ソフトウェア・グリッチの予防
Preventions against Software Glitches

冨永　章
Akira Tominaga
日本アイ・ビー・エム株式会社
IBM Japan, Ltd.

要旨

ソフトウェア規模の増大に伴い，ちょっとした間違い（グリッチ）が大事故に繋がるリスクが高まっている。組込み分野の欠陥は意外な結果となり目立ち易い。しかし，大企業の情報システムはもっと複雑な場合が多く，また機器のように比較的閉じた世界にはない。ソフトウェアの複雑化はグリッチを招き，社会問題ともなり得る。設計までの考慮漏れの場合はテストもされないため，防止策の立案は容易ではない。保険制度等はあるが，予防ではなく賠償策である。本稿ではグリッチの予防に必要な事を整理し，最近のプロジェクトで効果のあった策を示す。

1. はじめに

金星探査機マリナー1 号（1962 年）は，FORTRAN のループ処理でカンマをピリオドに間違えたために，失敗に至った。その後，カナダの放射線医療器 Therac-25 被曝事故（1985-1987 年），米長距離電話の不通（1990 年），パトリオットによる米軍味方誤爆（1991 年 28 名死亡），アリアン 5 型ロケットの失敗（1996 年）などは，よく知られたソフトウェア・グリッチの例である[1]。

最近では，携帯電話，2004 年米大統領選投票システム [2]，車のリコール [3]，旅客機の異常飛行[4]，空港 X 線検査麻痺 [5]，テレビ受信機の障害[6]など，報道例は枚挙に暇がない。これらの多くは組込みソフトウェアのグリッチであるが，生じる結果の意外性のため，報道の対象になり易いようである。

一方，エンタープライズ系のソフトウェアでも，数え切れないバグの中で，運の悪い間違いが影響の大きなグリッチとなり，社会的な関心事にもなっている。以下 2 章で，グリッチが生じる要因を分類し，どんな性格の間違いの予防が難しいか示す。3 章では，ソフトウェアに間違いを起こし易い他の背景を挙げる。4 章に今後の危険性を述べ，5 章で予防に必要なことを整理し，最近の予防例を説明する。

2. 要因の性格による分類

ソフトウェアの間違い防止の観点から，誤りの原因を作り出す要因をその性格で分類してみる。

2.1 単純なうっかりミス

古くからある間違いは，①変数の値超過，②配列の要素数超過，③初期化漏れ，④ゼロでの除算，⑤無限ループ，⑥番地等の仕様違反，⑦緩衝域不足，⑧属性変換ミス等で，ほぼ言語に依存しない。また，PC 等では⑨メモリーリーク（ゴミ掃除もれ），組込用小規模プロセッサーでは⑩スタック数超過等がある。コンパイラー，ハードウェア， OS，ミドルウェアが事前に検出することが殆どである。

2.2 知識不足による間違い

情報システム構築の基礎知識欠如による間違いは次のようなもので，上記に比べて単純ではない。

(1) 入力チェックの不備

アプリケーションでのチェック対象に，属性や桁数等の形式，検査数字やハッシュ・トータル，上限値／下限値，他の入力との相互関係，ファイル内容との関係などがある。検査対象は当該項目の処理経緯に依存するから，プログラミングではなく仕様の段階で定義されねばならない。

例えば，インターネット株取引の「売り指値」で，現市場価格から±20%を限界とし，下限未満をはねると想定する。もしこのチェックが漏れていた場合には，それだけ修正しても普通は不完全である。同じ要因で，「買い指値」の上限チェックも脱落している疑いが濃厚である。

(2) 障害対策の不備

障害対策には，フェールセーフ，フェールソフト，構成による RAS，フールプルーフ等の知恵が種々ある。しかし現場の常識かといえば，最近は少し疑わしい。例えば「安全のため開扉をスイッチで検知する場合，スイッチの壊れ方（開放・短絡・両方）に基づき，どの状態で検知するかを決める」などが理解されない現場では，安全に関わるグリッチを作り込むことになりやすい。

(3) デッドロックや排他ミスなど

設計段階で配慮しないと，発見後の対応はリソース使用の逐次化となるため，大きな無駄が生じる。しかし，最近の進んだ開発環境では，排他制御を知らない技術者がいても不思議ではない。

(4) プログラムの複雑化や錯綜，自己流の方法

オブジェクト指向やモジュール凝集度・結合度等の理解を欠くと，複雑化し欠陥を作り易い。また，第三者に通じない自己流の記述法が使われると，レビューにも耐えられず，潜在バグも検出されにくい。グローバルで標準的な手法の適用が必要な理由でもある。

以上の（1）（2）は無知が原因なので予防が難しい。考慮漏れにはテスト･ケースも作られない。設計のウォークスルーやインスペクションで検出されることはあるが，その後は潜在化しグリッチの元となる。他で生じた問題を分析する際に，要因に遡って同類の間違いを探すと発見できることがある。

2.3 要件の誤解

要件の誤解による欠陥は，開発側がどうテストしても検出されない，誤解が解けない限り重要な間違いのままとなる。しかしグリッチの観点だけでみれば，この種の間違いは「思いもよらない大事故」には繋がり難い。誤認した仕様でも，それなりにテストされ，結果がどうなるか分っているからである。

以上のように間違いを要因の性格で分類するなら，予防が一番困難で危険なのは， 2.2 節に記した「情報システムの基礎的ノウハウの欠如により生じる間違い」ということになる。

3. ソフトウェア・グリッチを産み出す他の背景

予防を考えるには，以上のように現場で間違える要因のほかに，現在のソフトウェア開発がおかれた環境にも留意する必要がある。

3.1 ソフトウェア技術進歩の相対的な緩慢さ

図 1 に示すように IT やネットワークの技術進歩は速く，5～7 年に 10 倍の割合で向上してきた。これに比べ，ソフトウェア開発の進歩は極端に遅い。ハードウェアの進展で IT 適用が拡大したが，開発技術が追従していない。開発の大幅な進歩がないと，ミスを完全には防ぎ難い状態が続く。

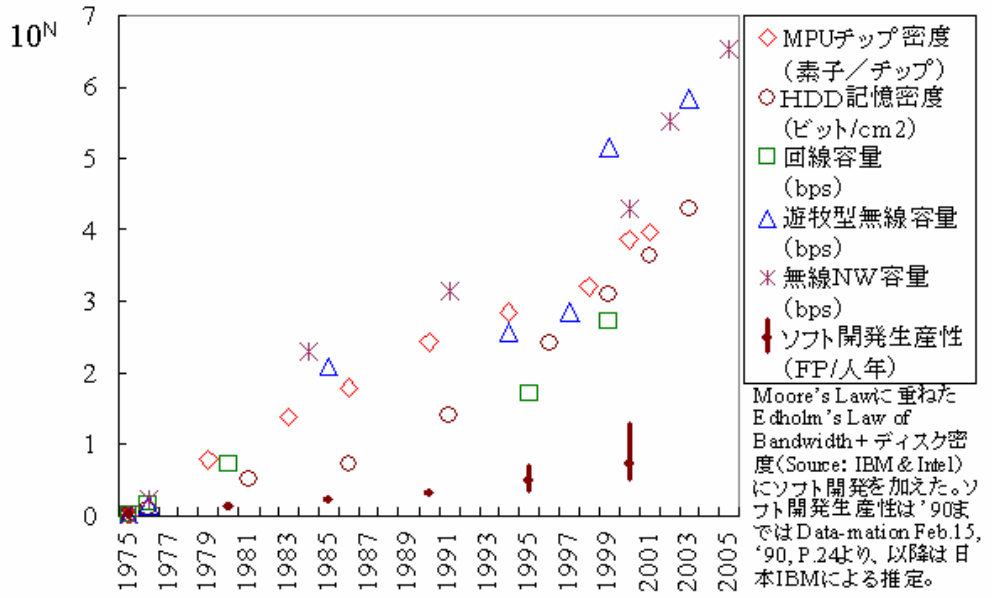

図 1　IT の進歩（1975 年を 1 とした場合の相対値）

3.2 経営におけるソフトウェア技術への関心の低さ

ソフトウェアは建設等に比し方法の自由度が高く，品質も効率も経営への価値も，選択肢次第で大きく変わる。経営者の多くが IT は経営の要と考えていても，このような点への一般的な理解不足は否めない。保守容易性には優先度が与えられない等も生じがちである。IT サービスの調達は概ね「人月価格」に傾き，国境のバリア低下に伴い人件費の安い国へ仕事が流れ始めた。

3.3 ソフトウェア・プロジェクトの脆さと難しさ

どの調査でもソフトウェア・プロジェクトの成功率は他より低い。品質が拙いと期限やコストの足を引き，逆に進捗やコストの問題は品質に皺寄せを与えやすい。そのような状況の中で，要員への動機付けが一段と重要化した。IT 分野のプロジェクト成功と再現性へ向けて，PM（プロジェクトマネジメント）自体を進歩させるべく，学術としての PM への効果的取組みが一層求められる。

4. 増大するソフトウェア・グリッチのリスク

旅客機に搭載されたソフトウェアは膨張し続け，1 億ソース行を越えた[7]機種もある。危険なグリッチも報じられている [4]。同様に入れる隙のあるモノには悉くマイクロプロセッサが入り込み，スクータ[8]やテレビ[6]など，何にでもソフトウェア・グリッチが生じている。いったんソフトウェアが入ると高機能化に際限がない。例えばカメラでは，複数コマから部分を合成し 1 枚を作るなど，様々な Computational Photography の機能が追求されている[9]。複雑化・大規模化の傾向は他でも同様である。

組込み系ほど膨張が急速でなくとも，エンタープライズ系システムも膨らみがちだ。銀行を例に出すなら，行政のガイド（例[10]）は顧客保護と言いつつ，結果としては情報システムを複雑化させている。不要な商品の積極的な廃止や，無駄を排除する単純化・軽量化が追求され難い。企業の施策の多くもソフトウェアにとっては規模増大につながっている。複雑化は欠陥増加につながり危険を増す。IT 業務賠償責任保険など（例 [11]，日本 [12]等）の職業保険もあるが，賠償対策であり予防策ではない。

5. 予防のために

以上から，ソフトウェア・グリッチによる事故を未然に防ぐために必要な点を整理する。まずシステム開発の立場では，①バグを防ぎ，②障害の影響を最小化すること。保守・運用の立場では，これらに加えて，③潜在化しているバグの発見が必要である。また，④経営によるソフトウェア技術への理解が求められ, IT そのものも, 技術として⑤ソフトウェア開発のコンピュータ化の格段の進歩が必要である。さらに，⑥PM の進歩による IT プロジェクトの安定化も強いニーズである。とくに，①②の前提としては，⑦システム構築の基礎知識の現場への浸透が必要である。

以上はどれも，日頃から随所で追求されている事項に見えるであろう。グリッチが絶対に許されない最近のある大プロジェクト（内容は非公開）で，これらを全て踏まえた上で各種点検を実施した。そこで特に効果を出した策を示したい。「考慮漏れに対するテストケースは作られない」など，2 章に記述した点も検証されたが，予防上の効果が顕著であったのは次の方策である。「1 件のバグが見つかれば，その原因解明と対処だけでは終らせず，原因を作り出した真の要因を求め，その要因が他のどんな原因を作り得るのか（**図 2**）を追究することで，他のバグを未然に発見する。」

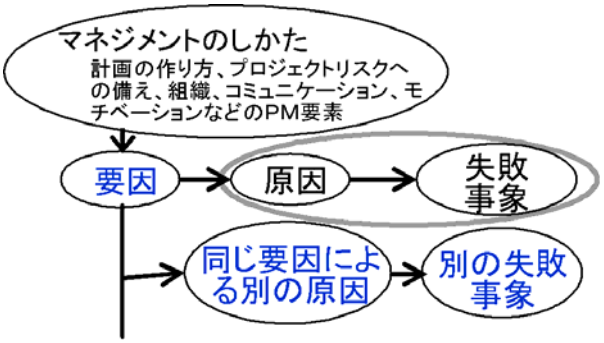

図 2　1 つの欠陥から他の潜在的欠陥を発見する策

実際に，テストによる検出バグは数件しかないのに，最終点検段階ではそれをトリガーにして， 2 桁に及ぶ数の重要バグが，机上で発見された。この点検で抽出されたバグは，それを対象としたテストケースもなく，もし見つからなければ社会に大影響をもたらすグリッチとなる筈であった。それらが未然防止されたわけで，関係者の信念と努力の成果といえる。複雑なプロジェクトでは，バグが 1 つあれば，他の類似バグが複数隠れている可能性が高い点も裏付けられた。

さて，このような方策は，「モダン PM」（1956 年以降世界的に培われてきた PM ノウハウ）の分野では，むしろ普通である。他の潜在的な問題を防止するため，原因を作り出したマネジメント上の要因を追求する。例えば，有名なチャレンジャーの事故の真の要因は，直接原因よりはむしろ，「コミュニケーションとリスクマネジメントにおける仕組みの欠陥」と結論付けられた[14]。このような追究方法は，複雑系に潜入しがちな未発見の問題を抽出し，未然防止に有用であるから，大規模システムではとくに活用すべき方策と考えられる。

6. まとめ

人力でのソフトウェア開発に，ミスは避けられない。しかしソフトウェアは複雑化・大規模化するのが宿命のようであるから，グリッチのリスクは高まる。ソフトウェア・グリッチの予防策は，一般的には範囲が広い上に，空振りにおわる可能性もあるため，計画的な活動が躊躇されるかもしれない。しかし，ソフトウェア技術に急速な進展がない限りは，グリッチの可能性は高まり続ける。

そのような状況の克服に役立つべく，大プロジェクトで実際に一番役立った予防法を紹介した。ここでの結論を言い換えるなら，モダン PM で培われたノウハウやそれらの原理を，ソフトウェアの問題予防に積極的に応用し，成果を得たものである。

参考文献

[1] T. Huckle, “Collection of Software Bugs,” http://www5.informatik.tu-muenchen.de/~huckle/bugse.html, 2006.

[2] W. Rivers. Pitt, ”Worse Than 2000: Tuesday's Electoral Disaster,” http://truthout.org/docs_04/printer_110804A.shtml, 2004.

[3] Julia Scheeres, “Teched-Out Cars Bug Drivers,” http://www.wired.com/news/autotech/0,63846-1.html, 2006.

[4] 渡辺弘美,”最近のシステム障害事例と情報システム保険”, http://www.jpsa.or.jp/info/06/20060912_us_it_softmarket.pdf, 2006.

[5] USA Today, “Airports warned over screening problem,” http://www.usatoday.com/news/nation/2006-03-31-airport-screening_x.htm, 2006.

[6] From Robert Silva, “Software Glitch Revealed in Sony LCD TVs,” http://hometheater.about.com/b/a/244754.htm, 2006.

[7] Patrice Chalin, “Software Requirements and Specification,” Concordia University, http://www.cs.concordia.ca/~dsrg/soen342/Introduction_Req.ppt, 2006.

[8] Oliver Bimber, “Computational Photography – The Next Big Step,” *IEEE Computer* Aug.2006, pp28-65.

[9] CBC News, “Segway recalls all scooters due to software glitch,” http://origin.www.cbc.ca/technology/story/2006/09/14/segway-recall.html, 2006.

[10]木村隆, “06 検査事務年度検査基本方針および検査基本計画-利用者保護に対する取組の検証に重点”, 金融財政事情 2006 年 9 月 18 日号, ㈱きんざい.

[11]スイス再保険会社, “新興市場における保険”, Sigma2005 Vol.5, http://www.swissre.com/INTERNET/pwsfilpr.nsf/vwFilebyIDKEYLu/MPDL-6MFH38/$FILE/Sigma_5_2005_jp.pdf, 2005.

[12]IT 保険ドットコム, http://www.it-hoken.com/nditgbs.html, 2005.

[13]D. Siewiorek and P.Narasimhan, “Fault-tolerant Architectures for Space and Avionics Applications,” Carnegie Mellon University, http://ti.arc.nasa.gov/projects/ishem/Papers/Siewiorek_Fault_Tol.pdf, 2005.

[14]NASA, “Report of the PRESIDENTIAL COMMISSION on the Space Shuttle Challenger Accident,” http://history.nasa.gov/rogersrep/v1ch2.htm, 1986-2006.