2 вариант: 1) ООП - полное определение 2) Процедурная - Структурная декомпозиция, примеры 3) Архитектура ПО - определение, примеры 4) Энтропия ПО, не менее 3 примеров 5) Закон Иерархических Компенсаций Седова в контексте АйТи,, не менее 7 примеров.

1)  
Объектно-ориентированное программирование (ООП) - это метод проектирования программ, направленный на упрощение работы с реальными и абстрактными системами за счёт моделирования их через объекты. Его основная цель - не просто организация кода, а управление доменной сложностью, то есть сложностью предметной области. Ключевым принципом является инкапсуляция, которая позволяет отделить «что делает объект» от «как он это делает» и скрыть изменчивость реализации за устойчивым интерфейсом. Наследование и полиморфизм служат средствами расширения и переиспользования поведения, облегчая развитие системы.

ООП строится на идее иерархий и сильной внутренней связности компонентов при минимальных внешних зависимостях. Это отражает естественную структуру сложных систем - как в биологии или физике, где крупные сущности состоят из более простых. Такой подход позволяет проектировать системы эволюционно, постепенно улучшая и реорганизуя их без полной перестройки.

ООП не решает организационные или математические проблемы, но значительно упрощает понимание, сопровождение и развитие программных продуктов. Таким образом, ООП - это инструмент управления сложностью и изменениям, а не просто набор синтаксических конструкций языка.

2)

Процедурная (структурная) декомпозиция - это метод проектирования программ, при котором задача разбивается на последовательность подзадач (процедур, функций). Каждая процедура выполняет строго определённое действие, а программа строится как иерархия вызовов - от общей задачи к частным шагам. Такой подход позволяет упростить понимание программы, повторно использовать код и локализовать ошибки.

Главная идея - «разделяй и властвуй»: сложная задача делится на более простые и понятные части. В отличие от объектно-ориентированного подхода, в процедурной декомпозиции акцент делается не на объектах, а на процессах (действиях).

Пример 1.  
При написании программы для вычисления площади треугольника можно выделить процедуры:

readData() - ввод сторон;

calcPerimeter() - вычисление периметра;

calcArea() - вычисление площади по формуле Герона;

printResult() - вывод результата.

Пример 2.  
В программе сортировки массива:

readArray() - ввод элементов;

sortArray() - сортировка (например, пузырьком);

printArray() - вывод отсортированных данных.

3)

Архитектура программного обеспечения - это структура и организация системы, определяющая основные компоненты, связи между ними и принципы их взаимодействия. Архитектура задаёт основу для проектирования, разработки и сопровождения программы, обеспечивая целостность и управляемость проекта. Она определяет, какие модули существуют, как они обмениваются данными и какие технологии используются.  
Примеры архитектур:

Монолитная архитектура - всё приложение представляет собой единое целое (например, старые ERP-системы).

Клиент-серверная - взаимодействие клиента (интерфейс) и сервера (логика, данные).

Многоуровневая (трёхзвенная) - разделение на уровни: представления, логики и данных.

Микросервисная архитектура - система из независимых сервисов, взаимодействующих через API.

4)

Энтропия программного обеспечения - это степень беспорядка и хаоса в коде, возникающая по мере его развития и изменений. Чем выше энтропия, тем труднее понимать, изменять и сопровождать систему. Она растёт, если не проводится рефакторинг и нарушаются архитектурные принципы.  
Примеры:

В проект добавляют новые функции, не соблюдая структуру - код становится непредсказуемым.

Несогласованные изменения нескольких разработчиков приводят к конфликтам и дублированию логики.

Отсутствие документации делает даже простые задачи трудными для новых участников.

5)

Закон иерархических компенсаций Седова гласит, что при упрощении одной части системы неизбежно усложняется другая, упрощение - не бесплатное.  
Примеры:

Использование фреймворков упрощает разработку, но усложняет разбор внутренних механизмов.

Облачные сервисы снижают нагрузку на локальные серверы, но повышают зависимость от сети и поставщика.

Автоматизация сборки экономит время, но требует сложной настройки CI/CD.

ORM скрывает SQL-запросы, но делает диагностику производительности труднее.

GUI-интерфейсы упрощают работу пользователя, но усложняют реализацию по сравнению с консолью.

Микросервисы упрощают масштабирование, но усложняют отладку и мониторинг.

Высокоуровневые языки (Python, Java) упрощают код, но снижают контроль над памятью и производительностью.