1)
Os princípios SOLID ajudam no desenvolvimento de software de maneira mais organizada, flexível e eficiente. Esses princípios são:

1 - Single Responsibility Principle(SRP): cada classe deve ter apenas uma responsabilidade, ou seja, seu foco é executar uma única função com o melhor desempenho possível.

2 - Open/Closed Principle(OCP): nesse princípio aa classes estão abertas a extensão, porém fechadas para a modificação. Sendo assim podem ser adicionados novos comportamentos sem alterar o código existente. Em .NET isso é feito usando herança e interface.

3 - Liskov Substitution Principle(LSP): esse princípio diz que os objetos de uma classe derivada podem substituir objetos da classe base sem que o programa pare de funcionar.

4 - Interface Segregation Principle(ISP): feito para evitar interfaces grandes e genéricas, focando em várias interfaces pequenas e mais específicas.

5 - Dependency Inversion Principle(DIP): nesse princípio o código deve depender de abstrações e não de implementações concretas.

2)
Os principais padrões de arquitetura de software usados em aplicações. NET ajudam a organizar o código deixando mais legível. Dois exemplos são:

1 - Model-View-Controller(MVC) : muito popular em aplicações web, sendo ótimo para a organização, manutenção e reutilização. Divide a aplicação em 3 partes, a primeira é a Model, que representa os dados e a lógica, onde as regras são definidas. A segunda parte é a View, onde os usuários veem e interagem(interface) e a terceira o Controller, que faz a ponte entre Model e View, recebe as entradas do usuário e processa chamando métodos do Model e atualizando a View.

2 - Clean Architecture: esse padrão é sobre estruturar o código mantendo limpo e fácil de entender, o tornando flexível e independente de frameworks, facilitando mudanças. O padrão propõe dividir a aplicação em várias camadas de responsabilidade específicas sendo elas camadas mais intensas, onde está contido as regras e entidades do domínio, camadas intermediárias, possuem os casos de uso e serviços da aplicação. E por último camadas externas que lidam com a interface do usuário, o banco de dados e frameworks externos.

3)
É importante pois deixa o código mais organizado e legível, além de que quando cada parte do sistema tem uma função clara, fica mais fácil realizar qualquer tipo de mudança sem quebrar o código. Uma das abordagens é o MVC que já foi citado na questão 2. Outra abordagem é a Layered Architecture, que também divide a aplicação em camadas, sendo elas a cada de apresentação que é a interface, a camada de aplicação que refere a lógica e

coordena as operações, a camada de domínio que contém a lógica de negócios principal e por último a camada de infraestrutura que lida com detalhes técnicos, como o acesso ao banco de dados.