Tạp chí Tin học và Điều khiển học, T. 13, S. 3 (1997) (94–102)

TÁI SỬ DỤNG PHẦN MỀM TRONG TIN HỌC QUẢN LÝ

PHẠM ĐỨC THÀNH

Abstract. Reuse of products is certainly not a new concept. howerer, with software cost estmated have reached billions VND, even a modest productivity gain through reuse translates into tremendous savings. Reuse can contribute to productivity gains in the following ways: achiving shorter development time, increasing software reliability and ensurring enhanced maintaibility. This paper presents some lessons and explains ths use of domain analysis as a method to select reusable components and the various components of a reusable component.

I - ĐẶT VẤN ĐỀ

Tái sử dụng phần mềm không phải là một khái niệm mới. Do chi phí xây dựng phần mềm đã đạt mức hàng tỷ đồng nên việc tái sử dụng phần mềm ở nước ta cần được đặt ra để tiết kiệm chi phí các nguồn lực trong tin học quản lý.

Tái sử dụng phần mềm được định nghĩa như một ứng dụng của một hoặc nhiều cấu thành phần mềm sang hệ thống mới hoặc để mở rộng hệ thống đang tồn tại.

Tái sử dụng phần mềm được xem xét tổng quát như một cơ hội để cải thiện hiệu quả của phần mềm nhờ rút ngắn thời gian phát triển, nâng cao độ tin cậy và khả năng bảo trì.

Các nhà phát triển phần mềm đã đánh giá ích lợi tiềm tàng của việc tái sử dụng và đang xây dựng các phương thức tái sử dụng một cách hiệu quả nhất trong quá trình phát triển phần mềm.

Các phần sau chúng ta sẽ phân tích chi tiết các khía cạnh của vấn đề đặt ra.

II - TÌNH HÌNH TÁI SỬ DỤNG PHẦN MỀM

1. Mức độ tái sử dụng phần mềm

Trong giai đoạn phát triển ban đầu của công nghệ thông tin (CNTT), máy tính điện tử (MTĐT) được dùng chủ yếu trong công tác tính toán khoa học kỹ

thuật, việc tái sử dụng (TSD) phần mềm được tập trung ở mức mã (xem [1], [2]), đó là hình thức đơn giản nhất. Cấu thành mã có khả năng tái sử dụng gồm có các hàm, các thủ tục hoặc các bó chương trình (package), thí dụ các hàm chuyển đổi ngày tháng, các hàm toán học, các thủ tục nhân, chia ma trận, v.v... Sau khi được xây dựng, phát triển, trắc nghiệm (test), thì cấu thành đó được lưu trong thư viện chương trình mẫu, các lập trình viên có thể sử dụng chúng trong công tác tính toán của mình. Ở đây, tiết kiệm của việc tái sử dụng mã có thể thu được nhờ:

a) Mỗi lần một phần mã được tái sử dụng thì các chi phí cho việc mã hóa (lập trình) sẽ được giảm thiểu.

b) Hơn nữa, tiết kiệm có được còn do không phải trắc nghiệm các cấu thành mã vì chúng đã được kiểm tra từ trước.

Nói chung kích thước mã càng lớn thì hiệu quả TSD càng cao. Tuy nhiên, kích thước đoạn mã (segment) tăng lên dẫn đến một số khó khăn cho người phát triển trong việc xác định sự kết hợp các chức năng đã mô tả. Một đoạn mã lớn thường có nhiều chức năng. Khó khăn đối với người phát triển là phải mô tả được một cách cô đọng các chức năng của nó.

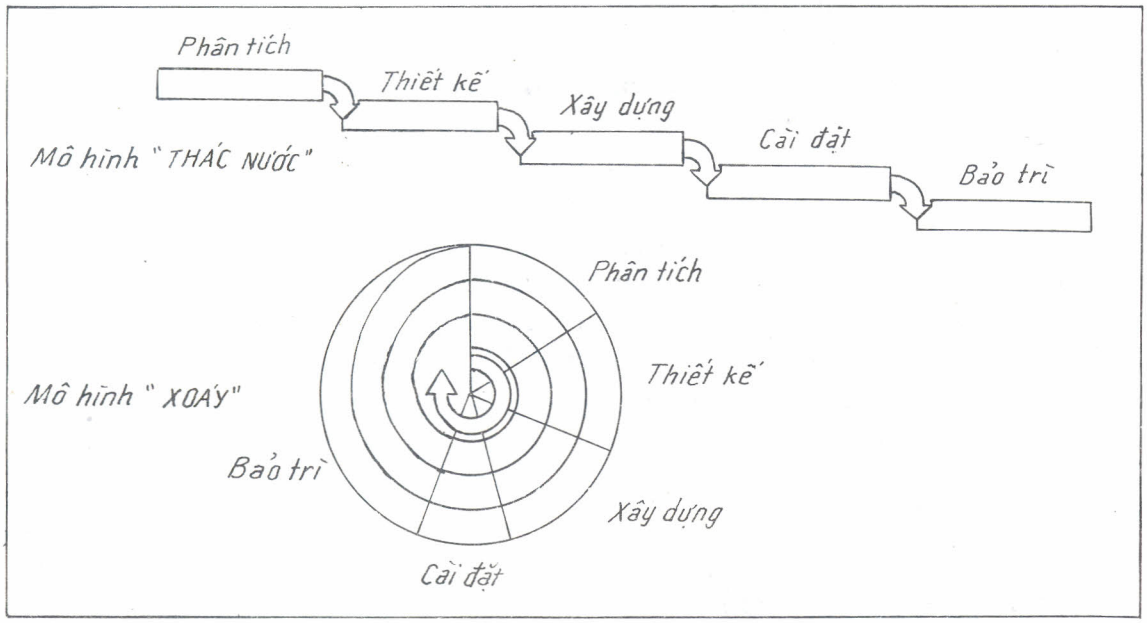
Giai đoạn phát triển tiếp theo của CNTT là khi MTĐT được dùng rộng rãi trong công tác quản lý kinh tế - xã hội: theo số liệu thống kê, có tới 80% các nhà tin học hoạt động trong lĩnh vực này. Hệ thống kinh tế - xã hội là hệ thống lớn và phức tạp. Tin học hóa các hệ thống đó đòi hỏi sự kết hợp chặt chẽ giữa các nhà tin học và các nhà chuyên môn nghiệp vụ. Ở đây vấn đề TSD cần được đặt ra từ đầu. TSD có thể tiến hành ở mọi giai đoạn của chu trình phát triển phần mềm (xem [3], [6]), theo mô hình thác nước và mô hình xoáy kết hợp, chúng bao gồm (xem hình 1):

- Phân tích (Systems analys)
- Thiết kế (Systems design)
- Xây dựng và trắc nghiệm (System build and test)
- Cài đặt và chuyển đổi (System introduction and transition)
- Bảo trì (maintenance of production _status system)

Mô hình thác nước (waterfall model) xuất hiện từ những năm 1970 (xem [4]), chủ yếu dựa trên cơ sở các phương pháp phát triển có cấu trúc. Nó tạo cho ta cái khung làm việc theo kế hoạch phát triển hệ thống từ trên xuống (top_down). Nó đòi hỏi từng giai đoạn phải được kết thúc có kết quả, thường thì đầu ra của giai đoạn trước là đầu vào của giai đoạn sau. Mô hình này dùng để phát triển hệ thống lớn, nghiệp vụ rõ ràng và tương đối ổn định.

Nhưng trong thực tế, các hệ thống, nhất là các hệ thống kinh tế xã hội thường hay biến động. Do đó Barry Boehm đưa ra mô hình xoáy (Spiral model). Tư tưởng chủ đạo ở đây là phát triển hệ thống theo phương pháp lặp. Nó cũng

phân chia thành giai đoạn nhưng với cách tiếp cận để phát triển hệ thống hết sức mạnh bạo và kiên quyết. Khi gặp hệ thống lớn, phức tạp, nghiệp vụ chưa rõ ràng, chưa ổn định thì ta chia nó thành các bài toán nhỏ có tính độc lập tương đối, có nghiệp vụ tương đối rõ ràng; sau đó quay lại dùng mô hình thác nước để phát triển hệ thống. Đây cũng chính là tư tưởng của phương pháp phát triển ứng dụng Designer/2000 và Development/2000 để hỗ trợ. Trong trường hợp này yêu cầu người thiết kế phải có trình độ nhất định, có nhiều kinh nghiệm, biết lường trước những gì sẽ phát sinh, sẽ phải giải quyết; biết tách ra các phần độc lập tương đối, biết xác định các phần có thể dùng chung. Như vậy, về bản chất, sự kết hợp hai mô hình này tạo điều kiện để thực hiện ý tưởng TSD phần mềm trong suốt quá trình phát triển hệ thống.



Hình 1

Tái sử dụng ở mức phân tích và thiết kế cho phép người phát triển bỏ qua phần mã chi tiết mà tập trung sự chú ý vào các chức năng chính tạo nên hệ thống. Thay vì xem xét cách mã hóa, người phát triển có thể tiếp xúc với các mô tả chức năng. Các cấu thành có khả năng tái sử dụng ở mức cao bao gồm các sơ đồ phân rã chức năng, các mô hình dữ liệu, sơ đồ quan hệ thực thể và các sơ đồ dòng dữ liệu.

2. Xác định cấu thành phần mềm có khả năng TSD

Xác định cấu thành phần mềm có khả năng tái sử dụng (Reusable Software Component) là vấn đề quan trọng. Cấu thành phần mềm có khả năng TSD có

thể được phát triển mới hoặc rút ra từ lĩnh vực phần mềm công cộng, phần mềm thương mại, từ các nhà thầu...

Cấu thành phần mềm có khả năng TSD mang các đặc trưng sau:

- Tính mềm dẻo: người phát triển có thể thích ứng hoặc sửa đổi để phù hợp với kiến trúc tổng thể của phần mềm sẽ xây dựng.

- Tính phát triển (expandability): người phát triển có thể sửa đối để mô tả nhu cầu ban đầu.

- Tính chuyển mang: có thể hoạt động trong các môi trường hệ điều hành khác nhau.

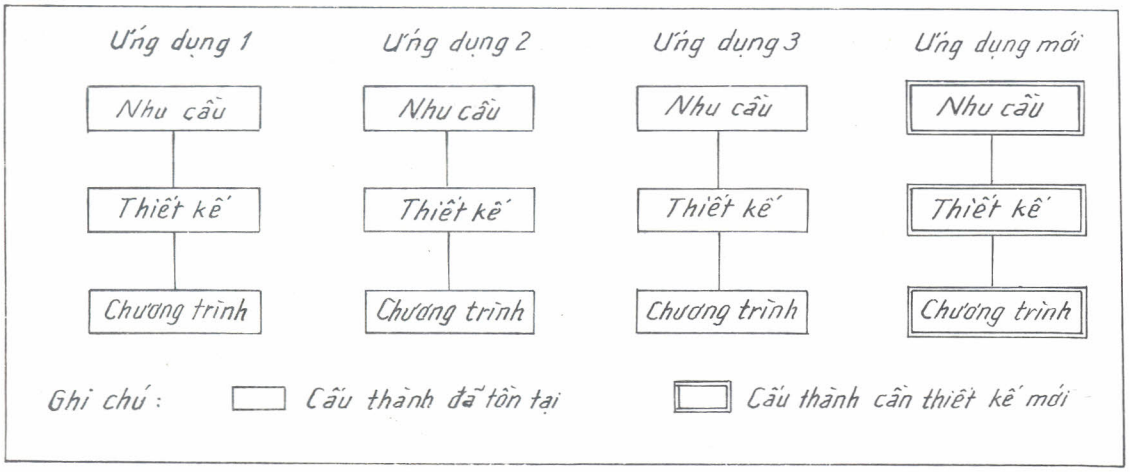
- Tính độc lập về ngôn ngữ: cho phép TSD trong bất kỳ môi trường ngôn ngữ lập trình nào: C++, Visual Basis, oracle form...

Như đã nêu ở trên, vấn đề TSD cần đặt ra ngay từ giai đoạn phân tích, thiết kế. Với một tổ chức như bộ, ngành, xí nghiệp thường có xu hướng tiến tới xây dựng một họ phần mềm ứng dụng mà chúng sẽ chia sẻ các đặc trưng thiết kế. Thí dụ, các ứng dụng như thuế, kế toán kho bạc, quản lý ngân sách, quản lý vốn, quản lý công sản,... của bộ Tài chính đều liên quan đến các đối tượng là mục lục ngân sách, hệ thống chỉ tiêu thu chi, danh bạ tỉnh thành... Nếu các cấu thành thiết kế có thể được TSD thì người phát triển có cơ hội để tập trung cho việc cài đặt các chức năng duy nhất của mình. Chừng nào một thiết kế còn chưa chứa các quyết định chi tiết để cài đặt thì tiềm năng TSD càng lớn. Như một kết luận, cấu thành thiết kế có khả năng TSD sẽ cung cấp lực đòn bẩy cao hơn, mạnh hơn nhiều so với việc đơn thuần TSD các chương trình (mã). Tần số TSD của cấu thành phần mềm phụ thuộc vào chức năng của nó. Nếu chức năng đó là cần thiết cho nhiều miền ứng dụng khác nhau thì tần suất TSD càng cao. Sự phân tích này thường được qui vào cái gọi là phân tích miền (domain analys).

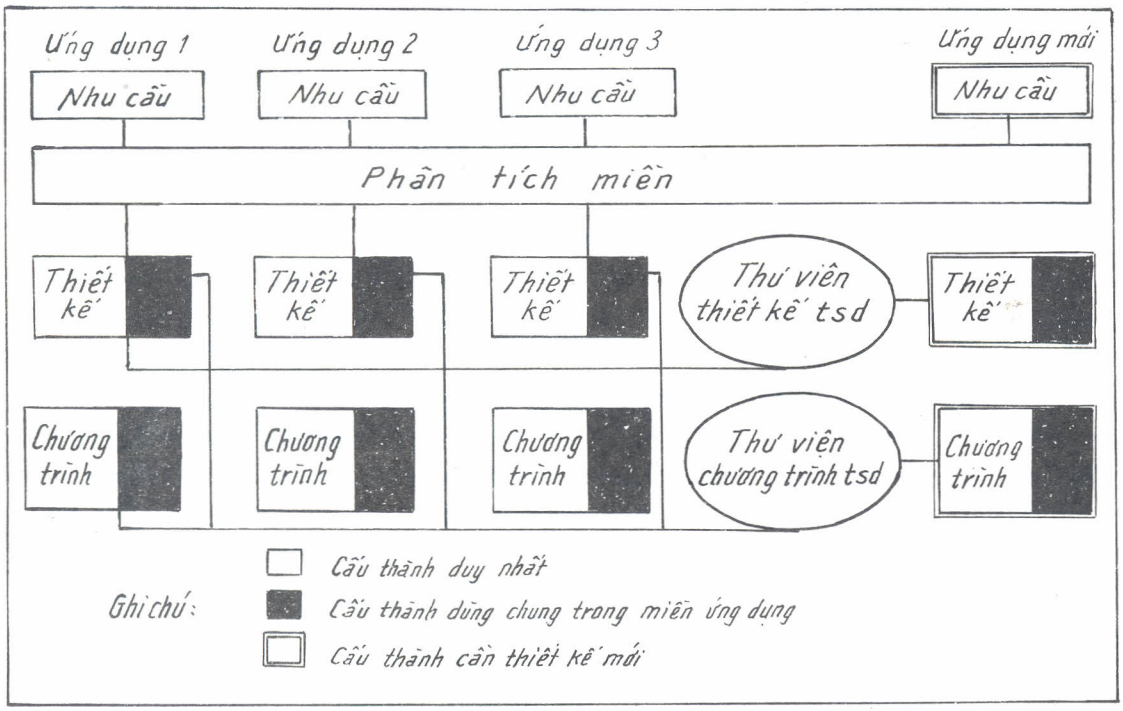
Vai trò của phân tích miền được trình bày trên hình 2 và 3 (xem [5]). Hình 2 chỉ ra cách thức mà các ứng dụng được phát triển thiếu phân tích nhu cầu, thiết kế, và mã hóa một cách độc lập với nhau mặc dù chúng sử dụng chung một số nhu cầu. Thiếu cơ chế để khai thác đặc tính chung sẽ dẫn đến TSD rất ít các cấu thành mã và thiết kế. Ngược lại, hình 3 cho ta thấy khả năng TSD của các cấu thành thiết kế và chương trình (Mã) từ trong phân tích miền ứng dụng.

Chúng ta thấy rằng tiềm năng TSD hoặc thích nghi của cấu thành phần mềm phụ thuộc vào mức độ hiểu biết về miền ứng dụng của người phân tích. Khi phân tích miền trở thành đặc trưng ứng dụng thì người hiểu biết tốt về miền ứng dụng này có thể thực hiện bằng cách áp dụng phương pháp mô hình hóa quá trình hoặc bằng phương pháp phân tích hướng đối tượng. Đã nhiều năm, giới tin học dùng mô hình để phân tích hệ thống, để mô tả hệ thống, để mô phỏng (imitate) hệ thống, để hiểu hệ thống và để trao đổi với giới nghiệp vụ chuyên môn. Bằng cách đó, những người đi trước giúp xác định các chức năng hoặc các quá trình chính mà được sử dụng chung trong các miền ứng dụng đó. Phân tích hướng đối

tượng là phương pháp mới. Một đối tượng là một cấu trúc dữ liệu và một tập chức năng thao tác trên cấu trúc dữ liệu chung đó. Thí dụ, trong miền ứng dụng quản lý kho thì hàng hóa vật tư, phiếu yêu cầu, khách hàng là các đối tượng có khả năng TSD lớn vì mọi hệ thống quản lý kho đều cần dùng các đối tượng đó.



Hình 2. Phát triển các hệ thống tương tự trong miền ứng dụng theo dạng ống khói



Hình 3. Phát triển hệ thống có sử dụng phân tích miền để xác định cấu thành TSD từ các hệ thống đang tồn tại

3. Công cụ xây dựng các cấu thành phần mềm có khả năng TSD

Hiện nay, trên thị trường có một số công cụ cho phép ta TSD thành quả lao động trong tin học quản lý như: ACM-designer, được dựa trên phương pháp MERISE có nguồn gốc từ Pháp, đây là công cụ tương đối cồng kềnh, tại Viện Công nghệ thông tin có cài đặt song chưa được khai thác rộng rãi. Power Builder của hãng Power Soft là công cụ mạnh có đầy đủ chức năng để phát triển các phần mềm ứng dụng. Ưu điểm của Power Builder là gọn nhẹ, thích hợp với máy tính nhỏ và giá bán hạ. Tuy vầy công cụ này cũng chưa được khai thác và sử dụng nhiều ở Việt Nam.

Designer/2000 của hãng Oracle, được giới thiệu nhiều ở Việt Nam từ năm 1995, cho ta công cụ để mô hình hóa quá trình nghiệp vụ (Business modelling), mô hình hóa quá trình phân tích hệ thống (System design), tạo ra cơ sở dữ liệu và các ứng dụng mẫu (Design wizard). Chúng tạo thành chu trình liên hoàn, các kết quả ở mức trên được sử dụng trực tiếp cho mức dưới. Toàn bộ từ điển dữ liệu hệ thống (bao gồm cả sơ đồ) được lưu trữ trong kho chứa (Repository). Designer/2000 cho phép ta cập nhật kho dữ liệu đó thông qua hệ thống hoa tiêu (Object Navigator). Để hoàn thiện ứng dụng, Oracle còn cung cấp công cụ phát triển (Developer/2000).

III - XÂY DỰNG NGÂN HÀNG CẤU THÀNH PHẦN MỀM CÓ KHẢ NĂNG TSD

1. Quan hệ cung cầu đòi hỏi một ngân hàng TSD

Việc TSD các cấu thành chỉ hấp dẫn nếu chi phí tổng thể để áp dụng nhỏ hơn chi phí để tạo ra nó từ đầu. Nếu người sử dụng phải đầu tư nhiều thời gian để tìm kiếm và phát triển các cấu thành đã lưu trữ trong ngân hàng TSD (NHTSD) thì họ sẽ thích tự mình phát triển hơn. Như đã nêu ở trên việc TSD các cấu thành phần mềm trong phạm vi lớn chắc chắn sẽ mang lại tiết kiệm chi phí lớn. Vì vậy điều cấp bách là phải tạo ra NHTSD có khả năng thu hút giá trị của các cấu thành phần mềm có khả năng TSD đã được thiết lập để đáp ứng tốt nhu cầu của người sử dụng.

Để có hiệu quả, cần xây dựng NHTSD có cơ chế hoạt động tốt, đảm bảo giao dịch thường xuyên và vững chắc giữa các nhà cung cấp và người sử dụng (không hẳn là người sử dụng cuối). Người cung cấp là người xác định khả năng TSD của cấu thành phần mềm, phát triển cấu thành phần mềm nhằm đáp ứng yêu cầu TSD đặt ra và làm cho cấu thành phần mềm đó có mặt trong ngân hàng. Nếu Người sử dụng là Người phát triển, thì họ có thể xem xét các cấu thành trong NHTSD và chọn cấu thành mà nó hỗ trợ nhu cầu của họ một cách đầy đủ nhất. Vì vậy, người sử dụng cũng cần có kỹ năng và kinh nghiệm để tìm kiếm

cấu thành, hiểu các đặc trưng của cấu thành phần mềm có khả năng TSD và thích ứng chúng vào nhu cầu của mình. Cơ chế hoạt động của NHTSD càng tốt thì cho hiệu quả càng khích lệ.

2. Động cơ thúc đẩy phát triển một ngân hàng

Việc thiết đặt NHTSD đòi hỏi chi phí trực tiếp đáng kể và chi phí quản lý thực hiện các dịch vụ tập trung, tuyên truyền, cung cấp cái nhìn rộng lớn về ích lợi của TSD phần mềm và khuyến khích chuẩn hóa. Cho dù phần mềm đó có được viết với mục đích TSD trong tương lai hoặc phần mềm đó có được lấy từ đâu thì cũng cần tạo khuôn dạng (đóng gói), thử nghiệm (test) và lập tài liệu (documentation). Do vậy cần có nguồn kinh phí tài trợ nhất định để xây dựng và duy trì hoạt động của nó.

Giá trị của NHTSD phù thuộc vào qui mô của nó và các công cụ sử dụng để lưu trữ, tổ chức, truy cập, và bảo trì các cấu thành phần mềm có khả năng TSD.

NHTSD càng phát huy được ưu việt của mình khi nó hoạt động trong môi trường mạng sở hữu bởi nhiều cơ quan tham gia phát triển phần mềm tương tự, như vậy sẽ mang lại sự tiết kiệm lớn. Họ cũng có thể cùng nhau phát triển các cấu thành phần có khả năng TSD mới nếu như họ không tìm thấy ở nơi nào khác.

Hy vọng rằng giá trị thu được của việc TSD phần mềm đã được thử nghiệm hoàn hảo và có kèm theo tài liệu tốt sẽ đóng vai trò quan trọng hơn trong quá trình phát triển lâu dài. NHTSD có vai trò tích cực trong việc điều hòa quyền lợi giữa người sử dụng và người cung cấp cấu thành phần mềm có khả năng TSD. Người ta khẳng định rằng thiếu chiến lược TSD là một yếu tố hạn chế động cơ thúc đẩy các nhà quản lý dự án và người phát triển sử dụng lại phần mềm. Quá trình TSD sẽ không tự nó tiến triển mà cần được xúc tiến bằng cơ chế khuyến khích nhất định.

Các nhà quản lý cần quan tâm hơn trong việc tiếp cận các vấn đề phi kỹ thuật như quyền sở hữu hợp pháp, bù đắp thu nhập cho các nhà phát triển các cấu thành phần mềm có khả năng TSD và phương pháp phân chia nội bộ giá trị đạt đạt được của việc TSD phần mềm để làm sống lại phần mềm đã phát triển. Các hợp đồng phải thỏa mãn yêu cầu của cả hai phía: người cung cấp và người sử dụng theo chiều hướng khuyến khích TSD.

Các nhà quản lý đầu tư phát triển có quan tâm, khuyến khích đầy đủ thì mới nâng cao khả năng, vai trò của mình trong việc thiết lập môi trường TSD. Thí dụ, muốn phát triển phần mềm có chất lượng cao, cần dũng cảm đưa ra qui chế tiền thưởng đối với một số loại sản phẩm, hoặc người phát triển bù đắp việc khai thác phần mềm từ NHTSD thay cho việc trả công họ tạo ra cấu thành mới từ đầu. Điều này khuyến khích các nhà phát triển sử dụng phương pháp luận của TSD phần mềm.

Người phát triển sẽ được nhận cả hai khoản tiền thưởng, một là khi cấu thành được đưa vào NHTSD, hai là tiền bản quyền tác giả mỗi lần cấu thành được TSD trong ứng dụng mới.

3. Hoạt động của NHTSD

Các hoạt động chính mà ngân hàng có thể thực hiện với danh nghĩa cộng đồng TSD là:

a) Xác định các miền có khả năng TSD

Khi phần mềm lấy từ các miền khác nhau thì cần xác định các cấu thành phần mềm mà chúng có chung các chức năng cơ bản. Điều này có thể thực hiện bằng một quá trình tìm hiểu như là phân tích miền. Mục đích của phân tích miền là xác định tính phổ biến bên trong miền ứng dụng mà tập trung ở lĩnh vực có tiềm năng lớn TSD và nhu cầu phát triển phần mềm lớn trong tương lai. Đó là một quá trình lặp lại đòi hỏi sự kiểm tra, xác minh về quyền lợi chung. Thiếu việc phân tích miền hoàn hảo thì các cấu thành có khả năng TSD không thể nhận biết một cách chính xác.

b) Tìm kiếm các cấu thành có khả năng TSD

Việc tìm kiếm các cấu thành có khả năng TSD là một sự cố gắng không phải là tầm thường. Các nguồn cung cấp lớn như các hệ thống cấp quốc gia, phần mềm trong lĩnh vực công cộng hoặc các phần mềm thương mại phải được xác định. Lợi ích đặc biệt trong việc tìm kiếm cấu thành này là danh tiếng của người cung cấp hoặc hồ sơ ghi chép về quá trình xây dựng cấu thành phần mềm có chất lượng.

c) Xác nhận các cấu thành có khả năng TSD

Cần xác nhận công việc của người phát triển các cấu thành có khả năng TSD. Mỗi công việc cần xác định chất lượng, khả năng bảo trì, trách nhiệm pháp lý. Việc định giá là phần rất quan trọng của quá tình xác nhận. Các cấu thành có khả năng TSD lúc đầu xem như là dự bị, sau được xác định và tiếp tục có giá trị trong suốt phần còn lại của chu kỳ sống. Sự định giá có thể loại bỏ các cấu thành không đáp ứng, xác định sự cần thiết phải thiết kế lại, tiến hành lập hồ sơ, xác lập các tham số quan trọng như: kích thước, tính hiệu quả, tính chuyển mang, tính bảo trì, khả năng TSD và cung cấp một số đánh giá trước về trở ngại và khuyến nghị các giải pháp thay thế khi cần thiết.

d) Tạo NHTSD thân thiện

Hệ thống NHTSD là cần thiết cho người phát triển để họ nhận biết, lấy lại và sử dụng các cấu thành có khả năng TSD. Phần mềm được lựa chọn để nhập vào NHTSD cần được tổng hợp theo một sơ đồ nhất định. Hệ thống thư mục cài đặt cần phải đơn giản, dễ sử dụng và phải cung cấp các mẫu tìm kiếm linh hoạt để người sử dụng tìm được cấu thành tốt nhất. Sau cùng, hệ thống này cần

có khả năng thích ứng với những thay đổi trong trường hợp xuất hiện các cấu thành mới có dạng đặc biệt.

e) Hỗ trợ người phát triển

Một môi trường TSD hiệu quả không thể được duy trì mà thiếu sự tin tưởng của người phát triển. Sự tin tưởng này đạt được bằng nhiều cách. Đáng kể nhất, đó là quá trình xác nhận đảm bảo chất lượng và có chuẩn mực của cấu thành trong NHTSD. Hơn nữa, yêu cầu và ích lợi của người phát triển cần được nghiên cứu định kỳ để đảm bảo tập trung sức mạnh vào các cấu thành có khả năng TSD.

Tóm lại, việc xây dựng NHTSD cho ta một số ưu việt: thứ nhất là thực hiện phân tích miền bao trùm phạm vi nhiều ứng dụng, làm tăng độ chính xác và mức độ thích hợp cho việc phát triển các ứng dụng tương lai. Thứ hai là với vị trí trung tâm của mình, NHTSD có đủ khả năng và quyền lực để thi hành nguyên tắc TSD một cách nghiêm túc. Các thủ tục xác nhận có thể đặt đúng vị trí để đảm bảo thống nhất chất lượng của các cấu thành có khá năng TSD. Điều đó ảnh hưởng trực tiếp đến sự tin tưởng của người sử dụng tạo cơ hội để phục vụ một cộng đồng lớn.

TÀI LIỆU THAM KHẢO

1. Basili V. R., *Software development a paradigm for the future,* Proceeding of the thirteenth annual international computer software and applicationa conference, Orlando, FL, 1989.
2. Romback H. D., *A comprehensive approach towards reuse*, 2nd international conference on software quanityassurance, oslo Norway, May 1990.
3. Mili H., Mili F., Mili A., *Reusing Software: issued and reseach directions*, IEEE Trans. Software Engineering, June 1995, 528-562.
4. Introduction appoaches Data Design, oracle, 1992.
5. Tung Xuan Bui, *A clearinghouse for Software reuse*, IV informatics week, Proceeding, Ho Chi Minh City, 1994.
6. Pham T. Nhu, *The life cycle for software evolution with reusable assests*, Hội nghị khoa học kỷ niệm 20 năm thành lập Viện CNTT, Hà Nội, 5-6/12/1996.

Viện Công nghệ thông tin
Trung tâm KHTN và CNQG

Nhận bài ngày 18-12-1996