Требования, backlog и декомпозиция задачи

User story, критерии приёмки и границы MVP

Косило Павел Андреевич

5 сентября 2026

Содержание

Цель и задачи занятия

Цель

Из черновика Proposal получить проверяемый MVP и индивидуальный backlog — не список фич и не «сделать CRUD».

Задачи

- отличить FR от NFR и записать user story;
- задать критерии приёмки;
- зафиксировать границы MVP и 10–15 элементов backlog;
- положить это в Git как Markdown и диаграмму.

Что уже есть после лекции 1

Proposal — старт анализа, не готовые требования. К этой паре репозиторий ещё не обязан быть зелёным.

Зафиксировано

- тема и проблема;
- аудитория;
- 5–7 сущностей;
- scope v1: что входит и что нет.

Ещё нет

- кто что может сделать;
- как понять, что история готова;
- упорядоченный backlog;
- явное «не делаем в MVP».

Зачем требования сверх Proposal

Проблема объясняет *зачем*. Требования — *что* считать сделанным.

- Proposal без историй легко расползается: «раз уж есть User, добавим роли, биллинг и мобильное».
- Код без критериев приёмки не с чем сравнить: «у меня работает» — не определение готовности.
- ИИ охотно допишет чужой продукт. Backlog держит модель (и вас) внутри вашей задачи.
- Scope v1 и MVP — одно и то же: минимальный полезный срез, не «версия на потом».

Требования

Functional и non-functional

Functional (FR)

Поведение системы для пользователя: что можно сделать и какой результат. Создать задачу. Назначить владельца. Оставить комментарий.

Non-functional (NFR)

Ограничения и свойства качества: насколько надёжно, безопасно, понятно и воспроизводимо работает система. Пароли не попадают в лог. Сборка воспроизводится одной командой.

NFR без измеримого критерия — пожелание, не требование.

NFR для студенческого MVP

Конкретно и проверяемо. Не SLA и не «высокая нагрузка».

Подходит

- сборка — одна команда `./gradlew build`;
- пароли не попадают в лог;
- одинаковый вход даёт одинаковый формат ответа.

Не подходит в v1

- «система должна быть удобной»;
- uptime 99,9 %;
- миллион пользователей;
- «промышленный уровень безопасности»;
- пустое название задачи нельзя сохранить (это FR или бизнес-правило).

User story

История — ценность для актора, не пункт в бэклоге разработки.

Как **роль**, я хочу **действие**, чтобы **результат**.

- **Роль** берётся из аудитории Proposal, не «пользователь системы» вообще.
- **Действие** — одно изменение в мире пользователя, не модуль целиком.
- Третья часть отсекает CRUD без смысла: если «чтобы» пустое — история ещё не готова.

Образец: тематика 1

Система управления задачами и проектами

История. Как участник команды, я хочу назначить владельца задачи, чтобы к дедлайну было ясно, кто отвечает.

Зачем так. Проблема из Proposal — договорённости в чате; история бьёт в неё, а не в «сущность Task».

В MVP

Задача, владелец, статус.

Не эта история

Диаграмма Ганта, биллинг, роли с матрицей прав.

Антипаттерны историй

Плохо	Почему
Сделать CRUD для всех сущностей	Нет актора и ценности; это план работ, не требование.
Как пользователь, я хочу систему управления задачами	История = весь продукт. Нельзя принять и нельзя оценить.
Добавить TaskService и репозиторий	Implementation-таск. Backlog этой недели — про пользователя.
Как админ, я хочу удобный интерфейс	Роль не из аудитории; «удобный» не проверяется.

Критерии приёмки

Acceptance criteria проверяют конкретную историю, а не весь проект.

Чеклист

2–5 утверждений, которые можно подтвердить или опровергнуть на работающем сценарии.

Given / When / Then

Дано состояние. Когда действие. Тогда наблюдаемый результат.

Автотесты появятся позже. Сейчас AC нужны, чтобы не спорить постфактум, что именно обещано в MVP. Definition of Done будет общим чеклистом готовности проекта.

AC к истории про владельца

Как участник, я хочу назначить владельца задачи ...

Чеклист

- у задачи ровно один владелец;
- владельца можно сменить;
- назначенный владелец отображается в карточке задачи.

Given / When / Then

Дано участник и задача без владельца.
Когда участник назначает владельца.
Тогда владелец сохраняется и отображается в карточке задачи.

Когда история слишком большая

INVEST: Independent, Negotiable, Valuable, Estimable, Small, Testable. На старте достаточно трёх.

Valuable

Пользователь узнает, что изменилось в его работе.

Small

Можно описать и принять за итерацию, не за семестр.

Testable

Есть AC: да/нет, а не «в целом ок».

Если AC не помещаются на слайд — историю нужно резать, а не писать роман.

Backlog

Что такое backlog

Упорядоченный список ещё не сделанной работы. Не канбан «для галочки».

- Выше — раньше и важнее для MVP; ниже — можно отложить.
- Для этой практики основные элементы backlog — user stories с AC. В общем случае backlog также может содержать баги, технические задачи и исследования.
- Deliverable курса: Markdown в Git (`BACKLOG.md` или `docs/backlog.md`).
- Issues и Jira — индустриальный аналог. Для сдачи не обязательны.

MVP и MoSCoW

MVP — минимальный полезный срез scope v1 из Proposal. Must покрывает обязательную ценность; остальное не раздувает v1.

Класс	Значит	Пример (тематика 1)
Must	Без этого продукт не решает заявленную проблему	Задача, владелец, статус
Should	Усиливает ценность, можно второй очередью	Комментарии к задаче
Could	Приятно, не держит MVP	Вложения, упоминания
Won't	Сознательно не в v1 — как «Не в v1» в Proposal	Биллинг, Гант, мобильное приложение

Декомпозиция

От темы к историям. Implementation-таски — не элементы backlog этой недели.

Уровень	Что это
Предметная область	Тема семестра: трекер задач, библиотека, склад …
Epic	Крупная возможность: «договорённости по задачам видны команде»
User story	Ценность для актора: назначить владельца, сменить статус, оставить комментарий
Не сейчас	Класс `TaskService`, схема БД, пункт Gradle — это реализация истории

Как резать историю

10–15 элементов backlog — в основном истории MVP, не 50 сгенерированных CRUD-пунктов.

Резать по

- актору (участник vs преподаватель);
- сценарию (счастливый путь, затем отказ);
- данным (одна сущность, не «весь домен»).

Не резать на

- «сделать модель», «сделать UI», «сделать тесты»;
- слои архитектуры;
- «настроить репозиторий» — это уже лекция 1.

Артефакты и ИИ

Git и Markdown

`PROPOSAL.md` **и backlog живут рядом с кодом.** README содержит ссылку на Proposal; оба документа читает человек и модель.

Что коммитить

```
README.md
PROPOSAL.md
BACKLOG.md
```

Можно `docs/`, если корень репозитория уже тесный.

Зачем не в чате

- история изменений в Git;
- преподаватель видит тот же текст;
- ИИ опирается на ваши границы, а не на «типичный трекер».

Диаграммы

Mermaid в Markdown. GitHub и GitLab рисуют его без лишнего ПО.

Минимум в сдаче

- акторы из аудитории;
- граница системы;
- 2–4 главных потока к MVP.

Не сейчас

- диаграмма классов;
- схема БД;
- последовательность на каждый CRUD.

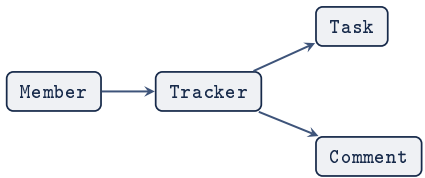
Диаграмма поясняет backlog, а не заменяет истории и AC.

Образец Mermaid

Один блок в `BACKLOG.md`. Слева исходник, справа — как его рисуют GitHub и GitLab.

```
mermaid

flowchart LR
  Member --> Tracker
  Tracker --> Task
  Tracker --> Comment
```



Участник работает с трекером; задача и комментарий — внутри системы.

Имена узлов — из вашего домена, не `User1`.

ИИ при анализе требований

Модель ускоряет черновик. Границы и смысл — ваши, из Proposal.

- Дайте модели *ваш* текст: проблема, аудитория, сущности, in/out v1.
- Просите истории *внутри* scope и дыры: кого забыли, что противоречит MVP.
- Черновик AC полезен; готовым он становится после вашей правки.
- Список «типичного трекера» без вашей проблемы — чужой продукт.

ИИ: правила

Можно

- набросать 10–15 историй по Proposal;
- найти пропущенного актора или дыру в MVP;
- переписать мутный NFR в проверяемый;
- набросать Mermaid по списку акторов.

Нельзя

- сдать сырой backlog модели без правок;
- подменить аудиторию «как в учебнике»;
- раздуть MVP, потому что «так бывает в Jira»;
- считать объём (50 историй) признаком качества.

Практика

Практическое задание

10–15 элементов backlog для выбранного проекта. Репозиторий лекции 1 может ещё дописываться — Markdown коммитится туда же.

1. Утвердить предметную область: номер темы или согласованная своя.
2. Зафиксировать MVP: in / out и сущности из Proposal; помнить о семестровой планке 5–7 взаимосвязанных сущностей.
3. 10–15 элементов backlog, преимущественно user stories: актор, ценность, AC (хотя бы 2–3 на историю). Истории должны покрывать ключевую сущность, поиск/фильтрацию и минимум два бизнес-правила.
4. Полезное дополнение: диаграмма Mermaid с акторами и границей системы.
5. Реализацию Spring Boot, PostgreSQL, тестов, Docker, CI и React на этой паре сдавать не требуется.
6. Файлы в Git: `PROPOSAL.md` и `BACKLOG.md`.

Критерий готовности

Готово, если

- Proposal и backlog находятся в Markdown в клоне;
- истории связаны с *вашей* проблемой, а не с абстрактным CRUD;
- есть MVP с явным Won't / «не в v1»;
- backlog покрывает ключевую сущность, поиск/фильтрацию и два бизнес-правила;
- у историй есть AC; диаграмма акторов — дополнение.

Ещё не готово

- backlog только в чате;
- одна строка «CRUD всех сущностей»;
- 50 пунктов без приоритета и без MVP;
- диаграмма классов вместо акторов.

Проектная деятельность

На этой неделе утверждается не код, а границы.

Утвердить

- предметную область;
- границы MVP (in / out);
- первоначальный backlog из 10–15 историй.

Ещё рано

- доменная модель в коде;
- выбор фреймворка «под все истории»;
- смена темы без разговора с преподавателем.

Итог

- FR — что пользователь может сделать; NFR — проверяемые свойства качества.
- User story и её AC задают проверяемый результат; «сделать CRUD» — не история.
- Backlog упорядочен; MVP — минимальный полезный срез scope v1; остальное — Won't до следующего среза.
- Артефакты — в Git: Markdown и диаграмма Mermaid. ИИ — черновик, не сдача.

Планируемый результат

Git, Markdown и индивидуальный backlog; диаграмма Mermaid — дополнение.

Вопросы?