Differece Between SA/SD & OMT

| ****Aspect**** | ****Structured Analysis / Structured Design (SA/SD)**** | ****Object Modeling Technique (OMT)**** |
| --- | --- | --- |
| **Model Priority** | Functional model is most important; dynamic model next; object model least important | Object model is most important; then dynamic model; functional model is least important |
| **System Organization** | Organized around **procedures (functions)** | Organized around **real-world or conceptual objects** |
| **Handling Requirement Changes** | Changes in function can break the design and require major restructuring | Changes in function are easier to handle by modifying operations; object structure remains stable |
| **System Boundary** | Has a clearly defined system boundary; difficult to extend across new boundaries | Easier to extend due to flexible object structure |
| **Process Decomposition** | Decomposition into sub-processes is **arbitrary** and can vary between developers | Decomposition is based on **problem domain objects**; more consistent among developers |
| **Reusability** | Low reusability due to inconsistent decomposition | High reusability due to consistent object identification |
| **Database Integration** | Awkward to integrate with databases; procedural code and database are separate | Easily integrates databases with code using a **uniform object paradigm** |
| **Real-world Modeling** | Less natural mapping to real-world entities | Closer to how people view and describe real-world systems |
| **Adaptability** | Less adaptable to changes in user perspective or domain | More adaptable and flexible to evolving requirements |
| **Code Maintainability** | Difficult to maintain as changes may ripple through procedures | Easier to maintain as object interfaces remain stable |
| **Scalability** | Not well-suited for large or complex systems | Better suited for large, complex, and evolving systems |
| **Modularity** | Function-based modules that are tightly coupled | Object-based modules that promote loose coupling |
| **Encapsulation** | Little to no support for encapsulation | Strong support for **encapsulation** of data and behavior |
| **Inheritance/Polymorphism** | Not supported | Fully supports **inheritance** and **polymorphism** |
| **Common Use** | Traditionally used in structured programming (e.g., C) | Commonly used in object-oriented programming (e.g., Java, C++) |
| **Design Consistency** | Design may vary significantly between teams | Encourages consistent designs across similar domains |

Difference Between JSD & OMT

| ****Aspect**** | ****JSD (Jackson System Development)**** | ****OMT (Object Modeling Technique)**** |
| --- | --- | --- |
| **Object-Orientation** | Begins with real-world consideration and is object-oriented in that sense, but identifies few entities with little structure. | Supports a rich mixture of data structure and relationships. |
| **Complexity** | Considered complex and difficult to comprehend, especially due to reliance on pseudocode. | Uses graphical models which are easier to understand. |
| **Best Use Case** | Designed for difficult real-time problems, may produce superior design in such cases. | More suitable for simpler and common problems. |
| **Ease of Understanding** | More obscure and harder to understand than data-flow and object-oriented approaches. | Easier to understand due to use of diagrams and graphical modeling. |
| **Actions vs Attributes** | Emphasizes **actions** over attributes. Example: A clerk allocates product to an order. | Emphasizes **attributes** more than actions. |
| **Terminology** | Refers to relationships as **actions** (e.g., “allocates”). | Refers to relationships as **associations**. |
| **Attributes Handling** | Attributes are considered confusing and often avoided. | Attributes play a prominent role in object modeling. |
| **Modeling Focus** | Actions are central, attributes are less prominent. | Attributes can diminish the importance of operations in object models. |