Chapter 1 Reuse and Component-Based Systems

1.1 Software Reuse and Components

Reuse is based on the observation that similar development circumstances recur in the development of computing systems frequently. Reuse is a very broad term covering the general concept of a reusable asset. Reuse of these assets is essential to reduce the development costs and time.

Reuse could happen at various system abstraction levels:
- Requirements and system specification
- System Design
- Code

Components are for system composition. Composition enables prefabricated parts to be reused by rearranging them in new composites. Components are reusable assets.

In this course, we focus on software components. To be specific, software components are binary units or program code of independent production, acquisition, and deployment that interact to form a functioning system.

The benefits of reuse are summarized as follows.

- Increased reliability. Reused components that have been exercised in working systems should be more reliable than new components. They have been tried and tested in a variety of different environments. Design and implementation faults are discovered and eliminated in the initial use of the components.
- Reduced process risk. If a component exists, there is less uncertainty in the cost of reusing that component than in the cost of development. This is an important factor for project management as it reduces the uncertainties in project cost estimation, especially when large components are used.
• Effective use of specialists. Instead of application specialists doing the same work on different projects, these specialists can develop reusable components which encapsulate their knowledge.
• Standards compliance. Some standards, such as user interface standards, can be implemented as a set of standard components. For example, reusable components may be developed to implement menus in a user interface. All application present the same menu formats to users. The use of standard user interfaces improves reliability as users are less likely to make mistakes when presented with a familiar interface.
• Accelerated development. Bringing a system to market as early as possible is often more important than overall development costs. Reusing components speeds up system production because both development and validation time should be reduced.

However, there are some costs and problems associated with reuse:

• Maintaining a component library. Populating a component library and ensuring that software developers can use this library can be expensive. Current techniques for classifying, cataloguing and retrieving software components are immature.
• Finding and adapting components. Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment.
• Not-invented-here syndrome. Some software engineers sometimes prefer to rewrite components as they believe that they can improve on the reusable component.

1.2 Component-based systems

Composite systems composed of software components are called component-based systems. Nowadays, these kind of systems are quite popular in IT communities.

Inevitability of components
- To speed up development process
- To reduce development cost

However, compared with make-all systems, component-based systems may have some lost in flexibility and competitive edge if they are not well organised with most suitable components.
A more precise definition of software component:
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can only be deployed independently and is subject to composition by third parties.

Characteristics of components
- A component is a unit of independent deployment. I.e., functionally self-contained and independent of the specific environment.
- A component is a unit of third-party composition. I.e., a component needs to encapsulate its implementation and interact with its environment through well-defined interfaces.
- A component has no persistent state. I.e., it is required that the component cannot be distinguished from copies of its own.

1.3 RAD Model and CBS

Rapid Application Development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time period (e.g., 60 to 90 days).

Used primarily for information systems applications, the RAD approach encompasses the following phases:
- Business modeling: rough information flow
- Data modeling: refined data flow
- Process modeling: process/transactions
- Application generation: rather than creating software using conventional third generation programming language the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary).
Automated tools are a must to facilitate the construction of the software. VB and JavaBeans could be viewed as kinds of these tools.

- Test and turnover: since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, other components must be tested and all interfaces must be fully exercised.

1.4 Overview of existing techniques

A number of software techniques and packages have been developed to support component-based system development. In this course, we will introduce and use Visual Basic as the primary technique. Meanwhile, we will provide conceptual overview of other popular techniques, such as JavaBeans, COM/DCOM, Active X and CORBA in the end of the semester.