ВЕСТНИК НАУКИ
И ТВОРЧЕСТВА

КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ КРОСС-УРОВНЕВОГО ТЕСТИРОВАНИЯ МНОГОУРОВНЕВЫХ ПРИЛОЖЕНИЙ: СКВОЗНОЕ ПОКРЫТИЕ ОТ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДО ВНУТРЕННЕЙ ЛОГИКИ МИКРОСЕРВИСОВ

Иванова Светлана Валерьевна,
ведущий специалист по тестированию,
ООО М Тех. Тестирование программного обеспечения, г. Казань

E-mail: ivsvetlana116@gmail.com

Аннотация. В статье рассматривается концептуальная модель кросс-уровневого тестирования многоуровневых приложений, ориентированная на сквозную проверку пользовательских сценариев от интерфейса до внутренней логики микросервисов. Обоснована необходимость перехода от раздельного тестирования отдельных уровней системы к связанному подходу, учитывающему пользовательские действия, API-запросы, бизнес-логику, микросервисные взаимодействия, данные, логи, метрики и трассировку. Показано, что традиционный подход к тестированию может приводить к фрагментарному покрытию, дублированию проверок и затруднению локализации дефектов. Предложенная модель позволяет выстраивать тестирование вокруг полного пути выполнения функции, выявлять проблемные участки архитектуры и повышать управляемость тестовой стратегии. Особое внимание уделено методике применения модели, метрикам оценки эффективности и ограничениям ее использования в распределенных микросервисных системах.

Ключевые слова: кросс-уровневое тестирование, многоуровневые приложения, микросервисная архитектура, API, бизнес-логика, тестовое покрытие, автоматизированное тестирование, CI/CD, трассировка, метрики качества.

Актуальность исследования

Актуальность исследования обусловлена тем, что современные программные приложения все чаще представляют собой сложные многоуровневые системы, включающие пользовательский интерфейс, API, бизнес-логику, базы данных и микросервисы. В таких системах качество работы зависит не только от корректности отдельных компонентов, но и от согласованности их взаимодействия. Ошибка может возникать на любом этапе

прохождения пользовательского запроса – от действия в интерфейсе до выполнения внутренней логики сервисов.

Особую значимость проблема приобретает в условиях распространения микросервисной архитектуры. Микросервисы позволяют разделять приложение на самостоятельные функциональные части, однако одновременно усложняют тестирование, поскольку итоговое поведение системы формируется через взаимодействие множества сервисов, API, баз данных и внешних компонентов. Поэтому проверка только отдельных уровней приложения не всегда позволяет выявить дефекты, возникающие на стыке компонентов [8, с. 59].

Традиционные виды тестирования решают важные задачи, но при их раздельном применении может возникать фрагментарность тестового покрытия. В результате связь между пользовательскими сценариями, бизнес-логикой и внутренними микросервисными операциями остается недостаточно прозрачной.

В связи с этим актуальной является разработка концептуальной модели кросс-уровневого тестирования многоуровневых приложений. Такая модель позволяет рассматривать тестирование как сквозной процесс: от пользовательского интерфейса до внутренней логики микросервисов. Ее применение может способствовать более полному покрытию критических сценариев, повышению надежности программного продукта и более точному выявлению причин дефектов в распределенных системах.

Цель исследования

Целью данного исследования является разработка и обоснование концептуальной модели кросс-уровневого тестирования многоуровневых приложений, обеспечивающей сквозное покрытие пользовательских сценариев от пользовательского интерфейса до внутренней логики микросервисов.

Материалы и методы исследования

Материалами исследования послужили теоретические положения в области тестирования программного обеспечения, подходы к построению многоуровневых и микросервисных приложений, а также практики API-тестирования, интеграционного тестирования, контрактного тестирования, CI/CD и наблюдаемости программных систем.

В работе использовались методы анализа, систематизации, сравнения, обобщения, структурного описания и концептуального моделирования.

Результаты исследования

Многоуровневое приложение представляет собой программную систему, в которой отдельные функции распределены между несколькими слоями: пользовательским интерфейсом, API, бизнес-логикой, хранилищем данных и сервисами, обеспечивающими выполнение отдельных операций. Такое разделение позволяет разграничивать ответственность компонентов, но одновременно требует проверки не только каждого уровня отдельно, но и их совместной работы.

В микросервисной архитектуре приложение состоит из небольших автономных сервисов, каждый из которых реализует отдельную бизнес-

функцию. Поэтому тестирование такой системы должно учитывать границы сервисов, сетевое взаимодействие, передачу данных, обработку ошибок и согласованность результатов между компонентами. В отличие от монолитного приложения, где большая часть логики сосредоточена внутри единой системы, микросервисное приложение зависит от корректной работы цепочки взаимосвязанных сервисов [9, с. 51].

В практике тестирования применяются разные уровни проверок: компонентное тестирование, интеграционное тестирование, системное тестирование и приемочное тестирование (таблица 1).

Таблица 1

Основные уровни тестирования многоуровневого приложения

Уровень тестирования	Что проверяется	Значение для многоуровневого приложения
Компонентное тестирование	Отдельные функции, классы, модули	Позволяет быстро выявлять ошибки в изолированной логике
Интеграционное тестирование	Взаимодействие компонентов и сервисов	Проверяет корректность обмена данными между частями системы
Системное тестирование	Работа приложения как единого целого	Показывает, выполняет ли система заявленные функции
Приемочное тестирование	Соответствие ожиданиям пользователя или заказчика	Подтверждает пригодность системы для практического использования

Источник: разработка автора.

Традиционный подход к тестированию часто строится по отдельным уровням: отдельно проверяется интерфейс, отдельно API, отдельно сервисы и база данных. Такой подход удобен для организации работ, но он не всегда показывает, как ошибка на одном уровне влияет на поведение всей системы. Например, интерфейс может корректно отображать форму, API может принимать запрос, но внутренняя логика сервиса может неверно обработать данные или передать их в другой компонент [4, с. 153].

Основная проблема заключается в разрыве между пользовательским действием и внутренним результатом его выполнения. Если тест проверяет только внешний ответ системы, он может не обнаружить ошибку во внутренней цепочке микросервисов. Если же тестирование проводится только на уровне отдельных сервисов, оно может не показать, как эта ошибка проявится для пользователя. Поэтому при традиционном подходе часть дефектов выявляется поздно – уже на этапе комплексного тестирования или эксплуатации [3, с. 257].

Еще одна проблема связана с дублированием проверок. Один и тот же сценарий может частично повторяться в UI-тестах, API-тестах и интеграционных тестах, но при этом не давать полного понимания, какой

именно участок системы покрыт тестами. Это увеличивает объем тестовой базы, усложняет сопровождение автотестов и снижает точность анализа результатов.

Для микросервисных систем также важна проблема наблюдаемости. При сбое необходимо понимать, через какие сервисы прошел запрос, где возникла задержка, какой компонент вернул ошибку и какие данные были переданы дальше [7, с. 249]. Без логирования, мониторинга и распределенной трассировки локализация дефекта становится сложной, особенно если один пользовательский сценарий затрагивает несколько сервисов.

Кросс-уровневое тестирование предполагает проверку приложения не только по отдельным слоям, но и по сквозным связям между ними. Его задача состоит в том, чтобы проследить выполнение функции от пользовательского действия до внутренней обработки данных в сервисах. Такой подход особенно важен для многоуровневых и микросервисных систем, где итоговый результат формируется через несколько взаимосвязанных компонентов [6, с. 299].

Логика кросс-уровневого тестирования представлена на рисунке 1.

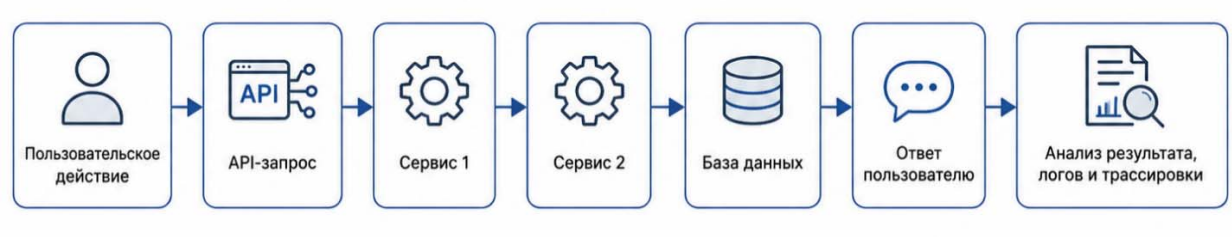

Рис. 1. Логика кросс-уровневого тестирования (разработка автора)

Концептуальная модель кросс-уровневого тестирования строится на связи между пользовательским сценарием, техническим маршрутом запроса и результатом выполнения бизнес-операции. В центре модели находится не отдельный тест, а полный путь функции через уровни приложения: интерфейс, API, сервисы, хранилище данных и средства наблюдаемости. Такой подход позволяет проверять не только факт выполнения действия, но и корректность всей цепочки обработки.

В модели выделяются три основные группы элементов: входные данные, уровни проверки и средства контроля результата. К входным данным относятся требования, пользовательские сценарии, API-спецификации и сервисные контракты. Уровни проверки включают интерфейс, API, бизнес-логику, микросервисы и данные. Средства контроля результата представлены логами, метриками, трассировками и отчетами автотестов.

Применение концептуальной модели начинается с анализа архитектуры приложения [5, с. 8]. На этом этапе необходимо определить основные уровни системы, точки входа, сервисы, базы данных и внешние интеграции. После этого выбираются критические пользовательские сценарии, которые имеют наибольшее значение для работы приложения: регистрация, авторизация,

оформление заказа, проведение платежа, изменение данных, получение отчета и другие ключевые операции.

Следующий этап связан с построением маршрута сценария. Для каждого пользовательского действия определяется, какой API-запрос выполняется, какие сервисы участвуют в обработке, какие данные создаются или изменяются, какой результат должен получить пользователь. Затем для каждого уровня подбираются соответствующие тесты: UI-тесты, API-тесты, интеграционные проверки, контрактные тесты и проверки данных.

После разработки тестов выполняется настройка наблюдаемости. Для кросс-уровневого тестирования важно, чтобы один запрос можно было проследить через разные компоненты системы. Поэтому в модель включаются логи, метрики и распределенная трассировка. Это позволяет не только обнаружить дефект, но и быстрее определить его источник.

Методика применения модели представлена в таблице 2.

Таблица 2

Методика применения модели

Этап	Содержание работ	Результат
1. Анализ архитектуры	Определяются уровни, сервисы и связи между ними	Карта компонентов приложения
2. Выбор сценариев	Отбираются ключевые пользовательские операции	Перечень критических сценариев
3. Описание маршрута	Фиксируется путь запроса через API, сервисы и данные	Сквозная цепочка выполнения функции
4. Разработка тестов	Создаются проверки для каждого уровня	Набор связанных тестов
5. Настройка наблюдаемости	Подключаются логи, метрики и трассировка	Возможность анализа причины сбоя
6. Интеграция в CI/CD	Тесты включаются в процесс сборки и поставки	Регулярная автоматическая проверка
7. Анализ результатов	Оцениваются дефекты, покрытие и стабильность тестов	Обновление тестовой стратегии

Источник: разработка автора.

Эффективность кросс-уровневого тестирования оценивается по тому, насколько полно тесты покрывают критические сценарии, насколько быстро выявляются дефекты и насколько точно определяется их причина. В отличие от оценки только количества тестов, данный подход ориентирован на качество покрытия: важно понимать, какие пользовательские действия, API-запросы, сервисы, бизнес-правила и операции с данными реально проверяются [2, с. 58].

Основными критериями эффективности являются полнота покрытия пользовательских сценариев, покрытие API и сервисных контрактов, выявление дефектов на границах компонентов, стабильность автотестов, скорость локализации ошибки и снижение дублирования проверок (таблица 3). Для

микросервисных систем также важны показатели наблюдаемости: наличие трассировки запроса, доступность логов и возможность связать ошибку с конкретным сервисом.

Таблица 3

Метрики оценки кросс-уровневого тестирования

Метрика	Что показывает	Практическое значение
Покрытие пользовательских сценариев	Какая часть ключевых сценариев проверяется тестами	Показывает связь тестов с реальными функциями системы
Покрытие API	Какие API-методы проверены	Позволяет оценить надежность технических точек входа
Покрытие сервисных контрактов	Проверены ли правила взаимодействия сервисов	Снижает риск ошибок при изменении сервисов
Покрытие бизнес-правил	Проверены ли основные условия выполнения операций	Помогает выявлять ошибки внутренней логики
Время локализации дефекта	Как быстро определяется источник ошибки	Ускоряет исправление дефектов
Стабильность автотестов	Как часто тесты дают воспроизводимый результат	Повышает доверие к автоматизированной проверке
Доля дефектов на стыке уровней	Сколько ошибок возникает при взаимодействии компонентов	Показывает проблемные зоны архитектуры
Дублирование тестов	Повторяются ли одни и те же проверки на разных уровнях	Помогает оптимизировать тестовую базу

Источник: разработка автора.

Дополнительно могут использоваться метрики DORA, применяемые для оценки эффективности поставки программного обеспечения. Они не заменяют тестовые метрики, но помогают оценить влияние тестирования на скорость и устойчивость разработки.

Практическая значимость модели заключается в том, что она позволяет связать тестирование с реальной архитектурой многоуровневого приложения (рисунок 2). Проверка строится не по отдельным тестам, а по полному пути выполнения функции: от пользовательского действия до обработки запроса сервисами и получения результата. Это помогает точнее определять, какие уровни системы покрыты тестами, где отсутствуют проверки и на каком этапе возникает ошибка.

Модель может применяться при разработке микросервисных приложений и в CI/CD-процессах, где важно быстро выявлять дефекты до выпуска изменений. Ее использование делает тестовую стратегию более управляемой, снижает дублирование проверок и помогает связывать результаты тестов с логами, метриками и трассировкой [10].

Ограничения модели связаны с необходимостью иметь актуальное описание архитектуры приложения: сервисов, API, потоков данных и зависимостей между компонентами. Без такой информации сложно построить

точную схему кросс-уровневого тестирования и корректно оценить полноту покрытия [1, с. 9].

Рис. 2 Практическое значение кросс-уровневой модели (разработка автора)

Дополнительная сложность возникает в микросервисных системах, где запрос может проходить через несколько сервисов, очереди сообщений, базы данных и внешние системы. Не всегда возможно полностью воспроизвести производственную среду, поэтому часть ошибок может быть связана с инфраструктурой, задержками сети или недоступностью внешних компонентов.

Дальнейшее развитие модели может быть связано с уточнением матрицы тестового покрытия, расширением контрактного тестирования, использованием распределенной трассировки и автоматическим анализом логов для выявления повторяющихся дефектов.

Выводы

Таким образом, для современных многоуровневых и микросервисных приложений недостаточно проверять отдельные компоненты изолированно. Более эффективным является кросс-уровневый подход, при котором тестирование связывает пользовательский сценарий, API-запросы, бизнес-логику, микросервисные взаимодействия и данные в единую цепочку проверки. Предложенная концептуальная модель позволяет повысить полноту тестового покрытия, снизить дублирование проверок, улучшить локализацию дефектов и сделать тестовую стратегию более управляемой. При этом эффективность

модели зависит от качества архитектурной документации, уровня автоматизации, наличия логов, метрик и распределенной трассировки.

Литература:

1. Алексеева Е.С. Использование тестовых дублеров в тестировании IT-продукта, построенного на микросервисной архитектуре // Прикладная математика и информатика: современные исследования в области естественных и технических наук: Материалы VI Международной научно-практической конференции (школы-семинара) молодых ученых. – 2020. – С. 7-11.

2. Долженко А.И., Ермолов И.А., Полиев А.Д. Мониторинг программного обеспечения, основанного на микросервисной архитектуре // Информатизация в цифровой экономике. – 2021. – Т. 2, № 2. – С. 55-62. – DOI 10.18334/ide.2.2.113382.

3. Караханова А.А. Анализ микросервисной архитектуры, монолитных приложений, архитектуры SOA // Синергия Наук. – 2020. – № 46. – С. 255-262.

4. Кичук А.П. Взаимодействие микросервисов через API на примере ERP системы // Информационные системы и технологии: теория и практика: научно-техническая конференция Института леса и природопользования СПбГЛТУ. – 2022. – С. 151-156.

5. Овчинников М.А., Гусев К.В. Модели жизненного цикла и непрерывная интеграция // Университетская наука: актуальные вопросы, достижения и инновации: сборник статей II Международной научно-практической конференции. – 2021. – С. 7-9.

6. Потапенко А.С. Сравнение монолитной и микросервисной архитектур // Научные горизонты. – 2020. – № 5(33). – С. 297-303.

7. Савченко Д.И., Радченко Г.И. Методология тестирования микросервисных облачных приложений // Суперкомпьютерные дни в России: Труды международной конференции. – 2015. – С. 245-256.

8. Сафонова Ю.А., Лемешкин А.В., Кравцов А.С. Функциональное тестирование программного продукта для онлайн сервиса заказа 3D печати изделий // Наука, общество, образование в условиях цифровизации и глобальных изменений: сборник статей II Международной научно-практической конференции. – 2022. – С. 58-61.

9. Черниченко А.А. Исследование микросервисной архитектуры при разработке приложений // Мобильный бизнес: перспективы развития и реализации систем радиосвязи в России и за рубежом: сборник материалов (тезисов) 44-й международной конференции РАЕН, Сейшельские острова. – 2019. – С. 50-53.

10. Шепель А. Особенности тестирования в проектах с микросервисной архитектурой // Актуальные исследования. – 2023. – № 47(177). URL: https://apni.ru/article/7529-osobennosti-testirovaniya-v-proektah-s-mikroservisnoj-arhitekturoj.