* методологию «разработки через поведение» (behaviour-driven development, BDD), основанную на гибкой методологии разработки.
* *для бизнеса удобно смоделировать систему, в которой будет определен единый язык (Domain specific language, DSL), основанный на бизнес области, такой, что бизнес словарь смогут использовать как менеджеры, так и инженеры.*
* В данный момент производители АС пытаются найти решение, предоставляющее возможность описывать тестовые сценарии в форме, оговариваемой внутри команды, состоящей заказчика, аналитика, разработчика и тестировщика.

# ANTLR4

Данный генератор парсеров является [LL(\*)](https://ru.wikipedia.org/wiki/LL-%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80)-анализатором.,

LL-анализатор называется LL(\*)-анализатором, если нет строгого ограничения для k и анализатор может распознавать язык, если токены принадлежат какому-либо регулярному множеству (например, используя детерминированные конечные автоматы).

 в 4-й версии значительна улучшена производительность парсинга благодаря использованию алгоритма Adaptive LL(\*). Данный алгоритм **совмещает** **преимущества** относительно медленных и непредсказуемых **алгоритмов** **GLL и**[**GLR**](https://en.wikipedia.org/wiki/GLR_parser), которые, однако, *способны разрешать случаи с неоднозначностью* (используются при разборе естественных языков), **и** стандартных**, быстрых LL-алгоритмов рекурсивного спуска**, которые, в свою очередь, не *способны разрешать все задачи с неоднозначностью*.

Суть алгоритма заключается в псевдопараллельном запуске LL-парсеров на каждом правиле, их кэшировании и выборе подходящего предсказания (в отличие от GLR, где допустимо несколько альтернатив). Таким образом, алгоритм является динамическим. Несмотря на то, что наихудшая теоретическая сложность алгоритма является O(n4), по факту скорость парсинга для существующих языков программирования является линейной. Также в четвертой версии сильно эволюционировала возможность восстановления процесса парсинга после обнаружения синтаксических ошибок.

# Реализация языка

**XL Driver Library**

Для реализации функционала симуляции автомобильных сетей используется универсальный программный интерфейс XL-Driver-Library, позволяющий получить доступ к интерфейсам аппаратных средств Vector.

Для разработки XL Driver Library приложений требуется подключить динамические библиотеки, которые находятся в открытом доступе на сайте компании Vector. Необходимые методы реализованы на языке С, поэтому для доступа к методам через java был реализован класс JNIVxlApi.java c использованием механизма Java Native Interface .

Java Native Interface (JNI) — стандартный механизм для запуска кода, под управлением виртуальной машины Java (JVM), который написан на языках С/С++ или Ассемблера, и скомпонован в виде динамических библиотек. Это даёт возможность вызова функции С/С++ из программы на Java, и наоборот.

Пользовательская библиотека наследует класс, который содержит методы, предоставляющие возможность использовать XL Driver Library.

Для использования разработанного парсера ANTLR4 представляет возможность сгенерировать шаблон проектирования Listener (слушатель). Который предполагает анализ определенного подмножества узлов дерева разбора. Узлы дерева разбора, не являющиеся листьями, соответствуют каким-либо синтаксическим правилам грамматики.

Класс Listener’a содержит описания методов, которые позволяют выполнять действия, необходимые при обходе синтаксического дерева.

При обходе абстрактного синтаксического дерева интерпретатор находит очередную инструкцию. Затем помощью рефлексии определяет метод с соответствующей аннотацией и добавляет ее в список необходимых для выполнения,

При завершении обхода AST инструкции поочередно выполняются в том порядке, в котором были заданы в тестовой спецификации.