1. Жизненный цикл ПО и стратегии разработки

ЖЦ ПО — период от идеи до вывода из эксплуатации.
Стандарты: ISO/IEC 12207, ГОСТ Р ИСО/МЭК 12207.
Основные процессы:

- Заказ: требования, выбор поставщика, контракт.
- Поставка: разработка и передача заказчику.
- Разработка: анализ → проектирование → кодирование → тестирование → внедрение.
- Эксплуатация: использование и поддержка.
- Сопровождение: исправления, адаптация, модернизация (самый длительный этап).
- Завершение: архивация, вывод.

Модели ЖЦ:

- **Каскадная**: строго последовательно, требования фиксируются в начале.
- **Итеративная/инкрементная**: по частям (инкрементам), каждый добавляет функционал.
- **Спиральная** (Боэм): итерации = витки — анализ требований → оценка рисков → разработка → оценка.
- **Гибкие (Agile)**: короткие циклы, активное участие заказчика, адаптация под изменения.

Стратегии разработки:

- **Монолитная**: единое целое.
- **Модульная**: разделение на независимые модули.
- **Компонентная (CBD)**: сборка из готовых компонентов.
- **SOA / микросервисы**: слабосвязные, переиспользуемые сервисы.
- **Прототипирование**: упрощённая модель для уточнения требований.
- **Open Source**: разработка на основе открытого кода.

ПО vs Продукт vs Система:

- **ПО** — программа для решения задачи разработчиком (минимум док-ции, нет гарантий).
- **Продукт** — готовое, документированное, тестируемое, с интерфейсом и гарантией — товар.
- **Система** — совокупность ПО, аппаратуры, данных, персонала — единое целое с новыми свойствами.

2. ЖЦ ПО по ISO/IEC 12207

Процессы делятся на 3 группы:

1. **Основные** (5): заказ, поставка, разработка (13 работ), эксплуатация, сопровождение.
2. **Вспомогательные** (8): документирование, УК, обеспечение качества, верификация («делаем ли правильно?»), аттестация («делаем ли нужное?»), решение проблем и др.
3. **Организационные** (4): управление, инфраструктура, улучшение, обучение.

3. Стратегии разработки: достоинства/недостатки

Стратегия	+	–
Каскадная	Простота, предсказуемость, хороша при стабильных требованиях	Негибкость, ошибки выявляются поздно, отсутствие обратной связи до конца
Инкрементная	Ранняя поставка, снижение рисков, улучшаемость	Требует хорошей архитектуры, дороже суммарно, полезная функциональность — только в конце
Эволюционная (Agile)	Гибкость, ранняя обратная связь, адаптация	Сложно планировать сроки, требует зрелой команды, риск «бесконечного» проекта

4. Выбор стратегии под проект

Зависит от:

- **Требований**: стабильные → каскад/инкремент; изменчивые → Agile.
- **Размера**: малые → Agile; крупные → инкремент/RUP.
- **Рисков**: высокие → спираль/V-модель.
- **Команды**: зрелая → Agile; новая → структурированные подходы.
- **Заказчика**: активный → Agile; пассивный → каскад.
- **Сроков/бюджета**: жёсткие → каскад с контрольными точками.

5. Инкрементная vs эволюционная стратегии

- **Инкрементная**: требования известны заранее; инкременты добавляют функционал к работающей системе.
- **Эволюционная**: требования уточняются итеративно; MVP → доработка по обратной связи.
  → Инкремент — «план + поставка», эволюционная — «поиск + адаптация».

6. Каскадные модели

- **Классическая (водопад)**: строго последовательно.
- **V-модель**: каждой фазе разработки — своя фаза тестирования (требования → приёмочные тесты и т.д.).
- **С промежуточным контролем**: каскад + инкрементная поставка (архитектура сразу, реализация по частям).

7. Классическая каскадная и с обратными связями

- **Классическая**: возвраты почти невозможны, ошибки обнаруживаются на тестировании.
- **С обратными связями**: допускаются возвраты для исправления ошибок → снижает риски, но усложняет планирование.

8. RAD-модели

RAD — быстрая разработка с активным участием заказчика и прототипированием. Варианты:

- **Базовая**: сбор требований → прототип → сборка → внедрение (60–90 дней).
- **На основе моделирования предметной области**: бизнес-модель → генерация кода.
- **Параллельная**: команды работают над модулями одновременно.
  → Подходит при активном заказчике, модульности и готовых компонентах.

9. Инкрементные модели

- **Классическая**: архитектура сразу, модули последовательно.

- **По ГОСТ 15271**: формальное документирование и приёмка каждого инкремента.
- **XP (экстремальное программирование)**: короткие спринты, TDD, парное программирование, рефакторинг.
  → Баланс между предсказуемостью и гибкостью.

10. Инкремент с уточнением требований и XP

- **С уточнением**: требования детализируются перед каждым инкрементом.
- **По ГОСТ**: строгое документирование и приёмка.
- **XP**: итерации 1–3 нед, TDD, активная обратная связь.
  → От жёсткого (ГОСТ) к гибкому (XP).

11. Эволюционные модели

- **Спираль Боэма**: итерации = витки: план → анализ рисков → разработка → оценка.
- **Прототипирование**: throwaway (для уточнения) или evolutionary (в рост продукта).
- **Agile**: Scrum, Kanban, XP — короткие циклы, приоритет у работающего ПО и заказчика.
  → Лучше при неопределённости и высоких рисках.

12. Эволюционные по ГОСТ и структурная модель

- **По ГОСТ 15271**: формализованное прототипирование с документированием и верификацией.
- **Структурная модель**: сначала архитектура, потом итеративное наполнение — баланс гибкости и контроля.
  → ГОСТ — для регулируемых проектов; структурная — для сложных систем.

13. Спиральная модель и упрощённые варианты

- **Спираль Боэма**: для крупных рискованных проектов (авиация, медицина).
- **Упрощённые**: итеративная разработка (без формального анализа рисков), прототипирование, RUP.

→ Боэм — для аэрокосмоса/медицины; упрощённые — для коммерческих проектов.

14. Выбор модели ЖЦ под проект

Факторы: стабильность требований, размер, риски, зрелость команды, участие заказчика, сроки.
→ Классификация проектов: малые (Agile), средние (инкремент/RUP), крупные (спираль/V-модель); регулируемые → V-модель; инновационные → Agile.

15. Процедура выбора модели ЖЦ

1. Анализ проекта (требования, риски, команда).
2. Сопоставление с моделями (матрица).
3. Выбор и адаптация (гибрид: каскад + Agile).
4. Документирование и внедрение.
5. Мониторинг и корректировка.
   → Правило: >70% требований стабильны → инкремент; иначе — Agile.

16. Адаптация модели под проект

Модель настраивается:

- по формальности (ГОСТ → Agile),
- по длительности итераций (спринты 1–4 нед),
- по ролям (объединение/делегирование),
- по управлению требованиями (жёстко/гибко).
  → Гибриды: каскад + прототипирование, инкремент + спринты, Agile в регулируемой среде.

17. Оценка качества ПО, стандартизация, документы

Качество (ISO 25010): функциональность, производительность, безопасность, сопровождаемость и др.

Стандартизация: ISO/IEC 12207, CMMI, ГОСТ 19 (ЕСПД).
Основные документы ЕСПД: ТЗ, ПЗ, Руководства (пользователя, админа), Формуляр.

18. Формирование документации, принципы

Документация — по ГОСТ 19 / ISO/IEC/IEEE 26511–26515. Требования: структура, нумерация, единообразие, запрет жаргона.
Современные тенденции: Docs-as-Code (Markdown + Git), автогенерация (Swagger, Javadoc), минимизация в Agile.

19. Стандарты качества документации

- **ГОСТ 19**: жёсткая регламентация, для госзаказов.
- **ISO/IEC/IEEE**: гибкие, фокус на содержании.
  → Современный подход: гибрид — формальность ГОСТ + содержание ISO + автогенерация.

20. Меры и метрики ПО: сложность и покрытие

- **Сложность**: цикломатическая (V(G) ≤ 10 — норм), Холстеда (объём, сложность), связность/сцепление.
- **Покрытие**: операторов (statement), ветвей (branch), условий (condition), MC/DC (для авиации).
  → Цель: снижать сложность, повышать покрытие (≥85% branch — хорошо).

21. Классические методологии и структурное программирование

Классика (1960–80-е): структурное программирование, анализ/проектирование, информационное моделирование.
→ **Структурное программирование** (Дейкстра): только 3 конструкции (последовательность, ветвление, цикл), без `goto`, модульность, нисходящий дизайн.

22. Принципы структурного программирования

1. Единственная точка входа/выхода.
2. Композиция из 3 базовых конструкций.
3. Нисходящий дизайн + модульность (один экран = один модуль).
4. Локальность данных + информационное сокрытие.
   → Реализация: `if` / `for` / `while` , процедуры, блоки видимости (Pascal, C).

23. Графическое представление, модульное проектирование, нисходящий дизайн

- **Блок-схемы**: прямоугольник (действие), ромб (решение), стрелки (поток).
- **Модульность**: низкое сцепление + высокая связность. Типы связности: функциональная (лучше) → случайная (хуже).
- **Нисходящий дизайн**: сверху вниз: главная функция → декомпозиция → элементарные модули.

24. Проектирование с псевдокодом и управляющими конструкциями

Псевдокод — неформальное, структурированное описание алгоритма (IF/WHILE/FOR/ВЫЗОВ).
Пример: валидация пароля → `CheckLength` , `CheckCase` , `CheckDigits` .
→ Преимущество: фокус на логике, а не синтаксисе; легко переводится в код.

25. CASE-технологии и IDEFO

CASE — автоматизация ЖЦ ПО (анализ, проектирование, генерация). Upper CASE — требования/архитектура; Lower CASE — код/тесты.
IDEFO — функциональное моделирование: блоки (действия) + стрелки (вход, управление, выход, механизм). Декомпозиция → иерархия диаграмм.

26–27. SADT, IDEFO, DFD

- **SADT** — предшественник IDEFO.
- **IDEFO**: функции + 4 типа стрелок. Синтаксис: 3–6 блоков на диаграмме, баланс интерфейсов.
- **DFD**: потоки данных: внешние сущности → процессы → хранилища (без управления!). Запрещены прямые связи сущностей и хранилищ.

28. DFD: понятия, синтаксис

Элементы: внешние сущности (□), процессы (—), хранилища (‖), потоки (→).
Правила: 1) потоки только через процессы; 2) у процесса ≥1 вход/выход; 3) баланс при декомпозиции; 4) имена — существительные.
→ Фокус: «какие данные куда движутся».

29–30. IDEF1X: сущности, атрибуты, связи

- **Сущность**: объект (прямоугольник: острый угол — независимая, скруглённый — зависимая).
- **Атрибуты**: PK (PK), FK (FK), AK (AK). 1NF: атомарность + уникальность PK.
- **Связи**:
  - идентифицирующая (сплошная + точка) → FK входит в PK дочерней;
  - неидентифицирующая (пунктир + точка/кружок) → FK вне PK;
  - обязательная (точка) / необязательная (кружок).
    → M:N → промежуточная сущность.

31. Реализация связей, неспецифические, JSD, Варнье-Орр

- **Реализация**: обязательная связь → FK NOT NULL; необязательная → FK NULL.
- **Неспецифические (M:N)**: расщепляются через пересекающую сущность.
- **JSD (Джексон)**: структура программы = структуре данных (последовательность/выбор/цикл).
- **Диаграммы Варнье-Орр**: иерархия фигурных скобок для структур.

32–33. UML: основы, классификация диаграмм

UML — стандарт визуального моделирования (OMG, ISO).

- **Структурные**: классов (главная), компонентов, развёртывания, пакетов.
- **Поведенческие**: use case (требования), последовательностей (взаимодействие), деятельности (процессы), состояний (жизненный цикл).
  → Использовать только нужные (обычно 4–5 диаграмм).

34–35. Диаграммы use case и классов

- **Use case**: актёры (stickman), прецеденты (овал), связи (ассоциация, include/extend, обобщение). Цель: договорённость по функционалу.
- **Классов**: прямоугольник (имя | атрибуты | методы), связи: ассоциация, агрегация (◇), композиция (◆), наследование (▲), реализация (---▷), зависимость (- - -▷).

36–38. → см. выше (дубли вопросов)

39. Цели и виды тестирования

Цель — выявить дефекты, снизить риски, подтвердить соответствие.
Виды:

- по уровню: модульное → интеграционное → системное → приёмочное;
- по подходу: «чёрный ящик» (требования), «белый» (код), «серый» (гибрид);
- по направлению: функциональное, нефункциональное (производительность, безопасность), регрессионное.

40. Планирование тестирования

Этапы: анализ → стратегия → ресурсы → расписание → данные → метрики.
Артефакты: тест-план (IEEE 829), чек-листы, тест-кейсы, RTM, отчёты.
→ В Agile: лёгкий план на спринт, фокус на DoD и тест-досках.

41. Тестовое покрытие

- **Операторов**: каждая строка выполнена ≥1 раз.
- **Ветвей**: каждая ветвь `if` → True/False.
- **Условий**: каждое простое условие → T/F (для `A && B` — 4 комбинации).
- **MC/DC**: каждое условие независимо влияет на результат (для авиации).
  → Цель: ≥85% branch coverage.

42. Тест-сценарии, пакеты, документация

- **Сценарий** — высокоуровневая цель («оформить заказ»).
- **Пакет** — набор тестов по признаку (функция, приоритет).
- **Документация**: тест-план, тест-кейсы (шаги + ожидаемый результат), баг-репорт (воспроизведение + severity/priority), RTM.
  → В Agile: кейсы в Jira, документация — в коде/чек-листах.

43. Анализ спецификаций, верификация и аттестация

- **Анализ**: поиск противоречий, неопределённостей, SMART-требований.
- **Верификация** (Verification): «делаем ли правильно?» (ревью, модульные тесты).
- **Аттестация** (Validation): «делаем ли нужное?» (системные тесты, UAT).
  → V-модель: левая сторона — разработка, правая — тестирование.

44–45. Связность и сцепление модулей

- **Связность** (внутри модуля): функциональная (лучше) → случайная (хуже).
- **Сцепление** (между модулями): по данным (лучше) → по содержимому (хуже).
  → Идеал: высокая связность + низкое сцепление.

46. Комментарии в структурном программировании

Цель: объяснить «почему», а не «что».
Правила:

- комментировать неочевидные решения, бизнес-логику, ограничения;
- не дублировать код/имена;
- использовать TODO/FIXME.
  → Лучший комментарий — тот, который можно удалить, переписав код понятнее.

47. Декомпозиция в IDEF0

Процесс: A-0 (контекст) → A0 (3–6 блоков) → рекурсивная декомпозиция.
Стратегии: функциональная, объектная, временная, организационная.
→ Критерии качества: 3–6 блоков на диаграмме, баланс интерфейсов, низкое сцепление.

48. Нормализация в IDEF1X

Цель: устранить избыточность и аномалии.
Этапы:

- **1NF**: атомарные атрибуты + PK;
- **2NF**: полная зависимость от PK (разбить составные ключи);
- **3NF**: нет транзитивных зависимостей (вынести зависимости между неключевыми).
  → В IDEF1X достигается через типы сущностей и связи.

49. Тестирование «чёрного ящика»

- **Классы эквивалентности**: разбить входы на валидные/невалидные группы → по 1 тесту на класс.
- **Граничные значения**: для каждой границы — само значение + ±1.
  → Комбинация: 3–6 тестов вместо сотен (например, возраст 17, 18, 19, 64, 65, 66).

50. Code review

Цель: найти дефекты, улучшить качество, обмен знаниями.
Виды: формальный (Fagan), парное программирование, PR-based.
Этапы: подготовка → отправка → проверка (логика, безопасность, стиль) → правки → одобрение → merge.
→ Правила: маленькие PR (<400 строк), фокус на сути, уважительные комментарии.