Chương 7

Thanh tra, chạy thử & xem xét mã nguồn

7.1 Giới thiệu

Trong các chương 3, 4, 5, 6 chúng ta đã giới thiệu nhiều kỹ thuật kiểm thử hộp đen lẫn hộp trắng. Điểm chung của các kỹ thuật này là phải chạy thật phần mềm trên máy tính với môi trường phù hợp để tìm lỗi của phần mềm.

Nhưng trong những thế hệ đầu tiên của máy tính, máy tính còn rất yếu và rất đắt, người lập trình chưa có cơ hội làm việc trực tiếp trên máy tính, họ chỉ viết chương trình trên giấy và đem chồng giấy miêu tả chương trình và dữ liệu cần xử lý đến trung tâm máy tính để đăng ký chạy. Khi nhận được kết quả, nếu thấy không vừa ý, họ sẽ phải tự mình đọc và xem xét mã nguồn trên giấy để tìm lỗi và sửa lỗi.

Hiện nay, không phải người kiểm thử nào cũng đọc mã nguồn, nhưng ý tưởng nghiên cứu mã nguồn vẫn được chấp nhận rộng rãi như là 1 nỗ lực kiểm thử hữu hiệu vì những lý do sau :

- kích thước và độ phức tạp về thuật giải của chương trình.
- kích thước của đội phát triển phần mềm.
- thời gian qui định cho việc phát triển phần mềm.
- nền tảng và văn hóa của đội ngũ lập trình.

Qui trình kiểm thử thủ công (chỉ dùng sức người, không dùng máy tính) được gọi là kiểm thử tĩnh, qui trình này có 1 số tính chất chính :

- Rất hữu hiệu trong việc tìm lỗi nên mỗi project phần mềm nên dùng 1 hay nhiều kỹ thuật này trong việc kiểm thử phần mềm.

- Dùng các kỹ thuật kiểm thử tĩnh trong khoảng thời gian từ lúc phần mềm được viết đến khi phần mềm có thể được kiểm thử bằng máy tính.
- Không có nhiều kết quả toán học đánh giá về các kỹ thuật kiểm thử tĩnh vì chúng chỉ liên quan đến con người.

Kiểm thử thủ công TPPM cũng đã đóng góp 1 phần cho tính tin cậy, công nghiệp của hoạt động kiểm thử thành công TPPM :

- Các lỗi được phát hiện càng sớm càng giúp giảm chi phí sữa lỗi và càng giúp nâng cao xác xuất sữa lỗi đúng đắn.
- Lập trình viên dễ dàng chuẩn bị tinh thần khi các kỹ thuật kiểm thử bằng máy tính bắt đầu.

Có nhiều phương pháp kiểm thử thủ công TPPM, trong đó 2 phương pháp quan trọng nhất là thanh kiểm tra mã nguồn (Code Inspections) và chạy thủ công mã nguồn (Walkthroughs). Hai phương pháp này có nhiều điểm giống nhau :

- Cần 1 nhóm người đọc hay thanh kiểm tra trực quan TPPM.
- Các thành viên phải có chuẩn bị trước, không khí cuộc họp phải là "họp các ý kiến thẳng thắn, chân thành".
- Mục tiêu của cuộc họp là tìm các lỗi chứ không phải tìm biện pháp giải quyết lỗi.
- Chúng là sự cải tiến, tăng cường của 1 phương pháp kiểm thử cũ hơn là phương pháp "desk-checking" mà chúng ta sẽ giới thiệu sau.

Hai phương pháp thanh kiểm tra mả nguồn và chạy thủ công mã nguốn có thể phát hiện được từ 30 tới 70% các lỗi viết mã nguồn và lỗi thiết kế luận lý của TPPM bình thường.

Tuy nhiên 2 phương pháp này không hiệu quả trong việc phát hiện các lỗi thiết kế cấp cao như lỗi do hoạt động phân tích yêu cầu TPPM :

- Các hoạt động con người thường chỉ tìm được các lỗi dễ.
- Do đó 2 phương pháp kiểm thử tĩnh này chỉ bổ trợ thêm cho các kỹ thuật kiểm thử chính qui bằng máy tính.

7.2 Phương pháp thanh kiểm tra mã nguồn

Bao gồm các thủ tục và các kỹ thuật phát hiện lỗi nhờ 1 nhóm người cùng đọc mã nguồn. Các vấn đề bàn cải liên quan đến phương pháp thanh kiểm tra là các thủ tục vận hành, các form kết quả cần tạo ra.

- Một nhóm thanh kiểm tra mã nguồn thường gồm 4 người :
- người điều hành (chủ tịch hội đồng), thường là kỹ sư QC
- lập trình viên TPPM cần kiểm thử.
- Kỹ sư thiết kế TPPM (nếu không phải là lập trình viên TPPM này)
- Chuyên gia kiểm thử

1. Người điều hành :

- nên là người lập trình kinh nghiệm, uy tín.
- không nên là tác giả TPPM cần kiểm thử và không cần biết chi tiết về TPPM cần kiểm thử.

2. Các nhiệm vụ :

- Phân phối các tài liệu cho các thành viên khác trước khi cuộc họp diễn ra. Lập lịch cho buổi họp thanh kiểm tra.
- Điều khiển cuộc họp thanh kiểm tra.
- Ghi nhận các lỗi phát hiện được bởi các thành viên.

Công việc chuẩn bị :

- Thời gian và địa điểm buổi họp : làm sao tránh được các ngắt quảng từ ngoài.
- Thời lượng tối ưu cho mỗi buổi họp là từ 90 tới 120 phút.

- Mỗi thành viên cần chuẩn bị thái độ khách quan, nhẹ nhàng trong buổi họp.

Các tài liệu cần có cho mỗi thành viên (đã phân phát trước khi cuộc họp diễn ra) :

- Mã gnuồn TPPM cần kiểm thử.
- Danh sách các lỗi quá khứ thường gặp (checklist).

Trong suốt cuộc họp, 2 hoạt động chính sẽ xảy ra :

1. Hoạt động 1 :

- Người lập trình sẽ giới thiệu tuần tự từng hàng lệnh cùng luận lý của TPPM cho các thành viên khác nghe.
- Trong khi thảo luận, các thanh viên khác đưa ra các câu hỏi và theo dõi phần trả lời để xác định có lỗi ở hàng lệnh nào không ? (Tốt nhất là người lập trình, chứ không phải thành viên khác) sẽ phát hiện được nhiều lỗi trong phần giới thiệu mã nguồn này).

2. Hoạt động 2 :

- Các thành viên cùng phân tích TPPM dựa trên danh sách các lỗi lập trình thường gặp trong quá khứ.

Sau cuộc họp thanh kiểm tra mã nguồn :

- Người điều hành sẽ giao cho người lập trình TPPM 1 danh sách chứa các lỗi mà nhóm đã tìm được.
- Nếu số lỗi là nhiều hay nếu lỗi phát hiện đòi hỏi sự hiệu chỉnh lớn, người điều hành sẽ sắp xếp 1 buổi thanh kiểm tra khác sau khi TPPM đã được sửa lỗi.

Chú ý : Các lỗi phát hiện được cũng sẽ được phân tích, phân loại và được dùng để tinh chỉnh lại danh sách các lỗi quá khứ để cải tiến độ hiệu quả cho các lần thanh kiểm tra sau này.

Các hiệu ứng lề tích cực cho từng thành viên :

- Người lập trình thường nhận được các style lập trình tốt mà mình chưa biết, cách thức chọn lựa giải thuật tốt để giải quyết bài toán, các kỹ thuật lập trình tốt,..
- Các thành viên khác cũng vậy.
- Quá trình thanh kiểm tra mã nguồn là 1 cách nhận dạng sớm nhất các vùng code chứa nhiều lỗi, giúp ta tập trung sự chú ý hơn vào các vùng code này trong suốt quá trình kiểm thử dựa trên mày tính sau này.

7.3 Checklist được dùng để thanh tra mã nguồn

Checklist liệt kê các lỗi mà người lập trình thường phạm phải. Đây là kết quả của 1 lịch sử thanh tra mã nguồn bởi nhiều người, và ta có thể bỏ bớt/thêm mới các lỗi nếu thấy cần thiết. Các lỗi mà người lập trình thường phạm phải được phân loại thành các nhóm chính :

1. Các lỗi truy xuất dữ liệu (Data Reference Errors)
2. Các lỗi định nghĩa/khai báo dữ liệu (Data-Declaration Errors)
3. Các lỗi tính toán (Computation Errors)
4. Các lỗi so sánh (Comparison Errors)
5. Các lỗi luồng điều khiển (Control-Flow Errors)
6. Các lỗi giao tiếp (Interface Errors)
7. Các lỗi nhập/xuất (Input/Output Errors)
8. Các lỗi khác (Other Checks)

Các lỗi truy xuất dữ liệu (Data Reference Errors)

1. Dùng biến chưa có giá trị xác định ?
   ```
   int i, count;
   for (i = 0; i < count; i++) {...}
   ```
2. Dùng chỉ số của biến array nằm ngoài phạm vi ?

```
int list[10];
if (list[10] == 0) {...}
```

3. Dùng chỉ số không thuộc kiểu nguyên của biến array ?

```
int list[10];
double idx=3.1416;
if (list[idx] == 0) {...}
```

4. Tham khảo đến dữ liệu không tồn tại (dangling references)?

```
int *pi;
if (*pi == 10) {...}   //pi đang tham khảo đến địa chỉ không hợp lệ - Null
int *pi = new int;
...
delete (pi);
if (*pi = 10) {...}     //pi đang tham khảo đến địa chỉ
                        //mà không còn dùng để chứa số nguyên
```

5. Truy xuất dữ liệu thông qua alias có đảm bảo thuộc tính dữ liệu đúng ?

```
int pi[10];
pi[1] = 25;
char* pc = pi;
if (pc[1] == 25) {...}          //pc[1] khác với pi[1];
```

6. Thuộc tính của field dữ liệu trong record có đúng với nội dung gốc không ?

```
struct {int i; double d;} T_Rec;
T_Rec rec;
read(fdin,&rec, sizeof(T_Rec);
if (rec.i ==10) {...}           //lỗi nếu field d nằm trước i
                                  //trong record gốc trên file
```

7. Cấu trúc kiểu record có tương thích giữa client/server không ?

```
Private Type OSVERSIONINFO
        dwOSVersionInfoSize As Long
```

```
        dwMajorVersion As Long
        dwMinorVersion As Long
        dwBuildNumber As Long
        dwPlatformId As Long
        szCSDVersion As String * 128 ' Maintenance string
    End Type
    Private Declare Function GetVersionEx Lib "kernel32" _
    Alias "GetVersionExA" (lpVersionInformation As
OSVERSIONINFO) As Long
```

8. Dùng chỉ số bị lệch ?

```
int i, pi[10];
for (i = 1; i <= 10; i++) pi [i] = i;
```

9. Class có hiện thực đủ các tác vụ trong interface mà nó hiện thực không ?

10. Class có hiện thực đủ các tác vụ "pure virtual" của class cha mà nó thừa kế không ?

Các lỗi khai báo dữ liệu

1. Tất cả các biến đều được định nghĩa hay khai báo tường minh chưa?

```
int i;
extern double d;
d = i*10;
```

2. Định nghĩa hay khai báo đầy đủ các thuộc tính của biến dữ liệu chưa?

```
static int i = 10;
```

3. Biến array hay biến chuỗi được khởi động đúng chưa ?

```
int pi[10] = {1, 5,7,9} ;
```

4. Kiểu và độ dài từng biến đã được định nghĩa đúng theo yêu cầu chưa ?

```
short IPAddress;
byte Port;
```

5. Giá trị khởi động có tương thích với kiểu biến ?
   short IPAddress = inet_addr("203.7.85.98");
   byte Port = 65535;
6. Có dùng các biến ý nghĩa khác nhau nhưng tên rất giống nhau không?
   int count, counts;

Các lỗi tính toán (Computation Errors)

1. Thực hiện phép toán trên toán hạng không phải là số?
   CString s1, s2;
   int ketqua = s1/s2;
2. Thực hiện phép toán trên các toán hạng có kiểu không tương thích ?
   byte b;
   int i;
   double d;
   b = i * d;
3. Thực hiện phép toán trên các toán hạng có độ dài khác nhau ?
   byte b;
   int i;
   b = i * 500;
4. Gán dữ liệu vào biến có độ dài nhỏ hơn ?
   byte b;
   int i;
   b = i * 500;
5. Kết quả trung gian bị tràn?
   byte i, j, k;
   i = 100; j = 4;
   k = i * j / 5;
6. Phép chia có mẫu bằng 0 ?
   byte i, k;

```
i = 100 / k;
```

7. Mất độ chính xác khi mã hóa/giải mã số thập phân/số nhị phân ?
8. Giá trị biến nằm ngoài phạm vi ngữ nghĩa ?
```
int tuoi = 3450;
tuoi = -80;
```
9. Thứ tự thực hiện các phép toán trong biểu thức mà người lập trình mong muốn có tương thích với thứ tự mà máy thực hiện? Người lập trình hiểu đúng về thứ tự ưu tiên các phép toán chưa ?
```
double x1 = (-b-sqrt(delta)) / 2*a;
```
10. Kết quả phép chia nguyên có chính xác theo yêu cầu không ?
```
int i = 3;
if (i/2*2) == i) {...}
```

Các lỗi so sánh (Comparison Errors)

1. So sánh 2 dữ liệu có kiểu không tương thích ?
```
int ival;
char sval[20];
if (ival == sval) {...}
```
2. So sánh 2 dữ liệu có kiểu không cùng độ dài ?
```
int ival;
char cval;
if (ival == cval) {...}
```
3. Toán tử so sánh đúng ngữ nghĩa mong muốn? Dễ lộn giữa = và ≠, <= và >=, and và or...
4. Có nhầm lẫn giữa biểu thức Bool và biểu thức so sánh ?
```
if (2 < i < 10) {...}
if (2 < i  &&  i < 10) {...}
```
5. Có hiểu rõ thứ tự ưu tiên các phép toán ?

if(a==2 && b==2 || c==3) {...}

6. Cách thức tính biểu thức Bool của chương trình dịch như thế nào ?
   if(y==0 || (x/y > z))

Các lỗi luồng điều khiển (Control-Flow Errors)

1. Thiếu thực hiện 1 số nhánh trong lệnh quyết định theo điều kiện số học ?
   switch (i) {
   case 1: ... //cần hay không cần lệnh break;
   case 2: ...
   case 3: ...
   }
2. Mỗi vòng lặp thực hiện ít nhất 1 lần hay sẽ kết thúc ?
   for (i=x ; i<=z; i++) {...} //nếu x > z ngay từ đầu thì sao ?
   for (i = 1; i <= 10; i--) {...} //có dừng được không ?
3. Biên của vòng lặp có bị lệch ?
   for (i = 0; i <= 10; i++) {...} //hay i < 10 ?
4. Có đủ và đúng cặp token begin/end, {} ?

Các lỗi giao tiếp (Interface Errors)

1. Số lượng tham số cụ thể được truyền có = số tham số hình thức của hàm được gọi ?
2. thứ tự các tham số có đúng không ?
3. thuộc tính của từng tham số thực có tương thích với thuộc tính của tham số hình thức tương ứng của hàm được gọi ?
   char* str = "Nguyen Van A";
   MessageBox (hWnd, str,"Error", MB_OK); //sẽ bị lỗi khi dịch ở chế độ Unicode
4. Đơn vị đo lường của tham số thực giống với tham số hình thức ?
   double d = cos (90);

5. Tham số read-only có bị thay đổi nội dung bởi hàm không ?
6. Định nghĩa biến toàn cục có tương thích giữa các module chức năng không ?

Các lỗi nhập/xuất (Input/Output Errors)

1. Lệnh mở/tạo file có đúng chế độ và định dạng truy xuất file?

```
if ((fdout = open ("tmp0", O_WRONLY| O_CREAT| O_BINARY, S_IREAD| S_IWRITE)) < 0)
        pr_error_exit("Khong the mo file tmp0 de ghi");
if ((fdtmp = open ("tmp2", O_RDWR | O_CREAT | O_BINARY, S_IREAD | S_IWRITE)) < 0)
```

2. Kích thước của buffer có đủ chứa dữ liệu đọc vào không ?

```
char buffin[100];
sl = read(fd, bufin, MAXBIN); //MAXBIN <= 100 ?
```

3. Có mở file trước khi truy xuất không ?
4. Có đóng file lại sau khi dùng không ? Có xử lý điều kiện hết file ?
5. Có xử lý lỗi khi truy xuất file không ?
6. Chuỗi xuất có bị lỗi từ vựng và cú pháp không ?

Các lỗi khác (Other Checks)

1. Có biến nào không được tham khảo trong danh sách tham khảo chéo (cross-reference)?
2. Cái gì được kỳ vọng trong danh sách thuộc tính ?
3. Có các cảnh báo hay thông báo thông tin ?
4. Có kiểm tra tính xác thực của dữ liệu nhập chưa ?
5. Có thiếu hàm chức năng ?

7.4 Phương pháp chạy thủ công mã nguồn

Giống như phương pháp thanh kiểm tra mã nguồn, phương pháp này bao gồm 1 tập các thủ tục và kỹ thuật phát hiện lỗi dành cho 1 nhóm người đọc mã nguồn.

Cần 1 cuộc họp dài từ 1 tới 4 giờ và không được ngắt quảng giữa chừng.

Nhóm chạy thủ công gồm 3 tới 5 người :

- Lập trình viên nhiều kinh nghiệm
- Chuyên gia về ngôn ngữ lập trình được dùng để viết mã nguồn
- Lập trình viên mới
- 1 người mà sẽ bảo trì phần mềm
- 1 người từ project khác, 1 người cùng nhóm với lập trình viên mã nguồn cần kiểm thử.

Các vai trò trong nhóm :

- Chủ tịch điều hành
- Thư ký (người ghi lại các lỗi phát hiện được)
- Người kiểm thử

Thủ tục ban đầu cũng giống như thủ tục ban đầu của phương pháp thanh kiểm tra mã nguồn.

Thủ tục trong cuộc họp :

- Thay vì chỉ đọc chương trình hay dùng danh sách lỗi thường gặp, các thành viên phải biến mình làm CPU để chạy thủ công mã nguồn.
- Người kiểm thử được cung cấp 1 tập các testcase.
- Trong cuộc họp, người kiểm thử sẽ thực thi từng testcase thủ công. Trạng thái chương trình (nội dung các biến) sẽ được ghi và theo dõi trên giấy hay trên bảng.

Lưu ý :

- Các testcase thường ở mức đơn giản để phục vụ như là phương tiện để bắt đầu và gợi câu hỏi người lập trình về logic thuật giải cũng như những giả định của anh ta.
- Thái độ của từng người tham dự là quan trọng.
- Các chú thích nên hướng đến chương trình thay vì đến người lập trình.
- Chạy thử thủ công nên có qui trình theo sau tương tự như đã diễn tả cho phương pháp thanh kiểm tra.
- Phương pháp chạy thủ công mã nguồn cũng tạo được các hệu ứng lề y như phương pháp thanh kiểm tra.

7.5 Phương pháp kiểm tra thủ công (Desk-checking)

Phương pháp kiểm tra thủ công có thể được xem như là phương pháp thanh kiểm tra hay phương pháp chạy thủ công mà chỉ có 1 người tham gia thực hiện : người này sẽ tự đọc mã nguồn, tự kiểm tra theo danh sách lỗi thường gặp hay chạy thủ công và theo dõi nội dung các biến dữ liệu.

Đối với hầu hết mọi người, phương pháp này không được công nghiệp cho lắm :

- Đây là 1 qui trình hoàn toàn không "undisciplined".
- Ta thường không thể kiểm thử hiệu quả chương trình do mình viết vì chủ quan, thiên vị....$\Rightarrow$ nên được thực hiện bởi người khác, chứ không phải là tác giả của phần mềm.

Kém hiệu quả hơn nhiều so với 2 phương pháp trước. Hiệu ứng synergistic của 2 phương pháp trước.

7.6 Phương pháp so sánh với phần mềm tương tự (Peer Ratings)

Phương pháp này không kiểm thử trực tiếp phần mềm, mục tiêu của nó không phải để tìm lỗi của phần mềm.

Đây là kỹ thuật đánh giá, so sánh các tính chất của các chương trình tương tự như chất lượng tổng thể, độ bảo trì, độ mở rộng, độ sự dụng, độ trong sáng...

Mục đích của phương pháp này là cung cấp cho người lập trình 1 sự tự đánh giá.

7.7 Kết chương

Chương này đã trình bày lý do vì sao chúng ta cần kiểm thử TPPM một cách tĩnh, nghĩa là không cần dùng máy tính chạy trực tiếp TPPM mà chỉ khảo sát xem xét TPPM thủ công thông qua mắt người.

Chúng ta đã trình bày các phương pháp kiểm thử tĩnh TPPM như Desk Checking, thanh kiểm tra, chạy thủ công, peer rating. Ứng với mỗi phương pháp, chúng ta đã trình bày các tính chất cơ bản của phương pháp đó, nguồn nhân lực cần thiết và qui trình thực hiện kiểm thử.