**1. Сколько времени программисты тратят на написание кода?**

Этот вопрос не имеет однозначного ответа, поскольку время, затрачиваемое на написание кода, сильно варьируется в зависимости от множества факторов:

* **Уровень опыта:**
  + **Джуниор (Junior) разработчики:** Могут тратить больше времени непосредственно на написание кода, чем на другие задачи.
  + **Мидл (Middle) разработчики:** Балансируют между написанием кода, проектированием и тестированием.
  + **Сеньор (Senior) разработчики:** Занимаются не только написанием кода, но и архитектурой, планированием, менторством, рефакторингом, поиском и решением проблем.
* **Сложность проекта:** На сложные проекты, конечно же, уходит больше времени на написание кода.
* **Методология разработки:**
  + В гибких методологиях (Agile, Scrum) разработка ведется итерациями, и написание кода может быть более интенсивным.
  + В традиционных методологиях (Waterfall) разработка может быть более последовательной.
* **Стек технологий:** Разные языки программирования и фреймворки могут потребовать разного времени на написание кода.
* **Культура компании:** В некоторых компаниях больше внимания уделяют планированию и тестированию, в других - быстрой разработке.

**Что говорят сами программисты (примерные оценки):**

* **Stack Overflow Developer Survey:** В исследованиях Stack Overflow (самый крупный ресурс для программистов) часто упоминается, что программисты тратят около **25-40% своего времени** на написание кода. Остальное время уходит на проектирование, тестирование, отладку, документирование, совещания и другие задачи.
* **Блоги и статьи разработчиков:** Многие разработчики утверждают, что **30-50% их времени** уходит на написание кода, а остальное – на другие задачи. Причём чем выше уровень разработчика, тем меньший процент времени у него уходит на непосредственно написание кода.
* **Внутренние оценки компаний:** По данным внутренних исследований разных IT-компаний, на написание кода уходит от **20% до 50%** рабочего времени. При этом компании стараются организовать рабочие процессы таким образом, чтобы разработчики могли сосредоточиться на коде и не отвлекались на другие дела.

**Общий вывод:**

Сами программисты, как правило, говорят, что на написание кода они тратят примерно **25-50% своего рабочего времени**. Остальное время уходит на другие задачи, связанные с разработкой.

**2. Из чего классически состоит контракт:**

Контракт — это юридический документ, который определяет права и обязанности сторон. Обычно контракты имеют следующую структуру:

1. **Преамбула (Preamble):**
   * Вводная часть, которая идентифицирует стороны контракта (полные наименования организаций, ФИО), их юридические адреса и реквизиты.
   * Указывается дата заключения контракта.
   * Иногда указываются причины заключения контракта и общие намерения сторон.
2. **Предмет контракта (Subject Matter of Contract):**
   * Четкое описание того, что является предметом контракта (товары, услуги, работы).
   * Определение конкретных задач, требований, характеристик, объемов.
3. **Обязанности сторон (Obligations of the Parties):**
   * Подробное описание обязанностей каждой из сторон (исполнителя и заказчика, продавца и покупателя и т.д.).
   * Сроки выполнения работ (оказания услуг, поставки товаров).
   * Требования к качеству работ (услуг, товаров).
   * Порядок взаимодействия сторон.
4. **Цена контракта и порядок расчетов (Contract Price and Payment Procedure):**
   * Общая стоимость контракта (сумма, подлежащая уплате).
   * Порядок и сроки оплаты (аванс, поэтапная оплата, постоплата и т.д.).
   * Валюта оплаты.
   * Может быть указана схема оплаты, график платежей.
   * Может быть указан порядок формирования цены (фикс. цена, оплата по часам, и пр.)
5. **Сроки действия контракта (Term of the Contract):**
   * Дата начала и окончания действия контракта.
   * Условия продления или расторжения контракта.
6. **Ответственность сторон (Liability of the Parties):**
   * Мера ответственности сторон за неисполнение или ненадлежащее исполнение обязательств.
   * Санкции за нарушение условий контракта (штрафы, пени).
   * Условия возмещения убытков.
7. **Конфиденциальность (Confidentiality):**
   * Обязательство сторон не разглашать конфиденциальную информацию, полученную в рамках контракта.
   * Срок действия обязательства по конфиденциальности.
8. **Форс-мажор (Force Majeure):**
   * Обстоятельства непреодолимой силы, освобождающие стороны от ответственности (стихийные бедствия, войны, эпидемии и т.д.).
   * Процедура действий в случае наступления форс-мажорных обстоятельств.
9. **Порядок разрешения споров (Dispute Resolution):**
   * Процедура разрешения споров, которые могут возникнуть в процессе исполнения контракта.
   * Возможность досудебного урегулирования споров.
   * Арбитражный суд (если требуется).
10. **Заключительные положения (Final Provisions):**
    * Прочие условия, которые стороны считают важными.
    * Указание на то, в скольких экземплярах составлен контракт.
    * Условия изменения контракта.
    * Юрисдикция.
11. **Реквизиты сторон (Details of the Parties):**
    * Полные юридические наименования сторон, адреса и банковские реквизиты.
    * Подписи сторон и печати (если применимо).

**Важно:**

* Это *общая структура*, и конкретный *состав пунктов* может варьироваться *в зависимости от типа контракта* и *требований законодательства*.
* Необходимо *обратиться к специалистам* для составления *юридически грамотного* контракта.

**3. Legacy, рефакторинг и техдолг — что это и какая между ними связь:**

* **Legacy (Легаси)**
  + **Что это:** Legacy-код (или legacy-система) — это устаревший код или система, которая *была разработана* и *используется длительное время*. Часто этот код *не соответствует современным стандартам* разработки, *сложен для поддержки и модификации*, *плохо документирован* или *написан с использованием устаревших технологий*.
  + **Характеристики:** *Сложность*, *отсутствие тестов*, *недостаточная документация*, *использование устаревших технологий*, *низкая производительность*, *высокий риск возникновения ошибок*.
* **Технический долг (Technical Debt)**
  + **Что это:** Технический долг — это метафора, описывающая ситуацию, когда *при разработке программного обеспечения* *сознательно или невольно принимаются решения*, *упрощающие разработку в краткосрочной перспективе*, но *увеличивающие стоимость поддержки и развития проекта в долгосрочной перспективе*.
  + **Причины возникновения:** Недостаток времени, недостаток опыта, нечеткие требования, принятие компромиссных решений.
  + **Виды:** *Слабый код*, *недостаточные тесты*, *отсутствие документации*, *дублирование кода*, *неправильная архитектура*, использование неоптимальных алгоритмов.
  + **Последствия:** Замедление разработки, увеличение количества ошибок, увеличение стоимости поддержки и развития.
* **Рефакторинг (Refactoring)**
  + **Что это:** Рефакторинг — это *процесс изменения* *внутренней структуры кода* *без изменения его внешнего поведения*. Цель рефакторинга — *улучшить читаемость*, *управляемость*, *производительность и расширяемость кода*.
  + **Цели:** *Упрощение кода*, *устранение дублирования*, *повышение модульности*, *улучшение понимания*, *подготовка к внесению изменений*.
  + **Как происходит:** Улучшение структуры кода, разбиение на более мелкие функции, переименование переменных и методов, создание абстракций.

**Связь между Legacy, рефакторингом и техдолгом:**

* **Legacy и техдолг:** Legacy-код часто является результатом накопления технического долга с течением времени. Устаревший код сложен для поддержки, часто не имеет документации и тестов, что приводит к усложнению разработки и повышению риска возникновения ошибок.
* **Рефакторинг и техдолг:** Рефакторинг – это *инструмент* для *управления техническим долгом*. Регулярный рефакторинг позволяет *уменьшить техдолг*, *упростить код* и *снизить затраты* на его поддержку и развитие.
* **Рефакторинг и Legacy:** Рефакторинг *может использоваться для работы с Legacy-кодом*, чтобы сделать его *более понятным, поддерживаемым и пригодным* для дальнейшего развития.

**Простыми словами:**

* **Legacy:** Старый, сложный и неудобный код.
* **Техдолг:** Результат поспешных решений, которые потом усложняют разработку.
* **Рефакторинг:** Способ сделать код лучше без изменения его функциональности.