API

API

(*Application Programming Interface или программный интерфейс приложения*) – это совокупность способов, протоколов, инструментов, с помощью которых различные программы обмениваются своими возможностями, данными, выполняют разные функции.

API — интерфейс прикладного программирования, набор методов, классов, библиотек, функций, обеспечивающих взаимодействие программ между собой. Этот интерфейс определяет, как они
взаимодействуют друг с другом, используя запросы и ответы. Документация API
содержит информацию о том, как разработчики должны структурировать эти запросы и ответы.

Зачем нужен API

API создан для удобства. Когда пользователь работает с планшетом или другим девайсом, ему не приходится вникать, как компьютер обрабатывает информацию: он просто нажимает на иконки в интерфейсе. Аналогичная ситуация с API. Благодаря программному интерфейсу разработчик может подключить свой продукт к другим системам для хранения файлов, отрисовки графики, воспроизведения видео или аудио. При этом ему не приходится писать собственный код или разбираться, как именно работает ОС. Такой алгоритм упрощает и ускоряет процессы.

Перечислим основные причины интереса программистов к применению API:

- Программный интерфейс дает инструментарий для работы с ПО. Например, OpenAl помогает работать со звуковыми библиотеками в приложениях. Экономит время для разработчиков — не нужно писать звуковое ПО.
- Связывает системы. С помощью API можно авторизоваться на сайте, используя аккаунт стороннего сервиса.

- По такому же принципу работают платежные системы, соединяясь с банковскими аккаунтами.
- Обеспечивает безопасность данных. Программный интерфейс выделяет данные, которые необходимо защищать. Таким образом, другие программы не смогут ими воспользоваться, если у них на это нет разрешения.
- Снижают стоимость программного продукта. Выгоднее пользоваться API, чем создавать ПО с нуля.

Функции API

Общего перечня функций API нет: набор инструментов, доступных клиентам, определяют разработчики. Например, с помощью API YouTube можно встроить видео, а с помощью API платежных систем собирать платежи, оформлять возвраты или обрабатывать споры.

Некоторые компании предлагают API в качестве отдельного программного продукта. Допустим, вы хотите встроить интерактивные карты на сайт интернет-магазина, чтобы покупатель мог найти ближайший пункт выдачи товара, и выбираете API Яндекс.Карт. Разрабатывать собственный картографический сервис не придется. Сервис сайта отправит запрос к сервису Яндекса на поиск организаций на карте, а после получения ответа отобразит данные в браузере покупателя.

Как и кем используется API

Практически все операционные системы (UNIX, Windows, OS X, Linux и т. д.) имеют API, с помощью которого программисты могут создавать приложения для
этой операционной системы. Главный API операционных систем — это множество
системных вызовов.
API обеспечивает соединение между приложениями или программами. На удаленном сервере API — это та часть, которая получает запросы и отсылает ответы, когда вы переходите на какой-нибудь сайт в браузере.
API обеспечивает коммуникацию и позволяет доступ к веб-странице.
Ваш запрос направляется к коду, который написал
кто-то другой и которых хранится на сервере. Этот код дает вам доступ к странице,

которая хранится где-то еще на удаленном сервере.
То есть ваш запрос не идет напрямую на сервер! Он идет к API, который решает,
что с ним делать.
API также делает возможным взаимодействие между двумя сайтами и изменение
данных. Например, если на веб-странице кликнуть кнопку «Поделиться в Twitter»,
это запустит взаимодействие с вашим аккаунтом в Twitter и изменит его путем добавления нового твита.
API может принимать разные формы в зависимости от того, где он используется.
Соответственно меняется и название:
Открытые или публичные API. Эти API выложены в открытый доступ для разработчиков и других пользователей (с минимальными ограничениями).
Партнерские API открыты только для стратегических деловых партнеров. Они не доступны публично, к ним могут обращаться только те, у кого есть право доступа.
Внутренние или приватные API. Эти API спрятаны от внешних пользователей и открыты только внутренним системам. Ими пользуются разные команды разработчиков в рамках одной компании, что позволяет им повысить продуктивность.
Такие компании как Facebook, Google и Yahoo публикуют свои API, чтобы подтолкнуть разработчиков создавать что-то, используя их возможности. Эти APIинтерфейсы предоставили нам все: от новых интернет-функций, позволяющих
просматривать сайты других сервисов, до приложений для мобильных устройств,
обеспечивающих легкий доступ к веб-ресурсам.

Разработчикам программный интерфейс позволяет:

- упростить и ускорить выпуск новых продуктов, так как можно использовать уже готовые API для стандартных функций;
- сделать разработку более безопасной, выведя ряд функций в отдельное приложение, где они будут скрыты;
- упростить настройку связей между разными сервисами и программами и не сотрудничать для разработки своего продукта с создателями различных приложений;

- сэкономить деньги, так как не нужно разрабатывать все программные решения с нуля.

Бизнесу API нужны, чтобы:

- проводить транзакции;
- интегрировать потоки данных с клиентами и партнерскими системами;
- повысить безопасность автоматизированных процессов;
- развивать собственные приложения;
- внедрять инновации, например, при работе с клиентами.
-

Правительствам API позволяют:

- обмениваться данными между ведомствами;
- взаимодействовать с гражданами, получать обратную связь.

Типы API

По типу доступа программные интерфейсы API бывают:

- **Внутренние.** Используют только сотрудники компании. Нужны для решения внутренних задач организации: снижения расходов и отладки процессов.
- **Партнерские.** Создают для партнеров и клиентов компании. Применяют для разработки веб-продуктов и сокращения издержек.
- **Публичные.** Их создают для привлечения внимания и продвижения веб-продукта и компании, продаж, разработки новых сервисов и приложений.

WEB API, которые используют для создания HTTP-служб:

- **RPC** (*Remote Procedure Call*). Использовалась, когда системы были связаны в локальных сетях. Принцип работы: вызов удаленных систем похож на вызов функций внутри программы. Яркие примеры таких систем – CORBA и DCOM.
- **SOAP** (*Simple Object Access Protocol*). Это протокол для обмена сообщениями в распределительной вычислительной среде. Помимо удаленного вызова процедур, SOAP способен отправлять и получать сообщения формата XML. Работает с протоколами прикладного уровня.

- **REST** (Representational State Transfer). Это архитектура ПО для веб-служб. Обеспечивает работу с любыми форматами, будь то сайт, flash-программа, приложение другого формата и другие. Благодаря тому, что данные передаются без дополнительных слоев, REST использует меньше ресурсов, так как на каждую передачу данных требует меньше запросов.
- **Websocket API** – это еще одна современная разработка web API, которая использует объекты JSON для передачи данных. WebSocket API поддерживает
  двустороннюю связь между клиентскими приложениями и сервером. Сервер
  может отправлять сообщения обратного вызова подключенным клиентам, что
  делает его более эффективным, чем REST API.

Типы API тестирования

- **Модульное тестирование (Unit testing)** — тесты, которые валидируют результаты отдельных методов.

  Модульные тесты обычно пишутся разработчиками. Но у нас на проекте эта команда всегда занята именно разработкой и нет соглашения о покрытии кода юнит-тестами. Считается, что юнит-тестирование — это хорошая практика, которая позволяет снизить технический долг и стоимость обслуживания системы в будущем. Внедрение же такого подхода, как всегда, это вопрос свободных ресурсов. Атомарность и изолированность методов API позволяет хорошо покрывать код тестами.

- **Тесты валидации.** В отличие от юнит-тестов, которые проверяют маленькие кусочки функциональности, валидационные тесты находятся на более высоком уровне и отвечают на набор вопросов, после ответа на которые программное обеспечение может перейти к следующему этапу разработки.

  Вопросы для валидации могут быть следующие:

  - Вопросы специфичные для продукта: “Это именно тот функционал, который был нужен?”
  - Вопросы ожидаемого поведения: “Этот метод ведет себя так, как ожидалось?”

- Вопросы, связанные с эффективностью: “Это метод использует ресурсы максимально независимо и эффективно?”

  Все эти вопросы служат для проверки API в разрезе согласованных критериев приемки. Еще они позволяют быть уверенным в соблюдении стандартов доставки ожидаемой конечной ценности и безупречном удовлетворение потребностей и требований пользователей.

Валидация в том виде, в котором она описана, является частью пользовательского тестирования. Доступ к методу включается на тестовом контуре после функционального тестирования и бизнес-подразделение проверяет его соответствие бизнес-требованиям. Например, корректность работы калькулятора, который использует API-вызовы через веб-интерфейс

- **Функциональные тесты.** Тесты, выполняющие конкретные методы API. Проверка количества активных пользователей через API, регрессионные тесты и выполнение тестовых случаев относятся к функциональным тестам.
- **E2E и UI тесты.** Тесты, которые запускают и проверяют сценарии конечного использования продукта, включая пользовательский интерфейс, API и БД. Проверяется результаты выполнения метода на каждом этапе транзакции.

  E2E проводится силами нескольких команд: Тестировщики фронта, тестировщики бэка, специалисты инфраструктурных технических отделов, ответственные за передаваемые данные. Предварительно лучше составить список тестов, который удовлетворит видению бизнеса. Можно взять и из плана тестирования, но там он часто избыточен. Этот список согласуется со всеми участниками e2e тестирования, чтобы к вам не было потом лишних вопросов.
- **Нагрузочное тестирование.** Увеличение количества пользователей не должно влиять на производительность приложения. Нагрузочное тестирование выявит проблемы производительности и доступности сервисов при масштабировании, а также проверит производительность API в нормальных условиях.

  Нагрузочное тестирование — ресурсоемкий процесс. У нас на проекте делается единожды перед сдачей очередной версии продукта, по просьбе заказчика или владельца.
- **Тесты на дефекты в процессе работы ПО.** Тесты, помогающие наблюдать за выполнением ПО и фиксировать проблемы, связанные с

неопределенностью параллелизма (race conditions), исключениями и утечкой ресурсов. Ниже содержится краткая информация об этих факторах.

- **Тесты мониторинга API**: Проверяют реализацию на различные ошибки, сбои внутренних обработчиков и другие неотъемлемые проблемы в кодовой базе API. Эти тесты гарантируют отсутствие дыр, которые могут привести к проблемам с безопасностью приложения.
- **Ошибки выполнения**: Проверка позитивных запросов к API на корректность ответа всегда присутствует в плане тестирования, однако проверка негативных сценариев не менее важная часть стратегии тестирования API.
- **Утечка ресурсов***. Негативные тесты для проверки сбоев в работе основных ресурсов API путем отправки некорректных запросов. Ресурсы здесь — это память, данные, уязвимости, операции тайм-аута и так далее.*
- **Отслеживание сбоев***. Обнаружение сбоев сетевого обмена. Сбои аутентификации из-за предоставления неверных учетных данных являются примером сценария обнаружения ошибок. Эти тесты гарантируют, что ошибки фиксируются, а затем устраняются.*

Контрактное тестирование или Consumer Driven Contract (CDC)

Тестирование контрактов гарантирует, что две стороны способны взаимодействовать друг с другом, путем проведения изолированной проверки того, поддерживают ли обе стороны сообщения, которыми обмениваются.

Одна сторона (потребитель) фиксирует связь с другой стороной (поставщиком) и создает контракт. Этот контракт представляет собой спецификацию запросов от потребителя и ответов от поставщика.

Код приложения автоматически генерирует контракты (в большинстве случаев это происходит на этапе модульного тестирования). Автоматическое создание гарантирует, что каждый контракт отражает актуальную реальность.

Когда потребитель опубликует контракт, им сможет воспользоваться поставщик. В своем коде (возможно, и в модульных тестах) поставщик выполняет проверку контракта и публикует результаты.

> На обоих этапах тестирования контракта мы работаем только с одной стороной, без какого-либо реального взаимодействия со второй. На практике мы следим, чтобы обе стороны могли общаться друг с другом по отдельным каналам. Таким образом, весь процесс остается асинхронным и независимым.

Если какой-либо из этих двух этапов завершится неудачей, то потребитель и поставщик должны договориться об устранении проблем с интеграцией. В некоторых случаях потребителю следует адаптировать интеграционный код, а в других — поставщику необходимо перенастроить API.

Контрактное тестирование — это **не** тестирование схемы. Тестирование схемы привязано к одной стороне без подключения к какой-либо другой. Контрактное тестирование проверяет взаимодействие с обеих сторон и гарантирует совместимость любых желаемых версий.

Контрактное тестирование — это **не** сквозное тестирование. В нем проводится комплексное тестирование группы совместно работающих сервисов, где тестируется вся система, начиная с пользовательского интерфейса и заканчивая хранилищем данных. Контрактное тестирование отдельно выполняет тесты для каждого сервиса. Они изолированы и не требуют запуска более одного единовременно.

Стратегия тестирования API. Функционал покрытия тестирования API

Стратегия тестирования - это высокоуровневое описание требований к тестированию, из которого впоследствии может быть составлен подробный план тестирования с указанием отдельных тестовых сценариев и тестовых случаев. Наша первая задача - это **функциональное тестирование**, чтобы убедиться, что API работает правильно.

Основными задачами функционального тестирования API являются:

- убедиться, что реализация API работает правильно, как и ожидалось - без ошибок!
- гарантировать, что реализация API работает в соответствии со спецификацией требований (которая позже становится нашей документацией по API).
- предотвратить регрессий между написанным кодом(merge) и релизом.

API как соглашение - сначала проверьте спецификацию!

API - это, по сути, соглашение между клиентом и сервером или между двумя приложениями. Перед тем, как начать любое тестирование функциональности, важно убедиться в правильности соглашения. Это можно сделать сначала проверив спецификацию (или само соглашение службы, например, интерфейс Swagger или ссылку OpenAPI) и убедившись, что:

- **эндпоинты правильно именованы**;
- **ресурсы и их типы правильно отражают объектную модель**;
- **нет отсутствующей или дублирующей функциональности**;
- **отношения между ресурсами правильно отражаются в API**.

Приведенные выше рекомендации применимы к любому API, но для простоты в этом посте мы предполагаем наиболее широко используемую архитектуру веб-API - **REST через HTTP**. Если ваш API спроектирован именно как RESTful API, важно убедиться, что **контракт REST действителен**, включая всю семантику, соглашения и принципы HTTP REST. (описание тут, тут, и здесь).

Если у вас общедоступный API, ориентированный на клиента, такое тестирование может быть вашим последним шансом убедиться, что все требования соглашения выполнены. После публикации и использования API любые внесенные вами изменения могут внести ошибки в код клиента. (Конечно, когда-нибудь вы сможете опубликовать новую версию API (например, /api/v2 /), но даже в этом случае обратная совместимость скорее всего будет обязательной).

Итак, какие аспекты API мы должны протестировать?

После того как мы проверили соглашение API, мы можем поразмышлять о том, что тестировать. Независимо от того, думаете ли вы об автоматизации тестирования или ручном тестировании, наши функциональные тест-кейсы имеют одинаковый набор **тестовых действий**. Они являются частью более широких **категорий тестовых сценариев** и их можно разделить на три **потока тестирования.**

Этапы тестирования API

Каждый тест состоит из тестовых шагов. Это отдельные атомарные действия, которые тест должен выполнять в каждом потоке тестирования API. Для каждого запроса API тест должен будет выполнить следующие действия:

1. **Проверьте корректность кода состояния HTTP**. Например, создание ресурса должно возвращать 201 CREATED, а запрещенные запросы должны возвращать 403 FORBIDDEN и т. Д.
2. **Проверьте полезную нагрузку ответа**. Проверьте правильность тела JSON, имен, типов и значений полей ответа, в том числе в ответах на ошибочные запросы.
3. **Проверьте заголовки ответа.** Заголовки HTTP-сервера влияют как на безопасность, так и на производительность.
4. **Проверьте правильность состояния приложения**. Это необязательно и применяется в основном к ручному тестированию или когда пользовательский интерфейс или другой интерфейс можно легко проверить.

5. **Проверьте базовую работоспособность**. Если операция была завершена успешно, но заняла неоправданно много времени, тест не пройден.

Категории тестовых сценариев

Наши тест-кейсы делятся на следующие общие группы *тестовых сценариев*:

- Основные позитивные тесты (прохождение успешного сценария по умолчанию)
- Расширенное позитивное тестирование с дополнительными параметрами
- Негативное тестирование с валидными входными данными
- Негативное тестирование с недопустимыми входными данными
- Деструктивное тестирование
- Тесты безопасности, авторизации и доступности (которые выходят за рамки этой статьи)

Тестирование успешного сценария по умолчанию проверяет базовую функциональность и критерии приемки API. Позже мы расширим положительные тесты, чтобы включить дополнительные параметры и дополнительные функции.

Следующая группа тестов - это **негативное тестирование**, при котором мы ожидаем, что приложение будет корректно обрабатывать проблемные сценарии как с валидным вводом пользователя (например, попытка добавить существующее имя пользователя), так и с недопустимым вводом пользователя (попытка добавить имя пользователя, которое имеет значение null).

Деструктивное тестирование - это более глубокая форма негативного тестирования, когда мы намеренно пытаемся сломать API, чтобы проверить его надежность (например, отправляя заведомо большое тело запроса в попытке переполнить систему).

Тестовые потоки

1. **Изолированное тестирование запросов** - выполнение одного запроса API и соответствующая проверка ответа. Такие базовые тесты - это

минимальные строительные блоки, с которых мы должны начинать. И нет смысла продолжать тестирование, если эти тесты упадут.

2. **Многоступенчатый рабочий поток с несколькими запросами** - тестирование серии запросов, которые являются обычными действиями пользователя, поскольку одни запросы могут зависеть от других. Например, мы выполняем запрос POST, который создает ресурс и возвращает автоматически сгенерированный идентификатор в своем ответе. Затем мы используем этот идентификатор, чтобы проверить, присутствует ли этот ресурс в списке элементов, полученных запросом GET. Затем мы используем PATCH для обновления новых данных и снова вызываем запрос GET для проверки этого обновления. И в завершении, мы УДАЛЯЕМ этот ресурс и снова используем GET, чтобы убедиться, что записи больше нет.
3. **Комбинированные тесты API и тесты веб-интерфейса** - это в основном относится к ручному тестированию, при котором мы хотим обеспечить целостность данных и согласованность между пользовательским интерфейсом и API.

   Мы выполняем запросы через API и проверяем действия через пользовательский интерфейс веб-приложения и наоборот. Цель этих потоков проверки целостности состоит в том, чтобы гарантировать, что, хотя на ресурсы влияют различные механизмы, система по-прежнему поддерживает ожидаемую целостность и согласованный поток данных.

Инструменты для тестирования API

1. SoapUI

SoapUI представляет собой консольный инструмент, предназначенный для тестирования API и позволяющий пользователям легко тестировать API REST и SOAP, а также Web-сервисы.

При помощи SoapUI, пользователи могут получить полный исходный документ и встроить предпочтительный набор функций, в дополнение к перечисленным ниже:

- Создать тест легко и быстро при помощи технологий перетаскивания объектов «мышкой» (Drag and drop) и метода «указания и щелчка» (Point-and-click)

- Быстро создать пользовательский код при помощи Groovy
- Мощное тестирование на основе данных: Данные загружаются из файлов, баз данных и Excel, поэтому они могут создать симуляцию взаимодействия пользователя и API.
- Создание комплексных сценариев и поддержка асинхронного тестирования
- Повторное использование скриптов: загрузка тестов и сканирование безопасности могут повторно использоваться в случае функционального тестирования всего в несколько шагов

2. Postman

Будучи изначально плагином браузера Chrome, теперь Postman расширяет свои технические решения вместе с оригинальными версиями как для Mac, так и для Windows.

Postman является отличным выбором API тестирования для тех, кто не желает иметь дела с кодировками в интегрированной среде разработки, используя тот же язык программирования, что и разработчик.

- Легкий в использовании клиент REST
- Богатый интерфейс делает этот инструмент простым и удобным в использовании
- Может использоваться как при автоматическом, так и при исследовательском тестировании
- Работает в приложениях для Mac, Windows, Linux и Chrome
- Обладает пакетом средств интеграции, таких как поддержка форматов Swagger и RAML
- Обладает функциями Run, Test, Document и Monitoring
- Не требует изучения нового языка программирования
- Позволяет пользователям легко делиться опытом и знаниями с другими членами команды, поскольку позволяет упаковывать все запросы и ожидаемые ответы и отправлять их коллегам.

3. Katalon Studio

Katalon Studio является бесплатным инструментом автоматического тестирования, предоставляющим общую среду для создания и выполнения UI функционала, служб API/Web и тестирования мобильных платформ.

Способность комбинировать уровни UI и Business (службы API/Web) для различных операционных сред (Windows, Mac OS, Linux) расценивается как значительное преимущество Katalon Studio перед аналогичными продуктами.

Katalon Studio поддерживает запросы SOAP и RESTful с различными типами команд (GET, POST, PUT, DELETE) с параметризованными возможностями.

Основные свойства:

- Поддержка теста комбинации между верификациями UI и API.
- Поддержка тестирования запросов как SOAP, так и RESTful.
- Сотни встроенных ключей для создания тестовых заданий.
- Поддержка одной из самых мощных библиотеки проверки утверждений AssertJ для создания динамических утверждений в BDD-стиле.
- Поддержка подхода, управляемого данными.
- Может использоваться как при автоматическом, так и при исследовательском тестировании.
- Отлично подходит как для профессионалов, так и для новичков

4. Tricentis Tosca

Tricentis Tosca представляет собой платформу непрерывного тестирования для Agile и DevOps. Среди преимуществ Tricentis Tosca следует отметить:

- Поддержку многих массивов протоколов: HTTP(s) JMS, AMQP, Rabbit MQ, TIBCO EMS, SOAP, REST, IBM MQ, NET TCP
- Интеграцию в циклы Agile и DevOps
- Максимизацию многократного использования и способность к сопровождению средств автоматизации тестирования на основе использования моделей

- Тесты API могут использоваться как на мобильных, так на браузерных и пакетных приложениях и т.д.
- Достигнута автоматизация, поддерживаемая новыми технологиями
- Снижено время, необходимое на проведение регрессивного тестирования

5. Apigee

Apigee является кросс-«облачным» средством тестирования API, позволяющим пользователям измерять и тестировать производительность API, обеспечивать техническую поддержку и разработку API при помощи других редакторов, таких как Swagger.

- Данный инструмент является многошаговым и находится под управлением Javascript
- Он позволяет разрабатывать, отслеживать, выполнять разворачивание и масштабирование API
- Идентифицирует проблемы путем отслеживания трафика API, уровня ошибок и времени ответа
- Легко создает прокси API из Open API Specification и выполняет их развертку в «облаке»
- Модели разворачивания в «облаке», локального разворачивания и гибридного разворачивания работают на основе одного кода
- PCI, HIPAA, SOC2, и PII для приложений и API
- Apigee разработан специально для цифрового бизнеса и задач с интенсивной обработкой данных на под управлением мобильных платформ API и приложений, которые управляют ним.

6. JMeter

JMeter (открытое программное обеспечение) широко используется для функционального тестирования API, однако изначально он создавался для нагрузочного тестирования.

- Поддерживает воспроизведение результатов тестирования
- Автоматически работает с файлами CSV, позволяя команде быстро создавать уникальные значения параметров для тестирования API.

- Благодаря интеграции между JMeter и Jenkins, пользователи могут включать тесты API в конвейерные обработки CI
- Данный инструмент может использоваться как для статического, так и динамического тестирования производительности ресурсов

7. Rest-Assured

Rest-Assured является общедоступным предметно-ориентированным Java-языком, который делает службу тестирования REST более простой и удобной.

- Обладает целым набором встроенных функционалов, наличие которых означает, что пользователю не придется кодировать заново.
- Надежно интегрирован со автоматизированной платформой Serenity, что позволяет пользователям комбинировать тесты UI и REST в рамках одной платформы и получать изумительные результаты.
- Поддерживает синтаксические конструкции BDD Given/When/Then
- Пользователям необязательно быть экспертами в области HTTP

8. Assertible

Assertible представляет собой инструмент тестирования API, который в первую очередь акцентируется на автоматизации процессов и надежности.

- Обеспечивает автоматическое тестирование API на каждом этапе процесса интеграции и поставки программного обеспечения.
- Осуществляет поддержку текущих тестов API после внедрения приложений и интегрирует их с привычными инструментами, такими как GitHub, Slack, и Zapier.
- Поддерживает проверку подлинности ответов HTTP при помощи готовых операторов подтверждения отсутствия ошибок, таких как проверка достоверности JSON Schema и проверка целостности данных JSON Path

9. Karate DSL

Karate DSL это новый инструмент для тестирования API, который помогает разрабатывать сценарии для BDD тестов на основе API простым способом, без написания характеристик этапов. Эти характеристики создаются самим KarateDSL, а поэтому пользователи могут запустить тестирование API легко и быстро.

- Встроен на вершине Cucumber-JVM
- Способен запускать тестирование и генерировать отчеты как любой другой стандартный проект Java
- Тест может быть разработан без обязательного владения знаниями языка Java
- Тесты могут легко создаваться даже непрограммистами
- Поддерживает конфигурацию продвижения /переключения, а также многопотоковое параллельное выполнение кода