Esaps & Café Projects, *IEEE Software*, Vol.19, No.4 (2002).
6) 渡辺晴美：組み込みソフトウェア開発技術：4.

プロダクトライン開発技術，情報処理，Vol.45, No.7 (2004).
7) Ganesan, D., Muthig, D. and Yoshimura, K.: Predicting Return-on-Investment for Product Line Generations, *10th International Software Product Line Conference* (*SPLC 2006*) (2006).
8) Lee, J. and Muthig, D.: Feature-Oriented Variability Management in Product Line Engineering, *Comm. ACM*, Vol.49, No.12, pp.55–59 (2006).
9) 吉村健太郎：製品間を横断したソフトウェア共通化技術—ソフトウェアプロダクトラインの最新動向，情報処理，Vol.48, No.2, pp.171–176 (2007).
10) Ommering, R.V. and Bosch, J.: Widening the scope of Software Product Lines — From Variation to Composition, *Proc. International Software Product Line Conference* (*SPLC*) (2000).
11) Steger, M., et al.: PLA at Bosch Gasoline Systems : Experiences and Practices, *Proc. Intl. Software Product Line Conference* (*SPLC*) (2004).
12) Thiel, S. and Hein, A.: Modeling and Using Product Line Variability in Automotive Systems, *IEEE Software*, Vol.19, No.4 (2002).
13) 吉村健太郎，宮崎泰三，横山孝典：オブジェクト指向組込み制御システムのモデルベース開発法，情報処理学会論文誌，Vol.46, No.6 (2005).
14) Basili, V.R., Briand, L.C. and Melo, W.L.: How Reuse Influences Productivity in Object-Oriented Systems, *Comm. ACM*, Vol.39, No.10 (1996).
15) 門田暁人，佐藤慎一，神谷年洋，松本健一：コードクローンに基づくレガシーソフトウェアの品質の分析，情報処理学会論文誌，Vol.44, No.8 (2003).
16) Yoshimura, K., Bayer, J., Ganesan, D. and Muthig, D.: Starting a Software Product Line by Reengineering a Set of Existing Product Variants, *Proc. SAE 2006 World Congress: In-Vehicle Software Session*, Paper No.2006-01-1238 (2006).
17) Yoshimura, K., Ganesan, D. and Muthig, D.: Assessing Merge Potential of Existing Engine Control Systems into a Product Line, *3rd Int. Workshop on Software Engineering for Automotive Systems* (2006).
18) Yoshimura, K., Ganesan, D. and Muthig, D.: Defining a Strategy to Introduce Software Product Line Using the Existing Embedded Systems, *6th ACM & IEEE Conference on Embedded Software* (*EMSOFT06*) (2006).
19) van der Linden, F., et al.: Software Product Family Evaluation, *5th International Workshop on Software Product Family Engineering* (*PFE 2003*) (2004).
20) Baker, B.: A Program for Identifying Duplicated Code, *Proc. Computing Science and Statistics: 24th Symposium on the Interface* (1992).
21) Fennel, H., et. al.: Achievements and exploitation of the AUTOSAR development partnership, *SAE Convergence 2006*, No.2006-21-0019 (2006).
22) Official website of the AUTOSAR partnership. http://www.autosar.org/ (visited on 07 Mar. 2007).
23) Kang, K.C., Kim, M., Lee, J. and Kim, B.: Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets — A case study, *Proc. International Software Product Line Conference* (*SPLC*) (2005).
24) Knodel, J., John, I., Ganesan, D., Pinzger, M., Usero, F., Arciniegas, J.L. and Riva, C.: Asset Recovery and Their Incorporation into Product Lines, *WCRE '05: Proc. 12th Working Conference on Reverse Engineering*, Washington, DC, USA, IEEE Computer Society, pp.120–129 (2005).
25) Arciniegas, J.L., et. al.: Architecture Reasoning for Supporting Product Line Evolution: An Example on Security, *Software Product Lines: Research Issues in Engineering and Management*, chapter 9, pp.327–372, Springer Verlag (2006).
26) 肥後芳樹，植田泰士，神谷年洋，楠本真二，井上克郎：コードクローン解析に基づくリファクタリングの試み，情報処理学会論文誌，Vol.45, No.5 (2004).
27) Kolb, R., Muthig, D., Patzke, T. and Yamauchi, K.: A Case Study in Refactoring a Legacy Component for Reuse in a Product Line, *Proc. IEEE International Conference on Software Maintenance* (*ICSM'05*) (2005).
28) Balazinska, M., et al.: Measuring clone based reengineering oppotunities, *Proc. 6th International Software Metrics Symposium* (1999).
29) Clements, P.C. and Northrop, L.M.: Sailon, Inc. A Software Product Line Case Study, Technical Report CMU/SEI-2002-TR-038 (2002).
30) Krueger, C.W. and Churchett, D.: Eliciting Abstractions from a Software Product Line, *Proc. Int. Work. on Product Line Engineering The Early Steps* (*PLEES*) (2002).
31) 山本哲男，松下 誠，神谷年洋，井上克郎：ソフトウェアシステムの類似度とその計測ツール SMMT，電子情報通信学会論文誌 D-I，No.6 (2002).
32) 成沢文雄，納谷英光，横山孝典：組み込み制御シ

ステムのためのオブジェクト指向コード生成ツール，情報処理学会論文誌，Vol.46, No.5 (2005).

(平成 18 年 11 月 16 日受付)
(平成 19 年 5 月 9 日採録)

吉村健太郎（正会員）

1976 年生．1999 年早稲田大学理工学部機械工学科卒業．2001 年同大学大学院理工学研究科機械工学専攻修士課程修了．同年（株）日立製作所日立研究所入社．2005 年から 2007 年まで仏国 Hitachi Europe SAS 出向．組み込み制御システムの研究に従事．ACM，IEEE，SAE，日本機械学会各会員．

ダルマリンガム ガネサン

印国 Anna 大学修了，印国 Jawaharlal Nehru 大学修了．現在，独国フラウンホーファ協会実験的ソフトウェア工学研究所研究員．リバース・エンジニアリング，ソフトウェア経済学，プロダクトライン工学の研究に従事．

ディルク ムーティック

1997 年独国カイザースラウテルン工科大学修了．2001 年独国フラウンホーファ協会実験的ソフトウェア工学研究所入所．現在，同研究所ソフトウェア工学部長．2002 年独国カイザースラウテルン工科大学博士号取得．IEEE 会員．