УДК 004.415.53 DOI: 10.26565/2519-2310-2019-1-07

АНАЛИЗ ИНСТРУМЕНТОВ ДЛЯ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Ольга Мелкозерова, Алексей Нарежний, Сергей Малахов

Харьковский национальный университет имени В.Н. Каразина, Харьков, 61022, Украина
olja.mex@gmail.com, o.nariezhnii@karazin.ua, mailgate@meta.ua

Рецензент: Вячеслав Калашников, д.ф.-м.н., проф., Технологический университет Монтеррея, 64849 Монтеррей, Нуево-Леон, Мексика
kalash@itesm.mx

Поступила: Май 2019.

Аннотация: *Тестирование качества программного обеспечения (ПО) является трудоемким и ответственным этапом его разработки. Это обуславливает практический интерес к автоматизации основных этапов тестирования. Как показывает практика, использование всевозможных инструментов тестирования ПО заметно облегчает этот процесс. Однако, принципиально важно применять тот или иной инструмент, учитывая специфику каждого конкретного случая. Это обстоятельство обусловлено большими объемами тестируемой информации и сложностью эксплуатационной документации. В работе приведен обзор и анализ возможностей существующих инструментов для проведения автоматизированного тестирования ПО, с описанием соответствующих технологий, назначения и области применения. Приведены примеры составления тестов с использованием Selenium, SerenityBDD и JMeter.*

Ключевые слова: *автоматизированное тестирование; инструменты автоматизированного тестирования; технологии автоматизированного тестирования.*

1 Введение

Тестирование программного обеспечения (ПО) является важнейшим этапом в ходе его разработки. Как известно, процесс тестирования ПО можно разделить на мануальное (выполняется вручную) и автоматизированное. Для автоматизированного тестирования ПО используются различные подходы и инструменты, которые способны частично заменить участие человека при проведении значительного количества рутинных процедур. В настоящее время существуют и в той или иной мере используется большое количество подобных программных инструментов. В связи с этим, в данной роботе представлен обзор соответствующих решений (инструментов и технологий) с кратким описанием их назначения и целевой областью применения. Кроме того, профильные специалисты выделяют еще и виды тестирования, которые следует рассматривать, как список действий, которые необходимо совершить при тестировании того или иного ПО.

2 Технологии автоматизированного тестирования

К технологиям автоматизированного тестирования относят [1]:

1) частные решения;
2) тестирование под управлением данными (*DDT - data driven testing*);
3) тестирование под управлением ключевыми словами (*KDT - Keyword Driven Testing*);
4) использование фрéймворков (*Frameworks*);
5) запись и воспроизведение (*Record & Playback*);
6) поведенческое тестирование (*BBT - Behaviour Driven Testing*).

Следует отметить, что один и тот же инструмент тестирования может использовать сразу несколько технологий автоматизированного тестирования. Например, фреймворк Selenium IDE [2], в настоящее время по своему прямому назначению практически не используется, однако очень удобен для обучения соответствующих специалистов. В данном случае, необходимо буквально несколько минут, для того чтобы человек, незнакомый с автоматизированным тестированием, написал свой первый тест. Selenium IDE (*Integrated Development Environment*) использует технологию записи и воспроизведения (*Record & Playback*), однако все

© Мелкозерова О., Нарежний А., Малахов С., 2019

его команды (например, «open» или «verifyText») – это уже тестирование под управлением ключевыми словами (*KDT)*.

В общем случае в связи с появлением фрейворков, ускорился процесс обучения специалистов в области обеспечения качества создаваемого ПО. Использование этой технологии сводится к следующим основным шагам:

1. Тестировщик ПО вручную выполняет тест-кейс, а средство автоматизации его записывает. Тест-кейс можно сохранить в html-формате (рис. 1).
2. Результаты записи представляются в виде кода автоматизированного теста на высокоуровневом языке программирования (*в некоторых случаях специально разработанном*).
3. Тестировщик редактирует полученный код.
4. Готовый код выполняется для проведения тестирования в автоматизированном режиме.

1_01		
open	https://www.olx.ua/uk/changelang/?lang=ru&l=https%3A%2F%2Fwww.olx.ua	
verifyText	css=span.link.inlblk > strong	Мой профиль
verifyText	//div[@id='lastwrapper']/div/div[2]/div/div[2]/ul/li[5]/a/span	Популярные запросы

Рис. 1 – Тест-кейс *(вариант)*

К инструментам Selenium следует отнести также: Selenium WebDriver и Selenium Server.

Selenium WebDriver – набор библиотек для различных языков программирования, позволяющий управлять браузером из программы, написанной на этом языке программирования. Представляет собой надежный фреймворк автоматизации, способный работать с любым браузером. Кроме того, позволяет разработать большой тестовый набор, включающий тесты с достаточно сложной логикой поведения и проверок.

Selenium Server – может принимать команды от удаленного компьютера, где работает сценарий автоматизации и исполнять их в браузере. Несколько серверов Selenium могут формировать распределенную сеть, которая называется Selenium Grid, что обеспечивает масштабирование стенда автоматизации.

Selenium поддерживает команды трех видов [2]: - действия (*actions*); - считыватели (*accessors*); - проверки (*assertions*).

Действия – это команды, которые, как правило, управляют состоянием приложения. Они совершают процедуры вроде «щелкнуть по той ссылке» или «выбрать эту операцию». Если действие не может быть выполнено, либо выполняется с ошибкой, то текущий тест прерывается.

К большей части действий можно добавить суффикс «AndWait» («подождать»), «ClickAndWait». Этот суффикс сообщает Selenium, что действие принудит браузер совершить запрос к серверу и что Selenium должен дождаться загрузки новой страницы.

Считыватели анализируют состояние приложения и сохраняют результаты в переменные *(к примеру, команда «storeTitle»)*. Кроме того, они используются для автоматической генерации проверок.

Проверки похожи на «Считыватели», однако, в отличие от них, проверяют соответствие текущего состояния приложения, ожидаемому. Например, удостовериться, что заголовок страницы имеет определенное название.

Технологию записи и воспроизведения (*Record & Playback*) использует также инструмент для тестирования производительности JMeter [3], инструмент UI Katalon Studio [4] и другие фреймворки.

3. Уровни тестирования

К уровням тестирования относят [5,6]:

- компонентное тестирование (модульное), сфокусированное на тестировании индивидуальных компонентов (*unit testing*);

- интеграционное тестирование. Выявляет дефекты интерфейса и ошибки между взаимодействующими компонентами или системами (*integration testing*);
- системное тестирование (*system testing*). Проверяет взаимосвязанные системы для верификации указанных требований (*выполняется инженерами по контролю качества*);
- приемочное тестирование (*acceptance testing*). Верифицирует соблюдение с требованиями, бизнес процессами и пользовательскими пожеланиями.

Модульное тестирование нацелено на поиск возможных дефектов и верификацию функционирования программных модулей, программ, объектов, классов [1], которые можно протестировать изолированно.

Компонентное тестирование может включать в себя, как тестирование функциональности и специфичных нефункциональных характеристик (таких как поведение ресурсов (*например, поиск «утечки» памяти*)) или тестирование надежности, так и структурное тестирование (*например, покрытие кода*). На практике компонентное тестирование, обычно, выполняется разработчиками, которые непосредственно пишут код. При этом, выявленные дефекты обычно исправляются сразу же, без занесения их в базу дефектов.

Для тестирования на этом уровне можно использовать семейство библиотек *XUnit*, к примеру для языка программирования Java следует использовать *TestNG* и *JUnit* - фреймворки для написания повторяющихся модульных тестов. Можно подключить через зависимости [7]. Зависимости – это те библиотеки, которые непосредственно используются в данном проекте для целей компиляции кода или его тестирования.

JUnit обеспечивает перегруженные методы для всех примитивных типов, объектов и массивов. Параметры: - ожидаемое значение и актуальное значение. Опционально первым параметром может быть сообщение, которое появляется, если тест «провален». Например, приведенная ниже строка проверяет, что два примитива равны:

Assert.assertEquals(expectedResult,result,0) .

Уровни интеграционного тестирования:

- компонентное интеграционное тестирование проверяет взаимодействие между программными компонентами и производится после компонентного тестирования;
- системное интеграционное тестирование проверяет взаимодействие между системами или между аппаратным обеспечением и может быть выполнено после системного тестирования.

4. Тестирование под управлением поведением (BBT)

Все технологии автоматизированного тестирования ПО имеют одну общую особенность: - они сфокусированы на технических аспектах поведения приложений. Кроме того, они обладают и общим недостатком: - с их помощью сложно проверить высокоуровневые пользовательские сценарии. Этот недостаток призвано исправить тестирование под управлением поведением, при котором акцент делается не на отдельных технических деталях, а на общей работоспособности приложения при решении типичных пользовательских задач.

Такой подход не только упрощает выполнение целого ряда проверок, но и облегчает взаимодействие между разработчиками, тестировщиками, бизнес-аналитиками и заказчиками ПО. В его основе лежит формула «*given – when – then*» [1].

Given (“имея, предполагая, при условии”) описывает начальную ситуацию, в которой находится пользователь в контексте его работы с приложением.

When (“когда”) описывает набор действий пользователя в данной ситуации.

Then (“тогда”) описывает ожидаемое поведение приложения.

Примером использования этой технологии является *Serenity BDD* [8].

Код для теста выглядит следующим образом (см. рис. 2):

```
@Test
public void verifySubCategory() throws InterruptedException {
    // GIVEN
    stepsForSerenity.a_user_visits_a_page(siteHomePage);
    // WHEN
    stepsForSerenity.the_user_chooses_category_Compurer();
    // THEN
    stepsForSerenity.the_user_can_see_subcategory_ITService("IT услуги");
}
```

Рис. 2 – Код теста для Serenity BDD *(пример)*

Один из шагов Serenity BDD:

```
@Step("Given the user visits a page {0}")
public void a_user_visits_a_page(String homePage) {
    this.siteHomePage = homePage;
}
```

Рис. 3 – Пример шага для Serenity BDD.

Также можно сгенерировать отчет (см. Рис. 4):

Steps	Outcome	Duration
Given the user visits a page https://pn.com.ua/	SUCCESS	0,05s
When the user chooses category Compurer	SUCCESS	37,49s
Then the user sees IT услуги subcategory	SUCCESS	1,62s
	SUCCESS	*39,51s*

Рис. 4 – Отчет о прохождении теста для Serenity BDD.

5. Тестирование производительности

Тестирование производительности (*performance testing*) – исследование «скоростных показателей» приложения при различной (по характеру и количественных показателях) нагрузке [1].

Нагрузочное тестирование (*load testing*) – исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов («запас прочности»).

Стрессовое тестирование (*stress testing*) – исследование поведения приложения при не декларированных изменениях нагрузки.

Объемное тестирование (*volume testing*) – исследование производительности системы при обработке различных объемов данных.

Тестирование масштабируемости (*scalability testing*) – исследование способности приложения увеличивать показатели производительности в соответствии с увеличением количества доступных данному приложению ресурсов.

Конкурентное тестирование (*concurrency testing*) – исследование поведения приложения, в случаях, когда ему приходится обрабатывать большое количество одновременно поступающих запросов, что вызывает конкуренцию между этими запросами за ресурсы (*базу данных, память, канал передачи данных, дисковую систему и др.*). Иногда под конкурентным тестированием понимают, также исследование работы многопоточного приложения и корректность синхронизации действий.

Тестирование надежности (*reliability testing*) - тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени или заданного количества операций.

Тестирование восстанавливаемости (*recoverability testing*) – тестирование способности приложения восстанавливать свои функции и заданный уровень производительности, а также восстанавливать свои данные в случае возникновения критической ситуации, приводящей к временной или частичной утрате работоспособности приложения.

Тестирование отказоустойчивости (*failover testing*) – эмуляция или реальное создание критической ситуации, с целью проверки способности приложения задействовать соответствующие встроенные механизмы, обеспечивающие предотвращение нарушения работоспособности, производительности и повреждения данных.

Элементы тест плана *JMeter* [9]:

- группы потоков (*Thread groups*);
- логические контролеры (*Logic controller*);
- типовые контроллеры (*Samle generating controller*);
- слушатели (*Listeners*);
- таймеры (*Timers*);
- соответствия (*Assertions*);
- конфигурационные элементы (*Configuration Elemetents*).

Группы потоков – начальные точки любого тест-плана (рис. 5). Все контроллеры и образцы должны быть в группе потоков. Другие элементы, такие как слушатели, могут располагаться под тест-планом, в котором они применяются для всех потоков групп. Элемент группы потоков управляет количеством потоков, который *JMeter* будет использовать для выполнения теста. В общем случае можно установить:

- количество потоков (пользователей) (*Number of Threads*);
- время наращивания (*Rump-Up Period*);
- количество повторов для выполнения теста.

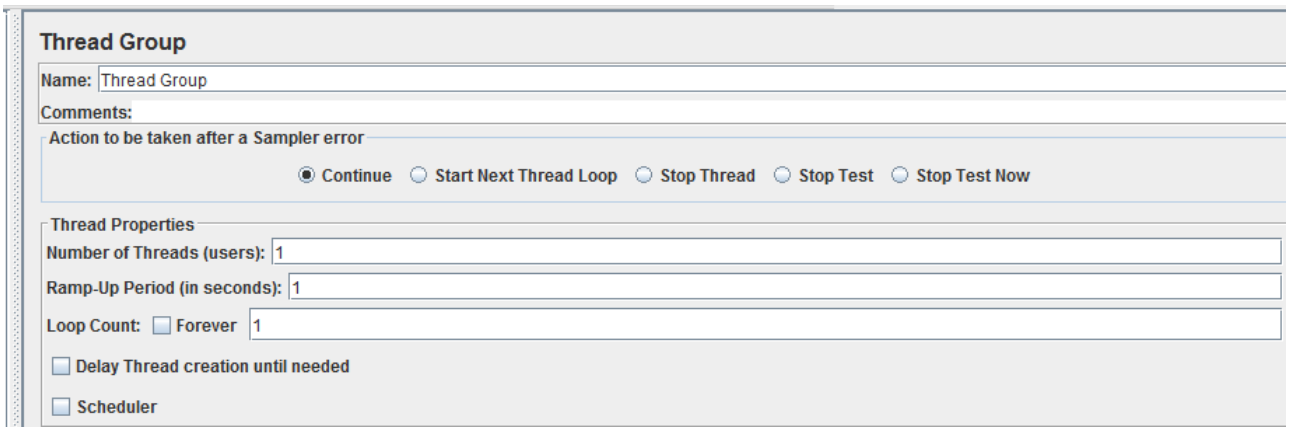

Рис. 5 – Группы потоков *Thread groups*.

Каждый поток будет исполнять свой тест-план полностью и абсолютно независимо от других тестовых потоков. Произведение потоков используется для моделирования конкурентных соединений для сервера приложения.

Rump-Up Period «сообщает» *JMeter*, сколько времени потребуется, чтобы нарастить количество потоков до полного количества. Например: - если используется 10 потоков, а Rump-*Up Period* 100 секунд, то *JMeter* «возьмет» 100 секунд для того, чтобы получить все 10 потоков. При этом каждый поток будет начинаться через 10 секунд (100/10) после того, как запустился предыдущий поток. Если есть 30 потоков, и время наращивания составляет 120 секунд, то каждый последующий поток будет задерживаться на 4 секунды.

Loop count – количество циклов теста, *forever* – тест будет длиться вечно.

Время наращивания должно быть достаточно длинным, чтобы избежать слишком больших нагрузок в начале теста и достаточно коротким, чтобы последние потоки начинали запускаться до того, как закончились первые.

Thread group также предоставляет планировщик *Scheduler* (рис. 6). Следует кликнуть на соответствующую кнопку, чтобы получить доступ к дополнительным полям, в котором пользователь может установить: - продолжительность (*Duration*) теста; - задержку запуска (*Startup Delay*); - время начала и окончания теста. Также пользователь может выбрать продолжительность и задержку, чтобы контролировать каждый поток. Здесь можно запускать тест в определенное время, например, ночью.

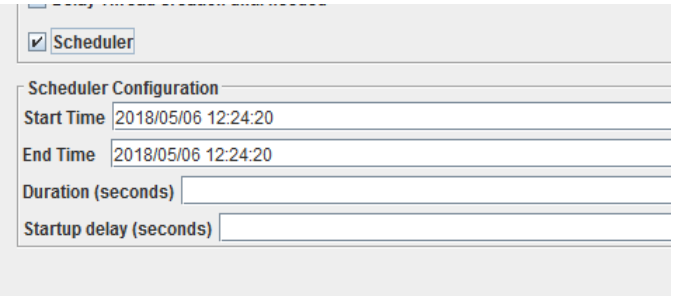

Рис. 6 – Группы потоков *Thread groups* *Scheduler* (*запуск теста ночью*)

JMeter имеет два типа контроллеров: типовые контроллеры (*Samplers*) и логические контроллеры (*Logical Controllers*). Они управляют процессом теста.

Типовые контроллеры отправляют запрос на сервер. Например: - добавить HTTP запрос, если вы хотите отправить HTTP запрос. Пользователь также может настроить запрос добавлением конфигурационных элементов к типовым контроллерам.

Логические контроллеры позволяют настроить логику, которую *JMeter* использует, когда отправляет запрос. Например: - пользователь может добавить *Interleave Logic Controller* в качестве альтернативы между двумя *HTTP* запросами типовых контроллеров.

Listener – это компонент, который показывает результаты образцов. Результаты могут быть представлены в дереве, таблице, графах или быть записаны в лог-файл.

Graph result listener – формирует простой граф (см. рис. 7-13). Зависимости, представленные на рис. 13-14 отображают следующие параметры (*см. цветовую маркировку*):

- данные - Data (черный цвет);
- среднее значение времени - Average (голубой цвет);
- стандартное отклонение - Deviation (красный цвет);
- пропускную способность - Throughtput (зеленый цвет) и медиану (Mediane).

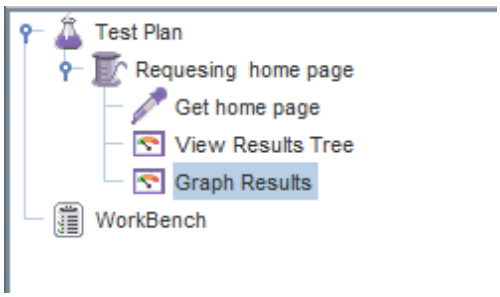

Рис. 7 – Фрагмент тест-плана для *Apache JMeter*

Пропускная способность отображает реальное количество запросов в минуту, которое обрабатывает сервер.

Составим наш тест-план (рис. 5). В него добавим *Thread Group - Requesting home page* (рис. 8) и *HTTP* запрос (рис. 9). Затем добавляем «Слушатели» (*Listeners*) *View Result* и *Tree Graph Results*. Сохраняем тест-план и запускаем на исполнение.

Tree Graph Results – дерево ответов всех образцов, позволяют пользователю увидеть отклик для всех образцов. В дополнение к показу отклика, можно видеть время этого отклика (*Sampler Result* – рис. 10) и некоторые коды для этих откликов (например, в *HTML* формате – рис. 11).

После запуска тест-плана, можно увидеть ответ на запрос (рис. 10 и 11), сообщение (отклик 200 ок). При этом если с сайтом возникнут проблемы, то сообщение будет выглядеть следующим образом – рис. 12. Результат запуска тест-плана при выполнении *Graph Result* представлен на рис. 13.

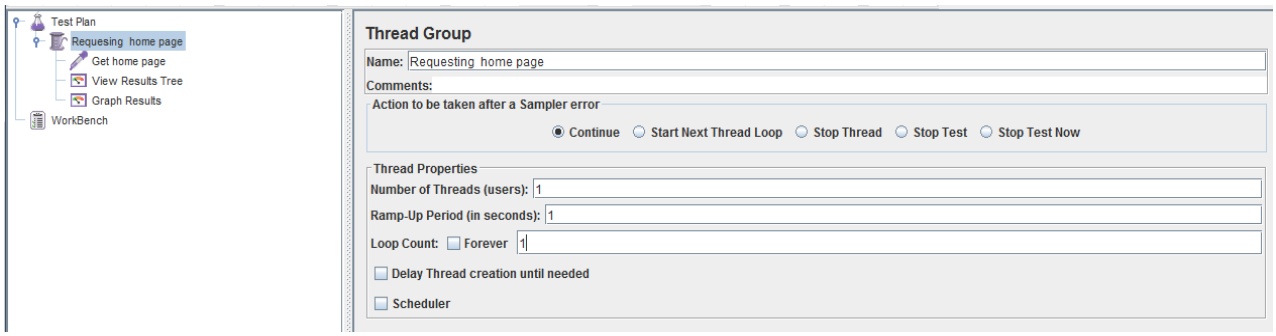

Рис. 8 – Изменение количества пользователей *Apache JMeter*

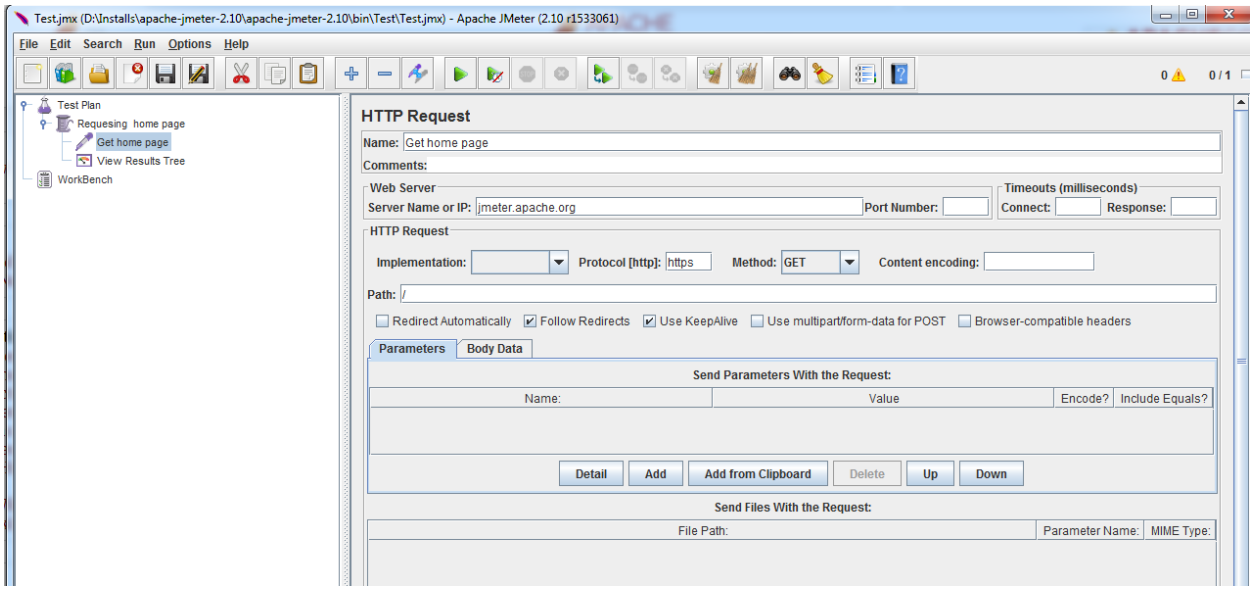

Рис. 9 – Заполнение *HTTP* запроса

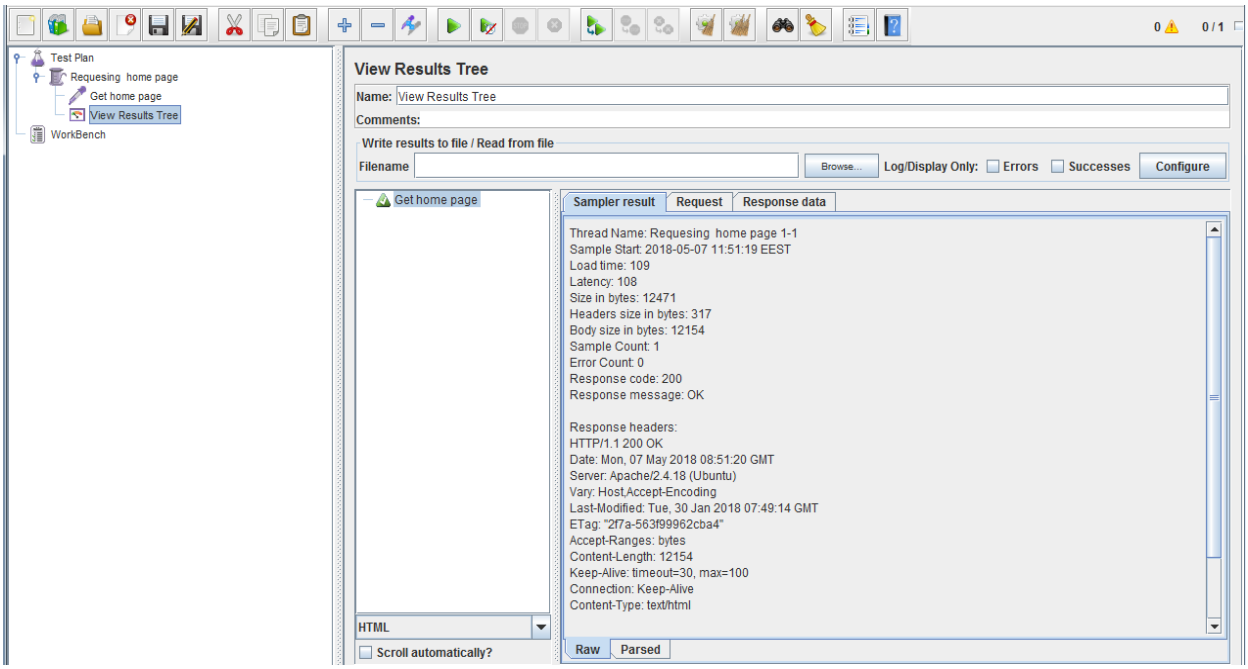

Рис. 10 – Результат запуска тест-плана при выполнении *View Result Tree*

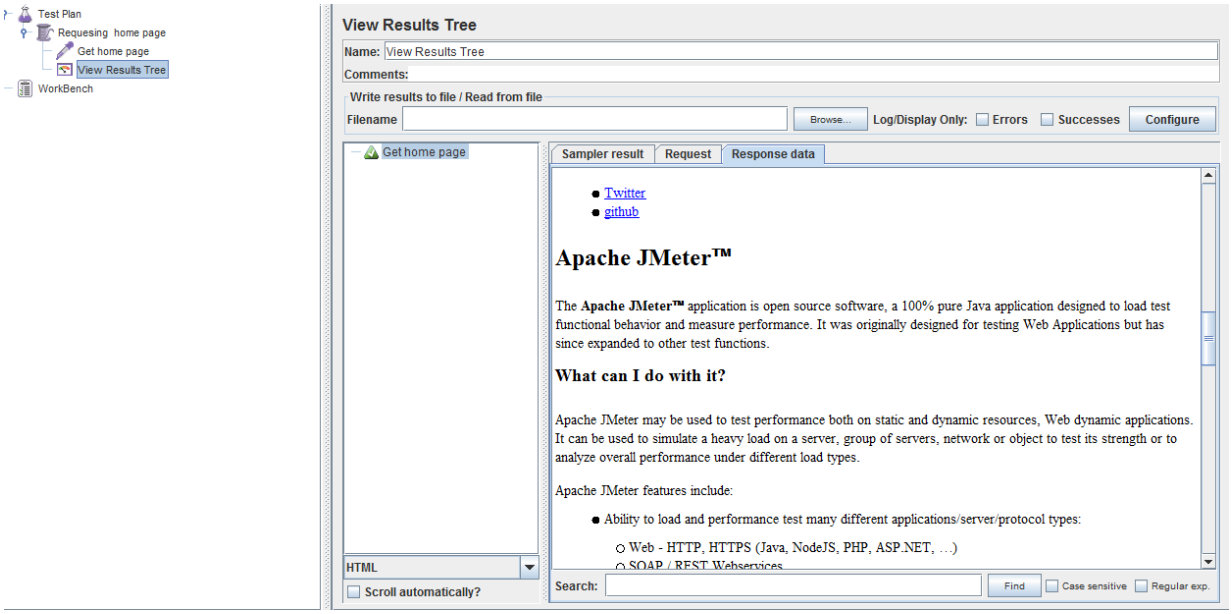

Рис. 11 - Результат запуска тест-плана при выполнении *View Result Tree*

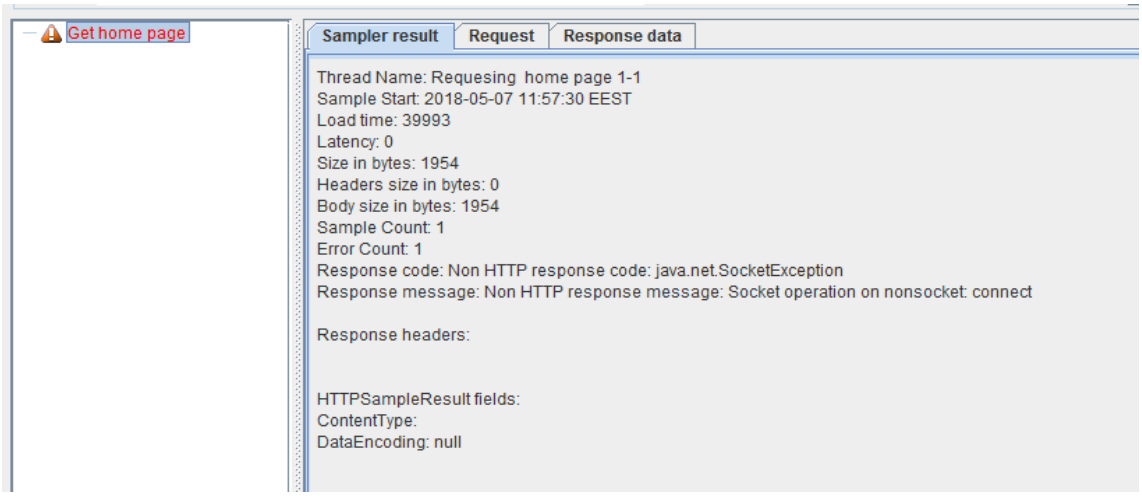

Рис. 12 - Результат запуска тест-плана при выполнении *View Result Tree*

Рис. 13 – Результат запуска тест-плана при выполнении *Graph Result*

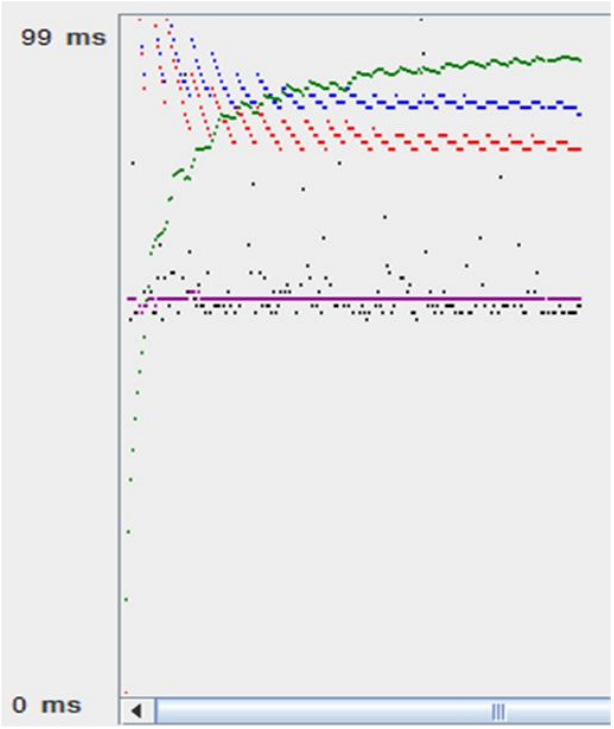

Рис. 14 – Фрагмент интерфейса рабочего окна «*Graph Results*» с результирующими зависимостями

5 Выводы

Автоматизированное тестирование, по сравнению с ручным, обладает целым рядом ощутимых преимуществ, где основные - это быстрота выполнения и отсутствие ошибок, обусловленных исключением человеческого фактора.

Тестирование производительности ПО - такой вид тестирования, где применима только автоматизация. Она также неизбежна на этапе разработки программного обеспечения. Но при ее применении также есть и недостатки, это большое количество времени на написание автотестов, их поддержку, необходим более квалифицированный персонал.

На этапе разработки ПО для модульного (компонентного) тестирования используют различные библиотеки тестирования (*JUnit, TestNG* и другие).

Для тестирования всей системы используют фреймворки, их существует большое множество. Например, для тестирования производительности сайтов используется *Jmeter*, который, в свою очередь, также позволяет тестировать *API* (*Application programming interface*). У данного вида тестирования есть определенные преимущества, в сравнении с *UI* тестированием:

1. Раннее тестирование – разработчики сначала делают *API*, а потом уже графический интерфейс. При этом существует возможность проверить логику раньше, чем дорисуют *GUI*.
2. Графического интерфейса может в принципе и не быть.
3. Скорость – вызов одного запроса занимает долю секунды. А вот через интерфейс повторить процедуру часто бывает сложно.
4. Автоматизация – даже если нет автотестов на уровне *API* приложения, всегда можно их создать вручную через *PostMan* или *Jmeter*.

На пользовательском уровне при проведении приемочного тестирования, хорошо себя зарекомендовали решения под управлением поведением, например: - *Serenity BDD* или *Cucumber*.

Ссылки

[1] Куликов С. Тестирование программного обеспечения. Базовый курс. URL: http://svyatoslav.biz/software_testing_book/ (дата звернення: 28.12.2017)
[2] SeleniumIDE. URL: https://www.seleniumhq.org/docs/ (дата звернення: 16.06.2019)
[3] Jmeter. URL: https://jmeter.apache.org/ (дата звернення: 16.06.2019)
[4] Katalon. URL: https://www.katalon.com/ (дата звернення: 16.06.2019)
[5] Основи инженеренерии качества / Андон Ф.И. та ін. Киев: Академпериодика, 2007. 672 с.
[6] Стандартный глоссарий терминов, используемых в тестировании программного обеспечения. International Software Testing Qualifications Board. – 2014. URL: https://docplayer.ru/340620-Standartnyy-glossariy-terminov-ispolzuemyh-v-testirovanii-programmnogo-obespecheniya.html
[7] Maven Reposotory. URL: https://mvnrepository.com / (дата звернення: 16.06.2019)
[8] Serenity. URL: http://www.thucydides.info/docs/serenity / (дата звернення: 16.06.2019)

Рецензент: В'ячеслав Калашников, д.ф.-м.н., проф., департамент систем і промислового виробництва Технологічного університету Монтеррея, Монтеррей, Мексика. E-mail: kalash@itesm.mx

Надійшло: Травень 2019.

Автори:
Ольга Мелкозьорова, к.т.н., старший викладач кафедри Безпеки інформаційних систем і технологій, Харківський національний університет імені В.Н. Каразіна, Харків, Україна**.**
E-mail: olja.mex@gmail.com
Олексій Нарєжній, к.т.н., доцент кафедри Безпеки інформаційних систем і технологій, Харківський національний університет імені В.Н. Каразіна, Харків, Україна.
E-mail: o.nariezhnii@karazin.ua
Сергій Малахов, к.т.н., с.н.с., Харківський національний університет імені В.Н. Каразіна, Харків, Україна.
E-mail: mailgate@meta.ua

Аналіз інструментів для автоматизованого тестування програмного забезпечення.
Анотація. Тестування якості програмного забезпечення (ПЗ) є дуже трудомістким та відповідальним етапом при його розробці. Це обумовлює практичний інтерес до автоматизації основних процедур при тестуванні. Як показує практика, використання можливих інструментів тестування ПЗ суттєво полегшує цей процес. Але принципово важливо застосовувати той чи інший інструмент, враховуючи специфіку кожного окремного випадку. Ця обставина обумовлена великою кількістю інформації, що тестується, та складністю експлуатаційної документації. У роботі наведено огляд та аналіз можливостей існуючих інструментів, для проведення автоматизованого тестування ПЗ, з описом відповідних технологій, призначенням та області використання. Наведено приклади складання тестів з використанням Selenium, Serenity BDD и JMeter.

Ключові слова: автоматизоване тестування; інструменти автоматизованого тестування; технології автоматизованого тестування.

Reviewer: Vyacheslav Kalashnikov, Doctor of Sciences (Physics and Mathematics), Full Prof., Department of Systems and Industrial Engineering, Campus Monterrey, Monterrey, Mexico. E-mail: kalash@itesm.mx

Received: May 2019.

Authors:
Olga Melkozerova, Ph.D., Senior Lecturer, Department of Security of Information Systems and Technologies, V.N. Karazin Kharkiv National University, Ukraine.
E-mail: olja.mex@gmail.com
Oleksii Nariezhnii, Ph.D., Associate Professor, Department of Security of Information Systems and Technologies, V.N. Karazin Kharkiv National University, Ukraine.
E-mail: o.nariezhnii@karazin.ua
Serhii Malakhov, Ph.D., Senior Researcher, Department of Security of Information Systems and Technologies, V. N. Karazin Kharkiv National University, Kharkiv, Ukraine.
E-mail: mailgate@meta.ua

Analysis of tools for automated software testing.
Annotation. Software quality testing (software) is a time-consuming and responsible stage for its development. This leads to a practical interest in automating basic testing procedures. As practice shows, the use of various software testing tools greatly facilitates this process. However, it is fundamentally important to use a particular tool, taking into account the specifics of each particular case. This circumstance is due to the large volumes of information being tested and the complexity of the operational documentation. The paper provides an overview and analysis of the capabilities of existing tools for automated software testing, with a description of the relevant technologies, purpose and scope. Examples of test preparation using Selenium, Serenity BDD and JMeter are given.

Keywords: Automated Testing; Automated Testing Tools; Automated Testing Technology.