Аделина Алексиева-Петрова

ИНЖЕНЕРСТВО НА ИЗИСКВАНИЯТА

Теория, практически упражнения и задачи

ИНЖЕНЕРСТВО НА ИЗИСКВАНИЯТА

Теория, практически упражнения и задачи

Аделина Алексиева-Петрова

Ръководството е предназначен за студенти от ОКС „Бакалавър“, специалност „Интелигентни системи и изкуствен интелект“, професионално направление 5.13 Общо инженерство.

София, 2026

Автор:
Аделина Алексиева-Петрова

Рецензенти:
доц. д-р инж. Върбинка Стоянова
доц. д-р инж. Иван Станков

Издател: Аделина Алексиева-Петрова

Първо издание
Формат: електронен, pdf

Година на издаване:
2026

ISBN: 978-619-04-0805-5

Съдържание

ГЛАВА 1
ИДЕНТИФИЦИРАНЕ НА ИЗИСКВАНИЯТА

1.1. Теоретични основи

1.1.1. Основни понятия

Изискване — В софтуерното инженерство понятието *„изискване“* се използва за обозначаване на условие или способност, която трябва да бъде изпълнена от дадена система, за да удовлетвори нуждите на потребителите или да постигне определена цел. Според стандартите на **IEEE** изискването може да бъде разглеждано като необходимост на потребителя, характеристика, която системата трябва да притежава, или документално описание на такава необходимост. В този смисъл изискванията представляват връзката между проблемите на реалния свят и функционалностите, които софтуерната система трябва да реализира.

Изискванията към една система могат да бъдат разделени на три основни категории:

Категория	Описание	Примери
Функционални изисквания	Описват поведението и функциите, които системата трябва да предоставя — какво трябва да прави системата.	Потребителят може да заема книга; системата изпраща напомняния; администраторът добавя нови записи.
Качествени изисквания (Нефункционални)	Описват характеристиките на качеството на системата.	Производителност, сигурност, надеждност, наличност, използваемост.
Ограничения	Допълнителни условия, които ограничават възможните решения.	Задължително използване на определена технология, стандарт или архитектура.

Функционалните изисквания могат допълнително да бъдат класифицирани според тяхното ниво и произход:

- **Системни изисквания —** описват поведението на системата на границата между нея и нейната среда.
- **Изисквания на заинтересованите страни —** представят желанията и потребностите на участниците.
- **Потребителски изисквания —** подмножество от горните, фокусирани върху крайните потребители.
- **Изисквания от областта —** произтичат от характеристиките на приложната предметна област.
- **Бизнес изисквания —** отразяват стратегическите цели на организацията.

Качествените изисквания се класифицират в следните категории според стандарта **ISO/IEC 25010**:

Категория	Описание	Ключов въпрос
Производителност	Поведение по отношение на време за реакция и използване на ресурси.	Колко бързо отговаря системата?
Сигурност	Защита на информацията и достъпността на системата.	Кой има достъп до какво?
Надеждност	Устойчивост при грешки и възстановимост.	Как се държи системата при повреда?
Използваемост	Лесота на употреба от крайните потребители.	Колко лесно е да се научи системата?
Поддържаемост	Лесота на промяна и разширяване на системата.	Колко струва добавянето на нова функционалност?
Преносимост	Способност на системата да работи в различни среди.	Работи ли системата на различни платформи?

Инженерството на изискванията (Requirements Engineering — RE) представлява систематичен и дисциплиниран подход към специфицирането и управлението на изискванията. Основната му цел е да осигури ясно разбиране за нуждите и желанията на заинтересованите страни и да намали риска от разработване на система, която не удовлетворява тези нужди. Процесът включва:

- Идентифициране на релевантните изисквания.
- Постигане на съгласие между заинтересованите страни.
- Документиране на изискванията.
- Управление на изискванията през целия жизнен цикъл на системата.

Защо инженерството на изискванията е критично важно?

- Намалява риска от провал на проекта и ограничава необходимостта от скъпи промени в по-късните етапи.
- Улеснява разбирането на проблема и възможните решения.
- Създава основа за оценка на необходимите ресурси.
- Служи като база за тестване на системата.

Без добре дефинирани изисквания разработването на качествен софтуер е практически невъзможно.

Заинтересованите страни (Stakeholders) са лица или организации, които имат влияние върху изискванията на системата или са засегнати от нейното функциониране. Те могат да участват пряко или косвено в процеса на разработка и често имат различни интереси и очаквания. В инженерството на изискванията е важно всички релевантни заинтересовани страни да бъдат идентифицирани и включени в процеса.

В реални системи заинтересованите страни могат да бъдат различни групи участници. Например в медицинска информационна система това могат да бъдат пациенти, лекари, медицински сестри, администратори, ИТ специалисти, както и управленски персонал. Всяка от тези групи има специфични нужди и очаквания към системата. Затова е необходимо инженерството на изискванията да вземе предвид различните перспективи, за да се осигури пълно и коректно описание на изискванията.

Системата може да бъде дефинирана като съвкупност от взаимосвързани елементи, които чрез координирано действие постигат определена цел. В контекста на софтуерното инженерство системата обикновено включва както софтуерни компоненти, така и хардуерни елементи, процеси и участници. Разбирането на системата като интегрирано цяло е важно условие за правилното дефиниране на изискванията и за успешното реализиране на нейната функционалност.

Инженерството на изискванията е приложимо при разработването на всякакъв тип системи, но в съвременната практика основният фокус е върху системи, в които софтуерът играе ключова роля. Такива системи могат да включват софтуерни компоненти, хардуерни устройства, както и организационни елементи като хора, процеси и нормативни изисквания. Примери за подобни системи са кибер-физичните системи, които комбинират софтуер и физически компоненти, както и социално-техническите системи, които включват взаимодействие между технологии и хора.

Не съществува универсален процес за инженерство на изискванията, който да бъде приложим за всички системи. Подходът трябва да бъде адаптиран към конкретния проект, неговия контекст и използвания модел на разработка. Например процесите могат да бъдат линейни и планово ориентирани или итеративни и гъвкави. Освен това върху процеса оказват влияние отношенията между доставчик, клиент и потребители, както и степента на участие на заинтересованите страни.

Инженерът по изискванията играе ключова роля в процеса на разработване на софтуерни системи. Основните му задачи включват извличане, документиране, валидиране и управление на изискванията. Освен това той трябва да притежава задълбочени познания в областта на инженерството на изискванията, за да може да дефинира подходящи процеси и да избере подходящи практики за конкретния проект. Важна част от неговата работа е и посредничеството между заинтересованите страни и техническите екипи, като по този начин се преодолява разликата между бизнес проблемите и техническите решения.

1.1.2. Фундаменталните принципи на инженерството на изискванията

Инженерството на изискванията се основава на девет фундаментални принципа, които насочват процеса на разработване на софтуерни системи. Тези принципи подчертават, че изискванията не трябва да се разглеждат изолирано, а като част от по-широк процес на създаване на стойност чрез софтуерни решения.

№	Принцип	Същност
1	**Ориентация към стойността**	Дейностите по дефиниране на изисквания имат смисъл само ако създават по-голяма полза, отколкото разходи.
2	**Заинтересовани страни**	Всички релевантни участници трябва да бъдат идентифицирани и включени в процеса.
3	**Споделено разбиране**	Всички участници трябва да имат обща представа за целите, функциите и ограниченията на системата.
4	**Контекст**	Разработването не може да се разглежда изолирано от средата, в която системата ще функционира.

№	Принцип	Същност
5	**Проблем, изискване, решение**	Всяка система се създава с цел да реши определен проблем — тази тристранна връзка трябва да се разбира и управлява.
6	**Валидиране**	Потвърждаване, че изискванията действително отговарят на нуждите и очакванията на заинтересованите страни.
7	**Еволюция на изискванията**	Промяната на изискванията е естествена и неизбежна — тя трябва да се управлява активно.
8	**Иновации**	Специалистът трябва да мисли проактивно и да предлага нови решения, надхвърляйки буквалните желания на потребителите.
9	**Систематична работа**	За всеки проект трябва да се конфигурира подходящ процес, съобразен с контекста и модела на разработка.

Принцип 1: Ориентация към стойността

Първият фундаментален принцип на инженерството на изискванията е ориентацията към стойността. Основната идея е, че дейностите по дефиниране и управление на изисквания имат смисъл само ако създават по-голяма полза, отколкото разходи. Добре формулираните изисквания намаляват риска от грешки, подобряват комуникацията между участниците и ограничават необходимостта от скъпи промени в по-късните етапи на разработката. Макар че усилията за инженерство на изискванията често се възприемат като разход в началото на проекта, те водят до значителни икономически ползи в дългосрочен план чрез намаляване на преработките и подобряване на качеството на крайния продукт.

В процеса на инженерство на изискванията е необходимо внимателно да се балансира между ползите от подробно специфициране на изискванията и ресурсите, необходими за тяхното формулиране. По-детайлните и прецизни изисквания допринасят за по-добра комуникация между заинтересованите страни и разработчиците, което намалява вероятността от недоразумения и грешки. В същото време тяхното създаване изисква значителни усилия и време. Следователно оптималното ниво на детайлност трябва да се определя в зависимост от контекста на проекта, неговата сложност, критичност и степента на риск.

Нивото на усилия, вложени в инженерството на изискванията, се определя от редица фактори. Сред тях са необходимото време и ресурси за формулиране на дадено изискване, степента, в която то допринася за успеха на системата, нивото на споделено разбиране между заинтересованите страни и разработчиците, както и наличието на референтни системи, които могат да служат като пример. Допълнителни фактори включват продължителността на цикъла за обратна връзка, характера на взаимоотношенията между клиент и доставчик, както и необходимостта от спазване на нормативни и регулаторни изисквания.

Принцип 2: Заинтересовани страни

Вторият фундаментален принцип на инженерството на изискванията е свързан със заинтересованите страни (stakeholders). Целта на всяка информационна система е да решава реални проблеми и да удовлетворява очакванията на различните участници, които имат интерес към нея. Тези участници могат да включват крайни потребители,

клиенти, оператори, регулаторни органи, както и разработчици. Поради разнообразието от роли и интереси е от съществено значение процесът на инженерство на изискванията да идентифицира и включи всички релевантни заинтересовани страни, за да се гарантира успешното разработване на системата.

За по-ефективно управление на изискванията заинтересованите страни могат да бъдат класифицирани според степента на тяхното влияние върху успеха на системата:

Категория	Определение	Пример (библиотечна система)
Критични	Чието игнориране може да доведе до провал на системата.	Читатели, библиотекари.
Основни	Значително влияние, но отсъствието им не води задължително до провал.	Администратор на системата, ИТ поддръжка.
Маловажни	Минимално влияние, но могат да допринесат с полезна информация.	Регулаторни органи, одитори.

Принцип 3: Споделено разбиране

Споделеното разбиране представлява ключов фактор за успешното разработване на софтуерни системи. То означава, че всички участници в проекта имат обща представа за целите на системата, за нейните функционалности и за начина, по който тя трябва да отговори на потребностите на потребителите. В традиционните планово ориентирани процеси това разбиране се постига чрез внимателно документирани и официално съгласувани изисквания. В гъвкавите методологии (agile) то често се базира на неформално, но добре споделено знание между членовете на екипа.

Създаването на споделено разбиране между участниците в проекта се влияе от различни фактори. Сред подпомагащите фактори се включват доброто познаване на предметната област, наличието на специфични стандарти за областта, успешният предишен опит в съвместна работа, наличието на референтни системи, споделената организационна култура и взаимното доверие между участниците. От друга страна, редица препятствия могат да затруднят този процес, като географската дистанция между екипите, отношенията на недоверие между клиент и доставчик, аутсорсингът, регулаторните ограничения, големите и разнородни екипи, както и високото текучество на персонала.

Принцип 4: Контекст

Четвъртият принцип на инженерството на изискванията подчертава значението на контекста при дефинирането на системните изисквания. Разработването на система не може да се разглежда изолирано от средата, в която тя ще функционира. В процеса на инженерство на изискванията трябва да се анализира как евентуалните промени в контекста могат да повлияят върху системата, кои реални изисквания от средата са релевантни за разработвания софтуер и как те могат да бъдат коректно трансформирани в системни изисквания. Освен това е необходимо да се формулират предположения за контекста, които трябва да бъдат изпълнени, за да може системата да функционира правилно.

В инженерството на изискванията понятието контекст се използва за обозначаване на частта от средата на системата, която е релевантна за разбирането на нейните изисквания и функциониране. Контекстът включва всички външни фактори, условия и взаимодействия, които оказват влияние върху системата. В тази връзка се дефинират още граница на контекста – разделителната линия между релевантните и нерелевантните аспекти на приложната област, както и граница на системата – границата между самата система и нейната външна среда. Понятието обхват (scope) определя кои аспекти могат да бъдат проектирани и контролирани при разработването на системата.

Принцип 5: Проблем, изискване и решение

Петият принцип на инженерството на изискванията разглежда връзката между проблем, изисквания и решение. Всяка система се създава с цел да реши определен проблем, възникнал в реалната практика. Когато потребителите не са удовлетворени от начина, по който извършват дадена дейност, възниква необходимост от ново решение. За да може разработваната система да изпълни ролята на ефективно решение, е необходимо изискванията към нея да бъдат внимателно идентифицирани, анализирани и формулирани.

Връзката между проблема, изискванията и решението често е сложна и взаимосвързана. Например техническата осъществимост на дадено изискване често може да бъде оценена едва след разглеждане на възможните технически решения. Прототипите, използвани за валидиране на изискванията, също представляват частични решения на първоначалния проблем.

> **Важна бележка**
> В практиката строгото разделяне между инженерство на изискванията, системен дизайн и имплементация рядко е напълно възможно.
> Въпреки това специалистите се стремят концептуално да разграничават проблема, изискванията и решенията — това улеснява анализа, комуникацията и документацията.

Принцип 6: Валидиране на изискванията

Шестият принцип се отнася до валидирането на изискванията. Валидирането представлява процес на потвърждаване, че даден елемент (система, продукт или част от нея) отговаря на нуждите и очакванията на заинтересованите страни. Основната цел на тази дейност е да се гарантира, че специфицираните изисквания действително описват необходимите функции и характеристики на системата. Без валидиране съществува риск разработваната система да не отговаря на реалните потребности на потребителите.

Валидирането включва проверка дали е постигнато съгласие между заинтересованите страни относно изискванията, дали техните нужди и очаквания са адекватно отразени в спецификациите и дали направените предположения за контекста на системата са реалистични. Тази дейност е основна част от процеса на инженерство на изискванията, тъй като без нея спецификацията на изискванията може да се окаже непълна, неточна или неприложима в реална среда.

Принцип 7: Еволюция на изискванията

Седмият принцип подчертава, че промяната на изискванията е естествена и неизбежна част от жизнения цикъл на софтуерните системи. Причини за промяна могат да бъдат:

- Изменения в бизнес процесите.
- Появата на нови конкурентни продукти.

- Промяна в приоритетите на клиентите.
- Развитие на технологиите или обратна връзка от потребителите.
- Открити грешки или неточни предположения по време на разработката.

Основното предизвикателство при еволюцията на изискванията е да се постигне баланс между гъвкавост и стабилност. От една страна, процесът трябва да позволява промени, тъй като опитите за тяхното игнориране са неефективни. От друга страна, прекалено честите и неконтролирани промени могат да доведат до значително увеличаване на разходите и до затруднения в разработката. Затова инженерите по изисквания трябва активно да управляват този процес чрез подходящи методи и инструменти..

Принцип 8: Иновации

Осмият принцип разглежда ролята на иновациите в инженерството на изискванията. Процесът не се свежда единствено до записване на желанията на заинтересованите страни. Често потребителите не могат ясно да формулират всички възможности за подобрение. Затова добрите специалисти по изисквания трябва да мислят проактивно и да предлагат нови решения, които да подобрят функционалността на системата и потребителското изживяване. Иновациите могат да бъдат както малки подобрения, така и значителни нововъведения.

Принцип 9: Систематична и дисциплинирана работа

Деветият принцип подчертава необходимостта от систематичен и дисциплиниран подход към инженерството на изискванията. За всеки проект трябва да се конфигурира подходящ процес, съобразен със спецификата на проблема, контекста и използвания модел на разработка. Не съществува универсален набор от практики, който да бъде приложим за всички ситуации. Поради това е важно внимателно да се подбират методите, артефактите и техниките, които най-добре отговарят на конкретния проект.

Въпреки че съвременните методологии често подчертават гъвкавостта и адаптивността, това не означава, че инженерството на изискванията трябва да се извършва хаотично или без структура.

Ключово послание
Дори при гъвкави процеси е необходимо ясно дефиниране на дейностите, артефактите и отговорностите.
Само чрез систематична и дисциплинирана работа може да се постигне високо качество на изискванията и успешна реализация на софтуерните системи.

1.2. Практическо приложение

Казус: Разработване на система „Онлайн библиотека“

Описание на казуса
Системата има за цел да предоставя централизирана и лесна за използване платформа за управление на книги — включително преглед на наличните книги, добавяне на нови записи, заемане и връщане на книги.

Таблица 1 — Функционални изисквания

Таблицата включва: уникален идентификатор (ID), описание на функционалността (Item), категория (модул), приоритет, заинтересовани страни и допълнителни коментари.

ID	Item	Category	Priority	Stakeholders	Comment
FR1	Системата трябва да предоставя списък с всички налични книги.	Модул „Каталог на книги“	**Major**	Читатели (Primary) Библиотекари (Primary) Администратор (Secondary)	
FR2	Системата трябва да позволява на потребителя да добавя нова книга чрез въвеждане на заглавие, автор и други данни.	Модул „Управление на книги“	**Critical**	Библиотекари (Primary) Администратор (Secondary) Читатели (Tertiary)	
FR3	При заявка за заемане, системата трябва да проверява дали книгата съществува и е налична, след което да промени статуса й на „заета“.	Модул „Заемане на книги“	**Critical**	Читатели (Primary) Библиотекари (Primary) Администратор (Secondary)	
FR4	Системата трябва да валидира дали дадена книга е заета и, ако е така, да промени статуса й на „налична“ при връщане.	Модул „Връщане на книги“	**Critical**	Читатели (Primary) Библиотекари (Primary) Администратор (Secondary)	
FR5	Ако потребителят заяви книга с несъществуващо ID, системата трябва да върне съобщение за грешка "Book not found".	Модул „Обработка на грешки“	**Critical**	Читатели (Primary) Библиотекари (Primary) Разработчици / ИТ (Secondary)	

Функционалните изисквания обхващат основните операции на системата, включително: преглед на наличните книги; добавяне на книга; заемане на книга; връщане на книга и обработка на грешки при невалидни заявки.

Нефункционалните изисквания към системата са дефинирани в Таблица 2 – Нефункционални изисквания. Таблицата описва нефункционалните изисквания по следния начин: уникален идентификатор (ID); текст на изискването (Item); категория (напр. Performance, Usability, Maintainability, Reliability); приоритет; източник и коментари и измерими критерии.

Нефункционалните изисквания обхващат аспекти като:

- производителност – системата да отговаря на заявки за под 1 секунда при нормално натоварване;
- удобство за използване (usability) – ясни и разбираеми съобщения за грешки;
- поддръжка и разширяемост (maintainability) – лесно добавяне на нови функционалности благодарение на слоестата архитектура;
- надеждност (reliability) – коректност и непрекъснатост на данните при нормална работа.

Таблица 2 – Нефункционални изисквания.

ID	Item	Category	Priority	Source	Comments / Критерий
NFR1	Системата трябва да отговаря на стандартни заявки (GET/POST) в рамките на под 1 секунда при нормално натоварване.	**Performance**	Should	Enterprise Architects	Меримо: < 1 sec
NFR2	Системата трябва да осигурява ясни и разбираеми съобщения за грешки, за да улеснява потребителя.	**Usability**	Should	Security & Compliance	100% текстови съобщения при грешки
NFR3	Системата трябва да бъде лесно разширяема чрез слоестата архитектура (Presentation → Service → Repository).	**Maintainability**	Should	Enterprise Architects	Добавяне на нов endpoint без промяна в повече от 1 слой
NFR4	Системата трябва да гарантира непрекъсната коректност на данните при нормална работа.	**Reliability**	**Must**	Quality Assurance	0 загубени записи в in-memory storage
NFR5	Системата трябва да се зарежда напълно при	**Performance**	Could	Quality Assurance	Меримо: < 2 sec initial load

ID	Item	Category	Priority	Source	Comments / Критерий
	първоначално стартиране за под 2 секунди.				
NFR6	API интерфейсът трябва да бъде лесно разбираем и да използва стандартизирани HTTP методи.	**Usability**	Should	Enterprise Architects	REST принципи

Нефункционалните изисквания обхващат четири ключови аспекта:

- **Производителност** — отговор на заявки за под 1 секунда при нормално натоварване.
- **Удобство за използване (Usability) —** ясни и разбираеми съобщения за грешки.
- **Поддръжка и разширяемост (Maintainability)** — лесно добавяне на нови функционалности чрез слоестата архитектура.
- **Надеждност (Reliability)** — коректност и непрекъснатост на данните при нормална работа.

1.3. Въпроси и задачи за упражнения

Следващите три казуса са проектирани така, че да ви помогнат да упражните идентифицирането на изисквания, класификацията на заинтересованите страни и прилагането на фундаменталните принципи на инженерството на изискванията. Препоръчително е да работите по казусите в групи от 2–3 студента.

Задачи към казусите:

1. Идентифицирайте основните заинтересовани страни. Класифицирайте ги като Primary, Secondary и Tertiary и обосновете класификацията.
2. Формулирайте най-малко пет функционални и три нефункционални изисквания. Попълнете ги в таблична форма по образеца от Таблица 1 и Таблица 2.
3. Определете кои от изискванията са критични за функционирането на системата.
4. Обсъдете рискове и проблеми, които могат да възникнат при разработването и поддръжката (сигурност на плащанията, защита на личните данни, надеждност).

Класификация на заинтересованите страни (Stakeholders)
Primary (Първични) — директни потребители на системата.
Secondary (Вторични) — поддържат или използват системата косвено.
Tertiary (Третични) — имат индиректен интерес или влияние върху системата.

Класификация на приоритетите на изискванията
Critical (Критично) — системата не може да функционира без това изискване.
Major (Основно) — важно изискване, но може да бъде реализирано в по-късен етап.
Minor (Незначително) — допълнителна функционалност, която подобрява системата, но не е задължителна.

Шаблон за попълване (Таблица 1) — Функционални изисквания:

ID	Изискване	Категория	Приоритет	Заинтересовани страни	Коментар
FR1					
FR2					
FR3					
FR4					
FR5					

Шаблон за попълване (Таблица 2) — Нефункционални изисквания:

ID	Изискване	Категория	Приоритет	Източник	Коментар / Критерий
NFR1					
NFR2					
NFR3					

1.3.1. Казус: Разработване на система за интелигентно паркиране

Описание на казуса
Градска администрация планира внедряване на интелигентна система за паркиране, която да подпомага шофьорите при намиране на свободни паркоместа в централната част на града. Системата ще включва мобилно приложение, предоставящо информация за наличните паркоместа в реално време и позволяващо извършване на онлайн плащания.

Основни цели на системата:

- Намаляване на времето за търсене на свободно паркомясто.
- Оптимизиране на използването на съществуващите паркинг пространства.
- Подобряване на управлението на градската транспортна инфраструктура.

В системата участват различни заинтересовани страни, включително шофьори, градска администрация, оператори на паркинги, доставчици на технологични решения и платежни оператори.

1.3.2. Казус: Разработване на онлайн магазин (E-commerce платформа)

Описание на казуса
Търговска компания планира разработването на онлайн магазин, който да позволява на клиентите да разглеждат и закупуват продукти чрез интернет. Системата ще включва уеб платформа за търсене на продукти, добавяне в количка и онлайн

плащания, както и административен панел за управление на продукти, поръчки и потребители. Освен това системата ще включва административен панел за управление на продукти, поръчки и потребители.

Основни цели на системата:

- Улесняване на процеса на онлайн пазаруване за клиентите.
- Разширяване на пазарния обхват на компанията чрез електронна търговия.
- Автоматизиране на управлението на продукти, поръчки и доставки.
- Подобряване на ефективността на бизнес процесите.

В системата участват различни заинтересовани страни, включително клиенти, администратори на онлайн магазина, търговски мениджъри, доставчици на продукти, платежни оператори и ИТ специалисти, които разработват и поддържат системата..

1.3.3. Казус: Разработване на система за хотелски резервации

Описание на казуса

Хотелска верига планира разработването на информационна система за управление на хотелски резервации, която да позволява на клиентите да проверяват наличността на стаи и да правят резервации онлайн. Системата ще включва уеб платформа или мобилно приложение за избор на дати, тип стая и допълнителни услуги, както и онлайн плащане.

Основни цели на системата:

- Улесняване на процеса на резервация за клиентите.
- Подобряване на управлението на хотелските ресурси.
- Намаляване на административната работа на хотелския персонал.
- Повишаване на ефективността и качеството на обслужване.

В системата участват различни заинтересовани страни, включително гости на хотела, рецепционисти, хотелски мениджъри, администратори на системата, платежни оператори и ИТ специалисти, които отговарят за разработването и поддръжката на системата.

ГЛАВА 2
СПЕЦИФИКАЦИИ НА ЕСТЕСТВЕН ЕЗИК. ДОКУМЕНТАЦИЯ ЗА ИЗИСКВАНИЯТА. РЕЧНИК.

2.1. Теоретични основи

2.1.1. Работен продукт

В инженерството на изискванията терминът *работен продукт* (work product) се използва за обозначаване на всеки записан междинен или краен резултат, създаден по време на даден работен процес. Такива резултати могат да включват документи, модели, диаграми или други артефакти, подпомагащи анализа, специфицирането и управлението на изискванията. В литературата термините *работен продукт* и *артефакт* често се използват като синоними. Работните продукти играят ключова роля в комуникацията между заинтересованите страни и разработчиците, тъй като служат като официално средство за документиране на знанията за системата.

Работните продукти в инженерството на изискванията се характеризират чрез няколко основни свойства:

Свойство	Описание
Цел	Определя каква информация трябва да съдържа продуктът и как ще бъде използван.
Размер	Обемът на документа — от кратка потребителска история до пълна SRS спецификация.
Форма на представяне	Естествен език, шаблони, формални модели, диаграми.
Жизнен цикъл	Как продуктът се създава, актуализира и поддържа в хода на проекта.
Начин на съхранение	Хранилище за документи, Wiki, система за управление на изисквания.

Разбирането на тези характеристики е важно, тъй като позволява правилен избор на подходящи артефакти в зависимост от конкретния проект и контекст.

Изискванията към системата съществуват на различни нива на абстракция — от общи бизнес цели до конкретни технически спецификации. В инженерството на изискванията се използва **йерархия на изискванията**, при която бизнес нуждите, домейн знанията и изискванията на заинтересованите страни определят системните изисквания и изискванията към подсистемите.

Защо нивото на детайлност е важно?
По-подробните изисквания намаляват риска от грешна интерпретация, но изискват повече ресурси за създаване.

Оптималният баланс зависи от сложността на проблема, степента на познаване на домейна, нивото на споделено разбиране между участниците и приложимите стандарти и регулации.

2.1.2. Естествен език в инженерството на изискванията

Естественият език е най-често използваното средство за описване на изискванията, тъй като е лесно разбираем и широко използван в ежедневната комуникация. Той позволява гъвкаво и изразително описание на системните характеристики.

Предимства	Рискове
Разбираем за всички заинтересовани страни.	Двусмислие и неясноти в описанието.
Гъвкавост при описание на сложни ситуации.	Непълни или противоречиви изисквания.
Не изисква специални инструменти или обучение.	Различни интерпретации от страна на участниците.
Лесна модификация и актуализация.	Трудна автоматична обработка и верификация.

За да се намали рискът от двусмислие, при писането на изисквания трябва да се следват определени правила:

- Използвайте кратки и добре структурирани изречения.
- Прилагайте последователна терминология в рамките на проекта.
- Избягвайте неясни изрази като „бързо“, „лесно“, „удобно“ — заменете ги с измерими критерии.
- Дефинирайте всеки специализиран термин в Речника на проекта.

Използването на шаблони е широко разпространена практика при създаването на работни продукти в инженерството на изискванията. Шаблоните осигуряват стандартизирана структура за документи и улесняват създаването на последователна и добре организирана документация. Те могат да бъдат под формата на шаблони за изречения, формуляри или цели документи.

2.1.3. Шаблон за изрази

Шаблоните за изрази представляват предварително дефинирани синтактични структури за формулиране на индивидуални изисквания. Използването им е добра практика, тъй като подпомага създаването на ясни и последователни формулировки.

Унифициран шаблон — ISO/IEC/IEEE 29148

```
[<Условие>]  <Субект>  <Действие>  <Обекти>  [<Ограничение>]
```

Пример: Когато клиентът потвърди поръчка в онлайн магазина,
системата трябва да изпрати имейл за потвърждение в рамките на 5 секунди.

Тази структура позволява ясно дефиниране на поведението на системата при определени условия. Използването на подобни шаблони намалява риска от двусмислие и улеснява проверката и валидирането на изискванията.

При формулиране на действие в практиката се използват следните конвенции за спомагателни глаголи:

Спомагателен глагол	Значение и употреба
„Ще“	Задължително изискване — системата трябва да изпълни това без изключение.
„Трябва“	Силно желателно изискване — не е абсолютно задължително, но е силно препоръчително.
„Може“	Предложение или опционална функционалност — системата може да предоставя тази възможност.

В зависимост от контекста могат да се използват различни типове шаблони за изисквания. Дефинирани са набор от шаблони за фрази, които са адаптирани към различни ситуации:

Повсеместни изисквания (Ubiquitous Requirements)

Изисквания, които трябва винаги да са в сила, независимо от контекста.

> **Шаблон: Повсеместно изискване**
> <Името на системата> трябва да <отговорът на системата>.
>
> Пример: Системата за хотелски резервации трябва да съхранява информация за всички направени резервации.

Изисквания, основани на събитие (Event-driven Requirements)

Изисквания, задействани от конкретно външно събитие или потребителско действие.

> **Шаблон: Изискване, основано на събитие**
> КОГАТО <незадължителни предварителни условия> <задействa>
> <Името на системата> трябва да <отговорът на системата>.
>
> Пример: КОГАТО клиентът изпрати заявка за резервация на стая, системата трябва да провери наличността и да потвърди резервацията.

Нежелано поведение (Unwanted Behavior)

Описание на ситуации, неправилни входове или грешки, на които системата трябва да реагира правилно.

> **Шаблон: Нежелано поведение**
> АКО <незадължителни предварителни условия> <причинител>,
> ТО тогава <Името на системата> трябва да <реакция на системата>.

Пример: АКО потребителят въведе неправилна парола три пъти,
ТО тогава системата трябва да блокира достъпа за 15 минути.

Кога да ползваме кой шаблон?
Повсеместно — за постоянни правила на системата, валидни при всякакви обстоятелства.
Основано на събитие — за реакции при конкретно потребителско действие или системно събитие.
Нежелано поведение — за обработка на грешки, невалидни входове и гранични случаи.

2.1.4. Потребителски истории (User Stories)

Потребителските истории са кратки описания на функционалностите от гледна точка на потребителя. Използват се широко в гъвкавите методологии, тъй като поставят акцент върху стойността, която системата предоставя.

Стандартен формат на потребителска история
„Като <роля> искам <функционалност>, за да <полза/цел>."

Пример: „Като клиент искам да резервирам маса онлайн,
за да си осигуря място в ресторанта."

Този подход се използва широко в гъвкавите методологии за разработка на софтуер, тъй като поставя акцент върху стойността, която системата предоставя на потребителите.

Критерии за приемане (Acceptance Criteria) представляват набор от условия, които дадена функционалност трябва да изпълнява, за да бъде приета като завършена. Те:

- Определят границите на потребителската история.
- Подпомагат създаването на общо разбиране в екипа.
- Служат като основа за разработването на тестове.
- Помагат да се определи кога дадена функционалност е готова.

Пример за пълна потребителска история
User Story: „Като клиент искам да поднова заемането на книга,
за да продължа да я ползвам."

Критерии за приемане:
1. Потребителят може да поднови само ако книгата не е просрочена.
2. Потребителят може да поднови само ако книгата не е резервирана от друг.
3. След подновяване системата изпраща потвърждение по имейл.
4. Новият краен срок се показва в профила на потребителя.

2.1.5. Шаблони за документи

Шаблоните за документи представляват предварително дефинирана структура, която се използва при създаването на документи с изисквания. Те осигуряват ясна организация на съдържанието и улесняват създаването на последователна документация.

Използването на шаблони има редица предимства, като осигуряване на последователна структура, подобряване на качеството на документацията и намаляване на риска от

пропуски. В същото време прекомерното механично използване на шаблони може да доведе до игнориране на важни детайли, които не се вписват в предварително определената структура.

Документът „Спецификация на системни изисквания“ (Software Requirements Specification – SRS) описва функционалните и нефункционалните изисквания към дадена софтуерна система. Той служи като официално средство за комуникация между заинтересованите страни, анализаторите, разработчиците и тестерите. SRS документът определя какво трябва да прави системата, какви ограничения съществуват и как системата ще взаимодейства с потребителите и други системи.

Примерна структура на SRS документ:

Част	Съдържание
Част I: Въведение	• Цел на системата • Обхват на разработката • Заинтересовани страни
Част II: Общ преглед	• Визия и цели на системата • Контекст и граници на системата • Обща структура и характеристики на потребителите
Част III: Системни изисквания	• Функционални изисквания (структура, функции, поведение) • Качествени (нефункционални) изисквания • Ограничения и интерфейси
Част VI: Референции	• Речник (Glossary)
Част V:	• Предположения и зависимости

Важна бележка за шаблоните
Прекомерното механично следване на шаблон може да доведе до игнориране на важни детайли, които не се вписват в предварително определената структура. Шаблонът трябва да служи като ориентир, а не като ограничение.

2.1.6. Речници (Glossary)

Речникът представлява списък от дефиниции на термините, използвани в даден проект. Неговата основна цел е да предотврати различни интерпретации на едни и същи понятия от страна на различните участници. Речникът трябва да съдържа:

- Дефиниции на специализирани термини от предметната област.
- Съкращения и акроними с пояснения за техните значения.
- Еднозначна терминология, използвана последователно в цялата документация.

Защо речникът е критично важен?
Централизираното управление на терминологията подпомага общото разбиране между заинтересованите страни и разработчиците.
Единната терминология намалява риска от грешки и недоразумения в документацията на изискванията.

Без Речник различните участници могат да имат различно разбиране за един и същи термин, което неизбежно води до грешки в реализацията.

2.2. Практическо приложение

Казус: Система за онлайн поръчки и резервации (Ресторант)

Описание на казуса
Да се дефинират изисквания към система, която предоставя функционалности за онлайн поръчки и резервации в ресторант, включително възможност клиентите да разглеждат менюто, да правят поръчки чрез уеб или мобилно приложение, да резервират маси в ресторанта и да извършват онлайн плащания. Освен това системата позволява проследяване на статуса на поръчките, така че клиентите да получават информация за етапа на изпълнение – например дали поръчката се подготвя, готова е за доставка или вече е доставена.

2.2.1 Примери по видове шаблони

1. Унифициран шаблон за индивидуални изисквания:

Унифициран шаблон — Пример
Когато потребителят въведе валидно потребителско име и парола,
системата ще предостави достъп до потребителския профил в рамките на 2 секунди.

2. Повсеместно изискване:

Повсеместно изискване — Пример
Системата за резервации трябва да съхранява информация
за всички направени резервации.

3. Изискване, основано на събитие:

Изискване, основано на събитие — Пример
КОГАТО клиентът изпрати заявка за резервация на маса,
системата за резервации трябва да провери наличността
на масите и да потвърди резервацията.

4. Нежелано поведение:

Нежелано поведение — Пример
АКО потребителят въведе неправилна парола три пъти,
ТО тогава системата трябва да блокира достъпа до акаунта за 15 минути.

5. Потребителска история (User Story):

ID	Потребителска история (User Story)	Критерии за приемане (Acceptance Criteria)
US-1	***„Като клиент искам да резервирам маса онлайн, за да си осигуря място в ресторанта.“***	1. Системата показва свободните часове. 2. Клиентът може да избере дата и брой гости. 3. След потвърждение системата записва резервацията. 4. Клиентът получава имейл за потвърждение.

2.2.2 Речник (Glossary)

Термин	Дефиниция
Потребител (User)	Лице, което използва системата за извършване на определени действия — разглеждане на информация, резервации или поръчки.
Администратор (Administrator)	Потребител с разширени права, управляващ системата, настройките и съдържанието ѝ.
Резервация (Reservation)	Процесът на запазване на услуга (хотелска стая, маса) за определена дата и час.
Поръчка (Order)	Заявка от потребител за закупуване на продукт или услуга чрез системата.
Онлайн плащане (Online Payment)	Процес на електронно плащане чрез банкови карти или платежни системи.
Система (System)	Софтуерното приложение, предоставящо функционалности за управление на поръчки, резервации и потребители.
База данни (Database)	Структурирано хранилище за данни — информация за потребители, поръчки и резервации.
Потребителски профил (User Profile)	Набор от данни, свързани с конкретен потребител: име, имейл и настройки на акаунта.
Потвърждение (Confirmation)	Съобщение или уведомление, информиращо потребителя, че дадено действие е успешно изпълнено.
Съобщение за грешка (Error Message)	Системно съобщение, уведомяващо потребителя за настъпила грешка или невалидно действие.

2.3. Въпроси и задачи за упражнения

Следващите три казуса са проектирани така, че да ви помогнат да упражните формулирането на изисквания по различните шаблони, създаването на потребителски истории с критерии за приемане и изграждането на речник на проекта. Препоръчително е да работите в групи от 2–3 студента.

Задачи към всички казуси:

1. Формулирайте примерни изисквания, използвайки и четирите шаблона: унифициран, повсеместен, основан на събитие и нежелано поведение.
2. Създайте най-малко 3 потребителски истории (User Story) с критерии за приемане.
3. Изгответе кратък речник (Glossary) с основните термини в системата.

Шаблон за попълване — Изисквания по видове:

Вид шаблон	Формулиране на изискването
Унифициран шаблон	
Повсеместно изискване	
Основано на събитие	
Нежелано поведение	

Шаблон за попълване — Потребителски истории:

ID	Потребителска история	Критерии за приемане
US-1	*Като <роля> искам <функционалност>, за да <полза>.*	1. 2. 3.
US-2	*Като <роля> искам <функционалност>, за да <полза>.*	1. 2. 3.
US-3	*Като <роля> искам <функционалност>, за да <полза>.*	1. 2. 3.

Шаблон за попълване — Речник:

Термин	Дефиниция

Напомняне — Четирите вида шаблони за изисквания
1. Унифициран шаблон: [Условие] Субект Действие Обект [Ограничение]
2. Повсеместно: "<Система> трябва да <отговор>."

3. Основано на събитие: "КОГАТО <събитие>, <Система> трябва да <отговор>."
4. Нежелано поведение: "АКО <причина>, ТО тогава <Система> трябва да <реакция>."

2.3.1. Казус 1: Online Store (E-commerce Platform)

Описание на казуса
Да се дефинират изисквания към система за **онлайн магазин**, която позволява на потребителите да разглеждат продукти, да добавят артикули в количка и да извършват онлайн покупки. Системата трябва да поддържа търсене на продукти, управление на поръчки, обработка на плащания и проследяване на статуса на поръчките.

Клиентите могат да използват **уеб или мобилно приложение**, за да разглеждат каталога с продукти, да добавят продукти в количката, да извършват онлайн плащане и да проследяват доставката на поръчката си. Администраторите на системата могат да добавят нови продукти, да актуализират информация за наличности и да управляват поръчките.

Системата трябва да предоставя информация за текущия статус на поръчките, например дали поръчката е **приета, обработва се, изпратена или доставена**.

2.3.2. Казус 2: Hotel Reservation System

Описание на казуса
Да се дефинират изисквания към система за хотелски резервации, която позволява на клиентите да проверяват наличността на стаи, да правят резервации и да извършват онлайн плащания. Системата трябва да позволява избор на тип стая, дата на настаняване и дата на напускане.

Потребителите могат да използват системата чрез **уеб или мобилно приложение**, за да търсят свободни стаи, да създават резервации и да получават потвърждение за тях. Персоналът на хотела може да използва системата за управление на резервациите, актуализиране на наличността на стаите и обработка на анулации.

Системата трябва да предотвратява **двойни резервации** и да предоставя информация за статуса на резервацията, например **потвърдена, отменена или приключена**.

2.3.3. Казус 3: Smart Parking System

Описание на казуса
Да се дефинират изисквания към **интелигентна система за паркиране**, която помага на шофьорите да намират свободни паркоместа в градска среда. Системата трябва да предоставя информация за наличността на паркоместа в реално време и да позволява извършване на онлайн плащане за паркиране.

Шофьорите могат да използват **мобилно приложение**, за да виждат свободни паркоместа на карта, да резервират паркомясто и да плащат за престоя си. Системата може да използва сензори или други технологии за автоматично отчитане на заетостта на паркоместата.

Системата трябва също да позволява на градската администрация да **наблюдава използването на паркинг местата и да управлява паркинг зоните**.

ГЛАВА 3
ТЕХНИКИ ЗА СЪБИРАНЕ НА ИЗИСКВАНИЯ

3.1. Теоретична основа

Събирането на изисквания (Requirements Elicitation) е систематичен процес на идентифициране, извличане и документиране на нуждите на заинтересованите страни по отношение на разработвана или оптимизирана система. Това е фундаментална и критична фаза в жизнения цикъл на всеки проект за разработка на информационна система или за оптимизация на бизнес процеси. Грешките, допуснати на тази фаза, обикновено са и най-скъпите за поправяне — те се открият едва когато системата вече е частично или напълно изградена.

Аналитикът разполага с различни техники за елиситация, всяка от които е подходяща за различен контекст, тип информация и категория заинтересовани страни. Умението да се избере правилната техника — или правилната комбинация от техники — е едно от ключовите компетенции на успешния бизнес анализатор.

3.1.1. Класификация на техниките

В практиката на бизнес анализа се използват следните основни групи техники за събиране на изисквания:

Група	Техники	Основна цел
Комуникационни	Интервю, Фокус група, Работна среща	Събиране на мнения, нужди и болезнени точки от хора
Наблюдение	Наблюдение на работното място	Разбиране на реалното поведение и скрити процеси
Документални	Анализ на документи, Анализ на интерфейси	Извличане на правила и изисквания от съществуващи артефакти
Творчески	Мозъчна атака (Brainstorming)	Генериране на нови идеи и иновативни решения

В следващите точки всяка техника е описана самостоятелно — с цел, стъпки за прилагане и практически насоки.

3.1.2. Анализ на документи (Document Analysis)

Анализът на документи е техника, при която аналитикът изследва съществуващи организационни документи с цел извличане на бизнес правила, процеси и изисквания. Документите могат да включват наръчници, процедури, политики, договори, отчети, формуляри и законови разпоредби.

Кога се прилага: Особено полезна е при проекти, в които организацията разполага с богата вътрешна документация, при разработване на система, която заменя

съществуваща, или когато заинтересованите страни имат ограничено свободно време за директна комуникация.

Стъпки за прилагане:

1. Идентифициране на целите и обхвата — определя се каква информация се търси и за каква цел.
2. Събиране и организиране — всички документи се събират, верифицират и категоризират по тип и актуалност.
3. Преглед на структурата и съдържанието — проверява се пълнотата, идентифицират се бизнес правила и зависимости; маркират се неясни, остарели или противоречиви части.
4. Анализ за бизнес изисквания — извличат се функционалните и нефункционалните изисквания, идентифицират се ограничения, допускания и зависимости.
5. Валидиране със заинтересованите страни — откритите несъответствия и пропуски се обсъждат с притежателите на процесите.
6. Резюмиране и доклад — формира се структуриран документ с констатации, препоръки и открити пропуски.

Практически съвет
Провеждайте анализа на документите ПРЕДИ интервютата — така влизате подготвени и задавате по-прецизни въпроси.
Маркирайте активно: дата на документа, автор, версия — остарелите документи са честа причина за грешни изисквания.

3.1.3. Интервю (Interview)

Интервюто е директна комуникация между аналитика и представител на заинтересованите страни с цел събиране на детайлна, контекстуална и нюансирана информация. То е една от най-широко прилаганите техники в бизнес анализа.

Видове интервюта:

- **Структурирано —** фиксирани въпроси без отклонения; подходящо при нужда от сравнимост между множество участници.
- **Полуструктурирано —** предварителни въпроси с гъвкавост за допълнителни; най-честото в практиката.
- **Неструктурирано —** свободна дискусия без сценарий; подходящо при ранно изследване на нова предметна област.

Стъпки за прилагане:

1. Определяне на цели и обхват — ясно дефинирайте какво трябва да разберете след интервюто.
2. Планиране и подготовка — изберете типа интервю, подгответе въпросник с отворени въпроси и уточняващи подкани.
3. Провеждане — изградете доверие, слушайте активно, задавайте уточняващи въпроси, водете бележки (или записвайте с разрешение).
4. Анализ и синтез — извличайте теми, болезнени точки и изисквания от бележките.
5. Валидиране и доклад — потвърдете констатациите с интервюираните; изгответе структуриран отчет.

Ключови принципи: Задавайте отворени въпроси. Избягвайте навеждащи формулировки. Уважавайте времето — придържайте се към 30–60 минути.

> **Практически съвет**
> Използвайте формули като: "Можете ли да опишете...?", "Какви трудности срещате с...?", "Защо смятате, че...?"
> Никога не задавайте въпроси, на които отговорът е 'Да' или 'Не' — те затварят разговора.

3.1.4. Фокус група (Focus Group)

Фокус групата е структурирана групова дискусия, водена от модератор, в която участват 6–12 представители на заинтересованите страни. Тя позволява едновременно събиране на разнообразни гледни точки и наблюдение на груповата динамика — нещо, което индивидуалното интервю не може да постигне.

Кога се прилага: При валидиране на концепции или прототипи, при изследване на потребителски нагласи и предпочитания, или когато е важно да се разбере как различните групи потребители виждат един и същи проблем по различен начин.

Стъпки за прилагане:

1. Планиране — дефинирайте цели, изберете 6–12 участника, определете модератор и подгответе ръководство с отворени въпроси.
2. Логистика — изберете удобно място (или виртуална платформа), уредете запис (ако е необходимо).
3. Провеждане — започнете със запознаване; задавайте отворени въпроси; следете да не доминира един участник; задълбочавайте с уточняващи въпроси.
4. Запис на ключови прозрения — водете детайлни бележки; улавяйте повтарящи се теми и нови идеи.
5. Анализ и доклад — идентифицирайте общи теми; кръстосвайте с данни от други техники; представете структуриран доклад.

> **Практически съвет**
> Ограничете сесията до 1–2 часа максимум — при по-дълги участниците губят концентрация.
> Осигурете психологически безопасна среда: напомнете, че няма грешни отговори.
> Като модератор останете неутрални — не изразявайте собствено мнение по темите.

3.1.5. Работна среща (Workshop)

Работната среща (Workshop) е структурирана среща с множество заинтересовани страни, чиято цел е съвместно решаване на проблем, събиране на изисквания или вземане на решения. За разлика от интервюто или фокус групата, работната среща е по-активна и резултатно ориентирана — в края на сесията се очаква конкретен артефакт или решение.

Кога се прилага: При сложни проекти, изискващи консенсус между множество отдели; при разработване на процесни карти, приоритизиране на изисквания или проектиране на решения.

Стъпки за прилагане:

1. Дефиниране на цели и обхват — изяснете каква е целта на срещата и какъв резултат се очаква.

2. Планиране на логистиката — изберете участниците (ключови заинтересовани страни, вземащи решения); определете формат (присъствен/виртуален) и продължителност; подгответе програма.
3. Водене на срещата — започнете със запознаване; представете контекста; използвайте интерактивни техники (мозъчна атака, SWOT, гласуване с точки); следете груповата динамика.
4. Документиране — записвайте решения и действия в реално време; използвайте визуални инструменти (бяла дъска, Miro).
5. Затваряне и следващи стъпки — обобщете резултатите; разпределете отговорности с крайни срокове; поискайте обратна връзка.
6. Следващи дейности — разпратете официален отчет; проследявайте изпълнението на действията.

3.1.6. Наблюдение (Observation)

Наблюдението е техника, при която аналитикът директно наблюдава потребителите в тяхната работна среда с цел разбиране на реалното им поведение, работни процеси и предизвикателства. Тя разкрива информация, която потребителите понякога не могат или не мислят да споделят по друг начин.

Видове наблюдение:

- **Активно (Participatory) —** аналитикът участва в процеса; получава по-дълбок контекст.
- **Пасивно (Non-Participatory) —** наблюдение без намеса; по-обективно, не влияе на поведението.
- **Структурирано —** използват се предварително дефинирани критерии и чеклисти.
- **Неструктурирано —** гъвкаво и отворено; подходящо за ново или неизследвано поле.

Стъпки за прилагане:

1. Дефиниране на цели — определете какво точно ще наблюдавате и защо.
2. Планиране — изберете правилната среда и момент; подгответе инструменти за наблюдение; информирайте участниците и вземете съгласие.
3. Провеждане — пристигнете рано; наблюдавайте активно или пасивно; водете детайлни бележки (времетраене на задачи, предизвикателства, взаимодействия).
4. Анализ и запис — идентифицирайте модели, тесни места и неефективности; сравнявайте с документацията.
5. Валидиране — обсъдете констатациите с наблюдаваните; уточнете неясни моменти.
6. Доклад — изгответе структуриран отчет с констатации, примери и препоръки.

Практически съвет
Бъдете ненатрапчиви — присъствието ви не трябва да влияе на естественото поведение на хората.
Наблюдавайте в реален контекст — не в изкуствена среда.
Кръстосвайте констатациите с интервюта или документи за потвърждение.

3.1.7. Мозъчна атака (Brainstorming)

Мозъчната атака е творческа групова техника за генериране на голям брой идеи за кратко време. Тя се основава на принципа на отложената критика — по време на генерирането на идеи не се допуска оценка или отхвърляне. Целта е количество, а не качество в началото.

Кога се прилага: При търсене на нови функционалности, решения на сложни проблеми, алтернативни подходи или когато екипът е в творчески застой.

Стъпки за прилагане:

1. Дефиниране на цел и подготовка — изяснете проблема; съберете разнородна група участници; подгответе инструменти (бяла дъска, стикери, Miro).
2. Подготовка на средата — изберете подходящо място; определете времева рамка (30–60 мин.); осигурете материали за улавяне на идеи.
3. Определяне на правила — насърчете всички идеи без изключение; забранете критиката по време на генерирането; насърчете надграждането върху чужди идеи.
4. Водене — започнете със запознаване; представете задачата; генерирайте идеи (round-robin, mind mapping или тихо писане); запишете всичко видимо за всички.
5. Приоритизиране и оценка — групирайте сходни идеи; гласувайте за най-перспективните; стеснете до приоритизиран списък.
6. Следващи стъпки — разработете план за действие за топ идеите; разпределете отговорности; проследявайте напредъка.

Техники за участие:

- **Round-Robin —** всеки участник предлага идея по ред; предотвратява доминирането на отделни гласове.
- **Mind Mapping —** централна тема с разклонения към подтеми; помага за структуриране на идеите.
- **Silent Brainstorming —** участниците пишат идеи мълчаливо преди споделяне; предотвратява групово мислене.

> **Практически съвет**
> Насърчавайте диви идеи — нереалистичните идеи могат да вдъхновят реалистични решения.
> Ротирайте фасилитатора при повтарящи се сесии за свежи гледни точки.
> Използвайте dot voting за бързо и справедливо приоритизиране в края.

3.1.8. Анализ на интерфейси (Interface Analysis)

Анализът на интерфейси е техника, при която аналитикът изследва как различни системи, приложения или компоненти взаимодействат помежду си. Целта е дефиниране на точките на интеграция, форматите на данни, протоколите за комуникация и изискванията за сигурност.

Видове интерфейси:

- **API интерфейси —** REST, SOAP, GraphQL; най-честите при уеб интеграции.
- **Системни интерфейси —** обмен на данни между вътрешни/външни системи (напр. ERP, CRM).
- **Хардуерни интерфейси —** свързаност между физически устройства и софтуер (напр. скенери, принтери).

Стъпки за прилагане:

1. Дефиниране на обхват — идентифицирайте кои системи взаимодействат; определете целта на анализа.
2. Идентифициране и категоризиране — класифицирайте всички интерфейси по тип.
3. Анализ на изисквания — идентифицирайте входни/изходни данни и трансформации; дефинирайте формати (JSON, XML, CSV); документирайте изисквания за сигурност и обработка на грешки.
4. Картиране на данните и работния процес — документирайте потока от данни; идентифицирайте зависимости и тригери; изградете диаграми на потока.
5. Идентифициране на рискове — анализирайте потенциални откази при интеграция; дефинирайте резервни механизми.
6. Валидиране с разработчиците и архитектите — потвърдете технически детайли и изисквания.
7. Документиране — изгответе Interface Specification Document (ISD) с всички детайли.

Практически съвет
Използвайте диаграми на последователности (sequence diagrams) за визуализация на интеракциите.
Включете разработчици и архитекти — без тях валидирането на технически детайли е невъзможно.
Планирайте за бъдещи интеграции — гъвкавостта от началото спестява скъпи промени по-късно.

3.1.9. Сравнение на техниките

Таблицата по-долу предоставя обобщен поглед върху всички седем техники и помага при избора на подходяща техника за конкретен контекст:

Техника	Тип инфо	Участие на хора	Времеемкост	Обекти-вност	Най-подходяща за
Анализ на документи	Структурирана	Минимално	Висока	Много висока	Начален анализ; съществуващи системи
Интервю	Контекстуална	Задължително	Умерена	Средна	Детайлни изисквания от конкретни роли
Фокус група	Групова / мнения	6–12 участника	Умерена	Средна	Валидиране на концепции; потребителски нагласи
Работна среща	Консенсусна	Множество роли	Висока	Висока	Сложни проекти; приоритизиране; процесни карти
Наблюдение	Поведенческа	Минимално	Висока	Много висока	Разбиране на реален работен процес

Техника	Тип инфо	Участие на хора	Времеемкост	Обекти-вност	Най-подходяща за
Мозъчна атака	Творческа / идеи	Група	Ниска	Ниска	Генериране на нови идеи и решения
Анализ на интерфейси	Техническа	Разработчици	Висока	Много висока	Интеграции между системи

Ключов принцип
Нито една техника не е достатъчна сама по себе си. Най-добрите резултати се постигат чрез комбиниране на техники: документалните (Анализ на документи, Анализ на интерфейси) осигуряват основата, комуникационните (Интервю, Фокус група, Работна среща) я обогатяват с контекст и мнения, а Наблюдението и Мозъчната атака разкриват скритото и иновативното.

3.2. Практическо решение: Система за онлайн библиотека

Примерна задача
Университетска библиотека желае да разработи нова онлайн система, която да замени настоящия ръчен процес на регистрация, заемане и връщане на книги. Новата система трябва да обслужва студенти, преподаватели и библиотекари и да се интегрира с университетската информационна система. Като бизнес анализатор трябва да съберете пълния набор от изисквания, като приложите всички налични техники.

3.2.1. Анализ на документи

Идентифицирани документи за анализ:

- Правилник за ползване на библиотеката
- Формуляр за регистрация на читател (хартиен)
- Карта за заемане на книга (хартиена)
- Каталог на книгите (Excel файл)
- Процедура за резервиране на книга
- Месечен отчет за просрочени книги
- Договор с предишен доставчик на библиотечен софтуер

Констатации от анализа

Документ	Ключова информация	Забележки / Пропуски
Правилник	Макс. 3 книги; 14-дневен срок; глоба 0.10 лв./ден	Не е актуализиран от 6 г. — правилата може да са се променили
Формуляр за регистрация	Три имена, ЕГН, факултет, специалност, год. обучение, имейл	Липсва поле за телефон — необходимо за SMS напомняния
Карта за заемане	ISBN, дати на заемане/връщане, подпис	Изцяло ръчен процес — без дигитален еквивалент

Документ	Ключова информация	Забележки / Пропуски
Каталог (Excel)	Инв. номер, автор, заглавие, брой наличности, местоположение	Няма поле за жанр, статус (заета/налична) и ISBN
Процедура за резервиране	Лична явка или телефон; резервация 48 часа	Липсва онлайн опция — ключов пропуск за новата система
Месечен отчет	Читатели, книги, дни закъснение — генериран ръчно	3–4 ч. ръчна работа — необходима е автоматизация

Извлечени изисквания от анализа на документи:

1. Системата поддържа регистрация с полета: три имена, ЕГН, факултет, специалност, год. обучение, имейл, телефон.
2. Читателят може да заема до 3 книги едновременно (при студенти).
3. Стандартен срок за заемане: 14 календарни дни.
4. Глоба при просрочие: 0.10 лв. за всеки ден закъснение.
5. Онлайн резервация на книга, валидна 48 часа.
6. Каталогът включва: инв. номер, ISBN, автор, заглавие, жанр, брой наличности, местоположение, статус.
7. Автоматично генериране на месечен отчет за просрочени книги.

3.2.2. Интервю

Участници и въпросник

Участник	Ключови въпроси
Главен библиотекар	1. Кои процеси отнемат най-много ресурси в момента? 2. Правилникът не е актуализиран от 2018 г. — има ли промени в правилата? 3. Как се обработват случаи на загубена или повредена книга? 4. Преподавателите имат ли различни права от студентите?
Библиотекар	1. Опишете типичния си работен ден — кои задачи се повтарят многократно? 2. Какви са най-честите оплаквания на читателите? 3. Как се справяте, когато книга е заета и друг я търси? 4. Колко дълго отнема генерирането на месечния отчет?
Студент	1. Колко често ползвате библиотеката и за какво? 2. Кои са най-големите неудобства при текущия процес? 3. Какви функции би трябвало да има онлайн системата? 4. Предпочитате ли напомняния по имейл, SMS или приложение?
ИТ координатор	1. Каква е текущата ИТ инфраструктура на университета? 2. Има ли съществуваща система за единен вход (SSO/LDAP)? 3. Какви са изискванията за сигурност и защита на данните? 4. Каква е очакваната натовареност на системата?

Резюме на резултатите от интервютата

Тема	Установено	Ново изискване
Права на достъп	Преподавателите имат право на 5 книги и 30-дневен срок	Различни потребителски профили: студент / преподавател / библиотекар / администратор
Напомняния	Студентите искат напомняне 3 дни преди срока	Автоматични имейл/SMS напомняния 3 дни преди изтичане на срока
Загубена книга	Плаща се двойна цена; записва се в регистър	Модул за управление на инциденти с финансов запис
Интеграция	Университетът има LDAP / Single Sign-On	Достъп само с университетски акаунт чрез SSO интеграция
Мобилен достъп	70% от студентите ползват предимно телефон	Адаптивен дизайн (responsive) или мобилно приложение

3.2.3. Фокус група

Организация
Провежда се фокус група с 8 участника: 5 студента от различни факултети, 2 преподаватели и 1 докторант. Модератор: бизнес аналитикът. Формат: присъствена среща, 90 минути.

Ключови теми от дискусията:

- **Търсене в каталога:** Всички участници посочват, че намирането на книга е бавно и неудобно. Искат филтриране по жанр, автор и наличност в реално време.
- **Чакален списък:** При заета книга участниците искат автоматично известяване при освобождаване — не само резервация за 48 часа.
- **История на заемания:** Студентите искат да виждат собствената си история — кои книги са заемали, кога и дали са ги върнали навреме.
- **Оценки и препоръки:** Иновативна идея: система за оценяване на книги от читателите и персонализирани препоръки.
- **Дигитални ресурси:** Няколко участника споменават нужда от достъп до дигитализирани версии на книги — не само физически.

Нови изисквания от фокус групата:

1. Списък на чакащите с автоматично известяване при освобождаване на заета книга.
2. Лична история на заемания за всеки потребител.
3. Система за оценяване на книги (рейтинг) от читателите.
4. Модул за дигитални ресурси (електронни книги / статии).

3.2.4. Работна среща (Workshop)

Цел и участници:
Провежда се половинднева работна среща (4 часа) с участието на: главния библиотекар, двама библиотекари, ИТ координатора, декан на студентски съвет и двама разработчици. Цел: приоритизиране на изискванията и проектиране на основните потоци на системата.

Резултати от работната среща:

Дейност на Workshop-а	Техника	Резултат
Картиране на текущия процес	Диаграма на процеса (Process Map)	Визуализиран AS-IS процес с идентифицирани тесни места
Приоритизиране на изисквания	Гласуване с точки	Топ 5 приоритетни функционалности определени от екипа
Проектиране на нов процес	SWOT анализ + групова дискусия	TO-BE процес с ясни стъпки и отговорности
Разпределение на действия	План на действие	5 действия с отговорни лица и срокове

Приоритизирани функционалности (резултат от dot voting):

1. Онлайн каталог с търсене в реално време — 24 гласа
2. Автоматични напомняния преди изтичане на срок — 21 гласа
3. SSO интеграция с университетската система — 19 гласа
4. Онлайн резервация и заемане — 18 гласа
5. Автоматично генериране на отчети — 16 гласа

3.2.5. Наблюдение

Планиране и провеждане:
Провеждат се 3 сесии на пасивно наблюдение (по 2 часа всяка) в реалната работна среда на библиотеката: сутринна сесия (пик на заемания), следобедна (пик на връщания) и сесия в края на семестъра (пик на натоварване).

Ключови наблюдения:

Категория	Наблюдение	Изискване/Препоръка
Процес на заемане	Средно 7 мин. на транзакция поради ръчно търсене в Excel	Скенер за баркод + автоматично обновяване на статуса при заемане
Опашки	В пиковите часове се образуват опашки от 8–12 студента	Самообслужващ терминал за заемане/връщане (self-service kiosk)
Грешки	3 грешки при ръчно попълване на карти за заемане за 2 ч.	Дигитален процес елиминира ръчните грешки; валидация на данни
Търсене на книги	Студентите питат библиотекаря средно 4 пъти на час за наличност	Публичен терминал за достъп до каталога в залата на библиотеката

3.2.6. Мозъчна атака

Организация:
Провежда се 45-минутна мозъчна атака с екипа на проекта (аналитик, 2 разработчика, дизайнер, представител на библиотеката). Инструмент: Miro (онлайн бяла дъска).

Генерирани и приоритизирани идеи

- **Геймификация (12 гласа):** значки и постижения за активни читатели; класация на най-четените книги.
- **AI препоръки (10 гласа):** алгоритъм, препоръчващ книги на база история на заеманията.
- **QR кодове (9 гласа):** всяка книга с QR код за бърз достъп до информация и заемане чрез телефон.
- **Клубове по интереси (7 гласа):** функционалност за създаване и управление на читателски клубове.
- **Интеграция с Moodle (6 гласа):** автоматично препоръчване на литература от учебните програми.

Прието за включване в обхвата (MVP): QR кодове за бързо заемане. Останалите идеи влизат в Product Backlog за бъдещи версии.

3.2.7. Анализ на интерфейси

Идентифицирани интерфейси:

Интерфейс	Тип	Протокол / Формат	Ключови изисквания
Библиотечна система ↔ SSO/LDAP	API (Системен)	LDAP / OAuth 2.0	Автентикация с университетски акаунт; ролева оторизация
Библ. система ↔ Унив. ИС (студентски данни)	API (REST)	JSON / HTTPS	Синхронизация на данни за студенти: факултет, специалност, статус
Библ. система ↔ Имейл сървър	Системен	SMTP / TLS	Изпращане на напомняния и известия; потвърждение за доставка
Библ. система ↔ SMS шлюз	API (REST)	JSON / HTTPS	SMS известия при просрочие и освобождаване на резервирана книга
Библ. система ↔ Баркод скенер	Хардуерен	USB HID / Bluetooth	Четене на ISBN/баркод при заемане и връщане; отговор < 500 ms
Библ. система ↔ Платежна система	API (REST)	JSON / HTTPS / PCI-DSS	Онлайн плащане на глоби; криптиране; потвърждение на транзакция

Идентифицирани рискове при интеграция:

- При недостъпност на SSO системата — достъп само за библиотекари с локален акаунт (fallback механизъм).
- При неуспешна SMS доставка — автоматично изпращане на имейл като резервен канал.
- При откази на баркод скенера — ръчно въвеждане на ISBN/инвентарен номер.

Изводи за студента-аналитик
Практическото решение демонстрира как седемте техники се допълват взаимно:
• Анализ на документи — дава формализираната основа (правила, процеси, ограничения).
• Интервю — добавя контекст, детайли и информация, недокументирана никъде.
• Фокус група — разкрива потребителски нагласи и генерира иновативни идеи чрез груповото взаимодействие.
• Работна среща — постига консенсус и приоритизира изискванията с участието на всички ключови страни.
• Наблюдение — разкрива реалния (а не декларирания) работен процес и скритите неефективности.
• Мозъчна атака — генерира иновативни идеи извън рамките на очевидното.
• Анализ на интерфейси — осигурява техническата пълнота на изискванията при интеграция.
Комбинирането им е гаранция за пълни, точни и валидирани изисквания.

3.3. Въпроси и задачи за упражнения

Следващите три казуса са проектирани така, че да ви помогнат да упражните и затвърдите прилагането на техниките за събиране на изисквания в различни бизнес контексти. За всеки казус са формулирани конкретни задачи, обвързани с отделни техники. Препоръчително е да работите по казусите в групи от 3–4 студента.

3.3.1. Казус: Разработване на система за интелигентно паркиране

Описание на казуса
Община в голям град желае да разработи система за интелигентно паркиране, която да замени настоящия ръчен контрол. Системата трябва да: следи в реално време кои паркоместа са свободни чрез IoT сензори; позволява на шофьорите да резервират паркомясто предварително чрез мобилно приложение; интегрира се с общинската информационна система за глоби; предоставя анализи и отчети за натовареността на паркингите по зони и часови пояси.

Задачи за изпълнение:

№	Техника	Условие на задачата	Очакван резултат
1	**Анализ на документи**	Проучете и анализирайте: (а) Наредбата за паркиране на вашия град; (б) Техническа спецификация на IoT сензори за паркиране (потърсете онлайн); (в) Примерна тарифа за платено паркиране.	Извлечете минимум 8 функционални изисквания. Идентифицирайте поне 3 пропуска или неясноти, които изискват допълнително изясняване.
2	**Интервю**	Проведете ролева игра: един студент играе ролята на: (а) Служител на общинската администрация; (б) Шофьор, ползващ платено паркиране ежедневно. Подгответе по 5 въпроса за всяка роля.	Доклад с ключови теми, болезнени точки и поне 5 нови изисквания, невидими от документите.

№	Техника	Условие на задачата	Очакван резултат
3	**Наблюдение**	Посетете физически паркинг в реалния свят (или симулирайте с видео от YouTube). Наблюдавайте 20 минути как шофьорите търсят паркомясто, плащат и напускат. Водете структуриран дневник на наблюденията.	Структуриран дневник с: времетраене на типичните действия, установени трудности и поне 3 изисквания, произтичащи пряко от наблюдението.
4	**Анализ на интерфейси**	Идентифицирайте всички системи, с които трябва да се интегрира платформата: IoT сензори, мобилно приложение, банков/платежен шлюз, общинска система за глоби, GPS/картна услуга.	Таблица на интерфейсите с: тип интеграция, протокол/формат, посока на данните и поне 2 риска за всеки интерфейс.
5	**Мозъчна атака**	В групата проведете 30-минутна мозъчна атака на тема: „Какви иновативни функции би могла да предложи системата за интелигентно паркиране?“ Използвайте техниката Silent Brainstorming + Dot Voting.	Списък с минимум 15 генерирани идеи; топ 3 приоритизирани идеи с обосновка; решение дали влизат в обхвата (in-scope/out-of-scope).
6	**Работна среща**	Организирайте и проведете 45-минутна симулирана работна среща с разпределени роли: общинска администрация, ИТ отдел, шофьори (потребители) и разработчици. Целта: приоритизиране на изискванията и договаряне на MVP обхват.	Протокол от срещата с: програма, взети решения, приоритизиран списък с изисквания (Must Have / Should Have / Could Have) и разпределение на следващите стъпки.

Краен продукт за предаване — Казус 3.3.1
1. Попълнена таблица с изисквания (минимум 15 функционални + 4 нефункционални).
2. Таблица на интерфейсите (минимум 5 интеграционни точки).
3. Протокол от симулираната работна среща.
4. Резюме: кои техники дадоха най-много стойност и защо.

3.3.2. Казус: Разработване на онлайн магазин (E-commerce платформа)

Описание на казуса
Малък семеен бизнес, занимаващ се с производство и продажба на ръчно изработени изделия, желае да създаде собствена онлайн платформа за продажба. В момента продажбите се осъществяват само чрез физически магазин и социални мрежи (Instagram, Facebook). Собственикът иска: онлайн каталог с продукти, количка и онлайн плащане; управление на поръчки и доставки; регистрация на клиенти с история на покупките; интеграция с куриерска фирма; базови маркетингови инструменти (промо кодове, бюлетин).

Задачи за изпълнение:

№	Техника	Условие на задачата	Очакван резултат
1	**Анализ на документи**	Анализирайте: (а) Политика за защита на личните данни на съществуващ онлайн магазин (напр. GDPR-съобразна политика);	Списък с изисквания, свързани с GDPR, правни задължения и бизнес правила. Идентифицирайте поне 4

№	Техника	Условие на задачата	Очакван резултат
		(б) Общи условия за ползване на платформа за електронна търговия; (в) Примерен ценоразпис и продуктов каталог (Excel/PDF).	правни/регулаторни изисквания и 6 функционални изисквания.
2	**Интервю**	Проведете ролева игра с двама участника: (а) Собственикът на бизнеса — съсредоточете се върху бизнес целите, ограниченията и болезнените точки; (б) Лоялен клиент на физическия магазин — съсредоточете се върху потребителското изживяване.	Два доклада от интервю (един за всяка роля) с ключови цитати, болезнени точки и изисквания. Сравнете дали двете гледни точки си противоречат.
3	**Фокус група**	Проведете 40-минутна симулирана фокус група с 5–6 члена на групата в ролята на потенциални онлайн купувачи. Ключови въпроси: „Какво ви кара да се доверите на нов онлайн магазин?“, „Какво ви разочарова при онлайн пазаруване?“, „Какви функции ви спестяват най-много време?“	Резюме с 5 ключови теми; списък с 8+ изисквания; анализ на груповата динамика (кой доминираше, кой беше пасивен, как се справихте).
4	**Анализ на интерфейси**	Идентифицирайте и документирайте интеграциите: платежен шлюз (Stripe/PayPal), куриерска фирма (Speedy/Econt), имейл маркетинг платформа (Mailchimp), социални мрежи (Meta/Instagram), счетоводен софтуер.	Interface Specification Document (ISD) с минимум 5 интеграции. За всяка: тип API, формат на данните, честота на обмен, изисквания за сигурност и резервен механизъм при отказ.
5	**Мозъчна атака**	Проведете мозъчна атака на тема: „Какво би направило този онлайн магазин различен от конкурентите?“ Използвайте Mind Mapping: централна тема → разклонения (Продукт, Доставка, Плащане, Лоялност, Маркетинг, Поддръжка).	Mind map (нарисуван или дигитален); списък с 20+ идеи; топ 5 с приоритет; обосновка защо останалите са отхвърлени или отложени.
6	**Работна среща**	Симулирайте Workshop за дефиниране на MVP (Minimum Viable Product). Роли: собственик на бизнеса, разработчик, дизайнер, маркетинг специалист. Използвайте техниката MoSCoW (Must/Should/Could/Won’t) за приоритизиране.	MoSCoW таблица с всички идентифицирани изисквания; дефиниран MVP обхват; обоснована аргументация защо дадено изискване е „Must Have“ а не „Could Have“.

Краен продукт за предаване — Казус 3.3.2

1. Консолидирана таблица с изисквания (минимум 20 функционални + 5 нефункционални).
2. ISD документ за минимум 5 интеграции.
3. Mind map от мозъчната атака.
4. MoSCoW таблица с дефиниран MVP обхват.
5. Сравнителен анализ: как гледните точки на собственика и клиента се различават.

3.3.3. Казус: Разработване на система за хотелски резервации

Описание на казуса
Верига от три бутикови хотела иска да замени настоящата система за резервации (ръчна + телефонна) с централизирана уеб-базирана платформа. Системата трябва да обслужва: директни резервации от гости (уебсайт и мобилно приложение); интеграция с онлайн туристически платформи (Booking.com, Airbnb); управление на стаи, цени и наличност; хотелски персонал (рецепция, домакинство); генериране на финансови отчети и анализи.
Всеки хотел има различни категории стаи, ценова политика и допълнителни услуги.

Задачи за изпълнение:

№	Техника	Условие на задачата	Очакван резултат
1	**Анализ на документи**	Анализирайте: (а) Примерен хотелски правилник и ценова листа; (б) Договор с онлайн платформа (Booking.com — публично достъпни общи условия); (в) Формуляр за регистрация на гост (изискван по закон); (г) Вътрешна процедура за почистване и поддръжка на стаи.	Таблица с извлечени изисквания по документ. Идентифицирайте: правните задължения (регистрация на гости), бизнес правилата (политики за анулиране, check-in/check-out) и интеграционните изисквания (Booking.com channel manager).
2	**Интервю**	Проведете ролеви интервюта с: (а) Управителя на хотела — стратегически цели, KPI и болезнени точки; (б) Рецепционист — ежедневни операции и чести грешки; (в) Гост, ползвал хотела — изживяване при резервиране и престой. Подгответе минимум 6 въпроса за всяка роля.	Три доклада от интервю. Консолидирана матрица на изискванията: за всяко изискване — кой го е споменал, каква е приоритетността му и дали е потвърдено от повече от един участник.
3	**Фокус група**	Организирайте фокус група с 6 участника в ролята на потенциални гости с различни профили: бизнес пътник, семейство с деца, двойка за уикенд, чуждестранен турист. Ключова тема: „Какво прави онлайн резервирането на хотел лесно или трудно?“	Резюме с теми по потребителски профил; сравнителна таблица: какво е важно за всеки тип гост; изисквания специфични за многоезичност, различни валути и специфични нужди (достъпност, паркинг, домашни любимци).
4	**Наблюдение**	Симулирайте наблюдение на рецепцията: един студент играе рецепционист, друг — гост с нестандартна заявка (ранен check-in, специална стая, промяна на резервация). Наблюдателите водят структуриран дневник с	Дневник на наблюдението с: хронология на действията, установени тесни места, ръчни стъпки, подходящи за автоматизация, и поне 5 изисквания, произтичащи от наблюдението.

№	Техника	Условие на задачата	Очакван резултат
		времетраене и идентифицирани проблеми.	
5	**Анализ на интерфейси**	Документирайте интеграциите: Channel Manager (Booking.com/Airbnb), платежен шлюз, система за електронни ключ-карти, имейл/SMS известия, счетоводна система, BI/Analytics платформа за отчети.	ISD документ с 6+ интеграции. Специален акцент върху: двупосочната синхронизация с Channel Manager (наличност, цени, резервации) и управлението на конфликти при едновременни резервации от различни канали.
6	**Работна среща**	Проведете Workshop за проектиране на TO-BE процеса на резервиране. Изградете Process Map (диаграма на процеса) — от търсене на стая до check-out. Използвайте техниката на SWOT анализ за оценка на рисковете при преминаване към новата система.	TO-BE process map (начертан или дигитален); SWOT анализ на прехода; Action Plan с минимум 6 действия, отговорни лица и срокове.
7	**Мозъчна атака**	Разгледайте само изискванията за достъпност и специализирани нужди (хора с увреждания, алергии, домашни любимци). Проведете Round-Robin мозъчна атака: всеки предлага по 2 идеи за как системата може да поддържа тези нужди.	Списък с идеи; топ 3 с план за реализация; обосновка как те повишават конкурентното предимство на хотела.

Краен продукт за предаване — Казус 3.3.3

1. Консолидирана таблица с изисквания (минимум 25 функционални + 6 нефункционални).
2. Матрица на изискванията с посочен участник-източник и степен на приоритет.
3. Сравнителна таблица на нуждите по тип гост (от фокус групата).
4. Рефлексия: кои техники се оказаха най-трудни за прилагане и защо.

Финален съвет

Успешното прилагане на техниките за събиране на изисквания не е механичен процес. То изисква аналитично мислене, активно слушане и умение да се свързват данните от различни източници в кохерентна картина. Всеки казус е различен — гъвкавостта при избора и комбинирането на техники е белег на опитен бизнес анализатор.

ГЛАВА 4
ФОРМАЛИЗИРАНЕ И МОДЕЛИРАНЕ НА ИЗИСКВАНИЯ

Контекстни диаграми. Use Case диаграми. Use Case спецификации.

4.1. Теоретични основи

4.1.1. Въведение в моделирането на изисквания

При работа с текстово описани изисквания, читателят неизбежно ги интерпретира през собствената си перспектива и опит. Често се прави опит за изграждане на мисловен модел на системата, базиран на прочетеното, което може да доведе до различни интерпретации при различни участници в процеса. Когато броят на изискванията е сравнително малък и управляем, е възможно да се запази цялостен поглед и добро разбиране на системата. С увеличаването на броя им обаче, тази способност постепенно намалява, като границата, при която това се случва, е индивидуална за всеки човек. Освен количеството, важна роля играят и фактори като сложността на изискванията, взаимовръзките между тях и нивото на абстракция.

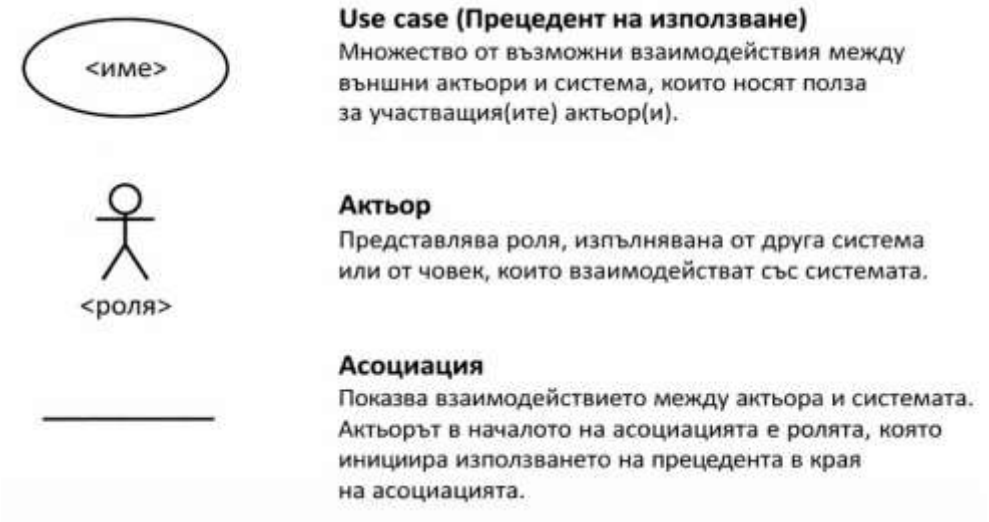
Поради тези причини често се налага изискванията, формулирани на естествен език, да бъдат прочитани многократно, за да се постигне пълно и коректно разбиране. Това се дължи на ограниченията на човешката когнитивна способност за обработка на информация, представена в неструктуриран текстов вид.

Моделирането на изисквания е ключов процес в софтуерното инженерство, който цели да представи системата по ясен и структуриран начин. Вместо да се разчита само на текстови описания, моделите използват визуални и формализирани представяния. Текстовите изисквания често страдат от двусмислие и различни интерпретации. Това може да доведе до грешки още в началните етапи на разработката. Моделите намаляват този риск чрез стандартизирани нотации. Комбинирането на текст и модели е най-ефективният подход. Докато текстът дава детайли, моделите осигуряват яснота и структура.

Моделът представлява **абстрактно** и опростено **представяне** на съществуваща част от реалността или на система, която предстои да бъде създадена. Той включва само онези елементи, които са релевантни за конкретната цел, като по този начин подпомага разбирането и анализа. Често моделите се представят визуално под формата на диаграми, които улесняват възприемането на изискванията и връзките между отделните компоненти.

Много модели са стандартизирани и намират широко приложение в различни области. Например, в архитектурата се използват информационни модели на сгради (BIM), които подпомагат проектирането, изграждането и управлението на строителни обекти. В електрониката стандартните схеми позволяват на специалистите да проектират, анализират и реализират електронни системи. Тази стандартизация улеснява обмена на знания и прилагането на добри практики между различни дисциплини.

Моделиращите езици, подобно на естествените езици, имат своя граматика и правила. Те дефинират както **допустимите елементи (синтаксис)**, така и тяхното **значение (семантика)** (фиг. 4.1). Това осигурява последователност и еднозначност при създаването и интерпретирането на модели, което е от съществено значение за ефективната комуникация между участниците в разработката.



Фиг. 4.1 Синтаксис (ляво) и семантика (дясно) за диаграма на потребителските случаи

Моделът на изискванията представлява концептуален модел, който изобразява изискванията към система, която предстои да бъде разработена, както и може да служи за представяне на съществуващата ситуация с цел анализ и разбиране на текущите проблеми. Като концептуален, моделът се характеризира с високо ниво на абстракция, при което реалността се свежда до нейната същност и се включват само релевантните елементи. В този контекст моделът е абстрактно представяне на част от реалността, което може да обхваща физически обекти, концепции или дори други модели.

В зависимост от целта си, моделирането може да бъде описателно или предписващо. Описателното моделиране се използва за представяне на съществуваща реалност и вече реализирани изисквания, докато предписващото моделиране описва желаното бъдещо състояние на системата и начина, по който то се различава от настоящото. Всеки модел се създава с конкретна цел – например за анализ на текуща система или за проектиране на нова, като тази цел определя нивото на детайлност и избора на включените елементи.

Реалността обикновено е сложна, поради което моделите я опростяват чрез редуциране на информацията. Това се постига чрез агрегиране (обобщаване и компресиране на данни) и селекция (избор на съществените детайли). По този начин моделите стават по-разбираеми и полезни за анализ, като позволяват на разработчиците да се фокусират върху ключовите аспекти на системата. Въпреки това, степента на опростяване трябва да бъде внимателно подбрана, тъй като прекомерното премахване на информация може да доведе до изкривяване на реалността.

Концептуалните модели често се изграждат чрез специализирани езици за моделиране, които могат да бъдат стандартизирани и формални. Пример за такъв широко използван език е UML (Unified Modeling Language), който осигурява обща нотация и улеснява комуникацията между участниците в разработката. Благодарение на тази стандартизация моделите са последователни, разбираеми и приложими в различни области.

Въпреки значителните предимства на визуалното представяне на изискванията чрез модели, този подход има и редица ограничения, които трябва да бъдат внимателно отчетени. Един от основните проблеми е свързан с поддържането на съгласуваност между различни модели, които описват различни аспекти на системата. Когато се

използват множество модели, е необходимо те да бъдат взаимно консистентни, което изисква висока степен на дисциплина, координация и контрол.

Допълнително предизвикателство представлява интегрирането на информацията от различни модели. За да се постигне пълно разбиране на системата и причинно-следствените връзки между нейните елементи, е необходимо всички модели да бъдат разгледани съвместно. Това може да увеличи сложността на анализа, особено при по-големи системи с множество взаимосвързани диаграми.

Ограниченият синтаксис на графичните езици за моделиране също поставя ограничения върху това каква информация може да бъде представена. Тъй като моделите са създадени с конкретна цел, не винаги е възможно всички аспекти на изискванията да бъдат изразени чрез тях. В такива случаи липсващата информация се допълва чрез текстови описания или отделни документи.

Накрая, моделите обикновено се фокусират основно върху функционалните изисквания. Изискванията, свързани с качество, ограничения или нефункционални характеристики, често не могат да бъдат адекватно представени чрез модели. Поради това те обикновено се описват чрез естествен език и се включват като допълващ елемент към моделите.

Моделите на изискванията се използват за представяне на системата от различни гледни точки, като всяка от тях подчертава определен аспект от нейната функционалност. При разработването на софтуерни системи функционалните изисквания обикновено се разглеждат чрез няколко основни перспективи, които допринасят за по-пълно и систематично разбиране.

Три основни перспективи при моделиране на функционални изисквания:

№	Перспектива	Описание и типични модели
1	**Структура и данни**	Фокус върху статичните характеристики — обекти, техните свойства и връзките между тях. Примери: диаграми на класове, E-R диаграми.
2	**Функции и потоци**	Акцент върху последователността от действия при обработка на данни. Примери: DFD диаграми, диаграми на дейности (Activity Diagrams).
3	**Състояние и поведение**	Описва динамиката на системата — как реагира на събития в зависимост от текущото си състояние. Примери: диаграми на състоянията, sequence diagrams.

Три критерия за качество на моделите:

Критерий	Какво измерва	Последствия при нарушение
Синтактично качество	Доколко моделът следва правилата на езика за моделиране.	Неясноти и погрешни интерпретации; грешни тестови случаи.
Семантично качество	Точността и пълнотата, с които моделът представя реалността.	Недоразумения между участниците; непълна спецификация.
Прагматично качество	Доколко моделът е подходящ за своята цел и аудитория.	Прекалено сложен или прекалено опростен модел за нуждите на проекта.

Ограничения на моделирането
Поддържането на съгласуваност между множество модели изисква висока дисциплина и координация.
Интегрирането на информация от различни модели може значително да увеличи сложността.
Ограниченият синтаксис на графичните езици не позволява изразяване на всички аспекти — особено нефункционалните изисквания.
Нефункционалните изисквания обикновено не могат да бъдат адекватно представени чрез модели.

4.1.2. Контекстна диаграма (Context Diagram) в структурирания анализ

Контекстните модели служат за онагледяване на взаимодействията между системата и нейната **външна среда**. Те отговарят на въпроса: с кого и как взаимодейства системата, без да навлизат в детайли за вътрешната ѝ логика.

Два основни инструмента за контекстно моделиране:

- **Диаграми на потока от данни (DFD) —** произлизат от структурирания анализ; показват системата като централен процес с входящи и изходящи потоци към външни участници.
- **UML диаграми на случаите на употреба (Use Case Diagrams) —** представят взаимодействието между актьорите и системата; допълват се с детайлни спецификации.

В структурирания анализ контекстната диаграма представлява специален вид диаграма на потока от данни (DFD), която изобразява системата като единен процес (фиг. 4.2). Тя служи за представяне на системата в нейната най-обща форма, като се поставя в центъра на диаграмата и се обозначава ясно с име. Основната ѝ цел е да покаже границите на системата и взаимодействията ѝ с външната среда.

Фиг. 4.2 Пример за диаграмите на потока от данни (DFD).

Външните участници в системата се представят чрез правоъгълници, наречени терминатори. Те могат да бъдат източници, които предоставят данни или услуги към

системата, или получатели, които приемат резултати от нея. Възможно е един и същ терминатор, например „Клиент“, да изпълнява и двете роли в зависимост от контекста. Взаимодействията между системата и терминаторите се изобразяват чрез стрелки, които показват потока на информация.

Всяка стрелка в диаграмата е означена с логическо име, което описва вида на предаваната информация, като например „данни за клиента“. На това ниво на абстракция детайлите са умишлено опростени и не се включват конкретни атрибути или технически средства за пренос (например уебсайт или електронна поща). Потокът може да включва както материални, така и нематериални елементи. Допълнителни детайли могат да бъдат добавени на по-късен етап, когато е необходимо по-задълбочено разбиране на системата от страна на заинтересованите страни.

Използването на диаграма на потока от данни (DFD) за моделиране на контекста на системата предоставя ясен и структуриран начин за идентифициране на взаимодействията със заобикалящата среда. Чрез нея могат да се разграничат интерфейсите между системата и външни участници като хора, отдели, организации или други системи. Това подпомага по-доброто разбиране на ролите и връзките в рамките на цялостната система.

Освен това, DFD позволява ясно да се определят обектите, които системата получава от своята среда, както и тези, които създава и предоставя обратно. Така се изгражда цялостна картина на входовете и изходите на системата, което е от съществено значение при дефинирането на изискванията и функционалността.

4.1.3. UML диаграми на случаите на употреба (Use Case Diagrams)

Диаграмата на случаите на употреба в UML представя **функционалността на системата** от гледна точка на нейните потребители и външни системи. Тя показва *какви услуги предоставя системата* и как участниците взаимодействат с нея.

Името на всеки случай на употреба обикновено се състои от глагол и съществително (например „Направи поръчка“), което дава кратко и ясно описание на съответната функция (фиг. 4.3).

Фиг. 4.3 Пример за UML диаграма на случаите на употреба.

Участниците (актьорите) представляват потребители или външни системи, които взаимодействат със системата. Актьорът, който инициира даден случай на употреба, е този, който получава ползата от неговото изпълнение. Връзката между актьора и случая на употреба се представя чрез асоциация, която показва наличието на взаимодействие, без да описва посока или поток от данни, както е при DFD диаграмите.

Диаграмата визуализира границата на системата чрез правоъгълник, в който са разположени случаите на употреба. Това ясно отделя вътрешната функционалност на системата от външните участници и подпомага определянето на обхвата на системата. По този начин се улеснява проверката дали всички основни функционалности са обхванати.

Шаблон за описание на Use Case (Use Case Specification)

Всеки случай на употреба се съпровожда от подробна спецификация, структурирана чрез стандартизиран шаблон. Тя включва контекст, участници, основен и алтернативни сценарии.

Поле	Описание
Use Case ID	Уникален идентификатор (напр. UC1, UC2).
Име	Кратка фраза с глагол в активна форма (напр. „Заеми книга“).
Основен актьор	Главният участник, инициирал сценария.
Цел	Каква стойност получава основният актьор при успешно изпълнение.
Заинтересовани страни	Всички участници с интерес към резултата от изпълнението.
Предварителни условия	Условия, изпълнени преди започване на потребителския случай.
Задействащо събитие	Събитието или действието, стартиращо изпълнението.
Основен поток (Main Flow)	Последователността от стъпки при нормалното (happy path) изпълнение.
Алтернативни потоци	Отклонения, грешки или специални случаи извън основния поток.
Постусловия	Крайно състояние след успешно или неуспешно изпълнение.
Допълнителна информация	Допълнителни бележки, ограничения или препратки.

Основните елементи на шаблона описват контекста и участниците в процеса. Предварителните условия и задействащото събитие определят кога и при какви обстоятелства започва изпълнението. Крайните условия описват резултата от изпълнението, независимо дали е успешно или не. Ядрото на шаблона е основният поток, който представя нормалното протичане на процеса стъпка по стъпка. Алтернативните

потоци и разширенията допълват този сценарий, като описват възможни отклонения, грешки или допълнителни поведения. Това позволява по-пълно покриване на всички възможни ситуации. Използването на такъв шаблон води до по-ясни, последователни и пълни изисквания. Той е особено полезен при сложни системи, където е важно да се разгледат всички възможни сценарии на взаимодействие между потребителите и системата.

Препоръки за качествено Use Case описание
Основният поток трябва да описва САМО нормалното изпълнение — без условни разклонения.
Всяко отклонение (грешка, специален случай) се описва в отделен алтернативен поток.
Когато сценариите са твърде сложни, допълнете с UML диаграма на дейностите или диаграма на последователността.
Именувайте потребителските случаи с глагол + обект: „Резервирай стая“, не „Резервация“.

4.2. Практическо приложение

Казус: Система за управление на онлайн библиотека

Описание на казуса
Система за управление на онлайн библиотека (Online Library Management System).
Основни функционалности: преглед на книги, заемане, връщане и добавяне на нова книга.
Участници: Потребител (User) и Администратор (Admin).

4.2.1. Контекстна диаграма (DFD)

Първата задача е да се представи **контекстна диаграма (DFD)** на система за управление на онлайн библиотека (фиг. 4.4). В центъра се намира системата („Online Library Management System“), която взаимодейства с външни участници и хранилище на данни. Целта на диаграмата е да покаже **какви данни влизат и излизат от системата**, без да се навлиза в детайли за вътрешната ѝ логика.

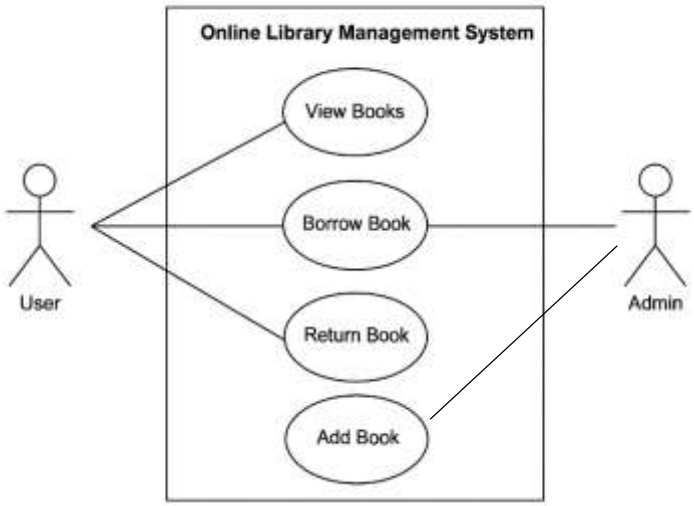
Фиг. 4.4 Контекстна диаграма (DFD) на система за управление на онлайн библиотека.

Основните външни участници са **потребителят (клиентско приложение)** и **администраторът**. Потребителят изпраща заявки към системата, като например преглед на книги, заемане или връщане. Системата обработва тези заявки и връща резултати, като списък с книги, потвърждение или съобщение за грешка. Администраторът от своя страна добавя нови книги към системата, което също е показано чрез поток от данни.

Диаграмата включва и **хранилище на книги**, което представлява база данни. Системата чете информация от него (например при търсене на книги) и записва нови данни (например при добавяне на книга). Това показва как системата управлява данните и поддържа актуално съдържание.

4.2.2. UML диаграма на случаите на употреба

Следващата диаграма е **UML диаграма на случаите на употреба (Use Case Diagram)** за система за управление на онлайн библиотека (фиг. 4.5). В центъра е системата („Online Library Management System“), оградена с правоъгълник, който показва нейната граница. Вътре са разположени основните функционалности (случаи на употреба), които системата предлага.



Фиг. 4.5 UML диаграма на случаите на употреба на система за управление на онлайн библиотека.

Диаграмата показва два основни актьора – **потребител (User)** и **администратор (Admin)**. Потребителят взаимодейства със системата чрез функции като преглед на книги, заемане и връщане на книги. Администраторът също взаимодейства със системата, като има достъп до определени функционалности (например управление или добавяне на книги).

Всеки овал в диаграмата представлява отделна функционалност на системата: „View Books“ (преглед на книги), „Borrow Book“ (заемане на книга), „Return Book“ (връщане на книга) и „Add Book“ (добавяне на книга). Линиите между актьорите и тези случаи показват кои участници използват съответните функции.

4.2.3. Use Case спецификации

И последно са описани два от случаите за употреба: заемане и добавяне на книга

UC1 — Borrow Book (Заемане на книга)

Поле	Стойност
Use Case ID	**UC1**
Име	Borrow Book — Заемане на книга
Основен актьор	User (Потребител)
Цел	Потребителят да може да заеме налична книга по нейното ID.
Заинтересовани страни	User, System
Предварителни условия	• Книгата съществува в системата. • Книгата е в статус „налична“. • Системата работи коректно (in-memory storage е активен).
Основен поток (Main Flow)	1. Потребителят изпраща заявка POST /books/{id}/borrow. 2. Системата получава ID на книгата. 3. Системата проверява дали книгата съществува. 4. Системата проверява дали книгата е налична. 5. Ако е налична, системата променя статуса ѝ на „заета“. 6. Системата връща потвърждение за успешно заемане.
Алтернативни потоци	**A1 — Книгата не съществува:** 1. На стъпка 3 системата открива, че ID не съществува. 2. Системата връща: "Book not found" (404). **A2 — Книгата е вече заета:** 1. На стъпка 4 системата открива статус „заета“. 2. Системата връща: "Book is already borrowed".
Постусловия	• При успех: статусът на книгата е „заета“. • При грешка: състоянието остава непроменено.

UC2 — Add Book (Добавяне на книга)

Поле	Стойност
Use Case ID	**UC2**
Име	Add Book — Добавяне на книга
Основен актьор	Admin (Администратор)
Цел	Администраторът да добави нова книга в каталога.
Заинтересовани страни	Admin, System
Предварителни условия	• Администраторът има право да добавя книги. • Данните за книгата включват заглавие, автор и валидни стойности.
Основен поток (Main Flow)	1. Admin изпраща заявка POST /books с данните за новата книга. 2. Системата валидира входните данни (заглавие, автор). 3. Системата генерира уникално ID за книгата. 4. Системата записва книгата в хранилището (in-memory). 5. Системата връща потвърждение с данните на новата книга.

Поле	Стойност
Алтернативни потоци	**A1 — Невалидни входни данни:** 1. На стъпка 2 системата открива липсващи или невалидни полета. 2. Системата връща съобщение: "Invalid book data".
Постусловия	• В системата съществува нова книга с уникално ID.

4.3. Въпроси и задачи за упражнения

Следващите три казуса изискват прилагане на техниките за моделиране на изисквания — контекстни DFD диаграми, Use Case диаграми и Use Case спецификации. За всеки казус следвайте методологията, демонстрирана в практическото приложение (т. 4.2).

Напомняне — Стъпки при моделиране на изисквания

1. Идентифицирайте актьорите.
2. Дефинирайте входовете и изходите на системата (контекстна DFD).
3. Идентифицирайте функционалностите (use cases) и ги свържете с актьорите.
4. Опишете поне един use case детайлно чрез шаблона за спецификация.
5. Дефинирайте основния и алтернативните потоци за всеки описан use case.

4.3.1. Казус: Разработване на система за интелигентно паркиране

Условие за изпълнение

Да се разработи модел на система за интелигентно паркиране, която подпомага шофьорите при намиране, резервиране и използване на паркоместа в градска среда. Потребителите проверяват налични паркоместа, резервират място и получават информация в реално време. Администраторът управлява паркинг зоните и следи системата.

Изисквания към студентите:

1. Идентифицирайте актьорите (напр. шофьор, администратор, IoT сензори).
2. Създайте контекстна DFD диаграма — покажете входовете и изходите на системата, терминаторите и потоците от данни.
3. Изградете Use Case диаграма — включете всички основни функционалности и ги свържете с правилните актьори.
4. Опишете поне един Use Case чрез шаблона за спецификация (с предварителни условия, основен и алтернативен поток, постусловия).
5. Дефинирайте основните входове и изходи на системата в таблична форма.

4.3.2. Казус: Разработване на онлайн магазин (E-commerce платформа)

Условие за изпълнение

Да се моделира система за онлайн търговия, която позволява на потребителите да разглеждат продукти, да правят поръчки и да извършват плащания.
Системата трябва да поддържа управление на продукти, обработка на поръчки и комуникация с външни услуги (напр. плащания и доставка). Администраторът управлява продуктите и поръчките.

Изисквания към студентите:

1. Да се идентифицират всички актьори (клиент, администратор, платежна система и др.).
2. Да се създаде контекстна диаграма (DFD).
3. Да се разработи Use Case диаграма.
4. Да се опишат минимум два Use Case.
5. Да се разграничат основен и алтернативни потоци.

4.3.3. Казус: Разработване на система за хотелски резервации

Условие за изпълнение

Да се разработи модел на система за хотелски резервации, която позволява на потребителите да търсят свободни стаи, да правят резервации и да получават потвърждение.

Системата обработва наличности, резервации и плащания, предоставя информация в реално време. Администраторът управлява стаите и резервациите.

Изисквания към студентите:

1. Идентифицирайте актьорите — клиент, администратор, платежна система, Channel Manager (Booking.com).
2. Създайте контекстна DFD диаграма с всички потоци от данни.
3. Изградете Use Case диаграма с пълния набор от функционалности.
4. Опишете поне един детайлен Use Case по шаблона. Фокусирайте се върху сценария с двойна резервация като алтернативен поток.

ГЛАВА 5
МОДЕЛИРАНЕ НА СТРУКТУРА И ПОВЕДЕНИЕ

Клас диаграми. Диаграми на последователността. Консистентност.

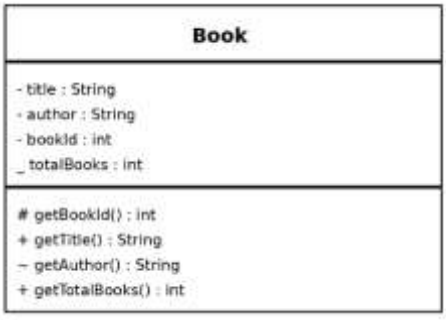
5.1. Теоретични основи

В процеса на разработка на софтуерни системи моделите играят ключова роля за представяне на **структурата на данните** и **поведението на системата**. Двата основни взаимодопълващи се типа модели са **диаграма на класовете** и **диаграмите на последователността**. Те заедно осигуряват цялостна представа за системата — *„какво е системата“* и *„как работи тя“*.

Тип модел	Какво описва
Клас диаграма	Статичната структура — какви класове съществуват, какви данни съдържат и как са свързани помежду си.
Диаграма на последователността	Динамичното поведение — как обектите (инстанции на класовете) взаимодействат във времеви ред при изпълнение на конкретен сценарий, т.е. как системата се държи при изпълнение на конкретен сценарий.

5.1.1. Диаграма на класовете — моделиране на структурата на данните

Клас диаграмите са основен инструмент за представяне на структурата на системата. Те показват класовете, техните атрибути (данни) и операции (методи), както и връзките между тях. Всеки клас се представя като правоъгълник, разделен на три секции: название, атрибути и методи (фиг. 5.1).



Фиг. 4.1 UML клас.

Анатомия на UML клас:

Секция	Съдържание и формат
Название (Name)	Горната секция — съдържа името на класа (главна буква, CamelCase).
Атрибути (Attributes)	Средна секция. Формат: `видимост име : тип [множество] = стойностПоПодразбиране`
Методи (Methods)	Долна секция. Формат: `видимост име(параметри) : типНаВръщане`

Видимост на атрибути и методи:

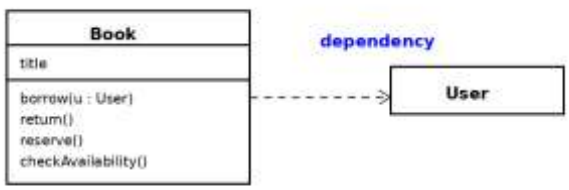
Символ	Видимост	Значение
+	Public	Достъпен от всички класове.
#	Protected	Достъпен само от класа и неговите наследници.
-	Private	Достъпен само в рамките на самия клас.
~	Package / Default	Достъпен само в рамките на пакета.

Видове връзки между класовете

Връзките между класовете са ключови за разбирането на структурата. Различните типове връзки отразяват различни форми на зависимост между класовете.

Тип връзка	UML нотация	Характеристика
Зависимост (Dependency)	Пунктирана стрелка (→)	Временна връзка — единият клас използва другия само при изпълнение на операция.
Асоциация (Association)	Права линия (с или без стрелка)	Структурирана логическа връзка между два класа. Може да включва мултиплицитет.
Обобщение (Generalization)	Линия с триъгълна стрелка към базовия клас	Йерархия „is-a-kind-of“. Наследникът притежава всички атрибути и методи на родителя.
Агрегация (Aggregation)	Линия с ◇ (празен ромб) при цялото	По-слаба „цяло-част“ — частите могат да съществуват независимо от цялото.
Композиция (Composition)	Линия с ◆ (запълнен ромб) при цялото	По-силна „цяло-част“ — частите не могат да съществуват без цялото. Споделен жизнен цикъл.

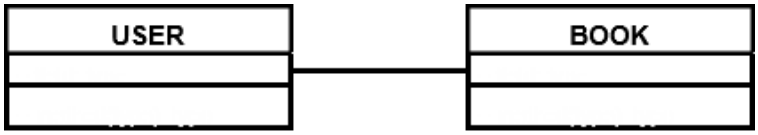
Фиг. 5.2 показва **зависимост (dependency)** между класовете **Book** и **User** в контекста на библиотечна система. Класът **Book** използва класа **User** в своя метод borrow(u : User). Това означава, че за да бъде заета една книга, е необходимо да се подаде обект от тип потребител. Тази зависимост е временна и възниква само при изпълнение на конкретна операция, без да създава постоянна връзка между двата класа. Пунктираната линия със стрелка показва, че **Book зависи от User**. Ако структурата на класа User се промени (например параметрите или типът му), това може да наложи промени и в класа Book.



Фиг. 5.2 Зависимост между класове в UML диаграмата на класовете.

Асоциацията представлява структурирана връзка между два или повече класа, която определя как те са свързани и взаимодействат помежду си. Тя показва наличието на

логическа зависимост между обектите, без да уточнява конкретния начин на реализация. Чрез асоциацията се моделира „кой с кого е свързан“ в рамките на системата.



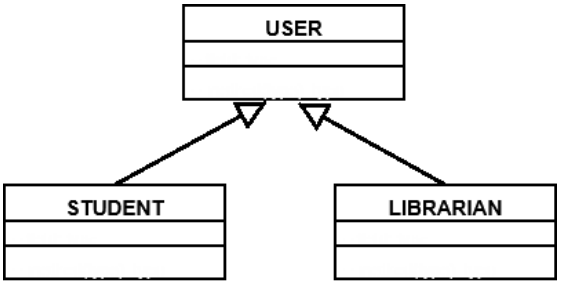
Фиг. 5.3 Асоциацият между класове в UML диаграмата на класовете.

Когато асоциацията свързва точно два класа, тя се нарича **бинарна асоциация**. Това е най-често срещаният тип връзка, например между класовете „Потребител“ и „Книга“, където потребителят може да заема книга (фиг. 5.3). В този случай връзката описва взаимодействие между два участника.

Когато една асоциация свързва повече от два класа, тя се нарича **n-арна асоциация**. Този тип връзка се използва при по-сложни ситуации, където взаимодействието включва няколко участника едновременно.

Обобщението (**Generalization**) представлява връзка между по-общо понятие (базов клас) и по-специализирано понятие (дъщерен клас). Тази връзка се описва като **„is-a-kind-of“**, т.е. един клас е вид на друг. Чрез нея се моделира йерархия от класове, в която общите характеристики се дефинират на по-високо ниво, а специфичните – в наследниците. Графично, обобщението в UML се представя чрез линия със затворена (празна) триъгълна стрелка, насочена към базовия клас, което ясно показва посоката на наследяване (фиг. 5.4).

Основна характеристика на обобщението е, че дъщерният клас може да се използва навсякъде, където се използва базовият клас. Дъщерният клас наследява всички атрибути и операции на базовия клас и може да ги разширява с допълнителни елементи. Това означава, че той е съвместим с него по отношение на поведението и интерфейса. Обратното обаче не е валидно – базовият клас не може да замести дъщерния, тъй като не съдържа неговите специфични характеристики.



Фиг. 5.4 Обобщение между класове в UML диаграмата на класовете.

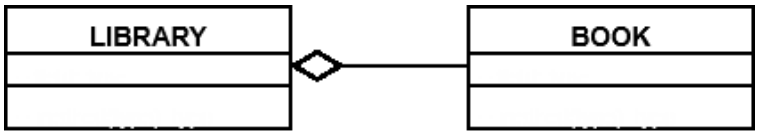
В примера на фиг. 4.3 User е общият клас, а Student и Librarian го наследяват. Това означава, че наследниците получават всички атрибути и методи на родителя и могат да добавят нови или да променят съществуващи.

Агрегацията (**Aggregation**) представлява специален вид асоциация, която моделира връзка от тип **„цяло–част“** между един обект (агрегат) и неговите съставни части. Тя се използва, когато е необходимо да се покаже, че дадени елементи принадлежат към по-голяма структура, но същевременно могат да съществуват независимо от нея.

При агрегацията връзката е **по-слаба форма на композиция**, тъй като частите не са напълно зависими от цялото. Те могат да бъдат споделяни между различни обекти или

да съществуват самостоятелно извън контекста на агрегата. Това означава, че връзката е по-скоро логическа, отколкото строго жизнено обвързана.

Важно е, че при агрегацията принадлежността на частите към цялото се реализира **чрез препратка (reference)**, а не чрез притежание. Това означава, че агрегатът „съдържа“ обектите, но не контролира напълно техния жизнен цикъл. В UML тази връзка се обозначава с **празен ромб (◇)** в края на класа, който представлява цялото.



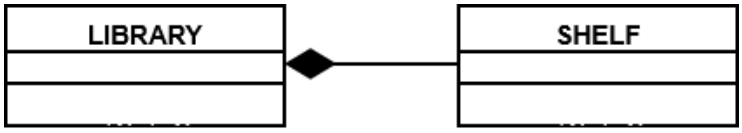
Фиг. 5.5 Агрегация между класове в UML диаграмата на класовете.

На фиг. 5.5. е показан пример с връзката агрегацията, т.е. библиотеката съдържа книги, но книгите могат да съществуват и извън нея.

Композицията (Composition) представлява по-силна форма на агрегация, при която съществува **силна зависимост между цялото и неговите части**. Тя моделира връзка от тип „цяло–част“, при която частите не могат да съществуват самостоятелно извън контекста на агрегата. Това означава, че жизненият цикъл на частите е изцяло обвързан с този на цялото.

Тази връзка се използва, когато дефиницията на цялото е непълна без неговите части и съществува силна взаимозависимост между тях. За разлика от агрегацията, при композицията принадлежността се реализира **по стойност (by value)**, което означава пълен контрол върху частите. В UML композицията се обозначава с **запълнен ромб (◆)** в края на класа, който представлява цялото (фиг. 5.6).

Една от основните характеристики на композицията е, че **частите принадлежат изцяло на агрегата** и не могат да бъдат споделяни с други обекти. Множествеността от страната на агрегата не може да надвишава едно, което означава, че всяка част е свързана само с един „собственик“. Освен това, веднъж създадена, тази връзка не се променя – частите не могат да бъдат прехвърляни към друг агрегат.



Фиг. 5.6 Композиция между класове в UML диаграмата на класовете.

Композицията предполага и **силна зависимост при създаване и унищожаване** – когато агрегатът бъде създаден, се създават и неговите части, а когато бъде унищожен, частите също престават да съществуват. В примера на фиг. 5.6 рафтовете (Shelf) съществуват само като част от библиотеката (Library).

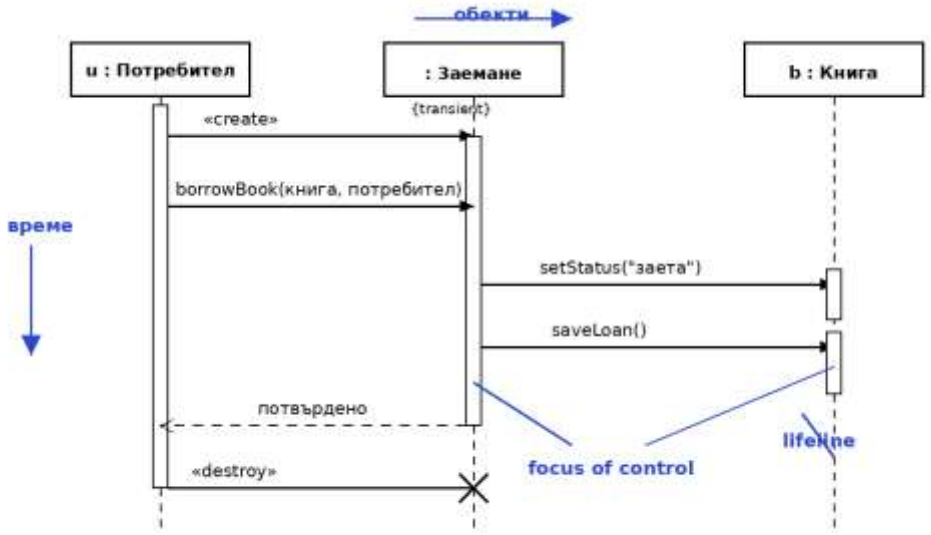
Агрегация vs. Композиция — ключовата разлика
Агрегация (◇): частите МОГАТ да съществуват самостоятелно. Реализира се чрез препратка (reference).
Композиция (◆): частите НЕ МОГАТ да съществуват без цялото. Реализира се по стойност (by value).
При композиция: жизненият цикъл на частите е изцяло обвързан с цялото.
При композиция: множествеността от страната на агрегата не може да надвишава 1.

5.1.2. Диаграми на последователността — моделиране на поведението

Диаграмата на последователността (Sequence Diagram) описва начина, по който обектите в една система взаимодействат помежду си във времето. Тя представя последователността от съобщения между обектите в хронологичен ред и показва как протичат процесите в рамките на даден сценарий. Чрез нея се моделира динамичното поведение на системата и се проследява как се изпълняват отделните операции.

Основните елементи включват (фиг. 5.7):

- Обекти (инстанции на класове)
- Жизнени линии (lifelines)
- Съобщения (messages) – описват извиквания на методи
- Актьори – външни участници



Фиг. 5.7 Диаграми на последователността в UML.

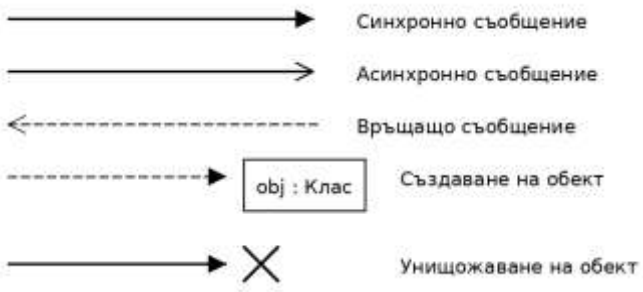
Обектите в диаграмата се представят чрез вертикални пунктирни линии, наречени **жизнени линии (lifelines)**. Те показват съществуването на обекта в определен времеви интервал. Всеки обект се означава с име и клас, като записът е от вида: имеНаОбект : имеНаКлас.

Актьорите представляват външни участници в системата (например потребители или други системи) и обикновено се разполагат в най-лявата или най-дясната част на диаграмата. Това улеснява проследяването на взаимодействието между системата и нейната среда.

Съобщенията са основният механизъм за комуникация между обектите. Те се изобразяват като хоризонтални стрелки от една жизнена линия към друга и показват извикване на операция или предаване на информация. Всяко съобщение има име и може да включва параметри или номер, който указва реда на изпълнение. Диаграмите на последователността използват различни видове съобщения, за да покажат как обектите взаимодействат помежду си (фиг. 5.8):

- Синхронно съобщение - представя извикване на операция, при което изпращащият обект изчаква резултат. Това означава, че изпълнението е блокиращо – процесът продължава едва след като получи отговор.

- Асинхронно съобщение - изпращащият обект не изчаква отговор и продължава изпълнението си. Използва се при паралелни или независими операции.
- Връщащо съобщение - показва връщането на резултат от извикана операция. Изобразява се с пунктирна линия и отразява отговора на предходно съобщение.
- Създаване на обект (create) - показва създаването на нов обект по време на изпълнение. Обектът започва да съществува от момента на получаване на това съобщение.
- Унищожаване на обект (destroy) - показва края на съществуването на обект. Обозначава се с „X“ върху жизнената линия.



Фиг. 5.8 Видове съобщения в диаграмата на последователността в UML.

Сценариите (scripts) представляват текстово описание на последователността от събития, която се моделира в диаграмата. Те подпомагат разбирането на логиката и често се използват заедно с диаграмата, за да се даде по-пълна представа за поведението на системата.

5.1.3. Връзка между клас диаграма и диаграма на последователността

Двата типа диаграми се използват съвместно, за да се гарантира, че системният дизайн е консистентен и пълноценен. Тяхната взаимовръзка е фундаментална за качеството на проектирането.

Клас диаграма (Структура)	Диаграма на последователността (Поведение)
Дефинира КАКВО съществува: класове, атрибути, методи, връзки.	Показва КАК се изпълнява: взаимодействие между обекти при конкретен сценарий.
Обектите в sequence диаграмата са ИНСТАНЦИИ на класовете от клас диаграмата.	Съобщенията в sequence диаграмата съответстват на МЕТОДИ от клас диаграмата.

Правила за консистентност между диаграмите
Всеки метод в клас диаграмата трябва да се появи поне веднъж като съобщение в sequence диаграмата.
Ако в sequence диаграмата има съобщение без съответен метод в клас диаграмата — дизайнът е НЕКОНСИСТЕНТЕН.
Ако в клас диаграмата има метод, който не се използва в нито една sequence диаграма — методът може да е ИЗЛИШЕН.
Нарушенията на консистентността са чести причини за грешки в по-късните фази на разработката.

5.2. Практическо приложение

Казус: Мобилно приложение за разпознаване на настроение и генериране на плейлист

Описание на казуса
Да се проектиране диаграма на класовете и диаграма на последователността на мобилно приложение за разпознаване на настроение и генериране на музикален плейлист. Приложението използва уеб камера, за да заснеме потребителя, разпознава лицето му, предсказва настроението и на базата на това настроение генерира подходящ музикален плейлист.

Основната функционалност на приложението е:

1) Потребителят отваря приложението.
2) Устройството (телефон/компютър) получава достъп до уеб камерата.
3) Уеб камерата заснема изображение на потребителя.
4) Потребителят избира опцията "Открий лице и предскажи настроение".
5) Приложението изпраща заявка към базата данни за възможните настроения (sentiments).
6) Базата данни връща настроението (например: тъжно, радостно, неутрално, ядосано).
7) Приложението показва настроението на потребителя.
8) Приложението изпраща заявка към базата данни за музика, подходяща за това настроение.
9) Базата данни генерира плейлист (списък от песни).
10) Приложението показва генерирания плейлист на потребителя.

Диаграма на класовете и диаграма на последователността включва следните класове: User, Device и Database (фиг. 5.9).

Фиг. 5.9 Диаграма на класовете на мобилно приложение за разпознаване на настроение и генериране на музикален плейлист.

Класът User представя „потребителя на приложението“ – човекът, който взаимодейства със системата. В контекста на UML, `User` често играе ролята на актьор (actor) в диаграма на последователността, но в клас диаграмата може да бъде представен като нормален клас, ако системата пази данни за него. Атрибутите са:

- name: String - името на потребителя (пример: "Иван");
- moodHistory: List<String> - история на предсказаните настроения (пример: ["радостен", "тъжен"]).

Основните методи са:

- openApp() - симулира или изпълнява действието на потребителя да стартира приложението.
- detectFace() - е „тригер“ за основната функционалност – разпознаване на лице и настроение.

Класът Device представлява хардуерното устройство (телефон, компютър, таблет), на което работи приложението. Той управлява уеб камерата, дисплея и комуникацията с базата данни. Основните атрибути са:

- cameraStatus: Boolean - дали камерата е достъпна (true/false);
- webcam: Object - обект, представляващ уеб камерата.

Методите на класа са:

- accessWebcam() - заявява достъп до уеб камерата на устройството. Проверява дали има разрешение.
- getPhoto() - заснема снимка от уеб камерата и я запазва временно.
- retrieveMood(photo) - изпраща заснетата снимка към базата данни (или AI модел), за да определи настроението.
- displayMood(mood) - показва на екрана на устройството предсказаното настроение (напр. "Твоето настроение е: радостен").
- retrieveMusic(mood) - изпраща заявка към базата данни за музика, подходяща за даденото настроение.
- getPlaylist(playlist) - показва генерирания плейлист на потребителя (списък от песни).

Класът Database представлява базата данни (или външна услуга), която съхранява настроенията, музикалната библиотека и правилата за генериране на плейлисти. Основните атрибути са:

- sentiments: List<String> - списък от възможни настроения (пример: ["радостен", "тъжен", "ядосан", "неутрален"]);
- musicLibrary: Dictionary - речник, който свързва настроение със списък от песни (пример: {"радостен": ["Song1", "Song2"], ...}) .

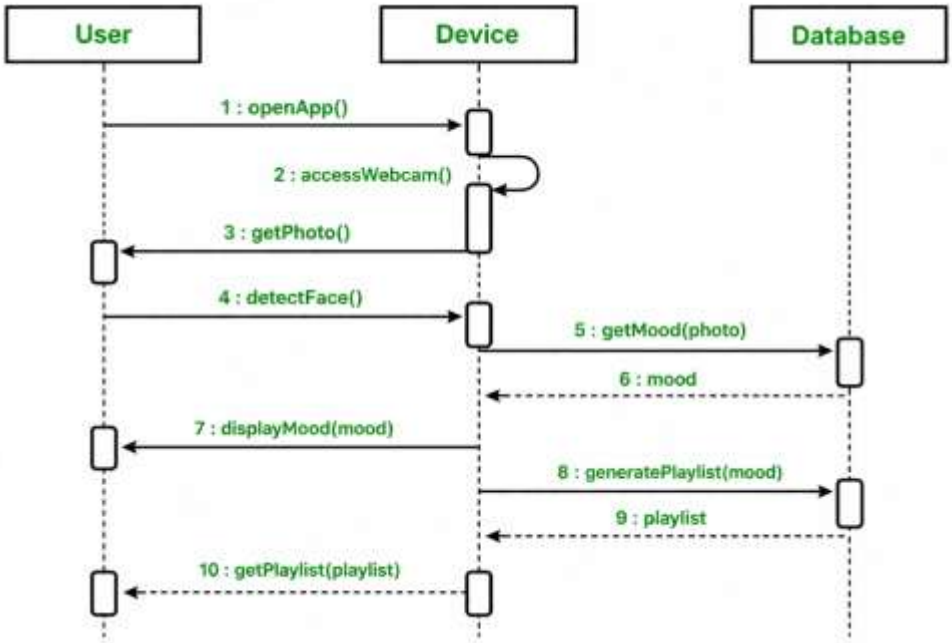
Методите на класа са:

- getMood(photo) - получава снимка, анализира я (например чрез AI модел) и връща настроението като текст.
- generatePlaylist(mood) - въз основа на подаденото настроение, генерира списък от песни от musicLibrary.

При създаването на диаграма на последователността е добре да се спазят следните изисквания:

- Всеки метод в диаграмата на класовете трябва да се появи поне веднъж като съобщение в диаграма на последователността.
- Ако в диаграма на последователността има съобщение, което не съответства на метод в нито един клас – дизайнът е неконсистентен.
- Обратно, ако в диаграмата на класовете има метод, който никога не се използва в диаграма на последователността – той може да е излишен.

На фиг. 5.10 е показана диаграмата на последователността



Фиг. 5.10 Диаграма на последователността на мобилно приложение за разпознаване на настроение и генериране на музикален плейлист.

5.3. Въпроси и задачи за упражнения

За всяка задача студентите трябва да изпълнят следните четири изисквания:

- **Диаграма на класовете —** включете всички класове с атрибути и методи, и връзките между тях.
- **Диаграма на последователността —** покажете пълния сценарий чрез съобщения между обектите.
- **Кратък анализ (5–10 изречения) —** обяснете как методите от клас диаграмата се използват като съобщения в sequence диаграмата.
- **Поне едно нарушение на консистентността —** посочете го и обяснете как бихте го открили.

Препоръки за изпълнение
Спазвайте правилото: всеки метод в клас диаграмата → поне едно съобщение в sequence диаграмата.
Именувайте обектите в sequence диаграмата по конвенцията: driver:Driver, server:ParkingServer.
Разграничете синхронните съобщения (изчакване на отговор) от асинхронните (без чакане).
Дефинирайте поне един алтернативен поток (напр. грешка при недостъпност на услуга).

5.3.1. Казус: Разработване на система за интелигентно паркиране

Условие на задачата
Система за интелигентно паркиране в голям град. Системата следи свободните места, позволява на шофьорите да резервират предварително и ги напътства до избрания паркинг.

Функционален поток (Сценарий):

1. Шофьорът отваря мобилното приложение.
2. Системата получава текущата локация на шофьора.
3. Приложението изпраща заявка към сървъра за налични паркинги в района.
4. Сървърът връща списък с паркинги и свободни места.
5. Шофьорът избира конкретен паркинг и час за резервация.
6. Системата проверява дали мястото е свободно в избрания час.
7. Системата запазва мястото временно (за 15 минути).
8. Системата потвърждава резервацията.
9. Системата показва маршрут до паркинга.

Въпроси за консистентност — Казус 5.3.1
1. Посочете за всяко съобщение в sequence диаграмата от кой клас и метод идва.
2. Какво нарушение на консистентността възниква, ако в клас ParkingServer липсва методът checkAvailability()?
3. Ако шофьорът може да анулира резервация, каква промяна в клас диаграмата е необходима?

5.3.2. Казус: Разработване на онлайн магазин (E-commerce платформа)

Условие на задачата
Онлайн магазин, в който потребителите разглеждат продукти, добавят ги в количка и завършват покупка. Системата поддържа търсене, управление на количка и проверка на наличност.

Функционален поток (Сценарий):

1. Потребителят отваря уебсайта/приложението.
2. Системата зарежда началната страница с категории продукти.
3. Потребителят избира категория (напр. "Телефони").
4. Системата изпраща заявка към базата данни за продуктите в тази категория.
5. Базата данни връща списък с продукти (име, цена, наличност).
6. Потребителят добавя продукт в количката.
7. Системата актуализира количката.
8. Потребителят натиска "Поръчка".
9. Системата проверява наличността на продуктите в количката.
10. Системата потвърждава поръчката и показва общата сума.

Въпроси за консистентност — Казус 5.3.2
1. Кой метод от ShoppingCart се използва, когато потребителят добавя продукт?
2. Ако в sequence диаграмата има съобщение calculateTotalPrice(), но в клас диаграмата на ShoppingCart този метод ЛИПСВА — какво трябва да се направи?
3. Ако плащането се обработва от PaymentGateway, как се отразява това в клас диаграмата?

5.3.3. Казус: Разработване на система за хотелски резервации

Условие на задачата
Система за хотелски резервации, позволяваща на клиенти да търсят свободни стаи, да резервират и да управляват своите резервации. Включва интеграция с платежен шлюз.

Функционален поток (Сценарий):

11. Клиентът отваря системата (уеб или мобилно приложение).
12. Клиентът въвежда дати на настаняване и напускане.
13. Системата изпраща заявка към базата данни за свободни стаи за тези дати.
14. Базата данни връща списък със стаи (тип, цена, капацитет).
15. Клиентът избира стая и въвежда лични данни.
16. Системата създава временна резервация.
17. Системата изпраща заявка за плащане към платежния шлюз.
18. Платежният шлюз потвърждава плащането.
19. Системата потвърждава резервацията и показва нейния номер.

Въпроси за консистентност — Казус 5.3.3
1. Кой метод на PaymentGateway се извиква в sequence диаграмата?
2. Ако в клас диаграмата на HotelSystem ЛИПСВА методът createTemporaryReservation(),
   може ли sequence диаграмата да бъде коректна? Обяснете защо.
3. Какви промени в клас диаграмата са необходими, ако клиентът може да анулира резервация?

ГЛАВА 6
МОДЕЛИРАНЕ НА ПРОЦЕСИ И СЪСТОЯНИЯ В UML

Диаграма на дейностите. Диаграма на състоянията.

6.1. Теоретични основи

Диаграмите за поведение в UML се използват за описание на **динамичните аспекти на системата** — как тя реагира на събития и как се променя във времето.

Диаграма	Предназначение
Диаграма на дейностите	Описва потоците от дейности, бизнес процеси, алгоритми и работни потоци (workflow).
Диаграма на състоянията	Описва как обект или система сменя своето състояние в отговор на събития — жизнен цикъл.

6.1.1. Диаграма на дейностите (UML Activity Diagram)

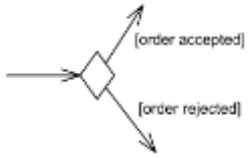
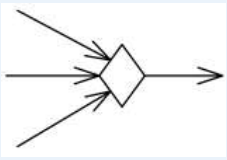
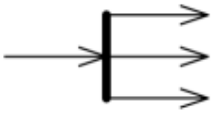
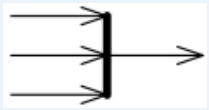
Диаграмата на дейностите (**Activity Diagram**) представя **потоците от дейности** в една система. Тя е подобна на традиционните блок-схеми (flowcharts), но е по-мощна — поддържа паралелни потоци, плавателни ленти (swim lanes) и обектни потоци.

Тя се използва за моделиране на:

- Моделиране на бизнес процеси — за визуализация на стъпките в даден работен процес.
- Описание на алгоритми — за представяне на сложна логика с разклонения и цикли.
- Детайлизация на Use Case — за разгъване на основен и алтернативни потоци.
- Моделиране на паралелна обработка — когато стъпки се изпълняват едновременно.

Основни елементи на диаграмата на дейностите:

Символ	Елемент	Описание
	**Начален възел (Initial Node)**	Запълнен черен кръг. Обозначава единствената начална точка на диаграмата. Всяка Activity Diagram има точно един начален възел.
	**Краен възел (Activity Final Node)**	Черен кръг в окръжност. Обозначава края на ЦЕЛИЯ поток от дейности. При достигане на краен възел всички активни потоци се прекратяват.
	**Краен възел на поток (Flow Final Node)**	Кръг с X. Прекратява само текущия контролен поток, без да засяга останалите. Използва се при грешки или изключения в паралелни потоци.
	**Дейност / Действие (Action / Activity)**	Правоъгълник с заоблени ъгли. Представлява изпълнима операция или стъпка в процеса. Именува се с активен глагол (напр. „Провери наличност“).

Символ	Елемент	Описание
→	**Контролен поток (Control Flow)**	Стрелка между дейности или възли. Показва реда на изпълнение. Може да носи условие (guard condition) в квадратни скоби: [условие].
	**Разклонение (Decision Node)**	Ромб. Представлява точка на вземане на решение с едно входящо и поне две изходящи ребра. Всяко изходящо ребро носи условие. Само едно условие може да е истина едновременно.
	**Сливане (Merge Node)**	Ромб с множество входа и един изход. Обединява алтернативни пътища след Decision Node. Различава се от разклонението по посоката на стрелките.
	**Разделяне (Fork Node)**	Дебела хоризонтална или вертикална линия. Разделя един поток на множество паралелни потоци, изпълнявани едновременно.
	**Обединяване (Join Node)**	Дебела хоризонтална или вертикална линия с множество входа. Синхронизира паралелни потоци — изчаква всички входящи потоци преди да продължи.

Разлика: Decision/Merge vs. Fork/Join
Decision/Merge — алтернативни потоци (само ЕДИН се изпълнява в даден момент).
Fork/Join — паралелни потоци (ВСИЧКИ се изпълняват едновременно).
Правило: всеки Fork трябва да бъде затворен от съответен Join.

Предимства и ограничения на диаграмата на дейностите:

Предимства	Ограничения
Лесно разбираема от нетехнически заинтересовани страни.	Не описва вътрешното поведение на обектите — само потока.
Поддържа паралелни потоци и синхронизация.	При сложни системи може да стане твърде обемна и трудна за четене.
Подходяща за визуализация на бизнес процеси.	Не показва явно кой обект е в дадено вътрешно състояние.

6.1.2. Диаграма на състоянията (UML State Diagram / Statechart)

Диаграмата на състоянията (**State Diagram / Statechart**) описва как един обект или система **сменя своето вътрешно състояние в отговор на събития**. Тя е базирана на концепцията за **крайни автомати (Finite State Machines)** и е въведена от Дейвид Харел (Harel, 1987), след което е стандартизирана в UML.

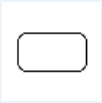
Докато диаграмата на дейностите описва *последователността от стъпки* (flow-oriented), диаграмата на състоянията описва *жизнения цикъл на обект* (state-oriented) —

какви са възможните му вътрешни конфигурации и при какви обстоятелства преминава от едно в друго.

Кога се използва диаграма на състоянията?

- Моделиране на жизнен цикъл на обект (напр. поръчка, резервация, книга).
- Проектиране на потребителски интерфейси с различни режими (напр. логнат/отписан).
- Вградени системи и комуникационни протоколи с ясно дефинирани състояния.
- Описание на event-driven поведение, при което събитията управляват потока.

Основни елементи на диаграмата на състоянията:

Символ	Елемент	Описание
	**Начално псевдосъстояние (Initial Pseudostate)**	Запълнен черен кръг. Не е истинско състояние — показва откъде започва жизненият цикъл. Не може да има входящи преходи.
	**Крайно състояние (Final State)**	Черен кръг в окръжност. Обозначава края на жизнения цикъл на обекта. Може да има повече от едно.
	**Състояние (State)**	Правоъгълник с заоблени ъгли. Описва вътрешна конфигурация на обекта. Може да съдържа entry/exit/do действия.
→	**Преход (Transition)**	Стрелка между две състояния. Синтаксис: събитие [условие] / действие. Задейства се при получаване на събитие (ако условието е изпълнено).

Анатомия на прехода (Transition)

Всеки преход може да съдържа три части, записани в следния формат:

Формат на преход
събитие [условие] / действие

Пример 1: borrow() [isAvailable] / setStatus("borrowed")
Пример 2: return() / setStatus("available")
Пример 3: timeout [reservation > 48h] / cancel()

Само събитието е задължително. Условието и действието са опционални.

Компонент	Обозначение	Описание
Събитие (Event)	събитие()	Тригерът, задействащ прехода. Може да е метод, съобщение или сигнал.
Условие (Guard)	[булев израз]	Условие, което трябва да е истина, за да се извърши преходът. Ако е false — преходът не се задейства.
Действие (Action)	/ действие()	Операция, изпълнявана при извършване на прехода. Атомарна и не може да бъде прекъсната.

Предимства и ограничения на диаграмата на състоянията

Предимства	Ограничения
Ясно показва жизнения цикъл на обект и всички му възможни състояния.	Не е подходяща за описание на потоци с много участници (по-добре — Activity Diagram).
Позволява моделиране на event-driven поведение с точни тригери и условия.	При много на брой състояния диаграмата може да стане трудна за поддръжка.
Лесно се валидира — лесно се открива дали всички преходи са покрити.	Не показва кой актьор предизвиква събитията.

Правилото за избор
Използвайте Activity Diagram, когато искате да покажете последователността на стъпки и/или кой изпълнява всяка стъпка (различни актьори).
Използвайте State Diagram, когато искате да покажете вътрешните конфигурации на обект и при какви събития той преминава от едно в друго.

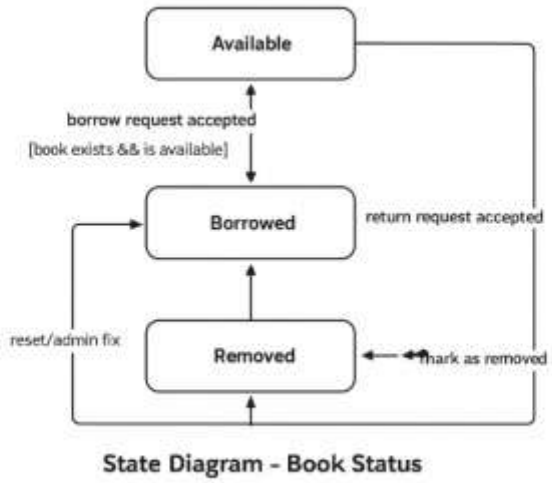
6.2. Практическо приложение

Казус: Система за управление на онлайн библиотека

Описание на казуса
Система за управление на онлайн библиотека, поддържаща операции за заемане и връщане на книги.
Ще моделираме: (1) процеса на заемане като Activity Diagram;
(2) жизнения цикъл на книга като State Diagram.

6.2.1. Диаграма на състоянията (State Diagram) — Жизнен цикъл на книга

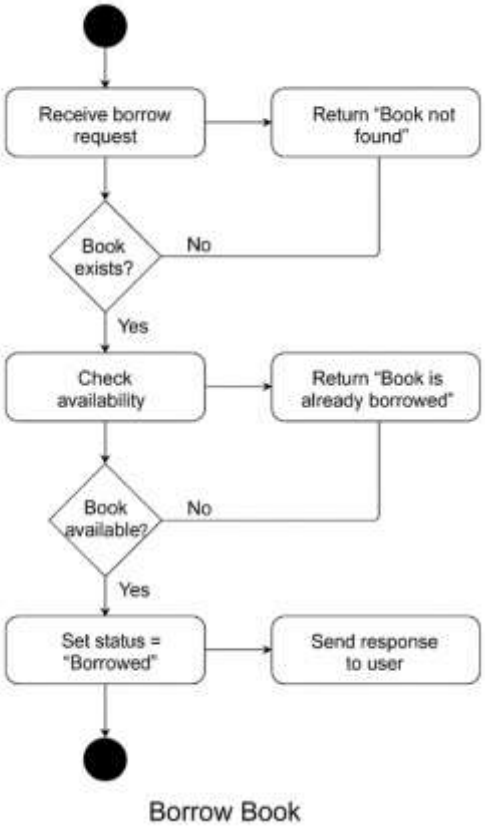
Диаграмата на състоянията описва жизнения цикъл на обект Книга (Book) в библиотечната система. Книгата може да бъде в три основни състояния, между които преминава при определени събития (фиг. 6.1).



Фиг. 6.1 Диаграма на състоянията на приложение за онлайн библиотека.

6.2.2. Диаграмата на дейностите (Activity Diagram) — Процес на заемане на книга

Диаграмата представя последователността от стъпки при процеса на заемане на книга. След получаване на заявката, системата проверява дали книгата съществува и дали е налична. В зависимост от резултата, системата или маркира книгата като „заета“, или връща съобщение за грешка.



Фиг. 5.10 Диаграмата на дейностите на приложение за онлайн библиотека.

6.3. Въпроси и задачи за упражнения

За всеки казус студентите трябва да изпълнят следните изисквания:

- **Activity Diagram —** начертайте диаграма на дейностите за основния процес.
- **State Diagram —** начертайте диаграма на жизнения цикъл на ключовия обект (паркомясто, поръчка, резервация).
- **Анализ на консистентността —** посочете как Activity Diagram и State Diagram се допълват за конкретния казус.

6.3.1. Казус: Разработване на система за интелигентно паркиране

Условие за изпълнение
Система за интелигентно паркиране в голям град. Шофьорите могат да намират, резервират и заплащат за паркоместа чрез мобилно приложение. Администраторът управлява зоните.
Сензорите следят заетостта на местата в реално време.

Процес за Activity Diagram (Заемане на паркомясто):

1. Шофьорът отваря мобилното приложение.

2. Системата определя текущата локация.
3. Системата показва свободни паркоместа в района.
4. Шофьорът избира паркомясто и часов диапазон.
5. Системата проверява дали е свободно в избрания период.
6. Ако е свободно — системата блокира го временно (15 мин.).
7. Шофьорът потвърждава и плаща.
8. Системата потвърждава резервацията и показва маршрут.

Обект за State Diagram: ParkingSpot (Паркомясто)
Идентифицирайте минимум **4 състояния** и всички преходи между тях.

6.3.2. Казус: Разработване на онлайн магазин (E-commerce платформа)

> **Условие за изпълнение**
> Онлайн магазин, в който потребителите разглеждат продукти, добавят ги в количка и завършват покупка. Системата обработва плащания и управлява статуса на поръчките. Администраторите управляват наличностите и поръчките.

Процес за Activity Diagram (Завършване на покупка):
1. Потребителят преглежда продукти и добавя в количка.
2. Потребителят натиска "Поръчай".
3. Системата проверява наличностите на всички продукти в количката.
4. Ако всичко е налично — системата изчислява общата сума.
5. Потребителят избира начин на плащане и потвърждава.
6. Системата изпраща заявка към платежния шлюз.
7. Платежният шлюз потвърждава или отхвърля плащането.
8. При успех — системата создава поръчка и уведомява потребителя.

Обект за State Diagram: Order (Поръчка)
Идентифицирайте минимум **5 състояния** за жизнения цикъл на поръчка.

6.3.3. Казус: Разработване на система за хотелски резервации

> **Условие за изпълнение**
> Система за хотелски резервации с онлайн търсене, резервиране и плащане. Клиентите могат да анулират резервации. Администраторите управляват стаите. Системата интегрира се с Channel Manager (Booking.com).

Процес за Activity Diagram (Процес на резервация):
1. Клиентът въвежда дати и брой гости.
2. Системата търси свободни стаи за тези дати.
3. Клиентът избира стая от списъка.
4. Клиентът попълва лични данни.
5. Системата проверява дали стаята е все още свободна (race condition).
6. Системата създава временна резервация (блокира стаята за 10 мин.).
7. Клиентът извършва плащане.
8. Платежният шлюз потвърждава. Системата финализира резервацията.
9. Системата уведомява Channel Manager и изпраща потвърждение на клиента.

Обект за State Diagram: Reservation (Резервация)
Идентифицирайте минимум **5 състояния** за жизнения цикъл на резервация. **Специален акцент:** включете сценария с двойна резервация като алтернативен поток.

ГЛАВА 7
ВАЛИДАЦИЯ НА ИЗИСКВАНИЯТА. КОНФЛИКТ МЕЖДУ ИЗИСКВАНИЯТА.

7.1. Теоретични основи

Инженерството на изискванията не приключва с тяхното събиране и документиране. Два от най-критичните процеси, следващи документацията, са **валидацията** (осигуряване, че изискванията са верни и пълни) и **управлението на конфликти** (разрешаване на противоречия между изисквания или заинтересовани страни).

7.1.1. Валидация на изисквания (Requirements Validation)

Валидацията на изисквания е процес на **проверка** дали изискванията са **пълни, последователни, верни** и реалистично реализируеми. Тя се различава от верификацията: верификацията проверява дали *системата е изградена правилно*, а валидацията — дали *е изградена правилната система.*

Защо валидацията е критично важна?
Грешките в изискванията се разпространяват нагоре по всички следващи фази на разработката. Колкото по-рано се открие грешка, толкова по-евтино е нейното отстраняване. Проучвания показват, че коригирането на грешка в изискванията след доставката на системата е до 100 пъти по-скъпо от нейното отстраняване по време на фазата на анализ.

Фаза на откритие	Относителна цена на поправка
Анализ на изискванията	1x (базова цена)
Проектиране (Design)	5x
Реализация (Implementation)	10x
Тестване (Testing)	20x
Доставка и производство (Production)	до 100x

Три аспекта на качеството на изискванията
Преди одобрение, всяко изискване трябва да е проверено по три измерения:

Аспект	Какво се проверява	Ключови въпроси
Съдържание (Content)	Коректност, пълнота, консистентност — изискването отразява ли нуждите на заинтересованите страни?	Вярно ли е изискването? Пълно ли е? Необходимо ли е? Верифицируемо ли е?
Документация (Documentation)	Следва ли шаблон, ясно ли е, еднозначно ли е? Правилна ли е терминологията и синтаксисът?	Ясно ли е? Следва ли формата? Еднозначно ли е?

Аспект	Какво се проверява	Ключови въпроси
Съгласие (Agreement)	Одобрено ли е от заинтересованите страни? Разрешени ли са конфликтите преди разработката?	Одобрено ли е? Конфликтите разрешени ли са?

Основни принципи на валидацията:

- **Включете правилните заинтересовани страни** — потребители, бизнес представители, разработчици и тестери трябва да участват в прегледа.
- **Разделете откриването от коригирането** — по време на инспекцията не се правят корекции; те следват след приключване на прегледа.
- **Валидирайте от различни гледни точки** — потребителят, архитектът и тестерът имат различни перспективи, всяка от които може да разкрие различни проблеми.
- **Сменяйте типа документация при нужда** — ако текстово описание е неясно, опитайте с диаграма или прототип.

7.1.2. Техники за валидация на изисквания

Техниките за валидация се разделят на два основни вида: статични (без реална работеща система) и динамични (с реална или симулирана система). Всяка техника е подходяща за различен контекст.

Техника	Описание	Характер	Формалност
Коментиране (Commenting)	Неформален индивидуален преглед. Рецензентът маркира неясни или грешни части директно в документа.	Статичен	Ниска
Walkthrough (Преход)	Леко-формален преглед, при който авторът представя изискванията стъпка по стъпка пред малка група. Фокус — споделено разбиране.	Статичен	Средна
Инспекция (Inspection)	Формален и систематичен процес. Преминава през фази: планиране, обзор, детекция на грешки, събиране. Грешките се документират. Най-ефективен метод.	Статичен	Висока
Perspective-Based Reading (PBR)	Прегледът се извършва от различни перспективи (потребител, архитект, тестер). Всяка перспектива използва специфични въпроси за проверка.	Статичен	Средна
Прототипиране (Prototyping)	Изгражда се прототип (изхвърлящ или еволюционен) за валидация на изисквания. Разкрива скрити	Динамичен	Средна

Техника	Описание	Характер	Формалност
	проблеми, недостъпни за статичен преглед.		
Алфа/Бета тестване (Alpha/Beta)	Ранен достъп на потребители до системата. Алфа — вътрешно; Бета — ограничена публична група. Събират се реални данни.	Динамичен	Средна
Чеклисти (Checklists)	Систематичен списък с въпроси, базиран на качествени критерии и опит от предишни проекти. Кратък, прецизен и комбинируем с другите техники.	Статичен	Ниска–Средна

Детайл: Инспекция (Inspection)

Инспекцията е най-ефективната и формална техника за статична валидация. Тя преминава през строго дефинирани фази:

№	Фаза	Описание
1	**Планиране**	Определяне на обхвата, участниците и критериите за проверка. Разпределяне на ролите.
2	**Обзор (Overview)**	Авторът представя кратко документа. Инспекторите получават материалите за самостоятелен преглед.
3	**Детекция на грешки**	Всеки инспектор самостоятелно търси проблеми. Грешките се записват без дискусия и коригиране.
4	**Събиране (Collection)**	Групова среща. Всеки представя намерените проблеми. Секретарят ги документира. НЕ се коригират — само се регистрират.
5	**Коригиране и следване**	Авторът коригира документираните грешки. Модераторът проверява дали са адресирани коректно.

Роли при инспекцията:

- **Организатор и модератор —** управлява процеса и гарантира обективност.
- **Автор —** написал документа; представя го, но не защитава грешките.
- **Читател —** чете документа на глас по време на срещата.
- **Инспектори —** търсят дефекти от различни перспективи.
- **Секретар —** документира намерените грешки.

Видове прототипи при валидация:

Вид прототип	Характеристика	Кога се използва
Изхвърлящ (Throw-away)	Изгражда се само за валидация. След потвърждение на изискванията се изхвърля и системата се разработва наново.	Ранни фази; при голяма несигурност в изискванията.
Еволюционен (Evolutionary)	Прототипът се развива итеративно и в крайна сметка се превръща в готовата система.	Когато изискванията са достатъчно стабилни; Agile разработка.

Списък за проверка на качество на изискванията

Списъкът за проверка за добро изискване (**Good Requirement Characteristics Checklist**) е инструмент за систематична проверка на качеството на отделно изискване или на целия набор от изисквания. Той е базиран на признатите характеристики на качественото изискване.

Характеристика	Определение	Въпроси за проверка
Недвусмислено (Unambiguous)	Изискването може да се тълкува само по един начин.	• Изразено ли е изискването с ясно дефинирани термини? • Могат ли читатели с различен опит да го интерпретират различно? • Елиминирани ли са всички неясни препратки („например“, „като“, „напр.“)?
Пълно (Complete)	Напълно описва очакваното поведение и набора от функции.	• Включени ли са всички идентифицирани изисквания? • Дефинирани ли са всички входове към системата? • Дефинирани ли са всички системи, с които продуктът трябва да взаимодейства? • Има ли липсваща информация в описанията? • Посочени ли са всички условия, при които изискването се прилага?
Необходимо (Necessary)	Системата не може да посрещне бизнес нуждите без него.	• Свързано ли е с документ от по-високо ниво (бизнес цели)? • Повтаря ли се информацията в документа с изисквания?
Консистентно (Consistent)	Изискванията не си противоречат помежду си, нито с управляващи спецификации.	• Съдържат ли описанията на различни изисквания противоречия? • Има ли противоречия между индивидуалните изисквания и общите системни изисквания?
Приоритизирано (Prioritized)	Изискванията са наредени по важност — Critical / Major / Minor.	• Приоритизирани ли са всички изисквания по степен на важност?
Измеримо (Measurable)	Може да се потвърди, че изискването е изпълнено (верифицируемо).	• Може ли изискването обективно да бъде измерено/тествано?
Разбираемо (Comprehensible)	Правилно описва поведението по разбираем начин за всички участници.	• Могат ли читателите на документа да разберат какво означава изискването? • Дефинирани ли са всички необичайни акроними?
Осъществимо (Feasible)	Реалистично — ползите от жизнения цикъл надвишават разходите.	• Може ли изискването да бъде реализирано с наличната технология? • Може ли да бъде реализирано при другите ограничения на системата? • Рентабилно ли е поддържането на специфицираната система?

Характеристика	Определение	Въпроси за проверка
Проследимо (Traceable)	Има ясни предшественици в по-ранни спецификации — без тълкувания или добавки.	• Уникално маркирано ли е и изразено ли е отделно, за да може да се референцира? • Адекватно ли покрива изискванията от документи от по-високо ниво?
Модифицируемо (Modifiable)	Изискванията са лесни за промяна или актуализация.	• Може ли да се поддържа история на промените за всяко изискване? • Използва ли се независимо номериране (без вградено авто-номериране)?

Как да използваме чеклиста?
За всяко изискване преминете всички въпроси в списъка за проверка.
Отговорете с: ✓ (да), ✕ (не), или ? (неясно).
Изискванията с отговор ✕ или ? трябва да бъдат преразгледани и коригирани.
Списъкът за проверка може да се комбинира с всяка друга техника (инспекция, walkthrough, PBR).

Таблица за резултати от валидацията

Таблица 7.1 — Шаблон за документиране на резултатите от валидация:

ID	Изискване	Техника	Статус	Бележки / Проблем
FR1				
FR2				
FR3				

7.1.3 Конфликт между изисквания

Конфликт между изисквания възниква, когато две или повече изисквания **си противоречат, са несъвместими или отразяват различни мнения** на заинтересованите страни. Неразрешените конфликти застрашават успеха на проекта и водят до забавяния, компромиси в качеството или дори провал.

Кога възниква конфликт?
Противоречие: "всички съобщения да са черни" vs. "грешките да са червени"
Несъвместимост: две изисквания не могат да бъдат реализирани едновременно.
Различни мнения: различни заинтересовани страни имат противоположни очаквания.

Видове конфликти:

№	Вид конфликт	Описание	Пример
1	**Съдържателен (Subject Matter)**	Различия в нуждите, произтичащи от различни закони, стандарти или сфери на дейност.	EU GDPR изисква по-кратък срок за съхранение на данни от националния закон.
2	**Данни (Data Conflict)**	Различни източници на информация или различни интерпретации на едни и същи данни.	Финансовият отдел и ИТ отделът използват различни дефиниции на „активен потребител“.
3	**Интереси (Interest Conflict)**	Противоположни цели на различни заинтересовани страни.	Потребителите искат максимална простота; сигурносттта изисква сложна автентикация.
4	**Ценности (Value Conflict)**	Сблъсък на принципи, убеждения или организационна култура.	Ускоряване на процеса vs. запазване на ръчен контрол на качеството.
5	**Взаимоотношения (Relationship)**	Негативни емоции, недоверие или лични противоречия между участниците.	Двама мениджъри не могат да се споразумеят поради предишен конфликт.
6	**Структурен (Structural Conflict)**	Неравенство на власт, ресурси, зависимости или ограничения на средата.	Отдел А не може да утвърди изисквания без одобрение от Отдел В, но В е зает.

Техники за разрешаване на конфликти

Изборът на техника зависи от вида на конфликта, отношенията между страните и наличното време. Техниките са наредени от предпочитана (добро доброволно съгласие) към последна (принудително решение):

Техника	Кога се използва	Подход	Описание
Споразумение (Agreement)	При конфликти на данни или разбиране	Консенсус	Обсъждаме докато всички страни постигнат доброволно съгласие. Резултатът е устойчив.
Компромис (Compromise)	При конфликти на интереси	Взаимни отстъпки	Всяка страна отстъпва частично от позицията си. Никой не получава всичко, но всеки получава нещо.
Гласуване (Voting)	При прости бинарни избори	Мнозинство	Гласуване при равни гласове. Бързо, но може да остави малцинство недоволно.
Варианти (Variants)	При технически разлики	Паралелни опции	Разработват се няколко варианта на решението. Избора следва по-късно след оценка.

Техника	Кога се използва	Подход	Описание
CAF / PMI анализ	При сложни решения с много критерии	Структурирана оценка	Таблица с **С**лабости/**У**якости/**И**нтерес (CAF) или **П**люсове/**М**инуси/**И**нтерес (PMI). Рационализира избора.
Овъррулване (Overruling)	При безизходица и ограничено време — последна опция	Авторитетно	По-висшестоящ взима еднолично решение. Бързо, но може да наруши ангажираността на страните.

Документиране на решението
Всяко разрешено противоречие трябва да бъде документирано. Документацията: предотвратява повтарянето на същия конфликт; запазва обосновката на решението; подпомага прозрачността и проследимостта.

Таблица 7.2 — Шаблон за документиране на конфликти и решения:

Конфликт ID	Изисквания	Заинтересовани страни	Тип конфликт	Техника	Решение
C-01					
C-02					
C-03					

7.3. Практическо приложение

Казус: Система за хотелски резервации

> **Описание на казуса**
> Ще валидираме изисквания и ще идентифицираме и разрешаваме конфликти между тях в контекста на системата за хотелски резервации, разглеждана в предишните глави.

7.3.1. Приложена валидация с чеклист

По-долу е показан примерен резултат от валидация на три изисквания чрез чеклист.

ID	Изискване	Техника	Статус	Бележки / Проблем
FR3	Системата трябва да предотвратява двойни резервации.	Чеклист (Consistency)	✓ **OK**	Консистентно с FR4 (проверка на наличност). Верифицируемо.
NFR2	Системата трябва да осигурява бързо потвърждение на резервация.	Чеклист (Measurable)	✗ **Проблем**	**Липсва измерим критерий.** Предложение: „системата потвърждава в рамките на 3 секунди“.

ID	Изискване	Техника	Статус	Бележки / Проблем
FR5	Клиентите получават имейл потвърждение след резервация.	Walkthrough	✓ OK	Пълно и измеримо. Въпрос: при неуспешна доставка на имейл?

7.3.2. Приложено разрешаване на конфликт

Примерен конфликт — Хотелска система
FR-A: "Резервациите могат да бъдат анулирани без такса до 24 часа преди настаняване."
— Заинтересована страна: Клиенти (искат гъвкавост)
FR-B: "Всяка анулация, независимо от времето, носи такса от 10% от стойността."
— Заинтересована страна: Хотелски мениджмънт (защита на приходи)

Поле	Стойност
Конфликт ID	C-01
Изисквания	FR-A vs. FR-B
Заинтерес. страни	Клиенти (Primary) vs. Хотелски мениджмънт (Secondary)
Тип конфликт	Интереси (3) — противоположни цели: гъвкавост за клиента vs. финансова сигурност за хотела
Техника	Компромис (Compromise)
Решение	**Компромисно изискване (FR-AB):** „Анулация без такса е допустима само ако е направена **повече от 48 часа** преди настаняване. При анулация в рамките на 48 часа се начислява такса от **10%** от стойността. При без анулиране таксата е **50%**.“

7.4. Въпроси и задачи за упражнения

За всеки казус студентите трябва да изпълнят двете задачи: валидация и управление на конфликти.

Обща инструкция
Задача 1 — Валидация: Изберете 3 изисквания от предишните упражнения. Валидирайте всяко чрез чеклиста.
Задача 2 — Конфликт: Идентифицирайте 2 противоречащи изисквания. Опишете: участниците, типа конфликт, избраната техника и документирайте решението.

7.4.1. Казус: Разработване на система за интелигентно паркиране

Контекст
Системата за интелигентно паркиране от предишните глави. Използвайте изискванията, дефинирани в предишните глави за тази система.

Задача 1. — Валидация на изисквания
Изберете **3 изисквания** (поне 2 функционални и 1 нефункционално). За всяко:

1. Приложете чеклиста за качество — отговорете на всеки въпрос с ✓ / ✗ / ?
2. Посочете техниката за валидация (чеклист, walkthrough, инспекция или PBR).
3. Документирайте статуса и намерените проблеми.

Шаблон за валидация:

ID	Изискване	Техника	Статус	Бележки / Проблем
FR-P1				
FR-P2				
NFR-P1				

Задача 2 — Конфликт между изисквания
Идентифицирайте **2 противоречащи изисквания** в системата за паркиране. Примерни конфликти за вдъхновение:

- **Конфликт А:** „Достъпът до системата е анонимен“ vs. „Всеки потребител трябва да е регистриран за плащане“
- **Конфликт В:** „Максимален брой резервации е неограничен“ vs. „Един потребител може да резервира само 1 място едновременно“

Шаблон за конфликт:

Поле	Стойност
Конфликт ID	
Изисквания в конфликт	
Заинтересовани страни	
Тип конфликт (1–6)	
Избрана техника	
Предложено решение	
Ново/компромисно изискване	

7.4.2. Казус: Разработване на онлайн магазин (E-commerce платформа)

Контекст
Онлайн магазин с управление на продукти, поръчки и плащания.
Заинтересовани страни: клиенти, администратори, платежен шлюз, куриерска фирма.

Задача 1. — Валидация на изисквания
Изберете **3 изисквания** за онлайн магазина. Приложете **различни техники за валидация** (не само чеклист). Включете поне един пример с PBR (perspective-based reading).

Шаблон за валидация:

ID	Изискване	Техника	Статус	Бележки / Проблем
FR-E1				
FR-E2				
NFR-E1				

Задача 2 — Конфликт между изисквания
Идентифицирайте **2 конфликтни изисквания** и приложете **CAF/PMI анализ** за вземане на решение. Примерен конфликт:

- **Конфликт:** „Системата запазва историята на поръчките 5 години“ vs. „Съгласно GDPR данните се изтриват след 3 години“

Шаблон за CAF/PMI:

Вариант / Решение	Плюсове (P)	Минуси (M)	Интересно (I)
Вариант A: 3 години (GDPR)			
Вариант B: 5 години (бизнес нужда)			
Вариант C: Компромис (анонимизация след 3 г.)			

7.4.3. Казус: Разработване на система за хотелски резервации

Контекст
Хотелска система с резервации, плащания и управление на стаи.
Заинтересовани страни: гости, рецепционисти, хотелски мениджъри, Channel Manager.

Задача 1 — Валидация чрез инспекция
Проведете **симулирана инспекция** на **5 изисквания** от хотелската система. Разпределете ролите в групата:

- Модератор — управлява процеса
- Автор — написал изискванията
- Инспектори (2-3) — проверяват от различни перспективи
- Секретар — документира намерените проблеми

Шаблон за инспекция (Казус 7.4.3.1):

ID	Изискване	Перспектива	Статус	Намерен проблем
FR-H1				

ID	Изискване	Перспектива	Статус	Намерен проблем
FR-H2				
FR-H3				
NFR-H1				
NFR-H2				

Задача 2 — Конфликт и разрешаване
Идентифицирайте **2 конфликта** в хотелската система. За всеки: определете типа, изберете техника, документирайте решението. Примерни конфликти:

- **Конфликт 1:** „Гостите могат да правят резервации без регистрация“ vs. „Системата изисква профил за проследяване на плащания“
- **Конфликт 2:** „Стаите се показват само на езика на интерфейса“ vs. „Международните гости искат описания на английски“

Шаблон за два конфликта:

Поле	Конфликт 1	Конфликт 2
Изисквания в конфликт		
Заинтересовани страни		
Тип конфликт (1–6)		
Техника за разрешаване		
Компромисно решение		