•Components and component models50
•CBSE processes50
•Component composition50

•Component-based software engineering (CBSE) is an approach to software development that relies on the reuse of entities called 'software components'.26
•It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific.26
•Components are more abstract than object classes and can be considered to be stand-alone service providers. They can exist as stand-alone entities.26

•Independent components specified by their interfaces.6
•Component standards to facilitate component integration.6
•Middleware that provides support for component inter-operability.6
•A development process that is geared to reuse.6

•Apart from the benefits of reuse, CBSE is based on sound software engineering design principles:5
•Components are independent so do not interfere with each other;5
•Component implementations are hidden;5
•Communication is through well-defined interfaces;5
•One components can be replaced by another if its interface is maintained;5
•Component infrastructures offer a range of standard services.5

•Standards need to be established so that components can communicate with each other and inter-operate.24
•Unfortunately, several competing component standards were established:24
•Sun's Enterprise Java Beans24
•Microsoft's COM and .NET24
•CORBA's CCM24
•In practice, these multiple standards have hindered the uptake of CBSE. It is impossible for components developed using different approaches to work together.24

•An executable service is a type of independent component. It has a 'provides' interface but not a 'requires' interface.45
•From the outset, services have been based around standards so there are no problems in communicating between services offered by different vendors.45
•System performance may be slower with services but this approach is replacing CBSE in many systems.45
•Covered in Chapter 1845

•Components provide a service without regard to where the component is executing or its programming language27
•A component is an independent executable entity that can be made up of one or more executable objects;27
•The component interface is published and all interactions are through the published interface;27

•Councill and Heinmann:17
•A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.17
•Szyperski:17
•A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.17

•Composable15
•For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself, such as its methods and attributes.15
•Deployable15
•To be deployable, a component has to be self-contained. It must be able to operate as a stand-alone entity on a component platform that provides an implementation of the component model. This usually means that the component is binary and does not have to be compiled before it is deployed. If a component is implemented as a service, it does not have to be deployed by a user of a component. Rather, it is deployed by the service provider.15
•Documented15
•Components have to be fully documented so that potential users can decide whether or not the components meet their needs. The syntax and, ideally, the semantics of all component interfaces should be specified.15
•Independent15
•A component should be independent—it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a 'requires' interface specification.15
•Standardized15
•Component standardization means that a component used in a CBSE process has to conform to a standard component model. This model may define component interfaces, component metadata, documentation, composition, and deployment.15

•The component is an independent, executable entity. It does not have to be compiled before it is used with other components.14
•The services offered by a component are made available through an interface and all component interactions take place through that interface.14
•The component interface is expressed in terms of parameterized operations and its internal state is never exposed.14

•Provides interface20
•Defines the services that are provided by the component to other components.20
•This interface, essentially, is the component API. It defines the methods that can be called by a user of the component.20
•Requires interface20
•Defines the services that specifies what services must be made available for the component to execute as specified.20
•This does not compromise the independence or deployability of a component because the 'requires' interface does not define how these services should be provided.20

•Note UML notation. Ball and sockets can fit together.21

•Components are accessed using remote procedure calls (RPCs).13
•Each component has a unique identifier (usually a URL) and can be referenced from any networked computer.13
•Therefore it can be called in a similar way as a procedure or method running on a local computer.13

•A component model is a definition of standards for component implementation, documentation and deployment.23
•Examples of component models23
•EJB model (Enterprise Java Beans)23
•COM+ model (.NET model)23
•Corba Component Model23
•The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.23

•Interfaces32
•Components are defined by specifying their interfaces. The component model specifies how the interfaces should be defined and the elements, such as operation names, parameters and exceptions, which should be included in the interface definition.32
•Usage32
•In order for components to be distributed and accessed remotely, they need to have a unique name or handle associated with them. This has to be globally unique.32
•Deployment32
•The component model includes a specification of how components should be packaged for deployment as independent, executable entities.32

•Component models are the basis for middleware that provides support for executing components.40
•Component model implementations provide:40
•Platform services that allow components written according to the model to communicate;40
•Support services that are application-independent services used by different components.40
•To use services provided by a model, components are deployed in a container. This is a set of interfaces used to access the service implementations.40

•CBSE processes are software processes that support component-based software engineering.8
•They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.8
•Development for reuse8
•This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.8
•Development with reuse8
•This process is the process of developing new applications using existing components and services.8

•Component acquisition is the process of acquiring components for reuse or development into a reusable component.46
•It may involve accessing locally- developed components or services or finding these components from an external source.46
•Component management is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored and made available for reuse.46
•Component certification is the process of checking a component and certifying that it meets its specification.46

•CBSE for reuse focuses on component development.7
•Components developed for a specific application usually have to be generalised to make them reusable.7
•A component is most likely to be reusable if it associated with a stable domain abstraction (business object).7
•For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.7

•Components for reuse may be specially constructed by generalising existing components.18
•Component reusability18
•Should reflect stable domain abstractions;18
•Should hide state representation;18
•Should be as independent as possible;18
•Should publish exceptions through the component interface.18
•There is a trade-off between reusability and usability18
•The more general the interface, the greater the reusability but it is then more complex and hence less usable.18

•Remove application-specific methods.12
•Change names to make them general.12
•Add methods to broaden coverage.12
•Make exception handling consistent.12
•Add a configuration interface for component adaptation.12
•Integrate required components to reduce dependencies.12

•Components should not handle exceptions themselves, because each application will have its own requirements for exception handling.33
•Rather, the component should define what exceptions can arise and should publish these as part of the interface.33
•In practice, however, there are two problems with this:33
•Publishing all exceptions leads to bloated interfaces that are harder to understand. This may put off potential users of the component.33
•The operation of the component may depend on local exception handling, and changing this may have serious implications for the functionality of the component.33

•Existing legacy systems that fulfil a useful business function can be re-packaged as components for reuse.38
•This involves writing a wrapper component that implements provides and requires interfaces then accesses the legacy system.38
•Although costly, this can be much less expensive than rewriting the legacy system.38

•The development cost of reusable components may be higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost.44
•Generic components may be less space-efficient and may have longer execution times than their specific equivalents.44

•Component management involves deciding how to classify the component so that it can be discovered, making the component available either in a repository or as a service, maintaining information about the use of the component and keeping track of different component versions.22
•A company with a reuse program may carry out some form of component certification before the component is made available for reuse.22
•Certification means that someone apart from the developer checks the quality of the component.22

•CBSE with reuse process has to find and integrate reusable components.10
•When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.10
•This involves:10
•Developing outline requirements;10
•Searching for components then modifying requirements according to available functionality.10
•Searching again to find if there are better components that meet the revised requirements.10
•Composing components to create the system.10

•Trust. You need to be able to trust the supplier of a component. At best, an untrusted component may not operate as advertised; at worst, it can breach your security.19
•Requirements. Different groups of components will satisfy different requirements.19
•Validation.19
•The component specification may not be detailed enough to allow comprehensive tests to be developed.19
•Components may have unwanted functionality. How can you test this will not interfere with your application?19

•Component validation involves developing a set of test cases for a component (or, possibly, extending test cases supplied with that component) and developing a test harness to run component tests.25
•The major problem with component validation is that the component specification may not be sufficiently detailed to allow you to develop a complete set of component tests.25
•As well as testing that a component for reuse does what you require, you may also have to check that the component does not include any malicious code or functionality that you don't need.25

•In 1996, the 1st test flight of the Ariane 5 rocket ended in disaster when the launcher went out of control 37 seconds after take off.3
•The problem was due to a reused component from a previous version of the launcher (the Inertial Navigation System) that failed because assumptions made when that component was developed did not hold for Ariane 5.3
•The functionality that failed in this component was not required in Ariane 5.3

•The process of assembling components to create a system.16
•Composition involves integrating components with each other and with the component infrastructure.16
•Normally you have to write 'glue code' to integrate components.16

•Sequential composition (1) where the composed components are executed in sequence. This involves composing the provides interfaces of each component.52
•Hierarchical composition (2) where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.52
•Additive composition (3) where the interfaces of two components are put together to create a new component. Provides and requires interfaces of integrated component is a combination of interfaces of constituent components.52

•Code that allows components to work together34
•If A and B are composed sequentially, then glue code has to call A, collect its results then call B using these results, transforming them into the format required by B.34
•Glue code may be used to resolve interface incompatibilities.34

•Parameter incompatibility where operations have the same name but are of different types.35
•Operation incompatibility where the names of operations in the composed interfaces are different.35
•Operation incompleteness where the provides interface of one component is a subset of the requires interface of another.35

•Address the problem of component incompatibility by reconciling the interfaces of the components that are composed.1
•Different types of adaptor are required depending on the type of composition.1
•An addressFinder and a mapper component may be composed through an adaptor that strips the postal code from an address and passes this to the mapper component.1
•Composition through an adaptor1
•The component postCodeStripper is the adaptor that facilitates the sequential composition of addressFinder and mapper components.1

•Address the problem of component incompatibility by reconciling the interfaces of the components that are composed.1
•Different types of adaptor are required depending on the type of composition.1
•An addressFinder and a mapper component may be composed through an adaptor that strips the postal code from an address and passes this to the mapper component.1
•Composition through an adaptor1
•The component postCodeStripper is the adaptor that facilitates the sequential composition of addressFinder and mapper components.1

•You have to rely on component documentation to decide if interfaces that are syntactically compatible are actually compatible.36
•Consider an interface for a PhotoLibrary component:36

•"This method adds a photograph to the library and associates the photograph identifier and catalogue descriptor with the photograph."41

•The Object Constraint Language (OCL) has been designed to define constraints that are associated with UML models.48
•It is based around the notion of pre and post condition specification – common to many formal methods.48

•-- The context keyword names the component to which the conditions apply47
•context addItem47

•As specified, the OCL associated with the Photo Library component states that:43
•There must not be a photograph in the library with the same identifier as the photograph to be entered;43
•The library must exist - assume that creating a library adds a single item to it;43
•Each new entry increases the size of the library by 1;43
•If you retrieve using the same identifier then you get back the photo that you added;43
•If you look up the catalogue using that identifier, then you get back the catalogue entry that you made.43

•When composing components, you may find conflicts between functional and non-functional requirements, and conflicts between the need for rapid delivery and system evolution.30
•You need to make decisions such as:30
•What composition of components is effective for delivering the functional requirements?30
•What composition of components allows for future change?30
•What will be the emergent properties of the composed system?30

•CBSE is a reuse-based approach to defining and implementing loosely coupled components into systems.None
•A component is a software unit whose functionality and dependencies are completely defined by its interfaces.None
•Components may be implemented as executable elements included in a system or as external services.None
•A component model defines a set of standards that component providers and composers should follow.None
•The key CBSE processes are CBSE for reuse and CBSE with reuse.None

•During the CBSE process, the processes of requirements engineering and system design are interleaved.37
•Component composition is the process of 'wiring' components together to create a system.37
•When composing reusable components, you normally have to write adaptors to reconcile different component interfaces.37
•When choosing compositions, you have to consider required functionality, non-functional requirements and system evolution.37