### 1. \*\*Как правильно структурировать техническое задание на разработку ПО? Пример структуры\*\*

Техническое задание (ТЗ) должно быть четким, полным и однозначным. Пример структуры, соответствующей ГОСТ 34.602-89 и рекомендациям IEEE 830:

1. \*\*Общие положения\*\*

- Назначение документа (цель проекта).

- Область применения ПО.

- Перечень нормативных документов (стандарты, ГОСТы).

- Термины и определения (глоссарий).

2. \*\*Описание бизнес-требований\*\*

- Цели и задачи проекта.

- Целевая аудитория.

- Бизнес-процессы, которые автоматизируются.

3. \*\*Функциональные требования\*\*

- Список функций системы (по модулям или ролям).

- Сценарии использования (Use Cases).

- Ограничения (например, интеграция с внешними системами).

4. \*\*Нефункциональные требования\*\*

- Производительность (нагрузка, время отклика).

- Безопасность (авторизация, шифрование).

- Надежность и отказоустойчивость.

- Технологии и платформы (языки, СУБД, ОС).

5. \*\*Интерфейсы\*\*

- API, интеграции с другими системами.

- Требования к UI/UX (макеты, доступность).

6. \*\*Стадии разработки и сроки\*\*

- Этапы проекта (анализ, разработка, тестирование).

- График работ (мильные камни).

7. \*\*Условия приемки\*\*

- Критерии приемки (тест-кейсы, метрики качества).

- Порядок сдачи-приемки.

\*\*Пример\*\*:

> \*\*3. Функциональные требования\*\*

> - Модуль авторизации:

> - Регистрация через email и соцсети.

> - Двухфакторная аутентификация.

---

### 2. \*\*Основные ошибки при составлении ТЗ и как их избежать\*\*

- \*\*Неоднозначные формулировки\*\*

Пример: "Система должна работать быстро" → Исправление: "Время отклика системы не должно превышать 2 секунд при нагрузке 1000 пользователей".

\*\*Как избежать\*\*: Использовать SMART-критерии (конкретные, измеримые требования).

- \*\*Игнорирование нефункциональных требований\*\*

Например, не указаны требования к безопасности или масштабируемости.

\*\*Как избежать\*\*: Добавить разделы по производительности, безопасности, юзабилити.

- \*\*Избыточная детализация или недостаток информации\*\*

Пример: Описание интерфейса без макетов → Исправление: Приложить прототипы экранов.

\*\*Как избежать\*\*: Использовать диаграммы, таблицы, примеры.

- \*\*Неучет изменений\*\*

\*\*Как избежать\*\*: Включить процесс внесения правок (например, Change Request).

---

### 3. \*\*Отличие ТЗ от брифа и SRS\*\*

- \*\*Бриф\*\* — краткий опросник для первичного сбора данных (цели, аудитория, бюджет). Не содержит технических деталей.

- \*\*ТЗ\*\* — детальный документ с требованиями к функционалу, технологиям и срокам. Основа для договора между заказчиком и исполнителем.

- \*\*SRS (Software Requirements Specification)\*\* — более формализованный документ (по стандарту IEEE 830), фокусируется на требованиях к ПО, часто используется в agile-проектах.

\*\*Пример\*\*:

- В брифе: "Нужен мобильный банк для клиентов".

- В ТЗ: "Приложение должно поддерживать авторизацию по FaceID, переводы между счетами, push-уведомления".

- В SRS: "Сценарий перевода: предварительная проверка лимитов, генерация SSL-ключа для транзакции".

---

### 4. \*\*Стандарты для оформления ТЗ\*\*

- \*\*ГОСТ 34.602-89\*\* — советский стандарт для ТЗ на информационные системы. Включает разделы: общие требования, функции системы, надежность, этапы разработки.

- \*\*IEEE 830-1998\*\* — международный стандарт для SRS. Акцент на структуре требований: введение, общее описание, спецификации.

- \*\*ISO/IEC/IEEE 29148\*\* — современный стандарт, заменяющий IEEE 830. Включает рекомендации по формулировкам требований.

\*\*Когда применять\*\*:

- ГОСТ 34 — для госзакупок в РФ.

- IEEE/ISO — для международных проектов или стартапов.

---

Блок 2

### \*\*2.1. Как правильно формулировать функциональные требования в ТЗ? Примеры\*\*

Функциональные требования описывают \*\*что\*\* система должна делать. Они должны быть:

- \*\*Конкретными\*\* (без расплывчатых формулировок),

- \*\*Измеримыми\*\* (чтобы можно было проверить выполнение),

- \*\*Связанными с действиями\*\* (например, "пользователь может...").

\*\*Примеры:\*\*

1. \*\*Регистрация пользователя:\*\*

> Система должна предоставлять возможность регистрации через email с подтверждением по ссылке в письме.

2. \*\*Поиск товаров:\*\*

> Пользователь может фильтровать товары по цене (от 100 до 10 000 руб.), категории и рейтингу.

3. \*\*Оплата заказа:\*\*

> Система должна интегрироваться с платежным шлюзом Stripe для обработки транзакций.

4. \*\*Уведомления:\*\*

> При изменении статуса заказа система отправляет email-уведомление клиенту в течение 5 минут.

5. \*\*Администрирование:\*\*

> Администратор может блокировать пользователей через панель управления с указанием причины.

\*\*Ошибка → Исправление:\*\*

- ❌ \*"Система должна быть удобной"\* → ✅ \*"Пользователь может выполнить заказ за 3 клика, начиная с главной страницы"\*.

---

### \*\*2.2. Что такое нефункциональные требования? Примеры для веб-приложения\*\*

Нефункциональные требования определяют \*\*как\*\* система должна работать:

- \*\*Производительность\*\*,

- \*\*Безопасность\*\*,

- \*\*Масштабируемость\*\*,

- \*\*Совместимость\*\*,

- \*\*Юзабилити\*\*.

\*\*Примеры для веб-приложения:\*\*

1. \*\*Производительность:\*\*

> Время загрузки страницы не должно превышать 2 секунд при 1000 одновременных пользователей.

2. \*\*Безопасность:\*\*

> Все данные передаются по протоколу HTTPS, пароли хранятся в хешированном виде (алгоритм bcrypt).

3. \*\*Доступность:\*\*

> Система должна быть доступна 99.9% времени в месяц (SLA).

4. \*\*Масштабируемость:\*\*

> Приложение должно поддерживать горизонтальное масштабирование (до 10 серверов).

5. \*\*Юзабилити:\*\*

> Интерфейс должен соответствовать стандарту WCAG 2.1 (контрастность текста не менее 4.5:1).

---

### \*\*2.3. Как описать пользовательские сценарии (use cases) в ТЗ\*\*

Use case — последовательность действий пользователя для достижения цели.

\*\*Структура описания:\*\*

1. \*\*Название сценария\*\* (например, "Оформление заказа").

2. \*\*Актор\*\* (кто выполняет: пользователь, администратор).

3. \*\*Предусловия\*\* (что должно быть выполнено до начала сценария).

4. \*\*Основной поток\*\* (шаги от начала до успешного завершения).

5. \*\*Альтернативные потоки\*\* (например, отмена заказа).

6. \*\*Постусловия\*\* (результат выполнения).

\*\*Пример для интернет-магазина:\*\*

```markdown

\*\*Сценарий:\*\* Оплата заказа

\*\*Актор:\*\* Покупатель

\*\*Предусловия:\*\* Пользователь авторизован, товар добавлен в корзину.

\*\*Основной поток:\*\*

1. Пользователь нажимает "Оформить заказ".

2. Система отображает форму выбора способа оплаты (карта, PayPal).

3. Пользователь вводит данные карты и подтверждает оплату.

4. Система проверяет данные через платежный шлюз.

5. При успешной оплате система отправляет чек на email.

\*\*Альтернативный поток:\*\*

- Если оплата отклонена, система показывает сообщение "Ошибка оплаты" и предлагает повторить.

\*\*Постусловия:\*\* Заказ переходит в статус "Оплачен".

```

---

### \*\*2.4. Как избежать двусмысленностей в требованиях\*\*

\*\*Проблемы:\*\*

- Слова "быстро", "удобно", "современный" — субъективны.

- Неявные условия (например, "система должна работать всегда").

\*\*Способы устранения:\*\*

1. \*\*Используйте шаблоны:\*\*

- \*\*SMART-критерии:\*\*

> Требование должно быть Specific (конкретным), Measurable (измеримым), Achievable (достижимым), Relevant (релевантным), Time-bound (ограниченным по времени).

- \*\*Given-When-Then\*\* (для сценариев):

> \*\*Given\*\* пользователь авторизован, \*\*When\*\* он нажимает "Удалить аккаунт", \*\*Then\*\* система запрашивает подтверждение.

- \*\*RFC 2119\*\* для уровней обязательности:

> "Должен" (обязательное требование), "Рекомендуется" (желательное), "Может" (опциональное).

2. \*\*Избегайте абстракций:\*\*

- ❌ \*"Интерфейс должен быть красивым"\* → ✅ \*"Шрифт основного текста — Roboto 14px, кнопки в стиле Material Design"\*.

3. \*\*Добавьте глоссарий:\*\*

- Определите термины (например, "Пользователь — лицо, зарегистрированное в системе").

4. \*\*Используйте диаграммы:\*\*

- UML-диаграммы, BPMN, схемы экранов.

5. \*\*Проверяйте тестируемостью:\*\*

- Каждое требование должно иметь критерий приемки.

> Пример:

> Требование: "Поиск выдает результаты за 1 секунду".

> Критерий: "При нагрузке 100 запросов/сек 95% ответов должны занимать ≤1 сек".

---

\*\*Итог:\*\*

- Функциональные требования — \*\*что делает система\*\*, нефункциональные — \*\*как она это делает\*\*.

- Use cases описывайте через шаги и условия.

- Избегайте двусмысленностей с помощью шаблонов (SMART, RFC 2119) и измеримых метрик.

Блок 3.

---

### \*\*3.1. Как описать архитектуру системы в ТЗ? Какие диаграммы использовать\*\*

Архитектура в ТЗ должна давать общее понимание структуры системы, взаимодействия компонентов и технологий.

\*\*Что включить:\*\*

- \*\*Тип архитектуры\*\* (монолит, микросервисы, серверless).

- \*\*Схема компонентов\*\* (база данных, сервер, клиентская часть).

- \*\*Технологический стек\*\* (языки программирования, фреймворки, СУБД).

\*\*Диаграммы:\*\*

1. \*\*UML (Unified Modeling Language):\*\*

- \*\*Диаграмма компонентов\*\* — показывает связи между модулями (например, API ↔ база данных).

- \*\*Диаграмма последовательностей\*\* — визуализирует взаимодействие объектов в сценарии (например, процесс оплаты).

- \*\*Диаграмма классов\*\* — для ООП-систем (описывает классы и их отношения).

2. \*\*BPMN (Business Process Model and Notation):\*\*

- Используется для описания бизнес-процессов (например, обработка заказа от клиента до доставки).

3. \*\*ER-диаграмма\*\* (Entity-Relationship):

— Показывает структуру базы данных (таблицы, связи).

\*\*Пример описания архитектуры:\*\*

> Система построена на микросервисной архитектуре. Основные компоненты:

> - \*\*API Gateway\*\* (Node.js) — маршрутизация запросов.

> - \*\*Сервис аутентификации\*\* (Python, JWT).

> - \*\*База данных\*\* (PostgreSQL, репликация Master-Slave).

> ![Диаграмма компонентов](пример ссылки на изображение).

---

### \*\*3.2. Нужно ли включать требования к интерфейсу (UI/UX)? Как их описать\*\*

\*\*Да\*\*, требования к UI/UX критичны для пользовательского опыта.

\*\*Что указать:\*\*

1. \*\*Стиль и гайдлайны:\*\*

> Интерфейс должен соответствовать Material Design 3.0. Шрифт: Roboto, размер заголовков — 24px.

2. \*\*Макеты экранов:\*\*

- Приложить ссылки на Figma/Adobe XD или описания ключевых экранов (например, "Главная страница содержит поисковую строку и блок рекомендаций").

3. \*\*Доступность (Accessibility):\*\*

> Соответствие стандарту WCAG 2.1 AA: контрастность текста ≥ 4.5:1, поддержка screen readers.

4. \*\*Интерактивность:\*\*

> Анимация загрузки данных длится не более 0.3 сек.

\*\*Пример:\*\*

> \*\*Форма регистрации:\*\*

> - Поля: email, пароль (скрыт точками), кнопка "Зарегистрироваться".

> - Валидация email в реальном времени (проверка формата).

> - При ошибке отображается сообщение красным цветом под полем.

---

### \*\*3.3. Как прописать требования к безопасности (security)\*\*

Требования к безопасности должны защищать данные и предотвращать угрозы.

\*\*Что включить:\*\*

1. \*\*Аутентификация и авторизация:\*\*

> Многофакторная аутентификация для администраторов. Роли: пользователь, модератор, админ (на основе RBAC).

2. \*\*Шифрование:\*\*

> Все данные передаются по TLS 1.3. Пароли хранятся в хешированном виде (алгоритм Argon2).

3. \*\*Защита от атак:\*\*

> Реализация CSRF-токенов, лимит запросов (100 в минуту с одного IP).

4. \*\*Стандарты соответствия:\*\*

> GDPR для обработки персональных данных, PCI DSS для платежей.

5. \*\*Резервное копирование:\*\*

> Ежедневные бэкапы баз данных с хранением на отдельном сервере (7 дней).

\*\*Пример требования:\*\*

> "При 5 неудачных попытках входа аккаунт блокируется на 15 минут, уведомление отправляется на email".

---

### \*\*3.4. Как указать требования к интеграции с другими системами\*\*

Интеграции описываются через форматы данных, протоколы и условия взаимодействия.

\*\*Что указать:\*\*

1. \*\*Форматы обмена данными:\*\*

> API использует JSON. Даты в формате ISO 8601 (2024-08-15T14:30:00Z).

2. \*\*Протоколы:\*\*

> REST API с HTTPS. Поддержка WebSocket для уведомлений.

3. \*\*Спецификация API:\*\*

- Endpoints, методы (GET/POST), примеры запросов/ответов.

- Обработка ошибок (HTTP-статусы 4xx/5xx с JSON-описанием).

4. \*\*Лимиты и SLA:\*\*

> Не более 100 вызовов API в минуту на один аккаунт. Время ответа API ≤ 500 мс.

5. \*\*Сторонние сервисы:\*\*

> Интеграция с Google Maps API для отображения геолокации.

\*\*Пример:\*\*

```markdown

\*\*Интеграция с платежным шлюзом:\*\*

- \*\*Endpoint:\*\* `POST /api/payments`

- \*\*Параметры запроса:\*\*

```json

{ "amount": 1000, "currency": "RUB", "user\_id": "123" }

```

- \*\*Ответ при успехе:\*\*

```json

{ "status": "success", "transaction\_id": "TX-2024-ABCD" }

```

- \*\*Ошибки:\*\* 400 — неверные данные, 503 — шлюз недоступен.

```

Блок 4.

---

### \*\*4.1. Как на основе ТЗ оценить трудоемкость разработки? Методы\*\*

Оценка трудоемкости зависит от детализации ТЗ. Основные методы:

1. \*\*Декомпозиция задач\*\*

Разбейте функциональные требования на подзадачи и оцените время для каждой.

Пример:

- Требование: "Регистрация через email и соцсети".

- Подзадачи:

- Верстка формы (8 часов).

- Интеграция с Google API (12 часов).

- Тестирование (6 часов).

2. \*\*Оценка по аналогии\*\*

Используйте данные из похожих проектов. Например, если в прошлом проекте интеграция с API заняла 20 часов, новая оценка будет близка к этой цифре.

3. \*\*Метод PERT (Program Evaluation and Review Technique)\*\*

Рассчитайте оценку по формуле:

> \*\*Время = (Оптимистичный + 4×Реалистичный + Пессимистичный) / 6\*\*

Пример:

- Оптимистично: 10 часов.

- Реалистично: 16 часов.

- Пессимистично: 24 часа.

- Итог: (10 + 4×16 + 24) / 6 = 17.3 часа.

4. \*\*Story Points (в agile)\*\*

Оценивайте сложность задач в условных единицах (например, по шкале Фибоначчи: 1, 2, 3, 5, 8).

Пример:

- "Разработка корзины покупок" — 8 story points.

5. \*\*Экспертные оценки\*\*

Привлеките разработчиков и архитекторов для оценки времени на каждое требование.

\*\*Ошибки:\*\*

- ❌ "На весь модуль авторизации — 2 недели" → ✅ Декомпозируйте на подзадачи.

- ❌ Игнорирование рисков → ✅ Добавьте буфер (15-20% от общего времени).

---

### \*\*4.2. Нужно ли включать в ТЗ этапы разработки и сроки? Как оформить\*\*

\*\*Да\*\*, этапы и сроки помогают синхронизировать команду и заказчика.

\*\*Как оформить:\*\*

1. \*\*Структура этапов:\*\*

- Анализ требований.

- Проектирование архитектуры.

- Разработка MVP.

- Тестирование (юнит-тесты, нагрузочные тесты).

- Внедрение и документирование.

2. \*\*Формат:\*\*

- \*\*Таблица с вехами (мильными камнями):\*\*

| Этап | Срок | Ответственный |

|-------------------------|----------------|---------------|

| Прототип UI/UX | 15.09.2024 | Дизайнер |

| Реализация API | 30.09.2024 | Бэкенд-разработчик |

| Тестирование безопасности | 10.10.2024 | QA-инженер |

- \*\*Диаграмма Ганта\*\* (визуализация сроков в Jira, Trello, Excel).

3. \*\*Условия:\*\*

- Укажите, что сроки могут корректироваться при изменении требований.

- Добавьте буфер на непредвиденные задержки (10-15%).

---

### \*\*4.3. Как прописать критерии приемки (acceptance criteria)\*\*

Критерии приемки — это условия, при которых требование считается выполненным.

\*\*Правила формулировки:\*\*

- Используйте шаблон \*\*Given-When-Then\*\*.

- Связывайте каждый критерий с конкретным требованием из ТЗ.

- Делайте критерии \*\*тестируемыми\*\*.

\*\*Примеры:\*\*

1. \*\*Для функционального требования:\*\*

> \*\*Требование:\*\* Пользователь может сбросить пароль через email.

> \*\*Критерий приемки:\*\*

> - Given: Пользователь нажимает "Забыли пароль?".

> - When: Вводит email, зарегистрированный в системе.

> - Then: Система отправляет письмо со ссылкой для сброса пароля в течение 2 минут.

2. \*\*Для нефункционального требования:\*\*

> \*\*Требование:\*\* Время отклика API ≤ 500 мс.

> \*\*Критерий приемки:\*\* При нагрузке 1000 запросов/сек 95% ответов укладываются в 500 мс (тест JMeter).

3. \*\*Для UI/UX:\*\*

> \*\*Требование:\*\* Кнопка "Купить" должна быть видна на всех экранах.

> \*\*Критерий приемки:\*\* На разрешениях от 320px до 1920px кнопка отображается в правом нижнем углу (проверка в Chrome DevTools).

\*\*Ошибки → Исправление:\*\*

- ❌ \*"Кнопка должна работать"\* → ✅ \*"При нажатии на кнопку 'Отправить' форма валидирует поля и выводит сообщение об успехе"\*.

Блок 5.

---

### \*\*5.1. Как описать требования к документации в ТЗ\*\*

Документация — обязательная часть проекта. В ТЗ нужно указать, какие виды документации требуются, их структуру и стандарты оформления.

#### \*\*Типы документации:\*\*

1. \*\*Пользовательская документация:\*\*

- Руководство пользователя (инструкции, скриншоты).

- FAQ или раздел помощи в интерфейсе.

2. \*\*Техническая документация:\*\*

- API-документация (Swagger/OpenAPI).

- Описание архитектуры и кода (для разработчиков).

- Документация по развертыванию и настройке.

#### \*\*Требования к документации:\*\*

- \*\*Формат:\*\* PDF, онлайн-вики (Confluence), Markdown.

- \*\*Язык:\*\* Русский/английский (или иной, если требуется).

- \*\*Структура:\*\* Оглавление, глоссарий, поиск по разделам.

- \*\*Обновления:\*\* Документация должна синхронизироваться с релизами ПО.

\*\*Пример:\*\*

> \*\*Пользовательская документация:\*\*

> - Руководство включает разделы: "Регистрация", "Оплата", "Настройки профиля".

> - Все скриншоты актуальны для версии 2.0.

> \*\*Техническая документация:\*\*

> - Swagger-спецификация для REST API с примерами запросов.

\*\*Ошибка → Исправление:\*\*

- ❌ \*"Документация должна быть понятной"\* → ✅ \*"Руководство пользователя включает пошаговые инструкции с анимированными GIF-скриншотами"\*.

---

### \*\*5.2. Нужно ли включать в ТЗ требования к тестированию? Как это сделать\*\*

\*\*Да\*\*, требования к тестированию снижают риски дефектов и формализуют процесс проверки.

#### \*\*Что включить:\*\*

1. \*\*Виды тестирования:\*\*

- Функциональное (проверка сценариев из ТЗ).

- Нагрузочное (например, JMeter для 5000 пользователей).

- Тестирование безопасности (сканирование уязвимостей OWASP Top 10).

2. \*\*Стандарты и инструменты:\*\*

> Автоматические тесты покрывают 80% кода (метрика JaCoCo). Инструменты: Selenium, Postman.

3. \*\*Критерии успешности:\*\*

> Все критические баги (Severity 1) должны быть исправлены перед релизом.

\*\*Пример требования:\*\*

> \*\*Тестирование UI:\*\*

> - Проверка отображения на браузерах Chrome, Safari, Firefox (последние 2 версии).

> - Адаптивность: экраны 320px, 768px, 1920px.

\*\*Ошибка → Исправление:\*\*

- ❌ \*"Система должна быть протестирована"\* → ✅ \*"Нагрузочное тестирование: 1000 RPS, время ответа ≤ 1 сек, процент ошибок ≤ 0.5%"\*.

---

### \*\*5.3. Как прописать условия поддержки и сопровождения ПО\*\*

Условия поддержки фиксируют обязательства исполнителя после сдачи проекта.

#### \*\*Что указать:\*\*

1. \*\*Сроки поддержки:\*\*

> Гарантийный период — 12 месяцев с момента приемки.

2. \*\*Исправление багов:\*\*

- Критические баги (система не работает) — исправление в течение 24 часов.

- Некритические (не влияют на функционал) — до 14 дней.

3. \*\*Обновления:\*\*

> Ежеквартальные обновления безопасности. Миграция на новые версии ОС/библиотек.

4. \*\*SLA (Service Level Agreement):\*\*

| Параметр | Значение |

|------------------------|------------------------|

| Доступность системы | 99.9% в месяц |

| Время реакции на запрос | 2 часа (в рабочее время) |

5. \*\*Процедура обращений:\*\*

> Запросы принимаются через тикет-систему (Jira, Redmine). Приоритет определяется согласно классификации багов.

\*\*Пример:\*\*

> \*\*Поддержка версий:\*\*

> - Исправления безопасности предоставляются для версий, выпущенных в течение последних 2 лет.

\*\*Ошибка → Исправление:\*\*

- ❌ \*"Мы поддерживаем систему после релиза"\* → ✅ \*"Гарантийная поддержка включает 3 бесплатных правки в месяц в течение 6 месяцев"\*.

Блок 6.