1. **Seguridad**: Se refiere a la capacidad del software para proteger datos y mantener la integridad del sistema frente a posibles amenazas o ataques. Ejemplo: Implementación de cifrado de datos utilizando AES (Advanced Encryption Standard) para proteger información confidencial en una aplicación de banca en línea.

2. **Fiabilidad**: Indica la capacidad del software para mantener un rendimiento constante y predecible bajo diferentes condiciones. Ejemplo: Uso de pruebas de regresión automatizadas para garantizar que las nuevas actualizaciones del software no introduzcan errores o fallos en funcionalidades previamente estables.

3. **Resiliencia**: Hace referencia a la capacidad del software para recuperarse rápidamente de fallos o interrupciones sin comprometer su funcionamiento. Ejemplo: Implementación de sistemas de copia de seguridad y restauración automatizados para garantizar la disponibilidad de datos críticos en caso de fallo del servidor principal.

4. **Robustez**: Se relaciona con la capacidad del software para manejar situaciones inesperadas o entradas incorrectas sin fallar. Ejemplo: Validación de formularios en una aplicación web para evitar que los usuarios introduzcan datos inválidos que podrían provocar errores en el sistema.

5. **Comprensibilidad**: Se refiere a la facilidad con la que los usuarios pueden entender y utilizar el software. Ejemplo: Diseño de una interfaz de usuario intuitiva con instrucciones claras y coherentes para guiar a los usuarios a través de las funcionalidades de una aplicación móvil.

6. **Capacidad de prueba**: Indica la facilidad con la que se puede probar el software para detectar errores y evaluar su calidad. Ejemplo: Uso de frameworks de pruebas como JUnit en Java o Pytest en Python para escribir y ejecutar pruebas automatizadas de unidades de código.

7. **Adaptabilidad**: Hace referencia a la capacidad del software para ajustarse y funcionar correctamente en diferentes entornos o situaciones cambiantes. Ejemplo: Desarrollo de una aplicación web que se adapte automáticamente a diferentes tamaños de pantalla para proporcionar una experiencia de usuario consistente en dispositivos móviles y de escritorio.

8. **Modularidad**: Se refiere a la capacidad del software para dividirse en módulos independientes y cohesivos que puedan ser desarrollados, probados y mantenidos por separado. Ejemplo: Diseño de una arquitectura de microservicios en la que cada función del sistema se implementa como un servicio independiente que se puede escalar y mantener de forma independiente.

9. **Complejidad**: Se refiere al nivel de dificultad del software en términos de su diseño, implementación y mantenimiento. Ejemplo: Evaluación de la complejidad ciclomática de una función mediante herramientas como

SonarQube para identificar áreas del código que pueden requerir refactorización para mejorar su legibilidad y mantenibilidad.

10. **Portabilidad**: Indica la capacidad del software para ejecutarse en diferentes plataformas y sistemas operativos sin necesidad de modificaciones significativas. Ejemplo: Desarrollo de una aplicación multiplataforma utilizando tecnologías como HTML5, CSS y JavaScript que se pueden ejecutar en diferentes navegadores web y dispositivos móviles.

11. **Usabilidad**: Se refiere a la facilidad con la que los usuarios pueden interactuar con el software para lograr sus objetivos de manera eficiente y satisfactoria. Ejemplo: Realización de pruebas de usabilidad con usuarios reales para identificar y corregir problemas de navegación y diseño en una aplicación de comercio electrónico.

12. **Reutilizabilidad**: Indica la capacidad del software para reutilizar componentes o módulos existentes en diferentes partes del sistema o en proyectos futuros. Ejemplo: Creación de una biblioteca de funciones comunes que puedan ser compartidas y reutilizadas en diferentes proyectos de desarrollo de software.

13. **Eficiencia**: Se refiere a la capacidad del software para realizar tareas utilizando la menor cantidad de recursos posible, como memoria, CPU y tiempo de ejecución. Ejemplo: Optimización de algoritmos y estructuras de datos para reducir el tiempo de procesamiento en una aplicación de análisis de datos en tiempo real.

14. **Capacidad de aprendizaje**: Indica la facilidad con la que los usuarios pueden aprender a utilizar el software sin necesidad de una formación extensa. Ejemplo: Inclusión de tutoriales interactivos y ayuda contextual dentro de una aplicación de software para guiar a los usuarios a través de las funcionalidades clave y mejorar su experiencia de aprendizaje.