Module 16

Software Reuse

Lesson 40

Reuse Approach

Specific Instructional Objectives

At the end of this lesson the student would be able to:

- Explain a scheme by which software reusable components can be satisfactorily classified.
- Search an item from the domain repository.
- Explain how a reuse repository can be maintained.
- Explain what is meant by an application generator.
- Identify the advantages of using an application generator compared to parameterized programs.
- Identify the shortcomings of application generator.
- Identify the steps that can be adopted for achieving organization-level reuse.
- Identify the non-technical factors that inhibit an effective reuse program.

Components classification

Components need to be properly classified in order to develop an effective indexing and storage scheme. Hardware reuse has been very successful. Hardware components are classified using a multilevel hierarchy. At the lowest level, the components are described in several forms: natural language description, logic schema, timing information, etc. The higher the level at which a component is described, the more is the ambiguity. This has motivated the Prieto-Diaz's classification scheme.

Prieto-Diaz's classification scheme: Each component is best described using a number of different characteristics or facets. For example, objects can be classified using the following:

- actions they embody
- objects they manipulate
- data structures used
- systems they are part of, etc.

Prieto-Diaz's faceted classification scheme requires choosing an n-tuple that best fits a component. Faceted classification has advantages over enumerative classification. Strictly enumerative schemes use a predefined hierarchy. Therefore, these forces to search for an item that best fit the component to be classified. This makes it very difficult to search a required component. Though cross-referencing to other items can be included, the resulting network becomes complicated.

Searching

The domain repository may contain thousands of reuse items. A popular search technique that has proved to be very effective is one that provides a web interface to the repository. Using such a web interface, one would search an item using an approximate automated search using key words, and then from these results do a browsing using the links provided to look up related items. The approximate automated search locates products that appear to fulfill some of the specified requirements. The items located through the approximate search serve as a starting point for browsing the repository. These serve as the starting point for browsing the repository. The developer may follow links to other products until a sufficiently good match is found. Browsing is done using the keyword-to-keyword, keyword-to-product, and product-to-product links. These links help to locate additional products and compare their detailed attributes. Finding a satisfactorily item from the repository may require several locations of approximate search followed by browsing. With each iteration, the developer would get a better understanding of the available products and their differences. However, we must remember that the items to be searched may be components, designs, models, requirements, and even knowledge.

Repository maintenance

Repository maintenance involves entering new items, retiring those items which are no more necessary, and modifying the search attributes of items to improve the effectiveness of search. The software industry is always trying to implement something that has not been quite done before. As patterns requirements emerge, new reusable components are identified, which may ultimately become more or less the standards. However, as technology advances, some components which are still reusable, do not fully address the current requirements. On the other hand, restricting reuse to highly mature components, sacrifices one of the creates potential reuse opportunity. Making a product available before it has been thoroughly assessed can be counter productive. Negative experiences tend to dissolve the trust in the entire reuse framework.

Application generator

The problem-oriented languages are known as application generators. Application generators translate specifications into application programs. The specification is usually written using 4GL. The specification might also in a visual form. Application generator can be applied successfully to data processing application, user interface, and compiler development.

Advantages of application generators

Application generators have significant advantages over simple parameterized programs. The biggest of these is that the application generators can express the variant information in an appropriate language rather than being restricted to function parameters, named constants, or tables. The other advantages include fewer errors, easier to maintain, substantially reduced development effort, and the fact that one need not bother about the implementation details.

Shortcomings of application generator.

Application generators are handicapped when it is necessary to support some new concepts or features. Application generators are less successful with the development of applications with close interaction with hardware such as real-time systems.

Re-use at organization level

Achieving organization-level reuse requires adoption of the following steps:

- Assessing a product's potential for reuse
- Refining products for greater reusability
- Entering the product in the reuse repository

Assessing a product's potential for reuse. Assessment of components reuse potential can be obtained from an analysis of a questionnaire circulated among the developers. The questionnaire can be devised to access a component's reusability. The programmers working in similar application domain can be used to answer the questionnaire about the product's reusability. Depending on the answers given by the programmers, either the component be taken up for reuse as it is, it is modified and refined before it is entered into the reuse repository, or it is ignored. A sample questionnaire to assess a component's reusability is the following.

- Is the component's functionality required for implementation of systems in the future?
- How common is the component's function within its domain?
- Would there be a duplication of functions within the domain if the component is taken up?
- Is the component hardware dependent?
- Is the design of the component optimized enough?
- If the component is non-reusable, then can it be decomposed to yield some reusable components?

- Can we parameterize a non-reusable component so that it becomes reusable?

Refining products for greater reusability. For a product to be reusable, it must be relatively easy to adapt it to different contexts. Machine dependency must be abstracted out or localized using data encapsulation techniques. The following refinements may be carried out:

- **Name generalization:** The names should be general, rather than being directly related to a specific application.

- **Operation generalization:** Operations should be added to make the component more general. Also, operations that are too specific to an application can be removed.

- **Exception generalization:** This involves checking each component to see which exceptions it might generate. For a general component, several types of exceptions might have to be handled.

- **Handling portability problems:** Programs typically make some assumption regarding the representation of information in the underlying machine. These assumptions are in general not true for all machines. The programs also often need to call some operating system functionality and these calls may not be same on all machines. Also, programs use some function libraries, which may not be available on all host machines. A portability solution to overcome these problems is shown in fig. 16.1. The portability solution suggests that rather than call the operating system and I/O procedures directly, abstract versions of these should be called by the application program. Also, all platform-related calls should be routed through the portability interface. One problem with this solution is the significant overhead incurred, which makes it inapplicable to many real-time systems and applications requiring very fast response.

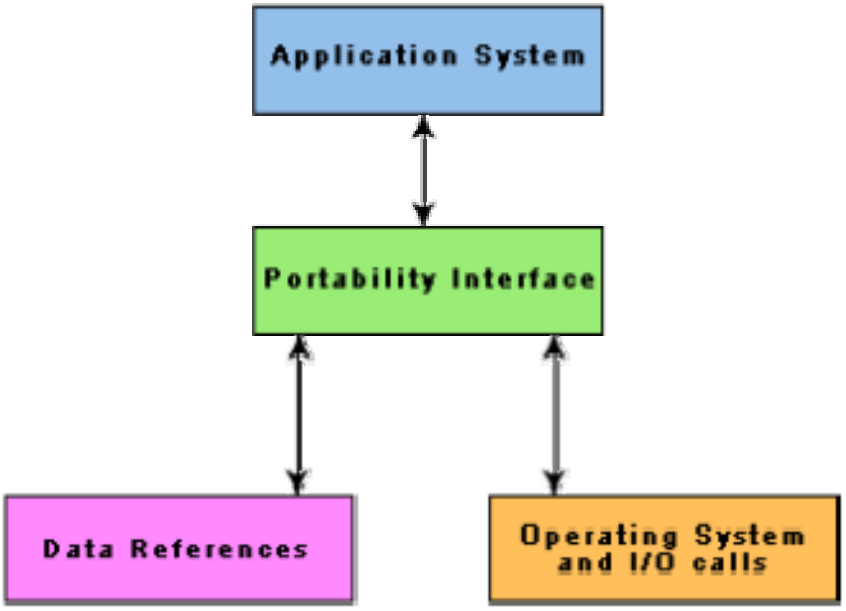

Fig. 16.1: Improving reusability of a component by using a portability interface

Factors that inhibit an effective reuse program

In spite of all the shortcomings of the state-of-the-art reuse techniques, it is the experience of several organizations that most of the factors inhibiting an effective reuse program are non-technical. Some of these factors are the following.

- Need for commitment from the top management.
- Adequate documentation to support reuse.
- Adequate incentive to reward those who reuse. Both the people contributing new reusable components and those reusing the existing components should be rewarded to start a reuse program and keep it going.
- Providing access to and information about reusable components. Organizations are often hesitant to provide an open access to the reuse repository for the fear of the reuse components finding a way to their competitors.

The following questions have been designed to test the objectives identified for this module:

1. Why is it important for an organization to undertake an effective reuse program?

2. What are the important artifacts that can be reused?

3. Why is reuse of software components much more difficult than hardware components?

4. Do you agree with the statement: “code” is the most important reuse artifact that can be used during software development.

5. Identify the reasons why reuse of mathematical software is so successful. Also, identify the reasons why the reuse of software components other than those of the mathematical software is difficult.

6. What are the issues that must be clearly understood for starting any reuse program?

7. Devise a scheme to store software reuse artifacts. Explain how components can be searched in that scheme.

8. What do you understand by the term reuse domain?

9. How does domain analysis increase software reusability?

10. Identify the stages through which a reuse domain progresses.

11. What do you understand by the term “faceted classification” in the context of software reuse? How does faceted classification simplify component search in a component store?

12. What is meant by the term “application generator”?

13. Why reuse is easier while using an application generator compared to a component library?

14. What are the shortcomings of an application generator?

15. How can you improve reusability of the components you have identified for reuse during program development?

Mark all options which are true.

1. Component-based software development leads to

□ high quality software product
□ reduced development cost
□ reduced development time
□ all of the above

2. Which of the following kinds of artifacts associated with software development can be reused?

□ requirements specification
□ design
□ code
□ knowledge
□ all of the above

3. The most abstract artifact associated with software development that can be reused is

□ requirements specification
□ design
□ code
□ knowledge
□ test cases

4. For efficient access, the reusable components are needed to be stored in which of the following systems?

□ Relational Database Management System (RDBMS)
□ Object-Oriented Database System (ODBMS)
□ both of RDBMS and ODBMS

5. Domain analysis identifies which of the following?

□ objects
□ operations
□ relationships among objects
□ all of the above

6. The actual construction of the reusable components for a domain is called

□ domain analysis
□ domain engineering
□ component creation
□ none of the above

7. Application generators can successfully be applied to

□ data processing application
□ user interface
□ compiler development
□ all of the above

8. Which of the following steps is required to achieve successful organization-level reuse?

   □ assess of an item's potential for reuse
   □ refine the item for greater reusability
   □ enter the product in the reuse repository
   □ all of the above

Mark the following statements as either True or False. Justify your answer.

1. We can easily create components that can be reused in different software development applications.

2. The reuse of commonly used mathematical functions is very much easy.

3. Classification of reusable components is very much required for component indexing and storage.

4. A component repository requires continuous maintenance.

5. Application generators translate specifications into application programs.