What is service-oriented architecture?

service-orientation provides us with a well-defined method for shaping software programs into units of service-oriented logic that we can legitimately refer to as services. Each such service that we deliver takes us a step closer to achieving the desired target state represented by the aforementioned strategic goals and benefits.

The four pillars of service-orientation are

• Teamwork – Cross-project teams and cooperation are required.

• Education – Team members must communicate and cooperate based on common knowledge and understanding.

• Discipline – Team members must apply their common knowledge consistently.

• Balanced Scope – The extent to which the required levels of Teamwork, Education, and Discipline need to be realized is represented by a meaningful yet manageable scope.

Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications. Each service provides a business capability, and services can also communicate with each other across platforms and languages. Developers use SOA to reuse services in different systems or combine several independent services to perform complex tasks.

For example, multiple business processes in an organization require the user authentication functionality. Instead of rewriting the authentication code for all business processes, you can create a single authentication service and reuse it for all applications. Similarly, almost all systems across a healthcare organization, such as patient management systems and electronic health record (EHR) systems, need to register patients. These systems can call a single, common service to perform the patient registration task.

Each service in an SOA embodies the code and *data* required to execute a complete, discrete business function The service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the *service* is implemented underneath, reducing the dependencies between applications.

The service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the *service* is implemented underneath, reducing the dependencies between applications.

This interface is a service contract between the service provider and service consumer. Applications behind the service interface can be written in Java, Microsoft .Net, Cobol or any other programming language, supplied as packaged software applications by a vendor (e.g., SAP), SaaS applications (e.g., Salesforce CRM), or obtained as open source applications. Service interfaces are

frequently defined using Web Service Definition Language (WSDL) which is a standard tag structure based on xml (extensible markup language).

The services are exposed using standard network protocols—such as SOAP (simple object access protocol)/HTTP or Restful HTTP (JSON/HTTP)—to send requests to read or change data. Service governance controls the lifecycle for development and at the appropriate stage the services are published in a *registry* that enables developers to quickly find them and reuse them to assemble new applications or business processes.

These services can be built from scratch but are often created by exposing functions from legacy systems of record as service interfaces.

In this way, SOA represents an important stage in the evolution of application development and integration over the last few decades. Before SOA emerged in the late 1990s, connecting an application to data or functionality housed in another system required complex point-to-point integration—integration that developers had to recreate, in part or whole, for each new development project. Exposing those functions through SOA *services allowed the developer to simply reuse the existing capability and connect through the SOA ESB architecture* (see below).

An ESB, or enterprise service bus, is an architectural pattern whereby a centralized software component performs integrations between application

Characteristics of SOA (from Book) … chapter 4

What are the benefits of service-oriented architecture?

Service-oriented architecture (SOA) has several benefits over the traditional monolithic architectures in which all processes run as a single unit. Some major benefits of SOA include the following:

Faster time to market

Developers reuse services across different business processes to save time and costs. They can assemble applications much faster with SOA than by writing code and performing integrations from scratch.

Efficient maintenance

It's easier to create, update, and debug small services than large code blocks in monolithic applications. Modifying any service in SOA does not impact the overall functionality of the business process.

Greater adaptability

SOA is more adaptable to advances in technology. You can modernize your applications efficiently and cost effectively. For example, healthcare organizations can use the functionality of older electronic health record systems in newer cloud-based applications.

P**romotes interoperability.** The use of a standardized communication protocol allows platforms to easily transmit data between clients and services regardless of the languages they're built on.

High availability. SOA facilities are available to anyone upon request.

Increased reliability. SOA generates more reliable applications because it is easier to debug small services than large code.

Scalable. SOA allows services to run on different servers, thus increasing scalability. In addition, using a standard communication protocol allows organizations to reduce the level of interaction between clients and services. Lowering the level of interaction allows applications to be scaled without adding extra pressure.

Principles of Service Orientation: Pg 38 (Book by Thomas Erl)