ソフトウェア工学 64－3
（１９８９．２．２）

ソフトウェアバグと人間の能力との相関関係

清野浩一
ＮＴＴ交換システム研究所

ソフトウェア開発については経験と勘と度胸から脱却し、科学的に管理する必要がある。
本稿ではプログラマーによる人為的な問題にスポットをあて人間のソフトウェア能力とソフトウェアバグとの相関関係についての定量化の一手法を提案する。
本手法では人間の能力、人間の不安定さ、人間の気力、人間の体力をパラメータとし、バイオリズム関数で近似し、これらのパラメータを用いてソフトウェアバグと人間の能力との相関関係を表現する。

CORRELATION BETWEEN SOFTWARE BUGS AND HUMAN CAPACITY

Koichi SEINO
NTT Communication Switching Laboratories
9-11 Midori-Cho 3-Chome Musashino-shi,
Tokyo 180 Japan

We propose a quantitative method as the correlation between software bugs and human software capacity, which take notice of artificial problems for programmers.
This method presents software bugs by means of parameters of human capacity, human instability, human physical strength and human spirit, which approximate violism function.

1.まえがき

大規模なソフトウェア開発におけるソフトウェアバグの定量化については、従来システム規模が膨大なこと、プログラマーによる人為的ミスが多いなどのため、定性的解釈の域を出ない場合が多く、[1][2]少しずつ定量化の研究がなされいた。しかしながら、従来のソフトウェア管理手法では、適用条件が限定されているため、特に人間の能力については何も定量化されていないので限られたシステムにしか応用できなかった。

本稿ではプログラマーに人為的ミスを発生させる要因を考慮して、人間の能力とバグ収束過程との関係を定量化する手法を提案する。本手法による提案はバグの収束過程をモデル化し、定量的に把握可能としたため、プロジェクトの進行管理の点で意義がある。

2章では、従来のソフトウェア管理手法を開発計画とソフトウェア品質の定量化の観点から述べる。

3章ではソフトウェアバグ要因をプログラマーの質の点から整理する。

4章ではソフトウェアのバグ件数をプログラマーのソフトウェアバグ発見能力をパラメータとして定量化する手法を述べる。

5章では従来のソフトウェア管理手法と本稿で述べた手法を比較検討する。

2.ソフトウェア管理手法の動向

ソフトウェア開発を科学的に管理する方法については、従来種々の方法があり、ここではその中でも実績のあるQuantitative Software Management社のソフトウェアのライフ・サイクル理論に基づいてソフトウェア開発の工期、工数、コストなどを見積もり、
開発計画を立案するためのツール「SLIM(Software Life-Cycle Management)」とメトリクス社の開発中のソフトウェアの品質を計測・評価する手法「ソフトウェア品質評価/測定技法(SQAM=Software Quality Assessment and Measurement)」について特徴[3]を述べる。

2.1 SLIM

SLIMは過去のデータを基に、ある与えられた条件下で、マンパワー方程式を用いて、納期、コストの上限、投入可能なマンパワーといった外的制約条件を入力パラメータとして、現実性のある開発期間の範囲(最短期間と最低コストの解も含めて)を得る。また、この方程式で使われる総工数と開発期間については、パトナムの「ソフトウェア方程式」で求める。しかし、SLIMのベースになっているパトナム・モデルがすべてのシステム開発に適用できるわけではない。即ち、パトナム・モデルは過去に実施されたシステム開発の実績データをもとに構築されたものであり、新規プロジェクトがパトナム・モデルのベースになっている過去の実績データと大きく異なっている場合は適用できない。

注)マンパワー方程式

$$Y' = \frac{K}{t_d^{\,2}} \times t \times \exp\left[\frac{-t^2}{2 t_d^{\,2}}\right]$$

さエ'て分
と総Yっ積
要はは従で
必Kd。間
で、tる時
t)、あを
間人)で一
時(年間ワ
は一人時。パ
'ワはるンる。
Yパ又なマあ
、ン月とはで
でマ人大数値
こる(最工た
これ数が総し

2.2 SQAM

SQAMとはソフトウェアを計測し、評価するための技法である。即ち、ソフトウェアの品質を定量的に把握するには、ソフトウェアの品質を測る尺度と計測方法が必要で、この技法のことをSQAMと呼んでいる。そこで、品質を測るため、SQAMでは品質を12のカテゴリに分類し、各カテゴリ（品質要素）を23の品質基準に分類している。

SQAMは上記の品質要素に必要な品質基準毎に定量化し、要求仕様、機能仕様，設計、コーディング、テスト、運用と保守、ドキュメンテーションといった開発の各工程で利用する。そして、プロジェクトの進捗を管理する上ではこのSQAMでの結果が一定の水準以下であれば、工程を進まず前工程の作業を継続させる。

3. ソフトウェアバグの要因

本章ではプログラマーが原因でバグを発生させる要因を整理する。

(1) 人間の能力差

人間（プログラマー）の持っている資質に大きく依存するが、本稿ではプログラマーとしての経験を重要視することとする。なぜなら、新人とベテランとでは要求仕様からソフトウェア仕様に変換する能力及びバグに対する解析に必要なデータの収集に大きな開きがある。

(2) 人間の能力の時間的変動

人間はそもそも機械のように一定の動作を継続することはできない。即ち、体調、精神的不安定さ等により人間の能力は時間的に変化する。(3)

(3) 人間の体力

ソフトウェアデバッグを行う環境にも左右されるが、人間の体力差はプログラムの設計・作成、バグ解析の解析時間及びデバッグの事前準備の遅れ等によるデバッグ効率に大きく影響する。
（例えば、開発マシンの都合によっては徹夜が続くような場合がある。このようなとき、明らかに体力差が歴然とし、進捗に大きな差がでてくる。）

(4) 人間の気力

開発期間の終了間際と初期の段階及び中間の段階とでは気力の充実度に大きな開きがある。これは開発の初期の段階では仕様の不備等があるが、気分的には希望に燃えているし、また、終了間際の段階ではあと少し頑張れば楽になれるという状況が考えられる。(4)

(5) 稼　　働

ソフトウェア開発線表に依存するが、稼働の増減は個人の担当するソフトウェア規模に大きな差が生じる。

そのため、稼働が少なければ、個人が担当するソフトウェア規模が多くなると同時に、もともと個人が把握できるソフトウェア規模があるので、その許容範囲を越えればバグの解析能力が落ち、進捗が遅れることになる。

(6) ソフトウェア規模

ソフトウェア規模が大きくなればなるほど、ソフトウェアバグを誘発する危険性がある。これは個人が把握しなければならないソフ

トウェア規模が増えるとともに、どうしても分割した機能単位とのインタフェースミス等が発生しやすいためである。

(7) 開発期間

上記の(1)～(5)になんらかの形（体力の消耗、集中力、稼働の平準化等）で影響を与える。

4. ソフトウェアバグと人間の能力との相関関係

ソフトウェアバグは3章で整理したようにソフトウェア規模や開発期間とともに、人間（個人）の能力に大きく依存していると考えられるので、人間一人でソフトウェア規模Kstepを開発する場合に、個人のバグ発見能力をパラメータとしてソフトウェアバグ件数と人間の能力との相関関係を以下に示す。

4.1 人間のバグ発見能力の定量化

人間のソフトウェアバグ発見能力を定量化するには、人間の能力、人間の気力、人間の体力の3要素の結びつきを示す必要がある。

人間の能力とは知識やソフトウェアセンス等に大きく依存するが、ここではプログラマーの経験を重視することとする。

また、人間の能力として見た場合、個人差があるので、初期値が異なるとともに立ちあがり時期も大きくずれるし、他のソフトウェアバグ要因に影響をうけることも予想される。

人間の気力はシステムの開発過程において、充実したりしなかったりと精神的な要素が多分にありソフトウェアの品質に大きな影響を及ぼす。一般的には人間の気力は一週間或いは一カ月の間に上昇気運のときと下降気運のときが周期性を持って変化する。そこで、このような周期性を持った人間の気力をバイオリズム関数で近似する。

従って、人間のソフトウェアバグ発見能力は体調や感情に左右されやすく、落ちこんだり、充実したりというふうに時間的に変動する。

そこで、ここでは人間（個人）のバグ発見能力をN(t)とし、初期値をAとする。このAは個人のバグ発見件数とする。また、個人の気力をバイオリズム関数a(t)とすると、

$$N(t) = A \times a(t) \quad \cdots (1)$$

$$a(t) = ((\sin(t)-\cos(t))/2(1+t) \quad \cdots (2)$$

但し、t：時間、
A：正の実数

と定義する。

4.2 ソフトウェアバグと人間の能力との相関関係

開発するプログラム規模K(step)に対するプログラムバグ総件数Kαと開発時間との相関関係を個人のバグ発見能力に焦点をあて、ある時刻で発見できるバグ件数を表現すると以下のようになる。[5][6]

f(t)	：ある時刻で発見できるソフトウェアバグ件数
t	：開発時間
K	：ソフトウェア規模(step)
Kα	：総バグ件数
N(t)	：人間のソフトウェアバグ発見能力
a(t)	：人間の気力

とすると、

$$f(t) = \frac{\text{残りのバグ件数}}{\text{ある時刻でのバグ発見能力}}$$

$$= \frac{K\alpha - \int_0^t f(t)\,dt}{N(t)} \quad \cdots (3)$$

但し、$K\alpha = \int_0^\infty f(t)\,dt \quad \cdots (4)$

となる。
従って、(3)の両辺を微分すると、

$$\frac{1}{f(t)} f'(t) = \frac{1 + N'(t)}{N(t)} \quad \cdots (5)$$

故に、(5)に(1)と(2)を代入して、(5)の両辺をtについて積分すると、f(t)は以下のようになる。

$$f(t) = \frac{1+t}{\left|\sin\left(t-\frac{\pi}{4}\right)\right|} \times \exp \sqrt{2} \cdot A \times \left(\frac{1}{2} \cdot \log \left| \frac{1+\cos\left(t-\frac{\pi}{4}\right)}{1-\cos\left(t-\frac{\pi}{4}\right)} \right| + \frac{\pi}{8} \cdot \log \left| \frac{1+\cos\left(t-\frac{\pi}{4}\right)}{1-\cos\left(t-\frac{\pi}{4}\right)} \right| - \left(t-\frac{\pi}{4}\right) - \frac{\left(t-\frac{\pi}{4}\right)^3}{3 \cdot 3!} - \frac{\left(t-\frac{\pi}{4}\right)^5}{15 \cdot 5!} - \cdots \right) + C \quad \cdots (6)$$

$$C = \frac{-2K\alpha}{A} - 2A(\sqrt{2}+1) \times \exp \sqrt{2}A \left[\frac{\pi}{4} \times \log(\sqrt{2}+1) + \frac{\pi}{4} \right. \left. \frac{1}{18} \times \left(\frac{\pi}{4}\right)^3 + \cdots \right] \quad (7)$$

5. 従来のソフトウェア管理手法との比較

本稿で述べているソフトウェアバグと人間の能力との相関関係については、「経験と勘と度胸」から脱却し、ソフトウェア開発を科学的に管理する一手法を人間の能力という観点から提案している。そこで、本稿で述べた手法との比較を表1に示す。

各手法の特徴は

(1) SLIM：ソフトウェア開発の工期、工数、コストなどを見積もり、開発計画を立案する。

(3) SQAM：開発中のソフトウェアの品質を計測・評価する。

(3) 本稿の手法：プログラムバグと個人の能力との相関関係によりプログラムファイルの品質を評価する。

である。そこで、表1に各手法の比較をメリット／デメリットの観点から示す。

表1.各手法の比較

手　法	長　所	短　所
(1) SLIM	・（最短）開発期間が確定できる。 ・総工数が確定できる。 ・ｺｽﾄ/利益が確定できる。 ・CPU 時間が確定できる。 ・作業を分割することができる。 ・マンパワー所要量が確定できる。 ・リスクの範囲を明確にすることができる。	・適用条件が限定されていて、以下の条件では適用不可。 ⅰ）新しいプログラミング技法や言語。 ⅱ）小規模のソフトウェア開発 ①ソース行数が5000行以下 ②マンパワーが3人以下 ③開発期間が6ヵ月以下 ④総工数が20人月以下 ⅲ）機能設計までの工程 ・適用するのに必要なパラメータ量が多い。 ・プログラム品質の評価ができない ・ﾄﾞｷｭﾒﾝﾄ品質の評価ができない。
(2) SQAM	・要求仕様品質が評価できる。 ・機能仕様品質が評価できる。 ・機能設計品質が評価できる。 ・詳細設計品質が評価できる。 ・ｺｰﾃﾞｨﾝｸﾞ品質が評価できる ・テスト品質が評価できる ・運用と保守が評価できる ・ﾄﾞｷｭﾒﾝﾄ品質が評価できる	・開発期間が確定できない。 ・総工数が確定できない。 ・マンパワー所要量が確定できない ・コストが確定できない。
(3) 本稿の手法	・ソフトウェア品質の評価ができない。 ・開発期間が確定できる。	・仕様品質が評価できない。 ・設計品質が評価できない。 ・ドキュメント品質が評価できない ・運用と保守の評価ができない。

従って、表1よりＳＬＩＭとＳＱＡＭの両手法ではソフトウェアのバグ件数から見たソフトウェアの品質を評価していないので、ある時点からさき、どのくらいの開発期間がかかるのか不明である。即ち、プロジェクトの進行管理が難しい。

本稿の手法ではプログラマー（またはシステム・エンジニア）の体力、能力、精神力、気力等を客観的に定量化し、信頼性に対する大きな要因である人的な要因を考慮したソフトウェアの品質を定量化した。その結果、本手法はソフトウェアの品質をベースにしてプロジェクトの進行管理に役立つことを明確にした。

6. あとがき

ソフトウェアの品質においては人間の役割が大きい。そのため、ソフトウェアの中に、種種の人為的ミスが避けられないのが現状である。本稿ではこれら諸要因の影響をソフトウェアバグと人間の能力との関係から評価し、ソフトウェアの品質を計測する手法を提案した。

一般に、ソフトウェアを開発するときは複数の人間が分担して開発するので、今後は複数の人間（プログラマー）間の関係を考慮した人間のバグ発見能力を定量化していく予定である。

7. 参考文献

（1）上村、樋口他：「ソフトウェア　信頼度成長モデルの検証」
電子情報通信学会・信頼性研究会　R88-4

（2）砂塚、三野村：「ソフトウェア品質管理技術（SQMAT）」
電子情報通信学会・信頼性研究会　R87-6

（3）　　：「米国に見るソフトウェア開発を科学的に管理する方法」
日経コンピュータ特集‘ソフトウェア品質管理への挑戦’（1984.10.1）

（4）藤垣　　：「ソフトウェア技術者の工程別負担分析」
情報処理学会　第37回全国大会　3L-2

（5）堀籠　　：「ロジスティック曲線によるDuaneモデルの改良」
電子情報通信学会・信頼性研究会　R87-1

（6）三浦　　：「ソフトウェアの品質評価法」
日科技連出版社，1981