**Доклад**

**"Программная подсистема тестирования знаний языков описания аппаратуры"**

**Астахов Сергей ИУ6-82Б**

**Лист 1**

Добрый день, уважаемые члены государственной аттестационной комиссии. Вашему вниманию предлагается выпускная квалификационная работа бакалавра Астахова Сергея "Программная подсистема тестирования знаний языков описания аппаратуры".

Цель работы: разработать программную подсистему тестирования знаний языков описания аппаратуры, реализующую следующие функции:

- добавление, удаление, редактирование образовательных материалов и заданий (для администратора);
- проверка правильности решения заданий пользователями (в т.ч. заданий на написание программного кода);
- анализ ошибок в пользовательских решениях;
- занесение результатов решения в базу данных;
- анализ статистики решения заданий.

Решаемые задачи:

- проведение анализа существующих систем тестирования знаний и используемых в них методов тестирования знаний, составление списка методов тестирования знаний для использования в программной подсистеме тестирования знаний языков описания аппаратуры;
- определение вариантов использования программной подсистемы тестирования знаний языков описания аппаратуры;
- выбор архитектуры и разработка структуры подсистемы;

- проектирование и реализация компонентов подсистемы;
- разработка технологии тестирования подсистемы;
- проведение автономного и комплексного тестирования подсистемы;
- проведение нагрузочного тестирования подсистемы.

**Лист 2**

В рамках исследовательской части был проведен анализ существующих образовательных порталов, посвященных изучению языков программирования и языков описания аппаратуры.

В ходе первичного анализа существующих ресурсов, посвященных языкам описания аппаратуры, было выявлено, что, как правило, в них отсутствует такая важная функция, как автоматическая проверка заданий на написание исходного кода. Некоторые рассмотренные образовательные порталы имели возможность интеграции сторонних модулей, в том числе модулей для работы с языками описания аппаратуры, однако процесс подключения таких модулей весьма трудоемок, а функциональность образовательных порталов, в которые они интегрируются, зачастую не позволяет предоставить информативную обратную связь об ошибках учащегося.

Данный факт обуславливает актуальность разработки собственной программной подсистемы тестирования знаний языков описания аппаратуры.

Далее, был проведен анализ методов тестирования знаний, используемых в рассмотренных системах. В ходе анализа были выделены классы методов тестирования знаний, представленный в таблице в левой верхней части листа.

В ходе анализа были рассмотрены достоинства и недостатки различных методов тестирования знаний. Был сделан вывод, что использование тестов с перекрестной проверкой и проверкой

преподавателем нежелательно (когда можно избежать использования подобных тестов без значительной потери в эффективности оценки знаний), так как использование данных методов замедляет процесс оценивания, не позволяя сделать его полностью автоматическим. Тестирование на написание исходного кода с проверкой по референсным значениям не всегда возможно как таковое, требует от учащегося установки стороннего ПО (среды разработки и т.д.) и не дает информативной обратной связи, поэтому использование этого метода тестирования так же нежелательно.

В результате анализа были выделены наиболее часто используемые методы тестирования знаний, которые необходимо было реализовать в рамках разработанной подсистемы. Они приведены в таблице в левой нижней части листа.

Кроме того, в ходе анализа были уточнены функциональные требования к разработанной подсистеме и составлена диаграмма вариантов использования подсистемы, она представлена в правой части листа. Обычный пользователь может изучать теорию, проходить тестирования, а также просматривать персональную и общую статистику работы с образовательным порталом. Администратор, кроме того, может осуществлять управление образовательными материалами.

**Лист 3**

Работа в рамках конструкторской части была начата с разработки структурной схемы информационной системы в нотации C4, представленной на графическом листе. Структурная схема была разработана для двух уровней модели C4: уровня контекста и уровня контейнеров.

Диаграмма уровня контекста показывает место разработанной подсистемы в информационной системе. Подсистема используется в рамках информационной системы образовательного портала, обращение к ней происходит из веб-приложения образовательного портала, реализующего такие функции, как предоставление графического интерфейса, управление учетными записями пользователей и др. Поскольку некоторые данные используются и веб-приложением и разработанной подсистемой, база данных используется совместно.

Диаграмма уровня контейнеров демонстрирует внутреннюю структуру подсистемы. Так как разработанная подсистема является весьма сложной, выполняемые в ней операции разнородны, могут потребовать использования различных языков и библиотек, а также некоторые из них могут занимать значительное время, при разработке было решено использовать микросервисную архитектуру.

Микросервисный подход обладает следующими преимуществами:

- позволяет использовать различные языки программирования для реализации различных компонентов;
- упрощает тестирование;
- позволяет использовать горизонтальное масштабирование для различных компонентов в зависимости от нагрузки на них.

На основе функциональных требований, предъявляемых к разработанной подсистеме и представленной диаграммы вариантов использования была разработана структура микросервисов, отвечающих за их реализацию:

- микросервис взаимодействия с БД — реализует операции над данными в БД;

- микросервис анализа решений (анализатор) — выполняет проверку и анализ пользовательских решений;
- микросервис синтеза устройств (синтезатор) — выполняет синтез устройств из Verilog-кода и симулирует их работу;
- микросервис разбора временных диаграмм, микросервис генерации временных диаграмм wavedrom — преобразуют временные диаграммы в удобные для хранения и обработки форматы;
- микросервис анализа статистики;
- основной микросервис — реализует верхнеуровневую логику подсистемы, связывает остальные микросервисы.

Большая часть микросервисов реализована на языке Golang.

**Лист 4**

Следующим шагом работы стала разработка даталогической схемы базы данных. После первичного анализа предметной области были выделены сущности "Пользователь", "Задание" и "Попытка решения". В дальнейшем поля сущности "Задание", требуемые для расчета статистических показателей и списка заданий, и поля, требуемые только при непосредственном просмотре и решении конкретного задания были разделены на две отдельные сущности, типы заданий были вынесены в отдельную таблицу. Так как в подсистеме используются задания различных типов, данные, необходимые для проверки задания, хранятся в сериализованном виде.

Далее, необходимо было разработать микросервис, осуществляющий работу с базой данных. Диаграмма компоновки этого микросервиса показана в правой части листа. Взаимодействие с каждой таблицей БД происходит через соответствующий класс, для доступа к базе данных используется MySqlDriver.

**Лист 5**

Теперь перейдем к диаграмме классов микросервиса взаимодействия с базой данных. Когда в микросервис поступает запрос на осуществление операций с базой данных, его тело распаковывается в структуру объект класс EncasultedMetaInfo, из которого можно считать тип запрашиваемой операции и таблицу, над которой она должна быть осуществлена. Затем, на основе информации о таблице, тело запроса повторно распаковывается в соответствующий тип, его указатель записывается в специальную переменную, хранящую нетипизированный указатель. После чего нетипизированный указатель преобразуется в указатель на соответствующий требуемой операции интерфейс, вызывается необходимая функция. Если функция возвращает результат (read, readAll), то он записывается в тело ответа через операцию сериализации нетипизированного указателя.

**Лист 6**

Теперь обратимся непосредственно к процессу проверки заданий на написание исходного кода.

При проверки таких заданий все начинается с того, что код описания устройства, написанный пользователем, и код тестов из базы данных поступают в микросервис синтеза устройств. В нем происходит симуляция работы устройства и генерация временной диаграммы его работы в формате VCD (что расшифровывается, как “value change dump”). Симуляция работы устройства происходит за счет программного обеспечения Icarus Verilog, которое помещено в единый docker-контейнер с микросервисом, написанном на языке Golang, являющимся оберткой для взаимодействия с сетью и организации хранения временных файлов с исходными кодами.

После этого временная диаграмма работы устройства поступает в микросервис разбора временных диаграмм, написанный на языке Python, который преобразует временную диаграмму в удобный для дальнейшей обработки формат JSON с помощью библиотеки PyDigitalWaveTools. Далее, микросервис генерации временных диаграмм wavedrom преобразует полученный JSON в формат, совместимый с движком визуализации временных диаграмм wavedrom (который позволяет визуализировать временные диаграммы прямо в веб-интерфейсе с помощью JavaScript).

(Стоит отметить, что при редактировании заданий администратором происходит аналогичный процесс симуляции работы устройства и в базу данных вносится уже временная диаграмма в wavedrom-совместимом формате).

Затем происходит обращение к микросервису анализа решений. Микросервис определяет тип задания и вызывает функцию его проверки аналогично тому, как происходит работа с объектами в микросервисе взаимодействия с базой данных. Затем результаты проверки заносятся в базу данных.

**Лист 7**

Теперь перейдем к технологической части работы. В рамках технологической части работы была разработана технология тестирования, проведено автономное, комплексное тестирование и нагрузочное тестирование.

Список тестов для автономного и комплексного тестирования был сформирован на основе подхода черного ящика, в частности метода классов эквивалентности. Количество тестов на каждый микросервис отражено в таблице в левой верхней части листа.

Тестирование было автоматизировано за счет библиотеки Pytest, а по его результатам с помощью библиотеки allure от компании “Яндекс” был сформирован отчет, предоставляющий подробную информацию о ходе тестирования через веб-интерфейс.

Нагрузочное тестирование было проведено для основного микросервиса, микросервиса взаимодействия с базой данных, микросервиса синтеза устройств и микросервиса разбора временных диаграмм. Для среднего времени запроса были заданы максимальные пределы от 300 мс до 1 с. Так как конфигурация использованного оборудования отличалась от требуемой в ТЗ, была установлена предельная нагрузка в 100 пользователей.

Все виды тестирования завершились успешно.

**Послесловие**

В результате проделанной бакалаврской работы спроектирована и реализована программная подсистема тестирования знаний языков описания аппаратуры. Была разработана технология ее тестирования, проведено автономное, комплексное и нагрузочное тестирование подсистемы. Результаты тестирования показали работоспособности подсистемы.

Спасибо за внимание.

Автор выступления - Астазов Сергей, группа ИУ6-82Б