Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК9 «Иностранные и русские языки»***

ДОМАШНЕЕ ЧТЕНИЕ №2

Эволюция языков программирования и их влияние на разработку программного обеспечения

The Evolution of Programming Languages and Their Impact on Software Development

ДИСЦИПЛИНА: «Английский язык»

Выполнил: студент гр. ИУК4-62Б _____ (____Губин Е.В._____)
(Подпись) (Ф.И.О.)

Проверил: _____ (___Артеменко О.А.__)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

The Evolution of Programming Languages and Their Impact on Software Development

Programming languages are the backbone of modern software development. From early machine code to high-level, human-readable syntax, the evolution of programming languages has dramatically influenced the way developers write code, interact with computers, and build increasingly complex systems. This text explores the historical development of programming languages, the paradigms they represent, and how they have shaped modern software engineering practices.

The earliest computers were programmed using binary machine code — sequences of 0s and 1s directly interpreted by the hardware. This process was tedious and error-prone. To improve efficiency, assembly language was introduced, allowing developers to write symbolic instructions instead of raw binary code. Though still low-level, assembly was a significant step forward, making programming more accessible and maintainable.

The 1950s saw the birth of the first high-level programming languages, such as Fortran (Formula Translation) and Lisp (List Processing). Fortran was designed for numerical and scientific computing, while Lisp pioneered functional programming and became a popular tool for artificial intelligence research. These languages abstracted many hardware details, enabling developers to focus more on logic and algorithms than machine specifics.

In the 1960s and 70s, the concept of structured programming emerged as a response to the “spaghetti code” problem — unstructured, difficult-to-read programs with tangled control flows. Languages like ALGOL, Pascal, and C embraced structured programming, promoting the use of loops, conditionals, and subroutines in place of arbitrary jumps (such as goto statements).

C, in particular, became extremely influential. It offered a good balance between low-level access and high-level control structures, making it ideal for systems programming. Unix, a cornerstone of modern operating systems, was written in C, reinforcing its importance.

The 1980s and 90s saw the widespread adoption of object-oriented programming (OOP), a paradigm that organizes code around “objects” — encapsulated units containing data and behavior. Smalltalk was one of the earliest OOP languages, but it was C++, and later Java, that brought OOP into mainstream use.

OOP made it easier to model real-world entities and encouraged modular, reusable code. Java, in particular, revolutionized enterprise development with its “write once, run anywhere” philosophy, thanks to the Java Virtual Machine (JVM).

With the explosion of the Internet in the mid-1990s, demand surged for dynamic, interactive web content. This led to the rise of scripting languages such as JavaScript, PHP, and Python. These languages prioritized ease of use and rapid development over performance.

JavaScript became the de facto language of the web, powering everything from basic interactions to full-featured web applications through frameworks like React, Angular, and Vue. PHP dominated server-side scripting, while Python gained popularity for its readability and versatility.

In the 2000s and beyond, new languages emerged to address evolving needs. C#, developed by Microsoft, brought Java-like OOP to the .NET ecosystem. Ruby, with its elegant syntax, was popularized by the Ruby on Rails framework. Go (Golang) emphasized simplicity and concurrency, ideal for cloud-native applications. Rust gained attention for its memory safety without garbage collection.

Meanwhile, functional programming experienced a resurgence, with languages like Haskell, Scala, and Elixir offering powerful tools for managing state and concurrency. Even traditionally imperative languages adopted functional features — for example, Java introduced lambda expressions in Java 8.

Today's software development often involves multiple languages and paradigms. A web application might use JavaScript on the front end, Python on the backend, SQL for database queries, and Bash for deployment scripts. Developers now choose languages based on their strengths for specific tasks rather than a one-size-fits-all approach.

Modern development also includes practices like DevOps, CI/CD, and infrastructure as code, which have led to the rise of domain-specific languages (DSLs) like Terraform's HCL or configuration formats like YAML and JSON. Developers not only write application code but also define how their applications are built, tested, and deployed.

In AI and data science, Python reigns supreme due to its rich ecosystem of libraries (TensorFlow, PyTorch, scikit-learn, etc.), but other languages like R and Julia also contribute significantly.

As technology continues to evolve, so will programming languages. New paradigms like quantum computing may require entirely new ways of thinking about programming, with languages like Q# or Quipper emerging to address these domains.

Moreover, AI-assisted coding tools like GitHub Copilot or ChatGPT are beginning to shift the developer's role — from writing every line of code to orchestrating and validating generated code. This may influence language design in the future, focusing more on intent-based or declarative syntax.

The evolution of programming languages reflects the broader story of computing itself — from low-level hardware control to high-level abstractions, from isolated systems to globally connected networks, and from manual coding to intelligent automation. Understanding this history helps developers appreciate the tools they use today and prepares them for the innovations of tomorrow.

Dictionary

backbone – основа, опора

error-prone – подверженный ошибкам

abstraction – абстракция

structured programming – структурное программирование

object-oriented programming (OOP) – объектно-ориентированное программирование

encapsulated – инкапсулированный (сокрытый внутри объекта)

concurrency – параллелизм, конкурентность

garbage collection – сборка мусора (автоматическое управление памятью)

domain-specific language (DSL) – предметно-ориентированный язык программирования

declarative syntax – декларативный синтаксис

Translation into Russian

Языки программирования — это основа современной разработки программного обеспечения. От раннего машинного кода до высокоуровневого, читаемого человеком синтаксиса, эволюция языков программирования кардинально изменила то, как разработчики пишут код, взаимодействуют с компьютерами и создают всё более сложные системы. В этом тексте рассматривается историческое развитие языков программирования, парадигмы, которые они представляют, и то, как они повлияли на современные практики программной инженерии.

Первые компьютеры программировались с использованием двоичного машинного кода — последовательностей из 0 и 1, которые непосредственно интерпретировались оборудованием. Этот процесс был утомительным и подверженным ошибкам. Для повышения эффективности был введён язык ассемблера, позволивший разработчикам писать символьные инструкции вместо чистого двоичного кода. Хотя это всё ещё был низкоуровневый язык, ассемблер стал значительным шагом вперёд, сделав программирование более доступным и сопровождаемым.

В 1950-х годах появились первые языки высокого уровня, такие как Fortran (сокр. от Formula Translation) и Lisp (сокр. от List Processing). Fortran был разработан для численных и научных расчётов, в то время как Lisp стал пионером функционального программирования и популярным инструментом в исследованиях по искусственному интеллекту. Эти языки абстрагировали многие детали аппаратного уровня, позволяя разработчикам сосредотачиваться на логике и алгоритмах, а не на спецификах машины.

В 1960-х и 70-х годах возникла концепция структурного программирования как ответ на проблему “макаронного кода” — неструктурированных, трудно читаемых программ со спутанной логикой управления. Такие языки, как ALGOL, Pascal и C, поддерживали структурное программирование, поощряя использование циклов, условных операторов и подпрограмм вместо произвольных переходов (например, операторов goto).

Особенно сильно повлиял язык C. Он предложил хороший баланс между низкоуровневым доступом и высокоуровневыми управляющими структурами, что сделало его идеальным для системного программирования. Операционная система Unix, ставшая краеугольным камнем современных ОС, была написана на C, что укрепило значение этого языка.

В 1980-х и 90-х годах широкое распространение получило объектно-ориентированное программирование (ООП) — парадигма, организующая код вокруг "объектов" — инкапсулированных единиц, содержащих данные и поведение. Smalltalk стал одним из первых языков ООП, но именно C++, а позже Java, сделали объектно-ориентированный подход массовым.

ООП упростило моделирование реальных сущностей и способствовало созданию модульного, повторно используемого кода. Особенно Java произвела революцию в корпоративной разработке благодаря философии “один раз написал — везде работает”, реализованной через виртуальную машину Java (JVM).

С бурным развитием интернета в середине 1990-х годов резко возрос спрос на динамичный и интерактивный веб-контент. Это привело к появлению скриптовых языков таких как JavaScript, PHP и Python. Эти языки ставили в приоритет простоту использования и быструю разработку, а не производительность.

JavaScript стал фактически стандартом языка для веба, обеспечивая всё — от базовых взаимодействий до полноценных веб-приложений с помощью фреймворков, таких как React, Angular и Vue. PHP доминировал в серверных скриптах, а Python завоевал популярность благодаря своей читаемости и универсальности.

В 2000-х годах и позже появились новые языки, отражающие изменяющиеся потребности. C#, разработанный Microsoft, принёс ООП, схожий с Java, в экосистему .NET. Ruby, с его элегантным синтаксисом, стал популярен благодаря фреймворку Ruby on Rails. Go (или Golang) сделал акцент на простоту и поддержку параллелизма, что делает его идеальным для облачных приложений. Rust привлёк внимание благодаря обеспечению безопасности памяти без использования сборки мусора.

В то же время функциональное программирование пережило возрождение. Языки такие как Haskell, Scala и Elixir предоставили мощные инструменты для управления состоянием и параллелизмом. Даже традиционно императивные языки начали внедрять функциональные элементы — например, в Java появились лямбда-выражения, начиная с версии Java 8.

Современная разработка программного обеспечения часто использует несколько языков и парадигм. Веб-приложение может включать JavaScript на клиентской стороне, Python на серверной, SQL для запросов к базе данных и Bash для скриптов развертывания. Разработчики теперь выбирают языки на основе их сильных сторон для конкретных задач, а не по принципу "один язык на всё".

Современная разработка включает в себя такие практики, как DevOps, непрерывная интеграция и доставка (CI/CD) и инфраструктура как код, что привело к появлению предметно-ориентированных языков (DSL), таких как HCL для Terraform, а также конфигурационных форматов, таких как YAML и JSON. Разработчики теперь не только пишут код приложений, но и определяют, как эти приложения собираются, тестируются и разворачиваются.

В области искусственного интеллекта и науки о данных Python занимает лидирующую позицию благодаря своему богатому набору библиотек (TensorFlow, PyTorch, scikit-learn и др.), но также активно используются и другие языки, такие как R и Julia.

По мере развития технологий будут развиваться и языки программирования. Новые парадигмы, такие как квантовые вычисления, могут потребовать совершенно новых способов мышления о программировании, что уже приводит к появлению языков вроде Q# и Quipper, специально разработанных для этой области.

Кроме того, инструменты с поддержкой ИИ, такие как GitHub Copilot или ChatGPT, начинают менять роль программиста — от написания каждой строки вручную к управлению и проверке сгенерированного кода. Это может повлиять и на дальнейший дизайн языков — они будут всё больше фокусироваться на декларативности и передаче намерений.

Эволюция языков программирования отражает более широкую историю самой вычислительной техники — от низкоуровневого управления оборудованием до высокоуровневых абстракций, от изолированных систем до глобально связанных сетей, и от ручного кодирования до интеллектуальной автоматизации. Понимание этой истории помогает разработчикам ценить инструменты, которыми они пользуются сегодня, и готовиться к инновациям завтрашнего дня.

Questions

1. What was the primary method of programming early computers?
2. How did assembly language improve the process of programming?
3. What were the main purposes of Fortran and Lisp?
4. Why was structured programming introduced in the 1960s and 70s?
5. What made the C programming language particularly influential?
6. How did object-oriented programming change the way developers wrote code?
7. Which languages popularized the object-oriented programming paradigm?
8. What role did Java play in enterprise software development?
9. Why did scripting languages like JavaScript and PHP become popular in the 1990s?

10. How has Python gained popularity across different domains?
11. What are some modern programming languages that emerged in the 2000s and their strengths?
12. Why is Rust considered a safe language for memory management?
13. How have functional programming concepts influenced modern languages like Java?
14. What is the significance of using multiple languages in a single software project today?
15. How are AI-powered tools like GitHub Copilot changing the role of programmers?

Retelling

1. Programming languages have evolved from low-level binary and assembly to high-level languages that simplify coding and improve productivity.
2. In the 1950s, languages like Fortran and Lisp introduced abstraction, allowing developers to focus more on logic than hardware.
3. Structured programming in the 1960s and 70s promoted cleaner, more readable code with languages like C.
4. Object-oriented programming became popular in the 1980s and 90s with languages like C++ and Java, emphasizing modularity and reuse.
5. The rise of the internet led to widespread use of scripting languages such as JavaScript, PHP, and Python.
6. Modern development often involves multiple languages and paradigms, chosen for their strengths in specific tasks.
7. Newer languages like Go and Rust address challenges like concurrency and memory safety.
8. The future of programming is influenced by AI tools and emerging fields like quantum computing, pushing languages toward more declarative and intention-focused design.