Using of Software Reuse Approaches to Develop UGELIB Web Application

Asaad Abdul-Kareem Al-Hijaj, Haidar M. Abdul-Nabi, Aziz Sabah Abdul Aziz
Dept. of Computer Science, College of Science, University of Basrah, Basrah ,Iraq.

Abstract:
In most engineering disciplines, systems are designed by composing existing components that have been used in other systems. Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on *systematic software reuse.*
The Application system reuse involves the reuse of entire application systems either by configuring a system for an environment or by integrating two or more systems to create a new application.This research discuss the development of systems by reuse-based model and explain their benefits, some problems and illustrates different approaches. Therefore, adopts tow of these approaches (COTS product reuse , Generative Programming) to build a web application project named UGELIB (User Growth Electronic Library) which designed as Web site consists of networked database. We use the appachi web server with SQL and PHP languages within that server to construct the web components. Some of these components are COTS and others are constructed by the project team. The site allows for visitors to explore, open, download, and uploads materials. Thus, this approach satisfy the public interest.

Keywords: Software reuse, reused based, Web Engineering, Software Engineering, Web Engineering Process, Web application development, CBSE, COTS.

المستخلص:

في اغلب التخصصات الهندسية تصمم النظم من خلال استغلال المكونات المتوفرة والتي استخدمت في نظم أخرى. لقد ركزت هندسة البرمجيات فيما مضى على التطوير أساسا ولكن التركيز الآن هو للحصول على البرمجيات الأفضل والأسرع والأقل كلفة وذلك من خلال اعتماد عملية تصميم مبنية على إعادة استخدام البرمجيات المنظم.إعادة استخدام التطبيقات يكون إما بتشكيل النظام بحيث يتلاءم مع بيئة ما، أو بتكامل نظامين أو أكثر لإنشاء تطبيق جديد.في هذا البحث نوقشت طريقة تطوير النظم بإعادة الاستخدام Reused-Based وفوائدها ومشاكلها وتوجهاتها المختلفة. ومن ثم تطرق البحث لاعتماد اثنان من تلك التوجهات وهي (منتجات المكونات الجاهزة COTS، والبرمجة التوليدية) لبناء تطبيق ويب UGELIB (مكتبة الكترونية تنمو من مستخدميها) حيث صمم التطبيق كموقع ويب يتضمن قاعدة بيانات شبكية. استخدمنا خادم الموقع appachi web server مع لغة SQL ولغة PHP لبناء مكونات الموقع. بعض تلك المكونات جاهزة COTS والبعض الآخر تم بناءه من قبل فريق تطوير المشروع. ليتسنى بعد ذلك دخول المستخدمين (الزوار) للاطلاع والإضافة وبما يحقق الفائدة العامة.

الكلمات المفتاحية: برمجيات بإعادة الاستخدام، إعادة الاستخدام، هندسة الويب، هندسة البرمجيات، عملية هندسة الويب، تطوير تطبيقات الويب، CBSE, COTS.

1. Introduction:

The design process in most engineering disciplines is based on reuse of existing systems or components. Mechanical or electrical engineers do not normally specify a design where every component has to be manufactured specially. The design on components, which have been tried, and tested in other systems. These are not just small components such as flanges and valves but include major subsystems such as engines, condensers and turbines [1].

Reuse saves time and effort. There are many techniques to realize reuse at every level of software development process. Those at the detailed design and code level are well known and documented [2].

Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on *systematic software reuse.*

2. Reuse-based software engineering:

The software units that are reused may be of radically different sizes. For example [1]:

1) Application system reuse: The whole of an application system may be reused either by incorporating it without change into other systems (COTS reuse) or by developing application families. Widely practised as software systems are implemented as application families. COTS reuse is becoming increasingly common.

2) Component reuse: Components of an application from sub-systems to single objects may be reused. Now seen as the key to effective and widespread reuse through component-based software engineering. However, it is still relatively immature.

3) Object and Function reuse: Software components that implement a single well-defined function may be reused. Common in some application domains (e.g. engineering) where domain-specific libraries of reusable functions have been established.

2.2 Benefits of reuse:

QSM Associates, Inc. reports that component-based development leads to a 70% reduction in development cycle time; an 84% reduction in project cost; and productivity index of 26.2, compared to an industry norm of 16.9 [3].

2.3 Reuse problems [1]:

- Increased maintenance costs.
- Lack of tool support.
- Not-invented-here syndrome.
- Maintaining a component library.
- Finding and adapting reusable components.

2.4 Reusable Software Resources

Four software resource categories that should be considered as planning proceeds [4]:

Off-the-shelf components. Existing software that can be acquired from a third party or that has been developed internally for a past project. COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project, and have been fully validated.

Full-experience components. Existing specifications, designs, code, ortest data developed for past projects that are similar to the software to be built for the current project. Members of the current software team have had full experience in the application area represented by these components. Therefore, modifications required for full-experience components will be relatively low-risk.

Partial-experience components. Existing specifications, designs, code, or test data developed for past projects that are related to the software to be built for the current project but will require substantial modification. Members of the current software team have only limited experience in the application area represented by these components. Therefore, modifications required for partial- experience components have a fair degree of risk.

New components. Software components that must be built by the software team specifically for the needs of the current project.

2.5 The reuse landscape:

Although reuse is often simply thought of as the reuse of system components, there are many different approaches to reuse that may be used. Reuse is possible at a range of levels from simple functions to complete application systems. The reuse landscape covers the range of possible reuse techniques as shown in Fig. (1) [1, 2]:

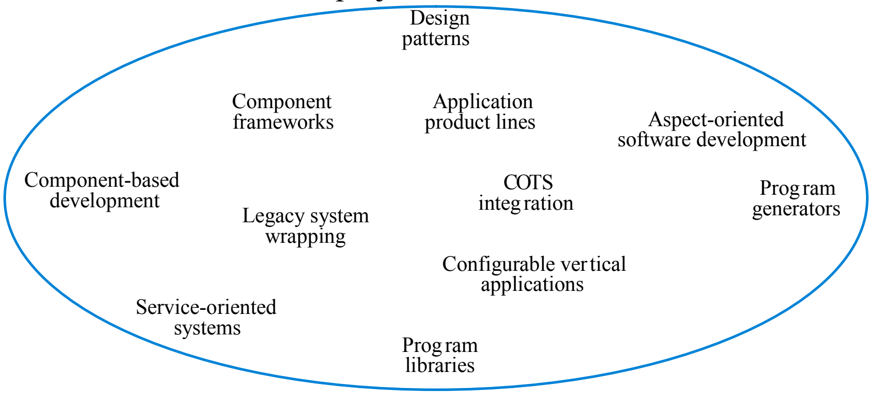


Fig. (1)- The reuse landscape.

2.6 Reuse Concept:

The concept of a software component is a "description of what a component does", The concept should communicate the intent of the component [5].

When you reuse program or design components, you have to follow the design decisions made by the original developer of the component. This may limit the opportunities for reuse.

However, a more abstract form of reuse is the reuse concept when a particular approach is described in an implementation independent way and an implementation is then developed. The two main approaches to the reuse concept are [1]:

- Design patterns;
- Generative programming.

2.6.1 Design patterns

The architectural design defines the relationship between major structural elements of the software, the "design patterns" that can be used to achieve the requirements that have been defined for the system, and the constraints that affect the way in which architectural design patterns can be applied [6].

2.6.2 Generator-based reuse

Program generators involve the reuse of standard patterns and algorithms. These are embedded in the generator and parameterised by user commands. A program is then automatically generated. Generator-based reuse is possible when domain abstractions and their mapping to executable code can be identified. A domain specific language is used to compose and control these abstractions as shown in Fig. (2) [1].

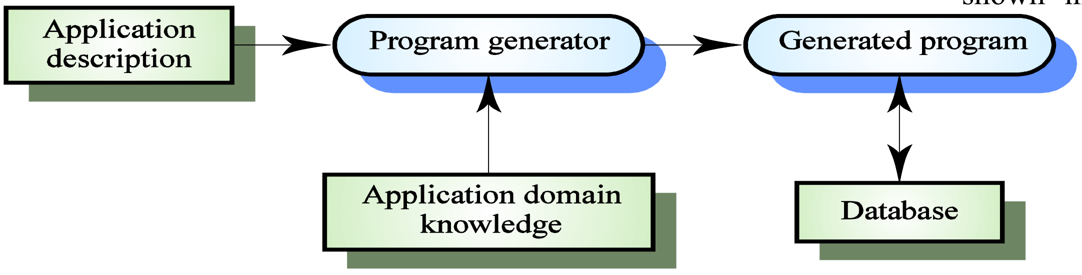


Fig. (2)
The program generation reuse process.

2.7Application system reuse

Involves the reuse of entire application systems either by configuring a system for an environment or by integrating two or more systems to create a new application.
Two approaches covered here:

- COTS product integration;
- Product line development.

2.7.1 Component-based SE development:

Component-based software engineering (CBSE) is a process that emphasizes the design and construction of computer-based systems using reusable software “components.” [7] describes CBSE in the following way: CBSE is changing the way large software systems are developed. CBSE embodies the “buy, don’t build” philosophy espoused by Fred Brooks and others. In the same way that early subroutines liberated the programmer from thinking about details, CBSE shifts the emphasis from programming software to composing software systems. Implementation has given way to integration as the focus. At its foundation is the assumption that there is sufficient commonality in many large software systems to justify developing reusable components to exploit and satisfy that commonality.

2.7.1.1 Components

These components, consisting of interface objects, application objects, and database objects, establish how the data are to be processed [2].
Components provide a service without regard to where the component is executing or its programming language [1]:

- A component is an independent executable entity that can be made up of one or more executable objects
- The component interface is published and all interactions are through the published interface

Components can range in size from simple functions to entire application systems.

2.7.1.2 Component interfaces

- Provides interface

Defines the services that are provided by the component to other components

- Requires interface

Defines the services that specifies what services must be made available for the component to execute as specified. As shown in Fig. (3) [1, 2]:

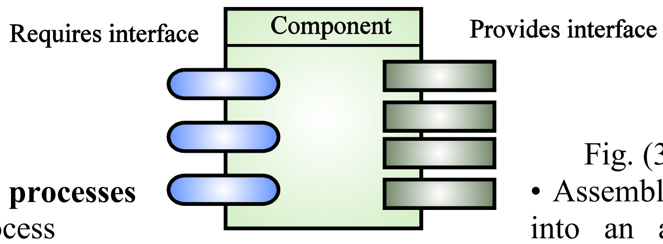


Fig. (3) The components interface

2.7.1.3 CBSE processes

The CBSE process yields [8]:

• Qualified components—assessed by software engineers to ensure that not only functionality, but performance, reliability, usability, and other quality factors conform to the requirements of the system or product to be built.

• Adapted components—adapted to modify (also called mask or wrap) unwanted or undesirable characteristics.

• Assembled components—integrated into an architectural style and interconnected with an appropriate infrastructure that allows the components to be coordinated and managed effectively.

• Updated components—replacing existing software as new versions of components become available.

CBSE usually involves a prototyping or an incremental development process with components being ‘glued together’ using a scripting language as shown in Fig. (4):

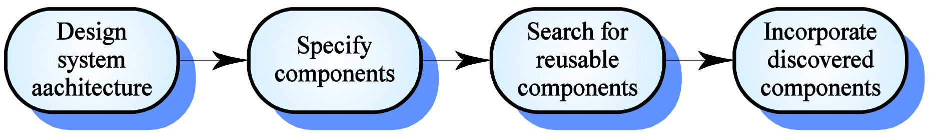

Design system aachitecture
Specify components
Search for reusable components
Incorporate discovered components

Fig. (4) The CBSE development process.

2.7.1.4 CBSE problems [1]:

- Component incompatibilities may mean that cost and schedule savings are less then expected.
- Finding and understanding components.
- Managing evolution as requirements change in situations where it may be impossible to change the system components.

3. The UGELIB Web System:

The resources being managed are the E-Books, Thesis, Magazines, Multimedia (Video, Audio, Images), and Programs (Utilities, Anti Viruses …), in the library. Additional domain-specific functionality (issue Licences, Download, Upload, search, etc.) must be added for this application as shown in Fig. (5) and Fig. (6):

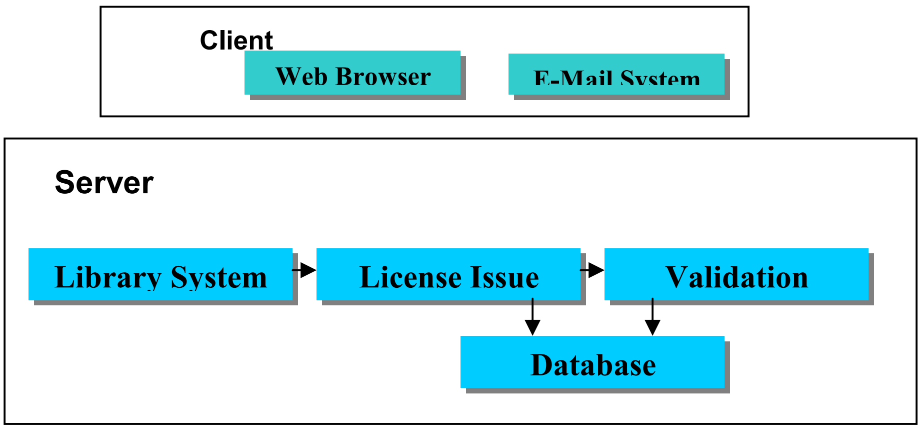


Fig. (5)
The UGELIB Client Server architecture.

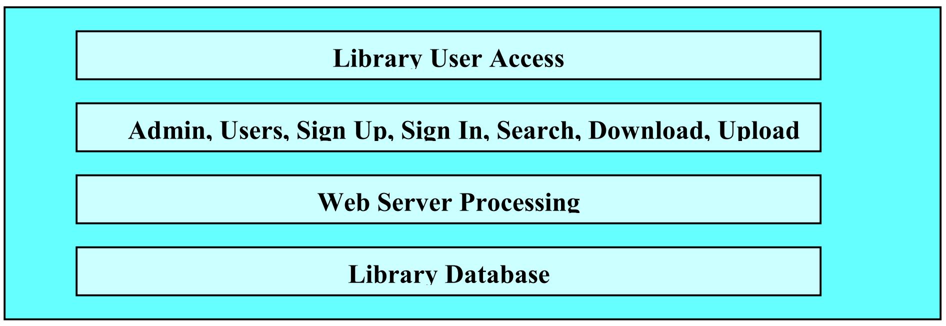


Fig. (6) The UGELIB architecture.

3.1 System modelling

System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers. Different models present the system from different perspectives [9]:

- *External perspective* showing the system's context or environment

- *Behavioural perspective* showing the behaviour of the system
- *Structural perspective* showing the system or data architecture

3.2 The Unified Modelling Language:

UML was devised by the developers of widely used object-oriented analysis and design methods. Has become an effective standard for object-oriented modelling. Object classes are rectangles with the name at the top, attributes in the middle section and operations in the bottom section. Relationships between object classes (known as associations) are shown as lines linking objects. Inheritance is referred to as generalisation and is shown 'upwards' rather than 'downwards' in a hierarchy as shown in Fig. (7) and Fig. (8):

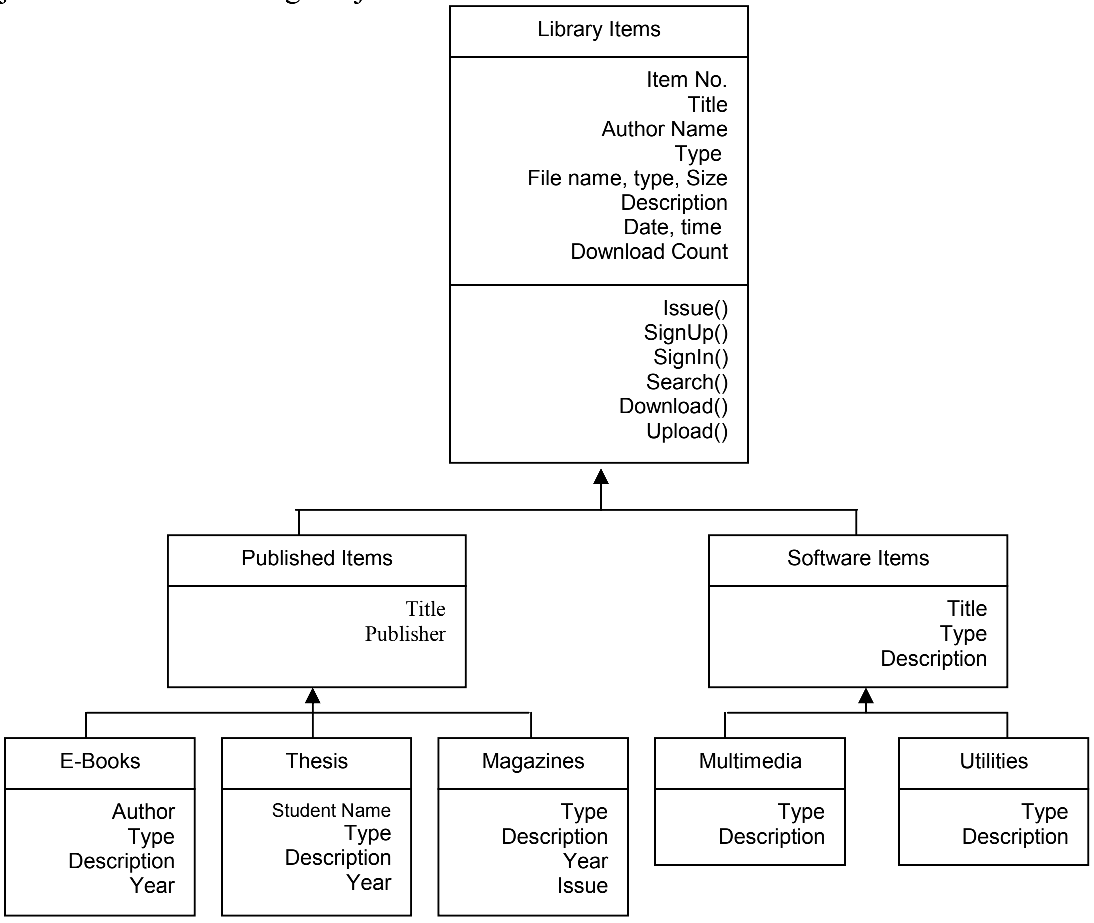


Fig. (7) The UGELIB class hierarchy.

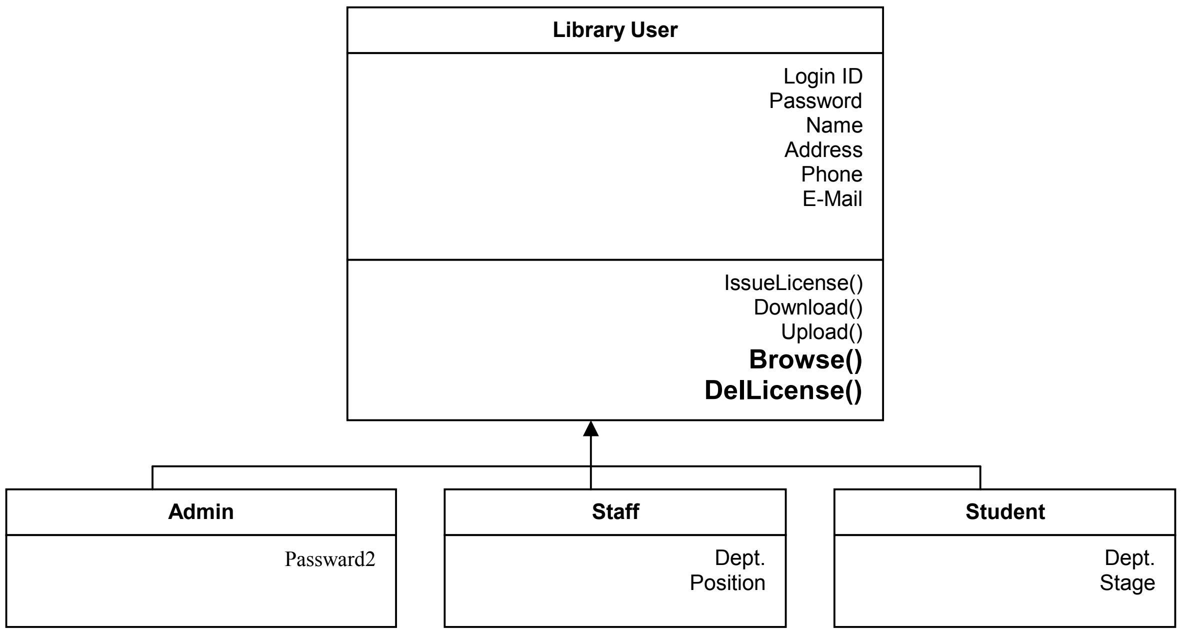


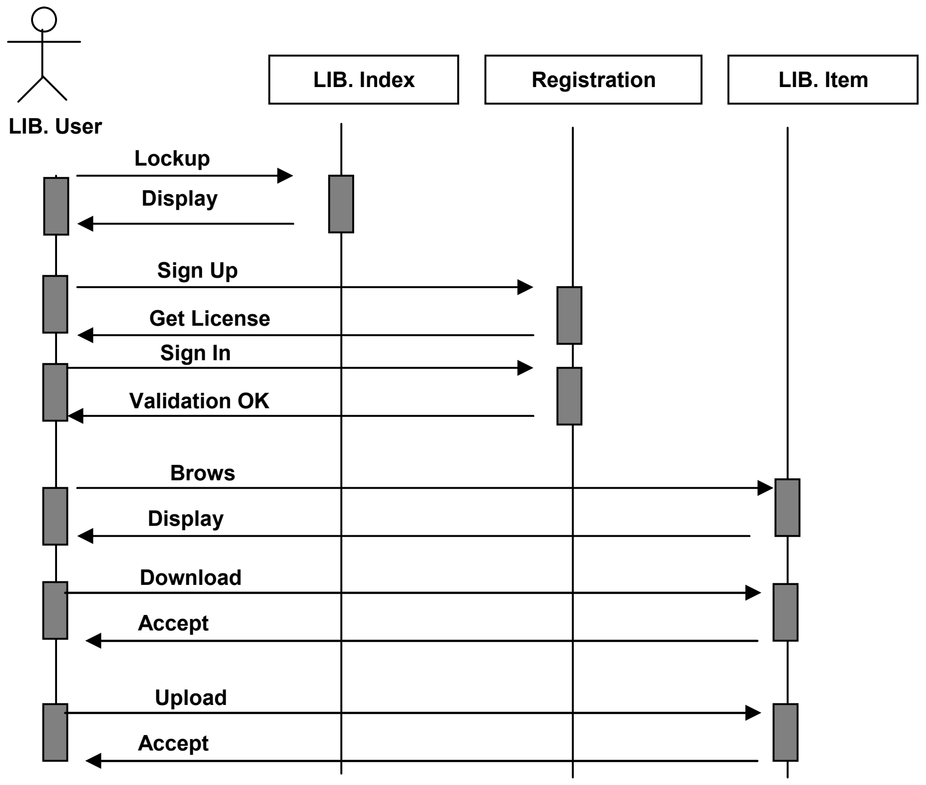
Fig. (8) The User class hierarchy.

3.3 Multiple inheritances

Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritances allows object classes to inherit from several super-classes. Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics. Makes class hierarchy reorganisation more complex.

3.4 Object behaviour modelling

A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a *use-case*. Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects as shown in Fig. (9). [1]



4. Conclusion

In this paper we have discussed how to engineer Web applications and focusing on Reuse-Based approaches . We introduced Web Design with CBSE as a method to design in Web applications. A design contains the specification of both the behavior and navigational structure of Web applications in a particular domain.

We have also introduced UGELIB web application. We showed that Web design can be mapped into an UGELIB model and then into a Web application by using the UGELIB -Web environment or other Web implementation tools. We showed that design web application could also be mapped in a straightforward way into object-oriented application by showing a specific architecture. One of the most important architectural components in Web Design application is Generic Navigational model. The model are recurrent COTS in Web applications since they usually deal with sets of similar objects. We are now incorporating other navigation method into UGELIB web application to enhance its accessibility power. We strongly believe that development; delivery and maintenance times in the Web domain require reuse-centric approaches. The systematic reuse of semi-complete design structures, as described by Web design application is a key approach for maximizing reuse in Web application development.

References

1. Ian Sommerville, “Software Engineering”, Seventh Edition, Pearson Education Limited, 2004.
2. Roger S. Pressman, “Software Engineering, A Practitioner’s Approach ”, Sixth Edition, McGraw Hill Co, 2005
3. Yourdon, E., “Software Reuse”, Application Development Strategies, vol. 6, no. 12, December, pp 1-16, 1994.
4. Bennatan, E.M., "Software Project Management: A Practitioner’s Approach", McGraw-Hill, 1992.
5. Whittle, B., "Models and Languages for Component Description and Reuse", ACM Software Engineering Notes, Vol. 20, no. 2, pp 76-89, April 1995.
6. Shaw, M. and D. Garlan, "Software Architecture", Prentice-Hall, 1996.
7. Clements, P.C., “From Subroutines to Subsystems: Component Based Software Development,” American Programmer, vol. 8, No. 11, November 1995.
8. Brown, A.W. and K.C. Wallnau, “Engineering of Component Based Systems,” Component-Based Software Engineering, IEEE Computer Society Press, pp. 7–15, 1996.
9. Asaad Abdul-Kareem Al-Hijaj, Haidar M. Abdul-Nabi; "Development of ISIRS Web Application Using Object Behavioral Modeling", Proceedings of the 1st International Conference on the KUFA University, Iraq 2009.