

МЕТОДИЧЕСКИЙ ПАКЕТ №1

МДК.01.02 «Поддержка и тестирование программных модулей»

Тестирование как часть процесса верификации программного обеспечения

Специальность 09.02.07 «Информационные системы и программирование»

Объем по рабочей программе: 4 академических часа

2026

Паспорт занятия

Цель занятия — сформировать у студентов инженерное понимание тестирования: зачем оно нужно, как связано с верификацией и валидацией, какие цели преследует и какие артефакты создаются в ходе тестовой деятельности.

Параметр	Содержание
Тема по рабочей программе	Тестирование как часть процесса верификации программного обеспечения
Объём	4 академических часа (две пары по 90 минут)
Формат	Теоретическое объяснение + разбор примеров + практический фрагмент + контрольные вопросы
Профессиональные компетенции	ПК 1.3 — выполнять отладку программных модулей; ПК 1.4 — выполнять тестирование программных модулей.

Основа пакета. Материал рассчитан как самостоятельный пакет: его можно использовать независимо от второго занятия. Терминология согласована с рабочей программой МДК.01.02 и актуальным ISTQB CTFL v4.0.1.

ЗАНЯТИЕ 1. Тестирование как часть процесса верификации ПО

Главная задача занятия — убрать бытовое представление о тестировании как о случайном запуске программы и сформировать инженерную картину: есть требования, риски, тестовые объекты, ожидаемые результаты, фактические результаты и решение о качестве продукта.

Пара	Время	Содержание
1-я пара	0–15 мин	Проблемная постановка: почему «у меня работает» не означает, что программа готова.
	15–40 мин	Что такое тестирование. Цели тестирования. Объект тестирования.
	40–65 мин	Верификация и валидация. Тестирование, QA и отладка.
	65–90 мин	Мини-кейс и первичная проверка понимания.
2-я пара	0–25 мин	Статическое и динамическое тестирование. Тестирование в ЖЦ ПО.
	25–50 мин	Основные активности тестирования и тестовые артефакты.
	50–75 мин	Разбор примера программного модуля и составление тестов.
	75–90 мин	Контрольные вопросы и закрепление.

1.1. С чего начать разговор о тестировании

Начните с простой фразы: «Программа запустилась и не упала. Можно ли считать её исправной?»

Обычно часть студентов ответит «да». Это удобная точка входа. Запуск подтверждает только очень маленькую часть свойств программы. Даже если приложение не завершилось аварийно, оно может:

- выдавать неправильный результат;
- правильно работать только на привычных данных, но ломаться на границах диапазона;
- терять данные или сохранять их не туда;
- нарушать требования безопасности;

- работать слишком медленно или потреблять слишком много памяти;
- вести себя правильно на компьютере разработчика, но неправильно в другой конфигурации;
- не соответствовать тому, что реально нужно пользователю.

Ключевой тезис. Тестирование нужно не для того, чтобы «доказать, что программа идеальна», а для получения информации о её качестве и рисках. Результат тестирования помогает принимать решение: можно ли выпускать продукт, что нужно исправить и что остаётся рискованным.

1.2. Что такое тестирование

Рабочее определение для студентов. Тестирование программного обеспечения — это организованный набор действий, с помощью которых проверяют программный продукт и связанные с ним материалы, обнаруживают дефекты и оценивают качество.

Важно подчеркнуть: тестирование шире, чем запуск тестов. Оно включает анализ требований, определение того, что и зачем проверяется, разработку тестовых случаев, подготовку данных, выполнение проверок, фиксацию результатов и подведение итогов.

Типичные цели тестирования

- Оценить требования, дизайн, код и другие рабочие продукты.
- Найти дефекты и вызвать наблюдаемые неправильные результаты, чтобы проблеме можно было исследовать.
- Проверить выполнение заявленных требований.
- Проверить достаточность охвата выбранных частей системы.
- Снизить риск выпуска продукта с неприемлемым качеством.
- Дать команде и заказчику факты для принятия решений.
- Повысить уверенность в качестве продукта.
- Проверить соответствие юридическим, договорным или отраслевым требованиям, если они есть.

Вопрос аудиторией. Пусть студенты сформулируют разницу между «найти все ошибки» и «снизить риск». Полное доказательство отсутствия дефектов в реальном ПО практически недостижимо, поэтому задача тестирования — системно искать проблемы и уменьшать неопределённость.

1.3. Что именно можно тестировать

Тестовый объект — это то, что в данный момент подвергается проверке. Им может быть не только готовая программа.

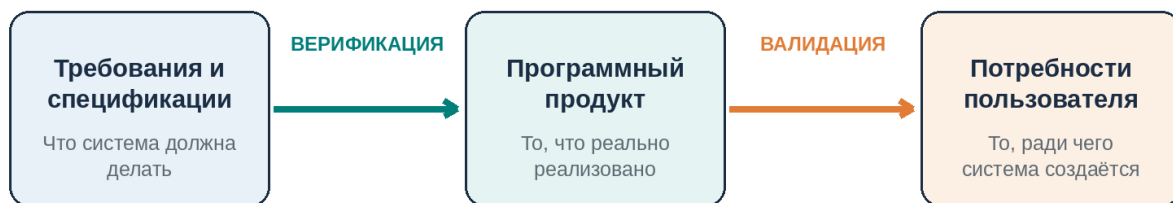
- Отдельная функция или класс.
- Программный модуль.
- API или интерфейс между модулями.
- База данных и запросы.

- Пользовательский интерфейс.
- Собранное приложение целиком.
- Требование, макет, спецификация, алгоритм или иной документ — если выполняется статическая проверка.

Пример. Если студент написал функцию расчёта стоимости заказа, объектом тестирования может быть сама функция. Если функция работает правильно отдельно, но после подключения к базе данных получает неверные цены — объектом интеграционной проверки становится взаимодействие компонентов.

1.4. Верификация и валидация

Тестирование в процессе разработки ПО



Тестирование помогает и проверять соответствие требованиям, и оценивать пригодность продукта для реального использования.

Оно снижает неопределённость и риск, но не доказывает абсолютное отсутствие дефектов.

Верификация отвечает на вопрос: «Правильно ли мы реализовали то, что было задано?» Это проверка соответствия требованиям, спецификациям и проектным решениям.

Валидация отвечает на вопрос: «То ли мы вообще сделали и решает ли продукт задачу пользователя?» Продукт может формально соответствовать спецификации, но быть неудобным или бесполезным в реальной работе.

Ситуация	Что проверяется	Ближе к чему
ТЗ требует пароль не короче 8 символов. Система отклоняет пароль из 7 символов.	Соответствие реализации явному требованию.	Верификация
Система формально принимает сложный пароль, но форма не объясняет пользователю требования и люди массово не могут зарегистрироваться.	Пригодность решения для реальной задачи и ожиданий пользователя.	Валидация

API должно возвращать HTTP 404 для отсутствующего объекта и действительно возвращает 404.	Соответствие спецификации интерфейса.	Верификация
Поиск работает по спецификации, но пользователи не могут найти товар по привычному названию.	Соответствие реальным потребностям.	Валидация

1.5. Тестирование, QA и отладка — не одно и то же

Тестирование оценивает продукт и обнаруживает проблемы.

Обеспечение качества (QA) шире и сильнее ориентировано на процессы: как организовать разработку и контроль так, чтобы дефектов возникало меньше и качество было воспроизводимым.

Отладка — деятельность по поиску причины конкретной проблемы в программном коде и её устранению. Тест может обнаружить отказ, но сам по себе не обязан объяснять, какая строка кода виновата.

Действие	Пример	Категория
Проверить, что поле «Возраст» не принимает отрицательное число.	Сравнить фактическое поведение с ожидаемым.	Тестирование
Ввести обязательный code review и чек-лист перед merge.	Снизить вероятность появления дефектов процессными мерами.	QA / обеспечение качества
Поставить breakpoint и увидеть, что age преобразуется в unsigned-тип.	Найти техническую причину обнаруженного неправильного поведения.	Отладка

1.6. Статическое и динамическое тестирование

Динамическое тестирование требует выполнения программы или её части. Мы задаём входные данные, наблюдаем поведение и сравниваем фактический результат с ожидаемым.

Статическое тестирование не требует запуска программы. Проверяются требования, код, схемы, документация. Это могут быть review, инспекция, анализ кода линтером или статическим анализатором.

Пример	Статическое / динамическое	Почему
Коллега читает pull request и замечает, что условие перепутано.	Статическое	Дефект найден без выполнения программы.
Тест запускает функцию с нулём и получает исключение.	Динамическое	Проблема проявилась при исполнении.
IDE сообщает, что переменная никогда не используется.	Статическое	Анализируется структура кода.

Нагрузочный тест создаёт 500 одновременных запросов.	Динамическое	Измеряется реальное поведение работающей системы.
--	--------------	---

Связь с современными процессами. Статические проверки особенно полезны тем, что позволяют находить дефекты раньше, до того как они превратятся в наблюдаемые отказы работающей программы. Это одна из причин идеи shift-left: часть проверок переносится ближе к моменту появления требований и кода.

1.7. Где тестирование находится в жизненном цикле ПО

Ошибочно представлять тестирование как последний этап после программирования.

Проверки должны сопровождать разработку: требования можно рецензировать, код — анализировать статически и тестировать на уровне компонентов, интеграции — проверять при соединении модулей, а готовую систему — оценивать целиком.

Этап разработки	Что можно проверять уже здесь
Требования	Однозначность, непротиворечивость, полноту, тестируемость, критерии приёмки.
Проектирование	Архитектурные риски, интерфейсы между компонентами, сценарии отказов.
Кодирование	Unit-тесты, статический анализ, code review, локальные проверки.
Интеграция	Контракты API, обмен данными, обработку ошибок между компонентами.
Системная сборка	Функциональные и нефункциональные свойства системы целиком.
Поддержка	Регрессионные проверки после исправлений, обновлений и изменений окружения.

В Agile и DevOps тестирование ещё теснее связано с разработкой. Автоматические проверки могут выполняться в CI после каждого изменения. Но автоматизация не отменяет анализа: сначала нужно решить, что именно проверять и какой риск закрывает тест.

1.8. Основные активности тестирования

Конкретный процесс зависит от проекта, но полезно дать студентам общую последовательность действий:

1. Планирование: определить цель, объём, риски, ресурсы, сроки и критерии завершения.
2. Мониторинг и управление: следить, что происходит с тестированием, и корректировать план.
3. Анализ тестовой основы: изучить требования и понять, что должно быть проверено.
4. Проектирование тестов: определить тестовые условия, входные данные и ожидаемые результаты.

5. Реализация: подготовить тест-кейсы, данные, окружение, скрипты или автоматизацию.
6. Выполнение: запустить проверки, сравнить ожидаемый и фактический результаты, зафиксировать отклонения.
7. Завершение: собрать итоги, оценить оставшиеся риски, сохранить тестовые материалы и выводы.

Методический акцент. Не обязательно заставлять студентов запоминать названия всех активностей в первый день. Важнее, чтобы они увидели: тестирование — это процесс с подготовкой, критериями и результатом, а не хаотичное нажатие кнопок.

1.9. Что такое тестовый случай

Тестовый случай (test case) — конкретная проверка с определёнными условиями, входными данными, действиями и ожидаемым результатом. Чем точнее ожидаемый результат, тем меньше спорных интерпретаций после выполнения теста.

ID	Условие / вход	Действие	Ожидаемый результат
TC-01	price = 1000, percent = 20	Вызвать CalculateDiscount	800
TC-02	price = 1000, percent = 0	Вызвать CalculateDiscount	1000
TC-03	price = 1000, percent = 100	Вызвать CalculateDiscount	0
TC-04	percent = -1	Вызвать функцию	Ошибка валидации / отклонение входа
TC-05	percent = 101	Вызвать функцию	Ошибка валидации / отклонение входа

1.10. Мини-кейс для аудитории: «скидка работает»

Покажите студентам функцию:

```
static decimal CalculateDiscount(decimal price, int percent)
{
    return price - percent;
}
```

Спросите: «Код компилируется? Да. Программа может выполняться? Да. Значит ли это, что функция правильная? Нет». Если цена 1000, а скидка 20%, ожидаем 800, но получаем 980.

Правильная идея расчёта:

```
return price * (1 - percent / 100m);
```

Дальше предложите студентам назвать тесты, которые они бы добавили. Обычно появляются 0%, 100%, отрицательные проценты, больше 100%, цена 0, дробные цены. Так естественно подводим их к мысли о классах данных и границах — эти техники будут изучаться дальше.

1.11. Практический фрагмент на 20–25 минут

Задание. Для функции расчёта доставки составить не менее 8 проверок. Код запускать не обязательно: сначала студенты должны спроектировать тесты на бумаге или в таблице.

```
static decimal DeliveryCost(decimal orderTotal, int distanceKm)
{
    if (orderTotal > 5000) return 0;
    if (distanceKm <= 5) return 200;
    return 200 + (distanceKm - 5) * 30;
}
```

Минимальные направления, которые должны появиться:

- заказ ровно 5000 и чуть больше 5000;
- расстояние 0, 5 и 6 км;
- отрицательное расстояние;
- нулевая сумма заказа;
- очень большое расстояние;
- вопрос: является ли бесплатная доставка при $\text{orderTotal} > 5000$ корректным требованием или должна быть ≥ 5000 ?

Методический смысл. Последний пункт специально не имеет «правильного» ответа без требования. Это важный учебный момент: тестировщик не должен выдумывать спецификацию. Если ожидаемое поведение не определено, нужно уточнение.

1.12. Семь полезных принципов, которые можно дать без перегруза теорией

1. Тестирование показывает наличие дефектов, но не доказывает их полное отсутствие.
2. Исчерпывающе проверить все входы и состояния почти всегда невозможно, поэтому тесты выбирают осмысленно.
3. Чем раньше найден дефект, тем дешевле обычно его исправить.
4. Дефекты часто концентрируются в ограниченном числе сложных или часто меняющихся компонентов.
5. Один и тот же набор тестов со временем перестаёт находить новые проблемы; проверки нужно пересматривать.
6. Подход к тестированию зависит от контекста: банковская система, игра и учебный сайт имеют разные риски.
7. Даже система без найденных дефектов бесполезна, если она не решает задачу пользователя.

1.13. Контрольные вопросы по первому занятию

Почему запуск приложения без падения не является достаточным тестом?

Ответ: Потому что отсутствие аварийного завершения подтверждает только один аспект поведения; могут быть неверные результаты, граничные дефекты, проблемы производительности, безопасности и несоответствие требованиям.

Чем тестирование отличается от отладки?

Ответ: Тестирование обнаруживает и описывает проблемы и оценивает качество; отладка ищет техническую причину дефекта в коде и устраняет её.

Чем верификация отличается от валидации?

Ответ: Верификация проверяет соответствие заданным требованиям; валидация — пригодность решения для реальных потребностей пользователей и заинтересованных сторон.

Можно ли тестировать документ, не запуская программу?

Ответ: Да. Review требований, спецификаций и кода относится к статическим проверкам.

Почему ожидаемый результат важен в тест-кейсе?

Ответ: Без него невозможно объективно определить, прошла проверка или нет; фактический результат не с чем сравнить.

Почему невозможно просто «проверить всё»?

Ответ: Количество комбинаций входов, состояний и окружений обычно слишком велико; приходится выбирать проверки по рискам и техникам.

Что делать, если требование неоднозначно?

Ответ: Зафиксировать неоднозначность и уточнить ожидаемое поведение у владельца требования/заказчика, а не подменять требование догадкой.

1.14. Итог первого занятия

Что должно остаться в голове. Студент должен уйти с занятия с четырьмя опорными идеями: тестирование — процесс; тестирование оценивает качество и риски; верификация и валидация не одно и то же; тестирование и отладка — разные активности. Если эти четыре идеи усвоены, дальнейшие темы про уровни, методы и виды тестирования будут ложиться на понятную основу.

Источники и нормативная основа

1. Рабочая программа профессионального модуля ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем», АНО ПОО ММКЦТ «Академия ТОП», 2023. МДК.01.02, тема 55.
2. ISTQB. Certified Tester Foundation Level Syllabus v4.0.1, 15.09.2024. Используются определения тестирования, целей тестирования, верификации, валидации, статического и динамического тестирования.
3. ISTQB. Certified Tester Foundation Level (CTFL) v4.0 — официальные материалы International Software Testing Qualifications Board.

Материал адаптирован для занятия СПО и не является официальным учебным курсом ISTQB.