

<p>Признаки сложного ПО:</p> <ol style="list-style-type: none"> 1. Комплексность (сложность); 2. Длительный жизненный цикл; 3. Работа в команде. 	<p>Проблемы при разработке сложного ПО:</p> <ol style="list-style-type: none"> 1. Нечеткие, меняющиеся требования; 2. Большое количество понятий предметной области; 3. Необходимость внесения изменений в код ПО; 4. Большое кол-во разработчиков. 	<p>Проблемы указателей и ссылок в C++:</p> <ol style="list-style-type: none"> 1. Может быть NULL; 2. Можно забыть инициализировать; 3. Может указывать на уже несуществующий объект; <p>Можно забыть выделить/освободить память;</p>
<p>Идиомы RAII: Использование механизма контроля зоны видимости объекта для управления ресурсами, которые он содержит.</p>	<p>Объектно-ориентированное программирование – парадигма программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархии абстрагирования.</p>	<p>Парадигма программирования – набор правил, концепций и абстракций, определяющий стиль программирования (подход к программированию). Объект – сущность, обладающая состоянием, поведением, свойствами и операции над ними (= экземпляр класса). Класс – шаблон, по которому создается объект, т.е. элемент, описывающий тип данных и его реализацию.</p>
<p>Свойства: Инкапсуляция – позволяет объединить данные и методы, работающие с ними, в классе (≠ сокрытие). Наследование – описывает новый класс на основе существующего с частичной или полной заимствованной функциональностью. Полиморфизм – использование объектов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.</p>	<p>Абстрагирование – выделение значимой информации и исключение незначимой. Сокрытие – принцип проектирования, заключающийся в разграничении доступа различных частей программы к внутренним компонентам друг друга ()</p>	<p>Состояние объекта – совокупность значений членов класса и состояний базовых классов. Изменение состояния объектов – изменение значения любого члена класса или состояния базового класса. Некорректное/недопустимое состояние объекта – недопустимая комбинация значений (состояний) членов класса и/или состояний базовых классов</p>
<p>Инвариант класса – утверждение, определяющее непротиворечивое состояние объектов этого класса. Нарушение инварианта класса – объект класса имеет некорректное состояние.</p> <ul style="list-style-type: none"> – Приводит к нарушению целостности кода, который использует данный класс – Приводит к недопустимому состоянию классов, которые его используют <p>Способы контроля инварианта класса – проверка инварианта в случаях, когда объект мог изменить свое состояние.</p>	<p>Вариативность состояния класса – способность объекта заданного класса изменять свое состояние без нарушения инварианта при выполнении стандартных операций над объектами.</p> <ul style="list-style-type: none"> – Неизменяемый – не изменяет свое состояние после создания – Копируемый – при копировании (создании, передаче в качестве параметра, присваивании) не нарушает своего состояния. – Некопируемый - объект может быть только переносим без нарушения своего состояния 	<p>Шаблоны проектирования – повторяемая структура взаимодействия объектов программы, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Роль ШП – универсальный язык для обозначения приемов проектирования ПО:</p> <ul style="list-style-type: none"> – Упрощает коммуникацию, обучение – Классифицирует приемы проектирования, что упрощает их повторное использование.
<p>Виды ШП:</p> <ul style="list-style-type: none"> – Низкоуровневые – идиомы программирования, связаны с устойчивыми решениями. – Шаблоны проектирования – уровень взаимодействия объектов программы. – Архитектурные шаблоны – охватывают архитектуру ПО. 	<p>ШП делегирование: Объект внешне выражает некоторое поведение, но на деле передает ответственность за выполнение этого поведения связанному объекту. Позволяет изменить поведение конкретного экземпляра вместо создания нового класса. Затрудняет оптимизацию по скорости в пользу чистоты абстракции.</p>	<p>Шаблон делегирования является фундаментальной абстракцией, на основе которой реализованы другие шаблоны.</p> 
<p>ШП функциональный проект: Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет ее с минимумом побочных эффектов на другие части программы. Функционально разработанные модули имеют низкое зацепление. Определение чистоты: если описание модуля включает связи «и», «или», тогда проект имеет более чем одно предназначение и возможно будет иметь побочные эффекты.</p>	<p>ШП неизменяемый интерфейс: Определяет интерфейсный класс, в котором определены методы, не изменяющие его состояние. Четко передает намерения о неизменяемости класса, но для библиотечных классов необходимо предусматривать заранее. (Альтернатива – неизменяемая обертка)</p>	<p>ШП интерфейс: Обеспечивает простой или более программно-специфический способ доступа к другим классам. Интерфейс может содержать набор объектов и обеспечивать простую высокоуровневую функциональность для программиста, использоваться в качестве «клея» между двумя различными API и т.д.</p>
<p>ШП контейнер атрибутов/свойств: Обеспечивает возможность динамически расширять атрибутивный состав класса – достигается путем добавления доп. атрибутов самому объекту в ассоциативный контейнер, вместо расширения класса новыми атрибутами. Плюсы: позволяет классу легко и быстро изменяться без изменения кода класса. Минус: теряется строгая типизация.</p>	<p>ШП абстрактная фабрика: Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. Реализуется созданием абстрактного класса, который представляет собой интерфейс для создания компонентов системы. Затем пишутся классы, реализующие этот интерфейс. Плюсы: изолирует конкретные классы, упрощает замену семейств продуктов, гарантирует сочетаемость продуктов. Минус: сложно добавить поддержку нового вида продуктов.</p>	<p>ШП строитель: Порождающий шаблон проектирования предоставляет способ создания составного объекта. Отделяет конструирование сложного объекта от его представления так, что в результате одного и того же процесса конструирования могут получаться разные представления (пример – переводчик). Плюсы: позволяет изменять внутреннее представление продукта, изолирует код, реализующий конструирование и представление, дает более тонкий контроль над процессом конструирования. Огранич.: алгоритм создания слож. объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой, процесс конструирования должен обеспечивать различные представления конструируемого объекта.</p>

<p>ШП одиночка: Задача: нужно гарантировать, что код работает с общим (глобальным) объектом. Порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру. Плюс: контролируемый доступ к единственному экземпляру. Минусы: глобальные объекты могут быть вредны для объектного программирования, в некоторых случаях приводят к созданию немасштабируемого проекта, усложняют написание модульных тестов и следование TDD.</p>	<p>ШП мост: Задача: нужно реализовать журналирование в различные места: файл, консоль, удалённый компьютер; для каждого варианта логгера нужны варианты реализации: однопоточковая и многопоточковая. Выделяется дополнительная иерархия, в которой реализовано представление изначальной иерархии. Позволяет разделять абстракцию и реализацию так, чтобы они могли изменяться независимо. Плюс: упрощает модифицирование кода реализации. Минусы: усложняет реализацию; ухудшает производительность.</p>	<p>ШП компоновщик: Задача: необходимо реализовать текстовый редактор, в котором элементы образуют иерархию единообразных объектов (пример: MS Word, HTML). Объединяет объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково. Плюсы: программу легко добавлять новые примитивные или составные объекты; код имеет простую структуру — примитивные и составные объекты обрабатываются одинаковым образом; позволяет легко обойти все узлы древовидной структуры. Минус: неудобно осуществить запрет на добавление объектов определенных типов.</p>
<p>ШП фасад: Задачи: обеспечить унифицированный интерфейс с набором разрозненных реализаций или интерфейсов, например, с подсистемой; нежелательно высокое связывание с этой подсистемой; реализация подсистемы может измениться. Определить одну точку взаимодействия с подсистемой — фасадный объект, обеспечивающий общий интерфейс с подсистемой, и возложить на него обязанность по взаимодействию с её компонентами. Особенности: <ul style="list-style-type: none"> Фасад — это внешний объект, обеспечивающий единственную точку входа для служб подсистемы. Реализация других компонентов подсистемы закрыта и не видна внешним компонентам. Фасад определяет новый интерфейс. </p>	<p>Принципы SOLID: The Single Responsibility Principle (SRP) — Принцип единственной ответственности: The Open Closed Principle (OCP) — Принцип открытости/закрытости: The Liskov Substitution Principle (LSP) — Принцип подстановки Барбары Лисков: The Interface Segregation Principle (ISP) — Принцип разделения интерфейса: The Dependency Inversion Principle (DIP) — Принцип инверсии зависимостей.</p>	<p>Принцип единственной ответственности (SRP): Каждый класс выполняет лишь одну задачу ШП Функциональный дизайн Антипаттерн «Божественный объект» Способы достижимости: <ul style="list-style-type: none"> использование приёма рефакторинга «выделение класса»; шаблон «фасад»; интерфейсы. Примеры: <ul style="list-style-type: none"> Система управления поливом и клапан; Ведение боя и правила боя. </p>
<p>Принцип открытости/закрытости (OCP): «Программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения». Специфика: <ul style="list-style-type: none"> близок к SRP; принцип позволяет избежать пересмотра связанного кода, модульных тестов, текстов самодокументирования и других артефактов ПО; добиться снижения трудозатрат. Способы достижимости: <ul style="list-style-type: none"> использование приёма рефакторинга «выделение класса»; шаблон «фасад»; интерфейсы. </p>	<p>Принцип подстановки Барбары Лисков (LSP): «Объекты в программе должны быть заменяемыми на экземпляры их подтипов без нарушения выполнения программы». Наследующий класс должен дополнять, а не изменять базовый. Следствия: <ul style="list-style-type: none"> нельзя усиливать предусловия и ослаблять постусловия методов производных классов; нельзя создавать новых мутаторов свойств, не предусмотренных базовым классом; производные классы не должны бросать исключения, не предусмотренные в базовом классе. </p>	<p>Принцип разделения интерфейса (ISP): «Много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения» Особенности: <ul style="list-style-type: none"> улучшает адаптивность кода; существенно упрощает рефакторинг. Примеры: <ul style="list-style-type: none"> Система управления поливом и клапан; Ведение боя и правила боя. </p>
<p>Принцип инверсии зависимостей (DIP): Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций. Примеры: <ul style="list-style-type: none"> Система управления поливом и клапан; Ведение боя и правила боя. </p>	<p>Антипаттерны: Паттерн проектирования (шаблон проектирования) Спагетти-код Божественный объект — Концентрация слишком большого количества функций в одной части системы (классе). Базовый класс-утилит — Наследование функциональности из класса-утилиты вместо делегирования к нему. Вызов предка — Для реализации прикладной функциональности методу класса-потомка требуется в обязательном порядке вызывать те же методы класса-предка. Одиночество — Неуместное использование паттерна одиночка. Френд-зона — Неуместное использование дружественных классов и дружественных функций в языке C++. Каша из интерфейсов — Объединение нескольких интерфейсов, разделенных согласно принципу изоляции интерфейсов (ISP), в один.</p>	<p>Лазанья-код — Чрезмерное связывание между собой уровней абстракции, приводящее к невозможности изменения одного уровня без изменения остальных. Радиоли-код — Объекты настолько «склеены» между собой, что практически не допускают рефакторинга. Копи-паст код — Копирование (и лёгкая модификация) существующего кода вместо создания общих решений. Золотой молоток — Сильная уверенность в том, что любимое решение универсально применимо. Фактор невероятности — Предположение о невозможности того, что сработает известная ошибка. Дым и зеркала — Демонстрация того, как будут выглядеть ненаписанные функции.</p>

<p>Непрерывная интеграция (CI) — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счёт наиболее раннего обнаружения и устранения ошибок и противоречий, но основным преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления.</p>	<p>Непрерывное развертывание (CD) — это подход к разработке программного обеспечения, при котором все изменения, вносимые в исходный код, автоматически развертываются в продакшн, без явной отмашки от разработчика. Как правило, задача разработчика сводится к проверке запроса на включение от коллеги и к информированию команды о результатах всех важных событий. Непрерывное развертывание требует, чтобы в команде существовала отлаженная культура мониторинга, все умели держать руку на пульсе и быстро восстанавливать систему.</p>	<p>Непрерывная доставка (CDE) — это подход к разработке программного обеспечения, при котором программное обеспечение производится короткими итерациями, гарантируя, что ПО является стабильным и может быть передано в эксплуатацию в любое время. Передача его происходит вручную. Подход позволяет уменьшить стоимость, время и риски внесения изменений путём более частых мелких обновлений в продакшн-приложение.</p>
<p>Enterprise (классическая разработка): Разработка разделяется на итерации; Каждая итерация разделяется на этапы; Каждый этап выполняется своим набором ролей; Каждый этап не может начаться без наличия артефакта (документа или продукта); Каждый этап заканчивается формированием артефакта; Артефакты используются для верификации; Такая схема позволяет достичь хорошего качества при низкой лояльности заказчика и среднем уровне квалификации сотрудников; Проблема в больших накладных расходах на поддержку жизненного цикла, а значит, в длительности итераций. В итоге, стоимость разработки значительна.</p>	<p>Agile (гибкая разработка): Переворачивает разработку в методологиях Enterprise, исключая документацию как необходимый связующий элемент верификации и связи этапов жизненного цикла. Позволяет радикально быстро реагировать на изменение требований. Более того, изменение требований принимается естественным и включено в процесс разработки, который становится более итерационным. Верификация выполняется с использованием продукта в конце каждой итерации. В результате может быть достигнута высокая производительность при автоматизации сложных, и особенно, плохо формализуемых предметных областей.</p>	<p>Domain Driven Design (DDD) — Набор принципов и схем, направленных на создание оптимальных структур объектов, устойчивых к изменениям Техника проектирования и разработки, которая хорошо подходит для гибкой разработки. Хорошо сочетается с микросервисной архитектурой. Преимущества: Позволяет автоматизировать незнакомые разработчикам предметные области, позволяет вести разработку итерационно, постепенно усложняя ПО, позволяет значительно ускорить разработку сложного ПО. Недостатки: Требует лояльности и гибкости заказчика, требует сплочённой и профессиональной команды.</p>
<p>Область (Domain) — предметная область, к которой применяется разрабатываемое программное обеспечение; Язык описания — используется для единого описания модели предметной области; Модель (Model) — описывает конкретную предметную область или её часть, является базой для автоматизации.</p> <p>Карта контекстов — схематическое изображение взаимодействия изолированных контекстов.</p> 	<p>Изоляция предметной области — отделение модели предметной области от инфраструктурных и других решений Необходима для: – замены архитектурных элементов без изменения кода модели предметной области; – распространение понятия ограниченного контекста на архитектурные решения. Для реализации используются различные архитектуры: MVC, многоуровневая архитектура, «чистая архитектура», гексогональная архитектура.</p>	<p>Многоуровневая архитектура: Размывание кода предметной области приводит к неадаптивности проекта. Программу, разделённую на уровни гораздо проще поддерживать, т. к. они имеют тенденцию развиваться разными темпами и обслуживать разные потребности. Уровни: – Интерфейс пользователя (уровень представления) – UI – Операционный уровень (уровень прикладных операций, уровень приложения) – AL – Уровень предметной области (уровень модели, уровень бизнес-логики) – DL – Инфраструктурный уровень (уровень доступа к данным) – IL</p>
<p>Интерфейс пользователя: Отвечает за вывод информации пользователю, интерпретацию команд пользователя. Внешним действующим субъектом может быть не человек, а другая программа.</p>	<p>Операционный уровень: Определяет задачи, связанные с конкретным действием в UI (команда пользователя или потребность в информации) и распределяет их между объектами предметной области. Не хранит состояний объектов предметной области. Может играть интегрирующую роль - взаимодействовать с операционными уровнями других систем. Наиболее близкий ШП: Facade</p>	<p>Инфраструктурный уровень – слой технических сервисов. Обеспечивает техническую поддержку для верхних уровней: передачу сообщений на операционном уровне; непрерывность существования объектов на уровне модели (хранение, транзакционность и т.д.); службы передачи сообщений; почтовые службы.</p>
<p>Уровень предметной области: Отвечает за: представление понятий прикладной предметной области, рабочие состояния, бизнес-регламенты (поведение модели). Этот уровень является главной, алгоритмической частью программы</p>	<p>Изолированный контекст – область применения конкретной модели с определёнными границами. Изолированные контексты дают членам группы разработки чёткое и общее представление о том, где следует соблюдать согласованность, а где можно работать независимо. В результате группы разработки могут разделять VCS и CI, а значит работать в разных темпах. Итог: изолированный контекст обеспечивает четко определенные безопасные гавани, позволяя моделям усложняться, не жертвуя концептуальной целостностью.</p>	<p>Шаблоны взаимодействия контекстов – Партнёрство: Команды в двух контекстах работают вместе. Эти команды устанавливают процесс координированного планирования разработки и совместное управление интеграцией. Они должны сотрудничать в процессе эволюции своих интерфейсов, чтобы учитывать потребности обеих систем. Взаимозависимые функции должны быть организованы так, чтобы завершение их разработки происходило в рамках одного выпуска.</p>

<p>Шаблоны взаимодействия контекстов – Разработка «заказчик-поставщик»:</p> <p>Две команды находятся в отношениях «заказчик-поставщик» (down-up), причём успех вышестоящей команды зависит от нижестоящей. Следует учитывать приоритеты команд «заказчиков» при планировании работы «поставщиков». Проводите переговоры и согласовывайте планы, учитывая потребности команд «заказчиков».</p>	<p>Шаблоны взаимодействия контекстов – Конформист:</p> <p>Команда-поставщик (up) по каким-то причинам не может удовлетворить потребности «заказчика» и предоставить свою модель для работы. Создайте слой объектов, имитирующих модель «поставщика».</p>	<p>Шаблоны взаимодействия контекстов – Предохранительный уровень (ACL):</p> <p>Команда «поставщик» по каким-то причинам не может обеспечить неизменный интерфейс своего модуля. Создайте трансляционный уровень (набор фасадных объектов), предназначенных для быстрой настройки взаимодействия.</p>
<p>Шаблоны взаимодействия контекстов – Служба с открытым протоколом (OHS):</p> <p>Модель является «поставщиком» для многих заказчиков. Определите протокол, предоставляющий доступ к вашей системе, как к набору служб. Откройте этот протокол для всех, кто захочет интегрироваться с вами.</p>	<p>Шаблоны взаимодействия контекстов – Общедоступный язык (PL):</p> <p>Взаимодействие между моделями требует общий язык. Используйте в качестве средства коммуникации хорошо документированный общий язык, который может выразить необходимую информацию о предметной области, выполняя при необходимости перевод информации между другими языками. Общедоступный язык часто сочетается со службой с открытым протоколом.</p>	<p>Шаблоны взаимодействия контекстов – Большой комок грязи:</p> <p>Выполняя анализ предметной области или существующей программной реализации предметной области, мы можем обнаружить, что существуют модули, в которых модели перемешаны, а границы стёрты. Нарисуйте границы вокруг такой смеси и обозначьте её как «большой комок грязи». Не пытайтесь применить изощрённое моделирование внутри этого контекста.</p>
<p>Шаблоны взаимодействия контекстов – Служба с открытым протоколом:</p> <p>Этот шаблон можно реализовать в виде REST-ресурса, с которым взаимодействует клиент из мира ограниченных контекстов проекта. Обычно службу с открытым протоколом представляют в виде RPC из интерфейса прикладного программирования, но её можно реализовать через механизм отправки сообщений.</p>		