**BÀI TẬP TUẦN 6**

NHÓM 10

|  |  |
| --- | --- |
| MSSV | Tên thành viên |
| 14520466 | Nguyễn Thị Trúc Linh |
| 14520891 | Võ Thanh Thiên Toán |
| 14520653 | Hoàng Đặng Tấn Phát |
| 14520631 | Nguyễn Minh Nhật |
| 14520409 | Trần Đình Khang |

**Exercise 6.1** Mục tiêu đạt được của bạn là gì khi cố găng hiểu chương trình

* Phạm vi vấn đề
* Hiệu quả thực thi
* Mối liên hệ Nhân – Quả (Cause-Effect)
* Mối liên hệ sản phẩm – Môi trường
* Đặc trưng Quyết định – Hỗ trợ

**Exercise 6.2** Tại sao hiểu chương trình là quan trọng?

* Phạm vi vấn đề
* Nắm bắt được kiến thức phạm vi khá quan trọng. Tác động đến vấn đề vùng phạm vi chuyên biệt, cụ thể.
* Hiệu quả thực thi
* Ở mức cao trừu tượng, nhân sự bảo trì cần phải nắm (dự đoán) kết quả chương trình sẽ được phát sinh kết quả gì từ đầu vào được cho mà không cần biết đơn vị chương trình được xây dựng để có kết quả tổng thể và kết quả được cho như thế nào.
* Ở mức thấp, họ cần biết kết quả mỗi đơn vị chương trình sẽ được tạo và thực thi.
* Kiến thức data flow, control flow, và thuật toán có thể thuận tiện hoàn thành thực thi mục tiêu này.

**Exercise 6.3** Giả sử bạn là lập trình viên, bạn được yêu cầu như sau (i) cung cấp tiện ích quản lý thông điệp cho hệ thống vận hành quản lý thông tin (MIS), và (ii) tích hợp hệ thống MIS vào gói văn phòng tự động. Những thông tin về MIS bạn cần làm gì, có tác động đến thay đổi không? Chỉ ra lý do.

1. Phân tích và tổng hợp info để quyết định restructure hay rewrite phân đoạn chương trình cụ thể hay không

2. Xác định tính khả thi của những thay đổi đề xuất và cho thông báo cấp quản lý bất kỳ những vấn đề thấy trước.

**Exercise 6.5:** Liệt kê những loại khác nhau của chiến lược hiểu chương trình, phân biệt giữa chúng

* Top – Down: Là chiến lược hiểu chương trình theo cách tổng quan đến chi tiết. Bằng việc nắm bắt cả chương trình từ những chức năng chính, tiếp theo là các chức năng thứ cấp và các hàm bổ trợ, chiến lược cung cấp cho ta cái nhìn bao quát từ lúc đầu.
* Bottom – Up: Là chiến lược hiểu chương trình theo cách tìm hiểu sâu từ những chức năng phụ, module nhỏ hỗ trợ cho đến các chức năng chính yếu tổng thể của toàn chương trình. Chiến lược cho ta có cái nhìn chi tiết nhất, hiểu rõ nhất khi đọc hiểu đến những vấn đề phức tạp hơn.
* Opportunistic: là chiến lược kết hợp giữa 2 chiến lược trên.

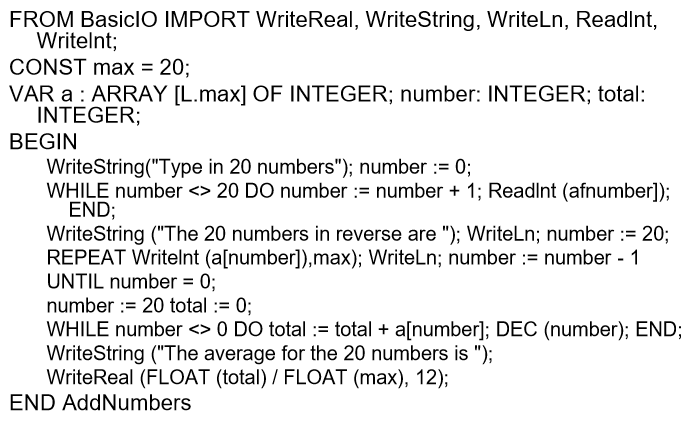
**Exercise 6.6:** Những chiến lược gì bạn đã dùng và trong những hoàn cảnh nào?

Nhóm đã từng dùng chiến lược top-down khi đang xem lại 1 chương trình đồ án của anh chị khóa trước trong môn học, đầu tiên nhóm review lại các chức năng chính gồm những gì, sau đó bắt đầu đi vào từng chức năng để tìm hiểu cach thức thực hiện bên trong nó(code, cách hàm, module thao tác liên quan đến chức năng đó).

**Exercise 6.7** Liệt kê và giải thích các yếu tố chính tác động đến việc hiểu một chương trình

* **Phạm vi kiến thức:** Người lập trình càng có kinh nghiệm trong phạm vi ứng dụng với ngôn ngữ lập trình của hệ thống, càng dễ và nhanh chóng hiểu chương trình và cũng như toàn bộ hệ thống cách hiệu quả.
* **Vấn đề thực thi:** tính nhất quán cùa chương trình: Kiểu/ cách thức đặt tên có đồng nhất và sát nghĩa không, cách chú thích có rõ ràng mach lạc không, kiến trúc của hệ thống có được xây dựng theo cấu trúc nhất định hay không, tính bảo mật,…
* **Tài liệu:** tài liệu đặc tả, hướng dẫn sử dụng có đúng với cách thức hoạt động của hệ thống hay không, cần phải cập nhật thường xuyên khi có thay đổi trong hệ thống để việc nắm bắt và bảo trì được dễ dàng hơn.
* **Tổ chức/thuyết minh chương trình:** Thuyết minh chương được cải tiến có thể cải thiện khả năng hiểu chương trình, thuận tiện biểu thức rõ ràng và chính xác mô hình chương trình và truyền thông của các mô hình này đối với người đọc chương trình, Nhấn mạnh đến luồng kiểm soát, cấu trúc phân cấp chương trình và tính logic và tổng hợp của người lập trình – mục đích gạch dưới cấu trúc và cải thiện tính dễ nhìn của chương trình nguồn qua cách sử dụng ngắt dòng, khoảng trắng, khối và tô bóng,..
* **Công cụ hỗ trợ nắm bắt thông tin:** Có công cụ được sử dụng để tổ chức và thể hiện chương trình nguồn theo cách thực hiện càng rõ ràng càng dễ đọc và như vậy càng dễ hiểu, nhiều công cụ đọc hiểu được thiết kế phục vụ trợ giúp cho người đọc hiểu, tăng tốc độ, qui trình hiểu. Tuy nhiên, đầu ra của công cụ này không cung cấp sự giải thích chức năng của mỗi thành phần.

**Exercise 6.8** Bạn có thể cải thiện khả năng đọc hiểu chương trình Hình 6.10 bằng những cách xử lý nào?



Có thể cải thiện như sau: giữa những từ khóa như FROM, IMPORT nên xuống dòng cho dễ nhìn, sau mỗi dấu “;” nên xuống dòng, trong câu lệnh “WHILE – DO – END” nên xuống dòng ở đoạn “DO”.

**Exercise 6.9** Liệt kê tất cả công cụ bảo trì đã có trong hệ thống của bạn. Có gắng thử 3 trong những công cụ này, với mỗi loại chức năng chính của chúng là gì và làm thể nào nó cải thiện khả năng đọc.

* Bugzilla

Phần mềm này được dùng bởi hàng nghìn công ty phần mềm như là một hệ thống tìm kiếm lỗi, khiếm khuyết. Nó giữ các danh sách lỗi của sản phẩm và cho phép các cá nhân hoặc tổ chức phát triển phần mềm trao đổi và giải quyết vấn đề chung với nhau. Bugzilla cho phép kĩ sư phần mềm quản lí chất lượng, danh sách lỗi và đăng các bảng sửa lỗi. Phần mềm này miễn phí về phần thuật, chúng ta cần phải trả tiền cho những phần mở rộng. Bugzilla là phần mềm bảo mật có khả năng quét và chỉnh sửa lỗi mâu thuẫn của database

* HP Quality Center

Dựa trên nền tảng web, ứng dung này giúp các kĩ sư phần mềm đặt sản phẩm của họ thông qua tiếng chuông bằng cách test chúng. Ứng dụng này có khả năng lên kế hoạch hoặc lịch trình để test, phân tích kết quả và quản lý vấn đề, lỗi. Có 3 phiên bản của ứng dụng này. Phần mở rộng nhất có khả năng đối phó với hàng trăm ứng dụng cùng thời điểm khi mà các quản lý ở nhiều địa điểm khác nhau. Phiên bản ban đầu có sẵn trên phần download của HP website.

* Micro Focus SilkPerformer

Là một phần mềm được sử dụng bởi các công ty, doanh nghiệp lớn để phát triển phần mềm dựa trên kiếm thử mở rộng. Micro Focus cho các nhà phát triển phần mềm có thể đẩy nhanh sự giải quyết vấn đề bằng cách tìm chúng trong vòng đời của quá trình phát triển phần mềm. Còn SilkPerformer hỗ trợ đầy đủ trên nền web 2.0. Phần mềm này chạy test có thể được nhìn thấy thông qua nhiều môi trường mà có nỗ lực xác định nhanh nếu điều kiện của chúng gây ra lỗi.

Kết thúc cuộc trò chuyện

Nhập tin nhắn...

**Exercise 6.10** Tại sao quan trọng đối với người bảo trì thu được hiểu biết tốt chiến lược nắm bắt chương trình khác nhau và vấn đề dựa trên kinh nghiệm

Có một chiến lượt nắm bắt chương trình tốt, và một kinh nghiệm dày dặn là điều cần thiết của người làm công việc bảo trì. Bảo trì phần mềm hay bất cứ các khâu nào trong công nghệ phần mềm đều cần đến kinh nghiệm, bởi thực nghiệm cho thấy, những vấn đề bảo trì ngoài những phát sinh mới thì hầu hết còn lại là những việc bạn đã thực hiện, những lỗi bạn đã từng fixed. Kinh nghiệm dày dặn cho bạn cách tổ chức công việc hợp lí, xử lí nhanh nhất, nó cũng là một liều thuốc tâm lí cho bạn. Đặt lại vấn đề xử lí bảo trì cho hệ thống thư viện chỉ 2 giờ, toàn bộ thời gian này chỉ đủ cho bạn…bấn tít mù lên vì chẳng biết phải bắt đầu làm gì, làm thế nào, và lúc đó hiểu được cái giá của kinh nghiệm. Ít nhất nó cho bạn sự bình tĩnh và cảm giác “sẽ làm được”- thứ rất xa xỉ với những người non kinh nghiệm. Người có kinh nghiệm hiểu được vấn đề ở đây trong khi người non kinh nghiệm bấn rối. Người có kinh nghiệm tự hoạch định được sẽ làm gì, từng bước, trong khi người non kinh nghiệm chẳng biết được mình có làm được hay không? Một điều đáng suy ngẫm cho việc bảo trì. Trải qua và học lấy, để có kinh nghiệm

**Bài tập Part 1:** **Question 60**

Consider the following two pieces of codes and choose the best answer

Code 1:

switch (x)

{

case 1:

cout << "x is 1";

break;

case 2:

cout << "x is 2";

break;

default:

cout << "value of x unknown";

}

Code 2:

if (x==1)

{

cout << "x is 1";

}

else if (x==2)

{

cout << "x is 2";

}

else

{

cout << "value of x unknown";

}

1. Both of the above code fragments have the same behavior
2. Both of the above code fragments produce different effects
3. The first code produces more result than second
4. The second code produces more result than first

**Exercise 7.1** Giải thích khác nhau giữa những loại khác nhau kỹ thuật reverse engineering và cho ví dụ thích hợp.

Reverse Engineering là thao tác phân tích ở mức thấp một hệ thống hoặc phần mềm từ đó hiểu được kiến trúc, dữ liệu, các chức năng và cách thức thực hiện các chức năng đó. Các thao tác cụ thể như sau:   
- Phân tích mã nhị phân của phần mềm   
- Theo dõi các thông tin tương tác (vào/ra) của phần mềm so với thế giới bên ngoài

Reverse Engineering (SE) là một kỹ thuật đối lập với Forward Engineering- một phương pháp tiếp cận phân tích hệ thống phần mềm theo hướng truyền thống. SE thường được thực hiện theo 2 kỹ thuật là Tái cấu trúc và Tái cơ cấu.

Tái cơ cấu có thể hiểu hiểu là cách dịch ngược chương trình theo hướng không thay đổi chức năng và ngữ nghĩa từ hình thức cũ sang một hình thức mới. Tuy nhiên trong hình thức này thì chương trình được hiểu sau khi Tái cơ cấu sẽ không đảm bảo được sự nguyên vẹn, tạo nên những sự thoái hóa và phức tạp. Do đó, cần phải cơ cấu lại một cách hợp lí hơn theo một tiêu chuẩn nhất định.

Tái cấu trúc, đây là quá trình xem xét và thay đổi một hệ thống mục tiêu để thực hiện một sửa đổi mong muốn. Tái cấu trúc bao gồm hai bước. Thứ nhất, kỹ thuật đảo ngược được áp dụng cho hệ thống đích để hiểu nó và thể hiện nó trong một hình thức mới. Thứ hai, kỹ thuật chuyển tiếp được áp dụng, thực hiện và tích hợp bất kỳ yêu cầu mới, do đó dẫn đến một hệ thống mới và nâng cao. Nó cung cấp một cơ chế cho phép nhà duy trì làm việc thông qua mã và hiểu mục đích của nó, sản xuất tài liệu và sửa đổi mã theo cách tăng cường bảo trì của nó, đồng thời hỗ trợ đặc tả lúc trước của một hệ thống từ source code có sẵn.

**Exercise 7.2** Thực hiện khôi phục đặc tả và thiết kế trên tất cả hay các phần của hệ thống phần mềm mà bạn không quen (hệ thống nên có ít nhất 2K dòng code độ lớn).

o Những kỹ thuật bạn dùng nhận diện đặc tả và thiết kế là gì và tại sao?

* Reverse engineering.

o Những hình thức thể hiện bạn xem là phù hợp cho những tác vụ này là gì? Chỉ ra lý do.

o Bài học kinh nghiệm mà bạn đã học được trong các công việc này là gì?

**Exercise 7.3** Một ngân hàng có substantial investment trong hệ thống phần mềm viết bằng Cobol ít nhất 1 triệu dòng code và chạy trên 20 năm. Nó được dùng cơ bản mỗi ngày để thực thi thao tác khác nhau như quản lý tài khoản khách hàng và loans. Sau vài năm cập nhật, cả dự định và không hoạch định – hệ thống trở nên quá đắt tiền để bảo trì. Kết quả là, ngân hàng muốn vài lời khuyên ở các bước tiếp để làm.

o Giả sử bạn được thuê làm việc như nhân viên tư vấn bảo trì.

o Bạn sẽ cho Ngân hàng lời khuyên gì?

o Chỉ ra lý do cho bất kỳ đề nghị mà bạn đã thực hiện

Một hệ thống lớn bao giờ cũng cần rất nhiều thời gian, nhân lực, cũng như gặp nhiều rủi ro trong quá trình bảo trì. Do đó, chắc chắn hệ thống đó phải được bảo trì định kỳ để giảm rủi ro và tài nguyên hao tốn. Nhưng sau một thời gian, dù được bảo trì định kì thì hệ thống cũng sẽ trở nên nặng nề, cồng kềnh và việc bảo trì càng ngày càng mất nhiều thời gian, công sức cũng như tiền bạc, nhiều rủi ro hơn. Thực sự thì em không biết Cobol là ngôn ngữ có ưu nhược điểm thế nào, và chưa tưởng tượng ra được khoảng 1tr dòng code là nhiều hay ít cho một phần mềm có tính năng tương tự khi viết bằng các ngôn ngữ khác mà em biết. Nhưng nếu được đề xuất, em nghĩ có thể thay thế bằng một hệ thống khác, có sức chịu tải hơn, dữ liệu có thể được convert lại để phù hợp với cấu trúc hệ thống mới. Em hoàn toàn tán đồng nếu ai đó bảo ý kiến thế là rất phí phạm, nhưng em nghĩ, hao tốn dành cho việc bảo trì hệ thống cũ cũng ngang ngửa hoặc hơn cho việc đầu tư hệ thống mới.