**Лекция 12**

**Проектирование программных систем**

Определение проекта ПС

Это набор схем, диаграмм, ТЗ и другой документации, содержащей описание программного продукта в объёме, достаточном для его конструирования.

Средства проектирования: блок-схемы, ER-диаграммы и CASE-диаграммы.

Проектированию обычно подлежат:

* [Архитектура ПО](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F);
* Устройство компонентов ПО;
* [Пользовательские интерфейсы](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F).

Участники процесса и их роли (основные):

1. Заказчик.
2. Руководитель проекта
3. Системный администратор
4. Администратор базы данных
5. Бизнес-аналитик
6. Системный аналитик
7. Тестировщик
8. ?? Задача документирования

В российской практике проектирование ведется поэтапно в соответствии со стадиями, регламентированными ГОСТ 2.103-68 :

Техническое задание (по ГОСТ 2.103-68 к стадиям разработки не относится),

Техническое предложение,

Эскизный проект,

Технический проект,

Рабочий проект.

**2. Техническое задание**

Основные стандарты, методологии и своды знаний, где упоминается ТЗ или SRS (Software (or System) Requirements Specification):

• ГОСТ 34

• ГОСТ 19

• IEEE STD 830-1998

• ISO/IEC/ IEEE 29148-2011

• RUP

• SWEBOK, BABOK и пр.

**2.1 ГОСТ 34**

**(**<http://www.rugost.com/index.php?option=com_content&view=article&id=96:gost-34602-89&catid=22&Itemid=53> **)**

Регламентирует структуру ТЗ на создание именно СИСТЕМЫ, в которую входят ПО, аппаратное обеспечение, люди, которые работают с ПО, и автоматизируемые процессы.

Согласно ГОСТ 34 техническое задание должно включать следующие разделы:  
  
1. Общие сведения  
2. Назначение и цели создания (развития) системы  
3. Характеристика объектов автоматизации  
4. Требования к системе  
5. Состав и содержание работ по созданию системы  
6. Порядок контроля и приемки системы  
7. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие  
8. Требования к документированию  
9. Источники разработки

При разработке ТЗ для государственных проектов Заказчики, как правило, требуют соблюдение именно этого стандарта.

**2.2 ГОСТ 19**

“ГОСТ 19.ххх Единая система программной документации (ЕСПД)” — это комплекс государственных стандартов, устанавливающих взаимоувязанные правила разработки, оформления и обращения программ (или ПО) и программной документации. Т.е. этот стандарт относится к разработке именно ПО.

Согласно [ГОСТ 19.201-78 Техническое задание, требования к содержанию и оформлению](http://www.rugost.com/index.php?option=com_content&view=article&id=46:19001-77&catid=19&Itemid=50) техническое задание должно включать следующие разделы:  
  
1. Введение;  
2. Основания для разработки;  
3. Назначение разработки;  
4. Требования к программе или программному изделию;  
5. Требования к программной документации;  
6. Технико-экономические показатели;  
7. Стадии и этапы разработки;  
8. Порядок контроля и приемки;  
9. Приложения.

Естественно ГОСТ 34 (и 19) уже устарели, и **я не люблю их использовать**, но при правильном интерпретации стандартов, можно получить хорошее ТЗ.

**3.IEEE STD 830-1998**

Достаточно хорошее определение стандарта [830-1998 — IEEE Recommended Practice for Software Requirements Specifications](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D1%82%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#.D0.A0.D0.B5.D0.BA.D0.BE.D0.BC.D0.B5.D0.BD.D0.B4.D1.83.D0.B5.D0.BC.D0.B0.D1.8F_.D1.81.D1.82.D0.B0.D0.BD.D0.B4.D0.B0.D1.80.D1.82.D0.BE.D0.BC_IEEE_830.5B1.5D_.D1.81.D1.82.D1.80.D1.83.D0.BA.D1.82.D1.83.D1.80.D0.B0_SRS) дано в самом его описании:

Описывается содержание и качественные характеристики правильно составленной спецификации требований к программному обеспечению (SRS) и приводится несколько шаблонов SRS. Данная рекомендуемая методика имеет своей целью установление требований к разрабатываемому программному обеспечению, но также может применяться, чтобы помочь в выборе собственных и коммерческих программных изделий.  
  
 Согласно стандарту, техническое задание должно включать следующие разделы:  
  
**1. Введение**

* 1. Назначение
* 2. Область действия
* 3. Определения, акронимы и сокращения
* 4. Ссылки
* 5. Краткий обзор

**2. Общее описание**

* 1. Взаимодействие продукта (с другими продуктами и компонентами)
* 2. Функции продукта (краткое описание)
* 3. Характеристики пользователя
* 4. Ограничения
* 5. Допущения и зависимости

**3. Детальные требования (могут быть организованы по-разному, например, так):**

* 1. Требования к внешним интерфейсам
  + 1. Интерфейсы пользователя
  + 2. Интерфейсы аппаратного обеспечения
  + 3. Интерфейсы программного обеспечения
  + 4. Интерфейсы взаимодействия
* 2. Функциональные требования
* 3. Требования к производительности
* 4. Проектные ограничения (и ссылки на стандарты)
* 5. Нефункциональные требования (надежность, доступность, безопасность и пр.)
* 6. Другие требования

**4. Приложения**  
**5. Алфавитный указатель**  
 На самом деле новичку достаточно трудно понять, что должно содержаться в данных разделах по вышеприведенной структуре (как и в случае с ГОСТом), поэтому нужно читать сам стандарт.  
Мне же больше нравится адаптированный шаблон Карла Вигерса, который я использую при разработки ТЗ для коммерческих компаний

ISO/IEC/ IEEE 29148-2011  
[Стандарт IEEE 29148-2011](https://www.iso.org/standard/45171.html) обеспечивает единую трактовку процессов и продуктов, используемых при разработке требований на протяжении всего жизненного цикла систем и программного обеспечения. Он приходит на смену стандартов IEEE 830-1998, IEEE 1233-1998, IEEE 1362-1998.  
  
Данный стандарт содержит два шаблона спецификации требований:  
  
• System requirements specification (SyRS)  
• Software requirements specification (SRS)  
  
System Requirements Specification (SyRS) определяет технические требования для выбранной системы и удобства взаимодействия предполагаемой системы и человека. Она определяет высокоуровневые требования к системе с точки зрения предметной области, а также информацию об общей цели системы, ее целевой среде и ограничениях, допущениях и нефункциональных требованиях. Она может включать в себя концептуальные модели, спроектированные для иллюстрации содержания системы, сценариев использования, основных сущностей предметной области, данных, информаций и рабочих процессов. Из определения следует, что это аналог ТЗ, описанного в ГОСТ 34.  
  
SyRS может содержать следующие разделы:  
  
1. Введение

* 1. Назначение системы
* 2. Содержание системы (границы системы)
* 3. Обзор системы
  + 1. Содержание системы
  + 2. Функции системы
  + 3. Характеристики пользователей
* 4. Термины и определения

2. Ссылки  
  
3. Системные требования

* 1. Функциональные требования
* 2. Требования к юзабилити
* 3. Требования к производительности
* 4. Интерфейс (взаимодействие) системы
* 5. Операции системы
* 6. Состояния системы
* 7. Физические характеристики
* 8. Условия окружения
* 9. Требования к безопасности
* 10. Управление информацией
* 11. Политики и правила
* 12. Требования к обслуживанию системы на протяжении ее жизненного цикла
* 13. Требования к упаковке, погрузке-разгрузки, доставке и транспортировке

4. Тестирование и проверка (список необходимых приемочных тестов, которые отражают зеркально раздел 3)  
  
5. Приложения

* 1. Предположения и зависимости
* 2. Аббревиатуры и сокращений

SRS это спецификация требований для определенного программного изделия, программы или набора программ (продукт), которые выполняют определенные функции в конкретном окружении. Из определения следует, что это аналог ТЗ, описанного в ГОСТ 19, а по структуре очень напоминает SRS из стандарта IEEE 830.  
  
SRS может содержать следующие разделы:  
  
1. Введение

* 1. Назначение
* 2. Содержание (границы)
  + 3. Обзор продукта
  + 1. Взаимодействие продукта (с другими продуктами и компонентами)
  + 2. Функции продукта (краткое описание)
  + 3. Характеристики пользователей
  + 4. Ограничения
* 4. Термины и определения

2. Ссылки  
  
3. Детальные требования

* 1. Требования к внешним интерфейсам
* 2. Функции продукта
* 3. Требования к юзабилити
* 4. Требования к производительности
* 5. Требования к логической структуре БД
* 6. Ограничения проектирования
* 7. Системные свойства ПО
* 8. Дополнительные требования

4. Тестирование и проверка (список необходимых приемочных тестов, которые отражают зеркально раздел 3)  
  
5. Приложения

* 1. Предположения и зависимости
* 2. Аббревиатуры и сокращений

Данный стандарт достаточно сложно найти в открытом виде в Интернете, но постараться можно, и опять же только на англ.

**RUP**  
Структура SRS в [RUP(Rational Unified Process)](https://ru.wikipedia.org/wiki/Rational_Unified_Process) представляет собой документ, в котором необходимо описать артефакты, полученные в процессе специфицирования требований.  
  
[Шаблон SRS в RUP](http://dit.isuct.ru/Publish_RUP/index.htm#core.base_rup/workproducts/rup_software_requirements_specification_934E66F.html) адаптирован из стандарта IEEE STD 830 и содержит два варианта:  
  
• Традиционный шаблон SRS со структурированными функциональными требованиями по функциям Системы, максимально похож на 830 стандарт.  
• Упрощенный шаблон SRS со структурированными функциональными требованиями в виде вариантов использования (use cases):  
  
1. Введение.

* 1. Цель.
* 2. Краткая сводка возможностей.
* 3. Определения, акронимы и сокращения.
* 4. Ссылки.
* 5. Краткое содержание.

2. Обзор системы

* 1. Обзор вариантов использований.
* 2. Предположения и зависимости.

3. Детальные требований

* 1. Описание вариантов использования.
* 2. Дополнительные требования.
* 3. Другие функциональные требования.
* 4. Нефункциональные требования.

4. Вспомогательная информация.  
  
Естественно, что в [Интернете можно найти шаблон и примеры SRS от RUP](https://www.google.ru/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=rup+srs+template&*).

**SWEBOK**.  
 [SWEBOK](https://www.computer.org/web/swebok), а также множество других методологий разработки ПО и сводов знаний при упоминании SRS ссылаются на вышеупомянутые зарубежные стандарты.  
Также стоит сказать, что для описания требований к АС и ПО используются и другие виды документов, кот каждый называет по разному: FRD (Functional Requirements Document), RD (Requirements Document), ПЗ (Постановка задачи или Пояснительная записка) и пр. Но это все производные документы от вышеупомянутых стандартов, не имеющих отраслевой стандартизации, хотя, в некоторых случаях, уже и с устоявшейся терминологией.

**А как же Agile?**

Одна фраза из [Манифеста Agile](http://agilemanifesto.org/): “Working software over comprehensive documentation”. Поэтому в Agile документации отводится совсем мало места.

Разработать АС без ТЗ можно (используя техники/рекомендации Agile), но вот в дальнейшем сопровождать — невозможно. Поэтому сразу задумайтесь, как вы будете писать ТЗ и другую документацию, при разработке ПО по Agile.

**Заключение**

Каждому проекту свое техническое задание. При правильном использовании любого из вышеперечисленных стандартов можно брать эти шаблоны для написания ТЗ, естественно адаптируя их под себя.  
  
 Содержание (наполнение) в ТЗ — самое главное! Он-лайн курс [Разработка и управление требованиями к ПО](https://www.webursitet.ru/product/kratkiy-kurs-sistemnogo-analiza.html).

 [пример ТЗ, который я писал много лет назад](https://drive.google.com/file/d/0B5xz4Vc-CUY1NjVoT0pFcnlEQU0/view?usp=sharing) (сейчас уже просто аналитиком давно не работаю, да и другие более удачные примеры запрещает открывать на всеобщее обозрение NDA).  
   
Также рекомендую ознакомиться со следующими материалами:

* Презентацией Юрия Булуя [Классификация требований к программному обеспечению и ее представление в стандартах и методологиях](http://2006.secr.ru/upload/files/63.pdf).
* Анализ требований к автоматизированным информационным системам. [Лекция 11: Документирование требований](http://www.intuit.ru/studies/courses/2188/174/lecture/4732).
* [Правила составления Software requirements specification](https://habrahabr.ru/post/52681/) (читать вместе с комментариями)
* [Примеры ТЗ и другой документации по разработке АС](http://aisup.economy.gov.ru/pubportal/) для МЭР
* [ГОСТ-овский стиль управления](http://gaperton.livejournal.com/49867.html). Статья Gaperton по правильной работе с ТЗ по ГОСТ

**Гостовский стиль управления**

<https://gaperton.livejournal.com/49867.html>

Многим, кто работал по ГОСТ-19 или 34, или по другим ГОСТ вариациям ЕСКД, этот стиль хорошо знаком. Он значительно отличается от западной "классики", хоть и сам является не меньшей классикой.

Многие из тех, кто с ним сталкивался, применяют его для внешнего взаимодействия, и считают формальностью и неизбежным злом.

Однако, в некоторых организациях, он же применяется и для управления работами сотрудников. Часто такое бывает, когда многие сотрудники работают по контрактам.  
 И вот тогда, когда вы не только выставляете "внешний интерфейс" в виде ГОСТ-19, **но и сами выдаете задания по его правилам - становится понятна вся его сила и весь смысл**.

Это очень простой, чрезвычайно эффективный, и, вероятно, наиболее недооцениваемый стиль из существующих. И этот стиль по-настоящему крут. Сейчас разберем его **отличительные особенности** по практике его применения. Без шелухи и формализма - только суть и дух только базовые принципы, общие для всех ГОСТ, посвященных организации работ.

Первое, и главное. **Единицей планирования** в ГОСТ является ТЗ Техническое задание.

Вокруг предназначения и сути этого документа много непонимания. Многие путают его с Requirements Specification. Но это НЕ Requirement Specification. Начать можно с того, что правильно составленное ТЗ обычно гораздо короче, и в секции "технические требования" содержит в **основном ключевые требования, определяющие успех всей работы**. Почему? Потому, что эта секция - только одна из секций документа, **который суть "Задание", а не требования.**

**Вторая главная секция** - **поэтапный план работ**. Это простая таблица - последовательность этапов выполнения работы. В ГОСТ есть список стадий и этапов, **но этот список носит рекомендательный характер.** На практике - у вас есть полная свобода в определении количества этапов, их названий, и состава работ.

Итак, что такое "этап"? **Этап имеет срок окончания**, название, описание результатов этапа, и (**наименее важная, информационная составляющая**) - **перечень видов работ-активностей, которые выполняются в рамках этапа**. Если оплата работ сдельная - в этой же таблице стоят суммы за каждый этап, и ТЗ - является приложением к договору подряда.  
 Почему активности - наименее важная составляющая? Потому, что ГОСТ не содержит **никаких** инструментов контроля, выполняются на самом деле эти активности, или нет**. Этап проверяется по результатам**.

А активности - указываются для того, чтобы заказчику было понятно, чем люди заниматься будут, и почему у этапа такие сумма и сроки.

Итак, что такое ТЗ? ТЗ, это комбинация ключевых требований, определяющих успех работ, и поэтапного плана данных работ. ТЗ - это единица планирования. Это не "требования". Это - **задание**.

ТЗ определяет, что должно быть сделано (ключевые признаки успеха проекта), в какие сроки, каковы промежуточные этапы (точки контроля). И кроме этого - он определяет требования к **проведению работ (дополнительные ограничения на выполнение**), к обеспечению работ и испытаний (что необходимо для выполнения работ и для проведения приемки - это обязан предоставить **выдающий задание**), а также - к проведению испытаний (обычно, там дается ссылка на документ «программа и методика испытаний»).

А еще в ТЗ есть очень короткая секция, в самом начале, с которой у большинства есть огромные проблемы. Но - в которой состоит весь дзен ТЗ. Называется "Цели и задачи работы". :)  
  
 В чем состоят проблемы? В неумении людей отличать цели от задач. Типичное содержимое - "целью работы является разработка системы Х. Задачами работы являются … Что здесь писать? являются проектирование, кодирование, и отладка системы Х".

Надо ли говорить, что разработка (т.е. процесс) не может являться целью работ? :) По опыту знаю - не только надо, но надо еще и пояснить. Смотрите:  
  
"Задача - достать денег в партийную кассу, предотвратив тем самым закрытие типографии" - таковы цели и задачи Грина в одном из эпизодов "Статского Советника" Акунина.  
  
"Задача - достать денег, с целью хорошо отдохнуть в Европе" - казалось бы, другая цель, но какая разница, если выполнять надо одну задачу - достать денег? А разница очень большая.  
  
 Типографию надо оплатить в течении трех дней. Более того - ради такой цели можно рискнуть очень многим, она высокоприоритетна. И - нужна вполне конкретная сумма.  
  
 Хорошо отдохнуть в Европе - цель не срочная, сильно рисковать ради ее достижения глупо, ибо отдыхать будет некому. Понятно? **Понимание цели дает исполнителю возможность додумать детали и расставить приоритеты**.  
  
 И в конце концов - раз у вас затруднение с формулировкой **цели** работ, и она дублирует задачи - может быть, работу вообще не стоит выдавать? Ну, раз выдающий задание не в состоянии внятно объяснить, зачем нужен результат работ? То, может, и работа не нужна?

Переходим на конкретику. Задача работы - получить работоспособную реализацию аудиокодека Dolby Digital.

**А вот с какой целью**? Проще выражаясь - зачем он нам нужен, ради чего вообще эта работа затеяна, для решения какой проблемы? Неважно, что вам это очевидно - **это необходимо довести до исполнителей**.  
  
 Если с целью испытаний архитектурного прототипа медиаплеера - это одно. Берем GPL-реализацию, и допиливаем. Главное в данной работе - архитектуру проверить как можно быстрее, а не ковыряться с одним из десятка кодека. .   
  
 Если же с целью использования в конечном продукте - тут уже совершенно другое дело. Абсолютно другие требования к качеству, и GPL лицензия не подойдет.  
  
 **Цель работ, т.е. решаемая работой проблема**, rationale, стоящее за задачей - **обязательно** должны быть обозначены явно. Для этого, в принципе, допустимо ввести отдельную "главу" в ТЗ - она называется как-то вроде "характеристика предметной области" (забыл, как она точно называется, см. ГОСТы в сети), где простым человеческим образом описать проблему. Можно это сделать в той же самой секции "цели и задачи". .   
  
 Здесь многие знатоки ЕСПД-ЕСКД со мной не согласятся. Ибо - общая практика такова, что к данной секции относятся наплевательски - лень мозги включать. Не без греха и автор этих строк - бывало, что писал в эту часть полную хрень. Но я скажу - вне всякого сомнения, "цели и задачи работы", вкупе с описанием решаемой проблемы - самая важная секция ТЗ. И умение отделять цели от задач - ключевое в составлении ТЗ и менеджменте вообще.  
  
Теперь два фокуса: .  
1) Вас могут не устраивать фиксированные даты окончания этапов. Пишете в соответствующей графе: "в соответствии с ведомостью исполнения". И все. .  
2) Вас может волновать отсутствие детально расписанных требований в документе ТЗ. Волноваться не надо - у вас есть «программа и методика испытаний» .

Этот документ - именно, что программа (то есть, тест-план), и методика (общие принципы построения тест-плана и организации тестирования). Этот документ - фиксирует требования в конструктивной форме. Вставляйте в него свои юз-кейсы и юзер-стори.

И включайте наличие первой версии этого документа в результат первого этапа. Далее - вы сможете отслеживать прогресс работ по количеству проходящих пунктов этого "документа". Вот и весь фокус.  
  
Вкратце - все. Но к самому интересному моменту мы только подошли. Вы видите ТЗ - это очень укрупненный план. Вот у вас есть ТЗ на весь проект. Что же делать дальше? Составлять WBS? Рисовать задачи?  
 Единственная "задача" - это ТЗ. **Правильный ответ - выделять частные, более мелкие ТЗ, на более мелкие работы**. И так - до самого мелкого уровня. Две главных секции - я показал. Это технические требования плюс поэтапный план.

Вот в этом и состоит второе важное отличие схемы ГОСТ от классической. **План работ - это совокупность заданий, на каждое из которых выписано ТЗ**. Результат каждого задания - конкретный и проверяемый. Проходит обязательную приемку. Задание может иметь промежуточные этапы - каждый из которых также проходит приемку.  
  
То есть, в отличии от классики, ГОСТ:  
1) Объединяет планирование и управление требованиями в единую технику.  
2) Явно фокусируется на результате работ, а не на процессах и активностях.

3) Опирается на одну и ту же технику при управлении программой и проектом. Он предполагает, что вы вместо выдачи "заданий", будете дробить крупные "проекты" на "подпроекты", концентрируясь на результатах каждой задачи и этапа.

4) Не предполагает отдельной роли "менеджера". Все, кто завязан в процесс - в той или иной степени инженеры. Задание-то "техническое", как ни крути.  
5) Совместим со стилем управления Auftragstaktik, про который я много писал.

Когда вы понимаете принцип, стоящий за ГОСТ (а ГОСТ были изначально разработаны для управления большими программами, вроде разработки комплексов ПВО), все становится очень просто и симпатично.

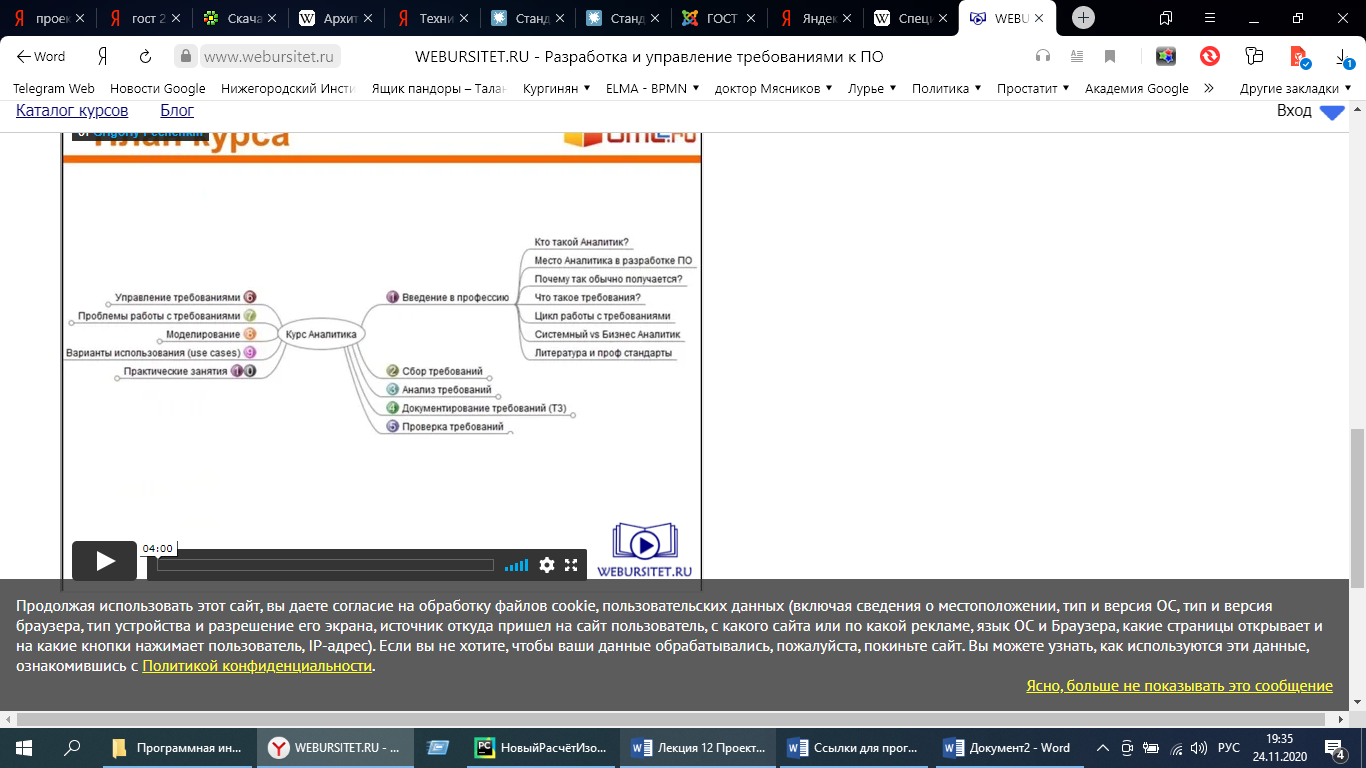
Настолько - что вы можете даже не писать документов-ТЗ. Честно. Смотрите: Берете Redmine: .   
1) Иерархия ТЗ - это иерархия проектов и подпроектов.  
2) Этапы - это "версии" для каждого проекта.  
3) Технические требования - можете в вики писать, а можете задавать тикетами. Это уж как душе угодно.  
  
 Вот вам и все управление. Просто, симпатично, и, самое главное - отлично работает.

Я даже скажу так. Это - одна из немногих техник, которая действительно работает на практике.

<https://habr.com/ru/post/328822/>

<https://habr.com/ru/post/139835/>

<https://habr.com/ru/company/it-grad/blog/276297/> Об архитектуре ПО



**Примеры ТЗ**

**Архитектура программного обеспечения**

**Архитектура программного обеспечения** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *software architecture*) — совокупность важнейших решений об организации программной системы. Архитектура включает:

* выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
* соединение выбранных элементов структуры и поведения во всё более крупные системы;
* архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение[[1]](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-1)[[2]](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-2).

[Документирование](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) архитектуры [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) (ПО) упрощает процесс коммуникации между разработчиками, позволяет зафиксировать принятые проектные решения и предоставить информацию о них эксплуатационному персоналу системы[[3]](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-_027dbb3c5573dbf5-3), [повторно использовать](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0) [компоненты](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) и [шаблоны](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) проекта в других.

Общепринятого определения «архитектуры программного обеспечения» не существует. Так, сайт [Software Engineering Institute](https://en.wikipedia.org/wiki/Software_Engineering_Institute" \o "en:Software Engineering Institute) приводит более 150 определений этого понятия[[4]](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-_3428190701c071f1-4)[[5]](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F#cite_note-5)