私はソフトウェアを測るのが好き！

二上貴夫
（株）東陽テクニカ

プログラムの善し悪しを作法という観点で議論しても水掛け論になる．善し悪しを定量的に測ることが必要だ．高信頼性システムのプログラムを測ってみると，日本製のものの品質は欧米のものの半分にも達していない．品質を向上させると同時に日本がソフトウェア生産の分野で世界をリードできる具体策を提案する．

ふっと魔が差して測ってしまう

私は測ることが好きである．なんで好きかといわれても困ってしまうが，仮説をたて，仕掛けを作り，額に汗して／寒気に震えながら測定し，結果の意外性にびっくりするといった起承転結が好きなのである．テーマはこだわらない，ちょっとした動機があると軽いのりで測ってしまう．いわゆるふっと魔が差してというやつだ．

眠さの程度を知るために眠さ測定課題なる四則演算問題の正解数を測ったし，スポンジケーキの焼け具合を非接触で知ろうと思ってケーキ厚みの時系列を測定したこともある．渋滞を定量化しようと思ったこともあった．自分がブレーキをかけている時間とアクセルを踏んでいる時間の比率をGo/Stop比と呼びあちこちで測ってみた．また，アイススケートの速さを測ろうとしてスケート靴に細工をし，電流テスターをにらみながらスケートリンクを右往左往したこともある．

こうしてあれこれ測ってみると，大方の場合には「まあ，こんなもんやろ」程度の結果が得られる．収縮率の大きなケーキは，きめが荒くてしっとり感がないなどケーキ焼きを知っている人なら誰でも同意してくれそうな結果である．しかし，時として「あれっ，なんでやねん？」と絶句したくなるような結果に遭遇する．先ほどのGo/Stop比の例では，都内では値が0.3以下で渋滞と感じた．これに対して郊外では0.6でも渋滞と感じてしまう．すなわち，都内では100分間の内たったの30分しか走れないでも能率的な移動ができていると感じ幸せなのに対し，田舎では60分間走れていても渋滞と感じ，イライラがつのるのだ！　この「なんでやねん」発見のスリルがたまらない．スケートの測定を継承した母校の後輩は，それを科学研究のテーマとして「なんでやねん」を繰り返した．そして，とうとう長野オリンピックで清水選手が履いたCFRP高剛性シューズの開発に結びつけた．

日本のソフトウェアの品質が低いのは，なんでやねん

こういう性向を持つ人間がソフトウェアの世界に首を突っ込むと，ソフトウェアを測ってみたくなる．というわけで私は，C/C++言語で書かれた日本のソフトウェアを測りつづけている．測定対象は，アプリケーションを問わず，高い信頼性が要求されるC/C++のプログラムである．逆に言えば，信頼性よりも利便性を優先するようなプログラムは対象としない．たとえばゲームは対象外，CADやCASEは半分くらいが対象，自動車，医療機器，FAロボットはすべて測定対象である．専用OSなども数こそ少ないが測定対象だ．

この測定から分かった「なんでやねん？」は，日本で書かれたC/C++プログラムの品質が欧米のものに比べて，半分にも満たないという事実だ．これは欧米の高信頼性プログラム50件約100万行と日本の相当プロジェクト120件200万行の欠陥率を比較した結果である．日夜真剣に製品を開発されているプログラマ諸氏には申し訳ないが，その品質は決してハードウェアとしての極薄パソコンやカメラと同列で誇れるものではない．つい「ハードは良いのに一体ソフトはなんでやねん」と言いたくなってしまう．どうも日本は低品質のソフトウェアを使って高信頼性システムを実現する技術を身につけているらしい．

プログラム欠陥減少にメトリックスは有効だ

さて，ここまで言ってしまった以上

は，その根拠を多少なりとも示しておく必要があろう．我々がやっているのは，プログラムの静的な欠陥を拾い上げ，その数をメトリックとして品質を評価するという根気のいる仕事である[1]．実務上の理由から静的な欠陥は，きわめて深刻な欠陥，深刻な欠陥，軽い欠陥の3段階に区別している．

インデントに一貫性がないのは誤った保守作業を招くので軽い欠陥の1つに数える．また，1つの関数の内部に戻り値を指定したreturn文と戻り値のないreturn文が混在している関数は明らかにどちらかが間違っている．呼び出す場所によって引数の数や型が違っている場合もある．これらのソフトはいったい何を計算するのだろうか？「たまに出鱈目を計算する」のである．これを3段階のどの欠陥ランクに指定すべきか議論の余地はないと思う．

こうして欠陥を測って密度を求めてみると，きわめて深刻な欠陥の出現率はおおむね0.1から1欠陥/KLOCである．これは，デバッグ途上のちぐはぐなプログラムではなく，製品としての動作試験が終了した段階のプログラムの調査結果である．64Kバイトの ROMで動作するプログラムは，大まかに約1万から3万実行行くらいのCプログラムで成り立っている．日本のお家芸の製品群の多くがこのサイズに入るだろう．そうしたサイズの製品にはだいたい1個から30個くらいのきわめて深刻な欠陥が含まれているのである．厄介なことに，こうした静的な欠陥は出荷前の実行テストでは発見が難しい．それは大勢の人とか長い実行時間をかけるとか，ここ一番というときに初めてバグとして発現する．ヨーロッパの大型ロケットであるアリアン5がたった1行の欠陥代入文のトラブルで自爆した経緯はよく知られている (http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html)．これを読むとアリアン5のソフトウェア品質の管理に問題があったことがよく分かる．きわめて深刻な欠陥は一度発現したらその破壊力たるやシステムにとって致命的である．ケーキ焼きと違って社会的に重大な影響を及ぼすシステムではソフトウェアを測る行為が障害防止に役立つと痛感する．

本製品のソフトウェア品質

コード	80KLOC以上
テストカバレッジ	99.7%以上
欠陥率	0.01個/KLOC以下

将来のソフトにはこんな品質表示ラベルがつくかもしれない

もう1つ，我々の発見したことは，古典的なメトリックス（複雑度とか行数，コメント率など）の良し悪しとソフトウェアの品質には強い相関があることだ．複雑度の高い関数の数が多くなるほど局所変数の値が不定のままで参照されるケースが増えるし，ネストの深いプログラムは試験が甘い，もしくは試験そのものが行われていないという結果だ．また，複雑な関数を多く作ったプロジェクトほどテストへの関心が低い傾向もある．テストカバレッジなどというメトリックは今の日本では，ほとんど聞かれない言葉である．こうしたプログラムの実行品質は当然のように低かった．

メトリックスは測定精度や再現性からして役に立たないという意見をよく聞く．確かにソフトウェアを測るというテーマは，工学上の厳密さを先に求めてしまうと学問としての成立が難しいと思う．しかし，実学としてメトリックスは有効だ[2]．

究極の品質向上ソリューション

ところで，日本製C/C++プログラムの品質，これをほったらかしておいてよいのだろうか？ ハードウェア製造立国として日本がすごいのは，個々の製品のばらつきを極限まで標準値に収束させる設計上の発想や製造の経験則や細かな対症療法の積み上げ努力だったと思う．規準範囲に入っていれば合格などというアメリカ流の甘い製品チェックをしていたのでは今日の日本はなかったはずだ．

ここでプログラム品質向上のための荒療治を提案したい．品質を開示してしまうことだ．それもアリアンよりも一歩進めてバグ発現以前に製品出荷の段階でメトリックスを表示するのだ．ハードウェアの世界ではMTBF（故障や不具合に至るまでの平均時間）指標をつかうという理性／常識が働いている．ソフトウェアの世界では誰かが歴史のどこかで，ソフトは正確に書けば絶対に間違わないのだから欠陥は許されないという信念を社会に植えつけてしまったようだ．「なんでやねん？」

論理は時間で劣化しないという真理を拡大解釈したのだろうか．あるいは「正確に書けば」というこの信念の前半が達成できないことをどこかで忘れてしまったのかもしれない．およそ生身の人間がなした結果（プログラム）に100%精密なものなどありえないことは常識以前の問題なのに．こんな常識以前の過ちをいつまでもほっておいたら，まずい．産業製品としてのソフトウェアになんとか欠陥許容値を認めることはできないだろうか？ そして

注釈：「なんでやねん？」の出典

この「なんでやねん？」は，娘の通う小学校の先生の教育手法である．体積の概念の授業で，先生は直方体と立方体の大きさを比較するクイズを出す．子供たちは答えと共にさまざまな理由を提示する．先生は理由について「なんでやねん？」と聞きながら，ビニール製のナンデヤネンハンマーを振りつつ体積の比較方法から計算法，単位系まで教えてしまう．娘の授業を参観しながら私もついナンデヤネンとつぶやきながら体積の比較計算にはまっていた．どうもナンデヤネンの語感には発想を誘うリズムがあるようだ．

欠陥を測り，誰にでも見える状態にしてその数を制御することで逆に品質の向上をはかれないか？ こうすればデバッグは，慙愧に堪えない後ろ向きの仕事ではなくなろう．また，機能満載バグ満載の製品よりも必須機能のみだが低欠陥な製品が一般市場で優位を占める可能性もある．

ちなみに私は最近，アンインストールの方法がはっきりしないツールは便利そうに見えてもマシンにインストールしないことにしている．これだけで私のシステムトラブルは半分に減り清々した日を過せている．

もちろん日々しのぎをけずっている一般企業が即こうした態度に出られるとは思えない．公的な機関だってバグの開示など許可するはずがない．多分，経団連あたりで今年はこの業界でやりましょうくらいの意思決定と指導力を発揮していただくことが要求されよう．同時に欠陥の詳細を開示した企業の免責システムも考慮する必要もあるだろう．大仕事だ．しかしこれをしないことには，現場は永遠に対症療法に明け暮れることになる．米国においてもこんな思い切った処置をしているところなど浅学の故かまだ聞いたことがない．日本がソフトウェアの製造技術でユニークなことをやれるチャンスに思えるのだが，皆さんのご意見はいかがだろうか？

学会はソフトウェア予防衛生学を

ただし，こうした実力行使の前段としてソフトウェア予防衛生学の確立が欲しいと思う．今の日本で起きているいろいろな症例を採取し診断を下しながら結果を体系化する．そしてメトリックデータは当然のことだが公開する．これも困難だし根気のいる仕事だろう．まさに政府直属の機関が施策として行ってもよいような事業である．

今日，赤痢菌に対する予防衛生システムはしっかりしている．この防衛システムは誰が作ったのだろうか？ 誰か先駆的な努力をした人物がおられるのだろうが仮にいたとしても，その背後に多くの医学関係者の共同作業があったはずである．ここで示したような作戦を最終的に実行するのが誰かは別として，少なくとも実行へのきっかけを作るのは本学会の役目と思える．

メトリックスは，単なるプログラムの品質管理にとどまらないソフトウェアの予防衛生学を確立するための強力な武器になろう．

参考文献
1）東陽テクニカ: QACクリニック, http://www.toyo.co.jp/ss/pdf/qac_clinic_1_j.pdf
2）Special Issues on "Measurement", IEEE Software Magazine, Vol.14, No.2(Apr. 1997).

（1998.6.1）

私は測られるのは嫌いです

大野　晋
日立ソフトウェアエンジニアリング（株）

私は自分の作ったプログラムを測られるのは嫌いです．というのは冗談ですが，日本のソフトウェアの品質に関する議論に居ても立ってもいられず，反論致します．

日本のソフトウェアが諸外国のものに比べて品質が悪いかどうかは別として，品質表示の提案は一見良さそうに思えます．しかし，ソフトウェアの特性を無視されているように思えます．ソフトウェアはたとえ，1件のバグであっても致命的なバグであれば大変な結果を及ぼすものです．このため，日本のソフトウェアハウスであっても，「バグが含まれている」のを知って，出荷はしないものと考えます．少なくとも，失敗が社会的に悪影響を及ぼすようなソフトウェアについては，その程度の品質管理をしているものと信じております．バグが本当にないかどうかは別にして，この意味では，出荷されるソフトウェアに「バグはない」のです．こう考えると，ソフトウェアの品質表示は少しうまくいきそうもありません．

そこで，次の2つのメトリクスに着目してはいかがでしょうか．

1つは，人のメトリクスです．ソフトウェアの品質は結局は人の問題だと思います．今まで，我々は個人攻撃を避けてきましたが，人の品質に関するメトリクスは避けては通れない気がします．個人的要因によるバグの作りこみ率，摘出率や取り残しの可能性，作り込むバグの傾向を押さえることは，その個人の成長の上でプラスに働くものと思います．また，バグと作り込み

情報処理学会第57回全国大会公開パネル討論会
テーマ：　日本の風土と21世紀のソフトウェア
日　時：　1998年10月5日（月）15：40～
資　料：　インタラクティブエッセイ，4月号～9月号
詳　細：　http://www.ipsj.or.jp/magazine/interessay.html

者の関係を見ていくと，必ず「またやる」ものなのです．これは，学校の卒業と共に，個人の傾向記録を付けるしかありません（取扱いは難しいですが）．

2つめは，組織のメトリクスです．人が集まって組織を作るわけですが，この組織がまた，バグと深い関係がありそうです．少し昔，若手プログラマーの研修のお手伝いをした際に，同じ課題を3チームに出しました．Aチームは，比較的にプログラミングが「できる」者達で，2〜3名が中心になり，どんどん作り始めました．Bチームは，平均的な人達で，ちょっとミーティングをして，すぐに手分けをして作り始めました．Cチームは，「大丈夫かな？」というレベルの人達で，何をやったらよいか分からずに時間を費やしていました．そこで，私は一言「皆でよく話し合って決めなさい」とだけ指示したところ，がやがやと相談しながら課題を始めました．さて，その結果，早く課題を動かした順番は，A，C，Bの順で，CはAとほぼ同時にできました．

このように，プログラムというものは個人の力だけではなく，組織の力により品質に差が出そうです．しかし，今までこの差を測った人はいないように思います．このような新しいメトリクスを見つけてはいただけないものでしょうか．

(1998.6.30)

当たるも八卦，当たらぬも八卦

三嶋良武
（株）三菱総合研究所

「ふ」っと魔が差して測ってしまう」には，思わずふむふむとうなずいてしまいました．さまざまな物理現象や社会現象も，データを詳しく見ると思わぬ法則が隠れていることが多いのと同様，ソフトウェアについても，まだまだ我々が気付いていない法則がたくさん隠れているような気がします．

私は最近ソフトウェア工学とは少し離れているのですが，大学時代から開発方法論やソフトウェアツールなどに関することをやってきました．そんな関係で，静的解析ツールや，メトリックス計測ツールも開発したことがあります．その時の経験を思い出すと，二上さんの「プログラム欠陥減少にメトリックスは有効だ」という主張には，かなりの程度賛同できます．

「日本のソフトウェアの品質が低いのは，なんでやねん」と書かれていますが，これに関しては，少し疑問も感じます．これは，エッセイにも書かれているように実際にプログラムを静的解析ツールにかけてみた結果からのことと思いますが，実際に実行してみても半分の信頼性しかないのでしょうか．たとえばワープロソフトを例にとれば，日本製と欧米製で品質に大きな差があるとは感じられないのです．信頼性とはちょっと離れますが，以前エディタを作ったとき，私も魔がさして，ユーザが利用するコマンドの傾向を測ったことがありました．その結果は，頻繁に使われるコマンドは全体の1割以下でした．それからすれば，ワープロでも，通常使っている部分は全体からみればかなり狭い部分でしょうから，それ以外の所に致命的なバグが潜んでいる可能性は十分に考えられます．ゲームは対象外とされていますが，ゲームソフトはテストがかなり難しいにもかかわらず，個人的にはかなりバグは少ないような気がします(もっとも，バグがあっても気が付かないのかもしれませんが，少なくともハングしてしまったという経験はあまりありません)．一度測ってみたいものですね．

静的な欠陥の量がある程度品質に比例することは理解できるのですが，この辺のデータをきちんと測り，蓄積してゆくことがきわめて重要と考えています．このようなテーマは論文ネタになりにくいので，なかなかきちんと継続的に研究する機会が少ないと思いますが，「学会はソフトウェア予防衛生 学を」とまではいかなくても，地道な研究にも十分に光をあてて欲しいと願っています．

メトリックスを採用する場合には，その尺度が明確でないと混乱が生じます．最近食品に賞味期限という表示がありますが，この解釈も人それぞれで，お店側にとっても，消費者にとっても無駄が生じているケースも多いようです．近い将来品質表示ラベルが付くか否かは別としても，メトリックスの持つ意味というか，指標としての共通性を高める努力を皆でしてゆく必要があるでしょう．

最後に，ソフトウェアの信頼性に関して最近ふと疑問に思ったことを書きます．その昔に使っていたパソコンやWSは数MIPS以下でした．それに対し，最近のマシンはその数百倍に相当する実行速度を備えていると考えられます．無駄なコードを実行していないとすれば，単位時間あたり数百倍のプログラムが実行されていることになります．とすると，もしプログラム中のバグの密度が変わらなければ，使い物にならないほどバグに遭遇するハズです．この話しをしたところ，最近のソフトウェア(特にGUI部分の多いもの)はライブラリなど，アプリケーションプログラマ以外の書いたコードがかなりの部分を占めているから機械語レベルで見た時のバグの密度はかなり下がっているので帳尻が合っているのでは，との意見がありました．とすると，日本製のライブラリを使うのはもって

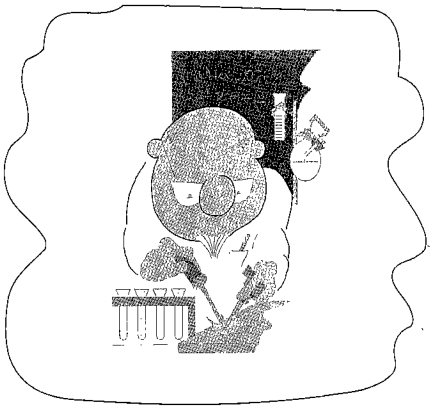
のほかということになりそうです．私も品質表示ラベルが欲しくなってきました．私も聞きたい，「なんでやねん?」

(1998.7.15)

好きこそものの上手なれ？

松本健一
奈良先端科学技術大学院大学

私も大学院生の時に魔が差して以来，ソフトウェアやその開発過程の計測（ソフトウェア計測）を主要な研究テーマとしてきました．好きで続けている最大の理由は，二上さんのおっしゃるように，ほとんどが常識的な結果の中にあって，「おや．計測のミスかな？」と思えるような意外な結果を見つけ出し新たな仮説や技術へと育て上げていく醍醐味が計測という作業の中にあるからかもしれません．ただし，同好の士はなかなか見つかりません．最近は学生と共に計測実験を行うことがほとんどですが，計測結果を手にした学生に「何か面白い結果が見つかった？」と尋ねても「いえ特に．」とだけ返答されることの多い毎日です．「ひょっとしたら重大な事実がこの計測結果の中に隠れていて，君が今それを見つけなければ永遠に明らかにされることはないぞ！」とつい怒鳴ったりもしますが，重大な事実が隠れている確率と自らの指導力の低さを冷静に考えれば二の句は継げません．余力がある時には自ら計測結果とにらめっこすることもありますが，大抵は，せっかくの計測結果がお蔵入りとなってしまいます．



さて，愚痴はこれぐらいにして，本題の「ソフトウェア品質表示」についてですが，大いに結構なのではないでしょうか．二上さんが注目されている「静的な欠陥」は，プログラムコードさえあれば機械的に計測可能のようですから，出荷前に最低限チェックすべき項目としてはいかがでしょうか．いわゆる「動的な欠陥」の計測は時間とコストがかかりますが，ソフトウェア信頼度成長モデル（Software Reliability Growth Model）などを用いた残存バグ数やMTBFの推定値は，計測体制が信頼できるしっかりしたものであれば，それなりに説得力があると思います．また，「品質≒バグ」という図式では少し古い印象を受けるので，使いやすさや保守のしやすさなどの指標も入れてはどうでしょうか．オブジェクト指向開発を行っている人々は，プログラムコードレベルではなく要求分析や設計の段階での計測結果を重視すべきであると主張するかもしれません．

「このソフトウェアにはバグがいくつ残っていますよ」といった後ろ向きな捉え方ではなく，「これだけの知識と技術を投入して開発したソフトウェアですので，安心してお使いください」といった前向きな捉え方ができるようになれば，「ソフトウェア品質表示」の実現性は高いと思います．確かに，現時点では，ソフトウェア計測を行っていることを積極的にアピールしているメーカは少数です．しかし，実際には，何年も前から計測結果を蓄積し，ソフトウェアの出荷判定や外注先から納品されるソフトウェアの検査に利用しているメーカは数多くあります．その中から品質表示を始めるメーカが出てこないとは限りません．ソフトウェア計測には多くの時間とコスト，さらに，地道なデータ蓄積が必要となりますから，自社の品質表示がデファクトスタンダードにでもなれば，いろいろな意味でメリットは大きいと思います．

ソフトウェアが工業製品である以上，計測や品質表示は避けて通れないものだと思います．ただし，ソフトウェア計測は容易な作業ではありません．計測対象に関するある程度以上の知識，具体的な計測方法やツールを作り上げる技術，データ収集を効率よく，かつ，正確に実施する技術，そして何よりも，計測結果から因果関係や法則（の断片）を見つけ出す分析力と洞察力が必要となります．これらすべてを1人の人間が持ち合わせているとはとても思えません．たとえ持ち合わせていたとしても，変化の激しいソフトウェアの世界では，新しい知識の獲得と柔軟な思考力の維持に多くの時間と労力を要します．冒頭に学生への愚痴を述べましたが，計測結果からユニークな発見をするのは，そうは言っても新しい知識と感性を持った学生です．「好きこそものの上手なれ？」というタイトルには，好きなだけでは計測はなかなか難しい，という消極的な意味だけではなく，ソフトウェアにかかわるさまざまな人々の知識や技術を（計測の好き嫌いにかかわらず）持ち寄ることが，これからのソフトウェア

計測に必要である，という思いを込めました．

(1998.7.17)

出荷するソフトにバグはない，危険性があるだけだ

二上貴夫
(株) 東陽テクニカ

●密かな欠陥は出荷される

私も大野さんと同じく開発者が生きたバグを承知で危険なソフトを出荷などしないと信じます．私が言いたかったのは，欠陥はバグとして認知されないかぎりどうしても出荷されてしまうという問題なのです．こうした欠陥はハードウェアのひび割れのようなもので，事故に至る場合もあれば，無事故のまま製品の寿命をまっとうする場合もあります．欠陥を抱えたまま出荷がかけられる時，開発者は次の2つの立場をとるのが普通です．

第1は，バグが出るかもしれないと予感しつつも出荷している立場です．これは，経験のあるリーダに多い．危ないと思いつつも明確なバグとして開発チームに問題を提示できないゆえに出荷を止める理由が得られないケースです．予感ぐらいで納期や経済的な要請は蹴れないのですが，悪い予感はだいたい的中します．私の知る範囲でも出荷後に重大なバグが発見され，改修がすむまでシステムが危険な状態にあったケースはいくつもあります．また，出荷レベルのソフトに対して開発責任者が予感として持っていた欠陥の存在が私たちの行った解析によって実際に発見されています．

第2に，品質を確信して出荷してしまう場合です．これは制御系などアプリケーションの専門家でプログラミングは必要だからやっているだけという人に多い．この人達がソフト開発を行うとプログラミング言語への誤解から生じる危険な記述，CPUやメモリのアーキテクチャに起因するプログラマが予想していなかった問題などが密かな欠陥として出荷されてしまうのです．蛇足ながらこの種の欠陥を私が発見し指摘してもそれが無視されてしまうことがあるのは実に悲しいことです．無視をする理由は，実行テストで問題がないからというケースがほとんどでした．

このどちらの立場でも出荷に当たって欠陥の出荷はまったく阻止できないことになります．担当者に悪意はないもののバグとして発現する可能性を持った欠陥は出荷され続けているのです．

●海外との比較は本当か？

次に，日本のコードはそんなに品質において問題があるのかという疑問を皆さんが示されているので根拠の概要を示しましょう．我々は，北米やヨーロッパの高信頼性プログラムの品質評価に用いられたアルゴリズムをそのまま日本で適用しこの結果を得ました．このアルゴリズムはたかだか5万行で，それ自体Cで記述されており重要な問題を適切に自動抽出してくれます．さらに自動抽出の対象としたコードの一部についてベテランが目視検査を行い両者を比較しました．この検証により，アルゴリズムはアプリケーションの意味を査読できないが（あたりまえか）プログラム言語のアーキテクチャから見たコード品質チェックとしてはきわめて良好であることが分かっています．ここでの品質チェックとしては，C言語規格の不備や曖昧性に抵触した表現はないか，コンパイラ自体の規格準拠精度に依存する表現はないか，プログラマの意図と異なる実行コードを生む可能性はないか，過去の失敗からの経験則に抵触しないかなどさまざまな視点があります．この良し悪しを数え上げた定量結果から半分以下の品質と評価したわけです．

●コード以外の侵入経路と予防メトリック

皆さんに指摘された通り，品質指標としてのメトリックはバグや欠陥の計数といったコードを直接の計算根拠とする以外にも各種の方法が考えられます．システム品質は，開発担当者が書き下すコードだけで決定できないのも確かです．例として，松本さんが指摘されているオブジェクト指向の観点からは，アプリケーションの視点での潜在欠陥とアーキテクチャ視点での潜在欠陥を分けることで性質として大きく異なる品質向上のアプローチが可能です．この場合，アプリケーションの視点ではコードとはまったく無関係に品質が決まります．将来はこの辺も含めて品質の制御技術を確立したいものです．ヨーロッパの自動車業界がMISRAという組織で自動車に搭載するソフトの品質チェックと要求される信頼度に応じた開発手法の明確化をしたのも松本さんのような思考に沿った分析の結果かもしれません．

この領域では，どのような測定をすると効果が高いかを判定できるデータを今後も着実に積み上げていくことが必要でしょう．

(1998.7.19)

～ 議論の続きは，次のURLをご覧ください．http://www.ipsj.or.jp/magazine/interessay.html ～