Практическое занятие № 23 «Интеграция системы»,

Практическое занятие № 24 «Оптимизация программной системы»,

Практическое занятие № 25 «Применение инструментальных средств интегрального и системного тестирования»

Цель занятия. Проанализировать методы тестирования при интеграции программных модулей

Теоретические сведения

Интеграция модулей

Вторым по важности аспектом тестирования (после проектирования тестов) является последовательность слияния всех модулей в систему или программу. Выбор этой последовательности (должен приниматься на уровне проекта и на ранней стадии) определяет форму, в которую записываются тесты, типы необходимых инструментов тестирования, последовательность программирования модулей, тщательность и экономичность всего этапа тестирования.

Существует несколько подходов, которые могут быть использованы для слияния модулей в более крупные единицы. В большинстве своем они могут рассматриваться как варианты шести основных подходов, описанных ниже.

Методы тестирования

Большая трудоемкость тестирования и ограниченные ресурсы приводят к необходимости систематизации процесса и методов тес­тирования. В *практике тестирования используются следующие по­следовательно применяемые методы:* ***статический, детерминиро­ванный, стохастический и в реальном масштабе времени.***

*Статическое тестирование - проводится без использова­ния ЭВМ путем просмотра текста программы после трансляции, проверки правил структурного построения программ и обработки данных. В качестве эталонов используются, во-первых, внутренние спецификации, а, во-вторых, коллективный опыт специалистов-тестировщиков. Применение статического тестирования достаточ­но эффективно. Для типичных программ, по данным фирмы IBM, можно находить от 30 % до 80 % ошибок логического проектирования и кодирования. Этот метод способствует существенному повыше­нию производительности и надежности программ, позволяет раньше обнаружить ошибки, а значит уменьшить стоимость ис­правления.*

*Детерминированное тестирование - это многократное выполнение программы на ЭВМ с использованием определенных, специальным образом подобранных тестовых наборов данных. При детерминированном тестировании контролируются каждая комбинация исходных данных и соответствующие результаты, а также каждое утверждение в спецификации тестируемой програм­мы. Этот метод наиболее трудоемкий, поэтому детерминирован­ное тестирование применяется для отдельных модулей в процессе сборки программы или для небольших и несложных программных комплексов.*

*Стохастическое тестирование предполагает использова­ние в качестве исходных данных множество случайных величин с соответствующими распределениями. Для сравнения полученных результатов используются также распределения случайных вели­чин. Стохастическое тестирование применяется в основном для обнаружения ошибок, а для диагностики и локализации ошибок приходится переходить к детерминированному тестированию с использованием конкретных значений исходных данных, из облас­ти изменения ранее использовавшихся случайных величин. Сто­хастическое тестирование наилучшим образом подвергается ав­томатизации путем использования датчиков случайных чисел (генераторов случайных величин) и применяется для комплексно­го тестирования ППП.*

*Тестирование в реальном масштабе времени осуществ­ляется для ППП, предназначенных для работы в системах реаль­ного времени. В процессе такого тестирования проверяются ре­зультаты обработки исходных данных с учетом времени их поступления, длительности и приоритетности обработки, динамики использования памяти и взаимодействия с другими программами. При обнаружении отклонения результатов выполнения программ от ожидаемых для локализации ошибок, приходится фиксировать время и переходить к детерминированному тестированию.*

Каждый из рассмотренных методов тестирования не исключа­ет применения другого метода, скорее наоборот, требование к по­вышению качества ППП предполагает необходимость подвергать их различным методам тестирования и их сочетаниям, в зависимо­сти от сложности ППП и области его применения.

**Восходящее тестирование**

При восходящем подходе программа собирается и тестируется снизу вверх. Только модули самого нижнего уровня («терминальные» модули; модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершаются и тестирование модулей, и тестирование сопряжений программы.

При восходящем тестировании для каждого модуля необходим *драйвер:* нужно подавать тесты в соответствии с сопряжением тестируемого модуля. Одно из возможных решений - написать для каждого модуля небольшую ведущую программу. Тестовые данные представляются как «встроенные» в эту программу переменные и структуры данных, и она многократно вызывает тестируемый модуль, с каждым вызовом передавая ему новые тестовые данные. Имеется и лучшее решение: воспользоваться программой тестирования модулей - это инструмент тестирования, позволяющий описывать тесты на специальном языке и избавляющий от необходимости писать драйверы.

**Нисходящее тестирование**

При нисходящем подходе программа собирается и тестируется сверху вниз. Изолировано тестируется только головной модуль. После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

При этом подходе возникают два вопроса: что делать, когда тестируемый модуль вызывает модуль более низкого уровня (которого в данный момент еще не существует), и как подаются тестовые данные. Ответ на первый вопрос состоит в том, что для имитации функций недостающих модулей программируются *модули-заглушки,* которые моделируют функции отсутствующих модулей. Фраза «напишите заглушку» часто встречается в описании этого подхода, но она способна ввести в заблуждение, поскольку задача написания «заглушки» может оказаться трудной. Ведь заглушка редко сводится просто к оператору RETURN, поскольку вызывающий модуль обычно ожидает от нее выходных параметров. В таких случаях в заглушку встраивают фиксированные выходные данные, которые она всегда и возвращает. Иногда это оказывается неприемлемым, так как вызывающий модуль может рассчитывать, что результат вызова зависит от входных данных. Поэтому в некоторых случаях заглушка должна быть довольно изощ­ренной, приближаясь по сложности к модулю, который она пытается моделировать.

Интересен и второй вопрос: в какой форме готовятся тестовые данные и как они передаются программе? Если бы головной модуль содержал все нужные операции ввода и вывода, ответ был бы простым: тесты пишутся в виде обычных для пользователей внешних данных и передаются программе через выделенные ей устройства ввода. Но так случается редко. В хорошо спроектированной программе физические операции ввода-вывода выполняются на нижних уровнях структуры, поскольку физический ввод-вывод - это абстракция довольно низкого уровня. Поэтому для того, чтобы решить проблему экономически эффективно, модули добавляются не в строго нисходящей последовательности (все модули одного горизонтального уровня, затем модули следующего уровня), а таким образом, чтобы обеспечить функционирование операций физического ввода-вывода как можно быстрее*.* Когда эта цель достигнута, нисходящее тестирование получает значительное преимущество: все дальнейшие тесты готовятся в той же форме, которая рассчитана на пользователя.

Нисходящий метод имеет как достоинства, так и недостатки, по сравнению с восходящим. Его достоинство заключается в том, что этот метод совмещает тестирование модуля, тестирова­ние сопряжений и частично тестирование внешних функций. С этим связано и другое его достоинство - когда модули ввода-вывода уже подключены, тесты можно готовить в удобном виде.

Нисходящий подход выгоден и в том случае, когда есть сомнения относительно осуществимостипрограммы в целом или если в проекте программы могут оказаться серьезные дефекты.

Преимуществом нисходящего подхода часто считают отсутствие необходимости в драйверах; вместо драйверов следует написать «заглушки». Однако это преимущество спорно.

Нисходящий метод тестирования имеет и недостатки. Основным из них является тот, что модуль редко тестируется досконально сразу после его подключения. Дело в том, что основательное тестирование некоторых модулей может потребовать крайне изощренных заглушек. Программист часто решает не тратить массу времени на их программирование, а вместо этого пишет простые заглушки и проверяет лишь часть условий в модуле. Он, конечно, собирается вернуться и закончить тестирование рассматриваемого модуля позже, когда уберет заглушки. Такой план тестирования не лучшее решение, поскольку об отложенных условиях часто забывают.

Второй недостаток нисходящего подхода состоит в том, что он может породить веру в возможность начать программирование и тестирование верхнего уровня программы до того, как вся программа будет полностью спроектирована. Эта идея на первый взгляд кажется экономичной, но обычно дело обстоит наоборот. Большинство опытных проектировщиков признает, что проектирование программы - процесс итеративный. Редко первый проект оказывается совершенным. Нормальный стиль проектирования структуры программы предполагает по окончании проектирования нижних уровней вернуться назад и подправить верхний уровень, внеся в него некоторые усовершенствования или исправляя ошибки, либо иногда даже закончить проект и начать все сначала, потому, что разработчик увидел лучший подход. Если же головная часть программы уже запрограммирована и оттестирована, то возникает серьезное сопротивление любым улучшениям ее структуры. В конечном итоге за счет таких улучшений обычно можно сэкономить больше, чем те несколько дней или недель, которые рассчитывает выиграть проектировщик, приступая к программированию слишком рано.

**Модифицированный нисходящий метод**

Применяя нисходящее тестирование в точном соответствии с предыдущим изложением, часто невозможно тестировать определенные логические условия, например ошибочные ситуации или защитные проверки. Нисходящий метод, кроме того, делает сложной или вообще невозможной проверку исключительных ситуаций в некотором модуле, если программа работает с ним лишь в ограниченном контексте (это означает, что модуль никогда не получит достаточно полный набор входных значений). Даже если тестирование такой ситуации в принципе осуществимо, часто бывает трудно определить, какие именно нужны тесты, если они вводятся в точке программы, удаленной от места проверки соответствующего условия.

Подход, называемый модифицированным нисходящим методом, решает эти проблемы: требуется, чтобы каждый модуль прошел автономное тестирование перед подключением к программе. Это решает перечисленные проблемы, но здесь требуются и драйверы, и заглушки для каждого модуля.

**Метод большого скачка**

Один из подходов к интеграции модулей - это метод большого скачка. В соответствии с этим методом каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу.

Метод большого скачка по сравнению с другими подходами имеет много недостатков и мало достоинств.

Заглушки и драйверы необходимы для каждого модуля. Модули не интегрируются до самого последнего момента, а это означает, что в течение долгого времени серьезные ошибки в сопряжениях могут остаться необнаруженными.

Метод большого скачка значительно усложняет отладку.

Если программа мала и хорошо спроек­тирована, метод большого скачка может оказаться приемлемым. Однако для крупных программ он обычно неприемлем.

**Метод сандвича**

Тестирование методом сандвича - это компромисс между восходящим и нисходящим подходами; попытка воспользоваться достоинствами обоих методов, избежав их недостатков.

При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу как снизу, так и сверху и встречаясь, где-то в середине. Точка встречи зависит от конкретной тестируемой программы и должна быть заранее определена при изучении ее структуры. Например, если разработчик может представить свою систему в виде уровня прикладных модулей, затем уровня модулей обработки запросов, затем уровня примитивных функций, то он может решить применять нисходящий метод на уровне прикладных модулей (программируя заглушки вместо модулей обработки запросов), а на остальных уровнях применить восходящий метод.

Применение метода сандвича - это разумный подход к интеграции больших программ, таких, как операционная система или пакет прикладных программ.

Метод сандвича сохраняет такое достоинство нисходящего и восходящего подходов, как начало интеграции системы на самом раннем этапе. Поскольку вершина программы вступает в строй рано, как в нисходящем методе, уже на раннем этапе получается работающий каркас программы. Поскольку нижние уровни программы создаются восходящим методом, то снимаются те проблемы нисходящего метода, которые были связаны с невозможностью тестировать некоторые условия в глубине программы.

**Модифицированный метод сандвича**

При тестировании методом сандвича возникает та же проблема, что и при нисходящем подходе. Проблема эта заключается в том, что невозможно досконально тестировать отдельные модули. Восходящий этап тестирования по методу сандвича решает эту проблему для модулей нижних уровней, но она может по-прежнему оставаться открытой для нижней половины верхней части программы. В модифицированном методе сандвича нижние уровни также тестируются строго снизу вверх. А модули верхних уровней сначала тестируются изолированно, а затем собираются нисходящим методом.

Таким образом, модифицированный метод сандвича тоже представляет собой компромисс между восходящим и нисходящим подходами.

Сравнение методов тестирования

С точки зрения надежности ПО стратегии тестирования можно оценить по семи критериям (табл. 9).



*Первый критерий - время до момента сборки модулей, поскольку это важно для обнаружения ошибок в сопряжениях и предположениях модулей о свойствах друг друга.*

*Второй критерий - время до момента создания первых работающих «скелетных» версий программы, поскольку здесь могут проявиться главные дефекты проектирования.*

*Третий и четвертый критерии касаются вопроса о том, необходимы ли заглушки, драйверы и другие инструменты тестирования.*

*Пятый критерий - мера параллелизма, который возможен в начале или на ранних стадиях тестирования (но не концу цикла тестирования).*

*Шестой критерий связан с ответом на вопрос: возможно ли проверить любой конкретный путь и любое условие в программе?*

*Седьмой критерий характеризует сложность планирования, надзора и управления в процессе тестирования.*

Оценим шесть подходов тестирования с помощью перечисленных критериев. В качестве исходного приближения для вы­полнения оценок приведен вариант грубой оценки. Прежде всего, следует взвесить относительное влияние каждого критерия на надежность программного обеспечения.

Ранняя сборка и раннее получение работающего каркаса программы, а также возможность тестировать любые конкретные условия, представляются наиболее важными, им дается вес 3.

Сложность подготовки заглушек, планирования и управления последовательностью тестов также важны, они получают вес 2.

Третий критерий, необходимость драйверов - вес 1 ввиду доступности общих инструментов тестирования.

Критерий, связанный с параллелизмом работы, имеет вес 1 (он может быть важен по другим причинам, но на надежность сильно не влияет).

Шестой критерий - вес 3.

Седьмой критерий получает вес 2.

В табл. 10 приведены результаты этой оценки. В каждой графе таблицы вес берется со знаком плюс или минус либо не учитывается, в зависимости от того, благоприятно, неблагоприятно или безразлично проявляется соответствующий фактор при рассматриваемом подходе. Модифицированный метод сандвича и восходящий метод оказываются наилучшими подходами, а метод большого скачка - наихудшим. Если способ оценки оказывается близким к вашей конкретной ситуации, следует рекомендовать модифицированный метод сандвича для тестирования больших систем или программ и восходящий подход для тестирования программ малых и средних.



Этапы тестирования

*Процесс тестирования ППП начинается проверкой правильно­сти работы отдельных модулей и заканчивается приемкой после испытания ППП при его сдаче заказчику или началом коммерче­ских продаж ППП. Рассмотрим типичные этапы работы тестировщиков.*

*Тестирование программных модулей - наиболее форма­лизованный и автоматизированный процесс тестирования.*

Основ­ная задача тестирования состоит в проверке обработки программными модулями поступающей информации и корректности, получающихся на выхо­де данных в соответствии с функциями, отраженными в специфи­кациях.

Проверяется корректность структуры модулей и их конструктивных основных компонентов: процедур, циклов, блоков, условий и т.д.

Тестирование планируется с учетом структуры модулей и особенностей обработки информации и осуществляется преиму­щественно детерминировано.

*Тестирование функциональных групп модулей предна­значено для проверки корректности решения крупных автономных Функциональных задач ППП. Проверяется правильность управ­ляющих и информационных связей между модулями, а также кор­ректность вычислений в процессе обработки информации. Значи­тельно возрастают сложность тестируемых объектов и соответственно объем тестов. Вследствие этого возрастают требования к автоматизации тестирования и затраты на его выполне­ние.*

Детерминированным тестированием проверяются структура групп программ и основные маршруты обработки информации. В ряде случаев результаты получаются методами стохастического тестирования. Эти методы пока слабо формализованы, и их при­менение в значительной степени зависит от конкретных функций тестируемой группы программ.

*Комплексное тестирование - сложный процесс, в котором завершается проверка корректности функционирования программ при правильных исходных данных, и осуществляются основные проверки при искажениях на входе.*

Проверяются надежность функционирования всего ППП в реальных условиях, эффектив­ность средств программой защиты и восстановления. Определя­ются корректность использования программами ресурсов компью­тера и функционирование программ в критических условиях. Формализация процесса тестирования на этом этапе наиболее трудна, и оценка полноты тестирования осуществляется преиму­щественно по степени выполнения функций и по характеристикам надежности функционирования ППП. Для этого применяются пре­имущественно стохастическое тестирование и тестирование в ре­альном времени.

Прежде чем приступить к тестированию программного ком­плекса в целом, нужно, чтобы составляющие его части (отдельные модули или функциональные группы модулей) были тщательно от­тестированы.

Сборка модулей в программный комплекс может осуществляться двумя методами: *монолитным и пошаговым.*

По­шаговая сборка может, в свою очередь, быть *восходящей (снизу-вверх) и нисходящей (сверху-вниз).*

В качестве примера рассмотрим программный пакет, состоя­щий из девяти модулей (рис. 1).

Монолитный метод сборки предполагает тестирование каж­дого из девяти модулей отдельно, а затем их одновременную сборку и тестирование в комплексе.

Для автономного тестирования каждого модуля нужен модуль-драйвер, обеспечивающий вызов и передачу тестируемому модулю необходимых входных данных и обработку результатов, и один или несколько модулей-заглушек, имитирующих функции модулей, вы­зываемых тестируемым модулем.

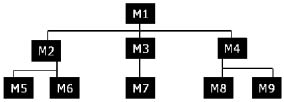


Рисунок 1– Структура пакета из 9 модулей

Для рассматриваемого примера модули-драйверы нужны для всех модулей, кроме модуля М1, а модули-заглушки нужны для всех модулей, кроме М5, М6, М7, М8, М9 (т.е. модулей самого низ­шего уровня).

Таким образом, при монолитной сборке необходимо разрабо­тать восемь модулей-драйверов и минимум девять модулей-заглушек.

Пошаговый метод предполагает, что модули тестируются не автономно, а последовательно подключаются к набору уже оттес­тированных ранее модулей.

Предположим, что тестируем сверху-вниз. Тогда для модуля М1 нужно разработать три заглушки. Далее подключается модуль М2, для которого нужно предварительно разработать две заглушки, и тес­тируются модули М1-М2. Затем заглушка М5 заменяется модулем М5 и тестируется цепочка М1-М2-М5.

Процесс продолжается до тех пор, пока не будет собран весь комплекс. Есть возможность неко­торого распараллеливания работ и автономного тестирования це­почек М1-М2-М5 (М6), М1-М3-М7, М1-М4-М8 (М9).

Оказывается, что при пошаговой сборке сверху-вниз нужно разработать девять заглу­шек, но не нужны драйверы.

При тестировании снизу-вверх процесс организуется следую­щим образом: тестируются модули низшего уровня - М5, М6, М7, М8, М9. Для каждого из них нужен драйвер.

Далее параллельно проводится тестирование М5-М2, М6-М2, М7-М3, М8-М4, М9-М4. Затем подключается модуль М1 и проводится комплексное тес­тирование всего пакета.

Таким образом, при восходящем тестиро­вании нужно будет разработать максимум восемь драйверов, но заглушки не потребуются.

Сравнивая монолитную и пошаговую сборки программ, можно отметить ряд достоинств и недостатков каждого из них.

Монолитная сборка требует больших затрат, так как предпола­гает дополнительно разработку драйверов и заглушек, в то время как при пошаговой сборке разрабатываются либо только заглушки, либо только драйверы.

При пошаговом тестировании раньше обнаруживаются ошибки в интерфейсах между модулями, поскольку раньше начинается сборка программы. При монолитном методе модули «не видят друг друга» до последней фазы. Но безусловным его преимуществом является большая возможность распараллеливания работ.

Системное тестирование (или испытание программного комплекса) предназначено в основном для проверки соответствия пакета прикладных программ техническому заданию и для оценки его пригодности к регулярной эксплуатации и сопровождению.

Для этого проверяются полнота и точность технической документации, качество функционирования пакета прикладных программ по всем требованиям технического задания.

Проверка пригодности к сопровождению включает тестирование настройки версий на условия конкретного применения и анализ удобства модифицирования версий пакета прикладных программ.

Если речь идет о заказном пакете прикладных программ, то при положительных результатах испытаний оформляется акт приемки пакета прикладных программ в опытную или промышленную эксплуатацию, и работа считается завершенной*.*

Несколько иначе организуются испытания коммерческих паке­тов прикладных программ, создаваемых по инициативе разработ­чиков для широкого круга пользователей при отсутствии конкретно­го заказчика.

Для таких коммерческих прикладных программ принято проводить испытания в два последовательных этапа, на­зываемых альфа- и бета-тестированием.

Эти испытания заключаются в нормальной и форсированной (стрессовой) опытной эксплуатации конечными пользователями программного продукта, в соответствии с сопроводительной документацией, и различаются количеством участвующих пользователей.

При альфа-тестировании привлекаются конечные пользо­ватели, работающие в той же компании, но не участвовавшие не­посредственно в разработке комплекса программ.

Для бета-тестирования привлекаются добровольные пользователи (потен­циальные покупатели), которым бесплатно передается версия пакета прикладных программ для опытной эксплуатации. При этом особое значение имеет выделение компетентных и доброжелательных пользователей, способ­ных своими рекомендациями улучшить качество испытываемых (например, пяти) программ. Их деятельность стимулируется бесплатным и ранним получением и освоением нового программного продукта, собственной оценкой его качества. Эти пользователи обязуются сообщать разработчикам сведения обо всех выявленных дефектах и ошибках, а также вносить изменения в программы и данные или заменять вер­сии по указаниям разработчиков.

Только после успешной эксплуатации и бета-тестирования ограниченным контингентом пользова­телей, руководителем проекта или фирмы-разработчика принима­ется решение о передаче пакета прикладных программ в продажу для широкого круга поль­зователей.

В последнее время фирмы-разработчики стали выкладывать бета-версии программ на свои Web-сайты, позволяя всем желаю­щим скачивать их, тем самым, увеличивая контингент доброволь­ных тестировщиков. Обратная связь организуется либо через сис­тему конференций на сайте разработчика, либо посредством электронной почты.