開発履歴に基づく GUI フレームワークとアプリケーション間のコードクローンの構築過程の調査

2008MI012 浅野貴之 2008MI244 瀧塚大樹 2009SE054 平野京志

指導教員: 横森励士

1 はじめに

ソフトウェアの保守工程においては，不具合の修正，新しい機能の追加，新しい環境への対応など，さまざまな修正がソフトウェアに加えられる．保守工程を困難にする原因のひとつとして，コードクローンが挙げられる．コードクローンとは全く同じ，あるいは一部が変わっただけの類似したコード断片の集合を指す．修正すべき箇所とコードクローンの関係を考慮して修正作業を行う必要が生じるので，保守作業が複雑になりやすく，実際の開発作業でも，定期的にコードクローンの除去が行われることも多い．GUI フレームワークに関係したアプリケーション内のクローン関係が実際の開発を通じてどのように生成され，解消されたかを調査した先行研究 [1] があるが，1 組のソフトウェアしか分析しておらず，その分析内容も不十分である．

本研究では，GUI フレームワークを利用する複数のアプリケーションに対して，GUI フレームワークに関連したアプリケーション内のクローン関係が開発を通じてどのように変化するのかを調査する．また，確認できたコードクローンを分類し，それらのコードクローンがどのように変化していくのかを分析する．これらの分析結果を元に，先行研究 [1] で得られた知見が妥当であるかを調査し，またバージョンの変化に伴ってコードクローンがどう解消されるのかをパターン化し，開発において注意するべき点などをまとめる．

2 背景技術

2.1 コードクローン

コードクローン [2] とは，ソースコードの中に散在する一致または類似する部分のことである．一から作るよりも，既存の確実に動作するプログラムの一部を書き写して作る方が，一時的には効率良く動作するプログラムを作成できる．これをコピーペーストプログラミングという．ただし，コピーペーストによるプログラムの作成は，保守性を著しく低下させる．同一処理の繰り返しがプログラム中に頻繁に現れると，そこにバグが発生した場合，全てのコードクローンを検査する必要が生じる．長期的に運用されている大規模なシステムにおいては，コードクローンの存在は時間とともに忘れられやすく，またその中のコードクローンをひとつひとつ調査するのは困難な作業となるので，多くの場合，コードクローンは除去されるべき存在であることが知られている．

2.2 コンポーネントグラフとクローン関係

ソフトウェアを構成するクラスがファイルを構成単位として考えると，ソフトウェアシステムは多数の部品で構成されていると言える．本研究ではソフトウェア内部に存在するクローン関係をコンポーネントグラフ上で表現する．コンポーネントグラフの頂点は部品を表現し，無向辺は両端の部品に類似したコード片が存在することを表す．

2.3 アプリケーションとフレームワーク間のクローン関係

フレームワークとはアプリケーションを開発する際に必要とされる汎用的な機能の集まりである．特定の機能を提供するフレームワークを用いることで，アプリケーションを容易に構築することができる．

ソフトウェアをフレームワークで提供されている機能を用いて実現しようとするとき，フレームワーク側に存在する利用例を参考に，アプリケーション側の利用例とよく似たコードを用いて実現することがある．この場合，図 1 のような形でフレームワークとアプリケーション間のクローン関係が生成される．もう 1 種類のフレームワークに関連したクローンの例としてフレームワークを利用しているコードを似た場面で同じように利用することがある．この場合，図 2 のような形のクローン関係がアプリケーション側の部品間で生成される．本研究では，これらの 2 種類のクローン関係に注目し，フレームワークに関連するクローンがどう変化していくか分析する．

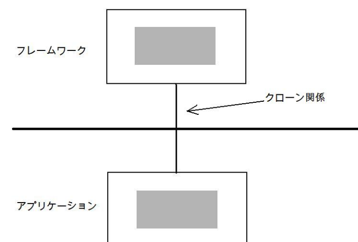

図 1 フレームワークとアプリケーションにまたがるコードクローン

2.4 先行研究

先行研究 [1] では，PetriNet のシミュレーションを行うアプリケーションである JARP について， GUI フレームワークである JHotDraw に関連したクローン関係を，バージョンごとに調査し，どのように変化しているのか分析し

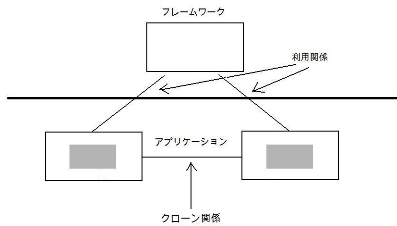

図 2 フレームワークの利用に関係したアプリケーション間のコードクローン

た．その結果，以下の 2 種類のコードクローンの存在を確認した．

- フレームワークからのコピーで生じるクローン
- 図 3 のように，フレームワークを利用する部分をクラスとして分離したり，継承関係でまとめた上で，それを利用する定型処理の部分が類似しているクローン

また，分析結果から次のようなコードクローンが存在すると推測されたが，そのようなコードクローンは発見できなかった．

- 図 4 のように，フレームワークを利用する部分が整理されていない状態でコピーされてアプリケーション内で生じるクローン

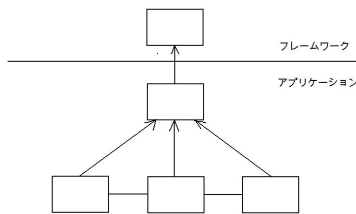

図 3 フレームワークを利用する部分が整理された状態でそれを利用する定型処理の部分が類似しているクローン

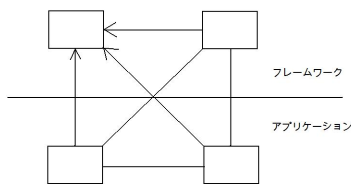

図 4 フレームワークを利用する部分が整理されていない状態でコピーされて実現するクローン

2.5 先行研究の問題点

先行研究 [1] ではひとつの事例しか調べておらず，十分な数のデータが採れていないので，普遍性という意味では信憑性は低い．複数のソフトウェアを対象に分析を行い，同じような傾向が得られるか調査しなければいけない．また，実際に存在が確認できた 2 種類のコードクローンの中間の状態があることが推測されているが，実際に存在するか検証する必要がある．

3 コードクローンの構築過程の分析

3.1 分析の目的

本研究ではフレームワークを利用するアプリケーションの間で，フレームワークに関連したクローン関係がどのようにソフトウェア内部で成長するのかを調査する．本研究では先行研究 [1] と同一の GUI フレームワークを分析対象とし，先行研究 [1] での分析結果との比較を行う．また，フレームワークを利用する部分が整理されていない状態でコピーされて実現するクローンが実在するかどうかを確認する．

GUI フレームワークは他のフレームワークに比べてプログラムの中で占める規模が大きく，さらに GUI フレームワークは使い方が同じ機能が複数あるので，コードクローンが多数あると推測される．複数のアプリケーションを対象に分析を進め，得られたクローン関係を分類し，その結果をもとに，バージョンの変化に伴って生成されたコードクローンがどのように解消されるのかをパターン化し，コードクローンの解消方法など開発において注意するべき点をまとめる．実際の開発におけるコードクローン除去のノウハウに関する知見を得る．

3.2 分析方法

実際のオープンソースソフトウェアの開発履歴を対象に，それぞれのバージョンにおけるコードクローン情報を抽出し，それらがどのように推移するかを分析する．コードクローンの抽出においては CCFinder を，結果の分析には Gemini[3] を用いる．CCFinder はコードクローン抽出ツールで，ソースコードをトークン単位で切り出し，一致するトークン列を計算することでコードクローンを抽出する．実験では 30 トークン以上一致したコード断片をコードクローンとみなした．

3.3 調査対象

本研究では，先行研究 [1] と同じ JHotDraw をフレームワークとして利用している Java アプリケーションを調査対象とした．JHotDraw は Java で書かれている構造化された図面エディタ用の 3D グラフィックスのフレームワークである．表 1 は調査対象となったアプリケーションの一覧で，横の数字はそれぞれのアプリケーションがリリースされた回数を示している．それぞれのアプリケーションは段階的に開発が行われており，それぞれのバージョンの

ソースコードを入手し，クローン関係の調査を行った．以下ではその分析の結果を報告する．

表 1　調査対象

アプリケーション名	ver 数
Renew	11
ChemSense	3
JStock	70

3.4　得られたクローン関係の分類

CCFinder を使用して Renew，ChemSense，JStock を調査したところ，複数のコードクローンが検出できた．本研究で検出されたコードクローンは 1 つのメソッドの中に 2 つ以上は存在しなかったので，コードクローンの名前はそれを含むメソッドの名前をそのまま使用している．さらにそれらを以下の A，B，C の基準で 3 種類に分類した．

A　フレームワークとアプリケーションの間にクローン関係を持つコードクローン

B　アプリケーション同士の間にクローン関係を持つコードクローン

C　A と B のクローン関係を合わせ持つコードクローン

この基準に従って，検出されたコードクローンを表 2，表 3，表 4 にまとめた．大部分のコードクローンが A，B に属しているが，C に属するクローンもいくつか存在していることがわかる．

A	B	C
setAttribute	accept	Vector handles
setParent	Vector handles2	FigureEnumeration get FiguresWithDependencies
createMenus	getTraceMode	
main	mouseUp	
createFontStyleMenu	createAddtionalTools	
drawFrame	updateText	
drawString	attachErrorFigure	
SearchReplaceFrame	processAttribute	
initComponents	File export	
DrawPlugin getCurrent	exportTextChildren	
	Drawing[] importFiles	
	CPNTextFigure	

表 2　分類したコードクローン：Renew

次に Renew から 2 つ，Chemsense から 1 つずつ得られた C のコードクローンについて，メソッド名と内容を以下に紹介する．

メソッド名：handles

内容：相対的な位置関係（北西，北東，南西，南東）を示す矢印を作る．Vector はインスタンスを格納するためのクラス．addElement は Vector クラスのインスタンス

A	B	C
setTool		draw
enable		
action Performed		
object clone		
selectGroup		

表 3　分類したコードクローン：ChemSense

A	B	C
initComponents	write	
	read	
	read2	

表 4　分類したコードクローン：JStock

(handle) に，何かしらのインスタンスを要素として格納（加える）していくメソッドである．ここでは Vector クラスのインスタンスを格納する．

メソッド名：getFiguresWithDependencies

内容：スーパークラスの getFiguresWithDependencies() は子との依存関係の図を出力するメソッドである．この getFiguresWithDependencies() は，親との依存関係も含めた図を出力するメソッドである．

メソッド名：draw

内容：r．x と r．y を始点として，右下に r．width と r．height だけ進み，この四角形の中を白で塗りつぶしたり，外周を黒で縁取りする．

3.5　コードクローンの解消方法の分類

確認したそれぞれのコードクローンについて開発を通じて解消されたかどうかを調査した．それぞれのアプリケーションの途中で解消されたコードクローンの数は以下の表 5 のようになった．

表 5　解消されたコードクローンの数

	A	B	C
Renew	7/10	3/12	0/2
ChemSense	0/5		0/1
JStock	1/1	1/3	

さらにこれをコードクローンの変化の内容ごとに分類した．

ア　全てのクローンに手が加えられている

イ　一部のクローンのみ変更されている

ウ　クローン関係だったコード断片の両方が無くなった

エ　クローン関係だったコード断片の片方が無くなった

ファイルの変更についてこのように分類した結果が，以下の表 6 である．今回の分析結果からは，全てのファイルに手を加えたり，ファイルの削除という形でクローンが解

消されるということは少なく，ほとんどのコードクローンが一部のファイルの変更によって解消されていたことがわかる．

	RenewのA	RenewのB	JStock
ア		updateText	read
イ	drawFrame accept setAttribute drawString	createAdditionalTools processAttribute	
ウ	createMenus		initComponents
エ	setParent main		

表 6　コードクローン解消法

コードクローンが解消された原因をそれぞれ調査したところ，大きく分けて 3 つにわけることができた．一つ目の例は，実行内容が変わったことによる解消で，例えば利用データの型の変更や，条件の追加や条件分岐先の追加，メソッド呼出の削除などが原因となっていた．二つ目の例は，実装方法自体が変更されたことによって解消された例が見られた．実装方法の変更により，記述すべきメソッドも変わりコードクローンを含む部分が削除された．最後の例は，記述スタイルが変わったことでコードクローンが解消された例が見られた．

4　考察

4.1　先行研究との比較及び考察

先行研究 [1] で存在が推測された，フレームワークを利用する部分が整理されていない状態でコピーされて実現するコードクローンは，実際に存在することが確認できた．しかし検出された数は少なく，フレームワークからコードをコピーすることと，既に出来ているクローンからコピーすることが同時に起きることは必ずしも多くないことがわかった．また，そのようなクローンにおいても途中の段階は現れなかった．このようなクローンは存在しないというわけではないが，実際に現れた場合には優先的に除去される対象となりやすいという可能性が考えられる．

4.2　コードクローンの解消に関する考察

実際に解消されたコードクローンの一部にしか手が加えられていないことが多く，コードクローンを考慮した検討がなされていないおそれがある例が見られた．フレームワークとアプリケーション間に存在するクローンについて考察すると，実行方法が変わった場合は，コードクローンを変更する際にフレームワークだけが変更されたならアプリケーションへ反映させるべきか検討しなければいけない．また，アプリケーションだけが変更されたのであればその変更自体が正しいかどうか確認しなければいけない．実装方法自体を変えた場合は，アプリケーション内部にクローン関係が依然として存在する可能性を考慮して検討を行う必要があると考えられる．

アプリケーション間に存在するクローンについて考察すると，コードクローンのもう片方に対する分析が必要不可欠である．クローンの一方にだけ手を加えると，機能が分化して一部だけが更新されたクローンができあがりやすい．このような不一致部分を含むクローンはギャップドクローンと呼ばれており，このようなコードクローンを検出するのは困難であることが多い．

分析においてはただ単純にコードクローンの数を分析するだけでは不十分で，このような形で検出されにくく，把握しにくくなるようにコードクローンが複雑化することで検出されなくなる場合も考慮しながら分析を行う必要があることがわかる．今回の結果から得られる指針として，実際の開発においてコードクローンの増減を評価する際は，ある部分が変更されたときに関連するコードクローンが同様に変更の検討がなされたかを合わせて分析する仕組みがより重要になると考えられる．

5　おわりに

本研究では，GUI フレームワークのひとつである JHotDraw を利用する 3 つのアプリケーションにおいて，GUI フレームワークに関連したコードクローンが，各バージョンにおいてどう存在しているのかを分析し，それらのコードクローンがどのように解消されたのか分類，考察した．分析結果からはフレームワークからのコードのコピーとアプリケーション側で実現したコードのコピーが連続して起こる事例はあまりなく，そのようなクローンは解消活動の対象になりやすいと推測した．またコードクローンに対する変更においては，クローン関係にある記述について変更が必要かどうか十分な検討を行うことが重要であることを確認した．

参考文献

[1] R. Yokomori, H. Siy, N. Yoshida, M. Noro, and K. Inoue, ”Evolution of Component Relationships between Framework and Application,” Journal of Computers, Computer Society of The Republic of China, Vol. 23, No. 2, pp.61-79, 2012.

[2] T. Kamiya, S. Kusumoto, K. Inoue, ”CCFinder: A multilinguistic token-based code clone detection system for large scale sourse code,”IEEE Transaction on Software Engineering, Vol. 28, No. 7, pp. 654-670, 2002.

[3] Y. Ueda, Y. Higo, T. Kamiya, S. Kasumoto, and K. Inoue: ”Gemini:Code Clone Analysis Tool”, Proceedings of 2002 International Symposium on Empirical Software Engine, vol.2, no. 386, pp.31-32, 2002.