Лабораторная работа 10. Особенности тестирования для объектно-ориентированного программирования

Цель работы: Изучить методы тестирования объектно-ориентированных программ

Теоретический материал

Программный проект, написанный в соответствии с объектно-ориентированным подходом, будет иметь Граф Модели Программы (ГМП), существенно отличающийся от ГМП традиционной "процедурной" программы. Сама разработка проекта строится по другому принципу - от определения классов, используемых в программе, построения дерева классов к реализации кода проекта. При правильном использовании классов, точно отражающих прикладную область приложения, этот метод дает более короткие, понятные и легко контролируемые программы.

Объектно-ориентированное программное обеспечение является событийно управляемым. Передача управления внутри программы осуществляется не только путем явного указания последовательности обращений одних функций программы к другим, но и путем генерации сообщений различным объектам, разбора сообщений соответствующим обработчиком и передача их объектам, для которых данные сообщения предназначены. Рассмотренная ГМП в данном случае становится неприменимой. Эта модель, как минимум, требует адаптации к требованиям, вводимым объектно-ориентированным подходом к написанию программного обеспечения. При этом происходит переход от модели, описывающей структуру программы, к модели, описывающей поведение программы, что для тестирования можно классифицировать как положительное свойство данного перехода. Отрицательным аспектом совершаемого перехода для применения рассмотренных ранее моделей является потеря заданных в явном виде связей между модулями программы.

Перед тем как приступить к описанию *графовой модели* объектно-ориентированной программы, остановимся отдельно на одном существенном аспекте разработки программного обеспечения на языке объектно-ориентированного программирования (ООП), например, C++ или C#. Разработка программного обеспечения высокого качества для MS Windows или любой другой операционной системы, использующей стандарт "look and *feel*", с применением только вновь созданных классов практически невозможна. Программист должен будет затратить массу времени на решение стандартных задач по созданию пользовательского интерфейса. Чтобы избежать работы над давно решенными вопросами, во всех современных компиляторах предусмотрены специальные библиотеки классов. Такие библиотеки включают в себя практически весь программный интерфейс операционной системы и позволяют задействовать при программировании средства более высокого уровня, чем просто вызовы функций. Базовые конструкции и классы могут быть переиспользованы при разработке нового программного проекта. За счет этого значительно сокращается время разработки приложений. В качестве примера подобной системы можно привести библиотеку Microsoft Foundation Class для компилятора MS Visual C++ [21]".

Работа по тестированию приложения не должна включать в себя проверку работоспособности элементов библиотек, ставших фактически промышленным стандартом для разработки программного обеспечения, а только проверку кода, написанного непосредственно разработчиком программного проекта. Тестирование объектно-ориентированной программы должно включать те же уровни, что и тестирование процедурной программы - модульное, интеграционное и системное. Внутри класса отдельно взятые методы имеют императивный характер исполнения. Все языки ООП возвращают контроль вызывающему объекту, когда сообщение обработано. Поэтому каждый метод (функция - член класса) должен пройти традиционное *модульное тестирование* по выбранному критерию C (как правило, C1). В соответствии с введенными выше обозначениями, назовем метод Mod_i, а сложность тестирования - $V(Mod_i,C)$.

Каждый класс должен быть рассмотрен и как субъект *интеграционного тестирования*. Интеграция для всех методов класса проводится с использованием инкрементальной стратегии снизу вверх. При этом мы можем переиспользовать тесты для классов-родителей тестируемого класса [22], что следует из принципа наследования - от базовых классов, не имеющих родителей, к самым верхним уровням классов.

Графовая модель класса, как и объектно-ориентированной программы, на интеграционном уровне в качестве узлов использует методы. Дуги данной ГМП (вызовы методов) могут быть образованы двумя способами:

- Прямым вызовом одного метода из кода другого, в случае, если вызываемый метод виден (не закрыт для доступа средствами языка программирования) из класса, содержащего вызывающий метод, присвоим такой конструкции название ***P-путь (P-path, Procedure path, процедурный путь)*** .

- Обработкой сообщения, когда явного вызова метода нет, но в результате работы "вызывающего" метода порождается сообщение, которое должно быть обработано "вызываемым" методом.

Для второго случая "вызываемый" метод может породить другое сообщение, что приводит к возникновению цепочки исполнения последовательности методов, связанных сообщениями. Подобная цепочка носит название *MM-путь (MM-path, Metod/Message path, путь метод/сообщение)* [23]. *MM-путь* заканчивается, когда достигается метод, который при отработке не вырабатывает новых сообщений (т. е. вырабатывает "сообщение покоя").

Пример *MM-путей* приведен на рисунке 10.1. Данная конструкция отражает событийно управляемую природу объектно-ориентированного программирования и может быть взята в качестве основы для построения *графовой модели* класса или объектно-ориентированной программы в целом. На рисунке 10.1 можно выделить четыре *MM-пути* (1-4) и один *P-путь* (5):

1. msg a → метод 3 → msg 3 → метод 4 → msg d

2. msg b → метод 1 → msg 1 → метод 4 → msg d

3. msg b → метод 1 → msg 2 → метод 5

4. msg c → метод 2

5. call → метод 5

Здесь класс изображен как объединенное множество методов.

Введем следующие обозначения:

Kmsg - число методов класса, обрабатывающих различные сообщения;

Kem - число методов класса, которые не закрыты от прямого вызова из других классов программы.

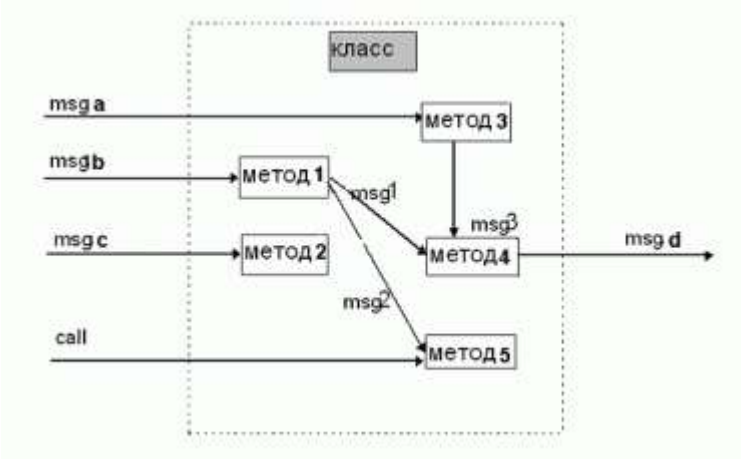

Рис. 10.1. Пример ММ-путей и Р-путей в графовой модели класса

Если рассматривать класс как программу Р, то можно выделить следующие отличия от программы, построенной по процедурному принципу:

- Значение Kext (число точек входа, которые могут быть вызваны извне) определяется как сумма методов - обработчиков сообщений Kmsg (например, в MS Visual C++ обозначаются зарезервированным словом afx_msg и используются для работы с картой сообщений класса) и тех методов, которые могут быть вызваны из других классов программы Kem. Это определяется самим разработчиком путем разграничения доступа к методам класса (с помощью ключевых слов разграничения доступа public, private, protected) при написании методов, а также назначении дружественных (friend) функций и дружественных классов. Таким образом, Kext = Kmsg + Kem, и имеет новый по сравнению с процедурным программированием физический смысл.

- Принцип соединения узлов в ГМП, отражающий два возможных типа вызовов методов класса (через *ММ-пути* и *Р-пути*), что приводит к новому наполнению для множества М требуемых элементов.

- Методы (модули) непрозрачны для внешних объектов, что влечет за собой неприменимость механизма упрощения графа модуля, используемого для получения *графа вызовов* в *процедурном программировании*.

С учетом приведенных замечаний, информационные связи между модулями программного проекта получают новый физический смысл, а формула *оценки сложности интеграционного тестирования* класса Cls принимает вид: V(Cls, C) = f (K_{msg}, K_{em})

В ходе *интеграционного тестирования* должны быть проверены все возможные внешние вызовы методов класса, как непосредственные обращения, так и вызовы, инициированные получением сообщений

Значение числа *ММ-путей* зависит от схемы обработки сообщений данным классом, что должно быть определено в спецификации класса. Например, для класса, изображенного в примере 5.4, сложность *интеграционного тестирования* V(Cls,C)=5 (множество неизбыточных тестов T для класса составляют 4 *ММ-пути* плюс внешний вызов метода 5, т. е. *Р-путь*).

Данные - члены класса (данные, описанные в самом классе, и унаследованные от классов-родителей видимые извне данные) рассматриваются как "глобальные переменные", они должны быть протестированы отдельно на основе принципов тестирования потоков данных.

Когда класс программы P протестирован, объект данного класса может быть включен в общий граф G программного проекта, содержащий все *ММ-пути* и все вызовы методов классов и процедур, возможные в программе рис. 10.2

Программа P, содержащая n классов, имеет сложность *интеграционного тестирования* классов

$$V(P,C) = \Sigma V(Cls_i, C)$$

Формальным представлением описанного выше подхода к тестированию программного проекта служит ***классовая модель программного проекта***, состоящая из дерева классов проекта рис. 10.3 и модели каждого класса, входящего в программный проект рис. 10.4.

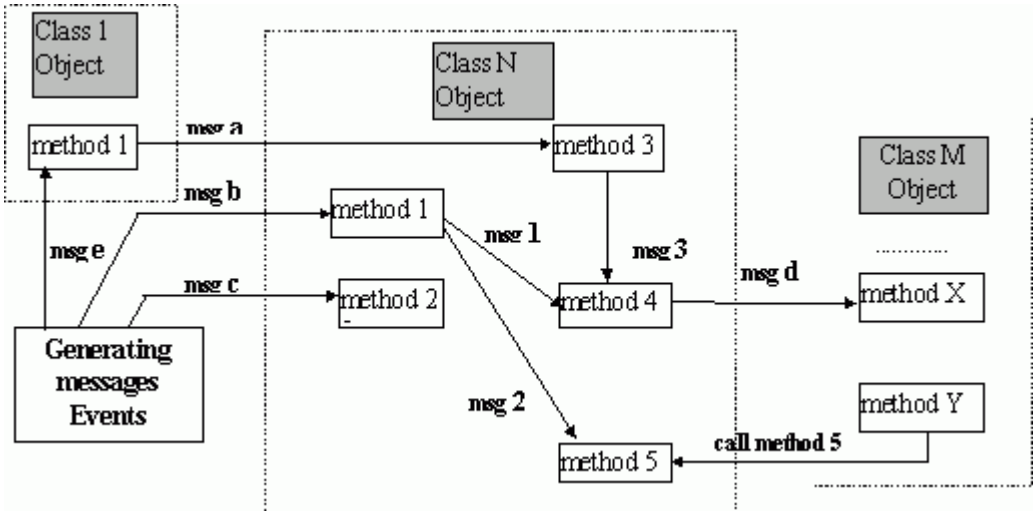

Рис. 10.2. Пример включения объекта в модель программного проекта, построенного с использованием ММ-путей и Р-путей

Таким образом и определяется *классовая модель* проекта для тестирования объектно-ориентированной программы. Как будет показано в дальнейшем, она поддерживает итерационный инкрементальный процесс разработки программного обеспечения.

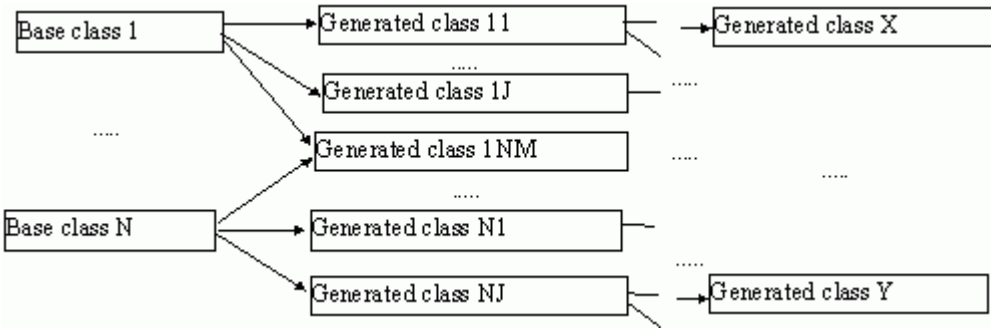

Рис. 10.3. Дерево классов проекта

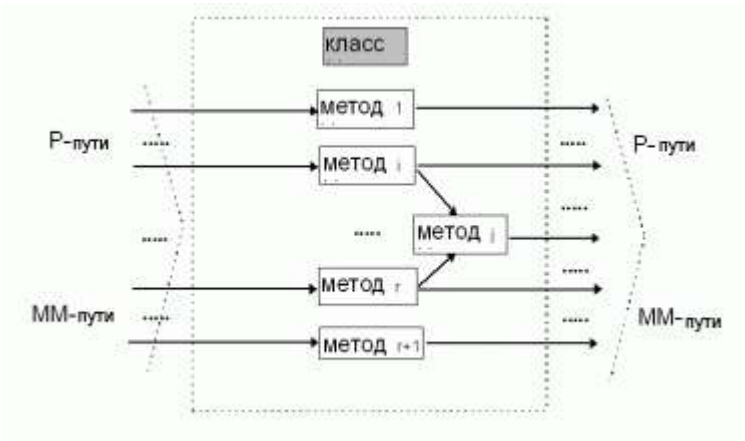

Рис. 10.4. Модель класса, входящего в программный проект

Методика проведения тестирования программы, представленной в виде *классовой модели* программного проекта, включает в себя несколько этапов, соответствующих уровням тестирования рис. 10.5:

1. На первом уровне проводится тестирование методов каждого класса программы, что соответствует этапу модульного тестирования.

2. На втором уровне тестируются методы класса, которые образуют контекст *интеграционного тестирования* каждого класса.

3. На третьем уровне протестированный класс включается в общий контекст (дерево классов) программного проекта. Здесь становится возможным отслеживать реакцию программы на внешние события

Второй и третий уровни рассматриваемой модели соответствуют этапу *интеграционного тестирования*.

Для третьего уровня важным оказывается понятие атомарной системной функции (АСФ) [23]. АСФ - это множество, состоящее из внешнего события на входе системы, реакции системы на это событие в виде одного или более *ММ-путей* и внешнего события на выходе системы. В общем случае внешнее выходное событие может быть нулевым, т. е. неаккуратно написанное программное обеспечение может не обеспечивать внешней реакции на действия пользователя. АСФ, состоящая из входного внешнего события, одного *ММ-пути* и выходного внешнего события, может быть взята в качестве модели для нити (thread). Тестирование подобной АСФ в рамках *классовой модели* ГМП реализуется довольно сложно, так как хотя динамическое взаимодействие нитей (потоков) в процессе исполнения естественно фиксируется в log-файлах, запоминающих результаты трассировки исполнения программ, оно же достаточно сложно отображается на классовой ГМП. Причина в том, что *классовая модель* ориентирована на отображение статических характеристик проекта, а в данном случае требуется отображение поведенческих характеристик. Как правило, тестирование взаимодействия нитей в ходе исполнения программного комплекса выносится на уровень *системного тестирования* и использует другие более приспособленные для описания поведения модели. Например, описание поведения программного комплекса средствами *языков спецификаций* MSC, SDL, UML.

Явный учет границ между интеграционным и системным уровнями тестирования дает преимущество при планировании работ на *фазе тестирования*, а возможность сочетать различные методы и критерии тестирования в ходе работы над программным проектом дает наилучшие результаты [24].

Объектно-ориентированный подход, ставший в настоящее время неявным стандартом разработки программных комплексов, позволяет широко использовать *иерархическую модель* программного проекта, приведенная на рис. 10.5 схема иллюстрирует способ применения. Каждый класс рассматривается как объект модульного и *интеграционного тестирования*. Сначала каждый метод класса тестируется как модуль по выбранному критерию С. Затем класс становится объектом *интеграционного тестирования*. Далее осуществляется интеграция всех методов всех классов в единую структуру - *классовую модель* проекта, где в общую ГМП протестированные модули входят в виде узлов (интерфейсов вызова) без учета их внутренней структуры, а их детальные описания образуют контекст всего программного проекта.

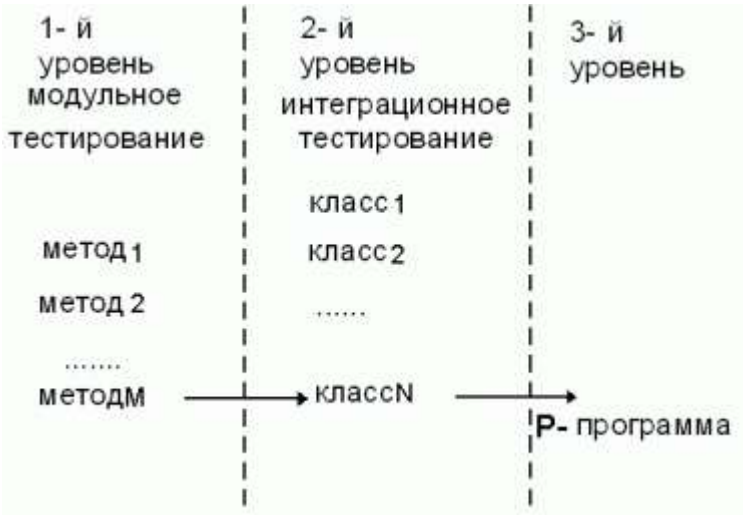

Рис. 10.5. Уровни тестирования классовой модели программного проекта

Сама технология объектно-ориентированного программирования (одним из определяющих принципов которой является инкапсуляция с возможностью ограничения доступа к данным и методам - членам класса) позволяет применить подобную трактовку вхождения модулей в общую ГМП. При этом тесты для отдельно рассмотренных классов переиспользуются, входя в общий набор тестов для программы Р.

Задание

1. Проанализировать программу, разработанную на практическом занятии № 6

2. Отобразить Дерево классов проекта

3. Отобразить Модель класса, входящего в программный проект

4. Отладить программу

5. Разработать тесты программного продукта рассмотренными методами.

6. Записать тесты, которые позволят пройти по ветвям классовой модели.

7. Протестировать разработанную Вами программу. Результаты оформить в виде таблиц

Тест	Ожидаемый результат	Фактический результат	Результат тестирования

8. Оформить отчет по лабораторной работе.

4 Содержание отчета

1. Цель работы.
2. Программа решения поставленной Вам задачи.

3. Схема программы (см. пп.2,3).

4. Таблицы тестирования программы (п.7).

6. Выводы по результатам тестирования (целью тестирования является обнаружение ошибок в программе).