Module 16

Software Reuse

Lesson 39

Basic Ideas on Software Reuse

Specific Instructional Objectives

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

- Explain the advantages of software reuse.
- Identify the artifacts that can be reused during software development.
- Explain the pros and cons of knowledge reused.
- Explain why reuse of commonly used mathematical functions is easy to achieve.
- Identify the basic issues that must be clearly addressed for starting any reuse program.
- Explain what is meant by domain analysis.

Advantages of software reuse

Software products are expensive. Software project managers are worried about the high cost of software development and are desperately look for ways to cut development cost. A possible way to reduce development cost is to reuse parts from previously developed software. In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.

Artifacts that can be reused

It is important to know about the kinds of the artifacts associated with software development that can be reused. Almost all artifacts associated with software development, including project plan and test plan can be reused. However, the prominent items that can be effectively reused are:

- Requirements specification
- Design
- Code
- Test cases
- Knowledge

Pros and cons of knowledge reuse

Knowledge is the most abstract development artifact that can be reused. Out of all the reuse artifacts i.e. requirements specification, design, code, test cases, reuse of knowledge occurs automatically without any conscious effort in this direction. However, two major difficulties with unplanned reuse of knowledge are that a developer experienced in one type of software product might be included in a team developing a different type of software. Also, it is difficult to remember

the details of the potentially reusable development knowledge. A planned reuse of knowledge can increase the effectiveness of reuse. For this, the reusable knowledge should be systematically extracted and documented. But, it is usually very difficult to extract and document reusable knowledge.

Easiness of reuse of mathematical functions

The routines of mathematical libraries are being reused very successfully by almost every programmer. No one in his right mind would think of writing a routine to compute sine or cosine. Reuse of commonly used mathematical functions is easy. Several interesting aspects emerge. Cosine means the same to all. Everyone has clear ideas about what kind of argument should cosine take, the type of processing to be carried out and the results returned. Secondly, mathematical libraries have a small interface. For example, cosine requires only one parameter. Also, the data formats of the parameters are standardized.

Basic issues in any reuse program

The following are some of the basic issues that must be clearly understood for starting any reuse program.

- Component creation
- Component indexing and storing
- Component search
- Component understanding
- Component adaptation
- Repository maintenance

Component creation. For component creation, the reusable components have to be first identified. Selection of the right kind of components having potential for reuse is important. Domain analysis is a promising technique which can be used to create reusable components.

Component indexing and storing. Indexing requires classification of the reusable components so that they can be easily searched when looking for a component for reuse. The components need to be stored in a Relational Database Management System (RDBMS) or an Object-Oriented Database System (ODBMS) for efficient access when the number of components becomes large.

Component searching. The programmers need to search for right components matching their requirements in a database of components. To be able to search components efficiently, the programmers require a proper method to describe the components that they are looking for.

Component understanding. The programmers need a precise and sufficiently complete understanding of what the component does to be able to decide whether they can reuse the component. To facilitate understanding, the components should be well documented and should do something simple.

Component adaptation. Often, the components may need adaptation before they can be reused, since a selected component may not exactly fit the problem at hand. However, tinkering with the code is also not a satisfactory solution because this is very likely to be a source of bugs.

Repository maintenance. A component repository once is created requires continuous maintenance. New components, as and when created have to be entered into the repository. The faulty components have to be tracked. Further, when new applications emerge, the older applications become obsolete. In this case, the obsolete components might have to be removed from the repository.

Domain analysis

The aim of domain analysis is to identify the reusable components for a problem domain.

Reuse domain. A reuse domain is a technically related set of application areas. A body of information is considered to be a problem domain for reuse, if a deep and comprehensive relationship exists among the information items as categorized by patterns of similarity among the development components of the software product. A reuse domain is shared understanding of some community, characterized by concepts, techniques, and terminologies that show some coherence. Examples of domains are accounting software domain, banking software domain, business software domain, manufacturing automation software domain, telecommunication software domain, etc.

Just to become familiar with the vocabulary of a domain requires months of interaction with the experts. Often, one needs to be familiar with a network of related domains for successfully carrying out domain analysis. Domain analysis identifies the objects, operations, and the relationships among them. For example, consider the airline reservation system, the reusable objects can be seats, flights, airports, crew, meal orders, etc. The reusable operations can be scheduling a flight, reserving a seat, assigning crew to flights, etc. The domain analysis generalizes the application domain. A domain model transcends specific applications. The common characteristics or the similarities between systems are generalized.

During domain analysis, a specific community of software developers gets together to discuss community-wide-solutions. Analysis of the application domain is required to identify the reusable components. The actual construction of reusable components for a domain is called domain engineering.

Evolution of a reuse domain. The ultimate result of domain analysis is development of problem-oriented languages. The problem-oriented languages are also known as application generators. These application generators, once developed form application development standards. The domains slowly develop. As a domain develops, it is distinguishable the various stages it undergoes:

Stage 1: There is no clear and consistent set of notations. Obviously, no reusable components are available. All software is written from scratch.

Stage 2: Here, only experience from similar projects is used in a development effort. This means that there is only knowledge reuse.

Stage 3: At this stage, the domain is ripe for reuse. The set of concepts are stabilized and the notations standardized. Standard solutions to standard problems are available. There is both knowledge and component reuse.

Stage 4: The domain has been fully explored. The software development for the domain can be largely automated. Programs are not written in the traditional sense any more. Programs are written using a domain specific language, which is also known as an application generator.