Chương 5

Kiểm tra chương trình

(Buổi 10)

Nội dung

- Kiểm tra mã nguồn (*code inspection*)
- Chạy chương trình bằng thủ công (*walkthrough*)
- *Desk checking*

Kiểm tra

- **Kiểm tra** (*inspection*) là một trong những hoạt động thực tiễn phổ biến nhất trong các dự án phần mềm. Mục đích của việc kiểm tra là để đạt được sự đồng thuận của tất cả các người thanh tra (*inspector*) về kết quả công việc và sử dụng nó trong dự án.
- Kết quả công việc thường được kiểm tra bao gồm:
  - các đặc tả yêu cầu của phần mềm (*software requirement specification*)
  - các kế hoạch kiểm tra (*test plan*).

Kiểm tra

- Trong một cuộc kiểm tra, một nhóm người họp lại để xem xét kết quả công việc. Một người điều hành (*moderator*) sẽ điều khiển cuộc họp.
- Mỗi người thanh tra (*inspector*) chuẩn bị cho cuộc họp bằng cách đọc kết quả công việc và ghi nhận các lỗi (*defect*).
- Mục đích của việc kiểm tra là xác định các lỗi.

Kiểm tra

- Trong một cuộc kiểm tra, một lỗi là một phần nào đó của kết quả công việc mà người thanh tra không đồng ý.
  - Ví dụ, nếu nhóm người đang kiểm tra các đặc tả yêu cầu phần mềm, mỗi lỗi là một đoạn văn bản trong tài liệu mà thanh tra không đồng ý.

Kiểm tra mã nguồn

- **Kiểm tra mã nguồn** (***code inspection***) **là một loại kiểm tra tĩnh (*static testing*) bằng cách xem xét mã nguồn để phát hiện các lỗi và tránh các lỗi này trong giai đoạn sau.**
- **Mục tiêu chính của kiểm tra mã nguồn là phát hiện các lỗi và có thể cải tiến quá trình nếu có.**
- **Một báo cáo kiểm tra liệt kê các kết quả phát hiện, trong đó bao gồm các số liệu có thể được sử dụng để hỗ trợ quá trình cải tiến cũng như sửa các lỗi trong tài liệu được xem xét.**

Kiểm tra mã nguồn

- Trước khi họp, các thành viên cần phải đọc các tài liệu để đảm bảo tính nhất quán.
- Người điều hành cuộc họp không được là tác giả của mã nguồn.
- Quá trình kiểm tra dựa vào các quy tắc và danh mục kiểm tra, các tiêu chí.
- Sau cuộc họp, cần có một quá trình theo dõi chính thức để đảm bảo các hành động khắc phục được hoàn thành một cách kịp thời.

Kiểm tra mã nguồn

- **Các thành viên của cuộc họp**
  - **Người điều hành (*moderator*): kỹ sữ kiêm tra chất lượng**
  - **Tác giả mã nguồn (*autor*): người lập trình (*programmer*)**
  - **Người thiết kế chương trình (*program's designer*), nếu không là tác giả mã nguồn**
  - **Người thanh tra (*inspector*): chuyên gia kiểm thử (*test specialist*)**

Kiểm tra mã nguồn

- **Người điều hành**
  - Nên là người lập trình có kinh nghiệm.
  - Không là tác giả của mã nguồn và không cần biết chi tiết của mã nguồn.
  - Nhiệm vụ:
    - Phân phát các tài liệu cho các thành viên trước khi họp và lập lịch cho cuộc họp.
    - Điều khiển cuộc họp.
    - Ghi chéo các lỗi được phát hiện.
    - Bảo đảm các lỗi này sẽ được khắc phục sau đó.

Kiểm tra mã nguồn

❖ Chuẩn bị cuộc họp

- Thời gian và địa điểm: tránh các yếu tố bên ngoài tác động đến cuộc họp.
- Thời lượng tối ưu của cuộc họp: từ 90 đến 120 phút.
- Các tài liệu:
  - Các chương trình
  - Danh mục (*checklist*) các lỗi cần kiểm tra

Kiểm tra mã nguồn

- **Trong cuộc họp**
  - **Hoạt động 1**
    - Người lập trình trình bày từng hàng lệnh để cho thấy luận lý của chương trình.
    - Trong khi thảo luận, các thành viên khác nêu ra các câu hỏi và theo dõi câu trả lời để xác định có lỗi hay không.
  - **Hoạt động 2**
    - Chương trình sẽ được phân tích dựa vào danh sách các lỗi lập trình thường gặp trong quá khứ.

Kiểm tra mã nguồn

❖ Sau cuộc họp

- Người lập trình nhận được danh sách các lỗi được phát hiện.
- Nếu có nhiều lỗi hoặc nếu có lỗi cần có sự chỉnh sửa lớn, người điều hành sẽ sắp xếp một buổi họp để kiểm tra lại chương trình sau khi các lỗi đã được sửa.
- Lưu ý: Danh sách các lỗi cũng được phân tích, phân loại và được dùng để hiệu chỉnh lại danh mục các lỗi cần kiểm tra để sử dụng cho các lần kiểm tra sau này.

Kiểm tra mã nguồn

❖ Các hiệu ứng lề có ích

- Người lập trình thường nhận phản hồi về phong cách lập trình, chọn các giải thuật và các kỹ thuật lập trình.
- Các thành viên khác biết được các lỗi sai của người lập trình và phong cách lập trình.
- Quá trình kiểm tra mã nguồn là một cách để xác định càng sớm các đoạn chương trình có thể xảy ra lỗi, giúp ta tập trung chú ý nhiều hơn vào các đoạn chương trình này trong quá trình kiểm thử dựa vào máy tính (*computer-based testing*).

Kiểm tra mã nguồn

- **Danh mục các lỗi cần kiểm tra**
  - Các lỗi tham chiếu dữ liệu (*Data Reference Error*)
  - Các lỗi khai báo dữ liệu (*Data-Declaration Error*)
  - Các lỗi tính toán (*Computation Error*)
  - Các lỗi so sánh (*Comparison Error*)
  - Các lỗi dòng điều khiển (*Control-Flow Error*)
  - Các lỗi giao tiếp (*Interface Error*)
  - Các lỗi nhập / xuất (*Input / Output Error*)
  - Các kiểm tra khác
  - Xem ebook The art of software testing, pg 27

Kiểm tra mã nguồn

- **Các lỗi tham chiếu dữ liệu (*Data Reference Error*)**
  - Tham chiếu một biến chưa được gán giá trị: phải gán giá trị ban đầu cho biến trước khi sử dụng biến này.

```
int n;
int m = 2 * n;
```

  - Chỉ số của phần tử mảng phải trong phạm vi kích thước của các chiều.

```
int a[10];
a[11] = 1;
```

  - Chỉ số của phần tử mảng phải là một số nguyên.

```
int a[10];
float x = 1;
a[x] = 5;
```

Kiểm tra mã nguồn

- **Các lỗi tham chiếu dữ liệu (*Data Reference Error*)**
  - **Tham chiếu đến một biến hoặc một vùng nhớ thông qua biến con trỏ thì vùng nhớ này phải tồn tại (*dangling reference*).**

```
int m = 2 * n; // chưa khai báo biến n
int *p;
int m = *p;    // chưa cấp vùng nhớ *p
delete p;
int n = *p;    // đã giải phóng vùng nhớ *p
```

Kiểm tra mã nguồn

- **Các lỗi tham chiếu dữ liệu (*Data Reference Error*)**
  - **Một vùng nhớ có nhiều tên biến khác nhau thì giá trị của vùng nhớ này phải tương thích với kiểu dữ liệu của các biến này.**

```
int a[10];
a[0] = 1;
char *b = a; // int không tương thích với char
```

Kiểm tra mã nguồn

- **Các lỗi tham chiếu dữ liệu (*Data Reference Error*)**
  - Biến con trỏ thuộc cấu trúc dữ liệu này tham chiếu đến vùng nhớ thuộc cấu trúc dữ liệu khác.

```
struct sinhvien
{
    int masv;
    string hoten;
};
struct giangvien
{
    string magv;
    string hoten;
};
```

```
sinhvien *p = new sinhvien;
p->masv = 1;
giangvien *q;
q = p; // khác cấu trúc
```

Kiểm tra mã nguồn

- **Các lỗi khai báo dữ liệu (*Data-Declaration Error*)**
  - Tất cả các biến phải được khai báo tường minh để xác định tầm vực của biến này.

```
int n;
float x;
```

  - Các thuộc tính của một biến phải được khai báo đầy đủ.

```
static int n;
```

  - Biến đơn, biến mảng hoặc biến chuỗi phải được gán giá trị ban đầu thích hợp khi khai báo biến này.

```
int n = 1.3;  // gán số thực cho biến nguyên?
int a[5] = {1, 2, 3}; // giá trị của a[3], a[4]?
int n = 123456789012; // số quá lớn
```

Kiểm tra mã nguồn

- **Các lỗi khai báo dữ liệu (*Data-Declaration Error*)**
  - Kiểu và chiều dài của một biến phải được khai báo theo đúng yêu cầu.

```
char hoten[5] = "Nguyen Trung Tien";
```

  - Không nên đặt các tên biến gần giống nhau vì dễ sử dụng nhầm các tên này.

```
string hten, hoten;
```

Kiểm tra mã nguồn

- **Các lỗi tính toán (*Computation Error*)**
  - Các toán tử phải được thực hiện trên các giá trị thích hợp.

```
int n = 1 + "a";
```

  - Gán giá trị của một biểu thức có chiều dài lớn hơn cho một biến có chiều dài nhỏ hơn.

```
int n = 12345678.0 * 1000;
```

  - Quá trình tính toán một biểu thức bị tràn số.

```
int n = 12345678 * 1000;
```

  - Chia cho số 0.

```
float x, y;
cin >> x;
y = 100 / x;
```

Kiểm tra mã nguồn

- **Các lỗi tính toán (*Computation Error*)**
  - Giá trị của một biến nằm ngoài miền trị hợp lệ.

```
int tuoi = 2000;
```

  - Nắm vững thứ tự thực hiện ưu tiên của các toán tử trong một biểu thức.

```
float x = - b / 2 * a;
```

  - Lưu ý kết quả của phép chia với các số nguyên.

```
i / 2 * 2 == i tùy thuộc vào i là chẵn hoặc lẻ
```

Kiểm tra mã nguồn

- **Các lỗi so sánh (*Comparison Error*)**
  - So sánh các biến có kiểu dữ liệu khác nhau.

```
int n = 10;
string s = "abc";
if (n == s) { ... }
```

  - So sánh các biến có chiều dài khác nhau.

```
int n = 1234;
char c = 'a";
if (n == c) { ... }
```

  - Sử dụng nhầm các toán tử trong biểu thức.

```
<  <=  >  >=  ==  !=  and  or  !
```

Kiểm tra mã nguồn

- **Các lỗi so sánh (*Comparison Error*)**
  - Các toán hạng của toán tử luận lý không là giá trị chân trị.

```
bool found = true;
string s = "abc";
if (found && s) { ... }
```

  - Sử dụng chưa đúng các toán tử so sánh và các toán tử luận lý trong một biểu thức.

```
if (1 < i < 10) { ... }
```

  - Nắm vững thứ tự thực hiện ưu tiên của các toán tử so sánh và các toán tử luận lý trong một biểu thức.

```
if ((a == 1)&& (b == 2) || (c == 3)) { ... }
if (x == 0 && (x / y) > z) { ... }
```

Kiểm tra mã nguồn

- **Các lỗi dòng điều khiển (*Control-Flow Error*)**
  - **Biến điều khiển vượt quá số nhánh.**

```
int i = 3;
switch (i)
{
   case 1: ...
   case 2: ...
}
```

  - **Mọi vòng lặp phải kết thúc.**

```
int i = 10; s = 0;
while (i > 3)
{
   s = s + i;
   i++;
}
```

Kiểm tra mã nguồn

- **Các lỗi dòng điều khiển (*Control-Flow Error*)**
  - Vòng lặp không bao giờ được thực hiện.

```
bool found = true;
while (! found) { ... }
```

  - Số vòng lặp bị sai.

```
for (int i = 0; i <= 10; i++) // 10 vòng lặp
{ ... }
```

  - Số lượng dấu { và dấu } của các khối lệnh phải bằng nhau.

```
for (int i = 0; i <= 10; i++) // 10 vòng lặp
{ ... }
```

Kiểm tra mã nguồn

- **Các lỗi giao tiếp (*Interface Error*)**
  - Các tham số thực và các tham số hình thức phải tương ứng với nhau về số lượng, kiểu dữ liệu, đơn vị đo lường và thứ tự xuất hiện trong danh sách tham số.

```
void Func(int n, string s) { ... }
gọi: Func(masv, hoten);
```

  - Tham số chỉ đọc bị thay đổi giá trị.

```
void Func(const int n)
{ ... n = 5; ... }
```

  - Hiệu ứng lề (*side-effect*) xảy ra giữa các chương trình con thông qua biến toàn cục.

Kiểm tra mã nguồn

- **Các lỗi giao tiếp (*Interface Error*)**
  - **Tham số thực là một biểu thức, tham số hình thức tương ứng là tham số biến (truyền bằng tham khảo).**

```
void Func(int &n)
gọi: Func(i + 3);
```

Kiểm tra mã nguồn

- **Các lỗi nhập / xuất (*Input / Output Error*)**
  - Khi khai báo một tập tin, cấu trúc khai báo của tập tin đúng với cấu trúc lưu trữ của tập tin này.
  - Các biến trong các phát biểu đọc / ghi tập tin tương thích với các giá trị lưu trữ trong tập tin.
  - Phải có đủ dung lượng để lưu trữ tập tin.
  - Phải mở tập tin trước khi sử dụng (đọc / ghi) tập tin này.
  - Phải đóng tập tin sau khi sử dụng tập tin này.
  - Phải phát hiện và xử lý dấu hiệu kết thúc tập tin *EOF* (*End-Of-File*).
  - Phải xử lý các lỗi khi nhập / xuất dữ liệu.
  - Các câu hiển thị hoặc in ra phải đúng chính tả và ngữ pháp.

Kiểm tra mã nguồn

- **Các kiểm tra khác**
  - **Kiểm tra tính hợp lệ của dữ liệu nhập.**
  - **Chương trình bị thiếu một số hàm.**
  - **Kiểm tra một cách cẩn thận các cảnh báo (*warning*) và các thông báo thông tin (*informational message*) của trình biên dịch.**
    - **Các cảnh báo yêu cầu người lập trình phải làm điều gì đó về các nghi ngờ. Các nghi ngờ này cần phải được xem lại.**
    - **Các thông báo thông tin liệt kê các biến chưa được khai báo, tên hàm không tồn tại, …**

Chạy chương trình bằng thủ công

- **Chạy chương trình bằng thủ công** (*walkthrough*) là thực hiện chương trình theo từng lệnh một bằng tay.
- Một cuộc họp liên tục từ 60 đến 120 phút.
- Thành viên bao gồm từ 3 đến 5 người.
  - Người lập trình nhiều kinh nghiệm
  - Chuyên gia về ngôn ngữ lập trình
  - Người lập trình mới có cách nhìn mới, không thiên vị
  - Một người sẽ bảo trì phần mềm
  - Một người từ dự án khác hoặc một người cùng nhóm với người lập trình

Chạy chương trình bằng thủ công

- **Vai trò:**
  - **Người điều hành**
  - **Thư ký ghi nhận tất cả các lỗi được phát hiện**
  - **Người đóng vai trò kiểm thử**

Chạy chương trình bằng thủ công

- **Chuẩn bị cuộc họp**
  - Thời gian và địa điểm: tránh các yếu tố bên ngoài tác động đến cuộc họp.
  - Thời lượng tối ưu của cuộc họp: từ 60 đến 120 phút.
  - Các tài liệu:
    - Các chương trình
    - Danh mục (*checklist*) các lỗi cần kiểm tra

Chạy chương trình bằng thủ công

- **Trong cuộc họp**
  - Các thành viên đóng vai trò là một máy tính.
  - Người đóng vai trò kiểm thử sẽ nhận một danh sách các *test-case* gồm các dữ liệu nhập và kết quả mong muốn dùng cho chương trình hoặc đơn thể cần kiểm thử.
  - Mỗi *test-case* sẽ được thực hiện theo từng lệnh một bằng thủ công. Trạng thái của chương trình (ví dụ giá trị của các biến) được theo dõi trên giấy hoặc trên bảng.

Chạy chương trình bằng thủ công

- **Lưu ý**
  - Bắt đầu bằng các *test-case* đơn giản và đặt câu hỏi cho người lập trình về lý luận lý và các giả định của họ.
  - Thái độ của các thành viên là bình phẩm.
    - Các bình luận về chương trình, không về người lập trình.
  - Các hiệu ứng lề của kỹ thuật kiểm tra mã nguồn cũng được áp dụng cho kỹ thuật chạy chương trình bằng thủ công.

Kiểm tra mã nguồn

❖ Sau cuộc họp

- Người lập trình nhận được danh sách các lỗi được phát hiện.
- Nếu có nhiều lỗi hoặc nếu có lỗi cần có sự chỉnh sửa lớn, người điều hành sẽ sắp xếp một buổi họp để kiểm tra lại chương trình sau khi các lỗi đã được sửa.
- Lưu ý: Danh sách các lỗi cũng được phân tích, phân loại và được dùng để hiệu chỉnh lại danh mục các lỗi cần kiểm tra để sử dụng cho các lần kiểm tra sau này.

Kiểm tra chuyên trách

- **Kiểm tra chuyên trách** (***desk-checking***) **là kỹ thuật kiểm tra mã nguồn hoặc kỹ thuật chạy chương trình bằng thủ công do một người thực hiện.**
  - **Một người đọc và kiểm tra chương trình dựa vào danh mục các lỗi cần kiểm tra, hoặc chạy chương trình bằng thủ công với dữ liệu kiểm thử.**

Kiểm tra chuyên trách

- **Kiểm tra chuyên trách tương đối không hữu ích.**
  - **Là một quá trình không khuôn phép.**
  - **Người kiểm thử thường làm việc không hiệu quả khi kiểm thử các chương trình của họ. Người kiểm thử không nên là tác giả của chương trình cần kiểm thử.**
- **Kiểm tra chuyên trách kém hiệu quả so với các kỹ thuật kiểm tra mã nguồn và chạy chương trình bằng thủ công.**
  - **Không có hiệu ứng lề của hai kỹ thuật này.**