「ただいま」――再現テストの願い事

――国際ソフトウェアテスト資格認定委員会への提案

（中国）　曹　海泉　2007/10/01

１．再現テストの定義

再現テスト(reproduction testing)とは、コンポーネントやシステムに発生した欠陥を、テスト技法及びテスト経験によって、意図的に再現させるテストのことです。（公式的の定義が見つかりませんので、自分の考えで定義します。）

２．筆者の現場体験

筆者の現場体験としては、再現テストは、各種テストの中で、最も困難で、かかる時間も一番長く、テスト担当者への要求も一番高いと思います。

例えば、出たり出なかったりしている欠陥をあげて見ましょう。この詳細は、下記の「添付資料　C言語のstaticグローバル変数の副作用」をご参照ください。この再現テストの担当者が筆者でした。再現テストの苦しさは、いまでも忘れられません。

この経験からみると、ソフトウェアの品質を向上するためには、再現テストの理論・技法に関する研究も非常に重要だと思います。しかし、残念なことに、国際ソフトウェアテスト資格認定委員会（International Software Testing Qualifications Board 以下にISTQBと呼ぶ）から提出された、テスト技術者資格制度Foundation Levelシラバスの中には、再現テスト(reproduction testing)の定義がありませんでした。詳細は、2007/01/10に翻訳した日本語版Ver1.0.1（Version 2005）[註1]を参照してください。

３．論文の目的

この論文は、ISTQBへ提案し、再現テストをテストの種類（テストタイプ、test type）の一部として、シラバス中の2.3 テストの種類（タイプ）に追加することを目的としています。即ち、現在の４つのテスト種類（タイプ）「機能テスト・非機能テスト・構造テスト・変更テスト」に、「再現テスト」を追加して、５つのテスト種類（タイプ）にしたいと思います。

「ただいま」と言えるのは、再現テストの願い事だけでなく、筆者の期待です。

４．再現テストの重要性

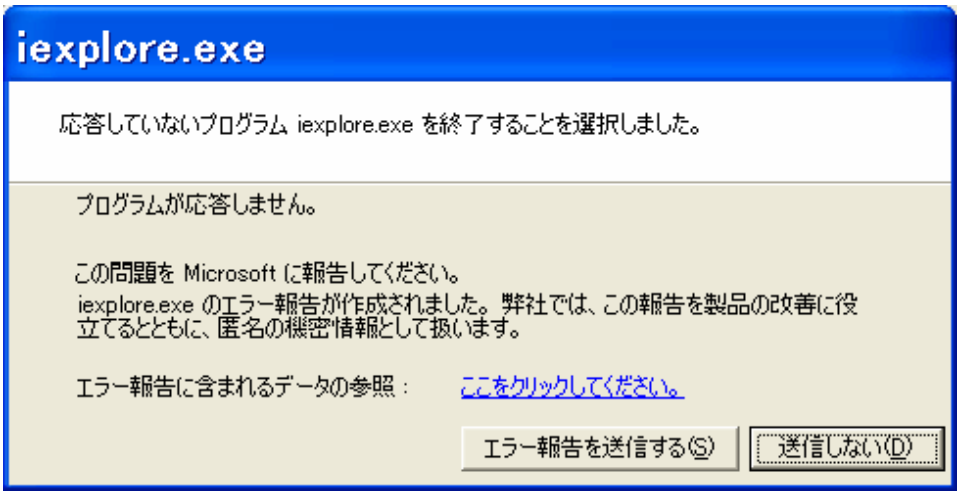
再現テストは、ソフト欠陥を修正する大前提です。再現できないと、欠陥コードも特定できないし、欠陥原因も解明できませんので、対策提案とコード修正の着手も不可能になります。当然、変更テスト（確認テスト、回帰テスト）も行うことができません。

以下、幾つか再現の難しいソフト欠陥を挙げます。これは筆者が遭遇した例です。

① Windows 系向けの応用システムの欠陥

この「読み込み違反」の欠陥は、年に３回発生しました。再現を試みましたが、100 回実行しても起こりませんでした。

② インタネット系向けの応用システムの欠陥



この「プログラムが応答しません」の欠陥は、インタネットを利用している時、月に 1 回の割合で発生しました。システムもダウンしてしまいました。しかしながら、集中的に５０回実行しても再現しませんでした。

③ 組込み系向けの応用システムの欠陥

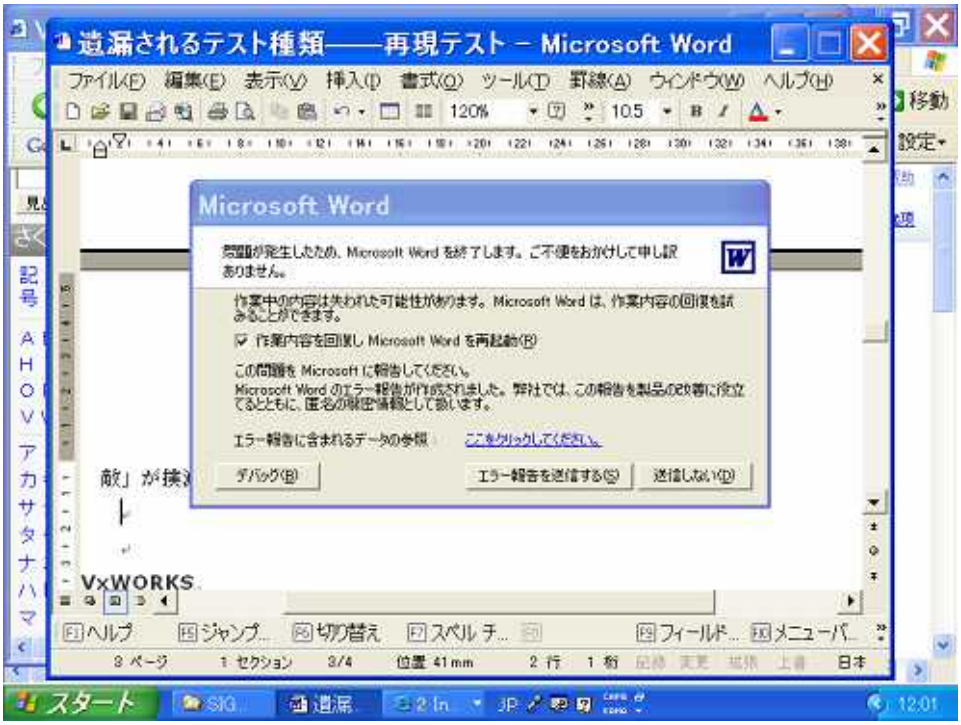
リアルタイム応用システムの「天敵」は、システム連続稼動中に、たまに突然止まって、何も入力を受け付けなくなって、ロックされた状態です。欠陥の再現性はとても薄いです。

筆者が、長年、ロボットのような機械のソフトの開発・保守・テストを担当しました。このシステムはマルチタスク・リアルタイムの原理により開発されました。複雑なため、コードのステップ数が 100 万ぐらいになりました。最初は、ＯＳは iRMX(*1)、言語はＣを採用しましたが、何年か後、信頼性を上げるため、ＯＳは VxWorks(*2)、設計はＵＭＬツール、言語はＣ++に変わりました。開発環境は当時の最先端のものになりました。

しかし、「天敵」は、消滅しませんでした。変更前に比べてほぼ同じ確率でまた現れていました。その後、心血を注いだ再現テストによって、100%再現できた「天敵」は撲滅できましたが、再現の難しい「天敵」はまだ残っています。

④　Microsoft 会社の応用システムの欠陥

この論文を作成している最中に、Word の致命的な欠陥が発生し、論文の一部分が失われました。世界トップレベルの Microsoft 社も、お客様の筆者に迷惑をかけたということです。筆者は、Microsoft 社にこの欠陥を送信しましたが、いつ解決できるかという返事を待っているところです。



つまり、再現の難しいソフト欠陥は、ソフトウェアの分野に広く存在しています。再現難のソフト欠陥を修正するため、再現テストを行わなければなりません。そのため、ソフトウェアの品質を向上するため、再現テストの理論・技法に関する研究も非常に重要だと思っています。

＊1　iRMX とは、Intel 社が x86CPU 用に開発した産業用のリアルタイム OS。25 年以上産業現場で実績のある、信頼性と性能が保証されたリアルタイム OS です。

＊2 VxWORKS とは、Wind River 社の組み込み機器向けリアルタイム OS。 高い信頼性や安全性が要求される航空宇宙･防衛分野で広く採用されており、近年では、一般の産業機器や、情報家電の組み込みコンピュータなどにも幅広く用いられている。

参考文献

[註 1]『テスト技術者資格制度 Foundation Level シラバス日本語版 Ver1.0.1（Version 2005）』

International Software Testing Qualifications Board（著）

Japan Software Testing Qualifications Board（翻訳），東京，2007.

その他

筆者のメールアドレス：quanhaicao@yahoo.co.jp

添付資料

C言語の static グローバル変数の副作用

――現場からの再現テストの報告書

実際に発生した再現テストを例として、再現テストの重要性と複雑性を示します。システム種類は、ロボットのような組込み系です。

１．欠陥の現象

機械が稼動中で突然止まって動かなくなりました。何も入力が受け付けられなくなり、クラッシュの状態になってしまいました。電源をＯＦＦして再度立ち上げた後、何日も正常に動くようになりました。同じ現象は 2 ヶ月前に一度発生しましたが、今度は 2 回目でした。

発生日：XXXX　　発生場所：XXXX 株式会社　　発生バージョン：Ver x.xx(xxx)

発生マシン番号：XXXX

２．欠陥の処置

欠陥番号：XXXX　　欠陥ランク：A（厳重度最高）　　対応ランク：A（緊急度最高）

対応納期：３日以内（XXXX 年 XX 月 XX 日まで)、お客様への説明書も添付

３．再現テスト

最初の４日間は、再現しませんでしたが、5 日目に、ようやく欠陥を 100%再現できました。

再現テストの経過

１日目――欠陥発生の環境の設定（機能テストに相当）

欠陥発生のマシン番号により、この機械のハード配置とソフト設定を調べ、できるだけ欠陥発生の環境と同じように設定しました。そして、欠陥発生のバージョンをインストールし、24 時間連続稼動しましたが、欠陥は再現できませんでした。

２日目――欠陥発生時の動作停止位置とデータの取得（仕様ベースのテストに相当）

このマシン運転は順番にA動作、B動作、C動作からなります。お客様へのお問い合わせの結果、停止位置は、B動作完了後C動作開始の箇所でした。生産データはお客様の機密情報のため、通常は入手しづらいですが、特別に協力していただき、データを入手できました。このデータをそのまま使って、１６時間連続稼動しましたが、欠陥は再現できませんでした。

３日目――生産データの分析とソースコードの解読（構造ベースのテストに相当）

この生産データの中のあるパーツに関する情報は、一般的なパーツより多いため、B動作完了後C動作開始の箇所で待ち時間が通常より長くなりました。それで、このパーツが実行されているとき、B動作完了後C動作開始の箇所で挙動を追跡しました。同時に、このタイミングに相当するソースコードの範囲を詳しく解読しました。特にポインタの不正アクセスと配列の添字のオーバー使用を重点的に解読しました

が、残念ながら、怪しいコードは見つかりませんでした。欠陥もまだ再現できませんでした。

４日目――タスクログの分析（経験ベースのテストに相当）

上司に要求された納期はすでに過ぎていました。何も進展していなかったので、厳しく叱責されました。そのとき、精神的にも肉体的にも限界に近づくように感じました。「頑張れ、もっと頑張れ！」と自分に言い聞かせました。冷静になり、考えを転換しました。コードの解読からタスクログの分析に切り替えました。タスクログとは、各タスクがＣＰＵに実行される順番の記録です。タスクログを分析すると、タスクの順番の流れが正常しているかどうかが分かります。しかし、タスクログが 1000 行以上もあり、1 日かかっても、意味のある情報は得られませんでした。またがっかりした一日となりました。当然、欠陥もまだ再現できませんでした。

５日目――タスクログの短縮で欠陥再現できた、まったく予想外、嬉しい（探索テストに相当）

疲労満杯！その時、上司が硬い表情で「いったい何時再現できますか？」と聞きました。私も分からなくて、無言のままで答えられませんでした。気持ちもどん底に落ちました。では、休憩しましょう！一人で休憩室で冷たいコーヒーを飲みながら、欠陥再現の方法を考え続けました。タスクログを短縮して 50 行となれば、分析しやすいし、時間も少ないし、いろんなタイミングで実施できるし、何か手がかりが分かるかもしれないと思いました。よ－し、行く、やるぞ！タスクログの行数定義を 50 行に変更しコンパイルし、機械へインストールしました。電源をＯＦＦして再度ＯＮし、機械を立ち上げ、稼動開始。1 分経過、３分経過、5 分経過、運転状況正常。とりあえず、トレイに行く。帰ってきて、さらに 10 分ぐらい経過しました。機械稼動の様子を見ると、止まった！静かに止まった！急ぎで停止位置を確認し、Ｂ動作完了後Ｃ動作開始の箇所でした！しかも、何も入力が受け付けられなく、クラッシュの状態になってしまいました。これは、お客様の欠陥現象と同じでした。欠陥再現された？と思い、わくわくしていました。そして、興奮の状態で何回も繰り返してやってみると、ほぼ、10 分ぐらい経過した時点で機械が同じように静かに止まって、クラッシュの状態になりました。予想外でしたが、ようやく欠陥を 100%再現できました！

４．原因分析

クラッシュの状態でシステムのロックアドレスを取りました。このロックアドレスによって、プログラムのステートメントを特定しました。（特定の過程も複雑の為省略）。即ち、taskLog[logLine] = taskNum; でした。これを見ると、すぐ原因が分かりました。システムは、添え字 logLine が配列 taskLog[ ]の定義範囲を越えている状態でアクセスされてしまったので、破滅的な結果になったのです。しかし、単純に前後コードを分析すると、添え字 logLine オーバーの可能性がまったくありませんでした。

```
if   (logLine + 1 < logMax ) { /* logline+1 が配列の範囲 logMax 以内の場合  */
    logLine = logLine + 1;
    taskLog[logLine] = taskNum;   /*  タスク番号の保存  */
}
else {
     logline = 0;
}
```

しかし、現実はなぜ配列の範囲 logMax を越えたのでしょうか？これは、三つの要素との関連があります。一つ目は、優先順位付のマルチタスクのシステム。二つ目は、リアルタイムＯＳのコントロール。三つ目は、添え字 logLine が static グローバル変数と定義されていたこと。

配列の範囲 logMax＝50 とすると、添え字 logLine＝48 の時点のケースを考えてみましょう。まず、優先権低のＡタスクは、if　(logLine + 1 < logMax)を実行し、48＋１< 50 なので、if 部に入って来ました。でも、logLine = logLine + 1;を実行していない（強調：その時点で添え字 logLine＝48）タイミングで、リアルタイムＯＳは、ＣＰＵの使用時間を優先権高のＢタスクに振り分けました。そのため、優先権低のＡタスクは待ち状態になっています。同時に、優先権高のＢタスクも、if　(logLine + 1 < logMax)を実行し、また logLine＝48 のため、48＋１< 50 により、if 部に入って、logLine = logLine + 1;と　taskLog[logLine] = taskNum;を実行しました。この時点で、logLine＝49 になりました。あいにく、リアルタイムＯＳは、再びＣＰＵの使用時間を、待ち状態の優先権低のＡタスクに返し、元の流れに従って logLine = logLine + 1; を実行しました。static グローバル変数の特性に基づいて、添え字 logLine が各タスクにより共同管理され、使用されました。そのため、logLine = logLine + 1=49 + 1＝50 に変わってしまいました。すると、次に、taskLog[logLine] = taskNum;（即ち、taskLog[50] = taskNum;）へアクセスして、taskLog[50]が配列定義範囲を越えていたので、システムがクラッシュしてしまいました。

つまり、結論としては、Ｃ言語の static グローバル変数を配列の添え字として使用していたことが原因です。そして、システムが破滅されたという現象は、Ｃ言語の static グローバル変数の副作用と言えます。

そのため、マルチタスクのシステムでは、Ｃ言語の static グローバル変数を配列の添え字として使用することは、注意しないといけません。静的考えではなく、多数タスク使用という動的思考です。

５．対策

配列の添え字について、static グローバル変数を使用せず、ローカル変数を採用しました。

６．確認テスト

欠陥を修正したシステムは、お客様のデータで三日間連続稼動し、止まるということはありませんでした。

検証結果：ＯＫ　　検証者：ＸＸＸＸ　　検証日：ＸＸＸＸ　　検証バージョン：ＸＸＸＸ

７．回帰テスト

欠陥を修正したシステムは、標準評価のデータで一日間連続稼動し、異常現象がありませんでした。

検証結果：ＯＫ　　検証者：ＸＸＸＸ　　検証日：ＸＸＸＸ　　検証バージョン：ＸＸＸＸ

その後、何年間か経ちましたが、この欠陥は、一回も発生しませんでした。ソフトの品質は、確実に向上できました。