B-9

Javaプログラムにおけるプログラム疲労の測定法の提案

A proposal about the method for measurement of program fatigue for Java program.

○箭内　直樹**　梶谷　義行**　荒　伸太郎**
Naoki Yanai　Yoshiyuki Kajitani　Shintarou Ara
金子　正人*　武内　惇*　藤本　洋
Masato Kaneko　Atsushi Takeuchi　Hiroshi Fujimoto

1．はじめに

新しいシステムを設計する際に，新規にプログラムを作成する場合と，既存のプログラムを変更する場合とどちらの方が時間，コストなどの面でより効率的かの判断が重要になる．定量的なデータに基づく判断を行うことができないため，プログラムの変更を行うにあたって，工数や開発期間が予定を大きく越えるという問題が起こることがある．プログラム疲労の考え方を導入し，その値を判断基準の一つとして使用する方法の研究を進めている．

この問題を解決するために昨年度までは C 言語を対象としたプログラム疲労の研究を進めてきたが，今年度から Java 言語の疲労も視野に入れ研究を進める．本稿では Java の特徴から見た，プログラム疲労の原因を示し，さらにその原因を基にした Java プログラムの疲労の測定法について述べる．

2．プログラムの疲労

2．1　疲労の概念

プログラムの変更を繰り返すに従って，プログラムの構造が複雑になり崩れていくことをプログラムが疲労すると考えている．その原因を金属疲労と対応してひずみと不純物に分類する．さらに，ひずみはモジュール間の構造を悪くするもの，制御構造を悪くするもの，式の構造を悪くするものの3つに細かく分類した．またひずみと，不純物の蓄積量を疲労量と呼ぶ．すなわち，プログラムの疲労はひずみ・不純物が溜まることによって生ずるものであると考える[6]．

2．2　Javaプログラムの特徴

Java 言語はオブジェクト指向言語であり，高性能な機能を備えている．継承システム，抽象クラス，多様性，オーバーライドなどの複雑な機能を有している．また，Java 言語は C++や SmallTalk などのオブジェクト指向言語とは異なり，多重継承の禁止，インターフェースプログラムの導入や例外処理の明示的な表現，スレッド操作の明示的な表現を可能にしている．

また，Java 言語においては，class(クラス)という構文が，モジュールとしての機能を持っている．クラスとはオブジェクトの雛形であり，モジュールとは再利用，情報隠蔽，分割コンパイルの単位である．

*日本大学工学部
**日本大学大学院工学研究科

さらに，Java におけるクラス間の関係は，クラスの継承，インターフェースの継承，インスタンス変数へのアクセス，他クラスのメソッドの呼び出し，例外を投げるなどの関係があり特徴的である．

以上の Java 言語の特徴から C 言語に比較して，プログラムの変更に伴い，プログラムの疲労の原因も多くなると考えられる．

それらオブジェクト指向独自の特徴について，プログラムの疲労の面から次のように考える．

(1)　フィールドのインスタンス変数の数

これはクラスが参照している型の種類の数であり，これが大きいクラスは，他のクラス及び，インターフェースからの影響が大きいクラスである．他のクラスまたはインターフェースを変更した場合に影響を受ける可能性が高いと言える．

(2)　メソッド呼び出し

各メソッドの中で呼び出されるメソッドの数であり，この値が大きくなるのは，そのメソッドが逐次的で手続き指向になってしまっている，と考えられるクラスの保守性を考えると，この値は小さいほうがよいと言える．

(3)　継承の深さ

例えば，特殊化による継承（すでに存在するクラスの役割や責務を拡張するためのクラス継承）を考えた場合に，スーパークラス中の何らかのコンストラクタに変更が生じると，サブクラスにも影響を与えてしまう可能性がある．よって，継承は深すぎてはならないと考えられる．

(4)　メソッドのオーバーライド

オーバーライドの数が少ないほど，特殊化による継承が行われていると言える．特殊化によるサブクラス化とは，スーパークラスの機能を拡張して，新しい型を定義することを意味し，この継承は望ましい．逆にオーバーライドの数が多いと，実装上の継承が行われていると言える．これはサブクラスがスーパークラスのコードやインスタンス変数を便宜的に利用するために行うサブクラス化のことで，この継承は変更が生じた場合，何らかの不具合が起こる可能性が増える．したがってメソッドのオーバーライドは少ないほうがよいと考える．

(5)　try ブロックのネストの深さ

try ブロックのネストが深い場合，ソースコードが難解になる．内側の try ブロックの中で発生した例外が外側の try ブロックに対応する catch 節により捕捉されてしまうなどの誤りも予測される．

(6)　公開しているインスタンス変数の数

オブジェクト指向言語である Java はカプセル化の概念に基づいている．インスタンス変数はすべて非公開とする

ことを目的とし，必要な場合にアクセサメソッドを用意するようにする．

3．Javaプログラムの疲労の原因

前項で述べたようにJavaの特徴を考えて，Javaプログラムの疲労の原因を金属疲労[3][5]と対比させ，ひずみ，不純物に分けて抽出した(表1)．ここでひずみはモジュール間の構造，制御構造，式の構造の変化であり，不純物は論理的に実行または評価されないコードである．

表1．Javaのプログラム疲労の原因項目

疲労の分類		疲労の原因
ひずみ	モジュール間の構造	モジュール強度が低くなる（機能的強度・情報的強度・連絡的強度・手順的強度・時間的強度・論理的強度・暗号的強度）
		モジュール結合度が高くなる（データ結合・スタンプ結合・制御結合・外部結合・共有結合・内容結合）
	制御構造	ネストが深くなる
		if文の数が多くなる
		switch-case文が多くなる
		tryブロックのネストが深くなる
		if文中の条件式が多くなる
		再帰メソッドの数が多くなる
		完結されていないif-else文が多くなる
	式構造	マジックナンバーの数が多くなる
		if,while,for文の本文が{ }で囲まれていない
		インスタンス変数とローカル変数の名前が重複している数
		クラス変数の名前が重複している数が増える
		ローカル変数宣言から使用するまでの距離が長い
		メソッド呼び出しの入れ子の数が多い
		キャスティングされていない演算が多くなる
		1行に複数に記述文が書かれることが多くなる
不純物		使われていないローカル変数が多くなる
		使われることの無い無駄なインスタンス変数が多くなる
		行われることのない演算式，代入式が多くなる
		実際にthrowされない例外のthrows宣言文

Javaプログラムにおけるモジュール間の構造はクラスの強度，クラス間の結合度によって示されると考える．最適なモジュール設計は，モジュール強度を高くすること，モジュール結合度を低くすることとされている[4]．クラス間の結合度を示す尺度として表2に挙げた項目が考えられる．

表2．クラス間の結合度を示す尺度

クラス変数，クラス時メソッドの呼び出しの数
呼び出すクラス変数，クラスメソッドの種類の数
フィールドのインスタンス変数の数
すべてのインスタンス変数の種類の数
インスタンス変数またはクラス変数を用いたメソッド呼び出しの数
呼び出すメソッドの種類の数
継承しているクラス，インターフェースの数
継承が深さ
上書きしたメソッドの数
直接的サブクラスの数
インスタンス変数への統合的な参照数
一つのクラスで公開（public）しているインスタンス変数の数
一つのクラスで公開（public）していないメソッドの数

4．Javaプログラムの疲労量の測定

表1の各項目を，コード静的解析ツール[5]で測定し，改版前と改版後で比較することで，プログラムの特性の変化量，すなわちプログラムが疲労した量(疲労量)を測定する．

5．おわりに

今回，Javaプログラムのモジュール内，モジュール間に注目して，プログラム疲労の原因を分類し，疲労の測定法を提案した．

今後，実際にプログラムを変更して疲労測定を繰り返し，疲労原因の適確さや尺度の重み付けを検討する．また，プログラム疲労測定の仕組みを実現させることにより，プログラムの変更可否判断が可能になるだけでなく，現在のプログラムの品質，変更作業の見積もり，疲労しているプログラムの疲労回復，疲労を起こさないプログラム記法について検討できると考えられる．

参考文献

[1] 滝ほか:“Cソースプログラムの疲労度の一考察”，情報処理学会第60回全国大会 2P-05,3,2000

[2] 荒ほか:“Javaプログラムの監査法に関する一考察”電子情報通信学会ソサエティ大会 9,2001

[3] Charlie R.Brooks:“金属の疲労と破壊”，内田老鶴圃,1999

[4] GLENFORD J.MYERS Reliable Software through Composite Design ,1976

[5] (株)東陽テクニカ:コード静的解析ツール JTaster

[6] 梶谷他:”モジュール間の関係に注目したプログラム疲労の概念の拡張に関する提案”,第1回情報科学技術フォーラム,2002