**Code Layout:**

1. **Indentation and Formatting:**
   * Follow a consistent indentation and formatting style throughout your codebase. Use tools like Prettier or the built-in formatting options in your IDE to maintain consistency.
2. **Comments:**
   * Include meaningful comments to explain complex sections of your code or to provide context for future developers. However, strive to write code that is self-explanatory, and use comments sparingly.
3. **Descriptive Naming:**
   * Use clear and descriptive names for variables, classes, methods, and other elements. This enhances readability and understanding.
4. **Organize Code Logically:**
   * Organize your code in a logical manner. Group related functionality together within classes and use meaningful class and method names.
5. **Separation of Concerns:**
   * Adhere to the principle of separation of concerns. Each class or method should have a single responsibility, making the code easier to understand and maintain.

**Reliability:**

1. **Exception Handling:**
   * Implement proper exception handling to gracefully manage errors and prevent unexpected system behavior. Use try-catch blocks judiciously.
2. **Bulkification:**
   * Write code that supports bulk operations to avoid hitting governor limits. Design your code to efficiently handle large volumes of data.
3. **Governor Limits:**
   * Be aware of Salesforce governor limits and design your code to stay within these limits. Use tools like limits methods to monitor your code's resource consumption.
4. **Testing:**
   * Implement thorough unit tests and leverage Salesforce's testing framework. Test not only positive scenarios but also negative and bulk test cases.
5. **Logging and Debugging:**
   * Use debug logs and system.debug statements for effective debugging. Ensure that your code includes appropriate logging to aid in identifying issues.

**Reusability:**

1. **Modular Design:**
   * Design your code in a modular way, creating reusable components or classes. This promotes code reuse and makes it easier to maintain.
2. **Apex Classes and Methods:**
   * Break down large functionality into smaller, reusable methods and classes. Encapsulate logic in methods that can be easily invoked from other parts of your code.
3. **Custom Metadata Types:**
   * Leverage Custom Metadata Types to configure and store data that can be reused across your application. This allows for more dynamic and configurable solutions.
4. **Design Patterns:**
   * Apply design patterns such as Singleton, Factory, or Strategy when appropriate. These patterns can enhance code organization and reusability.
5. **Use of Interfaces:**
   * Define interfaces to enforce a contract for classes implementing common functionality. This enables a consistent and extensible approach.
6. **Managed Packages:**
   * If applicable, consider creating managed packages for reusable components. This allows for easy distribution and installation in different Salesforce environments.