Модуль 2. Виды тестирования

Говоря о различных классификациях тестирования, стоит признать, что «всё очень сложно». Поэтому, те подходы к классификациям, которые мы будем использовать, не единственные допустимые. В альтернативной литературе или курсах могут встречаться иные способы разделения видов тестирования.

В данном модуле помимо чтения конспекта мы рекомендуем вам также обратиться к глоссарию, в котором представлены более краткие и официальные определения всех описанных видов.

Первый способ классификации, который мы рассмотрим, это классификация **по объекту тестирования**. Согласно такому способу тестирование может быть:

- **Функциональное** – проверяется функциональность ПО – способность решать возложенные на него задачи.
- **Производительности** – насколько хорошо ПО справляется со своими задачами: скорость, количество запросов в единицу времени и иные количественные характеристики.
- **Конфигурационное** – рассматривается то, насколько ПО корректно работает в различных конфигурациях. Сюда относят то, на каком оборудовании и операционных системах работает ПО, насколько хорошо оно конфигурируется, настраивается и исполняет свои задачи с различными настройками.
- **UI (интерфейса)** – тестируется исключительно интерфейс, а не функциональность самого продукта. Проверяется расположены ли кнопки на своих местах, выполняют ли они свои задачи, правильно ли работает анимация. Наверняка, вы хотя бы раз сталкивались с программами, интерфейс которых не проходил эту фазу.
- **Security/Compliance** – проверка соответствия различным политикам от регуляторов и организаций, которые накладывают те или иные ограничения на производимый софт. Это могут быть в частности стандарты безопасности или документы тех или иных государственных структур.

Теперь посмотрим на то, какие виды тестирования могут возникать на пересечении или быть подвидами ранее рассмотренных нами типов.

Тестирование производительности может быть:

- **Нагрузочное** – проверяется работоспособность компонента или системы в целом под определенной постоянной нагрузкой.
- **Стресс** – проверяется моментально высокая нагрузка на ПО. В случае баз данных это может быть высокое количество запросов в единицу времени. В случае пользовательских интерфейсов это может быть большое количество нажатий на одну и ту же кнопку.
- **Стабильности** – проверяется то, насколько хорошо ПО работает под определенной нагрузкой в течение длительного промежутка времени.

На пересечении **конфигурационного и UI тестирования** может возникнуть **тестирование совместимости**, направленное на проверку того, что интерфейс корректно работает при различных конфигурациях ПО и платформ (например, разработчики ПО часто самостоятельно решают, насколько одинаково будет выглядеть создаваемый продукт в различных версиях операционных систем).

Также в рамках **UI тестирования** иногда отдельно проводят:

- **UI функциональное тестирование**, направленное на проверку того, что все элементы интерфейса действительно выполняют свою работу (не просто отображаются, но и запускают те действия, которые они должны запускать).
- **Юзабилити тестирование** – проверка того, что пользователь действительно сможет пользоваться созданным ПО. Тестируется то, что иногда не входит в формальные требования: скорость реакции интерфейса, место положения кнопок друг относительно друга (например, для большинства локаций кнопка «вперед» находится справа, а кнопка «назад» - слева). Сюда же входит целый класс мероприятий, направленный на то, какая из двух конфигураций интерфейса больше подходит пользователям.

На срезе **UI тестирования** и **Security/Compliance тестирования** проводят **тестирование локализации**, к которому относится проверка того, что все внешние и внутренние компоненты системы корректно переведены, надписи помещаются в кнопках, а кнопки находятся на своих местах.

В дальнейшем в нашем курсе мы сосредоточимся на функциональном тестировании.

Функциональное тестирование можно разделять в зависимости от степени изолированности компонентов следующим образом:

- Тестирование продукта целиком – проверяется то, как продукт справляется со своими задачами.
- Тестирование отдельных компонент.
- Тестирование классов.
- Тестирование методов или функций.

Чаще всего процесс тестирования выстраивается снизу-вверх по представленному выше списку. Сначала тестируются функции, затем классы, затем классы, объединенные в компоненты, затем компоненты, объединенные в продукт.

Тестирование продукта целиком или групп модулей иногда называют **системным, «сквозным» или end-to-end тестированием**. Система проверяется сценариями, действие которых проходит через весь компонент или всю систему, например, начинается и заканчивается где-то в пользовательском интерфейсе и никаких проверок «внутри» системы не выполняется.

Тестирование продукта, модулей и классов может быть **интеграционным** – в том случае, когда проверяется интеграция модулей в продукт, классов в модули, а функций в классы.

Тестирование модулей, классов и функций может быть **unit или модульным тестированием** – проверкой на работоспособность некоторой условной «единицы». Важно отметить, что тестирование отдельного компонента или модуля unit-тестированием называется редко. Модуль состоит из большого количества компонент и крайне сложно воспринимать его как отдельную единицу. Однако существует определенный класс тестов, которые достаточно просты, но проверяют функциональность целого модуля. Такие тесты формально могут называться unit-тестами.

Unit-тестирование – это часто первый уровень тестирования, который выполняется самими разработчиками, которые пишут во многих случаях автоматизированные тесты, выполняемые при каждом действии по созданию/изменению кода. При создании unit-тестов часто используются дополнительные фреймворки, «заглушки», вспомогательные драйверы и фейковые (mock) объекты. Заглушки необходимы для того, чтобы изолировать тестируемый объект. Если тестируемый класс или метод зависит от поведения сторонних классов или методов, то в рамках unit-тестирования мы можем отказаться от развертывания всей необходимой инфраструктуры и вместо этого использовать заглушки с предопределенным поведением. Например, если тестируется коннектор к базе данных, то в рамках unit тестирования можно не подключаться к реальной базе, а создать объект-заглушку, которая отвечает от имени базы предопределенными ответами.

Интеграционное тестирование предполагает проверку нескольких компонент, объединенных друг с другом. Основная задача такого тестирования – выявление ошибок во взаимодействии: протоколах, обмене данными, несоответствии методов и вызовов. В этом типе тестирования также используются заглушки.

При системном тестировании проверяются сложные сценарии: например, при нажатии кнопки пользователем данные проходят через всю систему, попадают в базу, распределяются, выполняется обработка данных, пользователю демонстрируется результат.

При end-to-end или сквозном тестировании проверяется вся собранная система, а также взаимосвязанные системы, через которые проходят принимаемые и отправляемые данные.

Поговорим теперь о видах тестирования по знаниям о продукте:

Black-box тестирование (методом «черного ящика») – подход, при котором в процессе тестирования мы не можем заглядывать «внутрь» компонента. Чаще используется в рамках системного или при определенных видах интеграционного тестирования.

White-box тестирование (методом «белого ящика») – подход, при котором в процессе тестирования мы можем заглядывать «внутрь» компонента. Чаще используется при unit-тестировании, а также определенных типах интеграционного.

Grey-box тестирование (методом «серого ящика») – промежуточный вид тестирования, при котором у нас есть доступ лишь к ряду компонентов системы.

После прочтения данного материала мы просим вас перейти к выполнению задания для самостоятельной работы, представленного в данном модуле.