クラスの責務の大きさに着目したUML設計クラス図の構造評価

津田直彦[†1]　鷲崎弘宜[†1]　深澤良彰[†1]

設計モデルの評価は主に人手で行われるが、時間がかかり属人性も高い。将来設計モデルの評価を自動化するためには、定性的な評価と設計モデルの定量的な特徴の関係の分析が必要となる。我々はUML設計クラス図についてクラスが担う責務の大きさを属性数や関連数で測った。そしてETロボコン2010に提出された65個の設計クラス図に対する実験を通して、構造に対する評価が高い設計クラス図は責務が大きいクラスを多く含んでいることを、小規模な組み込みシステムのドメインで確認した。

Evaluating Structral Validity of UML Design Class Diagrams by Focusing on Size of Responsibility of Classes

Naohiko Tsuda ,[†1] Hironori Washizaki [†1]
and Yoshiaki Fukazawa[†1]

1. はじめに

設計モデルではUMLクラス図がよく利用されるが、設計者は自分が重要と感じた箇所を詳細に書くことが多いというアンケート結果が報告されている[1)2)]。そのため、システムの構造を十分検討し重要なクラスと重要でないクラスを把握した上で作られた設計クラス図は詳細なクラスをいくつか含むと考えられる。逆に、重要な箇所を設計者が把握せずに作った設計クラス図は詳細なクラスを含まないと考えられる。我々は属性数や関連数が多いクラス（責務が大きいクラス）も詳細に書かれたクラスとみなし、以下の仮説を立てた。「責務が大きいクラスが少ないことは設計クラス図の構造として妥当でない」

表1 責務の大きさを表すクラスメトリクス

略称	クラスメトリクス
NAttr	クラスが直接持つ属性の数
NOp	クラスが直接持つ操作の数
NAssoc	クラスが持つ直接持つ関連の数

クラス図からは属性数などの定量的な特徴（クラスメトリック）を測定できる。我々はETロボコン2010[3)]に投稿された65個の設計クラス図について、表1に示すクラスメトリクスを測定した。そして、責務が大きいクラスの数と設計クラス図の構造に対する定性評価が正順位相関することを確認した。

その際、i) 属性数 (関連数, etc) が多いクラスを判定するための閾値を単一のクラス図ごとに用意する手法を提案した。属性数が多いクラスを判定しようとする場合、従来はii) データセットとして複数のクラス図を用意してそれを元に属性数の閾値を求めていた。実験ではi) とii) の二つの手法で「責務が大きいクラスの数」を数えた。そしてそれぞれから異なる知見を得た。

2. 背　　景

2.1 クラスが担う責務の大きさ

クラスが担う責務には情報を把握する責務と振る舞いに関する責務があると言われている[4)5)]。表1に示すクラスメトリクスはクラスが担う責務の大きさを表す。NAttrとNAssocが大きければ関係するクラスが多いということなので、そのクラスは多くの情報を把握する必要がある。一方NOpが大きいクラスは多くの振る舞いを提供する。これらのクラスメトリクスに着目することで、責務が大きいクラスを見分けることができる。

2.2 責務が大きいクラスの判別

表1に示すクラスメトリックそれぞれについて閾値を用意する。例えば、NAttrの閾値が4である場合、NAttrが5以上のクラスは属性数で見た責務が大きい

†1 早稲田大学
Waseda University

クラスといえる。

2.3 クラス図とソースコードの違い

UML クラス図とソースコードの違いのひとつとして、詳細度がある。ソースコードは誰が書いても実行可能なものとして残されるが、設計図はそうではない。開発者・設計者間で共通の認識を得るのに十分であれば、UML 標準から外れた書き方やあいまいな表現が認められがちである。また、設計者は自分が重要と感じた箇所を詳細に書くことが多いというアンケート結果が報告されている。そして、システムのどの部分が重要かどうかの見極めは個人差によるものが大きいと考えられる。そのため、クラスの属性数や関連数などといった定量的特徴 (e.g., クラスメトリクス) の分布傾向がクラス図ごとに異なると考えられる。例えば、属性数が 0〜3 のクラスばかりのクラス図がある一方、属性数が 3〜8 ばかりのクラス図もありうる。

ソースコードの良し悪しを判定する基準はバグ数などによって客観的に与えられる。一方で設計クラス図の良し悪しはレビュアーによる定性評価で判定することが多い。そして、レビュアーはクラス図ごとの書かれ方の違いを踏まえた評価をしていると考えられる。

3. 提 案 手 法

設計クラス図では、クラスメトリックの分布傾向がクラス図ごとに異なる上に、評価基準もクラス図ごとに異なると考えられる。そこで我々は、責務が大きいクラスを判定するための閾値をあらかじめ用意するのではなく、単一の設計クラス図ごとに用意する手法を提案する。例えば、あるクラス図 A が 20 個のクラスを含む場合、20 個の NAttr を測定できる。この 20 個の NAttr の 70 パーセンタイルを閾値とするなどである。このようにして求めた閾値を使用して責務が大きいクラスを数えた場合、あらかじめ用意した閾値で数える場合とは違った結果が得られる。

4. 評 価 実 験

ソースコードと違い、UML クラス図などの設計モデルを公開するオープンリポジトリが少ない。そこで、我々は ET ロボコン 2010[3] に提出された 65 個の設計クラス図を実験に用いることにした。これらのクラス図には設計構造の妥当性評価が専門家により与えられている。

我々は 65 個の設計クラス図について、表 1 に示すクラスメトリクスを測定した。そして、責務が大きいクラスをそれぞれのクラスメトリックの観点で数え、責務が大きいクラスの数と設計クラス図の構造に対する定性評価のスピアマンの順位相関係数を求めた。

5. 結　　果

属性数と操作数については、単一のクラス図ごとに閾値を求める方法で数えた責務が大きいクラスの数と定性評価との間に正相関が認められた。一方で関連数についてはあらかじめ用意した閾値で数えた責務が大きいクラスの数と定性評価との間に正相関が認められた。上記の正相関が認められたことから、責務が大きいクラスの数が少ないとき設計クラス図の構造の妥当性が低く評価されていることを確認した。

6. 関 連 研 究

Vasilescu らはクラスメトリックなどのミクロな単位の測定値をシステム単位のマクロな形に集約することで、ソフトウェアの保守性分析の際にその進化の洞察を提供できると述べている[6]。我々が提案した手法では Vasilescu らが述べるマクロな集約が可能であり、クラスに対する責務割り当てが開発の進展につれてどのように変化していくかを洞察できる。

参 考 文 献

1) Lange, C., Chaudron, M. and Muskens, J.: In Practice: UML Software Architecture and Design Description, *IEEE Softw.*, Vol. 23, No. 2, pp.40–46 (2006).
2) Nugroho, A. and Chaudron, M.R.: A survey into the rigor of UML use and its perceived impact on quality and productivity, *Proc. Second ACM-IEEE Int. Symp. on Empirical software engineering and measurement*, ESEM '08, New York, NY, USA, ACM, pp.90–99 (2008).
3) Japan Embedded Systems Technology Association: *ET Robot Contest 2010 Assessment Criterion for Model (Japanese)* (2010). http://www.etrobo.jp/2010/gaiyou/model.php.
4) Larman, C.: *Applying UML and patterns: an introduction to object-oriented analysis and design*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1998).
5) Bellin, D. and Simone, S. S.: *The CRC card book*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997).
6) Vasilescu, B., Serebrenik, A. and van den Brand, M.: You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics, *Proc. 2011 27th IEEE Int. Conf. on Software Maintenance*, ICSM '11, Washington, DC, USA, IEEE Computer Society, pp.313–322 (2011).