Лекция №3

Тестирование интерфейсов

Тесно связанно с интеграционным тестированием

Выполняется в том случае, когда подсистемы интегрируются в более крупные подсистемы.

Цель тестирования интерфейсов – выявить один из двух классов ошибок (ошибки самих интерфейсов и ошибки вызванные неправильными представлениями о самих интерфейсах).

При тестировании отдельных объектов трудно выявить ошибки интерфейсов.

Типы интерфейсов:

1. Параметрические интерфейсы
   В качестве параметра передаются ссылки на данные или функции
2. Интерфейсы разделяемой памяти
   Существует участок памяти, доступный для чтения и записи многими модулями
3. Процедурные интерфейсы
   Одна подсистема инкапсулирует набор процедур, а другие подсистемы имеют возможность эти процедуры вызывать
4. Интерфейсы передачи сообщений
   Одна подсистема запрашивает какой-то сервис у другой, передавая ей сообщение, которое содержит сам запрос на действие и необходимые входные данные.

Типы ошибок в интерфейсах:

1. Неправильное использование интерфейса
   Один компонент обращается к другому и совершает ошибку передавая данные не тех типов, в ошибочном порядке или количестве
2. Неправильное понимание интерфейсов
   Один из модулей вызывает какой-то метод другого модуля, ожидая от него определённого поведения, но ожидаемое поведение не совпадает с реальным поведением вызываемого метода
3. Ошибки синхронизации
   Характерны для интерфейсов разделяемой памяти и, как правило, такие ошибки встречаются в системах реального времени

Сложность тестирования интерфейсов заключается в том, что ошибки могут проявятся только в нештатных ситуациях. Ошибки могут стать результатом взаимодействия ошибок в разных модулях.

Общие правила тестирования интерфейсов:

1. Необходимо просмотреть код и составить список вызовов к внешним компонентам
2. Разработать тестовые наборы, при которых данные, передаваемые внешним компонентам принимают граничные значения
3. Если между интерфейсами передаются указатели, необходимо тестировать с нулевыми указателями
4. При вызове компонента через процедурный интерфейс необходимо тестировать ситуацию, вызывающую сбой при тестировании компонента
   Наиболее распространённая ошибка, выявляемая на этапе тестирования интерфейсов, – это неправильное понимание интерфейса
5. При тестировании нужно передавать сообщения с интенсивностью в разы выше чем штатной работы. Эти же тесты позволят выявить ошибки синхронизации
6. При тестировании интерфейса общей памяти, то нужно менять порядок активации компонент
   Т.о. можно выявить ошибки, вызванные тем, что при проектировании были сделаны неявные допущения работы компонента

7. При тестировании интерфейсов статические методы тестирования более эффективны, чем динамические
8. Выявление ошибок интерфейсов

Нагрузочное тестирование

После полной интеграции системы появляется возможность оценить такие интеграционные свойства системы, как производительность и надёжность. Для того, что бы убедиться, что система может работать под заданной нагрузкой необходимо провести нагрузочное тестирование. Планируется серия тестов с постепенным увеличением нагрузки (повышение интенсивности подачи данных). Интенсивность поступлении входящих данных повышается до тех пор, пока не начнёт понижаться производительность системы. При превышении некоторого порога система начнет выходить из строя.

Нагрузочное тестирование предполагает, что мы проверяем работу системы, подавая данные с максимально допустимой нагрузкой. Интенсивность поступления входных данных постепенно повышаем до тех пор, пока система не сломается.

Тестируется поведение программы во время сбоя – важно знать как она себя ведёт во время сбоя, так как в процессе эксплуатации такие ситуации возможны. Необходимо убедится, что не происходят никакие необратимые действия, никакие критические данные не удаляются и никакие необратимые аппаратные сбои.

Нагрузочное тестирование позволяет выявить ошибки, которые невозможно выявить в нормальных режимах эксплуатации программы.

Тестирование ОО-систем

Основные уровни тестирования:

1. Тестирование отдельных методов (как чёрный так и белый ящик)
2. Тестирование отдельных классов (класс – чёрный ящик)
3. Тестирование кластеров объектов – тестирование взаимосвязанных групп объектов (замена нисходящего или восходящего тестирования)
4. Тестирование всей системы в целом

Методики тестирования ОО программ:

1. Тестирование сценариев и вариантов использования – наиболее эффективное
2. Тестирование потоков (событий или данных)
3. Тестирование взаимодействий между объектами (интеграционное тестирование для ОО)

Системное тестирование

Проводится после полной сборки ПП. Отличается от всех видов тестирование:

1. Рассматривает систему в целом
2. Тестирование основывается на пользовательском интерфейсе или API (внешние интерфейсы программы)
3. Задача тестирования – проверить работоспособность системы
4. Выявление аномалии: неправильное использование ресурсов системы, несовместимость системы с окружением, непредусмотренные сценарии использования ПП, неудобства использования и т.д.
5. Система рассматривается только как чёрный ящик

Критерии тестов системного тестирования:

1. Полнота решения функциональных задач
2. Стрессовое тестирование – на предельных объемах
3. Тестирование корректности использования ресурсов (утечка памяти, возврат ресурсов после использования)
4. Оценка производительности (скорости работы)
5. Тестируется эффективность защиты от искажённых данных и некорректных действий
6. Тестируется инсталляция и конфигурация на разных платформах
7. Проверяется корректность документации

Системное тестирование выполняется либо вручную, либо с помощью систем для автоматизации тестирования на основе интерфейса пользователя.

Регрессионное тестирование

Выполняется когда в систему были внесены изменения. Цель – определить минимальный достаточный набор тестов, что бы избежать полного перетестирования системы и ошибок взаимодействия изменённого модуля с другими.

Верификация и аттестация (валидация) информационных систем

Процесс проверки и анализа в ходе которого проверяется соответствие ПП спецификациям и ожиданиям заказчика. Верификация отвечает на вопрос правильно ли создана система (в соответсвии со спецификацией). Аттестация отвечает на вопрос правильно ли работает система (с точки зрения ожидания заказчика).

Подходы

1. Инспектирование ПО
   Предполагается проверка исходного кода, технического задания, докумментацией и т.д. некоторой инспекционной группой
2. Тестирование
   *2.1.*Тестирование дефектов
   *2.2.*Статистическое тестирование – анализ интегральных характеристик системы (производительность, надёжность)

Инспектирование

Более 60% ошибок можно найти просто просматривая код программы. Если формализовать подход, то до 90%. Инспекционная группа включает в себя следующие роли: автора, рецензента, инспектора и координатора. Автор представляет результаты своей работы, рецензент зачитывает исходный код(?), инспектор проводит тесты, координатор всем управляет. Для начала инспектирования необходимо наличие точной спецификации кода и знание участниками инспекционной группы стандартов разработки в этой команде, необходима синтаксически корректная версия программы.

Инспектирование:

1. Планирование – координатор составляет план инспектирования, проверяется что бы программа и её спецификация находились в завершённом состоянии
2. Предварительный просмотр – автор представляет свою программу, описывает зачем всё это нужно
3. Индивидуальная подготовка – каждый участник инспекционной группы самостоятельно просматривает код программы, ищет какие-то дефекты
4. Собрание инспекционной группы – не дольше 2х часов, внимание сосредоточено на выявление дефектов и аномалий, варианты исправления группа автору не предлагает
5. Исправление ошибок – автор модифицирует программу, исправляя онаруженные ошибки
6. Доработка программы (опц.) – координатор принимает решения, нужно ли проводить инспектирование повторно (к шагу 1). Если не требуется, то все обнаруженные дефекты документируются и утверждаются руководителем группы.

Статический анализ программ

Статические анализаторы программ – это инструментальные средства, которые проверяют исходный код программы и выявляют возможные ошибки и противоречия. Цель – привлечь внимание разработчика к аномалиям в программе.

Основные этапы:

1. Анализ потоков управления – анализируются циклы, точки входа/выхода, неиспользуемый код
2. Анализ использования данных – обнаружить переменные, используемые без инициализации или неиспользуемые присваивания, условные операторы с избыточными условиями.

3. Анализ интерфейса – проверяется согласованность отдельных частей программ (функций), правильность объявления функции и их использования (для языков с не строгой типизацией)
4. Анализ потоков данных – анализируется зависимость между выходными переменными и входными
5. Анализ ветвей программы – ветви, которые никогда не выполнятся и т.д.