Лекция 4. Объекты тестирования. Типы ошибок. Принципы тестирования

4.1. Объекты тестирования

В качестве **объектов тестирования** обычно рассматриваются компоненты программного обеспечения: система, сборка, интерфейсы, модули и другие (рис. 4.1), а также требования (есть мнение, что именно в них закладывается 50-70% всех проблем с ПО), архитектура, документация.

Рисунок 4.1 – Объекты тестирования

Цель тестирования – поиск ошибок. Число ошибок в том или ином компоненте ПП зависит от его сложности и размера. Числом обнаруженных ошибок определяется и эффективность тестирования. Кроме слова **«ошибка»** используют термины, «дефект», «проблема» «баг». Обычно они трактуются как синонимы и обозначают любое расхождение ожидаемого и фактического поведения системы. Но есть нюансы. Например, термин «дефект» – недостаток в компоненте или системе, а «ошибка» как бы подразумевает чью-то ответственность за нее, недостаток добросовестности программиста, леность или некомпетентность.

Отдельные небольшие части приложения (функции, модули и т.п.) необходимо проверить на работоспособность и правильность их функционирования на разных типах входных данных (в том числе и неправильных), в разных условиях. Это элементы, из которых строится система и их качество играет значительную роль в успешности проекта (даже, если удастся построить дом из гнилой древесины, вряд ли он прослужит долго и будет комфортным для его обитателей). Малый размер объектов позволяет осуществить проверку практически в полном объеме. Однако размер компонентов системы может бать сравнительно большим. В этом случае число проверок увеличивается и, возможно, проконтролировать получится не всё.

Работоспособность компонент ещё не гарантирует правильность работы их комбинаций и системы в целом. Необходимо выявление проблем в интерфейсах и взаимодействии между интегрируемыми ранее проверенными корректными компонентами (например, информация передается неправильно). Проверяться должны и информационные, и управляющие связи. Для тестирования

управляющих связей применяют тесты, составленные в соответствии со стратегией «белого ящика» по критерию покрытия всех маршрутов программы.

В конечном итоге проверяют интегрированную систему, как единое целое, собранное из частей, проверенных на предыдущих стадиях. Как правило, знаний о внутреннем устройстве системы при этом не требуется, взаимодействие с приложением осуществляется с позиций конечного пользователя. Кроме проверки на соответствие исходным требованиям к системе, работоспособности и правильности ее функционирования на разных типах входных данных необходимо проверить:

- работоспособность приложения в каждой из возможных его конфигураций.
- удобство установки (настройки, инсталляции).
- совместимость – способность приложения к взаимодействию с существующими системами и программными решениями: с браузерами и их версиями, с мобильными устройствами, с аппаратной платформой, операционной системой, сетевой инфраструктурой и так далее
- пользовательский интерфейс: его корректность и удобство.
- удобство использования
- производительность. Определяют, как быстро работает система или её часть в заданных условиях. Основными показателями производительности приложения являются время выполнения запроса, потребление ресурсов центрального процессора, потребление оперативной памяти и некоторые другие.
- безопасность и т.д.

Как уже отмечалось, в 70-е годы 20-го века акцент тестирования сместился в сторону доказательства неработоспособности программы в некоторых заданных условиях, а в 80-е годы тестирование перестало быть только заключительной стадией проекта и стало рассматриваться как элемент процесса обеспечения качества.

Особое внимание следует уделить тестированию требований. Требование – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи. Кроме функциональных, есть нефункциональные требования (например, максимально допустимое время отклика, число пользователей системы, удобство интерфейса, показатели качества и эффективности).

Требования позволяют понять, что и с соблюдением каких условий система должна делать и предоставляют возможность оценить масштаб изменений и управлять изменениями. Они являются основой для формирования плана проекта (в том числе плана тестирования), помогают предотвращать или разрешать конфликтные ситуации, упрощают расстановку приоритетов в наборе задач, позволяют объективно оценить степень прогресса в разработке проекта.

Если проблема в требованиях будет выяснена на стадии их определения, её решение может свестись к исправлению пары слов в тексте, в то время как недоработка, вызванная пропущенной проблемой в требованиях и обнаруженная на стадии эксплуатации, может даже полностью уничтожить проект.

В числе показателей качества требований, которые определяются при их тестировании следующие:

– Завершённость. Требование является полным и законченным с точки зрения представления в нём всей необходимой информации, ничто не пропущено по соображениям «это и так всем понятно».

– Атомарность, единичность. Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершённости и оно описывает одну и только одну ситуацию.

– Непротиворечивость, последовательность (consistency88). Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.

– Недвусмысленность. Требование должно быть описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок, должно допускать только однозначное объективное понимание и быть атомарным в плане невозможности различной трактовки сочетания отдельных фраз.

– Выполнимость. Требование должно быть технологически выполнимым и реализуемым в рамках бюджета и сроков разработки проекта

– Обязательность, нужность и актуальность. Если требование не является обязательным к реализации, оно должно быть просто исключено из набора требований. Если требование нужное, но «не очень важное», для указания этого факта используется указание приоритета. Также исключены (или переработаны) должны быть требования, утратившие актуальность.

– Прослеживаемость. Прослеживаемость бывает вертикальной и горизонтальной. Вертикальная позволяет соотносить между собой требования на различных уровнях требований, горизонтальная позволяет соотносить требование с тест-планом, тест-кейсами, архитектурными решениями и т.д.

– Модифицируемость. Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований. Можно говорить о наличии модифицируемости в том случае, если при доработке требований искомую информацию легко найти, а её изменение не приводит к нарушению иных описанных в этом перечне свойств.

– Ранжированность по важности, стабильности, срочности. Важность характеризует зависимость успеха проекта от успеха реализации требования. Стабильность характеризует вероятность того, что в обозримом будущем в требование не будет внесено никаких изменений. Срочность определяет распределение во времени усилий проектной команды по реализации того или иного требования.

– Корректность и проверяемость. Фактически эти свойства вытекают из соблюдения всех вышеперечисленных (или можно сказать, что они не выполняются, если нарушено хотя бы одно из вышеперечисленных). Проверяемость подразумевает возможность создания объективного тест-кейса (тест-кейсов), однозначно показывающего, что требование реализовано верно и поведение приложения в точности соответствует требованию

Объектом тестирования является и архитектура создаваемой системы. Архитектура ПП представляет собой набор структур или представлений, имеющих различные уровни абстракции и показывающих разные аспекты (структуру классов программного обеспечения, структуру развертывания, возможные

сценарии взаимодействий компонентов и пр.) программной системы. При этом уровень абстракции данного представления является аналогом масштаба географической карты и отражается в различных видах схем. Например, функциональный (логический) или структурный вид, вид процесса выполнения или вид развертывания. Также архитектура может иметь несколько представлений, отражающих интересы различных групп заинтересованных лиц. Например, вид с точки зрения действий пользователя.

Тестирование должно показать, что все требования распределены между компонентами архитектуры, что приложение с такой архитектурой работоспособно и будет удовлетворять предъявляемым к нему требованиям, что приложение с такой архитектурой реализуемо в принципе.

Также объектом тестирования является документация продукта и документация проекта: тест-кейсы и наборы тест-кейсов, технические спецификации (схемы баз данных, описания алгоритмов, интерфейсов и т.д.), листинги программ, различные руководства и т.п. Эта документация используется проектной командой во время разработки и поддержки продукта. Пользовательскую и сопроводительную документацию (встроенная помощь, руководство по установке и использованию, лицензионные соглашения и т.д.) применяют на этапе эксплуатации ПП. Маркетинговую документацию представители разработчика или заказчика используют как на начальных этапах (для уточнения сути и концепции проекта), так и на финальных этапах развития проекта (для продвижения продукта на рынке). Вся документация должна быть актуализирована, а до этого должны быть выявлены несоответствия между полученной системой и ее описанием.

4.2. Типы ошибок

Существуют различные классификации ошибок.

В зависимости **от тестируемого элемента программной системы** выделяют:

а) Ошибки ***модулей.*** Т.к. размер и функциональность компонентов обычно невелики, их сравнительно легко обнаружить.

б) Ошибки ***интеграции.*** Возникают при взаимодействии правильных и корректных компонентов. Выявить и предотвратить их труднее вследствие сложности взаимосвязей. Кроме того, число взаимодействий компонентов растет по законам комбинаторики и может оказаться очень большим.

в) ***Системные*** ошибки – это ошибки, обнаруживаемые при тестировании системы в целом. Обусловлены отклонением характеристик ПП и его функционирования в реальных условиях от планируемых при проектировании. Сложнее предсказать возникновение тех или иных ситуаций: разные прогоны программы с одними и теми же входными данными могут привести к различным результатам. Это ведет к увеличению сложности ошибок.

г) Ошибки ***документации***: в требованиях, в проекте, в руководстве пользователя и т.п.

По процессу, в котором ошибка проявилась, различают [6]:

1. Ошибки ***компиляции*** (в основном, синтаксические) – это ошибки, обнаруживаемые компилятором (транслятором, интерпретатором) при проведении синтаксического и частично семантического анализа. Обычно

сопровождается развернутым комментарием с указанием ее местоположения. Считаются самыми простыми;

2. Ошибки ***компоновки*** – ошибки, фиксируемые компоновщиком (редактором связей) при объединении модулей. Связаны с внешними ссылками.

3. Ошибки ***выполнения*** включают в себя следующее:

– ошибки *определения данных*: ошибки передачи (ввода-вывода), преобразования, перезаписи, неправильные данные.

– ошибки *логические*. Связаны с описанием логики работы программы: а) при *проектировании* (неверно построен алгоритм, сформирована структура данных, выбран метод обработки данных, и т.п.) и б) при *кодировании* (неверно реализован алгоритм, структура данных, метод обработки данных, допущены ошибки межмодульных интерфейсов и др.).

– ошибки *накопления погрешностей:* некорректное использование приближенных методов вычислений, некорректное отбрасывание дробной части, игнорирование ограничений разрядной сетки и т.п.

В классификации **по сложности диагностирования** синтаксические ошибки не рассматриваются. Выделяют:

1. Ошибки ***общего (несинтаксического) характера***: логические, ошибки в циклах, в описании переменных, ошибки при работе с массивами и другие. Не обнаруживаются в ходе синтаксического контроля, но сравнительно легко выявляются в ходе тестирования;

2. Ошибки ***специального вида***: связаны с применением препроцессора, с неправильным результатом операций, исчезающие ошибки и т.д. Особенно трудны для диагностирования.

Еще одна классификация**, по причине появления**:

1.Ошибки ***анализа*** (***алгоритмические).*** Связаны с неточной постановкой и спецификацией задачи; с неполным учетом ситуаций, которые могут возникнуть (например, пренебрежение возможностью появления отрицательных значений переменных, малых и больших величин); с неверным решением задачи (логические ошибки: не заданы начальные значения переменных, условия инициирования или окончания цикла, неправильно указаны ветви алгоритма).

2. Ошибки ***общего характера***, возникшие из-за недостаточного знания или понимания программистом языка программирования или самой машины, допущенные при кодировании алгоритма (например, команды, используемые в программе, не обеспечивают последовательности событий).

3. Ошибки ***физического характера*** (***программные)***, возникшие вследствие неправильной записи исходного текста программ на языке программирования (пропуск операторов, отсутствие необходимых данных, неверный формат данных) и ошибок трансляции программ в объектный код;

4. ***Синтаксические*** ошибки. Вызванные всякими нарушениями требований языка программирования (пропуск знака пунктуации, несогласованность скобок, неправильная запись имени переменной, зарезервированных слов). Примерами синтаксических ошибок, охватывающих взаимодействие двух или более операторов, могут служить: противоречивые команды, отсутствие условий окончания цикла, запрещенный переход. Далеко не все из них обнаруживаются транслятором.

5. ***Технологические*** ошибки: подготовки машинных носителей и документации, ввода программ в память компьютера и их вывода на отображающие средства

По процессу разработки, в ходе которого допущены (ошибки встречаются не только в программном обеспечении), выделяют: ошибки определения требований, ошибки проектирования, ошибки развертывания, ошибки документации и др.

По характеру проявления ошибки можно разделить на первичные и вторичные. ***Первичными*** называют исходные дефекты (именно о них шла речь выше). Источником первичных ошибок являются сложность ПП. Значительная часть системных ошибок в качественном программном обеспечении, является следствием непредсказуемого взаимодействия компонентов, результатом непредсказуемых побочных эффектов, вызываемых совершенно невинными, на первый взгляд, процессами. Не исключение и ошибки конкретных программистов.

Наиболее существенными **факторами**, влияющими на вероятность появления первичных ошибок, являются:

– применяемые технологии и уровень автоматизации процессов разработки;
– класс ПП, размер и типы тестируемых программных компонентов;
– эффективность тестирования;
– виды и достоверность эталонов.

Вторичные ошибки – результаты проявления первичных ошибок. Делятся на три категории:

- мелкие сбои: ущерб незначительный, работоспособности программного средства практически не снижается;
- ординарные отказы: снижают качество ПП, но наносимый ими ущерб находится в допустимых пределах;
- катастрофические отказы: ущерб значительный, влияет на безопасность применения ПО.

По критичности (влиянию на работоспособность программного продукта) выделяют ошибки:

- ***блокирующая*** (Immediate, Blocker). Приводит приложение в нерабочее состояние и не позволяет полноценно проводить его тестирование. Обычно выявляется в ходе первичного запуска новой версии ПП;
- ***критическая*** (Crit – Urgent) – имеются проблемы с выполнением ключевых функций приложения, но тестирование может быть продолжено;
- ***значительная*** (High), но не критичная. Есть проблемы вызова тестируемой функции, однако существуют другие способы ее активизации. И работа с приложением может быть продолжена;
- ***незначительная*** (Normal) – обычно очевидная проблема пользовательского интерфейса и локализации. Не затрагивает бизнес-логику тестируемого компонента;
- ***тривиальная*** (Low) – малозаметна, плохо воспроизводится, связана с проблемами сторонних библиотек или сервисов.

В некоторых организациях используется более простая шкала: ошибки значительные (Hight), средние (Medium), незначительные (Low.

В таблице 4.1 представлена классификация ошибок, выявленных при тестировании одного из программных продуктов, разработанных одной из вологодских компаний. Проект (далее Проект1) заключался в дополнении функционала в части кредитования возможностью построения различных графиков погашения задолженности по кредиту. Обнаруженные ошибки были разбиты по критичности и по типу. Были выделены следующие типы ошибок:

- интерфейсные – ошибки связанные с внешним видом программы (например, не описана клавиша, отсутствует пункт меню);
- ошибки в операциях. Проявились при выполнении операций, предусмотренных программой: выдача, погашение кредита, начисление процентов по кредиту и т.п.;
- ошибки при расчете графика – ошибки, возникшие при формировании графика погашения задолженности по кредиту. Данные ошибки были выделены в отдельный тип в связи с тем, что Проект1 изменяет работу с графиками погашения;
- ошибки проектирования. Связаны с некорректным описанием технического задания на разработку.

Таблица 4.1 – Результаты тестирования Проекта 1

Критичность ошибки	Количество ошибок				Итого
	Интерфейсные ошибки	Ошибки в операциях	Ошибки при расчете графика	Ошибки проектирования	
Hight	5	33	32	12	82
Medium	5	13	8	8	34
Low	4	0	1	1	6
Итого	14	46	41	21	122

Все выявленные проблемы должны быть зафиксированы в соответствующих документах.

4.3. Принципы тестирования

1. *Ошибки в программе есть* . Иначе работа тестировщика теряет смысл: обнаружение ошибок – цель тестирования. Хорошим считается тест, который выявляет наличие ошибки в ПП, а не демонстрирует правильность его работы.

2. ***Ожидаемый результат*** *– обязательный элемент теста (помимо совокупности исходных данных)*. Если он заранее не зафиксирован, то высока вероятность того, что полученные результаты будут приняты за ожидаемые.

Например, введены 3 одинаковых числа – стороны треугольника. Программа выдала сообщение: «Треугольник равнобедренный равносторонний». Вроде все верно, однако слово «равнобедренный» здесь лишнее.

3. *Простота проверки тестов* вручную или иным способом (например, бухгалтерский отчет создается по одним и тем же исходным данным с помощью новой и старой программы). *В* основном это касается вычисляемых данных и выводов, сделанных на основе этих вычислений.

4. *Готовность тестов* до начала процесса тестирования. Это относится и к исходным данным, и к ожидаемым результатам (часто тесты придумываются на ходу, причем только исходные данные).

5. Разработка п*ервых тестов* на этапе определения и анализа требований. Это позволяет уточнить постановку задачи, выявить узкие места в требованиях, нюансы, незаметные на первый взгляд, но проявляющиеся при попытке определить результат, соответствующий конкретным входным данным.

6. *Сформулированность целей* тестирования *и* критериев, которые будут свидетельствовать о достижении целей *до начала* тестирования.. Например, это может быть набор тестов, которые следует обязательно выполнить для обеспечения полноты тестирования. Независимо от применяемых критериев разработка тестов должна вестись систематически, по определенной методике.

7. Обязательность *фиксирования выполненных тестов и реально полученных результатов*. Отсутствие информации о пройденных тестах и результатах тестирования не позволяет отделить проверенные участки от непроверенных и оценить количество ошибок в программе, не дает информации для принятия решения об окончании тестирования.

8. Одинаково тщательные подготовка и выполнение тестов *для позитивного и негативного тестирования*. На практике часто ограничиваются тестированием на правильных входных данных, забывая о неправильных.

9. *Обязательность проверки как функционала ПП, так и отсутствия негативных последствий от работы программы (программа не делает того, чего делать не должна).* Особенно это важно для изменений в глобальной среде. Если программа выдает правильные результаты, но при этом затирает половину винчестера, то едва ли ее можно признать правильной.

10. Полное *объяснение результатов* теста.

11. *Недопустимость изменения программы* с целью упрощения тестирования*.* Тестировать после этого вы будете уже другую программу.

12. *Обязательность повторного тестирования исправленной программы.* Вероятность внесения в программу новой ошибки (параллельно с позитивными изменениями) оценивается в 20-50%. Для особо сложных систем эта вероятность может быть значительно выше.

13. *Ошибки кучкуются.* Предположение о равномерном распределении ошибок по всему тексту программы исходит из гипотезы о том, что свойства всех её частей примерно одинаковы. Однако, программы обычно неоднородны, и большая часть ошибок окажется в хуже специфицированных, спроектированных, закодированных частях. В результате вероятность выявления в модуле не замеченных ранее ошибок тем выше, чем больше ошибок обнаружено в нём в ходе предыдущих проверок.

14. *Автор программы не должен выполнять её окончательное тестирование.* Тестирование программы – процесс разрушительный. Цель его – выявление дефектов в программе. Психологически далеко не каждый программист способен отделить себя (не досмотрел, не понял, ошибся) от своего «шедевра» (неверно реагирует, неверно считает) и объективно оценить его достоинства и недостатки. Кроме того, при длительной работе с одним текстом взгляд «замыливается» (видит то, чего нет и не видит того, что есть).