特別研究報告

題目

類似プロダクト比較のためのディレクトリ構造の対応付け手法

指導教員
井上 克郎 教授

報告者
坂口 雄亮

平成 27 年 2 月 13 日

大阪大学 基礎工学部 情報科学科

平成 26 年度 特別研究報告

類似プロダクト比較のためのディレクトリ構造の対応付け手法

坂口 雄亮

内容梗概

ソフトウェアプロダクト開発において，再利用が盛んに行われている．バグが発見された場合，バグが含まれるプロダクトを再利用したプロダクトの修正を行わなければならない．しかし，開発管理が行われていない場合，再利用された部分を見つけ出すことが困難であり，プロダクトの比較が必要となる．プロダクトの 1 つの機能は，1 つのディレクトリにまとめられることが多いため，プロダクト比較はディレクトリ単位で行うことが効率的である．そのため，類似プロダクト間のディレクトリ構造を対応付ける必要があるが，ディレクトリの移動やリネームがあった場合，対応付けは困難である．

複数の類似プロダクトの比較を支援するために，複数プロダクトの各ディレクトリ構造を自動的に 1 つのディレクトリ構造に対応付ける手法を提案する．類似したファイルをもとに，各プロダクトのディレクトリを対応付けてまとめる．そのため，ディレクトリの移動やリネームがある場合でも対応付けることができ，ディレクトリ単位での比較を行うことが可能となる．ケーススタディの結果，ディレクトリの移動やリネームがあった場合でも，ディレクトリの対応付けを正しく行うことができた．しかし，対応付いていると考えられるディレクトリの対応が取れない場合や，ディレクトリ構造的に対応していないであろうディレクトリが対応付けられる場合もあったため，今後の課題として，ディレクトリ名やディレクトリ構造を考慮した対応付けが，今後の課題として必要である．

主な用語

ディレクトリ
クラスタリング

目 次

1　はじめに

ソフトウェアプロダクト開発において，開発のコストを削減するためにソースコードの再利用が盛んに行われている．再利用は，既存のプロダクトを複製して改変する派生開発やライブラリの取り込みといった形で行われる事が多い．[1] このため，あるプロダクトのソースコードにバグが発見された場合，バグが含まれるプロダクトを再利用している他のプロダクトの修正もしなければならない．しかし，どのプロダクトからどの機能を再利用したかということや，どのプロダクトから派生されたか，というような開発管理がしっかりと行われていない場合，プロダクト間で共通する機能や追加された機能を調べるために，プロダクトの比較が必要となる．

Jonge が述べている [2, 3] ように，一般に，プロダクト内の 1 つの機能は，1 つのディレクトリにまとめられていることが多いため，複数の類似プロダクトにおいて，ディレクトリ単位で比較することが効率的であるといえる．Duszynski らは，複数のソフトウェアの対応したディレクトリ間の比較を行った．[4] 彼らはディレクトリ内のファイルを用いてディレクトリ間の類似度の可視化を行い，ソフトウェアの再利用の可能性を示している．しかし，彼らも述べているように，プロダクトの派生などによるディレクトリの移動やリネームが起こった場合，プロダクト間で対応したディレクトリを見つけ出すことは困難である．

類似プロダクトのディレクトリの対応付けは，Holten ら [5] によって行われているが，既存研究では，複数の類似プロダクトの対応付けはなされていない．

そこで，本研究では，複数の類似プロダクトの比較を支援するために，複数プロダクトのそれぞれのディレクトリ構造を 1 つの擬似的なディレクトリ構造に自動的に対応付ける手法を提案する．これにより，ディレクトリの移動やリネームが行われた場合でも，複数のプロダクトを比較を行うことが可能となる．例えば，バグ修正などにより，あるプロダクトのソースコードの変更があった場合，対応するディレクトリを見ることで，別のプロダクトのソースコードの変更も容易になる．対応付け手法の主な考え方は，類似したファイルを基に，ディレクトリを対応付ける．対応付けたまとまりを 1 つのディレクトリとみなして，親子関係を設定することで，1 つのプロダクトを構成するようなディレクトリ構造を構築する．

以降，第 2 章では，本研究の背景について述べる．第 3 章で提案手法の説明をする．第 4 章では，手法に基づいたツールによるケーススタディとその考察を行う．最後に第 5 章で，まとめと今後の課題について述べる．

2　背景

2.1　類似プロダクトの開発

プロダクト開発において，既存のプロダクトの再利用がなされている．具体的には，別のプロダクトのある機能のソースコードを再利用し，既存のプロダクトに組み込んだり，既存のプロダクトをそのまま再利用し，機能の拡張や修正を行い派生として新しいプロダクトを開発される．[1, 6] このように，再利用することにより，既存部分を新たに開発するコストの削減をすることができる．

プロダクトの開発をプロダクト全体で共通な共通性と，各製品で取捨選択する機能である可変性に分けて開発管理するソフトウェアプロダクトライン開発という概念がある．[7, 8] これの利点として，開発のコスト削減だけでなく，プロダクトの品質の向上と，是正保守の効率の向上が挙げられる．ソフトウェアプロダクトライン開発を導入することで，重大なバグがコア資産に発見された場合，開発された製品全体の修正が適切に行われるため，製品の不具合の発生を未然に防ぐことができる．

しかし，必ずしもソフトウェアプロダクトライン開発のように，プロダクトの管理がなされた開発が行われるわけではない．[9] その場合，どのプロダクトからどの機能を再利用したかということや，プロダクト間の派生関係といったような管理がなされないまま，細かな改変がなされた類似したプロダクトが独立して多く生まれる．その結果，機能を拡張したり，再利用によってバグを取り込んだ場合の修正などのコストが増加するため，プロダクト間で共通の機能と変更が加えられた機能を明確にするためにプロダクトの比較を行う必要がある．

2.2　類似プロダクトの比較

類似したプロダクトの比較を行うためには，ファイル単位での比較を行うことが一般的である．

Yoshimura らは，プロダクト間でコードクローンが含まれるファイルを検出し，可視化を行った．[10] また大規模なプロダクトを比較する技術として，ファイルクローン検出がある．[11] この手法では，大規模なプロダクト同士を比較して，共通するファイルを高速に見つけ出すことができる．

2.2.1　プロダクトの機能とディレクトリ

Jonge は，プロダクトの 1 つのディレクトリは，1 つのコンポーネントを表現することが多いことを示した．[2] また，Java などの言語では 1 まとまりの機能を 1 つのパッケージに

まとめて扱うことができるが，これらもファイルシステムの上ではディレクトリとして表現される．機能拡張やバグ修正は，単一のソースファイルに対してだけでなく，同一のディレクトリ内にある複数のファイルを修正することで実現されることも多い．そのため，プロダクトの比較を行う際には，ディレクトリ単位でどのような機能が共通しているのか，どの機能に差異があるかを見ることが効率的であると言える．

2.2.2 ディレクトリ単位の比較

プロダクトのディレクトリ単位の比較について，Duszynski らの研究 [4] がある．この研究では，複数のプロダクトの対応しているディレクトリ間の比較を，内部ファイルのソースコードの差分をとることで行っており，特に 5 つ以上の対応したディレクトリ比較の結果の可視化を行う難しさとそれを解決する手法について述べている．

しかし，比較を行う前にあるプロダクトのあるディレクトリが，他方のどのディレクトリと同様の機能を実現するファイルを持っているか，というディレクトリ間の対応関係をとらなければならない．プロダクト派生において，ディレクトリの移動やリネームが行われた場合，ディレクトリの対応をとることは難しいということも述べられている．

2.3 ディレクトリ構造の対応付け

ディレクトリ構造の対応付けについては，Holten らの研究 [5] がある．この研究では，2 つのプロダクトのディレクトリ構造において，プロダクト間で対応しているディレクトリを見つけ出し，可視化する手法について述べられている．

また，ディレクトリの内容に着目した比較も行われている．Yoshimura らは，2 つの類似プロダクトをマージするために，コードクローン検出を利用してディレクトリの対応付けを行った．[12]

しかし，複数のプロダクトの対応付けは既存研究ではなされていない．

3　提案手法

本研究では，複数の類似プロダクトの比較を支援するために，複数プロダクトの各ディレクトリ構造を自動的に 1 つのディレクトリ構造に対応付ける手法を提案する．手法の概要は，類似したファイルをもとに，各プロダクトのディレクトリを対応付けてまとめる．まとめたものをディレクトリとみなして，ディレクトリツリーを作成する．これにより，ディレクトリの移動やリネームが行われた場合でも，対応しているディレクトリは 1 つにまとめることができる．作成されたディレクトリツリーを見ることにより，対応しているディレクトリのファイルがまとめられているため，再利用による脆弱性を解消したりすることが可能になる．

本手法は，複数プロダクトのソースコードを入力すると，それぞれのディレクトリ構造が対応付けられ，単一のディレクトリ構造を出力する．

提案手法は，以下の手順で実現される．

step1 ファイルを持つディレクトリのクラスタリング

各プロダクトのファイルを持つディレクトリを対象とする．ディレクトリ内のファイルを基に，各ディレクトリ間の類似度を計算する．類似度の高いディレクトリを集めたクラスタを構築する．

step2 ファイルを持たないディレクトリのクラスタリング

各プロダクトのファイルを持たないディレトリを対象とする．step1 で構築されたクラスタを基に，ファイルを持たないディレクトリのクラスタ構築を行う．

step3 ディレクトリ構造の構築

各プロダクトのルートディレクトリを含むクラスタが複数ある場合，1 つにまとめ，それをルートディレクトリクラスタとする．ルートディレクトリクラスタ以外の各クラスタについて，親ディレクトリに相当する親クラスタを設定する．

そして，クラスタをディレクトリとみなし，ディレクトリ構造を構築する．親子関係がループする部分がある場合，構築したディレクトリ構造に含まれないクラスタが存在する．それらを構築されたディレクトリ構造に接続する．

以下では各 step についての詳細を説明する．

3.1　step1 ファイルを持つディレクトリのクラスタリング

この step では，ファイルを持つディレクトリのクラスタリングを行う．

ファイル内容が類似しているディレクトリを対応しているとみなす．具体的には，ディレクトリ間の類似度がある閾値以上の場合対応しているとみなす．類似度は，ディレクトリ内のファイルを基に計算する．次に，類似度が閾値以上のディレクトリ同士をクラスタとして1つに集める．

ただし，この step において，空ファイルのみを持つディレクトリは扱わない．

3.1.1 ディレクトリ間の類似度

ファイルを持つディレクトリ間の類似度は Jaccard 係数を用いる．Jaccard 係数の値は，2つの集合の共通部分が大きいほど 1 に近くなり，共通部分が全くない場合は 0 となる．Jaccard 係数は類似文字列を検索する際に多く用いられている．以下に類似度の計算の方法を示す．

ディレクトリ a,b から全ファイルの行の集合 a_l, b_l を作成する．ここで行の集合とは，ディレクトリ内の全ファイルをテキストファイルとみなし，空白空行を除去し，行ごとに分割して，1 行を 1 つの要素とした集合である．ただし，要素の重複を許すものとする．次に，ディレクトリ a,b 間の類似度 $sim(a, b)$ は次の式で表される．

$$sim(a, b) = \frac{|a_l \cap b_l|}{|a_l \cup b_l|} = \frac{|a_l \cap b_l|}{|a_l| + |b_l| - |a_l \cap b_l|} \tag{1}$$

ここで，| 集合 X| は，集合 X の要素数を指す．

類似度の計算を，同じプロダクト間を除く，ファイルを持つディレクトリの全ての組み合わせに対して行う．

3.1.2 クラスタリング

次に，類似度がある一定の閾値 t 以上の場合，2 つのディレクトリを要素とするクラスタを作成する．ただし，既にどちらかのディレクトリがあるクラスタに属しているならば，クラスタリングされていないディレクトリは，そのクラスタにクラスタリングされる．また，どちらのディレクトリも既にあるクラスタに属しているならば，それぞれ別のクラスタの場合，2 つのクラスタを 1 つのクラスタにまとめる．

この手順で，どのクラスタにもクラスタリングされないディレクトリがある場合は，それのみを要素とする独立したクラスタを作成する．

step1 のクラスタリングの例を図 1，2 に示す．この例は 3 つのプロダクトのファイルを持つディレクトリの一部を表す．図 1 において，ディレクトリ間で辺が引かれているものは，類似度が閾値以上の組み合わせを表す．図 2 は，類似度によりクラスタリングされたまとまりを表す．A，B，C，E ディレクトリが 1 つのクラスタにまとめられ，ディレクトリ D，F は，独立したクラスタとなっている．ここで，プロダクト 1 のディレクトリ A，B とプロダ

クト 3 のディレクトリ E 間の類似度は，閾値未満であるが，ディレクトリ C を介して同じクラスタに属している．

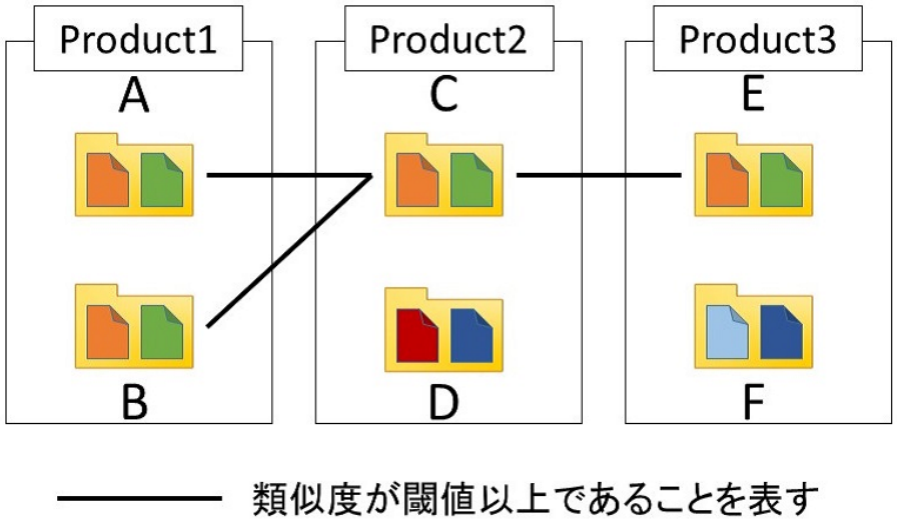

図 1: ファイルを持つディレクトリの類似関係

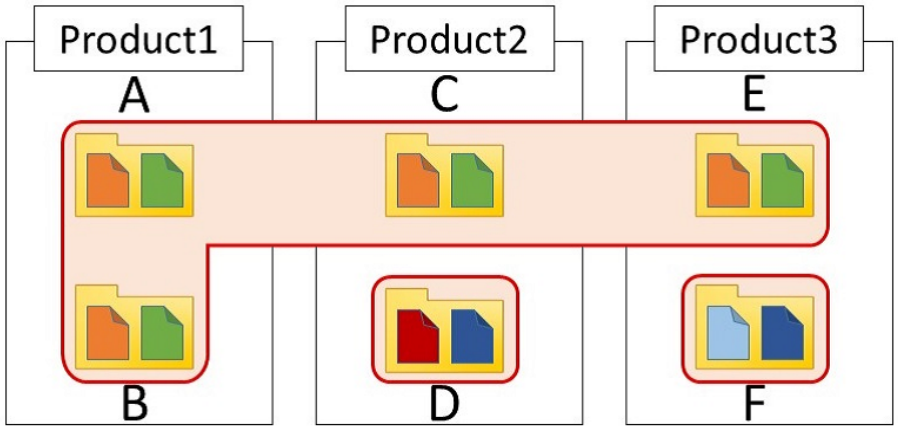

図 2: ファイルを持つディレクトリのクラスタリング例

3.2 step2 ファイルを持たないディレクトリのクラスタリング

この step では，ファイルを持たないディレクトリのクラスタリングを行う．
ファイルを持たないディレクトリは，ディレクトリ構造の末端にあるものと，階層途中に

あるものに分けられる．クラスタリングは，末端にあるディレクトリを先に行い，次に，階層途中のディレクトリを行う．

以下にそれぞれのクラスタリングについて説明する．

1. 末端ディレクトリのクラスタリング

   末端ディレクトリは，ファイルもサブディレクトリも持たない．そのため，親ディレクトリを基にして，ディレクトリを対応付ける．親ディレクトリが属するクラスタが同一の場合，ディレクトリが対応しているとみなし，それらを要素とするクラスタを作成する．例を図 3 に示す．

   また，図 4 のように，親ディレクトリのクラスタが他と異なる場合や，親ディレクトリがファイルを持たずクラスタが未定義である場合，それのみを要素とする独立したクラスタを作成する．

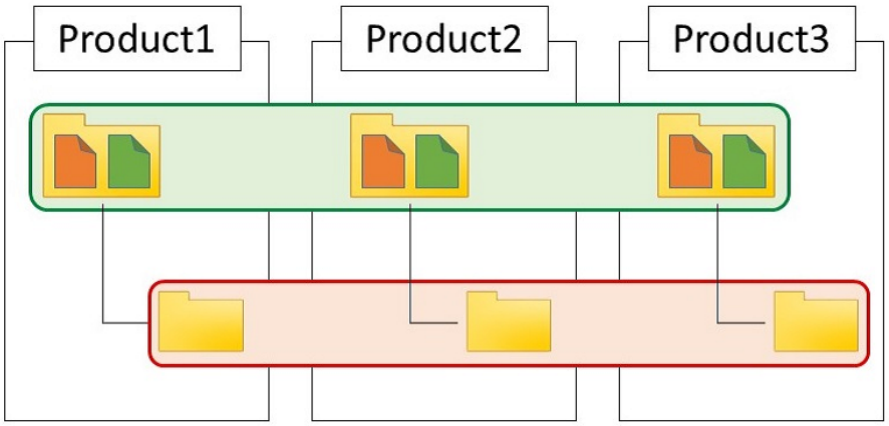

図 3: ファイルを持たない末端ディレクトリのクラスタリング例

2. 階層途中のディレクトリのクラスタリング

   サブディレクトリが属するクラスタが 1 つでも同一の場合，ディレクトリが対応しているとみなし，それらを要素とするクラスタを作成する．

   また，サブディレクトリが属するクラスタが全て異なる場合，それのみを要素とする独立したクラスタを作成する．例を図 5 に示す．

3.3 step3 ディレクトリ構造の構築

step3 では，1 つのクラスタを 1 つのディレクトリとしてみなし，ディレクトリ構造を構築し出力する．ディレクトリ構造を構築するため，まずルートディレクトリクラスタを設定

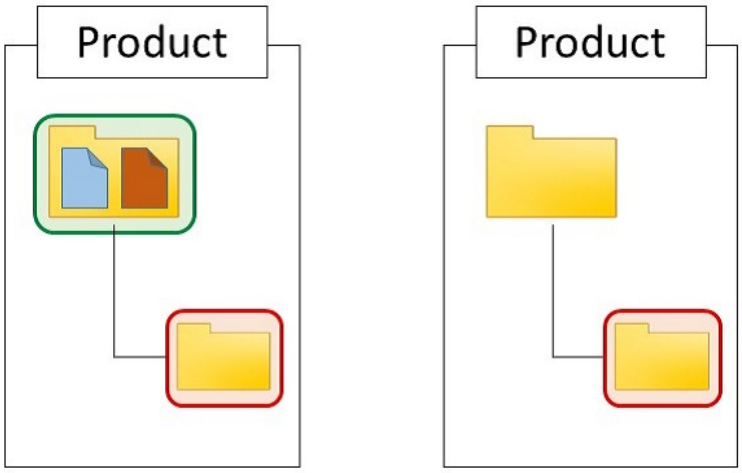

図 4: 独立したクラスタになる例

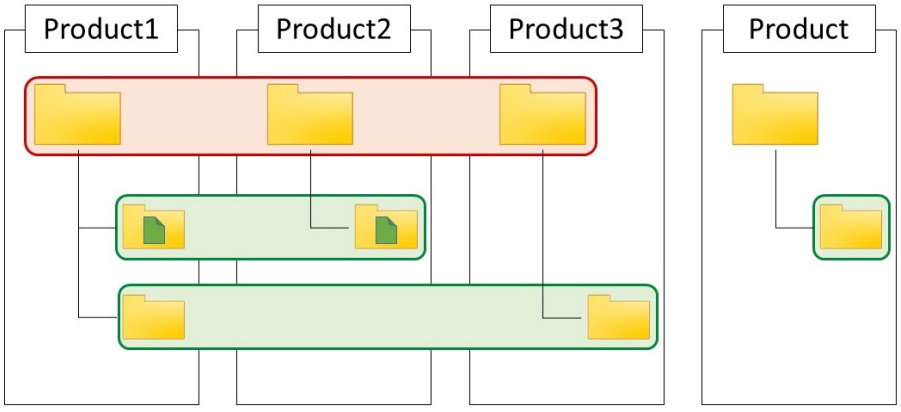

図 5: ファイルを持たない階層途中のディレクトリのクラスタリング例

し，それ以外のクラスタについては，親クラスタを設定する．そして，クラスタを親クラスタに接続することでディレクトリ構造を構築することができる．

3.3.1 ルートディレクトリクラスタ

ディレクトリ構造において，ルートディレクトリは，最上層に位置し，ただひとつである．これと同様にクラスタについても，ルートディレクトリに相当するルートディレクトリクラスタを設定する．

各プロダクトのルートディレクトリが含まれるクラスタを 1 つのクラスタにマージする．このクラスタをルートディレクトリクラスタとする．

3.3.2 親クラスタ

ディレクトリ構造において，ルートディレクトリ以外のディレクトリは，ただひとつの親ディレクトリをもつ．これと同じようにクラスタについても，ルートディレクトリクラスタ以外のクラスタ全てに，親ディレクトリに相当する親クラスタを設定する．

それぞれのクラスタについて，構成するディレクトリの親ディレクトリが含まれるクラスタを調べる．その中で一番多いものを親クラスタと設定する．ただし，複数の候補がある場合，親クラスタは入力順により 1 つに定まるものとする．

3.3.3 構築

まず，ルートディレクトリクラスタをルートディレクトリとみなす．次に，ルートクラスタを親クラスタとするクラスタをサブディレクトリとみなし，接続する．この手順を，再帰的に行うことでディレクトリ構造が出来上がる．

3.3.4 ループ構造の解消

本来，ディレクトリ構造は，ループをもつことはない．しかし，1 つの例として，図 6 のようなディレクトリ構造のプロダクト X，Y を入力した場合を考える．これらのディレクトリは，クラスタ A〜G にクラスタリングされ，親クラスタを設定し，ディレクトリ構造を構築すると，ルートクラスタ A を根とするディレクトリ構造と，ループ構造を含む構造ができてしまう．

未接続であるクラスタの集合をつくる．ルートクラスタを根とするディレクトリ構造について，図 7 のように，未接続クラスタがサブクラスタになりうるかどうかの探索を，深さごとに根から順に行う．ある深さ D に所属するクラスタが，未接続クラスタを構成するディレクトリの親ディレクトリが含まれるクラスタであった場合，その未接続クラスタをリスト

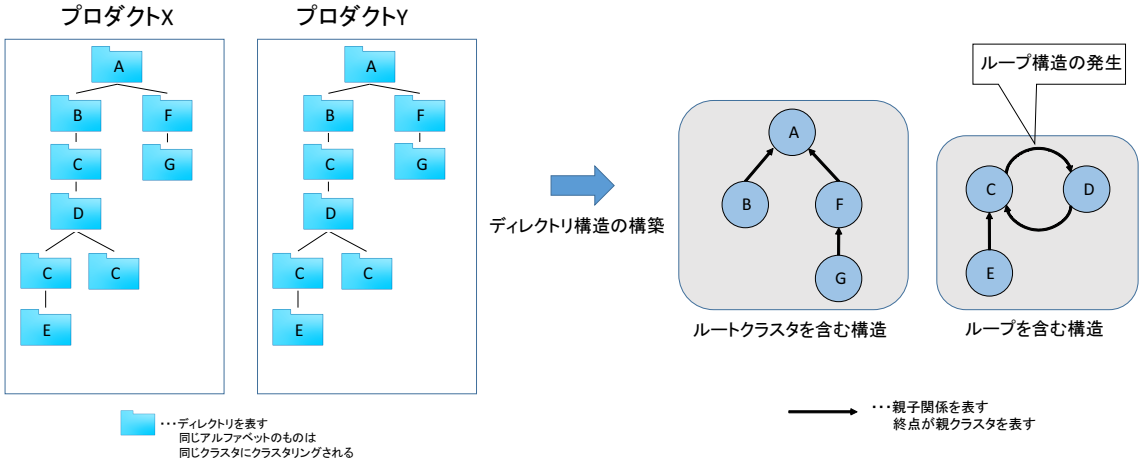

図 6: ループが発生する例とその構造

アップし，未接続クラスタの集合から除外する．図 7 の場合，深さ②において，未接続クラスタ C がクラスタ B のサブクラスタになりうるため，クラスタ C をリストアップする．

ある深さ D の探索を終えた後，リストアップされたクラスタに対して，順に以下の処理を行う．まず，深さ D の中で，どのクラスタが親クラスタとなるかよいかを決める．決め方は，構成するディレクトリの親ディレクトリが含まれるクラスタを調べ，その中で深さ D のクラスタを考える．深さ D の中で一番多いクラスタを親クラスタと設定し，接続する．ここでも，複数の候補がある場合，入力順により 1 つに定まるものとする．次に，未接続クラスタ内で，先ほど親クラスタに接続されたクラスタに接続されているものを接続していく．接続されたクラスタは，未接続クラスタの集合から除外する．ただし，親クラスタが何であるかは step4 で決めたものであり，また，リストアップされた他のクラスタは，再帰的に構築する処理では接続させない．図 8 の場合，リストアップされたクラスタ C について，親クラスタを深さ②のクラスタの中から選ぶ．この場合クラスタ B が親クラスタになるので接続する．そして，クラスタ C を親ディレクトリとするクラスタ D，E を未接続クラスタ集合から選び接続する．

この処理を，未接続クラスタ集合がからになるまで，深さ D を増やしながら続ける．

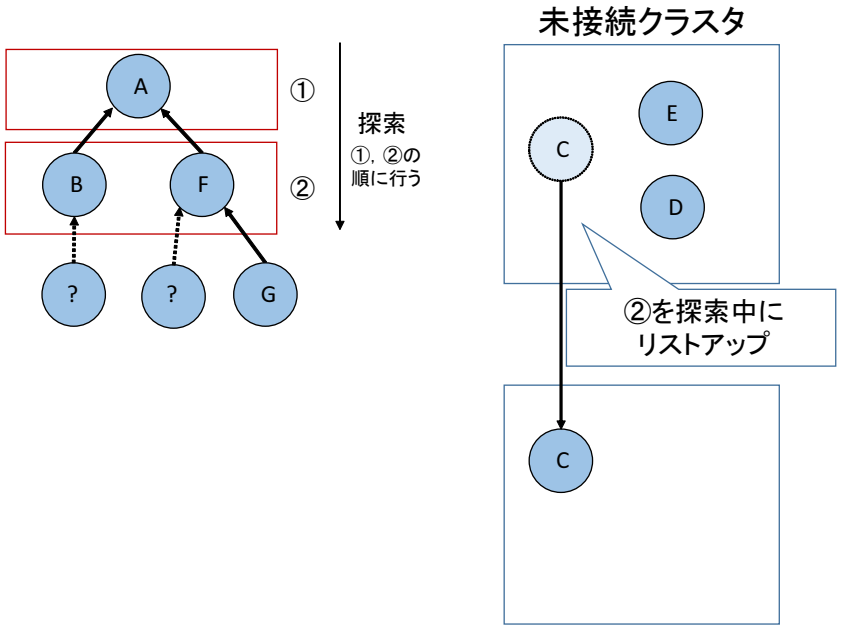

図 7: 探索とリストアップ

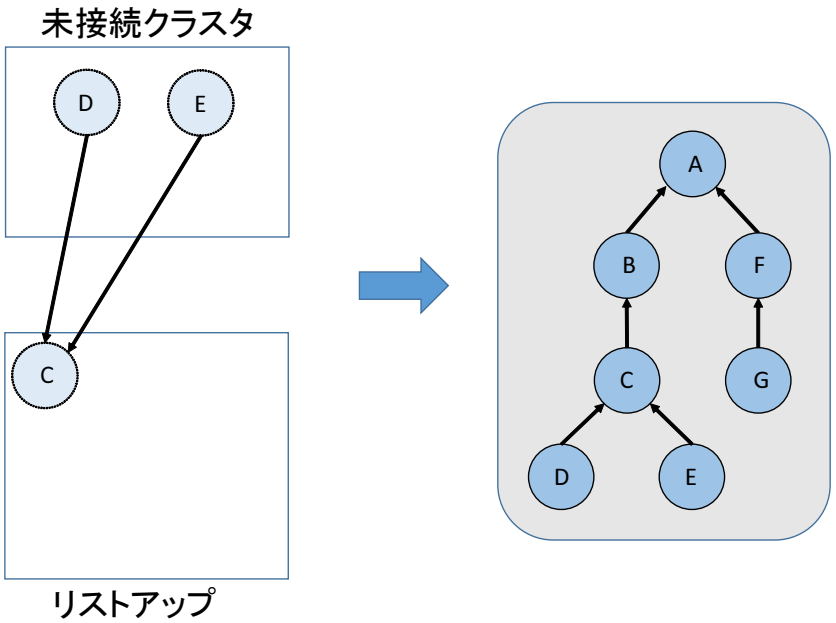

図 8: ループ構造の解消後の構造

4　ケーススタディ

提案手法をツールとして実装し，ケーススタディを行った．

4.1　ツールの説明

ツールは，1 クラスタを 1 ディレクトリとして出力する．また，クラスタに属するディレクトリがもつファイルを，出力するディレクトリにコピーを行う．この時，ファイル名と内容が一致する場合，コピーするファイル数は 1 つとする．ファイル名が一致するが内容が異なる場合，異なるファイルの数だけコピーを行う．

4.1.1　出力例

出力の例を，以下の図 9，10 に示す．図 9 は，出力されたディレクトリ構造の一部である．これは，入力したプロダクトの該当部分のディレクトリ構造と同じであった．また図 9 で示される boot ディレクトリの内容は図 10 のようである．出力されるファイルは，どのプロダクトからコピーされたかわかるようにファイル名の後ろにプロダクト名を付け足している．

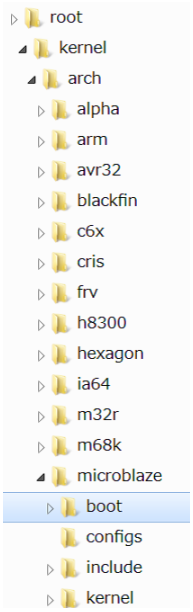

図 9: 出力されたディレクトリ構造の一部

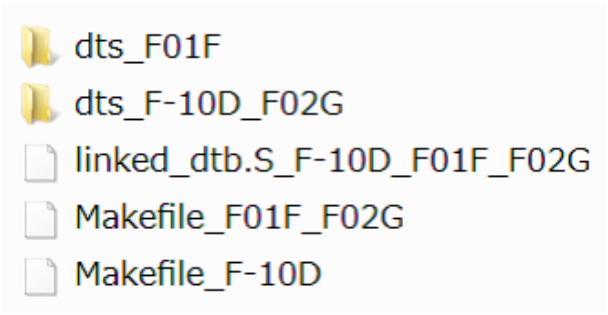

図 10: boot ディレクトリの内容

4.2　対象

ケーススタディの対象として，同一の製造者から発売された複数の Android 端末のソースコードを解析した．富士通が NTT DoCoMo 向けに製造した Android 端末のソースコード

から，発売時期の違う 3 つの端末のソースコードを公式 Web サイト [13] より取得した．入力の規模を表 1 に示す．ここでは，手法 step1 で用いる，ファイルを持つディレクトリ間が対応付いていることを表す閾値を 0.6 とした．

表 1: 入力した Android 端末

機種名	発売日	サイズ	総ディレクトリ数
F-10D	2012/ 7/20	1.33GByte	8,456
F-01F	2013/10/24	1.17GByte	7,528
F-02G	2014/11/19	2.13GByte	13,347

4.3 結果例

4.3.1 全プロダクトに共通のディレクトリ

全端末に共通のディレクトリとして，boot というディレクトリの例を示す．入力した 3 機種共にこのディレクトリは共通した構造位置にあり，ディレクトリ内容も共通であった．ディレクトリ boot の内容を図 11 に示す．対応付けられ出力されたディレクトリ boot の内容を図 12 に示す．これは，図 10 の再掲である．

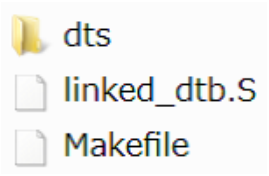

図 11: 共通のディレクトリの内容

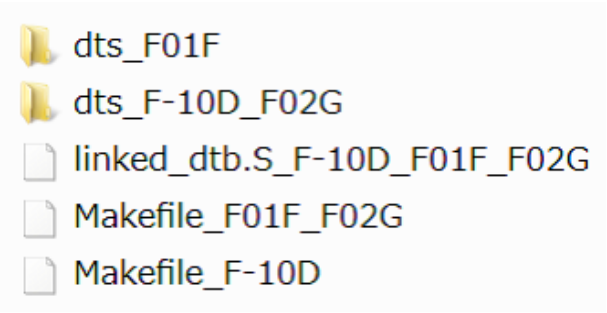

図 12: 出力されたディレクトリの内容

4.3.2 ディレクトリの移動

F-10D から F-01F の間でディレクトリ partitions の移動が見られた．F-10D のディレクトリ partitions 付近のディレクトリ構造を図 13 に，F-01F のディレクトリ partitions 付近のディレクトリ構造を図 14 に，対応付けられ出力されたディレクトリ partitions 付近のディレクトリ構造を図 15 に示す．

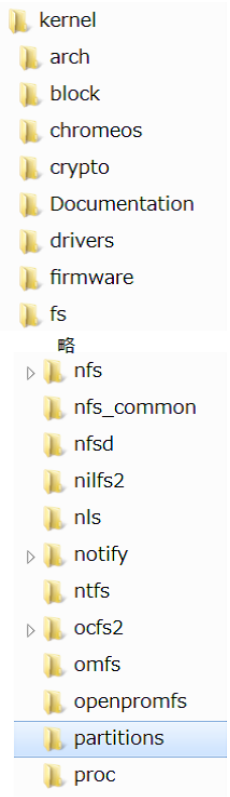

図 13: F-10D の partitions 付近のディレクトリ構造

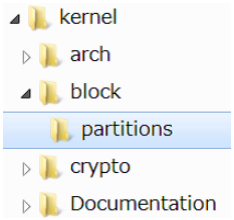

図 14: F-01F の partitions 付近のディレクトリ構造

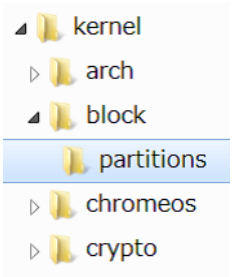

図 15: 出力された partitions 付近のディレクトリ構造

4.3.3　ディレクトリ名の変更

F-01F から F-02G の間でディレクトリ名の変更が見られた．F-01F の該当するディレクトリ名は ”VMCore ” であり，その親ディレクトリ llvm の内容を図 16 に示す．F-02G の該当するディレクトリ名は ”IR ” であり，その親ディレクトリ llvm の内容を図 17 に示す．また，対応付けられ出力されたディレクトリ llvm の内容を図 18 に示す．

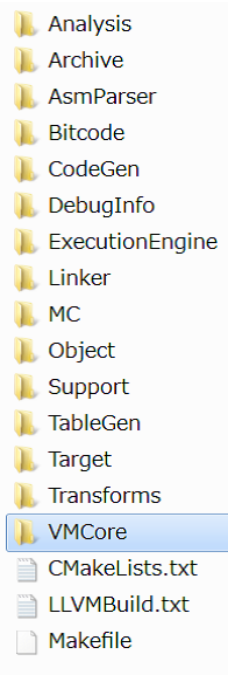

図 16: F-01F のディレクトリ llvm の内容

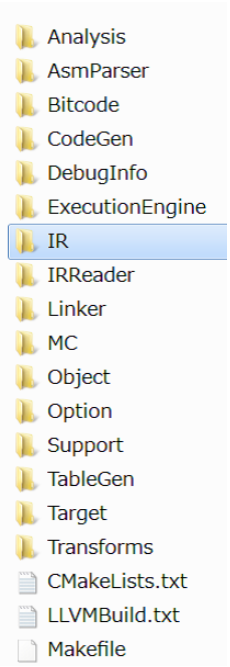

図 17: F-02G のディレクトリ llvm の内容

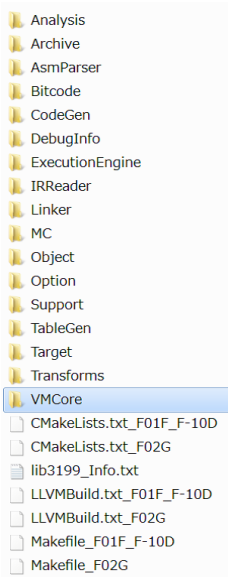

図 18: 出力された llvm の内容

4.3.4　ディレクトリ構造が異なる例

入力と出力のディレクトリ構造が異なることがある．一例として，F-10D のルート直下のディレクトリ構造を図 19 に，対応付けられ出力されたルートディレクトリ直下のディレクトリ構造の一部分を図 20 に示す．

4.4　考察

4.3.1 のように全体を通して，ファイルを持つディレクトリの対応付けが行うことができたことを確認した．4.3.1 の例では，linked_dtb.S というファイルは，3 機種とも共通で持っており，内容も同じであるため，出力では 1 つのファイルとなっている．また Makefike と

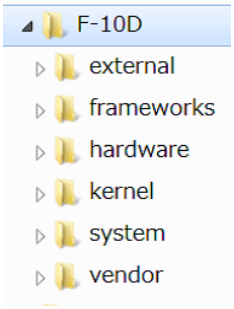

図 19: F-10D のルート直下のディレクトリ構造

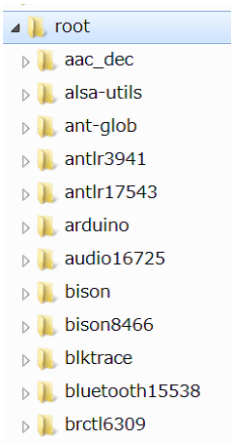

図 20: 出力されたルートディレクトリ直下のディレクトリ構造の一部分

いうファイルは，名前は共通であったが，F-10D のみ内容が異なっていたため，F-10D の Makefile と，F-01F，f-02G の Makefile の 2 つが出力されている

ディレクトリの移動が起こった場合でも対応付けを行うことができた．その出力例が 4.3.2 である．F-10D のディレクトリ partitions は，ディレクトリ kernel のサブディレクトリ fs のサブディレクトリである．一方，F-01F，F-02G のディレクトリ partitions は，ディレクトリ kernel のサブディレクトリ block のサブディレクトリである．それぞれのディレクトリ間類似度は，F-10D と F-01F，F-02G では 0.9 を超えた．F-01F と F-02G では，1.0 であった．そのため，この 3 つのディレクトリは対応していると言え，派生の中でディレクトリ移動が起こったと考えることができる．また，出力されたディレクトリ partitions は，ディレクトリ kernel のサブディレクトリ block のサブディレクトリであり，親子関係が多くとれている block を親ディレクトリとしていることもわかる．

リネームが行われた場合でも対応付けを行うことができた．その出力例が 4.3.3 である．F-01F のディレクトリ llvm にはサブディレクトリ IR はなく，また F-02G のディレクトリ llvm にはサブディレクトリ VMCore はない．リネームが行われているディレクトリ間の類似度は，F-10D と F-01F では，1.0 であった．F-10D，F-01F と F-02G では 0.8 を超えた．そのため，この 3 つのディレクトリは対応していると言え，派生の中でリネームが起こったと考えることができる．

しかし，対応付けがうまく行えないことも 4.3.1，4.3.4 の例からわかる．図 12 をみると，サブディレクトリ dts が 2 つ出力されている．F-01F のディレクトリ dts は他の 2 機種との類似度が閾値を下回ったためこのようなことが起きた．次に，ルートディレクトリクラスタは，22 のディレクトリから構成されていることがわかった．そのため，入力プロダクトの

ルートディレクトリは 6 個のサブディレクトリを持つが，出力されたルートディレクトリは，108 個のサブディレクトリを持つこととなった．これは，階層の深さが異なるファイルを持つディレクトリが対応付けられ，それに応じて階層の深さが異なるファイルを持たないディレクトリが順に対応付けられたため起こったと考えられる．

Android 端末を入力し，出力されたディレクトリ構造のディレクトリ数は，16,030 であり，そのうちファイルを持つディレクトリ数は 14,045 であった．3 機種の内最新の F-02G のファイルを持つディレクトリ数が 11,862 であることを考えると，全体としてディレクトリ構造が対応付けられたといえる．

5　まとめと今後の課題

本研究では，複数の類似プロダクトの比較を支援するために，複数プロダクトの各ディレクトリ構造を自動的に 1 つのディレクトリ構造に対応付ける手法を提案した．ケーススタディの結果，複数プロダクトのディレクトリの対応付けが行うことができた．ディレクトリの移動やリネームが行われた場合でも正しく対応をとることができた．しかし，対応していると考えられるが対応付けが行われていないディレクトリや，ディレクトリ構造的に対応していないと考えられるディレクトリが対応付けられているものもあった．

今後の課題として，ディレクトリの対応付けを，ファイルに基づいたものだけでなく，ディレクトリ名やディレクトリ構造を考慮して行う必要があると考えられる．

謝辞

本研究において, 常に適切な御指導及び御助言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 井上 克郎 教授に心より深く感謝いたします.

本研究において，随時適切な御指導及び御助言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 松下 誠 准教授に深く感謝いたします.

本研究において, 逐次適切な御指導及び御助言を頂き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 石尾 隆 助教に深く感謝いたします.

本研究において，様々な御指導及び御助言を頂き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 神田 哲也 氏に深く感謝いたします.

最後に，その他様々な御指導，御助言等を頂い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井上研究室の皆様に深く感謝いたします.

参考文献

[1] Naohiro Kawamitsu, Takashi Ishio, Tetsuya Kanda, Raula Gaikovina Kula, Coen De Roover, and Katsuro Inoue. Identifying source code reuse across repositories using lcs-based source code similarity. In *Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulation*, pages 305–314, 2014.

[2] Merijn de Jonge. Build-level components. *IEEE Transactions on Software Engineering*, 31(7):588–600, 2005.

[3] Merijn De Jonge. Multi-level component composition. In *Proceedings of the 2nd Groningen Workshop Software Variability Modeling*, number 2004-7, page 01, 2004.

[4] Slawomir Duszynski, Jens Knodel, and Martin Becker. Analyzing the source code of multiple software variants for reuse potential. In *Proceedings of the 18th Working Conference on Reverse Engineering*, pages 303–307, 2011.

[5] Danny Holten and Jarke J. van Wijk. Visual comparison of hierarchically organized data. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC Conference on Visualization*, pages 759–766, 2008.

[6] Julia Rubin, Andrei Kirshin, Goetz Botterweck, and Marsha Chechik. Managing forked product variants. In *Proceedings of the 16th International Software Product Line Conference*, pages 156–160, 2012.

[7] Charles W. Krueger. Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 282–293, 2002.

[8] 野中誠, 桜庭恒一郎, and 舟越和己. 組込みソフトウェア製品ファミリにおける是正保守の予備的分析. In **情報処理学会研究報告**, volume 2009-SE-166, pages 1–8, 2009.

[9] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. Managing cloned variants: a framework and experience. In *Proceedings of the 17th International Software Product Line Conference*, pages 101–110, 2013.

[10] Kentaro Yoshimura and Ryota Mibe. Visualizing code clone outbreak: An industrial case study. In *Proceeding of the 6th International Workshop on Software Clones*, pages 96–97, 2012.

[11] 佐々木裕介, 山本哲男, 早瀬康裕, and 井上克郎. 大規模ソフトウェアシステムを対象としたファイルクローンの検出. **電子情報通信学会論文誌** *D*, 94(8):1423–1433, 2011.

[12] Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig. Assessing merge potential of existing engine control systems into a product line. In *Proceedings of the International Workshop on Software Engineering for Automotive Systems*, pages 61–67, 2006.

[13] 携帯電話 (開発者向けサポート情報) FMWORLD.NET（個人） : 富士通. `http://spf.fmworld.net/fujitsu/c/develop/sp/android/`.