Санкт-Петербургский государственный университет
Кафедра системного программирования

Автоматическая генерация интеграционных тестов для веб приложений

Артур Игоревич Алексеев, группа 21.Б11-мм

Научный руководитель: к. ф.-м. н. Д. А. Мордвинов, доцент кафедры системного программирования
Консультант: М. П. Костицын, инженер-исследователь, ООО «ИИнтелКод»
Рецензент: к. т. н. Д. С. Степанов, старший академический консультант, ООО «Техкомпания Хуавэй»

Санкт-Петербург
2025

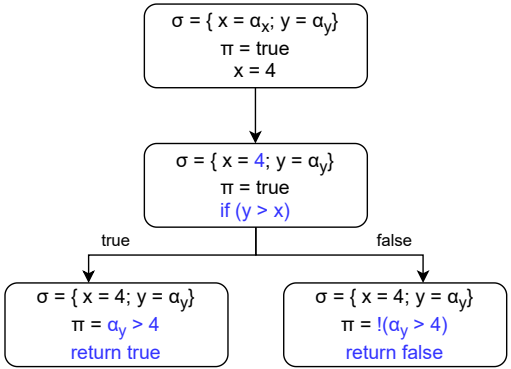
Введение. Символьное исполнение

Символьное исполнение — техника, которая позволяет моделировать исполнение программного кода в условиях неопределённости входных данных.

Символьное состояние:

- Текущая инструкция
- Память программы (σ)
- Ограничение условия пути (π)

```
bool target(int x, int y) {
    x = 4;
    if(y > x) return true;
    return false;
}
```



Введение. Символьное исполнение

Проблемы символьного исполнения:

- Требования к ресурсам: рост количества состояний, большое число шагов
- Трудность исследования внешних вызовов

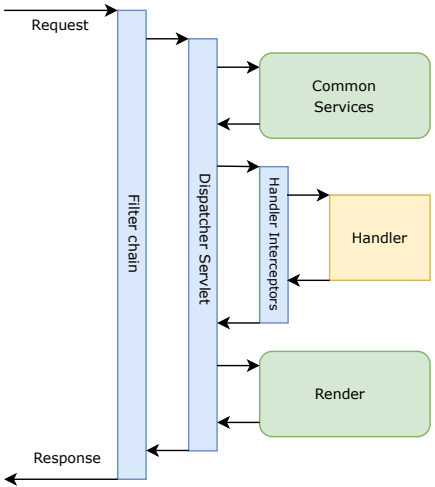
Подходы к решению проблемы:

- Динамическое символьное исполнение — комбинирование классического символьного анализа и реального (конкретного) исполнения
- Аппроксимации — упрощение сложных к исследованию участков кода

Введение. Фреймворк Spring

- Согласно опросу Developer Ecosystem, для разработки веб-приложений в экосистеме Java 72% программистов используют Spring Boot
- Удобная модель внедрения зависимостей
- Поддерживает MVC

Введение. Обработка запроса



Введение. Виды тестов

Модульные тесты:

- Проверяют корректность работы только одного модуля

Интеграционные тесты:

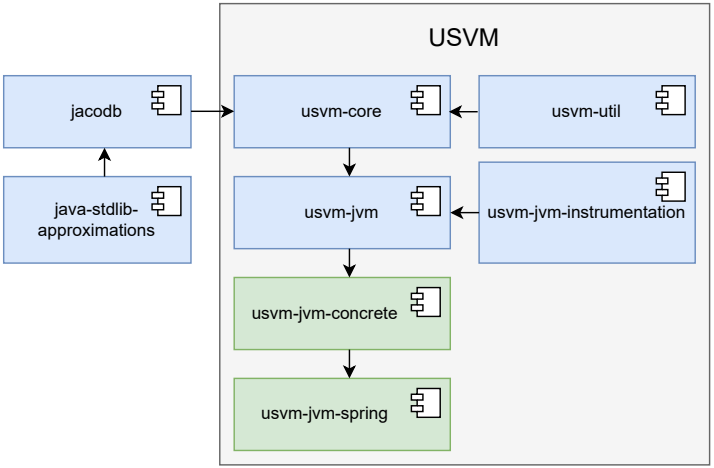
- Проверяют корректность работы системы целиком, от начала получения запроса до выдачи ответа

Компонентные тесты:

- Проверяют работу компонентов в изоляции, имеется возможность мокирования

Для создания компонентных или интеграционных тестов в фреймворке Spring существует `MockMvc`. Этот механизм позволяет начать обработку запроса без запуска сервера и работы с сетью.

Введение. USVM



Постановка задачи

Целью данной работы является разработка подсистемы для автоматической генерации интеграционных тестов для веб-приложений. Для достижения цели были сформулированы следующие **задачи**:

1. Выполнить обзор особенностей фреймворка Spring, техники символьного исполнения, проекта USVM и существующих решений в области автоматической генерации интеграционных тестов
2. Спроектировать архитектуру решения для автоматической генерации интеграционных тестов для веб-приложений
3. Реализовать спроектированное решение в проекте USVM
4. Провести сравнение результатов реализованной системы с аналогами

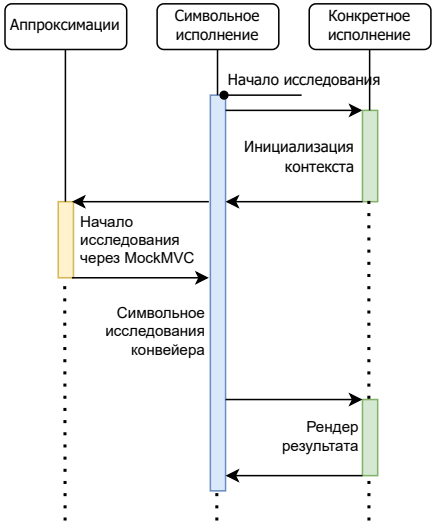
Существующие решения

- Инструмент UnitTestBot Java:
  - Плагин для IntelliJ IDEA
  - Фаззинг серого ящика для Spring-приложений
- RESTler, инструмент для фаззинга:
  - Тестирование веб-приложений методом чёрного ящика
  - Создаёт цепочки запросов
- EvoMaster, фаззинг веб-приложений:
  - Тестирование методом чёрного или белого ящика
  - Эволюционный алгоритм

Архитектура. Алгоритм

Ключевые особенности:

- Точка входа — искуственно созданный тестовый класс
- Инициализация исполняется конкретно
- После построения контекста исследование начинается через `MockMvc`
- Когда ветвление больше невозможно — происходит конкретизация



Архитектура. Конкретное исполнение

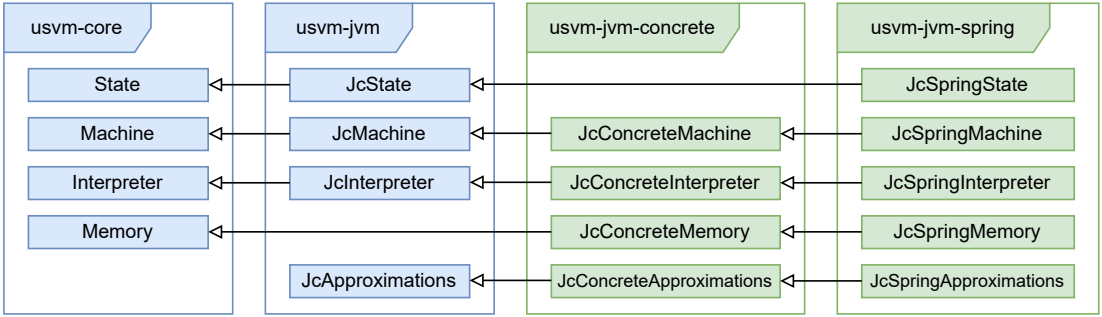
Конкретное исполнение — вызов реального метода с конкретными значениями.

Для конкретного исполнения необходимо иметь реальные объекты в памяти и уметь с ними работать.

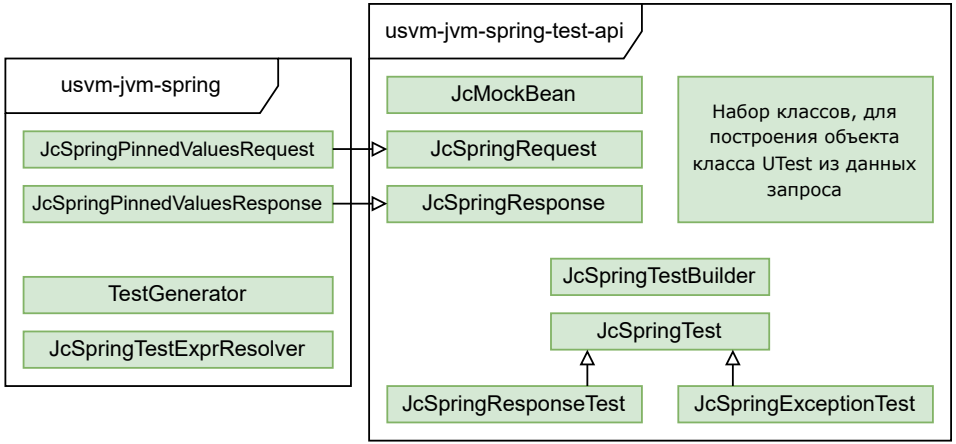
Конкретная память:

- Хранит объекты в виде настоящих Java-объектов
- Поддерживает конкретные вызовы
- Умеет конкретизироваться

Архитектура. Символьная машина



Архитектура. Генерация тестов из состояния



Архитектура. Работа с символьным запросом

Входные данные хранятся в символьном состоянии отдельно от запроса для обеспечения его конкретности.

Методы, ответственные за обработку входных данных, аппроксимируются:

- Методы вычисления аргументов метода контроллера (`ArgumentResolver`)
- Методы получения данных напрямую из запроса (`getHeader`)
- Методы работы с телом запроса: привязка модели, обработка JSON

Тестирование

Результаты тестирования синтетических бенчмарков (7 минут)

Инструмент	Покрытие, %	Кол-во тестов	Корректность, %
EvoMaster	46	50	90
UnitTestBot Java	40	55	92
USVM	70	267	87

Результаты тестирования проекта spring-petclinic (6 минут)

Инструмент	Покрытие, %	Кол-во тестов	Корректность, %
EvoMaster	23	26	100
UnitTestBot Java	46	63	98
USVM	61	64	78

Результаты

В ходе работы были получены следующие результаты:

- Выполнен обзор используемых подходов и символьного движка USVM, а также существующих решений в области генерации тестов: UnitTestBot Java, RESTler и EvoMaster
- Разработан алгоритм для символьного анализа веб-приложений на фреймворке Spring и спроектирована архитектура подсистемы, реализующей данный алгоритм в проекте USVM
- Реализована подсистема в USVM на языках программирования Kotlin и Java
- Проведено тестирование решения и выполнено сравнение с существующими аналогами

Характеристики тестового стенда

Тестирование проводилось на системе, которая имеет следующие характеристики.

- Операционная система — Microsoft Windows 10.
- Процессор — AMD Ryzen 5 5600X, 3.7 ГГц.
- Объем оперативной памяти — 32 Гб.