1. Kiểm thử tích hợp

Các chương trước chủ yếu tập trung vào các kỹ thuật kiểm thử các hàm đơn lẻ, còn gọi là kiểm thử đơn vị. Kiểm thử tích hợp tập trung vào việc kiểm thử khi ghép nối các đơn vị này, hay tổng quát hơn là các mô-đun đã được kiểm thử đơn vị. Kiểm thử ở mức này gọi là kiểm thử tích hợp. Kiểm thử tích hợp giúp kiểm tra sự tương thích giữa các mô-đun. Kiểm thử hệ thống và kiểm thử chấp thuận ở phần sau sẽ kiểm tra toàn bộ hệ thống so với đặc tả và yêu cầu của người sử dụng.

Một mô-đun phần mềm (hay còn gọi là một thành phần) là một phần tử tương đối độc lập trong một hệ thống. Khái niệm mô-đun mang tính tương đối. Mô-đun có thể đơn giản là một hàm, một thủ tục, một lớp, hay một tập các phần tử cơ bản này kết hợp với nhau để cung cấp một dịch vụ tích hợp mới. Các mô-đun thường có giao diện rõ ràng để giao tiếp với các mô-đun khác. Một hệ thống là một tập các mô-đun kết nối với nhau theo một cách nhất định để thực hiện một mục đích đã đặt ra.

Trong các dự án lớn có hàng chục hoặc hàng trăm người lập trình, hệ thống thường được chia thành các mô-đun để nhiều nhóm cùng phát triển. Các mô-đun thường được kiểm thử độc lập và kiểm thử ở mức này gọi là kiểm thử đơn vị. Người lập trình thường chịu trách nhiệm thực hiện kiểm thử đơn vị. Các kỹ thuật kiểm thử hộp đen (kiểm thử chức năng) và kiểm thử hộp trắng (kiểm thử cấu trúc) chủ yếu để kiểm thử ở mức này. Sau khi các đơn vị được kiểm thử xong, chúng được ghép với nhau để tạo thành mô-đun lớn hơn, hay hệ thống con, hoặc thành hệ thống phần mềm tùy độ lớn và phức tạp của hệ thống. Việc ghép này không hề đơn giản vì lúc này các lỗi ở mức giao diện giữa các mô-đun có thể xảy ra. Việc kiểm tra lỗi trong quá trình ghép này là kiểm thử tích hợp và kiểm thử hệ thống. Kiểm thử tích hợp thường phải được thực hiện trước và làm trong nội bộ đội phát triển. Khi kiểm thử tích hợp đã ổn định, kiểm thử hệ thống mới được tiến hành để đảm bảo hệ thống hoạt động tốt ở môi trường thật.

Ba lý do chính chúng ta cần kiểm thử tích hợp là:

- Các mô-đun có thể do các nhóm khác nhau làm. Dù đã có thống nhất với nhau từ trước về giao diện của các mô-đun, việc hiểu sai, nhầm lẫn, và chủ quan nhiều khi vẫn xảy ra trên thực tế. Phần sau chúng ta sẽ xem những nguyên nhân có thể gây ra lỗi giao diện này.

- Các mô-đun thường được kiểm thử với các hàm giả trước khi tích hợp, hoặc là với hàm giả (stub), hoặc hàm giả gọi (driver). Các hàm giả (stub) chỉ trả về giá trị kết quả với một số tham số định trước, mô phỏng một vài trường hợp của

hàm thật. Các hàm giả gọi (driver) gọi nhiều mô-đun khác theo các đường đi khác nhau nên nếu hàm giả gọi này không được kiểm thử hết tất cả các đường đi thì khó có thể khẳng định việc thay hàm giả gọi bằng hàm thật chắc chắn không sinh ra lỗi.

- Một số mô-đun bản chất là phức tạp nên dễ có lỗi hơn. Chúng ta cần xác định mô-đun gây ra lỗi nhiều nhất.

Kiểm thử tích hợp kết thúc khi toàn bộ các mô-đun được tích hợp đầy đủ với nhau, các lỗi phát hiện được sửa chữa. Khi hệ thống có nhiều mô-đun thì cũng có nhiều cách ghép chúng lại. Các cách ghép khác nhau sẽ kéo theo các phương pháp kiểm thử tích hợp khác nhau và mỗi trong chúng đều có các ưu nhược điểm. Các phần sau chúng ta sẽ xem xét các yếu tố gây lỗi tích hợp, các phương pháp ghép nối các mô-đun chính, và ưu nhược điểm của việc kiểm thử tích hợp tương ứng.

1.1 Các loại giao diện và lỗi giao diện

Mô-đun hóa là một nguyên lý quan trọng trong thiết kế phần mềm và các mô-đun tương tác với nhau qua các giao diện để thực hiện các yêu cầu chức năng của hệ thống. Một giao diện giữa hai mô-đun cho phép một mô-đun truy cập dịch vụ cung cấp bởi mô-đun kia. Giao diện có cả cơ chế chuyển điều khiển (thực thi) và chuyển dữ liệu giữa các mô-đun. Ba loại giao diện chính chúng ta thường gặp là:

- **Giao diện gọi hàm/thủ tục (procedure call):** một hàm trong một mô-đun gọi một hàm trong một mô-đun khác. Phía gọi sẽ chuyển điều khiển cho mô-đun được gọi. Phía gọi cũng có thể chuyển dữ liệu cho hàm được gọi. Ngược lại hàm được gọi cũng có thể chuyển dữ liệu trả về cho hàm gọi khi nó trả điều khiển về cho hàm gọi.

Trong ví dụ dưới đây, khi thực thi lệnh đầu tiên trong hàm main(), điều khiển sẽ được được chuyển cho hàm print_str() và dữ liệu là xâu ký tự "Hello␣World!" cũng được chuyển cho hàm được gọi (hàm print_str). Giao diện ở đây chính là chữ ký/khai báo hàm void print_str(char*).

Đoạn mã 10.1: Giao diện gọi hàm/thủ tục

```
# include  <stdio .h>
```

```
void print_str( char* str){ printf("% s", str);
}
int main ( void ){
print_str(" Hello ␣World !"); return 0;
}
```

• **Giao diện bộ nhớ chia sẻ (shared memory):** một khối bộ nhớ được chia sẻ giữa hai mô-đun. Khối bộ nhớ này có thể do một trong hai mô-đun cấp phát, hoặc cũng có thể do một mô-đun thứ ba cấp phát. Một mô-đun sẽ ghi dữ liệu lên khối bộ nhớ và mô-đun kia đọc dữ liệu từ khối bộ nhớ.

Trong ví dụ dưới đây hàm main và hàm print_str sử dụng bộ nhớ chia sẻ là biến str để trao đổi dữ liệu giữa các hàm này. Hàm main() ghi dữ liệu và hàm print_str() đọc dữ liệu. Trong trường hợp này, bộ nhớ cho biến str không được cấp phát mà sử dụng hằng ký tự.

Đoạn mã 10.2: Giao diện bộ nhớ chia sẻ

```
# include <stdio .h> char* str;
void print_str ()
{
printf("% s", str);
}


int main ( void )
{
str = " Hello ␣World !"; print_str ();
return 0;
}
```

• **Giao diện truyền thông điệp (message passing):** một mô-đun tạo một thông điệp và gửi thông điệp đó cho một mô-đun khác. Dạng này rất phổ biến trong các hệ thống có nhiều tiến trình, khách-chủ hay các hệ thống trên nền Web, dịch vụ Web.

Trong Đoạn mã 10.3 chương trình tạo một đường ống (bằng hàm pipe()) để tiến trình cha liên lạc với tiến trình con (sinh ra bởi hàm fork()). Sau khi sinh ra tiến trình con, tiến trình cha truyền một xâu ký tự vào đường ống và tiến trình con sử dụng vòng lặp để đọc dữ liệu từ đường ống và in ra màn hình.

Lỗi giao diện là lỗi gắn với các cấu trúc tồn tại bên ngoài môi trường của mô-đun nhưng được mô-đun đó sử dụng [BP84]. Một số lỗi giao diện này được phân loại như sau [PE85]:

1.*Không đủ chức năng:* lỗi này do một mô-đun giả thiết sai về mô-đun kia. Mô-đun cung cấp dịch vụ không hoạt động như mô-đun sử dụng mong đợi - cố tình hoặc ngoài ý muốn của người lập trình mô-đun cung cấp dịch vụ.

2. *Thay đổi tính năng:* một mô-đun được sửa đổi nhưng các mô- đun sử dụng nó không được điều chỉnh theo nên chức năng của hệ thống bị ảnh hưởng.

Đoạn mã 10.3: Giao diện truyền thông điệp

```
// ---------------------------------------------
// Excerpt from " Linux  Programmer's Guide - Chapter~6"
//( C) opyright 1994 -1995 , Scott  Burkett
// ---------------------------------------------
#include <stdio .h> # include <unistd
# include  <sys / types .h>

pid_t childpid ;
char string [] = " Hello , ␣world !\ n", readbuffer [80];

pipe ( fd );
// fd [0] is opened for reading ( input side);
// fd [1] is opened for writing ( output side).

exit( 1 );
}

if ( childpid == 0) {
// child closes  input  side of pipe , & writes
close ( fd [0]);

// Send " string"  through  the  out.  side  of  pipe
write ( fd [1], string , ( strlen ( string )+ 1 ));
}
else {
// parent proc. closes  out. side  of pipe ,\&  reads
close ( fd [1]);
//  Read  in  a  string  from  the  pipe
nbytes = read ( fd [0], readbuffer , sizeof( readbuffer ));
// Print the obtained string
printf(" parent: received ␣string :\% s", readbuffer );
}
return 0;
```

}

3. *Sử dụng giao diện không đúng:* một mô-đun đã sử dụng không đúng giao diện của mô-đun được gọi. Với giao diện hàm việc sử dụng sai này có thể do truyền tham số không đúng thứ tự.

4. *Hiểu giao diện không đầy đủ:* một mô-đun khi thiết kế đã giả thiết một số điều kiện của tham số đầu vào, nhưng phía gọi lại không để ý đến giả thiết này nên đã truyền các tham số nằm ngoài giả thiết. Ví dụ hàm tìm kiếm nhị phân giả sử đầu vào là một mảng được sắp, nhưng phía gọi không sắp xếp mảng này trước khi gọi thì lỗi xảy ra thuộc kiểu này.

5. *Không xử lý lỗi trả về:* một mô-đun được gọi có thể trả về một mã lỗi nhưng mô-đun gọi lại không kiểm tra lỗi, coi nó là kết quả. Hoặc mô-đun được gọi bổ sung thêm mã lỗi trả về nhưng mô-đun gọi chưa kịp biết/sửa.

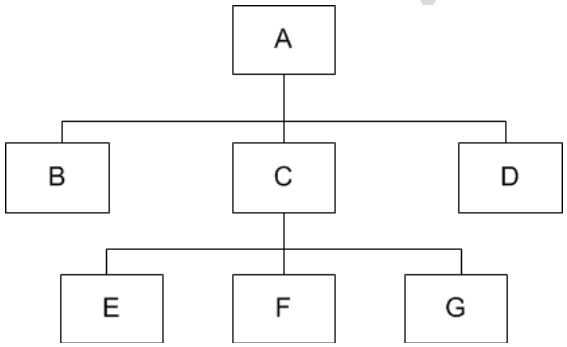
6. *Hiệu ứng phụ với tham số hoặc tài nguyên:* một mô-đun có thể sử dụng tài nguyên không mô tả trong giao diện. Ví dụ một mô-đun sử dụng file tạm tên là "temp", nhưng khi tích hợp một mô-đun khác cũng sử dụng file tạm với tên này sẽ gây lỗi xung đột. Hay ví dụ hàm strdup trong ngôn ngữ C cấp phát bộ nhớ mới và trả về con trỏ đến xâu mới. Nếu bên gọi không giải phóng bộ nhớ thì lỗi dò bộ nhớ sẽ xảy ra.

7. *Các vấn đề phi chức năng:* Các yêu cầu phi chức năng như tốc độ chỉ được nêu ra khi chúng có thể gây vấn đề. Các yêu cầu này ngay cả khi không nêu ra thì chúng ta vẫn ngầm định là chúng phải chạy không quá chậm. Khi tích hợp các vấn đề này mới thường phát sinh.

1.2 Tích hợp dựa trên cấu trúc mô-đun

Một chương trình trong ngôn ngữ C hay Java thường có hàm main, hàm này sẽ gọi các hàm khác trong thân của nó. Các hàm khác này

lỗi gọi tiếp các hàm khác nữa. Rộng hơn là một hệ thống có nhiều mô-đun thì theo cấu trúc phân cấp này chúng tạo thành một cấu trúc hình cây như trong Hình 10.1. Sơ đồ lớp trong nhiều ngôn ngữ hướng đối tượng sẽ có cấu trúc này.



Hình 10.1: Cấu trúc phân cấp mô-đun.

Khi đã có các đơn vị là đỉnh của cây chúng ta có thể lắp dần chúng với nhau và kiểm thử tích hợp trong quá trình lắp. Thứ tự lắp các mô-đun vào cây sẽ dẫn đến các chiến lược kiểm thử tương ứng. Có bốn cách ghép thông dụng là từ trên xuống (top down), dưới lên (bottom up), song song của cả trên xuống và dưới lên gọi là bánh kẹp (sandwich), và một cách đơn giản khác là chỉ kiểm thử sau khi đã ghép hết tất cả các mô-đun (bigbang). Nhược điểm của kiểm thử theo kiểu bigbang là khi có lỗi thì chúng ta rất khó để xác định vị trí của lỗi (khó định vị lỗi). Nếu chúng ta ghép dần dần và kiểm thử luôn thì khi có lỗi xuất hiện, chúng ta tập trung vào các mô-đun vừa ghép với nhau sẽ dễ xác định lỗi hơn. Chú ý ở đây chúng ta giả sử các mô-đun đã được kiểm thử đơn vị xong, nên kiểm thử tích hợp có thể coi là kiểm thử giao diện giữa các mô-đun.