Лекция 8. Программный инструментарий, автоматизированные средства тестирования ПП

Подготовка новых версий программного обеспечения в короткие сроки и без снижения качества требует соответствующего (быстрого) тестирования. Проблему ускорения тестирования и увеличения тестового покрытия позволяет решить автоматизация этого процесса.

8.1. Целесообразность автоматизации тестирования

Автоматизированное тестирование – это набор техник, подходов и инструментальных средств, позволяющий возложить на машину ряд обязанностей тестировщика, таких как запуск и выполнение тест-кейсов, составление отчета по результатам тестирования.

При создании тест-кейсов для автоматизированного тестирования не стоит уповать на интуитивную понятливость компьютера: все описания должны быть предельно чёткими. В частности, для ожидаемого результата следует указать конкретные признаки его корректности. Например, для загружаемой страницы привести ее название, перечислить поля и другие элементы, которые должны на ней присутствовать. В тест-кейсе следует избегать ссылок на специфические компоненты конкретного инструмента, аппаратной или программной платформы и, по возможности, использовать наиболее универсальные способы взаимодействия с тестируемым приложением (например, лучше «Открыть главную страницу приложения», чем «Открыть http://application/.1...»). Также автоматизированные тест-кейсы должны быть независимыми (не ссылаться на предыдущие или последующие тесты).

Тест-кейс, записанный на языке сценариев (scripting language), называется **скриптом.** Это набор инструкций для автоматической проверки. Фактически это программа, и при ее записи следует придерживаться стандартов кодирования. Тестовые скрипты для нагрузочного тестирования рекомендуется разделять на 3 части: precondition, test case action, postcondition, (инициализация, рабочая часть скрипта. корректное завершение) т.к. разные части тестов имеют разную частоту выполнения.

Целью автоматизации является повышение эффективности процесса тестирования: и в смысле уменьшения затрат за счёт высвобождения специалистов, и в смысле повышения качества ПП за счёт увеличения тестового покрытия и эффективности выявления ошибок.

Преимущества автоматизации:

- Исключение «человеческого фактора», т.е. низкой квалификации, невнимательности, усталости и т.п. тестировщика: никто не застрахован от ошибок. Автоматизированная система строго следует инструкциям и ничего не напутает.
- Высокая скорость тестирования: возможности машины превосходят возможности человека.
- Способность средств автоматизации выполнить непосильные для человека тест-кейсы и низкоуровневые действия с приложением, операционной системой, каналами передачи данных и т.д.
- Минимизация затрат на поддержку тест-кейсов и их выполнение без вмешательства человека (достаточно широко распространено, например, ночное тестирование).

— Возможность сбора, анализа и представления в удобной для восприятия форме значительных объёмов данных, автоматической рассылки и сохранения отчетов о результатах тестирования.

Однако, следует понимать, что автоматизация не происходит сама по себе, и более того, она влечет за собой серию **проблем** [8].

Во-первых, это необходимость высокой квалификации персонала. Автоматизация часто рассматривается как проект внутри проекта (есть требования, планы, методы, инструменты, код и т. д.). Развитие этого направления методом проб и ошибок может отрицательно сказаться на бюджете, длительности и самой идее этого процесса. Привлечение специалистов поможет автоматизировать процесс тестирования на профессиональном уровне и в сжатые сроки. Но и далее техническая квалификация сотрудников, обеспечивающих автоматизацию тестирования, должна быть существенно выше, чем у занимающихся ручным тестированием.

Во-вторых, одной из главных **проблем** автоматизированного тестирования считается его трудоемкость. Значительные ресурсы могут тратиться на:

— разработку тестового каркаса для конкретного проекта. Можно говорить о разработке приложения для тестирования создаваемого ПП;
— поддержку тест-кейсов. Изменение требований, смена технологического домена, доработка функционала, интерфейсов (пользовательских и программных) ведёт к частичной или полной непригодности тест-скриптов и необходимости их актуализации. Чем чаще изменения, тем больше работы; Изменение кода тестов может занять столько же времени, сколько и изменение основного кода
— приобретение и поддержку инструмента для автоматизации. Скромный функционал свободно распространяемых инструментов вряд ли подойдет для серьёзных проектов, а лицензионное ПО имеет обычно высокую стоимость. Разнообразие средств автоматизации усложняет проблему выбора инструмента и может повлечь за собой финансовые риски. Также могут потребоваться затраты на обучение персонала работе с приобретённым средством автоматизации.

В третьих, проблемой является и пропуск ошибок, на проверку которых тест-скрипт не рассчитан. Но этот дефект может найти тестировщик в ходе ручного тестирования, обратив внимание на некоторые детали.

Решение о целесообразности автоматизации тестирования принимают, взвесив все «за» и «против». При этом следует понимать, что автоматизация также требует более тщательного планирования и управления рисками и хорошей организации и документирования процесса контроля качества в проектах. Также не стоит забывать, что эффект от автоматизации наступает не сразу и не всегда, что принятие неверных решений влечет за собой финансовые потери.

Автоматизации **подлежат** различные **виды тестирования.**

Первым пунктом в этом списке стоит **нагрузочное** тестирование (тестирование производительности). Средства автоматизации позволяют создавать нагрузку с интенсивностью, недоступной человеку, собирать и анализировать большой объём данных о параметрах работы приложения. Без автоматизации выполнение тестирования производительности трудно представить. Как следствие – широкий выбор и высокая стоимость инструментов для нагрузочного тестирования от разных производителей.

Следом идёт **регрессионное** тестирование. ПО **регулярно** проверяют на корректность функционирования после внесения в него изменений. При этом число тестов, которые надо выполнить неуклонно растёт от версии к версии.

Конфигурационное тестирование – направлено на проверку работоспособности компонентов архитектуры в различном окружении, обычно указанном в требованиях. Аналогичные задачи решает тестирование совместимости. В обоих случаях необходимо выполнение одних и тех же тест-кейсов на большом множестве входных данных, под разными платформами и в разных условиях.

Очень большого количества проверок требуют также тестирование **безопасности** (права доступа, пароли по умолчанию, открытые порты, уязвимости и т. д.), **инсталляции** (множество рутинных операций по проверке работы инсталлятора, размещения файлов в файловой системе, содержимого конфигурационных файлов, реестра и т. д.), **модульное** тестирование и некоторые другие.

Автоматизация целесообразна для длительных, рутинных, требующих повышенного внимания операции (например, проверить все 30000 ссылок на предмет того, что все они ведут на реально существующие страницы). И там, где тестированию подвергают процессы проходящие на уровнях более глубоких, чем пользовательский интерфейс (тестирование **интеграции**, приложений без графического интерфейса, вообще не предназначенных для взаимодействия с человеком компонентов)

Тестовые скрипты иногда составляют даже для одноразовых проверок нового функционала, если без автоматизации нельзя обойтись. Обычно, впоследствии эти тест-кейсы применяют для регрессионного тестирования.

Таким образом, инструменты для автоматизированного тестирования берут на себя решение рутинных задач и выполнение непосильных человеку тест-кейсов, увеличение тестового покрытия, ускорение тестирования и высвобождение времени тестировщиков для интеллектуальной работы.

С другой стороны, существуют перечень задач, где по-прежнему требуется человеческое мышление и участие. Это разработка тест-кейсов, анализ результатов тестирования, составление отчётов, тестирование удобства использования и доступности. Затраты на автоматизацию не окупятся, если тест-кейсы, достаточно выполнить всего несколько раз, если требования к ПП не стабильны.

8. 2. Технологические подходы к автоматизации тестирования

Автоматизированное тестирование может осуществляться на уровне кода и на уровне пользовательского интерфейса (GUI-тестирование).

Широкое распространение получило GUI-тестирование приложений. Оно осуществляется через графический интерфейс пользователя (имитируются действия пользователя с помощью специальных тестовых фреймворков) и не требует доступа к исходному коду.

Основными технологиями автоматизации тестирования являются:

1. Запись и воспроизведение. *Утилиты записи и воспроизведения* – первое поколение средств автоматизации тестирования. Инструмент фиксирует действия тестировщика во время ручного тестирования и может воспроизвести их в ходе проверки приложения. Достоинства – простота и высокая скорость создания тест-кейсов. Однако, требуется серьёзная доработка полученного кода, а при любых изменениях тестируемого ПП необходимо повторное создание сценариев тестирования.

2. Частные решения *(Написание сценария)*. Для решения каждой отдельной задачи тестирования пишется своя программа. Возможно, на специально разработанных для автоматизации тестирования скриптовых языках. Быстро и

просто. Но требуется привлечение программистов высокого уровня, созданные скрипты направлены преимущественно на GUI-тестирование, их сложно объединить в систему, много времени надо на поддержку. Почти невозможно повторное использование.

3. Тестирование под управлением данными (DDT). В тестовом скрипте обычно задается навигация по приложению, чтение источников данных, ведение журнала действий пользователя. Входные данные и ожидаемые результаты хранятся вне тест-кейса: в центральном хранилище или базе данных, csv или xls файлах и т. д. В результате один и тот же тест-кейс можно повторять многократно с разными данными. Логика выполнения тест-кейса зависит от значений переменных. Для изменения логики тест-кейс надо переписывать.

4. Тестирование под управлением ключевыми словами (KDT). Ключевые слова – это действия, которые следует выполнить в ходе тестирования. Теперь вне тест-кейса хранятся не только входные данные и ожидаемые результаты, но и особенности поведения. Соответственно и входные/выходные данные и поведение могут быть изменены без изменения кода тест-кейса. В результате, конечный тест представляет собой не программный код, а описание последовательности действий с их параметрами (например, "завести в базе данных пользователя с логином XXX и паролем YYY"). Это даёт возможность создавать тесты людям, не имеющим навыков программирования. В числе недостатков – сложность выполнения низкоуровневых операций.

5. Тестирование под управлением поведением (BDT). Развитие идей DDT и KDT. Акцент делается на бизнес-сценариях (высокоуровневых пользовательских сценариях) без выполнения мелких проверок. Естественно, что значительное число функциональных и нефункциональных дефектов останутся не замеченными. Поэтому такие тест-кейсы должны быть дополнены классическими низкоуровневыми.

6. Использование фреймворков. **Фреймворк** (каркас) – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. Содержит в себе большое количество разных по назначению библиотек. В тестировании ПП – это, фактически, конструктор, позволяющий использовать остальные подходы. Фреймворк обеспечивает программный интерфейс (API) управления процессом выполнения тест-кейсов. Можно отметить мощность, расширяемость и гибкость фреймворка: легко добавлять, удалять, редактировать существующие сценарии тестирования, операции и пакеты запуска. После разработки фреймворка и проведения краткого обучения работе с ним, требования к квалификации специалистов, покрывающих систему автотестами, существенно снижается, достаточно навыков работы XML. К недостаткам относится относительная сложность (особенно – в-создании фреймворка).

8.3. Инструменты, применяемые при тестировании программ

Существуют различные инструменты для обнаружения и исследования проблем в разных узлах системы: приложениях, базах данных, сети, обработки на клиентской стороне, балансировки нагрузки. Средства автоматизации тестирования можно поделить на **две группы**: 1) инструменты функционального и 2) инструменты нагрузочного тестирования. Кроме этого, существуют так называемые 3) средства поддержки процесса тестирования.

1) Инструменты **функционального тестирования**. Функциональное тестирование базируется на стратегии «черного ящика и направлено на проверку правильности выполнения функций ПП на различных наборах тестовых данных.

Является элементом различных видов тестирования: регрессионного, конфигурационного, модульного, интеграционного и некоторых других (отсюда и название группы). В числе основных функций этих инструментов создание, поддержка и выполнение тест-кейсов, сбор и анализ информации по результатам тестирования.

К инструментам **функционального** тестирования относят:

А) ***Тестовые мониторы***. Включают a) ядро системы, содержащее основные программы тестирования и оформления результатов, b) тестовую базу с исходными тестами и эталонными результатами, c) настраиваемое тестовое пространство. При запуске тестового монитора тесты из тестовой базы прогоняются через ПО, что отображается в тестовом пространстве.

Б) ***Средства отслеживания тестового покрытия.*** Выявляют тестовое покрытие программы и участки кода, пропущенные при тестировании. Часто в качестве вспомогательного средства используют профайлер. Примеры систем: Rational PureCoverage; Java Test Coverage and Instrumentation Toolkits .

В) ***Средства динамического построения профиля программы.*** Выявляют фрагменты программы, исполняемые при запуске. Программу запускают с включенным режимом добавления кода с целью генерации информации для построения профиля. Далее активизируют профайлер. В простейшем случае *результатом* работы профайлера будет исходный текст программы, где для каждого линейного участка указано, сколько раз этот участок был исполнен. Таким образом можно выявить код, оптимизация которого даст наибольший эффект, собрать статистические данные о работе тестируемой программы, узнать вклад каждого модуля программы в длительность выполнения операций.

Примеры профайлеров: gnuprof, ProDelphi, prof для java.

Г) ***Системы построения срезов программы.*** Целью построения среза является определение частей кода, влияющих на значение интересующей переменной в некоторой позиции. *Статический* срез учитывает все возможные пути исполнения программы, *динамический* – свойства программы для определенного входа (набора тестовых данных). Чем короче срез, тем проще обнаружить ошибку. Действие системы основано на отслеживании трассировки исполнения программы в обратном направлении. Однако, существуют методы, вычисления динамического среза в процессе исполнения программы, вычисления статических срезов для неструктурированных программ.

Д) ***Отладчики***. Обеспечивают трассировку программы, позволяют выполнять остановки в указанных точках или при заданных условиях. Точкой останова называют некоторое условие, при выполнении которого отладчик прерывает выполнение программы. После прерывания *возможна пошаговая отладка* программы. Также можно изменить значение переменной и продолжить выполнение программы с новыми значениями. Для наблюдения за состоянием программы в отладчике предусмотрен целый ряд специальных окон. Можно просматривать и, при необходимости, изменять текущие значения переменных, ячеек памяти, регистров процессора.

Отладчики бывают *интегрированные* (встроены в системы программирования, например, отладчики Visual Studio, Borland Pascal) и независимые (например, SoftIce). Также существуют *специализированные* отладчики**,** например: средства динамической отладки распределения памяти; средства отладки многопоточных и параллельных приложений; отладочные компиляторы. Например, thd (Thread Analiser) – нитевой анализатор многопоточных приложений, looptool – анализатор распараллеливания программ.

Интегрированные средства отладки.. Для Большинство современных систем программирования включают средства отладки того чтобы контролировать

выполнение программы и наблюдать её состояние, нужно запустить её под управлением отладчика.

На рынке средств **функционального** тестирования представлены главным образом продукты компаний: HP (QuickTest Professional, WinRunner), IBM (Robot, Functional Tester), Borland (SilkTest) и AutomatedQA (Test Complete). Большинство инструментов ориентировано на работу с Web-приложениями и приложениями, написанными на .Net или Java.

Например, инструмент ***HP Quicktest Professional*** (QuickTest Pro или QTP, HPE UFT) предназначен для автоматизации функционального и регрессионного тестирования. Используется для тестирования веб-приложений, клиент-серверных приложений, .NET-приложений, Java-приложений. Для записи сценариев тестирования служит язык Visual Basic Scripting Edition. Однако записать тесты можно путем автоматической регистрации действий пользователя при работе с тестируемым приложением. Записанные сценарии могут исполняться многократно с целью проверки работоспособности ПО. В числе достоинств инструмента:

- интеграция с Mercury Business Process Testing и Mercury Quality Center;
- уникальное распознавание смарт-объектов;
- наличие механизма обработки ошибок;
- возможность создания параметров для объектов, контрольных точек и таблиц, управляемых данными;
- автоматизация документирования хода и результатов тестирования.

Большой популярностью достаточно длительное время пользуется инструмент ***Selenium***. Это бесплатная среда для тестирования веб-приложений в различных браузерах и платформах, таких как Windows, Mac и Linux. Selenium помогает тестировщикам писать тесты на разных языках программирования, таких как Java, PHP, C #, Python, Groovy, Ruby и Perl. Selenium предлагает функции записи и воспроизведения для написания тестов без изучения Selenium IDE. К недостаткам инструмента относят необходимость владения продвинутыми навыками программирования и написания скриптов для полноценной работы с продуктом и ограниченность функционала в сравнении с платными аналогами.

2) Инструменты **нагрузочного тестирования** сложнее, чем средства автоматизации функционального тестирования, так как работают на уровне протокола (средства автоматизации регрессионного тестирования работают на уровне объектов графического пользовательского интерфейса). Они фактически «перехватывают» трафик между тестируемым приложением и сервером и представляют его в виде, удобном для работы.

Кроме поддержки протокола тестируемого приложения инструменты нагрузочного тестирования должны иметь встроенные средства мониторинга параметров серверов, средства анализа результатов и построения отчетов о нагрузочном тестировании. Должны предусматривать возможность гибкой настройки сценариев нагрузочного тестирования. Следует отметить, что для большинства видов тестирования производительности используется один и тот же инструментарий, умеющий выполнять типовые задачи.

Лидерами средств нагрузочного тестирования являются HP (LoadRunner) и IBM (Robot и Performance Tester), В числе коммерческих инструментов: Rational Performance Tester (IBM Rational), Silk Performer (Borland (Segue)), NeoLoad (Neotys); *бесплатных* – Jmeter, Grinder.

Например, в Яндексе, как один из основных инструментов измерения производительности, применяется ***Яндекс.танк***. Основные возможности: встроенный мониторинг ресурсов тестируемого сервера по протоколу SSH; автоматическая остановка теста по заданным условиям; вывод результатов

в консоль и в виде графиков; подключение своих модулей. Танк позволяет создать 3 вида нагрузки: 1) постоянная (указывается количество запросов и время), 2) линейно-нарастающая (указывается начальное и конечное значение и время), 3) ступенчато-нарастающая (указывается начальное и конечное значение, шаг увеличения нагрузки и время). Указанные виды нагрузки можно комбинировать в одном тесте.

OpenSTA позволяет провести нагрузочное испытание HTTP/HTTPS сервисов, создавать тестовые сценарии на специализированном языке SCL (Script Control Language). Для упрощения создания и редактирования сценариев используется специальный инструмент Script Modeler: ходите по сайту, собирая ссылки, которые будут сохранены в скрипт. Все параметры запроса поддаются редактированию, возможна подстановка переменных. Настройки прокси задаются в самом скрипте, поэтому можно указать несколько серверов. Реализована возможность организации распределенного тестирования. Каждая из машин такой системы может выполнять свою группу заданий, а repository host осуществляет сбор и хранение результатов. Результаты тестирования, включающие время откликов, количество переданных байт в секунду, коды ответа для каждого запроса и количество ошибок выводятся в виде таблиц и графиков.

Apache JMeter является Java-приложением с открытым кодом, предназначен для нагрузочного тестирования не только веб-приложений и их отдельных компонентов (скрипты, сервлеты, Java объекты и др.), но также FTP-серверов, баз данных (с использованием JDBC) и сети. Функциональность расширяется с помощью плагинов. Поддерживается SSL (через Java Secure Sockets Extension). Возможно проведение тестов как с использованием графического интерфейса, так и из командной строки. Использование Java подразумевает кроссплатформенность. Распространяется под Apache License. В JMeter предусмотрены механизмы авторизации виртуальных пользователей, поддерживаются пользовательские сеансы, шаблоны, кэширование и последующий анализ результатов теста, функции позволяют сформировать следующий запрос, основываясь на ответе сервера на предыдущий. Есть возможность проводить распределенные тесты. В этом случае один из компьютеров является сервером (bin/jmeter-server.bat), который управляет клиентами и собирает итоговую информацию. Для работы достаточно запустить ApacheJMeter.jar или в консоли jmeter.bat (Windows) или jmeter.sh (*nix).

Примером **универсального** инструмента, который кроме функционального, автоматизирует также модульное и нагрузочное тестирование является ***Testing Anywhere***. Запись GUI сценариев и простота использования без особых навыков программирования дают значительное преимущество в создании тестов. Язык тестов: визуальное проектирование. Программный инструмент является платным, предназначен для тестирования веб- и windows-приложений, поддерживает разные платформы. Поддерживаемые технологии: VB.NET, C#, C++, Win32, VB6, AJAX, ActiveX, JavaScript, HTML, Delphi, Java, Perl, 32 bit apps, 64 bit apps, Oracle Forms, PHP, Python, Macromedia Flash 1.0-8.0, Adobe Flash 9.0 & later.

3) Кроме выполнения тест-кейсов тестировщик вправе переложить на вычислительную технику и ряд других работ. Какие-то действия можно автоматизировать, создав собственные простые программки. Например, разработать скрипт, который будет по журналам работы средства автоматизации формировать таблицы и графики по заданным параметрам. Или написать программу для создания и проверки резервных копий.

В ряде случаев возможно применение универсальных средств: офисных приложений, систем управления базами данных (СУБД), анализаторов текста и т.п.

Специальные **средства поддержки процесса** тестирования, позволяют вести учет требований и тест-кейсов, анализировать покрытия требований тестами, управлять ходом тестирования, вести учет обнаруженных дефектов и т.п. Например, HP Quality Center, Rational Quality Manager – Web-приложения, предназначенные для управления тестированием и контролем качества на всех этапах процесса разработки.

Учитывать обнаруженные в программах ошибки и пожелания пользователей позволяют *системы отслеживания проблем* (багтрекеры). Обычно они представляют собой интерфейс к базе данных, в которой собрана информация, связанная с проблемой:

— основная информация об ошибке (номер, описание, приоритет, состояние);

— место проявления ошибки (ПП, версия, компонент, особенность);

— персоналии (менеджер; инженер, выставивший ошибку; адреса рассылки);

— анализ ошибки и исправление (тест, на котором проблема проявляется; предлагаемое исправление; комментарии; информация для пользователей);

— влияние исправления (на другие компоненты, документацию к ПП).

Примеры багтрекеров: Bugzilla, PVCS Tracker, JIRA, Redmine.

8.4. Выбор инструмента

Можно выделить несколько факторов, на которые следует обратить внимание при выборе инструмента:

1) **Функционал.** Инструмент должен предоставлять возможность построения и настройки сценариев тестирования, автоматической регистрации обнаруженных дефектов, анализа результатов, создания отчетов о тестировании. Для **функционального** тестирования важна **поддержка конкретной среды** разработки, наличие сценариев восстановления. Для инструментов **нагрузочного** тестирования требуется **поддержка** используемого тестируемым приложением **протокола**, наличие встроенных средств мониторинга параметров серверов.

2) **Наличие документации и технической поддержки, популярность инструмента.** Современные инструменты тестирования достаточно сложны, что затрудняет их освоение. Популярность говорит об удобстве применения программного средства. Наличие многочисленной группы активных пользователей и форума по конкретному инструменту поможет быстрее получить ответ на возникший вопрос.

3) В**озможность интеграции** инструментов тестирования с используемым в компании программным обеспечением.

4) Степень **распознавания** инструментом для автоматизации **элементов управления** в тестируемом приложении. Если элементы не распознаются, и не удаётся найти соответствующий плагин или модуль, то от инструмента лучше отказаться.

5) **Удобство** инструмента **для написания новых скриптов** и **поддержки существующих.** Можно на простом примере проверить, сколько требуется на это времени, насколько можно структурировать код, его читаемость и т.д.

6) **Стоимость** инструмента. Нецелесообразно покупать дорогостоящие инструменты для «одноразового» тестирования. Дешевле обойдется приобретение временных лицензий (аренда лицензий). Не стоит забывать о затратах на внедрение инструмента, адаптацию, обучение сотрудников работе с ним.