ソフトウェア工学 115−19
（1997. 8. 1）

オブジェクトの進化モデル構築に向けて

中谷 多哉子† 玉井 哲雄†

オブジェクト指向システムは，保守性が高く利用者の要求を満足し続けることができると言われている．しかし，開発者は，要求変更に対応するためにクラスの作り替えや新しいクラスへ置き換えるといった設計変更を繰り返しながら，特定のクラスを再利用可能なクラスにするために，クラスの分割や統合，継承構造の変更といった設計変更を行っている．従来から，再利用可能なクラスについて，その定性的な特徴については評価されてきたが，本稿では，システムに定義されたクラスを3種に分類し，それぞれの種について，システムの成長に従って変化する様子を定量的に捉え，その再利用可能性について議論することを試みる．

調査の結果は，開発者や管理者が特定のクラスの再利用可能性を判断するための情報として役立てることができるだろう．

Towards Constructing an Object Evolution Model

TAKAKO NAKATANI † and TETSUO TAMAI †

Object-oriented systems are said to satisfy users by changing themselves without much developing efforts. But activities of reconstructing classes and replacing classes are still required to make a system robust and make classes reusable. Class reusability has been evaluated qualitatively in many works. We intend to quantitatively clarify class reusability by means of measuring evolution processes.

In this paper, classes are classified into three categories: the boundary species, the domain species and the common species. Observing class evolution processes for each species, we gather information of class evolution processes, which could help developers and managers find when classes are ready to be reused.

1. はじめに

ソフトウェアは利用者の要求に基づいて開発されるが，利用者の要求は，ソフトウェア導入の影響を受けて変化するという性質を持っている[2]．そのため，開発の初期に確定した要求だけを満たしたソフトウェアは，利用者を満足させ続けることはできない．そこで，継続的に開発を進める漸進的開発が推奨されるようになった[4]．漸進的開発では，保守を開発と区別する必要はなく，継続的な開発の一部と捉える．

オブジェクト指向システムでも漸進的開発は一般的に行われている[7]．しかし，開発の過程では，利用者要求の変化だけでなく，その後の再利用性にも配慮しなければならない．オブジェクト指向システムの再利用には次のような種類がある．

- システムを作り替える際に行われる，古いシステムに定義されていたクラスの再利用
- 同一の問題領域を対象とするプロジェクト間で行われるクラスの再利用
- 特定の問題領域に依存せずに行われるクラスの再利用

開発されたすべてのクラスが上記の再利用の対象となるわけではなく，クラスによっては短期間で新しいクラスに置き換えられて破棄されるものもある．このような議論は，これまで定性的になされてきた．我々は，特定のクラスが再利用可能か否かを判断するためには，再利用されるクラスと再利用されないクラスとで，変化の過程にどのような違いがあるかを定量的に調査する必要があると考える．

調査では，継続的な設計の変化をクラスという個体の進化と捉える．そして，システムに定義されたクラスの種類によってどのように進化の過程に差異

† 東京大学大学院総合文化研究科広域科学専攻
Graduate School of Arts and Sciences,
University of Tokyo
e-mail: {tina, tamai}@graco.c.u-tokyo.ac.jp

が生じるかを明らかにし，再利用可能なクラスの特徴を定量的に示す．

本稿は次の構成になっている．2. で，定量的な観測を行うためのメトリクスについて説明し，3. でシステム内で果たす役割に基づいて分類したクラスの種と，クラスに変化を起こす要因について議論する．4. で，クラスの種別ごとの進化過程を，事例システムで観測された結果を示しながら検討し，5. では，クラスの種に基づいて，クラスの進化過程がどのような段階を経て進むかをまとめる．

2. 観測の方法

2.1 メトリクス

システムに定義されているクラスの変化を定量的に計測するために，静的メトリクスと動的メトリクスを適用する[14]．静的メトリクスは，Chidamber らのオブジェクト指向設計メトリクス（以降 CK メトリクスと略す）に基づいて定義する[3]．ただし，クラスの複雑度を表す WMC (Weighted Methods Per Class) にはメソッド数を[1]，また，クラス階層の深さを表す DIT (Depth of Inheritance Tree) には，ライブラリクラスから計測するクラスまでの深さの値を用いる[12]．更に，CK メトリクスの他の静的メトリクスとして，クラスの行数とクラス木の数を追加する．

CK メトリクスのうちクラス間の結合度を表わす CBO (Coupling Between Object Classes) は動的メトリクスを適用した計測値から求める．動的メトリクスは，プログラム実行時に収集したプロファイルデータから，あるインスタンスが他のインスタンスに送ったメッセージの種類と回数をクラスごとに求めた集計結果に対して次のように定義する．

- 各クラスが受信したメッセージ名とメッセージの送信元のクラス名の組みに対する受信回数
- 送信したメッセージ名と受信先のクラス名の組みに対する送信回数

動的メトリクスを適用して計測した値から，クラス間の結合度 CBO の他に，計測対象のクラスの受動的，あるいは能動的な役割の特徴を求めることができる．

2.2 計測結果のまとめ方

過去の我々の研究では，静的メトリクスを用いて収集した計測が，いずれも小さな値に最頻値を持ち，右に尾を引く分布になることを明らかにした[12]．また，個々のクラスについては，クラス階層の組み替えなどの設計変更によって計測値の値が減少するという傾向も観測した[11),13]．

本稿ではクラスの複雑度 WMC に着目してクラスの変化とクラスの再利用性との関係について議論する．計測値をもとにクラスの変化率を求める際，たとえば第 i-1 版から第 i 版へシステムが成長するとき，クラスの複雑度の変化率 δWMC_i は，第 i-1 版のメソッド数 WMC_{i-1} と第 i 版のメソッド数 WMC_i を用いて，次の式で表す．

$$\delta WMC_i = \frac{WMC_i - WMC_{i-1}}{WMC_{i-1}}$$

3. クラスの種と変化の要因

3.1 クラスの分類

システムは利用者の要求を満足するために変化し，また将来の利用者要求に対応するために変化する．これらのシステムの変化はクラスの変化へ還元することができる．オブジェクト指向では，GUI を伴う多くのアプリケーションプログラムで MVC (Model-View-Controller) アーキテクチャ[6]に基づくフレームワークが採用されている[9]．このフレームワークが採用されている理由は，利用者の要求変更の 40%以上を占める[8]と言われている利用者インタフェースの変更に対して，特定のクラスだけで対処できる構造をアプリケーションプログラムに持たせることができるためである．MVC フレームワークの View クラスと Controller クラスは，利用者インタフェースの変更要求をこれらのクラスの変更によって対処する役割を持っている．

最近のフレームワークでは，View と Controller を 1 つのクラスとして定義する場合が多いが，特定の問題領域に依存せずに再利用できる部品として実績がある．Model クラスは，問題領域のクラスとして定義されるものであり，このクラスの中には同一の問題領域を対象とするプロジェクト間で再利用されるクラスも含まれる．そこで，以上の Model および View+Controller として分類されるクラスについて，システムの成長に伴う変化の様子を調査し，再利用可能なクラスと再利用の対象とはならな

いクラスとの変化の差を求めることにした．

調査に当たり，システムに定義されているクラスを，すでに再利用実績のある汎用クラス群と，View+Controller クラスおよび Model クラスを含むその他のクラスに分類し，それぞれのクラス群をクラスの種として定義する．以下に進化の調査を行ったクラスの種を示す．

- 汎用領域種：汎用クラスに相当するクラス群
- 境界領域種：View と Controller に該当するクラス群
- 問題領域種：Model に該当するクラスおよび上記以外のクラス群

3.2 クラスの変更要因

クラスの変更要因として，利用者の要求変更と，再利用性や拡張性，可読性の向上を目的とした設計変更を考えることができる．これらの要因を利用要因，開発要因と呼ぶことにする．

3.2.1 利用要因

利用要因には，利用者の要求変更の他に，システムを取り巻く他のシステムのインタフェースの変更などを含む．しかし，ハードウェア構成の変更や言語の変更は含まないことにする．ハードウェア構成や言語の変更がシステムに及ぼす影響は大きく，システムの進化や世代交代[10]として論じる規模になるためである．

3.2.2 開発要因

利用要因でクラスを変更する際，開発者は，再利用性や拡張性を意識してクラスを拡張／縮小したり，分割／統合したり，継承構造を変更したり，新しいクラスへ置き換えたりする．ひとつの要求変更に対して，これらの設計案からどの設計案を選択するかは開発者の意思で決まる．また，開発者は，システム開発を進めるに従って問題領域の理解を深め，その問題領域に固有の要求変更の特徴を認識し，それが次の設計案の選択を左右することもある．したがって，開発要因には，開発者が持っている設計に対する意図の他に，開発者のシステムに対する理解の深さを含む．

4. 進化の観測

3 つのシステムについて，システムの成長に従って変化するクラスの様子を，先に説明したメトリクスを適用して定量化した．調査では，必要に応じて開発者へのインタビューも行っている．

4.1 観測対象の概要

調査対象として選択した 3 システムについて，開発概要を以下に示す．システムの最終的な規模は，システムに定義されていたクラスの数，メソッドの数，行数の計測値と，それぞれの値に対する第 1 版との差を倍率で表した．

[システム A]
- 版数：4
- 計測期間：7 か月
- 開発者の人数：1
- 開発言語：Visual Smalltalk
- 開発プロセス：漸進型
- システム概要：シミュレーションを構成するためのエディタ
- システム規模：第 4 版のクラス数，メソッド数，行数はそれぞれ 52，927，8677 で，第 1 版のそれらの 3.25 倍，5.56 倍，4.80 倍であった．

[システム B]
- 版数：4
- 計測期間：7 か月
- 開発者の人数：1
- 開発言語：Visual Smalltalk
- 開発プロセス：漸進型
- システム概要：入金消し込みシステム
- システム規模：システム規模：第 4 版のクラス数，メソッド数，行数はそれぞれ 62，2644，20470 で，第 1 版のそれらの 1.68 倍，3.51 倍，3.07 倍であった．

[システム C]
- 版数：14
- 計測期間：3 か月
- 開発者の人数：4
- 開発言語：Visual Smalltalk
- 開発プロセス：ウォータフォール型
- システム概要：証券管理システム
- システム規模：第 14 版のクラス数，メソッド数，行数はそれぞれ 133，1487，14934 で，第 1 版のそれらの 1.51 倍，2.13 倍，2.27 倍であった．

システム A と B は，毎月顧客にシステムを提供し，その後に要求変更を受け付けて漸進的に開発を進めた．システム C はソフトウェア製品として販売することを目標に開発されていたシステムであり，計測期間中に利用者の要求変更を受け付けることはなく，1 週間ごとに収集した開発途中の成果物を計測対象とした．

4.2 計測結果

システム A，B に定義されていた各クラスが，システムの成長期間に増減させた複雑度 WMC の変化率を調査した結果，クラスによって変化の大きさ

や変化する時期に差があることがわかった．

表1　クラス群別の変化率(%)の分布
Table 1　Distribution of change rate for each class species (%)

システムA：境界領域種	ver.1-2	ver.2-3	ver.3-4
中央値	7.1	32.5	0.0
標準偏差	44.3	72.1	14.3
最小	-37.1	-30.0	-30.8
最大	133.3	260.0	33.3
標本数	11	15	28
システムA：問題領域種	**ver.1-2**	**ver.2-3**	**ver.3-4**
中央値	0.0	0.0	0.0
標準偏差	6.8	43.0	6.1
最小	0.0	-40.0	-22.2
最大	16.7	100.0	2.0
標本数	6	12	19
システムB：境界領域種	**ver.1-2**	**ver.2-3**	**ver.3-4**
中央値	46.8	5.8	0.0
最小	-	-	-
最大	-	-	-
標本数	1	1	2
システムB：問題領域種	**ver.1-2**	**ver.2-3**	**ver.3-4**
中央値	46.9	2.5	0.0
標準偏差	132.3	59.7	5.1
最小	-78.6	-36.4	-4.4
最大	421.1	266.7	27.6
標本数	27	54	56
システムB：汎用領域種	**ver.1-2**	**ver.2-3**	**ver.3-4**
中央値	-0.7	0.0	0.0
標準偏差	1.0	0.0	0.0
最小	-1.4	0.0	0.0
最大	0.0	0.0	0.0
標本数	2	12	3

ここで，システムAおよびBの，境界領域種，問題領域種，汎用領域種に属するクラスに対して，変化率の分布を求めた結果を表1に示す．また，表2には，3システムで観測を行った各期間において，変化しなかったクラスの割合を示した．ただし，システムAおよびBの境界領域種は利用者インタフェースを担うクラス群であり，システムCの境界領域種はデータベースとのインタフェースを担うクラス群である．

4.2.1　境界領域種の進化過程

表1では，境界領域種は他の種に比べて計測値に散らばりが大きく，また表2によると，ver.3-4の期間で境界領域種の46.4%のクラスが変化していな

表2　無変化のクラスの割合(%)
Table 2　Rate of unchanged classes(%)

システムA	ver.1-2	ver.2-3	ver.3-4
境界領域種	27.3	13.3	46.4
問題領域種	83.3	50.0	84.2
再利用可能...	83.3	100.0	100.0
(クラス数)	(6)	(6)	(6)
システムB	**ver.1-2**	**ver.2-3**	**ver.3-4**
問題領域種	22.2	33.3	82.1
再利用可能...	40.0	22.2	100.0
(クラス数)	(5)	(9)	(10)
汎用領域種	50.0	100.0	100.0
汎用領域種α	-	100.0	-
(クラス数)	-	(10)	-
汎用領域種β	50.0	100.0	100.0
(クラス数)	(2)	(2)	(3)
システムC	**ver.00-01**	**ver.01-02**	**ver.02-03**
境界領域種	25.8	79.4	91.2
問題領域種	31.1	76.8	88.0
汎用領域種	100.0	100.0	100.0
	ver.03-04	**ver.04-05**	**ver.05-06**
境界領域種	67.7	97.1	100.0
問題領域種	60.7	9.5	1.1
汎用領域種	100.0	100.0	100.0
	ver.06-07	**ver.07-08**	**ver.08-09**
境界領域種	100.0	100.0	100.0
問題領域種	97.9	90.6	100.0
汎用領域種	50.0	100.0	100.0
	ver.09-11	**ver.11-12**	**ver.12-13**
境界領域種	100.0	100.0	94.1
問題領域種	100.0	100.0	100.0
汎用領域種	100.0	100.0	100.0

いのに対して，問題領域種では84.2%に達するクラスで変化が観測されていない．このことから，システムAでは，境界領域種の方が他の種に比べてクラスによる変化率の値に差があり，全体として不安定な期間が継続していることがわかる．

そこで，このような種による差異がどのような要因によって発生するかを調べた．インタビュー調査から，システム開発終盤のver.3からver.4にかけて将来の変更のための設計変更を行わなかったことが確認できたので，これらの差異が開発要因のよって発生したと言うことはできない．したがって，境界領域種が不安定な期間を継続させたのは，利用要因の影響と考えてよいだろう．クラスが変化し続けている場合，クラスの仕様も揃っておらず，再利用後の変更が多発する可能性が高いという点で，汎用性は低いと思われる．したがって，ここで観測された境界領域種の不安定さは，再利用に不適切なクラ

スの状態を表わしていると考えられる．

システムBでは，境界領域種は最終版でも2クラスしかない．実際の利用者インタフェースはVisualSmalltalkのWorkbenchと呼ばれる再利用環境で定義されているから，システム固有のクラスを定義する必要はなかったようである．システムCでも同様の開発方針が選択され，利用者インタフェースを担う境界領域種は定義されていなかった．

4.2.2 問題領域種の進化過程

システムAとBの問題領域種は，管理，制御の役割を持つオブジェクトと，それ以外の静的なデータを保持するオブジェクトとに分類することができる．管理制御クラスとして，動的メトリクスを用いた計測結果から2つ以上のクラスと双方向のメッセージ送受信を行っているクラスを抽出し，開発者にも確認を依頼して分類を確定した．そして，システムAとBの問題領域種に対してこの分類を適用し，それぞれの変化率の分布を求めて表3に示した．表3では，システムAとBの管理制御クラスが，他のクラスよりも先に安定する傾向を共通に観察できる．

システムAの管理制御クラスが行っていたメッセージ送受信に動的メトリクスを適用したところ，送信したメッセージ名と受信先のクラス名の組みから，ここには多相メッセージが用いられていることがわかった．多相メッセージのインタフェースを確定すれば，システムが成長するに従ってメッセージの送信先が多様化してもメッセージを送出する管理制御クラスを変更する必要はない．このような設計は，管理制御クラスを利用要因から切り離すために有効な手段である．これはテンプレートメソッド[5]と呼ばれるデザインパターンを用いたフレームワークに該当する．

システムBの管理制御クラスは，利用者インタフェースの部品と問題領域のオブジェクトやデータベースとを接続し，利用者の操作によってオブジェクトのデータ管理を行うクラスである．ここでも制御管理クラスはそれ以外のクラスと共にアプリケーション固有のフレームワークを構成しており，このフレームワークの構造はver.1の開発時から変化していない．

これらのフレームワークが，今後，システムが作り替えられるときも再利用されるか否かは不明である．しかし，以上のクラスの進化過程から，システムのフレームワークがver.1やver.2といった開発の初期段階で確定することを確認できた．そして，フレームワークが確定した後，管理制御クラスが安定段階に達し，更にその他の問題領域種が安定期へ向かう経過を観測できた．

4.2.3 再利用可能な問題領域種の進化過程

システムAの再利用可能な問題領域種は，6クラスのうち1クラスがver.1からver.2へ至る過程で複雑度が1増えたのを除いて，ver.4まで全く変化がない．この傾向は汎用領域種の特徴と似ている．システムAの再利用可能な問題領域種は，物理量や定形的な計算を行うクラスである．このようなクラスは利用要因を受けることはないから，開発当初から汎用的な性質を持っていたと考えられる．

システムBの再利用可能な問題領域種には，開発当初，再利用を意識しないクラスとして開発され，後に再利用可能なクラスとして登録されたものも含まれている．表4で見られるように，後で説明する汎用領域種に比べて，再利用可能な問題領域種のクラス群はver.1からver.3へ至る過程の変化率が大きい．しかし，ver.3からver.4へ至る過程では，汎用領域種と同様に変化率は0になった．このように変化率が変化する原因として，2つの要因を考えることができる．まず，再利用性が意識されない時期には利用要因による影響を受け，そして，再利用性を高める際，利用要因の影響下から切り離すという開発要因の影響を受ける．

4.2.4 汎用領域種の進化過程

汎用領域種は，その起源によって2種類に分けることができる．汎用領域種αは，多くの評価を受けたとは言えない他の開発者が別のシステム開発で作ったクラス群で，汎用領域種βは，システムの開発者が所属する部署で開発され，すでにいくつかのプロジェクトで利用された実績を持つクラス群である．

汎用領域種を2種類に分類した結果，表2に示すように，汎用領域種αおよび汎用領域種βは，安定した傾向を示している．これらの安定傾向は，利用要因の影響を受けなかったシステムCにおけるver.05（開発開始1か月後）以降のクラスの傾向と

表3　問題領域種の変化率の分布

Table 3　Distribution of change rate for domain species

システムA：管理制御クラス			
	ver.1-2	ver.2-3	ver.3-4
中央値	-	-30.3	0
標準偏差	-	67.4	0.9
最小	-	-40.0	0.0
最大	-	100.0	2.0
標本数	-	4	5
システムA：管理制御クラス以外			
	ver.1-2	ver.2-3	ver.3-4
中央値	-	59.0	0.0
標準偏差	-	29.6	8.9
最小	-	38.1	-22.2
最大	-	80.0	0.0
標本数	-	2	8
システムB：管理制御クラス			
	ver.1-2	ver.2-3	ver.3-4
中央値	40.0	1.6	0.0
標準偏差	126.4	58.5	0.8
最小	-5.3	-36.4	0.0
最大	400.0	216.7	4.0
標本数	13	26	26
システムB：管理制御クラス以外			
	ver.1-2	ver.2-3	ver.3-4
中央値	129.4	2.2	0.0
標準偏差	141.3	46.0	7.8
最小	11.1	-4.3	-4.3
最大	421.1	200.0	27.3
標本数	9	19	20

同様である．この比較データから，汎用領域種が安定しているのは，利用要因の影響を受けない性質を持っているためと結論できる．

表4に，システムBの再利用可能な問題領域種および汎用領域種α，β，システムCの再利用可能な問題領域種，汎用領域種βの変化率の分布を示した．それぞれの起源による汎用領域種の変化の過程を比較し，汎用領域種の特徴を詳細に調査した結果について，次に議論しよう．

4.2.5　汎用領域種αの進化過程

システムBの汎用領域種αはver.2からver.3へ至る期間で使われ，ver.4では削除された．開発者へインタビューしたところ，再利用したクラスの仕様が大きすぎたため，必要最小限の機能をメソッドとして既存のクラスに組み込んでいたことがわかった．

これは，汎用領域種でも開発要因へ対処する必要があり，システムの中で破棄される可能性もあるという例を表わしている．また，再利用実績が少ない汎用領域種は，新しいシステムに導入された際に規模の大きい変化が起こることもある．この点に関しては，利用実績と汎用領域種の再利用時の変化について，更に調査を行う必要がある．

4.2.6　汎用領域種βの進化過程

表4　再利用可能な種の変化率(%)の分布

Table 4　Change rate distribution of reused species(%)

システムB:再利用可能な問題領域種			
	ver.1-2	ver.2-3	ver.3-4
平均	-1.9	36.3	0.0
最小	-78.6	-10.5	0.0
最大	50.0	266.7	0.0
標本数	6	10	12
システムB:汎用領域種α			
	ver.1-2	ver.2-3	ver.3-4
中央値	-	0.0	-
最小	-	0.0	-
最大	-	0.0	-
標本数	0	10	0
システムB:汎用領域種β			
	ver.1-2	ver.2-3	ver.3-4
中央値	-0.7	0.0	0.0
最小	-1.4	0.0	0.0
最大	0.0	0.0	0.0
標本数	2	2	3
システムC:再利用可能な問題領域種			
	ver.00-05	ver.05-10	ver.10-13
中央値	0.0	11.8	0.0
最小	0.0	0.0	0.0
最大	0.0	33.3	0.0
標本数	6-7	8	8
システムC:汎用領域種(β)			
	ver.00-05	ver.05-10	ver.11-13
中央値	0.7	0.0	0.0
最小	0.0	0.0	0.0
最大	1.4	0.0	0.0
標本数	2	2	2

システムBの汎用領域種βはシステムCでも再利用されていた．いずれの例も導入当初に小さな変化がある．表4で観察できるように，システムBのクラスに起きていた変化は増加傾向を持っているが，システムCのクラスの変化は減少傾向を持っている．したがって，両者は異なる変更である．システムB，Cの開発者にインタビューしたところ，これらの変化は，休業日の設定や営業日の判定方法などの例外処理の追加や不具合の解消によるものであった．以上の調査結果から，導入時の変化は，再利用するシステムの利用要因および開発要因へ対処するための変化と考えられる．

5. 議　　論

5.1 境界領域種の再利用

境界領域種，特に利用者とのインタフェースを担うクラスは，利用者の要求変更の影響を直接受けて開発期間中に変化が継続し，クラスによってその変化の差も大きいことを観測した．

この種のクラスが再利用性を高める例はまだ観測できていないが，システムBやシステムCで利用者インタフェースが再利用部品を組み合わせることによって構築されていたことを考えると，利用要因に対処して変化が継続して起こる境界領域種と，利用要因の影響下にない境界領域種が存在することになる．

アプリケーションプログラムが開発されるときには，アプリケーションフレームワークが使用されるが，システムB, Cで採用したWorkbenchのアプリケーションフレームワークの構造自体は安定しており，開発要因によって変化するものではなかった．

5.2 問題領域種の再利用

理論的には，問題領域種を利用要因へ対処し続ける部分と，利用要因の影響を受けない部分とに切り離すことができれば，後者を再利用可能なクラスとして進化させることができるはずである．さらに再利用可能なクラスは，問題領域に特殊化したクラスと問題領域に依存しないクラスとに分類できる．いずれもクラスも，システムAに見られた再利用可能な問題領域種で観測された物理量を保持するクラスといった要求変更が起こりえない場合を除いて，漸進的開発を繰り返す過程で徐々に進行する．

システムAの問題領域種がver.3からver.4にかけて安定した理由が利用要因の影響下から脱したことを意味しているとは一概に結論できないが，少なくとも，このような定量的な計測で不安定な状態を表わすクラスは再利用には向かない．しかし，システムCのように利用要因の変動がない場合は，クラスが安定していたもどのクラスが再利用可能かを客観的に判断することはできない．

したがって，問題領域種を再利用する際には，そのクラスの仕様だけでなく，現在どの程度，変化の要因の影響下から切り離されているかといった進化の過程を定量的に参照する必要がある．

5.3 汎用領域種の再利用

汎用領域種は，多くのクラスからメッセージを受けるので，設計変更，特にメッセージインタフェースの変更は他のクラスへ与える影響が大きい．そのため，システムCの汎用領域種がver.00版ですでに完成され，それ以降変化しなかったように，汎用領域種は開発の初期に仕様を決定し，以降の変更を避ける設計でなければならない．

計測結果でも，汎用領域種が新しいソフトウェアへ移植されたときは，そのソフトウェア固有の利用要因や開発要因に対処するために多少変化することもあるが，基本的に各要因に起因する変化の影響を受けない性質を持っていることが確認できた．

5.4 オブジェクトの進化モデル構築に向けて

クラスというオブジェクトの進化過程の調査結果をもとに，オブジェクトの進化モデルの構築を試みた．オブジェクトの進化の特徴として次のことが明らかになった．

・境界領域種は利用要因の影響を強く受けて変化を繰り返すが，アプリケーションから独立した再利用可能な境界領域種も存在する．今回の調査では両者を繋ぐ進化過程を観測することはできなかったが，両者を繋ぐ進化の経路は必ず存在すると思われる．

・問題領域種は境界領域種よりも小さな変化を繰り返す傾向を持ち，問題領域から独立したり問題領域に特殊化して再利用性を高めるとクラスは安定し，定量的な変化が観測されなくなる．

・アプリケーションに依存した問題領域種の中にはドメインフレームワークが形成される．フレームワークは，メッセージインタフェースの安定からクラスの安定といった進化の過程を経て成長する．

・特定のアプリケーションに依存しないWorkbenchが提供しているフレームワークなどのアプリケーションフレームワークは，問題領域に依存せず，利用要因や開発要因の影響を受けることはなく，変化しない．

・汎用領域種はすべての影響を受けにくい位置にあり，再利用時に多少変化することはあっても，大きな変化を起こさない．

以上の各種の進化過程は，利用要因の影響を受ける平面と開発要因の影響を受ける平面で囲まれた，

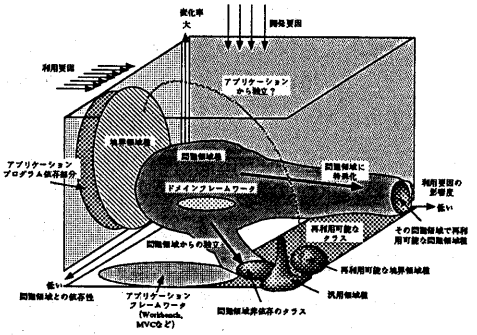

図1　オブジェクトの進化モデル

Fig. 1　Object evolution model

問題領域との依存性を表わす軸，利用要因の影響度の大きさを表わす軸，変化率の大きさを表わす軸を持つ図に表現することができる．図1にオブジェクトの進化過程を示した．

6. ま と め

本稿では，クラスを3つの種に分類し，それらの進化過程について開発事例を用いて観測した．その結果，オブジェクト指向システムにおいて，アプリケーションのクラスが再利用可能なクラスへ進化するには，様々な影響を受けて変化を繰り返すことが明らかとなった．今後開発されるクラスには，本稿で示したような定量的な変化の経歴を添付することによって，再利用を円滑に進めることができるようになると期待する．

謝　辞

本研究は，通商産業省ならびに情報処理振興事業協会の推進する独創的情報技術育成事業の一環として行われたものである．研究の機会を与えてくださった同事業関係者の皆様に感謝いたします．また，進化の計測に際して，ご協力いただきました（株）SRA の友枝敦氏，松田晴美氏，近藤博次氏に感謝いたします．

参 考 文 献

1) Basili, V. R. & Melo, W. L. : "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transaction on Software Engineering*, Vol. 22, No.10, pp. 751-761 (1996).
2) Lehman, M. M. & Belady, L. A. : *Program Evolution*, Academic Press (1985).
3) Chidamber, S. R. & Kemerer, C. F. : "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol.20, No.6, pp. 476–493 (1994).
4) Davis, A. M. : "A Strategy for Comparing Alternative Software Development Life Cycle Models," *IEEE Transaction on Software Engineering*, Vol. 14, No. 10, pp. 1453–1461 (1988).
5) Gamma, E. et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, MA (1994).
6) Goldberg, A.: *Smalltalk-80, The Interactive Programming Environment*, Addison-Wesley (1984).
7) Henderson-Selleres, B. & Edwards, J. M. : "The Object-Oriented System Life Cycle," *Communications of ACM*, Vol. 33, No. 9, pp. 142-159 (1990).
8) Lientz, B. P. & Swanson, E. B. : *Software Maintenance Management*, Addison-Wesley (1980).
9) Lewis, T., et. al.: *Object-Oriented Application Frameworks*, Prentice Hall (1995).
10) Tamai,T. & Torimitsu,Y. : "Software Lifetime and its Evolution Process over Generations," *Proc. of the Conference on Software Maintenance*, November, pp. 63–69 (1992).
11) T. Nakatani, T. Tamai, A. Tomoeda, H. Sakoh,: Quantitative Analysis on Evolution Process of Object-Oriented Systems, *Proc. of the International Symposium on Future Software Technology*, pp. 49–56 (1996).
12) 中谷多哉子, 玉井哲雄, 友枝敦, 酒匂寛,: オブジェクト指向によるシステムの進化を表すメトリクスの検討, ソフトウェアシンポジウム論文集, SEA, pp. 52–62 (1996).
13) 中谷多哉子, 玉井哲雄, 友枝敦, 酒匂寛,: オブジェクト指向システムの進化プロセスの定量的分析, 日本ソフトウェア科学会第 *13* 回大会論文集, pp. 389–392 (1996).
14) 友枝敦, 松田晴美, 玉井哲雄, 中谷多哉子 : オブジェクトの組織化と進化に関する研究報告書, IPA 独創的先進的情報技術に係わる研究, 情報処理振興事業協会 (1997).