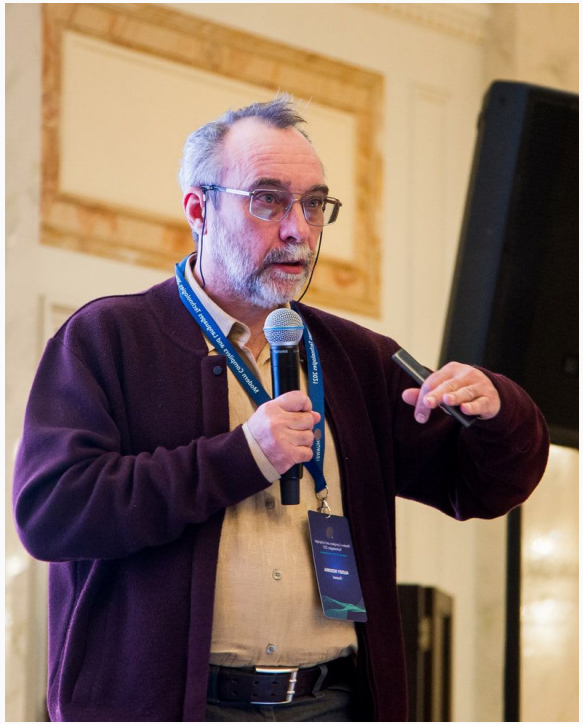
# Языки программирования. Лекция 3. Как разрабатывать. От личного к общему

Алексей Недоря, январь 2026

# Профессия: Языки программирования и компиляторы



**Краткая профессиональная биография**:

- Первый компилятор: 1984
- Компиляторы для 9 языков программирования
- Участие в стандартизации языков программирования
  - Modula-2 ISO/IEC
  - Oberon-2 Oakwood Guidelines
- Системная архитектура
- Разработка 7 языков программирования

- t.me/vorchalki_o_prog
- http://digital-economy.ru/avtory/aleksei-nedoria-synergetic-lab-ru
- https://ontonet.org/gruppy/vorchalki-o-programmirovanii
- http://алексейнедоря.рф/

@VORCHALKI_O_PROG

# Ликбез: Разработка языков программирования

**Прочитано:**

- 26.11.2025: История. Зачем и почему. запись
- 17.12.2025. Как разрабатывать. Личная история. Модула-0, Модула-X, Вир/a0. запись

**21.01.2026:**

- Краткое содержание предыдущей серии
- Вир/a1
- На что опираться при разработке языка?
- Что такое современный язык программирования?
- Личный фазовый переход
- Первый корпоративный (язык К1)
- Требования к языку

**Следующие лекции:**

- Что, как и проблемы
- ?

# Краткое содержание предыдущей серии и Уроки

- Внесение (правильных) ограничений может сократить трудоемкость разработки в десятки и сотни раз: Модула-0, К1, Тривиль.
- Точечные и органичные конструкции языка могут существенно увеличить выразительность языка и продуктивность разработчика: Модула-X, Тривиль.
- Убирание зависимостей очень сильно влияет на скорость и качество разработки: Модула-0 (и вообще Кронос), Вир/а0, Тривиль
- Для необычных задач надо искать необычные решения: Вир/а0, Тривиль, Арвиль

# Вир/a0 (кратко из предыдущей лекции)

**Основные решение для Вир/a0:**

- Выбросить все лишнее, как минимум: Delphi и DLL
- Простейший язык программирования
- Максимально простой компилятор (чтобы не отвлекаться на дополнительные работы):
  - Табличный разбор
  - Генерация для простой стековой машины
  - Табличная эмуляция команд стековой машины через команды x86
- Простейший бинарный формат, в котором есть только необходимое
- Слой, отделяющий от платформы (Windows)

**Язык Вир/a0**:

- Русский язык с пробелами (ЯРМО)
- Разделение семантики по синтаксису:
  - вызов внутренний
  - вызов тезаурусный
  - вызов …
- Типовая система: отсутствует! (FORTH)
  - Один тип данных: Слово64
  - Целое число или указатель
  - Вещественные через вызовы
- Нет описаний переменных
- Семантический анализ: отсутствует! (как в Модуле-0 - пиши правильно)
- Правое присваивание (легче генерить)
- Оператор “проверить” (guard, гл. шампур)

# Вир/a1: Постановка задачи

**2018**

- Разработка языка Вир/a1 для замены Вир/a0
- Основные требования в постановке задачи (исправить a0):
  - “Хорошая” типовая система
  - Полная проверка семантики
  - Переносимость
- Сохранить
  - Компонентность (сборка)
  - Использование в рамках Вир (совместное с Вир/a0)

С чего начать разработку “хорошего” языка?

- Проверять другие языки? Go, Rust
- Перечислять нужные конструкции?

# Вир/а1: На что опираться при разработке “хорошего” языка?

Нужно построить основу, состоящую из

- Большой цели
- Развернутой формулировки задачи
- Требований, проработанных с нужным уровнем детализации

Основа позволяет:

- упростить разработку за счет отбрасывания не соответствующих решений
- не пропустить важные свойства и конструкции
- проверять сделанные решения на соответствие
- сделать явным изменение требований

# Вир/a1: На что опираться при разработке “хорошего” языка?

**Постановка задачи для Вир/a1:**

- Декабрь 2018: Компонентный ассемблер для цифрового пространства. Часть 1
- Январь 2019: Компонентный ассемблер. Часть 2. Дух языка

По сути, любое современное приложение – это распределенная система, состоящая из нескольких (множества) распределенных компонент, работающих на разных устройствах, часто виртуальных.

… если мы хотим создать **современную систему разработки распределенных программ**, надо проводить эксперименты, а для того, чтобы проводить эксперименты просто и быстро, нужен экспериментальный язык и экспериментальная среда программирования.

Попытка делать эксперименты на каком-то из существующих языков, к сожалению, как показывает опыт автора, приводит к тому, что большая часть времени тратится на обходы ограничений существующих языков, компиляторов и сред исполнения (RTS).

# Промежуточная точка личной истории: середина 2019 года

Пять лет работы системным архитектором (2014-2019) позволили выделить и сформулировать проблемы отрасли и наметить пути их решения:

- Сентябрь 2018: Триада языков программирования (http://алексейнедоря.рф/?p=298)
- Май 2019: Об изготовлении программ и ежиках в тумане
- Июнь 2019: Всеплатформенная разработка или если б я был султан

Необходимость разработки **современных** языков программирования стала для меня очевидной.

# Alan Kay о современных языках программирования

**Интервью 2016 года:**

- Q: Do we "really" need more programming languages?
- Alan Kay: We could use a few "good ones" (meaning ones that really are about the **realities, needs and scales of the 21st century**).

- Q: Could you list top 3 good ones as per your opinion?
- Alan Kay: I meant, we could "use three good ones", not that I **knew of three**.

https://news.ycombinator.com/item?id=11939851

# Что я понимаю под современным языком программирования?

Современный язык программирования:

- Входит в семейство (минимизация накладных расходов на взаимодействие)
- Безопасность памяти
- Отсутствие неопределенного поведения
- Безопасный на достаточном уровне в соответствии с предметной областью
- Поддержка структурности: как минимум модульность, желательно компонентность
- Поддержка параллельности (гарантии корректности)
- Формальные доказательства, как минимум корректности типовой системы

Всегда ли мы делаем/будем делать на 100% современные языки программирования?

- не обязательно, могут быть другие задачи

# Лето 2019: фазовый переход

К лету 2019

- я осознаю необходимость разработки семейства современных языков
- на своем опыте пришел к пониманию того, как делать языки, включая:
  - С чего начать
  - Требования к безопасности памяти и ссылок
  - Требования к типовой системе
  - Обязательность поддержки мультиплатформенности
  - Необходимость разных решений для разных целевых областей. Чем “страньше” область, тем “страньше” решения

Меня позвали в RRI делать язык

# Язык К1: Первый корпоративный

**Август 2019.**

- Начало разработки языка для мобильных приложений
- Первые 3 месяца полностью заняты выработкой требований с точки зрения
  - Бизнеса
  - Продуктивности разработчика (читабельность, понимаемость, расширяемость)
  - Безопасности: типов, указателей и т.п.
  - Полноты и ортогональности типовой системы (читать здесь)
  - Возможности эффективной реализации компилятора, инструментов и runtime
  - Эффективности программ (performance, startup time, memory footprint)
  - Что нельзя делать и как нельзя делать
- Примерно через 3 месяца началась разработка языка и прототипирование компилятора

**Декабрь 2021**

- Язык готов на 90%, компилятор, библиотеки, runtime. Работает на Windows, Linux и на мобильной платформе с UI

# Тривиль (2022): пример проработки задачи и требований

**Задача:** язык для разработки компиляторов и экосистемы

- современный, надежный, удобный
- простой и понятный
- русскоязычный
- минимально достаточный

| **Требование** | **Важность** | **Уточнение требования** |
|---|---|---|
| Продуктивность разработчика | Высокая | ● легкость чтения, понимания, написания<br>● выразительность (есть все необходимые конструкции) |
| Скорость разработки | Высокая | ● быстро работающий компилятор и другие инструменты<br>● минимизация времени на поиск/исправление ошибок и на отладку => требование **безопасности языка** |
| Минимизация объема работы | Высокая | ● простой язык (минимальный набор конструкции)<br>● в языке нет ничего, кроме того, что необходимо<br>● простой компилятор |
| Безопасность языка | Высокая | ● безопасность ссылок<br>● управление памятью: сборка мусора<br>● модульность |
| Производительность | Низкая | |

# Зачем нужны требования? Защита языка

Самая сложная и трудоемкая и неблагодарная часть разработки языка, это объяснить, что предлагаемая конструкция, свойство или поведение не может быть добавлено в язык.

Как это делается

- Выделяем “железные” требования, которые нельзя нарушать, они должны быть выбиты на скрижалях и подписаны кровью начальства. Остальные иногда нарушать можно, но редко и низенько.
- Если предложение очевидно нарушает железное требование или несколько других, то оно отбрасывается сразу.
- Если не видно сразу, то берется в работу и рассматриваются
  - какие есть варианты решения
  - сделано ли это в других языках, как сделано и какие последствия
  - как это решение взаимодействует с другими (feature interaction)
- Во многих случаях изучение последствий приводит к “доказательству от противного”, то есть к тому, что последствия нарушают требования.

# История и состояние языков на начало 2026 г.

- Вир/a0: 2006-2012, язык, компилятор, среда разработки, приложения, всего 0.5M строк кода
- Вир/a1: 2018-2019, прототип языка и компилятора
- К1: 2019-2021, язык (90%), компилятор, интеграция, запуск на мобильных платформах
- Static ArkTS: в работе с 2022
- Тривиль: 2022-2023, версия завершена в 2023 году
- Арс и Арвиль: 2023-2025, прототипы, работа заморожена
- Язык системный: предварительная работа начата в сентябре 2025

# Вопросы

EOF