

ソフトウェア開発におけるデザイン進化のモデル

下滝 亜里[†] 青山 幹雄[‡]

南山大学 大学院 数理情報研究科[†]

南山大学 数理情報学部 情報通信学科[‡]

1. はじめに

進化容易なソフトウェアを設計するには、要求変化とそれに伴うデザイン構造の変化の対応関係の理解が重要となる。しかし、この関係を表現するための統一的なモデルは十分に議論されていない。本稿では、要求変化に伴ってデザイン構造が変化する過程をデザイン進化として捉え、デザイン進化をモデル化する方法を提案する。モデルの表現力を、例題を基に評価する。

2. 要求のメタモデル

要求のメタモデルを図 1 に示す。要求はデザインにより実現される。要求を変化させるドライバには、ユーザ、開発者、環境がある。ユーザは、ソフトウェアに対する機能的・非機能的な要求変化を引き起こす。機能追加や性能向上などである。開発者は、拡張性や保守性といった品質や、ログインなどの補助的な機能の観点から要求を変化させるドライバとなる。環境とは、ここでは、利用しているライブラリやフレームワークなどを表す。ライブラリのバージョンアップによる API 変化などにより要求変化が起こる。

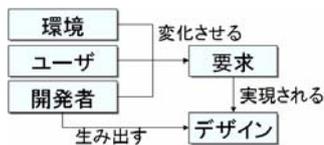


図 1 要求のメタモデル

3. デザインのメタモデル

ソフトウェアのデザインプロセスとは要求を満たす問題に対する解を探索するプロセスである。デザインとは、プロセスの出力であり、ソフトウェアの抽象記述である。ソースコードなどがそれに対応する。デザインはデザインパラメータから構成される[1]。クラス、メソッド、属性などはデザインパラメータと見なせる[2]。デザイン構造とはデザインパラメータとその間の関係である。依存関係や継承関係などがある。図 2 にデザインプロセス、デザイン、デザインパラメータ、デザイン構造の関係を示す。

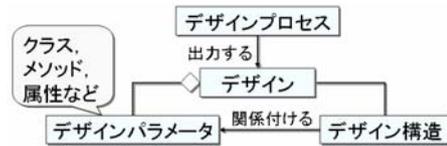


図 2 デザインのメタモデル

4. デザイン進化のモデル

デザイン進化の一般的なモデルを図 3 に示す[4]。要求変化をモデル化するために要求変化オペレータ C_n を導入する。 C_n は、要求 R_n に適用されるオペレータであり、適用後の要求を R_{n+1} と定義する。デザイン構造を変化させるオペレータとしてプリミティブデザイン進化オペレータ E_n^i を導入する。デザイン進化を、任意の数のプリミティブデザイン進化オペレータのシーケンス：

$$E_n = E_n^1 + E_n^2 + \dots + E_n^{m-1} + E_n^m \quad (1)$$

の適用であると定義する。ここで、 E_n をデザイン進化オペレータと呼ぶ。 E_n^i は、既存のデザイン S_n^{i-1} に適用されるオペレータであり、オペレータ適用後のデザインを S_n^i とする。 E_n をデザイン S_n ($= S_n^0$) に適用することによりデザイン S_{n+1} ($= S_n^m$) が得られるとする。ここで m は、要求 R_{n+1} を満たすデザイン S_{n+1} を得るために適用するオペレータの数とする。

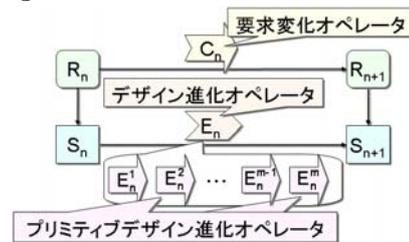


図 3 デザイン進化の一般モデル

5. 要求変化オペレータ

要求変化オペレータのモデルを図 4 に示す。オペレータには、追加、削除、変更の 3 つがある。要求の集合を以下で表す。

$$R_n = \{r_n^1, r_n^2, \dots, r_n^{k-1}, r_n^k\} \quad (2)$$

追加オペレータは、 R_n に新しい要求 r を追加する。

削除オペレータは、 R_n から要求 r_i を除く。

変更オペレータは、 R_n における要求 r_i を変更する。変更オペレータは、削除オペレータ適用後に追加オペレータを適用することでモデル化できる。

A Model for Design Evolution in Software Development
Asato Shimotaki[†], Mikio Aoyama[‡]
[†]Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
[‡]Dept. of Information and Telecommunication Engineering, Nanzan University



図4 要求変化オペレータのモデル

6. デザイン進化オペレータ

デザイン進化オペレータはプリミティブとコンポジットに分類できる(図5)。プリミティブデザイン進化オペレータとは、「メソッド名の変更」や「メソッドの移動」など、細粒度の構造変化である。要求変化の粒度が小さい場合は、このオペレータを一回適用することにより要求が満たされるため、それ自体がデザイン進化オペレータとなる。一方、要求の粒度が大きい場合は、プリミティブデザイン進化オペレータの複数回の適用が必要である。この場合のデザイン進化オペレータをコンポジットデザイン進化オペレータと呼ぶ。



図5 デザイン進化オペレータのモデル

6.1. デザイン進化オペレータの例

デザイン進化オペレータの例には、個々のリファクタリングやデザインパターン、モジュラオペレータ[1]がある。モジュラオペレータとは、モジュール化のために構造を変化させる行為である。モジュールの分離や追加などのオペレータが知られている[1]。リファクタリングやデザインパターンの多くは、コンポジットデザイン進化オペレータである。モジュール追加のオペレータは、プリミティブデザイン進化オペレータと見なせる。

7. 評価

ATM (Automated Teller Machine) シミュレータを例[3]として、表1に示す要求変化に伴うデザイン変化の過程に沿って、デザイン進化のモデルの表現力を評価した。ATMの仕様は有限状態機械によりモデル化した(図6)。

最初の要求変化として、ATMの状態遷移を制御するコア機能の実現がある(図7)。要求追加オペレータを R_0 に適用し R_1 を得る：

C_0 =要求追加(コア機能), R_0 = $\{\}$, R_1 = $\{\text{コア機能}\}$

要求を満たすために、クラス追加オペレータとStateパターンを用いる：

E_0 =クラス追加(ATMMain)+Stateパターン(Context=ATMContext, State=ATMState, ConcreteState={Wait,...,Pay})

Stateパターンはコンポジットデザイン進化オペレータであるため、 E_0 は次のように展開できる：

E_0 =クラス追加(ATMMain)+クラス追加(ATMContext)+クラス追加(ATMState)+クラス追加+(Wait)+...+クラス追加(Pay)+継承関係の確立

(super=ATMState, sub=Wait)+...+継承関係の確立 (super=ATMState, sub=Pay)

E_0 を S_0 に適用し S_1 を得る。この変化をDesign Structure Matrix[2]を用いて確認した。残りの要求変化に対しても同様の手順を繰り返し、提案モデルがデザイン進化を表現できることを確認できた。

従来研究[2]では、モジュラオペレータ[1]を基にデザイン進化を扱っているが、要求変化とデザイン構造の変化の関係を表現するデザイン進化のモデルは扱っていない。一方、本稿のモデルは、デザイン構造の変化だけでなく要求の変化に対してもオペレータを導入しており、要求変化とデザイン進化を統一的に表現できる。

表1 ATMシミュレータの要求変化

n	要求変化
1	ATMのコア機能の実現
2	コマンドラインユーザインタフェース(CUI)の実現
3	メモリ使用量の削減
4	シミュレータの並列化
5	GUIによるCUIの置き換え

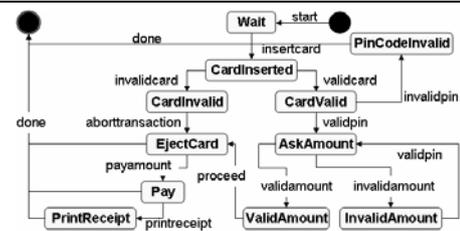


図6 有限状態機械によるATMのモデル化

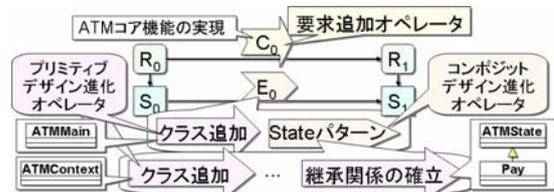


図7 ATMコア機能の実現

8. まとめと今後の課題

本稿では、要求変化に伴いデザイン構造が変化する過程をデザイン進化として捉え、モデル化する方法を提案した。例題を用いてモデルの表現力を評価した。今後、デザイン進化オペレータが受け取る引数の定義やオペレータ適用の前提条件の分析を検討する。

参考文献

- [1] C. Y. Baldwin and K. B. Clark, *Design Rules: The Power of Modularity*, MIT Press, 2000.
- [2] C. V. Lopes and S. Bajracharya, Assessing Aspect Modularizations Using Design Structure Matrix and Net Option Value, *Trans. on AOSD I*, LNCS 3880. Springer, 2006, pp. 1-35.
- [3] J. van Gurp and J. Bosch, Design Erosion: Problems and Causes, *J. of Systems & Software*, Vol. 61, No. 2, Mar. 2002, pp. 105-119.
- [4] 下滝 亜里, 青山 幹雄, ソフトウェアデザインにおける進化モデルの枠組み, 情報処理学会 ソフトウェア工学研究会, Vol. 2006-SE-154, No. 12, Nov. 2006, pp. 73-80.