# восходящее тестирование программного обеспечения

Подготовила студентка
группы БПИ-31
Макарова Ирина

# История

- История тестирования программного обеспечения отражает эволюцию разработки самого программного обеспечения. В течение длительного времени разработка программного обеспечения уделяла основное внимание крупномасштабным научным программам, а также программам министерства обороны, связанным с системами корпоративных баз данных, которые проектировались на базе универсальной ЭВМ или миникомпьютера. Тестовые сценарии записывались на бумагу. С их помощью проверялись целевые потоки управления, вычисления сложных алгоритмов и манипулирование данными. Окончательный набор тестовых процедур мог эффективно протестировать всю систему полностью. Тестирование обычно начиналось лишь после завершения плана-графика проекта и выполнялось тем же персоналом.

# Определения

- Тестирование -- процесс выполнения программы с намерением найти ошибки.
- Тестирование ПО - это процесс его исследования с целью получения информации о качестве. Целью тестирования является выявление дефектов в ПО. С помощью тестирования нельзя доказать отсутствие дефектов и корректность функционирования анализируемой программы. Тестирование сложных программных продуктов является творческим процессом, не сводящимся к следованию строгим и четким процедурам.

- Тестирование программного обеспечения охватывает целый ряд видов деятельности, весьма аналогичный последовательности процессов разработки программного обеспечения. Сюда входят постановка задачи для теста, проектирование, написание тестов, тестирование тестов и, наконец, выполнение тестов и изучение результатов тестирования. Решающую роль играет проектирование теста.

# 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 сопряжении программы.

- При восходящем тестировании для каждого модуля необходим драйвер: нужно подавать тесты в соответствии с сопряжением тестируемого модуля. Одно из возможных решении -- написать для каждого модуля небольшую ведущую программу. Тестовые данные представляются как «встроенные» непосредственно 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.
- Здесь отсутствуют проблемы, связанные с невозможностью или трудностью создания всех тестовых ситуаций, характерные для нисходящего тестирования. Драйвер как средство тестирования применяется непосредственно к тому модулю, который тестируется, где нет промежуточных модулей, которые следует принимать во внимание. Анализируя другие проблемы, возникающие при нисходящем тестировании, можно заметить, что при восходящем тестировании невозможно принять неразумное решение о совмещении тестирования с проектированием программы, поскольку нельзя начать тестирование до тех пор, пока не спроектированы модули нижнего уровня. Не существует также и трудностей с незавершенностью тестирования одного модуля при переходе к тестированию другого, потому что при' восходящем тестировании с применением нескольких версий заглушки нет сложностей с представлением тестовых данных.

# Что это такое?

- Восходящее тестирование - это прекрасный способ локализации ошибок. Если ошибка обнаружена при тестировании единственного модуля, то очевидно, что она содержится именно в нем - для поиска ее источника не нужно анализировать код всей системы. А если ошибка проявляется при совместной работе двух предварительно протестированных модулей, значит, дело в их интерфейсе. Еще одним преимуществом восходящего тестирования является то, что выполняющий его программист концентрируется на очень узкой области (единственном модуле, передаче данных между парой модулей и т.п.). Благодаря этому тестирование проводится более тщательно и с большей вероятностью выявляет ошибки.

# Недостатки

- Главным недостатком восходящего тестирования является необходимое написания специального кода-оболочки, вызывающего тестируемый модуль. Если он, в свою очередь, вызывает другой модуль, для него нужно написать «заглушку». Заглушка - это имитация вызываемой функции, возвращающая те же данные, но ничего больше не делающая.
- Понятно, что написание оболочек и заглушек замедляет работу, а для конечного продукта они абсолютно бесполезны. Хороший набор оболочек и заглушек - это очень эффективный инструмент тестирования.
- В противоположность восходящему тестированию, стратегия целостного тестирования предполагает, что до полной интеграции системы ее отдельные модули не проходят особо тщательного тестирования.

# Преимущество

- Преимуществом такой стратегии является то, что нет необходимости в написании дополнительного кода. Поэтому многие руководители выбирают этот способ из соображений экономии времени - они считают, что лучше разработать один обширный набор тестов и с его помощью за один раз проверить всю систему. Но такое представление совершенно ошибочно, и вот почему.
- Очень трудно выявить источник ошибки. Это главная проблема. Поскольку ни один из модулей не проверен как следует, в большинстве из них есть ошибки. Получается, что вопрос не столько в том, в каком модуле произошла обнаруженная ошибка, сколько в том, какая из ошибок во всех вовлеченных в процесс модулях привела к полученному результату. И когда накладываются ошибки нескольких модулей, ситуацию может быть гораздо труднее локализовать и повторить.

- Кроме того, ошибка в одном из модулей может блокировать тестирование другого. Как протестировать функцию, если вызывающий ее модуль не работает? Если не написать для этой функции программу-оболочку, придется ждать отладки модуля, а это может затянуться надолго.
- Трудно организовать исправление ошибок. Если программу пишут несколько программистов (а именно так и бывает в больших системах), и при этом неизвестно, в каком модуле ошибка, кто же будет ее искать и исправлять? Один программист будет указывать на другого, тот, выяснив, что его код ни при чем, снова обратится к первому, а в результате будет сильно страдать скорость разработки.
- Процесс тестирования плохо автоматизирован. То, что на первый взгляд кажется преимуществом целостного тестирования - отсутствие необходимости писать оболочки и заглушки, - на самом деле оборачивается его недостатком. В процессе разработки программа ежедневно меняется, и ее приходится тестировать снова и снова. А оболочки и заглушки помогают автоматизировать этот однообразный труд.

# Нисходящее тестирование

- Существует и еще один принцип организации тестирования, при котором программа так же, как и при восходящем способе, тестируется не целиком, а по частям. Только направление движения меняется - сначала тестируется самый верхний уровень иерархии модулей, а от него тестировщик постепенно спускается вниз. Такая технология называется нисходящей (top-down). Обе технологии - и нисходящую и восходящую - называют также инкрементальными.
- При нисходящем тестировании отпадает необходимость в написании оболочек, но заглушки остаются. По мере тестирования заглушки по очереди заменяются на реальные модули.
- Мнения специалистов о том, какая из двух инкрементальных стратегий тестирования более эффективна, сильно расходятся. На практике вопрос выбора стратегии тестирования обычно решается просто: каждый модуль по возможности тестируется сразу после его написания, в результате последовательность тестирования одних частей программы может оказаться восходящей, а других - нисходящей

# Определение

- Нисходящее тестирование (называемое также нисходящей разработкой не является полной противоположностью восходящему, но в первом приближении может рассматриваться как таковое, 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 немедленно возникает 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 «просто 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. Это иногда 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 прост:
- Тесты пишутся в виде обычных для пользователей внешних данных и передаются программе через выделенные ей устройства ввода. Так, однако, случается редко. В хорошо спроектированной программе физические операции ввода-вывода выполняются на нижних уровнях структуры, поскольку физический ввод-вывод -- 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.
- Остается еще один вопрос: в какой форме пишутся тесты до тех пор, пока не будет достигнута эта цель? Ответ: они включаются в некоторые из заглушек.

# Достоинства

- Нисходящий метод имеет как достоинства, так и недостатки по сравнению с восходящим. Самое значительное достоинство -- в том, что этот метод совмещает тестирование модуля, тестирование сопряжении и частично тестирование внешних функций. С этим же связано другое его достоинство -- когда модули ввода-вывода уже подключены, тесты можно готовить в удобном виде. Нисходящий подход выгоден также в том случае, когда есть сомнения относительно осуществимости программы в целом или если в проекте программы могут оказаться серьезные дефекты.
- Преимуществом нисходящего подхода очень часто считают отсутствие необходимости в драйверах; вместо драйверов вам просто следует написать «заглушки». Как читатель сейчас уже, вероятно, понимает, это преимущество спорно.

# Недостатки

- Нисходящий метод тестирования имеет, к сожалению, некоторые 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 определенно не лучшее решение, поскольку об отложенных условиях часто забывают.

# Недостатки

- Второй тонкий 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 совсем 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 выбросить проект и начать все сначала, потому что разработчик внезапно 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.

# Технология восходящего и нисходящего тестирования

- При восходящем тестировании, применяемом обычно для небольших программ, сначала тестируют отдельные периферийные блоки, а затем переходят к тестированию центральной части, которая, разумеется, взаимодействует только с отлаженными уже блоками.
- При нисходящем тестировании, используемом для достаточно больших программ, параллельно с контролем периферийных блоков (или даже до начала их контроля) производится и контроль центрального блока, выполняемого на компьютере совместно с имитаторами периферийных блоков, называемых заглушками. В задачу имитаторов входит моделирование работы соответствующих блоков с целью поддержать функционирование центрального блока. Обычно заглушки выдают простейший результат, например константу и сообщение о факте своего участия в работе. Вместо постоянной величины на наиболее поздней стадии тестирования может выдаваться и случайная величина в требуемом диапазоне.

Спасибо за внимание!