リバースエンジニアリングによる実用的な設計情報復元に向けて

野田 訓広[1,a)]　小林 隆志[1,b)]

概要：ドキュメントが不十分なレガシーシステムに対し，リバースエンジニアリングによりソフトウェアの構造・振る舞いを復元する手法が多く研究されている．レガシーシステム理解に役立つ近年の有力な研究を紹介し，実用に際しての課題を提起・議論する．

Towards Design Recovery for Practical Use by Reverse Engineering

1. はじめに

IT モダナイゼーションやソフトウェア保守など，システムの改変・置換を行う上で，開発者はシステムの構造・振る舞いを十分に理解する必要がある．システムの構造・振る舞いを理解する際には，仕様書・設計書等のドキュメントが重要な役割を持つ．しかし，度重なる仕様変更・改変や，時間・コストの制約から，ドキュメントがシステムの最新状態を正しく反映していないことが多くあり，システムの構造・振る舞い理解が困難となっている．

この問題を解決するため，リバースエンジニアリングによりプログラム理解を支援する手法が多く研究されている [1], [2]．それらの中には，ソフトウェアの仕様復元に焦点をあてたもの，設計復元に焦点をあてたものなど様々な手法が存在するが，本稿では特に，設計復元の技術を取り上げて議論する．

リバースエンジニアリングによる設計情報の自動復元では，復元された情報量が，開発者の求める情報量よりも巨大なものとなることが多くある．そのため，情報の抽象化・削減等の対策（情報の取捨選択）が重要となり，いくつもの手法が提案されている．それらの手法は，例えば，システムの中で重要となるクラスを推定したり，設計パターンを基に設計意図の検出を試みることで，情報の取捨選択を上手く行っている．しかし，開発者が求める情報は，タスクに応じて様々に変化し得るため，実用の際にはいくつかの障害が発生すると考える．本稿では，近年の有力な設計復元手法を紹介し，実用化に向けた課題を議論する．

[1] 東京工業大学
Meguro-ku, Tokyo 152–8552, Japan
[a)] knhr@sa.cs.titech.ac.jp
[b)] tkobaya@cs.titech.ac.jp

2. リバースエンジニアリングによる構造図・振る舞い図の復元・抽象化

システムを理解するためには，その構造と振る舞いを理解する必要がある．本節では，近年の，リバースエンジニアリングによるソフトウェア構造・振る舞い図の復元手法について述べる．リバースエンジニアリングにより復元された設計情報は，設計意図の理解に重要な情報以外にも，実装詳細のような些末な内容を多く含むため巨大なものとなる．このため，情報の抽象化・削減等の対策が重要となる．

ソフトウェア構造図の復元・抽象化

構造図の復元は，システムの実行ファイル等に含まれるクラス情報をもとに，クラス図等を復元することにより行われる．

Thung らは，クラスの属性数や依存辺数・他クラスとの連結性などを指標に，独自の分類器により分類を行い，各クラスが重要か否かを分類している [3]．非重要と判定されたクラスを削減することで，復元されたクラス図の情報削減を行う．開発者が作成したクラス図 (forward design により作成されたクラス図) と，復元後に情報削減したクラス図を比較することで評価を行い，提案手法が精度の高い結果を得ることを確認している．しかし，Thung らの手法のように分類器を用いる手法は，トレーニングデータに対して手動でラベル付けするコストがかかり，実用的でない

面がある．この問題に対し，Yang らは，より少ないデータに対しての手動ラベル付けのみで，クラスが重要か否かを精度良く分類する手法を提案している [4]．

ソフトウェア振る舞い図の復元・抽象化

振る舞い図の復元は，多くの手法において，システムの実行履歴に含まれるメソッド呼び出し情報等をもとに，シーケンス図等を復元することにより行われる．

実行履歴中には，繰り返し処理や再起呼び出しに起因する類似構造が多く含まれる．Myers らは，このような類似構造を特定・抽象化することで，情報量の削減を図っている [5]．しかし，類似構造の圧縮だけでは削減できる情報量に限りがあり，加えて，単純な機械的圧縮を行うと，設計意図を理解する上で重要な情報が欠損してしまう恐れがある．この問題に対し，我々は，設計パターンに着目することで，関連の強いオブジェクト群や設計意図の認識を試み，それに基づいた抽象化手法を提案してきた [6], [7]．

別のアプローチとして，実行履歴中におけるフェイズ（機能・タスクの区切れ・単位）を特定し，実行履歴をフェイズ単位に分割するといったものもある．渡辺らはオブジェクトの生成タイミングに着目することでフェイズを特定・分割する手法を提案している [8]．Pirzadeh らは，フェイズ分割した実行履歴から，層化抽出法によりイベントをサンプリングすることで，抽象的な振る舞い図を形成する手法を提案している [9]．

また，様々な抽象化・情報削減手法を組み合わせた可視化・デバッグ・解析を行う統合的な環境の開発も行われている [10]．

3. 実用化に向けた課題

2 節で述べた通り，様々な抽象化手法が提案されており，リバースエンジニアリングによる設計復元において問題となる膨大な情報量への対策は十分に取られつつある．しかし，実用あたっての障害がまだ多く存在すると考える．本節では，実用化に向けた課題を提起・議論する．

タスクに応じた情報の取捨選択

タスクに応じて，開発者の求める情報は変化する．2 節で紹介した抽象化手法はいずれも，システム全体の構造・振る舞いの概略を捉えるような場面で効力を発揮する．しかし，開発者が，特定の機能・障害に特化・横断した情報のみを得たい場合などには，求める情報が抽象化により削除されてしまったり，興味のない情報が多く含まれるなどの状況が多く発生し得ると考える．ユーザのクエリや開発環境の情報を抽象化手法の入力に加え，タスクに応じた情報の取捨選択が行えるような手法の開発が必要と考える．

また，システムの理解・改変作業が進むにつれ，開発者が必要とする情報の粒度は変化する．しかし，既存手法は，抽象化のレベルが固定，もしくは，低コストで柔軟に変更できないことが多い．段階的な抽象化など，抽象化レベルを柔軟に変更できるような手法の開発が必要と考える．

振る舞い解析における実行時オーバーヘッド

振る舞い図のリバースエンジニアリングにおいては，振る舞いのロギングにかかるパフォーマンスオーバーヘッドが問題となる．システムのあらゆる箇所に解析用のロギングコードを埋め込むと，非常に大きなオーバーヘッドが生じるため，解析の実行可能性・開発者の時間コストの面から実用上の問題が生じる．低オーバーヘッドを維持しつつ解析精度を保つ手法や，ロギングすべき箇所を低コストで特定する手法の開発が必要と考える．

対象システム構成の制約

既存研究の多くは，オブジェクト指向言語（特に Java）を意識した手法を開発しており，プログラミング言語の制約が存在するものも多い．また，Web アプリケーションなどの分散システムに対する有用性の実証評価例も少ない．システム構成に応じた制約の明確化・解消も課題となると考える．

謝辞　本研究の一部は科研費（#24300006, #26280021, #15H02683）の助成を受けた．

参考文献

[1] Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L. and Koschke, R.: A Systematic Survey of Program Comprehension through Dynamic Analysis, *IEEE Trans. Softw. Eng.*, Vol. 35, pp. 684–702 (2009).

[2] Bennett, C., Myers, D., Storey, M.-A., German, D. M., Ouellet, D., Salois, M. and Charland, P.: A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams, *J. Softw. Maint. Evol.*, Vol. 20, No. 4, pp. 291–315 (2008).

[3] Thung, F., Lo, D., Osman, M. H. and Chaudron, M. R. V.: Condensing Class Diagrams by Analyzing Design and Network Metrics Using Optimistic Classification, *ICPC'14*, pp. 110–121.

[4] Yang, X., Lo, D., Xia, X. and Sun, J.: Condensing Class Diagrams With Minimal Manual Labeling Cost, *COMPSAC'16*, pp. 22–31.

[5] Myers, D., Storey, M.-A. and Salois, M.: Utilizing Debug Information to Compact Loops in Large Program Traces, *CSMR'10*, pp. 41–50.

[6] Noda, K., Kobayashi, T. and Agusa, K.: Execution Trace Abstraction Based on Meta Patterns Usage, *WCRE'12*, pp. 167–176.

[7] Toda, T., Kobayashi, T., Atsumi, N. and Agusa, K.: Grouping Objects for Execution Trace Analysis Based on Design Patterns, *APSEC'13*, pp. 25–30.

[8] Watanabe, Y., Ishio, T. and Inoue, K.: Feature-level Phase Detection for Execution Trace Using Object Cache, *WODA'08*, pp. 8–14.

[9] Pirzadeh, H., Shanian, S., Hamou-Lhadj, A., Alawneh, L. and Shafiee, A.: Stratified Sampling of Execution Traces: Execution Phases Serving As Strata, *Sci. Comput. Program.*, Vol. 78, No. 8, pp. 1099–1118 (2013).

[10] Jayaraman, S., Jayaraman, B. and Lessa, D.: Compact visualization of Java program execution, *Software: Practice and Experience*, (online), DOI: 10.1002/spe.2411 (2016).