# Жизненный цикл проекта

## Основные понятия

***Жизненный цикл программного обеспечения*** — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Типичные этапы жизненного цикла проекта в пределах одной итерации разработки:

* постановка целей
* формирование команды
* уточнение требований
* исследование выполнимости
* проектирование
* планирование
* реализация
* тестирование
* стабилизация
* выпуск
* сопровождение

Этапы не обязательно чётко определены и разделены, они могут пересекаться во времени, пропускаться, объединяться, повторяться, и т. д.

## Этапы жизненного цикла

### Постановка целей

Цели формулируются заказчиком и документируются в формате, зависящем от методологии (от салфетки до спецификации в пяти томах). Цели учебного проекта сформированы заказчиком — компанией Apriorit — в виде общего описания проекта.

### Формирование команды

Обычно в состав команды входит как минимум руководитель, разработчики и тестировщики. Роль руководителя в вашей команде выполняет наставник. Роли разработчиков (и тестировщиков) — выполняете вы.

### Уточнение требований

Важно понимать, каким образом после реализации можно будет проверить, что цели проекта достигнуты. Одним из инструментов, полезных на этом этапе, являются *сценарии использования* (use cases). Также этот этап может привести к корректировке целей проекта.

### Исследование выполнимости

Кроме того, не менее важно понимать, возможно ли вообще достичь поставленных целей. На этом этапе проводят поиск и исследование аналогов; реализуют небольшие прототипы решений; исследуют окружение, в котором будет существовать продукт.

### Проектирование

На этом этапе следует определить, как должен выглядеть ваш продукт, чтобы он смог достичь поставленных целей, с учётом информации полученной на предыдущих этапах. Здесь хорошими инструментами могут послужить *диаграммы последовательности* (как описание реализации сценариев использования), *диаграммы модулей или классов* (как описание необходимых архитектурных решений). Исходя из описания архитектуры продукта следует составить стратегию его тестирования.

### Планирование

Перед началом инженерной работы над проектом важно составить структурированный план работ, определить дату завершения работ и необходимый бюджет. План поможет организовать график работы и распределить нагрузку между участниками таким образом, чтобы она была выполнена в полном объеме и в заданный срок.

### Реализация

Вся работа по реализации проекта обычно разбивается на несколько отдельный задач, каждая из которых проходит несколько этапов:

* написание кода
* коммит кода в репозиторий
* обзор кода (code review)
* рефакторинг
* подготовка сборки в тестирование

В результаты *работы над задачей* у вас получится один или несколько *коммитов* (не следует объединять несвязанные задачи в один коммит). Ваши изменения стоит показывать другим разработчикам (проводить *code review*). Другие разработчики могут предлагать вам улучшения (или вы сами можете заметить возможности для этого), для чего вам может потребоваться дополнительное время на *рефакторинг* вашего кода. В конце работы над задачей следует подготовить итоговую *сборку для тестирования*.

### Тестирование

Проверка программного продукта осуществляется на основании выработанных сценариев использования, по подготовленным ранее тест планам, в соответствии с принятой ранее стратегией тестирования. Результатом этого этапа будет список выявленных дефектов.

### Стабилизация

Задачей этого этапа является устранение дефектов, выявленных на предыдущем этапе, и недопущение появления новых дефектов. В любом случае, обычно требуется несколько повторений этих двух этапов (это стоит учитывать при планировании).

### Выпуск

После финального тестирования продукт отправляется заказчику. Продукт считается выпущенным, когда заказчик подтверждает, что продукт полностью удовлетворяет поставленным целям. В случае учебного проекта выпуском можно считать успешное завершение защиты проекта.

### Сопровождение

После выпуска к продукту могут появиться новые требования (которые потребуется удовлетворить), либо конечные пользователи могут выявить дефекты, пропущенные на предыдущих этапах (которые следует исправлять).

В случае учебного проекта этапа сопровождения не будет, но его следует учитывать при проектировании: среди всего прочего, продуманная стратегия обработка ошибок и автоматизированное тестирование упрощают сопровождение.

# Планирование работы

## Что следует включить в план

При планировании, кроме времени на реализацию, учитывайте время на:

* Написание документации, рисование диаграмм и прочая бумажная работа (если она ещё не была сделана на этапе проектирования)
* Рефакторинг после ревью наставником или коллегами.
* Подготовку тестовой среды
* Ознакомление с инструментами
* Проведение тестирования, юнит тесты, автотесты.
* Устранение ошибок и недочетов (багфикс)
* Подготовка презентации проекта
* Время на коммуникацию
* Так же стоит задуматься о рисках, которые могут случиться во время работы. Например, какая то функциональность может не сработать или повести себя не совсем так как вы ожидаете, кто то может заболеть и другое.

### Пример плана в MS Project



Шаблон живет тут:

<https://drive.google.com/open?id=1v31ZqQLAnsfKr0aKfpCnt_WCQVRNGB9w>

Подробная инструкция:

<https://docs.google.com/document/d/1IWk8IBCxRMFMZ3S_nYvKNs_QIhikxXBoa4AAj8J-t_Y>

# Разработка проекта

## Ревью изменений

***Code review*** — это анализ чужого кода, с целью выявить ошибки, соответствие кода поставленной задаче, следование общему стилю написания кода в проекте и не только.

### Зачем это нужно?

Как говорится, «одна голова хорошо, а две (или более) лучше». Code review помогает на раннем этапе выявить возможные ошибки в логике написанного кода, случайно пропущенные ошибки, которые не в состоянии показать компилятор. Другие участники проекта знакомятся с новым кодом и функциональностью, а также происходит обмен опытом и знаниями.

### Инструменты проведения code review

Самый примитивный способ: открыть код или коммит в IDE или на GitHub, посмотреть изменения, свои комментарии отправить автору любым удобным способом (чат, почта, телефон, голуби).

У GitHub есть встроенный инструмент для выполнения ревью с помощью *пулл-реквестов* (pull requests). Если вкратце: автор пишет код, размещает его в отдельной ветке, после чего создается пулл-реквест в общую ветку. Ревьюверы смотрят на предлагаемые изменения прямо на GitHub, с возможностью добавления комментариев прямо рядом с конкретными строчками кода. В этот же пулл-реквест добавляются правки, выполняемые для после ревью. Когда изменения больше не нуждаются в правках, код объединяется с общей веткой.

Подробнее можно почитать тут:

* [Proposing changes to your work with pull requests](https://help.github.com/articles/proposing-changes-to-your-work-with-pull-requests/)
* [Reviewing changes in pull requests](https://help.github.com/articles/reviewing-changes-in-pull-requests/)

## Система контроля версий (Git)

Определение термина “Система контроля версий” и зачем это нужно, можно почитать в [Википедии](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8).

[Документация Git](https://git-scm.com/book/ru/v2), на официальном сайте (на русском языке). **Must read:**

* [1.3 Основы Git](https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git)
* [1.5 Установка Git](https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git)
* [1.6 Первоначальная настройка Git](https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9F%D0%B5%D1%80%D0%B2%D0%BE%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-Git)
* [2.2 Запись изменений в репозиторий](https://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%97%D0%B0%D0%BF%D0%B8%D1%81%D1%8C-%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9-%D0%B2-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9)
* [2.3 Просмотр истории коммитов](https://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%9F%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2)
* [2.5 Работа с удалёнными репозиториями](https://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D1%83%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F%D0%BC%D0%B8)
* [3.2 Основы ветвления и слияния](https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B2%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D1%8F)
* [3.5 Удалённые ветки](https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5-%D0%B2%D0%B5%D1%82%D0%BA%D0%B8)
* [3.6 Перебазирование](https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%9F%D0%B5%D1%80%D0%B5%D0%B1%D0%B0%D0%B7%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

### Начало работы

Для начала, если Git не установлен на рабочей машине, его необходимо установить: [Git Downloads](https://git-scm.com/downloads). По ссылке также можно найти GUI клиенты под Linux, Mac, Windows.

Далее необходимо указать ваше имя и адрес электронной почты. Эта информация будет записываться в каждый ваш коммит.

$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com

Ключ --global означает, что Git будет использовать эти настройки для всех репозиториев. Чтобы указать другие параметры для конкретного репозитория, нужно зайти в каталог проекта и выполнить эти команды без флага --global.

### Эффективное использование git

Ключ к эффективному использованию системы контроля версий лежит в частом использовании git commit, но (относительно) редком использовании git push. Создание коммита (git commit) сохраняет ваши изменения на вашем компьютере. Отправка коммитов в удалённый репозиторий (git push) делает ваши изменения доступными другим участникам проекта, а также сохраняет их на удалённом сервере.

#### Когда делать git commit?

*Создавать новые коммиты можно и нужно относительно часто:* для каждого логически завершённого этапа отдельных задач, из которых состоит проект. Например, для каждого написанного метода (или класса, если они небольшие). За день работы у вас обязательно наберётся как минимум пара-тройка коммитов.

#### Когда делать git push?

Когда ваша задача по вашему мнению решена, коммиты следует опубликовать с помощью git push. После этого другие участники проекта смогут провести code review, когда вы создадите pull request.

Независимо от этого, *в конце рабочего дня следует выполнить git push*, даже если задача решена лишь частично. Это необходимо для создания «резервной копии» вашей работы на удалённом сервере, чтобы не потеряли всё, если с вашим компьютером случится что-то ужасное. В этом случае создавать pull request не следует.

#### Feature branches

Чтобы не мешать своими частыми коммитами другим участникам проекта, работу следует вести в отдельных ветках (*feature branches*), которые по мере завершения работы синхронизируются с главной веткой (*master*).

Пример использования git в течение типичного рабочего дня:

$ git checkout master # синхронизировать состояние master

$ git pull

$ git checkout -b new-feature # создать ветку для своей подзадачи

< работа >

$ git add -p # завершили этап №1

$ git commit

< ещё работа >

$ git add -p # завершили этап №2

$ git commit

$ git push origin new-feature # отправили сделанную на данный момент

# работу на удалённый сервер

### Что делать, если все сломалось?

Git может быть опасным в руках новичка. Бездумное выполнение непонятных команд, найденных в интернете, может привести к потере ваших изменений, порче репозитория, и сложностям с совместной работой.

Несколько советов, которые помогут избежать проблем:

* не выполняйте git rebase в ветках, над которыми работает кто-то кроме вас (например, в master)
* не выполняйте git push --force
* сложные действия вроде merge или rebase стоит сначала делать во временной ветке, которую при любом раскладе можно будет удалить и начать все заново

#### Reflog

С помощью команды git reflog можно посмотреть журнал изменений. И даже в случае, когда коммиты были утеряны из-за rebase/reset, их можно найти в reflog и восстановить. Подробнее: [Обслуживание и восстановление данных](https://git-scm.com/book/ru/v1/Git-%D0%B8%D0%B7%D0%BD%D1%83%D1%82%D1%80%D0%B8-%D0%9E%D0%B1%D1%81%D0%BB%D1%83%D0%B6%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B8-%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

#### Как отменить “git add” до коммита?

$ git reset имя\_файла

#### Как отменить последний коммит?

$ git reset --soft HEAD~1

При использовании ключа --soft изменения из последнего коммита появятся в staged статусе (как будто они были добавлены с помощью git add). Вы можете доделать работу и закоммитить все изменения позже.

При использовании ключа --hard изменения из последнего коммита, а также все ваши текущие изменения будут удалены.

# Тестирование проекта

***Тестирование*** — один из процессов разработки ПО, нацеленный на обеспечение качества продукта. Цель тестирования — убедится, что продукт соответствует описанным требованиям и может использоваться для поставленных целей, и определить дефекты в нем. Тестирование должно начинаться как можно раньше в жизненном цикле ПО, так как чем раньше найден дефект, тем быстрее и дешевле можно его исправить.

## Типы тестирования

**Функциональное тестирование** — основывается на тестировании функциональности, указанной в требованиях. Оно включает в себя также тестирование безопасности: тестирование способности ПО предотвращать доступ неавторизованных пользователей к внутренним данным или функциональности.

**Нефункциональное тестирование** — тестирование нефункциональных характеристик компонента или системы, таких как эффективность, надежность, переносимость, удобство использования и т. д.

**Тестирование, связанное с изменениями** — включает в себя повторное тестирование и регрессионное тестирование.

*Повторное тестирование* — тестирование, во время которого выполняются тесты, выявившие ошибки во время последнего запуска, для подтверждения успешности исправления ошибок.

*Регрессионное тестирование* — тестирование ранее протестированной программы, проводящееся после модификации. Его целью является подтверждение того, что процесс модификации не внес или не активизировал ошибки в областях, не подвергавшихся изменениям. Проводится после изменений в коде программного продукта или его окружения.

## Уровни тестирования

**Модульное тестирование** — проверка модуля приложения отдельно от всей системы. Проводится при помощи специализированных фреймворков и отладчиков. В рамках тестирования на вход модуля подаются тестовые данные и анализируется результат, получаемый после выполнения модуля.

**Интеграционное тестирование** — проверка взаимодействия между компонентами (компонентное-интеграционное тестирование), а также взаимодействия с различными частями системы: ОС, hardware и т. д. (системное интеграционное тестирование).

**Системное тестирование** — тестирование системы целиком. На этом этапе проводится как функциональное, так и нефункциональное тестирование. Существует два подхода к данному тестированию:

* *Основанный на требованиях* — для каждого требования спецификации пишется один или более тест-кейсов.
* *Основанный на сценариях использования* (use-cases) — требования описаны в виде сценариев использования, по которым и пишутся тест-кейсы.

**Приёмочное тестирование** — зачастую проводится на стороне заказчика. Проводится после достижения необходимого уровня качества ПО. В рамках данного тестирования определяется удовлетворяет ли система приемочным критериям и выносится решение о том, принимается ПО заказчиком, или нет.

## Тестовая документация

### Тест-план

***Тест-план*** — документ, описывающий весь объем работ по тестированию: описание объекта тестирования, стратегии, расписания, критерии начала и окончания тестирования, необходимое в процессе работы оборудование, специальные знания, а также оценку рисков с вариантами их разрешения.

Тест-план должен содержать следующие пункты:

1. Краткое описание объекта тестирования
2. Список компонентов и функциональности, которые будут тестироваться.
3. Описание стратегии тестирования, а именно: какие виды тестирования будут проводится? будет ли это только функциональное тестирование, или для системы будут проводиться и другие виды, например: нагрузочное тестирование, конфигурационное тестирование и т. д.
4. Описание и последовательность работ:
   1. подготовка к тестированию: написание тестов, настройка окружения;
   2. собственно тестирование: выполнение всех написанных тестов;
   3. анализ результатов: для проваленных тестов выписываются баг-репорты, а также оценивается, достигли ли мы критерия выхода из тестирования.
5. Критерии начала и окончания тестирования:

* Критериями начала тестирования могут быть наличие всей необходимой тестовой документации (всех тестов), окончание работ по разработке и т. д.
* Критериями окончания тестирования могут быть: проведение всех запланированных видов тестирования (если после этого не подразумевается исправление обнаруженных недостатков и повторное тестирование) или же проведение всех запланированных видов тестирования и отсутствие проблем уровня Blocker и High.

### Написание тестов

Как упоминалось выше, для каждого требования или сценария использования пишутся ***тест-кейсы***, описывающие совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функциональности или её части.

Имеет следующую структуру:

* **Summary:** короткое описание проверки
* **Preconditions:** список действий или условий по подготовке системы к проведению основной проверки (например: The application is installed on PC. UAC is turned off)
* **Steps:** список действий, которые приводят систему к необходимому состоянию для получения результата проверки
* **Expected result:** описание ожидаемого результата проверки.
* **Priority:** приоритет тест-кейса, обычно от 1 (наименьший) до 3 (наибольший). Показывает то, насколько прохождение данного тест-кейса важно для общего качества продукта.
* **Status:** статус прохождения тест-кейса. По умолчанию пустое поле. После прохождения теста ему присваивается статус *Passed*, если полученный результат совпадает с ожидаемым, или *Failed*, если полученный и ожидаемый результаты отличаются.

В первую очередь требования покрываются положительными тест-кейсами (кейсы, в которых используются корректные данные, над которыми система должна производить действия). Данные тест-кейсы имеют высокий приоритет.

Далее требования покрываются негативными тест-кейсами (кейсы в которых используются некорректные данные, чтобы убедится, что система правильно их обрабатывает и не приходит после них в неисправность). Обычно подобные тест-кейсы имеют низкий приоритет, но ошибки при их проверке могут нанести системе катастрофический вред. В связи с этим пренебрегать ими не стоит.

### Пример тест-кейса

Представим что у нас есть требование:

*Сервис должен автоматически запускаться после перезагрузки сервера.*

Для проверки данного требования нам понадобятся два тест-кейса:

* перезагрузка системы, когда сервис включен
* перезагрузка системы, когда сервис предварительно выключен.

Если описать их в таблице, выглядеть это будет следующим образом:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Summary | Preconditions | Steps | Expected Result | Priority |
| Restart system when service is running | Service is running | 1. Restart System  2. Open Task Manager -> Services  3. Check service status | Service is running | 3 |
| Restart system when service is not running | Service is not running | 1.Restart System  2. Open Task Manager -> Services  3. Check service status | Service is running | 2 |

Тест-кейсы, написанные для тестирования одного модуля, объединяются в *тестовые наборы* (test suite).

Обычно порядок прохождения тест-кейсов в наборе определяется по приоритету: в первую очередь проверяются тест-кейсы с приоритетом 3, затем — 2 и 1.

## Анализ результатов тестирования

По результатам прохождения тестов из набора составляется ***баг-репорт*** — документ описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Подобные отчёты состоят из следующих пунктов:

* **Summary:** краткое описание проблемы, которого достаточно, чтобы сразу понять в чем ошибка. Обычно кратко отвечает на вопросы: что произошло, при каких обстоятельствах или после каких действий и в каком компоненте системы.
* **Description:** подробное описание проблемы, состоит из:
  + *Preconditions:* условия, которые должны быть выполнены для возникновения проблемы
  + *Steps:* список шагов, необходимых для воспроизведения проблемы
  + *Actual Result:* непосредственно описание сути ошибки или проблемы
  + *Expected Result:* описание ожидаемого поведения системы в данной ситуации
  + *Attachments:* логи, дампы, скриншоты с проблемой, и т. д.
* **Environment:** описание окружения, на котором воспроизводится проблема
* **Build:** версия, в которой найден баг
* **Priority:** Приоритет бага, определяющий порядок исправления. Различают следующие приоритеты (от высокого к низкому):
  + *Blocker* — проблема блокирует дальнейшее тестирование системы. Исправляется в первую очередь!
  + *High* — ошибка влияет на основную бизнес-логику или же приводит систему в нерабочее состояние.
  + *Normal* — ошибка не сказывается на основной бизнес-логике, или есть обходные пути для использования функциональности системы.
  + *Low* — ошибка косметическая и не влияет на функциональность, или же воспроизводится с крайне низкой регулярностью.

В зависимости от выбранного критерия выхода из тестирования и приоритета баг-репортов принимается решение о том, какие из проблем необходимо исправлять. Естественно, что блокирующие проблемы обязательны для исправления, в то время как низкоприоритетные проблемы вполне можно не исправлять немедленно.

Баг-репорты отслеживаются в специальных баг-трекинговых системах, где у них всегда есть один статусов: *open, in progress, in testing, closed, reopened*; а также человек, ответственный за судьбу конкретного баг-репорта.

# Презентация проекта

Презентация будет представлена на проекторе.

Общее требование к оформлению слайдов:

* размер слайдов — 16:9;
* цвет текста должен быть контрастным: тёмный текст на темном фоне использовать не следует (и наоборот), он не читабельный;
* размер шрифта в теле слайда должен быть не меньше 28 пунктов — текст должен читаться с дальнего конца комнаты;
* каждый слайд должен содержать заголовок, у которого шрифт больше, чем в теле слайда;
* много текста на слайде не должно быть;
* использование иллюстраций приветствуется.

Содержание презентации:

* представление авторов, название проекта;
* в чём суть проекта, основные требования;
* существующие аналоги, их плюсы и минусы, чем ваш проект лучше;
* поддерживаемые ОС вашего продукта;
* что было в спецификации, что реализовали, что изменили (почему), что не успели;
* как вы распределили и организовали работу;
* насколько точным вышел план работ, составленный в начале;
* использованные технологии, общая архитектура проекта, какие паттерны использовали, и т. п.:
  + если есть: диаграммы классов, прецедентов, последовательности;
  + на диаграммах не следует перечислять все методы и все сущности — их плохо видно из дальнего конца зала;
  + интересные классы лучше вынести в отдельные слайды и рассказать про них детально;
* интересные понравившиеся штуки при реализации;
* демонстрация проекта;
  + как тестировался проект, какие проблемы нашли при тестировании;
  + общее впечатление, замечания, предложения, улучшения;
* использованные источники.