**Component-Based Architecture**

Component-based architecture focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.

معماری مبتنی بر کامپوننت بر تجزیه طراحی به اجزای عملکردی یا منطقی فردی متمرکز است که نمایانگر رابطهای ارتباطی کاملاً تعریف شده و حاوی روشها ، رویدادها و خصوصیات است. سطح بالاتری از انتزاع را فراهم می کند و مسئله را به مشکلات فرعی تقسیم می کند که هر یک با پارتیشن های مؤلفه همراه است.

The primary objective of component-based architecture is to ensure **component reusability**. A component encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit. There are many standard component frameworks such as COM/DCOM, JavaBean, EJB, CORBA, .NET, web services, and grid services. These technologies are widely used in local desktop GUI application design such as graphic JavaBean components, MS ActiveX components, and COM components which can be reused by simply drag and drop operation.

هدف اصلی معماری مبتنی بر کامپوننت اطمینان از قابلیت استفاده مجدد از اجزاء است. یک کامپوننت عملکرد و رفتارهای یک عنصر نرم افزاری را در یک واحد باینری قابل استفاده مجدد و قابل استقرار قرار می دهد. چارچوب های کامپوننت استاندارد بسیاری مانند COM / DCOM ، JavaBean ، EJB ، CORBA ، .NET ، خدمات وب و خدمات شبکه وجود دارد. این فناوری ها به طور گسترده ای در طراحی برنامه GUI دسک تاپ محلی مانند اجزای گرافیکی JavaBean ، اجزای MS ActiveX و اجزای COM مورد استفاده قرار می گیرند که با استفاده از عملیات کشیدن و رها کردن قابل استفاده مجدد هستند.

Component-oriented software design has many advantages over the traditional object-oriented approaches such as −

* Reduced time in market and the development cost by reusing existing components.
* Increased reliability with the reuse of the existing components.

طراحی نرم افزار کامپوننت گرا نسبت به رویکردهای سنتی شی گرا مانند -  
• با استفاده مجدد از اجزای موجود ، کاهش زمان در بازار و هزینه توسعه.  
• با استفاده مجدد از اجزای موجود ، قابلیت اطمینان بیشتر شده است.

## What is a Component?

A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.

یک مؤلفه یک مجموعه عملکردی تعریف شده ، قابل حمل ، قابل تعویض و قابل استفاده مجدد است که اجرای آن را محصور کرده و آن را به عنوان یک رابط سطح بالاتر صادر می کند.

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behavior common to all components within an architecture.

یک کامپوننت یک شیء نرم افزاری است ، که برای تعامل با سایر کامپوننت ها ، محاصره کردن عملکردهای خاص یا مجموعه ای از قابلیت ها در نظر گرفته شده است. این یک رابط کاملاً مشخص تعریف شده است و مطابق با یک رفتار پیشنهادی معمول برای همه کامپوننت های یک معماری است.

A software component can be defined as a unit of composition with a contractually specified interface and explicit context dependencies only. That is, a software component can be deployed independently and is subject to composition by third parties.

یک کامپوننت نرم افزاری را می توان به عنوان واحدی از ترکیب با رابط مشخص شده با قرارداد و وابستگیهای صریح به آنها تعریف کرد. یعنی یک کامپوننت نرم افزاری می تواند به طور مستقل مستقر شود و توسط اشخاص ثالث در معرض ترکیب قرار گیرد.

### Views of a Component

A component can have three different views − object-oriented view, conventional view, and process-related view.

یک کامپوننت می تواند دارای سه دیدگاه متفاوت باشد - نمای شی گرا ، نمای معمولی و نمای مرتبط با فرآیند.

**Object-oriented view**

A component is viewed as a set of one or more cooperating classes. Each problem domain class (analysis) and infrastructure class (design) are explained to identify all attributes and operations that apply to its implementation. It also involves defining the interfaces that enable classes to communicate and cooperate.

یک کامپوننت به عنوان مجموعه ای از یک یا چند کلاس همکاری مشاهده می شود. هر کلاس دامنه مشکل (تجزیه و تحلیل) و کلاس زیرساختی (طراحی) برای شناسایی کلیه ویژگیها و عملکردهایی که برای اجرای آن اعمال می شود توضیح داده شده است. همچنین شامل تعریف رابط هایی است که کلاس ها را قادر به برقراری ارتباط و همکاری می کنند.

**Conventional view**

It is viewed as a functional element or a module of a program that integrates the processing logic, the internal data structures that are required to implement the processing logic and an interface that enables the component to be invoked and data to be passed to it.

این به عنوان یک عنصر کاربردی یا یک ماژول برنامه مشاهده می شود که منطق پردازش ، ساختارهای داده داخلی را که برای اجرای منطق پردازش و یک واسط مورد نیاز هستند ، مورد استفاده قرار می دهد که می توان از این مؤلفه فراخوانی شد و داده ها را به آن منتقل کرد.

**Process-related view**

In this view, instead of creating each component from scratch, the system is building from existing components maintained in a library. As the software architecture is formulated, components are selected from the library and used to populate the architecture.

در این دیدگاه ، سیستم به جای ایجاد هر مؤلفه از ابتدا ، از اجزای موجود که در یک کتابخانه نگهداری می شوند ، ساخته می شود. با تدوین معماری نرم افزار ، مؤلفه هایی از این کتابخانه انتخاب شده و برای جمع آوری معماری مورد استفاده قرار می گیرند.

* A user interface (UI) component includes grids, buttons referred as controls, and utility components expose a specific subset of functions used in other components.
* Other common types of components are those that are resource intensive, not frequently accessed, and must be activated using the just-in-time (JIT) approach.
* Many components are invisible which are distributed in enterprise business applications and internet web applications such as Enterprise JavaBean (EJB), .NET components, and CORBA components.
* یک مؤلفه رابط کاربر (UI) شامل شبکه ها ، دکمه هایی که به عنوان کنترل نامیده می شوند و اجزای ابزار زیر مجموعه خاصی از توابع مورد استفاده در سایر اجزا را در معرض نمایش قرار می دهند.  
  • سایر انواع متداول قطعات متشکل از منابع هستند ، که اغلب به آنها دسترسی نمی یابند و باید با استفاده از رویکرد فقط زمان (JIT) فعال شوند.  
  • بسیاری از مؤلفه ها نامرئی هستند که در برنامه های شغلی سازمانی و برنامه های وب اینترنتی مانند Enterprise JavaBean (EJB) ، مؤلفه های دات نت و اجزای CORBA توزیع می شوند.

### Characteristics of Components

* **Reusability** − Components are usually designed to be reused in different situations in different applications. However, some components may be designed for a specific task.
* **Replaceable** − Components may be freely substituted with other similar components.
* **Not context specific** − Components are designed to operate in different environments and contexts.
* **Extensible** − A component can be extended from existing components to provide new behavior.
* **Encapsulated** − A A component depicts the interfaces, which allow the caller to use its functionality, and do not expose details of the internal processes or any internal variables or state.
* **Independent** − Components are designed to have minimal dependencies on other components.

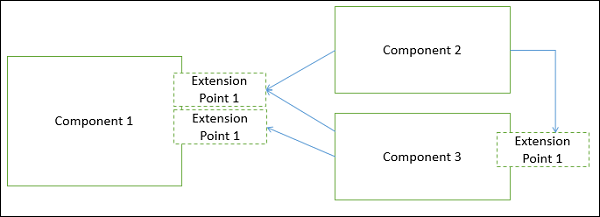
قابلیت استفاده مجدد - قطعات معمولاً برای استفاده مجدد در موقعیت های مختلف در برنامه های مختلف طراحی می شوند. با این حال ، برخی از مؤلفه ها ممکن است برای یک کار خاص طراحی شوند.  
• قابل تعویض - ممکن است اجزاء با اجزای مشابه دیگر آزادانه جایگزین شوند.  
• نه متن خاص - کامپوننت ها برای کار در محیط ها و زمینه های مختلف طراحی شده اند.  
• گسترده - یک مؤلفه را می توان از اجزای موجود گسترش داد تا رفتار جدیدی ارائه دهد.  
• محصور شده - یک جزء رابط ها را نشان می دهد ، که به تماس گیرنده امکان می دهد از قابلیت های خود استفاده کند ، و جزئیات فرآیندهای داخلی یا متغیرهای داخلی یا حالت را در معرض نمایش قرار نمی دهد.  
• Independent - کامپوننت ها طوری طراحی شده اند که حداقل وابستگی به سایر مؤلفه ها داشته باشند.

## Principles of Component−Based Design

A component-level design can be represented by using some intermediary representation (e.g. graphical, tabular, or text-based) that can be translated into source code. The design of data structures, interfaces, and algorithms should conform to well-established guidelines to help us avoid the introduction of errors.

طراحی سطح مؤلفه را می توان با استفاده از برخی از واسطه ها (به عنوان مثال گرافیکی ، جدولی یا متن محور) ارائه داد که می تواند به کد منبع ترجمه شود. طراحی ساختار داده ها ، رابط ها و الگوریتم ها باید با دستورالعمل های درست تطابق داشته باشد تا به ما در جلوگیری از معرفی خطاها کمک کند.

* The software system is decomposed into reusable, cohesive, and encapsulated component units.
* Each component has its own interface that specifies required ports and provided ports; each component hides its detailed implementation.
* A component should be extended without the need to make internal code or design modifications to the existing parts of the component.
* Depend on abstractions component do not depend on other concrete components, which increase difficulty in expendability.
* Connectors connected components, specifying and ruling the interaction among components. The interaction type is specified by the interfaces of the components.
* Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific interactions.
* For a server class, specialized interfaces should be created to serve major categories of clients. Only those operations that are relevant to a particular category of clients should be specified in the interface.
* A component can extend to other components and still offer its own extension points. It is the concept of plug-in based architecture. This allows a plugin to offer another plugin API.
* • سیستم نرم افزاری به واحدهای قابل استفاده مجدد ، منسجم و محصور شده تجزیه می شود.  
  • هر مؤلفه رابط مخصوص به خود را دارد که درگاه های مورد نیاز را تعیین کرده و درگاه های مورد نیاز را ارائه می دهد. هر مؤلفه اجرای مفصل خود را پنهان می کند.  
  • یک جزء باید بدون نیاز به ایجاد کد داخلی یا اصلاح طراحی به قسمت های موجود در مؤلفه ، گسترش یابد.  
  • بستگی به مؤلفه انتزاع به سایر اجزای بتن بستگی ندارد که باعث افزایش مشکل در هزینه می شوند.  
  • اتصالات به اتصالات متصل می شوند ، تعامل بین مؤلفه ها را تعیین و حاکم می کنند. نوع تعامل توسط واسطهای اجزا مشخص می شود.  
  • تعامل مؤلفه ها می تواند به صورت فراخوانی روش ، فراخوانی های ناهمزمان ، پخش ، تعامل با پیام ارسال شود ، ارتباطات جریان داده ها و سایر تعامل های خاص پروتکل.  
  • برای کلاس سرور ، رابطهای تخصصی ایجاد می شود تا به دسته های عمده مشتری ارائه دهد. فقط آن دسته از عملیات که به یک دسته خاص مشتری مربوط هستند باید در رابط مشخص شوند.  
  • یک مؤلفه می تواند به سایر مؤلفه ها گسترش یابد و هنوز نقاط تمدید خود را ارائه دهد. این مفهوم معماری مبتنی بر افزونه است. این به یک افزونه اجازه می دهد تا یک API افزونه دیگر ارائه دهد.



## Component-Level Design Guidelines

Creates a naming conventions for components that are specified as part of the architectural model and then refines or elaborates as part of the component-level model.

برای مؤلفه هایی که به عنوان بخشی از مدل معماری مشخص شده اند ، یک کنوانسیون نامگذاری ایجاد می کند و بعنوان بخشی از مدل سطح مؤلفه ها ، تصفیه شده یا تفسیر می شود.

* Attains architectural component names from the problem domain and ensures that they have meaning to all stakeholders who view the architectural model.
* Extracts the business process entities that can exist independently without any associated dependency on other entities.
* Recognizes and discover these independent entities as new components.
* Uses infrastructure component names that reflect their implementation-specific meaning.
* Models any dependencies from left to right and inheritance from top (base class) to bottom (derived classes).
* Model any component dependencies as interfaces rather than representing them as a direct component-to-component dependency.
* • نامهای مؤلفه های معماری را از حوزه مسئله بدست می آورد و اطمینان می دهد که آنها برای همه ذینفعان که مدل معماری را مشاهده می کنند ، معنی دارند.  
  • اشخاص فرآیندی تجارت را که می توانند بطور مستقل و بدون وابستگی وابسته به نهادهای دیگر وجود داشته باشند ، استخراج می کنند.  
  • این موجودات مستقل را به عنوان مؤلفه های جدید تشخیص و کشف می کند.  
  • از نام مؤلفه های زیرساخت استفاده می کند که معنای خاص اجرای آنها را منعکس می کند.  
  • هرگونه وابستگی از چپ به راست و وراثت از بالا (کلاس پایه) به پایین (کلاسهای مشتق شده) را مدل می کند.  
  • هرگونه وابستگی به مؤلفه را به عنوان رابط ها و نه نشان دادن آنها به عنوان یک وابستگی مستقیم به مؤلفه ، از الگوی خود استفاده کنید.

## Conducting Component-Level Design

Recognizes all design classes that correspond to the problem domain as defined in the analysis model and architectural model.

تمام کلاسهای طراحی که مطابق با حوزه مسئله مطابق با مدل تحلیل و مدل معماری است را تشخیص می دهد.

* Recognizes all design classes that correspond to the infrastructure domain.
* Describes all design classes that are not acquired as reusable components, and specifies message details.
* Identifies appropriate interfaces for each component and elaborates attributes and defines data types and data structures required to implement them.
* Describes processing flow within each operation in detail by means of pseudo code or UML activity diagrams.
* Describes persistent data sources (databases and files) and identifies the classes required to manage them.
* Develop and elaborates behavioral representations for a class or component. This can be done by elaborating the UML state diagrams created for the analysis model and by examining all use cases that are relevant to the design class.
* Elaborates deployment diagrams to provide additional implementation detail.
* Demonstrates the location of key packages or classes of components in a system by using class instances and designating specific hardware and operating system environment.
* The final decision can be made by using established design principles and guidelines. Experienced designers consider all (or most) of the alternative design solutions before settling on the final design model.
* • کلیه کلاسهای طراحی که با حوزه زیرساخت مطابقت دارند را تشخیص می دهد.  
  • کلیه کلاس های طراحی را که به عنوان اجزای قابل استفاده مجدد به دست نیامده ، توصیف می کند و جزئیات پیام را مشخص می کند.  
  • رابط های مناسب برای هر مؤلفه را شناسایی کرده و ویژگی ها را بطور مفصل شرح داده و انواع داده ها و ساختار داده ها را برای اجرای آنها تعریف می کند.  
  • جریان پردازش درون هر عملیات را با جزئیات با استفاده از شبه کد یا نمودارهای فعالیت UML توصیف می کند.  
  • منابع داده مداوم (پایگاه داده ها و پرونده ها) را توصیف می کند و کلاس های لازم برای مدیریت آنها را مشخص می کند.  
  • بازنماییهای رفتاری برای یک کلاس یا مؤلفه را توسعه داده و توسعه دهید. این امر می تواند با تهیه نمودارهای حالت UML ایجاد شده برای مدل آنالیز و بررسی همه موارد استفاده که مربوط به کلاس طراحی هستند ، انجام شود.  
  • نمودارهای استقرار را برای ارائه جزئیات بیشتر در مورد اجرای طرح می کند.  
  • با استفاده از نمونه کلاس و تعیین سخت افزار خاص و محیط سیستم عامل ، مکان بسته های کلیدی یا کلاس های قطعات را در یک سیستم نشان می دهد.  
  • تصمیم نهایی با استفاده از اصول طراحی و دستورالعملهای تعیین شده می تواند گرفته شود. طراحان با تجربه قبل از استقرار در مدل طراحی نهایی ، همه (یا بیشتر) راه حل های طراحی جایگزین را در نظر می گیرند.

### Advantages

* **Ease of deployment** − As new compatible versions become available, it is easier to replace existing versions with no impact on the other components or the system as a whole.
* **Reduced cost** − The use of third-party components allows you to spread the cost of development and maintenance.
* **Ease of development** − Components implement well-known interfaces to provide defined functionality, allowing development without impacting other parts of the system.
* **Reusable** − The use of reusable components means that they can be used to spread the development and maintenance cost across several applications or systems.
* **Modification of technical complexity** − A component modifies the complexity through the use of a component container and its services.
* **Reliability** − The overall system reliability increases since the reliability of each individual component enhances the reliability of the whole system via reuse.
* **System maintenance and evolution** − Easy to change and update the implementation without affecting the rest of the system.
* **Independent** − Independency and flexible connectivity of components. Independent development of components by different group in parallel. Productivity for the software development and future software development.
* • سهولت استقرار - با دسترسی به نسخه های جدید سازگار ، جایگزین کردن نسخه های موجود بدون تأثیر بر سایر اجزا یا سیستم به طور کلی آسان تر است.  
  • کاهش هزینه - استفاده از مؤلفه های شخص ثالث به شما امکان می دهد هزینه توسعه و نگهداری را گسترش دهید.  
  • سهولت توسعه - مؤلفه ها رابط های شناخته شده ای را برای فراهم آوردن قابلیت های تعریف شده ، ایجاد می کنند و امکان توسعه بدون ایجاد تأثیر بر سایر قسمت های سیستم را فراهم می کنند.  
  قابل استفاده مجدد - استفاده از مؤلفه های قابل استفاده مجدد بدان معنی است که می توان از آنها برای پخش هزینه های توسعه و نگهداری در چندین برنامه یا سیستم استفاده کرد.  
  • اصلاح پیچیدگی فنی - یک جزء با استفاده از یک کانتینر کامپوننت و خدمات آن ، پیچیدگی را اصلاح می کند.  
  قابلیت اطمینان - قابلیت اطمینان کلی سیستم از آنجا که قابلیت اطمینان هر یک از اجزای فردی قابلیت اطمینان کل سیستم را از طریق استفاده مجدد افزایش می دهد افزایش می یابد.  
  • تعمیر و نگهداری سیستم و تکامل - بدون تغییر در بقیه سیستم آسان و آسان برای تغییر و به روز رسانی در اجرای.  
  • مستقل - استقلال و اتصال انعطاف پذیر اجزاء. توسعه مستقل از مؤلفه ها توسط گروه های مختلف به طور موازی. بهره وری برای توسعه نرم افزار و توسعه آینده نرم افزار.