Оценка сложности тестирования и методика тестирования объектно-ориентированного программирования. пример интеграционного тестирования

Оценка сложности тестирования и методика тестирования объектно-ориентированного программирования. пример интеграционного тестирования

Оценка сложности тестирования в объектно-ориентированном программировании (ООП) зависит от множества факторов, связанных с особенностями ООП-парадигмы: наследования, полиморфизма, инкапсуляции, динамического связывания и передачи сообщений между объектами. Ключевыми параметрами являются:

- Количество точек входа в класс (Kext=Kmsg+Kem*Kext=Kmsg+Kem*), где Kmsg*Kmsg* — число методов, обрабатывающих сообщения, а Kem*Kem* — число методов, доступных для прямого вызова из других классов.
- Число P-путей (процедурных путей) — последовательностей прямых вызовов методов внутри класса или между классами.
- Число MM-путей (Method/Message путей) — цепочек выполнения методов, связанных передачей сообщений. MM-путь заканчивается, когда метод не порождает новых сообщений («сообщение покоя»). multiurok.ru +1
- Глубина иерархии наследования — влияние родительских классов на дочерние может усложнять тестирование, так как изменения в родительском классе могут повлиять на поведение подклассов.
- Сложность взаимодействия объектов — количество сообщений и их маршрутов, а также возможные неочевидные комбинации взаимодействий между объектами.

Формула оценки сложности тестирования класса Cls*Cls* может иметь вид:

V(Cls,C)=f(Kmsg,Kem,глубина иерархии,сложность взаимодействий),$V(Cls,C)=f(Kmsg,Kem$,глубина иерархии,сложность взаимодействий),

где ff — функция, учитывающая перечисленные параметры.

Методика тестирования ООП включает несколько подходов и стратегий:

1. Инкрементальное тестирование «снизу вверх». Интеграция проводится поэтапно: от базовых классов к верхним уровням иерархии. При этом можно использовать тесты для классов-родителей тестируемого класса, что обусловлено принципом наследования. multiurok.ru +1
2. Анализ потока данных и потока управления. Методы основаны на построении управляющего графа (УГ) класса, где узлы — методы, а дуги — связи между ними (P- и MM-пути). Тестовые сценарии генерируются для покрытия всех возможных путей в графе. cyberleninka.ru +1
3. Состояние-ориентированное тестирование. Учитывает, что результат метода может зависеть от текущего состояния объекта. Тестовые случаи генерируются для проверки поведения методов в различных состояниях объекта. cyberleninka.ru +1
4. Событийно-ориентированное тестирование. Учитывает событийную управляемость ООП: передача управления осуществляется через генерацию и обработку сообщений. Тестируются цепочки обработки событий (MM-пути). multiurok.ru +1
5. Тестирование интерфейсов и абстракций. Проверяется корректность реализации интерфейсов, абстрактных классов и их взаимодействие с конкретными реализациями.
6. Использование моков (mocks) и заглушек (stubs). Для изоляции компонентов и имитации внешних зависимостей применяются

фреймворки для мокинга или техника внедрения зависимостей (dependency injection).

7. Тестирование исключений. Проверяется, что объекты корректно обрабатывают ошибки и исключения.
8. Проверка единообразия. Оценивается соблюдение единообразия в именовании, стилях кодирования и соответствии дизайн-паттернам.

Пример интеграционного тестирования в ООП

Рассмотрим систему управления библиотекой, состоящую из классов Book, Member и Library. Класс Library управляет взаимодействием между книгами и членами библиотеки.

Цель тестирования: проверить, что при выдаче книги члену библиотеки состояние книги (доступность) и данные о выданной книге в профиле члена обновляются корректно.

Шаги тестирования:

1. Создать экземпляры классов: book = new Book("ISBN-123", "Название", true) (книга доступна) и member = new Member("ID-456", "Имя").
2. Вызвать метод Library.lendBook(book, member).
3. Проверить, что:
    - атрибут isAvailable у book изменился на false;
    - в списке выданных книг у member появилась запись о book.

Что проверяется:

- корректность взаимодействия между объектами классов Book, Member и Library;

- правильность обработки сообщений (например, если метод lendBook генерирует события, которые обрабатываются другими частями системы);
- соблюдение инвариантов (книга не может быть выдана, если она уже недоступна).

Возможные сложности:

- если класс Library зависит от других сервисов (например, системы учёта), потребуется использовать моки для имитации их поведения;
- необходимо учитывать, что состояние объектов может изменяться в результате предыдущих операций, поэтому важно правильно инициализировать тестовое окружение.

Таким образом, интеграционное тестирование в ООП фокусируется на проверке взаимодействия между классами и объектами, учитывая специфику ООП-принципов и особенностей управления потоком выполнения.