Санкт-Петербургский государственный университет

Программная инженерия

Слесарев Илья Дмитриевич

Разработка автоматизированной системы проверки качества JavaScript-кода

Выпускная квалификационная работа

Научный руководитель:
ст. преп. Сартасов С. Ю.

Рецензент:
инженер-консультант ООО "САП Лабс" Забус Е. Р.

Санкт-Петербург
2018

SAINT-PETERSBURG STATE UNIVERSITY

Software engineering

Ilya Slesarev

Development of automated system for checking JavaScript code quality

Graduation Thesis

Scientific supervisor:
assistant prof. Stanislav Sartasov

Reviewer:
Engineer at SAP Labs Eugeny Zabus

Saint-Petersburg
2018

Оглавление

Введение

«Хороший код», «качество кода» – это понятия, которые имеют как довольно общее определение, так и определенную специфику в рамках компании или даже конкретного проекта. Некачественный код характеризуется смешением стилей программирования, это приводит к усложнению разработки. Для того, чтобы избежать смешения разных стилей написания кода в одном проекте, существуют руководства, описывающие общие принципы работы с кодом. Такие документы могут быть разработаны в масштабе компании, рабочей группы или проекта. Обычно в них описываются общеизвестные аспекты, например:

- рекомендации по использованию регистров в названиях переменных, методов или классов, такие как CamelCase, snake_case, kebab-case [4];
- запрет на использование функции вывода в консоль для приложений, исполняемых в браузере;
- рекомендации по расстановке скобок, ограничивающих логические блоки.

В частных случаях, в рамках конкретных проектов, руководства могут содержать и другие рекомендации:

- запрет на использование "магических" констант: цветов "#42f4d9" или чисел "54362";
- запрет на использование url-ссылок на ресурсы, не входящие в список разрешенных;
- ограничение на длину и количество строк в файле с исходным кодом.

Если программисты не следуют стандартам, читаемость кода ухудшается, появляются проблемы с безопасностью, а время, затрачиваемое на разработку, увеличивается. Например, использование печати в консоль в веб-приложении может привести к тому, что пользователь увидит информацию, которая ему не предназначена. Неиспользование

точки с запятой в конце строк JavaScript-кода может привести к неопределенному поведению программы.

В каждой языковой среде существуют свои правила и практики разработки. В рамках проекта рассматривается написание приложений на языке JavaScript с использованием закрытого фреймворка SAP UI5 [9] – это набор инструментов, основанный на технологиях HTML5, CSS3, OData, поддерживающий разработку как мобильных, так и веб-приложений. SAP UI5 используется для создания сложных бизнес решений с множеством данных и сервисов, вследствие этого были разработаны наборы правил для стандартизации клиентского кода. Но при разработке некоторые правила нарушаются и игнорируются, таким образом возникла задача проверки качества клиентского кода в проектах.

Проверку кода можно производить с помощью статического [11] и динамического анализа [2]. Для статического анализа не требуется запуск программы, с помощью него можно обнаружить неопределенное поведение или ошибки в повторяющемся коде. Динамический анализ требует запуск программы, но дополнительно помогает найти программные ошибки, определить используемые ресурсы. Статический анализ кода встроен в продукт компании SAP Web IDE [8], однако зачастую клиенты могут игнорировать замечания, пользоваться устаревшей версией Web IDE, где не была встроена проверка или использовать другую среду разработки. Впоследствии необходимо вручную проводить проверки, чтобы обнаружить блоки кода, несоответствующие описанным стандартам.

Существует множество программных решений для выполнения автоматической проверки JavaScript-кода, например, ESLint [17], JsInspect [12]. Однако в ситуации, когда клиент компании просит сделать оценку качества кода, необходимо произвести многочисленные действия, например:

- применить набор инструментов;
- свести результаты в один отчет и выдать его клиенту;
- учесть особенности продукта клиента (к примеру, использование

SAP UI5).

Поэтому возникает необходимость в разработке системы, которая позволила бы оптимизировать и ускорить процесс оценки качества кода, выделить основные проблемы, наметить пути решения и обладала расширяемостью, необходимой для увеличения функциональности проверок. Таким образом моей задачей стала автоматизация процесса, который ранее выполнялся вручную не оптимально по затрачиваемым ресурсам.

1. Постановка задачи

Целью работы является разработка прототипа системы автоматизированной проверки качества JavaScript-кода в контексте проекта компании SAP. Для достижения данной цели были поставлены следующие задачи.

- Разработать требования к системе оценки качества кода.
- Сделать обзор существующих программных решений для анализа качества кода.
- Разработать архитектуру прототипа для автоматизированных проверок.
- Реализовать прототип.
- Провести апробацию прототипа.

2. Требования к системе

В разделе описываются требования с обсуждением причин, по которым они были выдвинуты. Требования классифицированы на функциональные и нефункциональные.

2.1. Функциональные требования

2.1.1. Функциональность проверки

Проверка должна включать в себя общие положения из руководств по JavaScript-коду, а также положения, разработанные для фреймворка SAP UI5. Необходимо также предусмотреть возможность отключения проверок по конкретным правилам из-за специфики конкретных проектов.

2.1.2. Расширяемость

В рамках проекта рассматривается проверка исключительно JavaScript-кода, но должна быть предусмотрена возможность расширения системы:

- динамической проверкой кода;
- дополнительными статическими проверками.

Необходимо создать удобный инструментарий для внедрения новых правил, специфичных для фреймворка SAP UI5.

2.1.3. Графический веб-интерфейс

Основная задача разрабатываемого прототипа системы – предоставить разработчикам компании SAP возможность без лишних затрат времени и сил провести анализ клиентского кода. В связи с этим система должна обладать графическим веб-интерфейсом, который после загрузки кода проектов будет отображать статистический отчет в таблицах, а также отчет о конкретных местах кода, непроходящих проверки.

2.1.3.1. Таблицы с количественными данными

Данные о количестве найденных ситуаций в коде должны быть разбиты на две таблицы одного формата по названиям колонок. Первая таблица должна включать в себя все правила, разработанные специально для фреймворка SAP UI5, вторая должна включать в себя правила, специфичные для языка программирования JavaScript.

2.1.3.2. Таблицы с данными о положении ситуации в коде

Результаты проверки с указанием местонахождения ситуации в коде являются наиболее важными из всех результатов работы системы, вследствие этого интерфейс для получения таких данных должен быть простым и понятным. Таблица должна иметь следующие поля:

- путь к файлу;
- название правила;
- строка и столбец начала ситуации;
- строка и столбец конца ситуации;
- исчерпывающее сообщение о ситуации;
- важность ситуации:
  - предупреждение;
  - ошибка.

2.1.3.3. Отчет о ситуациях по конкретному правилу

Необходимо предусмотреть возможность получения отчета по выбранному правилу. Файл отчета должен содержать набор ситуаций вместе с блоками кода, где были обнаружены ситуации.

2.1.4. Загрузка кода проектов

- Необходимо реализовать возможность загрузки кода двумя способами:

  - непосредственно через загрузку файлов в систему;
  - через указание url-ссылки на репозиторий проекта.

- Необходимо реализовать возможность анализа нескольких проектов за одну итерацию, для получения более объективной статистики.

2.2. Нефункциональные требования

2.2.1. Ограничение на платформу для реализации

Во многих компаниях, в том числе SAP, на компьютерах сотрудников установлен корпоративный образ операционной системы. Ввиду этого требуется реализация поддержки операционных систем Windows и MacOS. Разработка должна вестить на языке JavaScript. Веб-интерфейс должен быть реализован в виде fiori приложения [10] для приведения прототипа системы к стандарту приложений компании SAP, а также для реализации кроссплатформенности решения.

2.2.2. Тестирование прототипы системы

Первым этапом тестирования должна быть апробация прототипа на некоммерческих, внутренних проектах компании для оценки качества работы прототипы. Последующим этапом необходимо провести тестирование системы на реальных проектах.

3. Обзор предметной области

3.1. Обзор существующих подходов и решений

В мире современной разработки программного обеспечения существует множество решений для статического и динамического анализа кода. Универсального подхода для этой процедуры пока не выработано, потому что нет единых стандартов. Существуют компании, которые занимаются исключительно оценкой качества клиентского кода, они могут применять собственные инструменты для анализа и передавать клиенту конечный отчет, или продавать в пользование клиенту свои инструменты. Таким образом можно выделить основные типы существующих решений:

- анализ кода с помощью встроенного в среду разработки инструмента;
- передача программного кода сторонней компании для анализа;
- анализ программного кода собственным (или купленным) инструментом компании.

В соответствии с требованиями к процессу анализа кода был выбран подход реализации собственного инструмента на базе существующих.

3.2. Обзор инструментов

3.2.1. Инструмент ESLint

ESLint [17] - это утилита, которая позволяет проводить анализ качества кода, написанного на любом выбранном стандарте языка программирования JavaScript [6]. С помощью него можно не только привести код к единому стилю, но и найти места, где код отличается от описанного в конфигурации стиля. Также обладает отличной интеграцией с различными инструментами разработки, например средой программирования WebStorm [3].

ESLint обладает достаточно гибкой структурой, и вследствие этого программисты имеют возможность делать "надстройки" над инструментом в виде плагинов или дополнительных правил обработки кода.

3.2.2. Инструмент JsLint

JsLint [7] - это узкоспециализированный инструмент для анализа качества JavaScript кода. Данное решение просто в использовании, но в свою очередь не обладает такой расширяемостью как ESLint: вместо файла с множеством настроек достаточно запустить одну команду и получить отчет о слабых местах в коде. Использование этого инструмента исключительно полезно для небольших проектов, чтобы разработчики не тратили время на настройку и отладку более мощных решений, таких как ESLint.

3.2.3. Инструмент JsInspect

Дублирование кода - это часто встречающаяся проблема даже у опытных программистов. Если разработчик не имеет достаточно опыта в проектировании отдельных модулей или системы в целом, или архитектура проекта вовсе принадлежит другому человеку, то зачастую в коде встречаются повторы некоторых блоков, методов или даже целых классов. Для поиска таких ситуация используется инструмент jsInspect [12]. Принцип его работы заключается в анализе синтаксического дерева кода [1] с определенным порогом, определяющим наименьшее подмножество узлов для анализа. Количество узлов, как и другие параметры для анализа, указываются в файле настроек, например, можно игнорировать одинаковые названия свойств объектов.

3.2.4. Инструмент JsPrettier

JsPrettier [13] - это инструмент для улучшения качества JavaScript-кода, но непосредственно для анализа его использование невозможно. Основной задачей инструмента является приведение кода к стандарту, заранее описанному разработчиками пакета. JsPrettier поддерживается

любыми операционными системами, в свою очередь возможность интеграции имеется только в редактор текста Sublime [5], что доставляет неудобство, если программист работает в других средах разработки программного обеспечения.

3.2.5. Среда разработки Web IDE

SAP Web IDE - Web-среда разработки, существующая для упрощения работы с интерфейсами, бизнес-логикой и базами данных, а также SAP Fiori [10] приложениями. Встроенные в среду разработки плагины проводят статический анализ кода приложения. Такой подход обладает своими достоинствами и недостатками, к примеру: плагины для статического анализа не расширяемы - разработчик не имеет возможности добавить дополнительных проверок; плагины не переносимы на другие редакторы кода. С другой стороны плагины не требуют времени и усилий программиста для настройки, анализ производится «из коробки».

3.3. Выводы

Таблица 1: Преимущества и недостатки существующих инструментов

	Анализ кода	Возможность интеграции	Простота настройки	Устранение проблем	Расширяемость
ESLint	+	+	−	+	+
JsLint	+	+	+	+	−
JsInspect	+	+	−	−	−
JsPrettier	−	+	+	−	−
Web IDE	+	−	+	+/−	−

Таким образом, в результате анализа составленной таблицы 1 за основу был взят инструмент ESLint. Расширяемость данного решения

позволила разработать прототип, который удовлетворяет узкоспециализированным требованиям к системе. Также в совокупности к ESLint применение инструмента JsInspect позволило улучшить качество анализа из-за возможности определить дублирование программного кода.

4. Архитектура системы и особенности реализации

С учетом требований и анализа существующих решений, была разработана базовая архитектура системы в виде модулей (рис. 1). Такая архитектура позволила получить следующие возможности расширения продукта:

- горизонтальное развитие:
  - добавление обновленных правил в проверку кода;
  - добавление утилит для статических проверок;
  - получение различных статистик по проектам;
- вертикальное развитие:
  - добавление динамического анализа проектов.

Внутренняя логика системы реализована в виде трех основных модулей: получение кода от клиента, обработка кода различными инструментами, обработка результатов проверки для генерации отчетов.

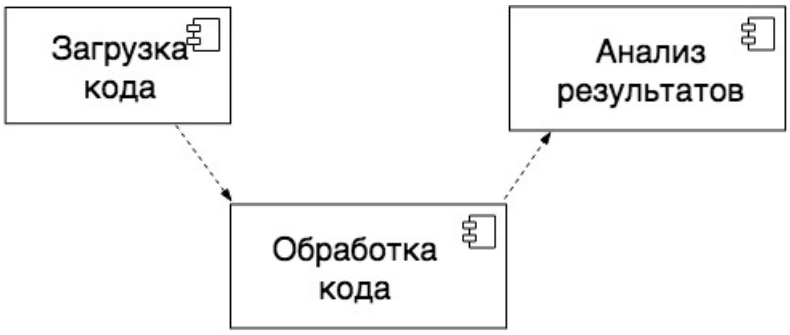

Рис. 1: Архитектура системы

Таким образом файлы JavaScript-кода, полученные из первого модуля, обрабатываются в модуле работы с кодом, и впоследствии по выбранным критериям проводится анализ результатов и создаются отчеты в виде графиков и таблиц.

4.1. Модуль графического интерфейса

С учетом поставленных требований для загрузки проектов был разработан прототип графического веб-интерфейса системы. Полученные по ссылке или непосредственной загрузкой файлы проектов передаются в модуль работы с кодом. В элементе интерфейса checkbox отображаются правила, которые сработали при проверке загруженного кода. Для вывода отчета о количестве сработавших правил, выбираются только те, которые интересуют клиента. Для каждого правила печатается название и количество срабатываний. Суммарно выводится количество проверенных файлов и строк кода для оценки процентного содержания "некачественного кода".

4.2. Модуль работы с кодом

Модуль обработки JavaScript-кода состоит из двух основных компонентов:

- поиск блоков повторяющегося кода;
- анализ кода по определенным шаблонам.

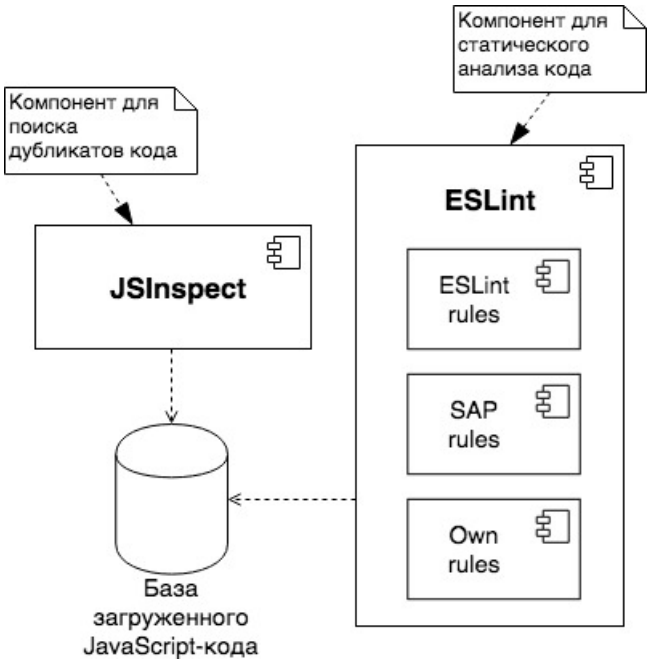

Рис. 2: Обработка кода

Базой для этого модуля стал инструмент ESLint [17]. Основным пре-

имуществом выбранного подхода является возможность расширять набор правил для обработки кода, а также подключать дополнительные инструменты.

В библиотеке инструмента ESLint находится набор правил для проверки кода, которые можно включать или отключать по своему усмотрению. Наборы правил объединяются в конфигурации, такие как "google" [16] или "eslint:recommended". Также в компании SAP в 2015 году был разработан набор правил, направленных на специфику программирования на языке JavaScript в фреймворке SAP UI5. Однако с того времени одни правила потеряли свою актуальность, а другие требуют доработки. ESLint представляет исходный код программы в виде абстрактного синтаксического дерева (АСД) [1] – это помеченное ориентированное дерево, в котором вершины сопоставлены с операторами языка программирования, а листья – с операндами. Все операторы (и соответственно вершины в дереве) имеют свои типы, например, "IfStatement" у условного оператора или "Program" у корневой вершины. А каждое правило, разработанное для инструмента ESLint, представляется в виде объекта с полями, названия которых – типы операторов, а определение – функции проверки кода. Таким образом при запуске утилиты ESLint для каждой вершины проверяются только те правила, в которых был указан тип конкретно этой вершины, а не все правила из набора.

Для работы с собственными правилами реализованы: обход в ширину [15] для взаимодействия вершин АСД на одном уровне вложенности и обход в глубину [14] – на разных уровнях вложенности. Это не тривиальная задача ввиду того, что каждый тип вершины обладает своей структурой, а значит хранит ссылки на следующие вершины в разном виде. Но для того, чтобы найти части кода, которые используют устаревшие библиотеки и стили оформления, достаточно описать поведение для вершин типа "VariableDeclaration". Список собственных правил расширяем и на данный момент представляет собой следующие:

- запрет на использование библиотек и стилей оформления из списка устаревших;

- запрет на повторное объявление констант.

Дополнительно для улучшения качества анализа кода применяется инструмент JSInspect. В конфигурации задается сложность проверки – порог количества вершин, начиная от которого утилита считает, что блок кода повторяется. Например, если два логических блока идентичны и их размер составляет 15 вершин, а порог установлен 20, то такая ситуация не будет считаться дублированием кода. JSInspect также работает с АСД, однако интерфейсом для дополнительной настройки инструмент не обладает, поэтому результат может быть получен только в виде полного отчета о повторяющихся блоках кода.

4.3. Модуль обработки результатов проверки

Рис. 3: Обработка результатов проверки

Данные, полученные после проверки кода, передаются в третий модуль, где проводится анализ результатов, генерация отчетов и графиков. В зависимости от полученных требований к проверке конкретного проекта, генерируются отчеты:

- о процентном содержании повторяющегося кода;
- о количестве объявленных, но не использующихся переменных;
- о количестве явно объявленных цветов, url-ссылок и другие.

В конечном итоге предоставляется отчет о всех случаях, определенных как ”некачественный код”, с указанием имени файла, строки и столб-

ца, где была обнаружена ситуация, а также следующими полями для обозначения найденных правил:

- строгость (ошибка или предупреждение) правила;
- название правила;
- типом вершины, для которой сработало правило;
- сообщением об ошибке.

5. Апробация прототипа

Система была протестирована на 35 проектах, разработанных с использованием фреймворка SAP UI5. Для подсчета общего количества файлов с JavaScript-кодом и количества строк кода использована утилита cloc [18]. В 264 файлах содержится:

- 7500 пустых строк;
- 8200 строк комментариев;
- 49600 строк кода.

Полученная статистика по наиболее часто сработавшим правилам изложена в табл. 2 и 3. В таблице правил, специфичных для фреймворка, выделены только те, для которых количество ситуаций привысило 40. Это не означает, что замечания по другим правилам менее важны. Например, правило "sap-no-hardcoded-color" сработало 15 раз, значит разработчик 15 раз нарушил стиль оформления, что впоследствии может привести к ошибке.

Таблица 2: Набор правил для фреймворка SAP UI5 с частотой срабатывания больше 40

Название правила	Количество срабатываний
sap-no-ui5base-prop	150
sap-no-dom-access	116
sap-no-dom-insertion	65
sap-no-element-creation	50
sap-cross-application-navigation	46

Таблица 3: Набор общих правил для языка JavaScript с частотой срабатывания больше 300

Название правила	Количество срабатываний
no-undef	838
camelcase	616
valid-jsdoc	565
eqeqeq	387
no-extra-semi	307

На основе полученных результатов после работы прототипа системы, сделан вывод о том, что частоты срабатывания правил, разработанных для фреймворка, на порядок меньше частот для правил специфичных для языка JavaScript, однако правила для фреймворка более полезны в коммерческой разработке. Большинство из них разработаны написаны таким образом, что их игнорирование неминуемо приведет к ошибке на каком-либо из этапов разработки приложения.

Заключение

В ходе выполнения выпускной квалификационной работы были достигнуты следующие результаты.

- Разработаны требования к системе.
- Сделан обзор существующих решений для анализа качества кода.
- Разработана архитектура системы, основанная на принципе простоты расширяемости.
- Реализован прототип системы, включающий следующие компоненты:
  - набор уникальных правил для оценки клиентского кода;
  - набор скриптов для анализа качества кода;
  - графический веб-интерфейс, выполненный с использованием инструмента SAP UI5.
- Апробация продукта проведена на 35 проектах, разработанных с использованием SAP UI5.

Список литературы

[1] Abstract syntax tree. — URL: https://goo.gl/NaLywu (online; accessed: 13.04.2018).

[2] Dynamic program analysis. — URL: https://goo.gl/RenpKs (online; accessed: 26.04.2018).

[3] JetBrains. WebStorm IDE. — URL: https://www.jetbrains.com/webstorm/ (online; accessed: 13.04.2018).

[4] Letter case. — URL: https://goo.gl/1AEW8n (online; accessed: 7.05.2018).

[5] Ltd Sublime HQ Pty. Sublime. — URL: https://www.sublimetext.com/ (online; accessed: 13.04.2018).

[6] Morelli Brandon. Different between ES6, ES8, ES 2017, ECMAScript. — URL: https://goo.gl/hNwTZt (online; accessed: 24.04.2018).

[7] Reid. JSLint. — URL: https://github.com/reid/node-jslint (online; accessed: 13.04.2018).

[8] SAP. Web IDE. — URL: https://www.sap.com/developer/topics/sap-webide.html (online; accessed: 10.04.2018).

[9] SAP. Фреймворк SAPUI5. — URL: https://sapui5.hana.ondemand.com/ (online; accessed: 10.04.2018).

[10] SAP Fiori apps. — URL: https://www.sap.com/products/fiori.htmls (online; accessed: 26.04.2018).

[11] Static program analysis. — URL: https://goo.gl/vSo9JG (online; accessed: 26.04.2018).

[12] danielstjules. Jsinspect. — URL: github.com/danielstjules/jsinspect (online; accessed: 10.04.2018).

[13] jonlabelle. JsPrettier. — URL: github.com/jonlabelle/SublimeJsPrettier (online; accessed: 10.04.2018).

[14] Алгоритм обхода в глубину. — URL: http://e-maxx.ru/algo/dfs (online; accessed: 8.05.2018).

[15] Алгоритм обхода в ширину. — URL: http://e-maxx.ru/algo/bfs (online; accessed: 8.05.2018).

[16] Рекомендуемая google конфигурация ESLint. — URL: https://github.com/google/eslint-config-google (online; accessed: 8.05.2018).

[17] Сайт проекта ESLint. — URL: https://eslint.org (online; accessed: 10.04.2018).

[18] Утилита для посчета количества файлов и строк кода. — URL: https://github.com/kentcdodds/cloc (online; accessed: 9.05.2018).