1. Os pilares da orientação a objetos discutidos nessa disciplina são: Herança, Abstração e Encapsulamento. Herança se trata da possibilidade de reaproveitamento de código, onde classes mais genéricas implementam características e comportamentos gerais e classes mais específicas podem herdá-los e estendê-los. Abstração se trata da capacidade de isolar determinadas informações de uma classe utilizando modificadores de visibilidade (definindo o que é público, protegido e privado) protegendo as informações de serem acessadas e modificadas diretamente. Por fim o Encapsulamento se trata da capacidade de uma referência para uma classe genérica poder referenciar instâncias dessa classe genéricas ou de qualquer outra classe filha (que herda dessa classe genérica).

Os pilares da orientação a objetos discutidos nessa disciplina são: Herança, Abstração, **Polimorfismo** e Encapsulamento. Herança se trata da possibilidade de reaproveitamento de código, onde classes mais genéricas implementam características e comportamentos gerais e classes mais específicas podem herdá-los e estendê-los. Abstração **diz respeito a capacidade de criar uma classe de dados apenas definindo atributos e métodos que definam o que aquela entidade é e o que ela pode fazer, sem descrever em detalhes como essa entidade realiza (implementa) esse comportamento**. **Encapsulamento** se trata da capacidade de isolar determinadas informações de uma classe utilizando modificadores de visibilidade (definindo o que é público, protegido e privado) protegendo as informações de serem acessadas e modificadas diretamente. Por fim o **Polimorfismo** se trata da capacidade de uma referência para uma classe genérica poder referenciar instâncias dessa classe genéricas ou de qualquer outra classe filha (que herda dessa classe genérica).

2. A arquitetura de software é uma área de conhecimento complementar a engenharia de software. Ela trata de todos os aspectos sobre o processo de desenvolvimento, estudando, desenvolvendo e aplicando ferramentas para ajudar no ganho de produtividade e qualidade no processo de desenvolvimento de software. Já a engenharia de software é focada nos projetos dos sistemas de software e a maneira que os componentes desses sistemas se comunicam entre si. Enquanto a engenharia de software foca no processo, a arquitetura de software foca no projeto.

A arquitetura de software é uma área de conhecimento complementar a engenharia de software. Ela **é focada nos projetos dos sistemas de software e a maneira que os componentes desses sistemas de comunicam entre si**. Já a engenharia de software **trata de todos os aspectos sobre o processo de desenvolvimento, estudando, desenvolvendo e aplicando ferramentas para ajudar no ganho de produtividade e qualidade no processo de desenvolvimento de software**. Enquanto a engenharia de software foca no processo, a arquitetura de software foca no projeto.

3. O escopo da arquitetura de software abrange vários aspectos do desenvolvimento de sistemas de software. Envolve decidir quais padrões arquiteturais (ou adaptação deles) devem ser utilizados; modelar diagramas para representar uma visão geral do sistema; especificar quais os requisitos de desempenho, segurança, escalabilidade, entre outros e quais e como os componentes devem ser implementados para

garanti-los; definir as responsabilidades e funcionalidades de cada componente do ponto de vista funcional; considerar o ciclo de vida do sistema como um todo, planejando as suas possíveis correções e evoluções; estudar e consolidar padrões e boas práticas para guiar o processo de desenvolvimento; analisar pontos críticos de falha e decidir o que se pode ou não abrir mão para atingir os objetivos do sistema; manter uma documentação detalhada sobre o sistema; e realizar testes automatizados para garantir a qualidade do sistema de software.

O escopo da arquitetura de software abrange vários aspectos do desenvolvimento de sistemas de software. Envolve decidir quais padrões arquiteturais (ou adaptação deles) devem ser utilizados; modelar diagramas para representar uma visão geral do sistema; especificar quais os requisitos de desempenho, segurança, escalabilidade, entre outros e quais ~~e como~~ **os componentes** ~~devem ser implementados~~ **devem compor o sistema** para garanti-los; definir as responsabilidades e funcionalidades de cada componente do ponto de vista funcional; considerar o ciclo de vida do sistema como um todo, planejando as suas possíveis correções e evoluções; estudar e consolidar padrões e boas práticas para guiar o processo de desenvolvimento; analisar pontos críticos de falha e decidir o que se pode ou não abrir mão para atingir os objetivos do sistema; manter uma documentação detalhada sobre o sistema; e **validar se a arquitetura construída atende às necessidades do projeto**. ~~realizar testes automatizados para garantir a qualidade do sistema de software~~.

4. O papel do arquiteto de software é mais gerencial e mais orientado ao código, enquanto o papel do engenheiro de software é mais consultivo e orientativo.

O papel do **engenheiro** de software é mais gerencial e mais orientado ao código, enquanto o papel do **arquiteto** de software é mais consultivo e orientativo.

5. MVC é um padrão arquitetural focado na criação de aplicativos móveis. Ele estende a arquitetura cliente-servidor incluindo componentes que tornam a navegação mais dinâmica. Os componentes são: (M) Modelos, que representam as entidades do sistema e implementam rotinas para retenção dos dados manipulados; (V) Visualizações, que representam como as páginas serão apresentadas para o usuário podendo ter lacunas a serem preenchidas pelo sistema para dar mais dinâmica; e (C) Comandos, que são uma lista de funcionalidades que podem ser executadas pelo sistema.

MVC é um padrão arquitetural focado na criação **de plataformas *web*** ~~aplicativos móveis~~. Ele estende a arquitetura cliente-servidor incluindo componentes que tornam a navegação mais dinâmica. Os componentes são: (M) Modelos, que representam as entidades do sistema e implementam rotinas para retenção dos dados manipulados; (V) Visualizações, que representam como as páginas serão apresentadas para o usuário podendo ter lacunas a serem preenchidas pelo sistema para dar mais dinâmica; e (C) **Controladores**, que **implementam as rotinas para trazer os dados dos modelos para as visualizações.**

6. SPA é uma extensão do MVC, focada em aplicativos móveis. Com esse padrão arquitetural, as páginas ganham mais desempenho por evitar o carregamento completo todas as vezes, possibilitando o carregamento só dos componentes que sofreram modificações. Para isso, as páginas são divididas em componentes que são responsáveis pelo fluxo de dados entre os *templates* (a parte visual das páginas) e as regras de negócio (código que implementa a comunicação com outros elementos da arquitetura). Além disso, esses componentes podem ser organizados em módulos de funcionalidades e podem ser importados em outros projetos. Por fim, existem os serviços, que tem a responsabilidade de fazer a comunicação com sistemas externos (como APIs e *backends*). Esses serviços podem ser injetados nos componentes para permitir a interação com esses dados externos.

SPA é ~~uma extensão do MVC, focada em aplicativos móveis. Com esse~~ **um** padrão arquitetural **onde** as páginas ganham mais desempenho por evitar o carregamento completo todas as vezes, possibilitando o carregamento só dos componentes que sofreram modificações. Para isso, as páginas são divididas em componentes que são responsáveis pelo fluxo de dados entre os *templates* (a parte visual das páginas) e as regras de negócio (código que implementa a comunicação com outros elementos da arquitetura). Além disso, esses componentes podem ser organizados em módulos de funcionalidades e podem ser importados em outros projetos. Por fim, existem os serviços, que tem a responsabilidade de fazer a comunicação com sistemas externos (como APIs e *backends*). Esses serviços podem ser injetados nos componentes para permitir a interação com esses dados externos.

7. O padrão MOM estabelece um mecanismo assíncrono de troca de mensagens entre aplicações. Esse modelo é diferente do modelo cliente-servidor, onde uma aplicação cliente faz uma requisição para uma aplicação de servidor e espera por uma resposta. Nesse modelo todas as aplicações podem se comportar tanto como servidores (nesse caso chamadas de inscritas) ou como clientes (nesse caso chamadas de publicadoras). Para que essa operação funcione corretamente é necessária a criação de uma série de tópicos que serão gerenciados por um componente central chamado de *broker*. As aplicações publicadoras vão escrever em um determinado tópicos e o *broker* vai enviar essa mensagem para todas as aplicações inscritas nesse tópico. Algumas das implementações dessa tecnologia são o RabbitMQ, o MQTT e o Spring Boot.

O padrão MOM estabelece um mecanismo assíncrono de troca de mensagens entre aplicações. Esse modelo é diferente do modelo cliente-servidor, onde uma aplicação cliente faz uma requisição para uma aplicação de servidor e espera por uma resposta. Nesse modelo todas as aplicações podem se comportar tanto como servidores (nesse caso chamadas de **publicadoras**) ou como clientes (nesse caso chamadas de **inscritas**). Para que essa operação funcione corretamente é necessária a criação de uma série de tópicos que serão gerenciados por um componente central chamado de *broker*. As aplicações publicadoras vão escrever em um determinado tópico e o *broker* vai enviar essa mensagem para todas as aplicações inscritas nesse tópico. Algumas das implementações dessa tecnologia são o RabbitMQ **e** o MQTT ~~e o Spring Boot~~.

8. O SOA é um padrão arquitetura que surgiu com a necessidade da padronização da comunicação entre um grande volume de aplicações. Esse padrão estende o modelo clássico cliente-servidor estabelecendo um mecanismo flexível (que pode ser usado por diferentes tipos de aplicações) mas ao mesmo tempo padronizado (por meio de do protocolo REST). Nesse tipo de padrão arquitetural, as aplicações podem oferecer seus serviços por meio de APIs que podem ser consumidas por qualquer outra aplicação desde que use requisições REST. Nesse modelo a documentação dessas APIs não é mais necessária já que a comunicação foi padronizada.

O SOA é um padrão arquitetura que surgiu com a necessidade da padronização da comunicação entre um grande volume de aplicações. Esse padrão estende o modelo clássico cliente-servidor estabelecendo um mecanismo flexível (que pode ser usado por diferentes tipos de aplicações) mas ao mesmo tempo padronizado (por meio de do protocolo REST). Nesse tipo de padrão arquitetural, as aplicações podem oferecer seus serviços por meio de APIs que podem ser consumidas por qualquer outra aplicação desde que use requisições REST. Nesse modelo a documentação dessas APIs **é fundamental para estabelecer que tipo de serviço está sendo oferecido e que tipo de dados os *endpoints* dessas APIs esperam receber e o que eles retornam como resposta.** ~~não é mais necessária já que a comunicação foi padronizada~~.

9. Os princípios que guiam o projeto de bons sistemas de software são baseados em quatro conceitos base: integridade conceitual, ocultamento de informações, coesão e acoplamento. De maneira geral um software precisa se manter íntegro conceitualmente, ou seja, fazer o que tem que ser feito de maneira padronizada e coerente ao longo do tempo. Além disso é importante o uso das abstrações corretas, ocultando toda informação que não for fundamental para o entendimento e uso dos componentes de um sistema e utilizando interfaces públicas para sua utilização. Um componente de software precisa ainda ser coeso, ou seja, evitar a dependência forte e/ou nebulosa com outros componentes. Por fim, um componente de software precisa ter um baixo acoplamento, ou seja, implementar apenas uma única funcionalidade e todos os demais métodos e atributos desse componente atuem para implementar essa única funcionalidade. Os princípios SOLID (responsabilidade única, fechamento para extensão e abertura para modificações, substituição de Liskov, segregação de interfaces e inversão de dependências) e de Demeter (que estabelece que um determinado método só pode invocar os métodos de outras classes, de objetos passados como parâmetros, de objetos instanciados pelo próprio método ou de atributos da classe desse método) definem regras que ajudam a dar um suporte mais prático para cada um desses quatro conceitos fundamentais.

Os princípios que guiam o projeto de bons sistemas de software são baseados em quatro conceitos base: integridade conceitual, ocultamento de informações, coesão e acoplamento. De maneira geral um software precisa se manter íntegro conceitualmente, ou seja, fazer o que tem que ser feito de maneira padronizada e coerente ao longo do tempo. Além disso é importe o uso das abstrações corretas, ocultando toda informação que não for fundamental para o entendimento e uso dos componentes de um sistema e utilizando interfaces públicas para sua utilização. Um componente de software precisa ainda ser ter **um baixo**

acoplamento, ou seja, evitar a dependência forte e/ou nebulosa com outros componentes. Por fim, um componente de software precisa **ser coeso**, ou seja, implementar apenas uma única funcionalidade e todos os demais métodos e atributos desse componente devem atuar para implementar essa única funcionalidade. Os princípios SOLID (responsabilidade única, fechamento para **modificação** e abertura para **extensão**, substituição de Liskov, segregação de interfaces e inversão de dependências) e de Demeter (que estabelece que um determinado método só pode invocar os métodos **da mesma classe**, de objetos passados como parâmetros, de objetos instanciados pelo próprio método ou de atributos da classe desse método) definem regras que ajudam a dar um suporte mais prático para cada um desses quatro conceitos fundamentais.