ソフトウェア設計の保守性を評価し改善する構造診断手法

Software Structure Diagnosis Method to Evaluate and Improve Design Maintainability

岡本　　渉
■OKAMOTO Wataru

現在のソフトウェア開発では，既存製品に対して仕様変更や機能拡張を行う“派生開発”と呼ばれる開発スタイルが多い。派生開発の開発効率は，ソフトウェア設計の保守性に大きく左右される。長年，仕様変更や機能拡張を重ねると，不適切な設計変更によって保守性が低下し，開発効率の低下を招く場合がある。

そこで東芝は，効率的な派生開発を実現するため，ソフトウェア設計の保守性を評価し，改善するための構造診断手法を開発した。この手法では，より高精度に保守性を評価するため，ソフトウェア構成要素の規模や設計変更確率を考慮した評価指標を定義するとともに，それにより検出される設計上の問題を効果的に改善する。

In the software development field, derivative development by changing the specifications of existing products and adding new functions to them has been increasing in recent years. Efficiency in this type of software development is dictated by maintainability, which is the capability to adapt the design of a software in this way. However, numerous modifications of a software over the long term can lead to reduced maintainability due to inadequate design changes and the resulting degradation of its structure.

To realize more efficient derivative development of software from existing software products, Toshiba has developed a software structure diagnosis method to evaluate and improve the maintainability of software design. We have defined a software metrics taking into consideration the sizes of software modules and probability of changes, and confirmed that this method can improve maintainability by identifying and displaying design problems.

1　まえがき

現在のソフトウェア開発では，既存製品への仕様変更や機能拡張など，ベースとなるソフトウェアを改変する派生開発が多くを占める。ソフトウェアの改変のしやすさを保守性と呼び，保守性に優れたソフトウェアであれば，高い派生開発効率を実現できる。しかし，長年の派生開発によってアドホックな設計変更が積み重なると，ソフトウェアの構造は複雑化し，保守性は低下する。その結果，開発効率も低下する。

ソフトウェア設計の保守性を維持するには，継続的に設計を評価し，問題があればそれを改善していく必要がある。従来から，設計を定量的に評価するための手段として，例えばオブジェクト指向設計に対するCKメトリクス[(1)]など，設計評価指標を用いることが提案されている。従来の指標では，ソフトウェア設計の構成単位（モジュール）は全て同規模・同質なものとして扱われている。しかし，実際のソフトウェアでは，各モジュールの規模にばらつきがある。そのため，従来の評価指標では保守性が適切に評価されないという問題があった。

そこで東芝は，モジュール規模のばらつきを考慮した設計評価指標を定義し，それにより検出される設計上の問題を効果的に改善する手法を開発した。ここでは，設計評価指標と設計改善手法について述べるとともに，その適用事例を述べる。

2　設計評価指標

保守性に優れたソフトウェア設計の考え方として，疎結合であることや高凝集であることが挙げられる。

疎結合とは，モジュール間の結合性に関する性質で，モジュール間の依存関係が少ないことを意味する。典型的な依存関係には，モジュールをまたいだ関数の呼出しやグローバル変数の参照がある。依存先モジュールの設計が変更されると，依存元モジュールのふるまいも変わる可能性があるため動作確認が必要になる。依存関係が少なければ設計変更の影響を受けるモジュールも少なく，それだけ効率的に開発が進められる。そこで結合性に関して，設計変更の影響範囲に着目し，“影響度”と呼ぶ指標を定義する。

一方，高凝集とは，ソフトウェアを構成する各モジュールに求められる性質で，モジュール内部が相互に密接に関連する要素で成り立っていることを意味する。凝集性が低い，すなわち相互に関連しない要素が多いと，そのモジュールの設計変更に際して，無関係な要素についてまで調査やテストを行う可能性がある。その結果，開発効率の低下を招く。ここでは，凝集性に関して，“凝集度”と呼ぶ指標を定義する。

2.1　影響度

影響度とは，あるモジュールの設計が変更された際，どれくらいの範囲にその影響が及ぶかを示す指標である。

2.1.1 影響度算出における課題

影響度の算出方法として，各モジュールに依存しているモジュール数の平均値を算出する方法がある[2]。例えば，**図1**に示すようなモジュール構造であれば，影響が及ぶモジュール数は平均2個になる。

この算出方法で問題となるのは，各モジュールの規模が同程度にそろっていることを前提としている点である。モジュール規模のばらつきが大きいと，モジュール数で評価する方式では適切に保守性を評価できない可能性がある。例えば，**図2**に示す二つの構造を比べてみると，モジュールCが影響を及ぼすモジュール数は，構造1で3，構造2では2となる。したがって，モジュール数で考えると，構造2のほうが影響範囲が狭く設計変更が容易と判定される。しかし，Cが影響を及ぼすソースコードの総量は構造1でも構造2でも同じであり，設計変更時に必要となるソースコードの調査やテストの量は同程度であると考えられる。

また，保守性を評価するうえでは，モジュール規模の違いに起因した，各モジュールの設計変更の起こりやすさの違いも考慮すべきである。従来の指標に見られる，モジュール数で割って影響度の平均値を算出するという処理は，どのモジュールにも等確率で設計変更が発生することを前提としている。しかし，モジュール規模のばらつきが大きければ，設計変更が発生するモジュールにも偏りが生じる。平均的には，巨大モジュールほど設計変更が生じやすく，極小モジュールでは変更が生じにくい。そのため，平均的な影響範囲として，モジュール数で割った値では不適切な評価となるおそれがある。

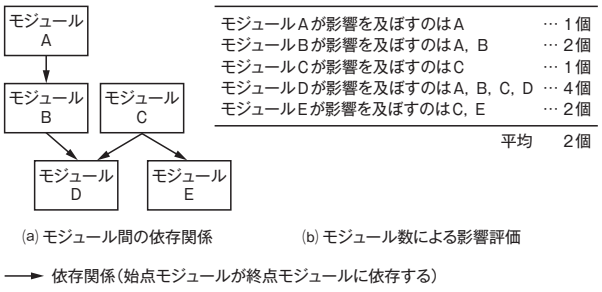


図1．モジュール数による影響範囲の評価 ― 各モジュールが影響を及ぼすモジュール数を求め，平均をとって評価する。

Evaluation of impact range by module counts

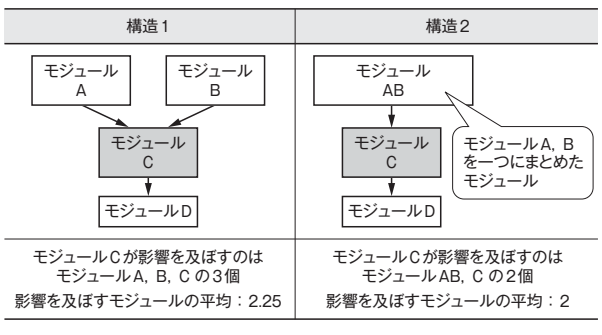


図2．モジュール数で影響範囲を評価する際の問題 ― 構造2は構造1に比べて二つのモジュールA，Bをまとめたものであるが，モジュール数で影響範囲を評価すると構造2のほうが保守しやすいと評価されてしまう。

Problem of impact range evaluated by module counts

2.1.2 影響度算出方式

ここでは，前述の課題を踏まえ，影響度を定義する算出式に次の2点を組み込んだ。

(1) 各モジュールの影響範囲は，影響を及ぼすモジュール数ではなく，ソースコード行数で評価する。

(2) 構造の平均的な影響範囲は，各モジュールに対する設計変更の発生しやすさ（変更確率）を重みとして，影響範囲の平均値を取る。変更確率は，各モジュールの規模に比例するものとして扱う。

影響度の算出式を式(1)に，算出例を**図3**に示す。

$$\text{影響度} = \sum_{m \in S} \left\{ \left(\sum_{m' \in Dep(m)} LoC(m') \right) \cdot \frac{LoC(m)}{LoC(S)} \right\} \Big/ LoC(S) \quad (1)$$

S：ソフトウェアを構成する全モジュールを要素とする集合

Dep(m)：モジュール m 自身も含め，m に依存している全モジュールを要素とする集合

LoC(m)：モジュール m のソースコード行数

2.2 凝集度

凝集度とは，各モジュールの内部の構成要素について，それらが相互に関連している度合いを示す指標である。

2.2.1 凝集度算出における課題

モジュールの凝集性を示す指標の一つとして，LCOM*（Lack of Cohesion in Methods）[3]が挙げられる。これは，モジュール内の変数一つ当たりにアクセスする，モジュール内の関数の数の平均割合をベースに算出される。ここでは，共通の内部変数を利用している関数群を相互に関連するものと扱い，その割合が高いほど，凝集性は高いと評価される。

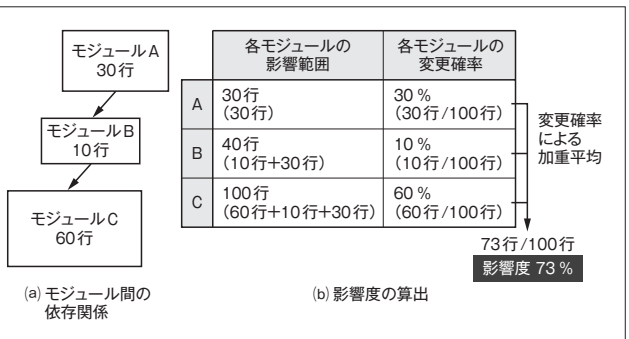


図3．開発した算出式に基づく影響度算出の例 ― 各モジュールの影響範囲をソースコード行数で算出したうえで，各モジュールの規模を重みとして，それらの加重平均を算出する。

Example of calculation of impact by use of proposed formula

一般論文

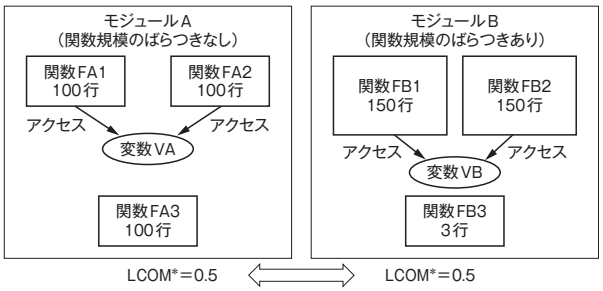


図4．関数規模にばらつきがあるモジュールに対するLCOM*算出の問題 ― モジュールBでは，相互に関連のある関数群がソースコード行数の99％以上を占めているが，LCOM*の値はモジュールAと同じ値が算出されてしまう。

Problem of lack of cohesion of methods (LCOM) for modules consisting of different size functions

前節の影響度の場合，モジュール規模のばらつきが評価に及ぼす影響を述べたが，凝集性の場合は，関数規模のばらつきが問題となる。例えば，**図4**に示す二つのモジュールA，Bは，ともにLCOM*が0.5と算出される。しかし，Bでは，その大部分がFB1及びFB2という相互に関連のある関数で構成されており，凝集性はAに比べて高いと考えられる。

2.2.2　凝集度算出方式　関数規模のばらつきを踏まえて，モジュールの凝集性を評価できるよう，凝集度を“モジュールのソースコードにおける任意の2行が相互に関連する確率”とした。ここで，関連の有無は，関数を単位として評価する。例えば，関数F1と関数F2に関連がある場合，F1に含まれる全てのソースコード行と，F2に含まれる全てのソースコード行は関連があると扱う。また，関数間の関連の有無については，LCOM*同様，共通の内部変数を利用しているか否かによって判断する。

凝集度の算出式を式(2)に示す。

$$\text{凝集度} = \sum_{f1 \in F} \sum_{f2 \in F} \text{Rel}(F1, F2) \cdot \text{SizeRatio}(F1) \cdot \text{SizeRatio}(F2) \qquad (2)$$

F：モジュール内の全ての関数を要素とする集合

Rel(F1, F2)：関数 F1 と F2 の間に関連があれば 1，なければ 0

SizeRatio(F1)：関数 F1 のソースコード行数の，全行数に占める割合

このように凝集度を定義すると，図4に示した二つのモジュールの凝集度は，A＝0.56，B＝0.98となり，関数規模のばらつきを踏まえた凝集性の評価が得られる。

3　設計改善手法

この章では，前章で述べた影響度に関する設計改善手法について述べる。凝集度が個々のモジュールの保守性を評価するものであるのに対して，影響度はソフトウェア構造全体の保守性を評価するものである。そのため，派生開発工数に及ぼす影響は，凝集度よりも影響度のほうが大きいと考えられる。そこで設計改善手法として，影響度を低減させる手法が求められる。

影響度が高くなってしまう構造の典型として，“循環依存”と呼ばれる構造が挙げられる。循環依存とは，複数モジュールが相互に依存し合う構造を指す。この構造において，影響度が高くなるのは，循環依存を構成するモジュール群のうち，どれを設計変更しても，他の全てに影響が波及してしまうためである。影響度低減のためには，できるだけ少ない変更量で，最大限の循環依存を解消できる手法が求められる。

開発した手法の概要を以下に述べる。この手法では，まず，循環依存を形成する依存関係の中から“逆流依存”を抽出する。ここで，逆流依存とは，循環を形成する依存のうち，もっとも依存の本数が少ない関係を指す。**図5**に示す例では，モジュールA，B，Cが循環依存の関係にあり，このうちもっとも依存本数が少ないCからAへの依存を逆流依存とする。この例では，単純な一つの循環における逆流依存を抽出したが，実際のソフトウェア構造では，複数の循環が組み合わさり，複数の逆流依存が存在するケースが多い。そこで，逆流依存を抽出する際には，DSM（Design Structure Matrix）[4]と呼ばれる手法を用いる。

次に，各逆流依存を削除したときの，循環依存の解消量を算出する。循環依存の解消量は，逆流依存の削除によって循環から外れるモジュールの総ソースコード量で定量化する。ここで削除対象とするのは，1か所の逆流依存だけでなく，複数箇所の逆流依存の組合せも対象とする。逆流依存の全ての組合せに対して，循環依存の解消量と削除対象の依存本数から，もっとも効果的な削除方法を選択する。

図6に示す例には，3か所の逆流依存が含まれている。これらの逆流依存の組合せに対して，循環依存の解消量を算出し，削除する依存本数1本当たりの循環依存の解消量を“改善効率”として求めることで，逆流依存①と②を削除することがもっとも効果的であると判明する。

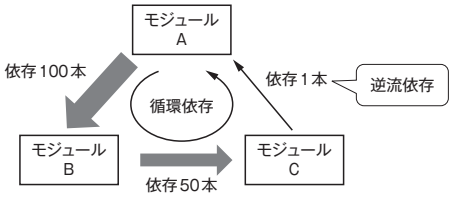
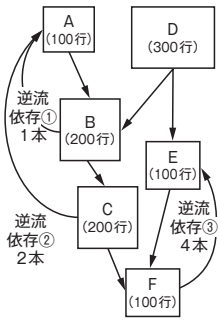


図5．循環依存から逆流依存を抽出 ― 循環依存の中でもっとも依存本数の少ない箇所を逆流依存とする。

Extraction of backward dependency from cyclic dependencies



(a) モジュール間の依存関係

削除対象の依存箇所	循環依存の解消量（行）	削除対象の依存本数（本）	改善効率
逆流依存①	0	1	0.00
逆流依存②	200	2	0.10
逆流依存③	200	4	0.05
逆流依存①＆②	500	3	0.16
逆流依存①＆③	200	5	0.04
…		…	

改善効率＝
（循環依存の解消量／削除対象の依存本数）
／ソースコード行数の合計（1,000）

(b) 逆流依存削除に対する改善効率

図6. 効果的な循環依存解消につながる逆流依存の探索 — 複数の逆流依存を組み合わせて，もっとも改善効率の良い組合せを探る。

Search for backward dependencies to effectively reduce cyclic dependencies

4　適用事例

開発した設計改善手法を，ある組込みソフトウェアに適用した結果を**図7**に示す。改善前のソフトウェア構造では，ソースコード全体の81％が循環依存に含まれていた。また，影響度も83％と，設計変更時には広範囲にわたってソースコード調査やテストが必要になる構造となっていた。これに対して，逆流依存となっている4か所を特定したうえで，それらの様々な組合せに対する循環依存の解消量を算出した（**図8**）。その結果，4か所の逆流依存のうち，3か所の依存（依存の本数にして8本）を削除することがもっとも効果的と判明した。改善後の構造は，ほとんどの循環依存が解消され，循環依存に関与するソースコードは，全体の4％にまで減少した。また，これにより影響度も改善前の構造に比べて37％低減した。

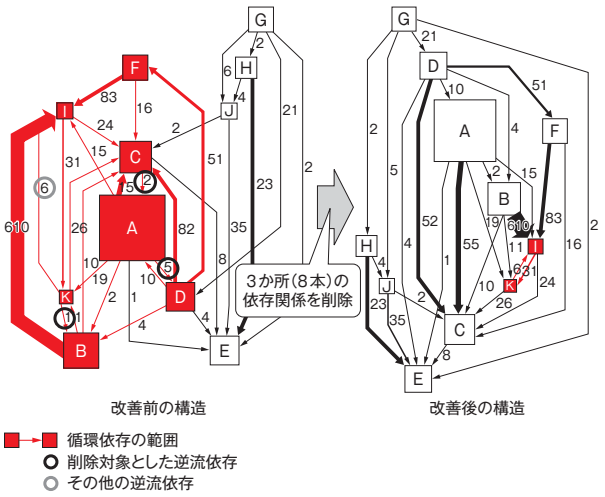


図7. 開発した手法による循環依存の改善例 — 逆流依存を削除すると循環依存を改善できる。

Example of improvement of cyclic dependencies through restructuring by newly developed method

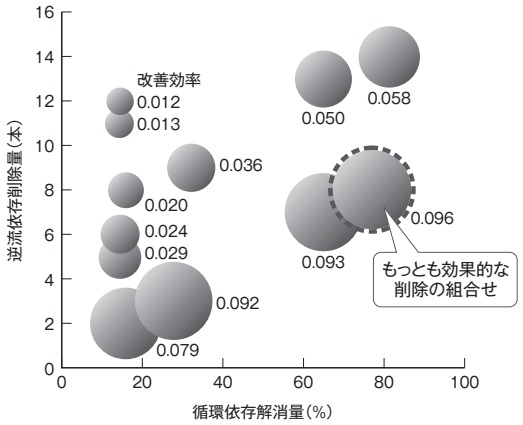


図8. 逆流依存の全ての組合せに対する改善効果 — 循環依存の解消量が大きいほど，また逆流依存の削減量が小さいほど，円の大きさで示す改善効果は高くなる。

Improvement effects for various combinations of backward dependencies

5　あとがき

効率的な派生開発のためには，保守性の高いソフトウェア設計が不可欠である。ここでは，ソフトウェア設計の保守性を評価し改善する構造診断手法について述べた。具体的には，保守性をより高精度に評価できるよう，モジュール規模のばらつきを考慮した影響度及び凝集度という二つの評価指標を定義した。また，影響度を悪化させる主要因である，モジュール間の循環依存を効果的に解消する手法について述べた。

今後は，低凝集度のモジュールに対する改善手法や，影響度を更に改善するための手法など，問題に応じた設計改善手法の整備を進めていく。

文　献

(1) Chidamber, S.R.; Kemerer, C.F. A Metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering. **20**, 6, 1994, p.476－493.
(2) テクマトリックス. “Lattix－アーキテクチャ分析ツール－”. <http://www.techmatrix.co.jp/quality/lattix/index.html>, （参照2013-08-20）.
(3) Henderson-Sellers, B. Object-Oriented Metrics: Measures of Complexity. USA, Prentice Hall, 1995, 252p.
(4) DSMweb.org. The Design Structure Matrix (DSM). <http://www.dsmweb.org/>, (accessed 2013-08-20).
(5) 岡本　渉 他. ソフトウェア設計の保守性を評価・改善するための構造診断手法の提案，情報科学技術フォーラム. **11**, 2012, p.253－256.

岡本　　渉　OKAMOTO Wataru
ソフトウェア技術センター　ソフトウェア設計技術開発担当
参事。ソフトウェア設計技術の開発に従事。
Corporate Software Engineering Center

一般論文