Turiba University

Robin Singh

History of Programming Language

PROFESSIONAL BACHELOR THESIS

Computer Systems

Author: **Robin Singh**
***Thesis* Advisor:** **Jānis Pekša**

Riga, 2025

History of Programming Languages

Objective

- To trace the historical development of programming languages from early mechanical computers to contemporary high-level languages.
- To examine key milestones and innovations that shaped modern programming paradigms.
- To analyze how programming languages evolved to address the changing needs of software development.
- To understand the transition from machine-specific assembly code to portable, human-readable programming languages.

Introduction

The story of programming languages is fundamentally a story of abstraction. In the early days of computing, programmers worked directly with machine code, painstakingly translating their logic into sequences of binary instructions. This process was not only time-consuming but also highly error-prone and machine-specific. As computers became more powerful and their applications more complex, the need for more accessible ways to communicate with machines became apparent.

Throughout the 20th century, the development of compiler theory and language design transformed programming from a specialized skill requiring intimate knowledge of hardware into a more accessible discipline. High-level programming languages introduced syntax that resembled mathematical notation and even natural language, allowing programmers to focus on solving problems rather than managing hardware details.

This report examines the major phases in programming language evolution, from the earliest attempts at creating machine instructions to the sophisticated languages we use today. Each era brought its own innovations and challenges, shaped by the hardware limitations of the time, the theoretical advances in computer science, and the practical needs of developers.

The Dawn of Programming

The concept of programming predates electronic computers by over a century. Between 1842 and 1849, Ada Lovelace worked on translating Luigi Menabrea's memoir about Charles Babbage's Analytical Engine. What made her work remarkable was not just the translation but the extensive notes she added. These notes included a detailed method for calculating Bernoulli numbers using the engine, which many historians now recognize as the world's first published algorithm intended for machine execution.

Although Babbage's Analytical Engine was never built during his lifetime, the theoretical framework established by Lovelace demonstrated that machines could be programmed to perform complex mathematical operations. This vision would not be realized in practice for another century, but it laid the conceptual groundwork for programmable computing.

The Birth of High-Level Languages

Between 1942 and 1945, Konrad Zuse designed Plankalkül, which is now recognized as the first high-level programming language. Zuse developed this language for his Z1 computer, incorporating concepts that were remarkably advanced for the time. However, Plankalkül was not implemented during Zuse's lifetime, partly due to the disruptions of World War II and partly due to limited interest in his work outside of Germany. The language remained largely unknown until it was rediscovered by historians decades later.

Short Code and Early Experiments

In 1949, John Mauchly proposed Short Code, one of the first attempts at creating a high-level language for electronic computers. Short Code allowed programmers to write mathematical expressions in a more understandable form compared to machine code. However, it had a significant drawback: the program had to be interpreted and translated into machine code every time it ran, making execution much slower than programs written directly in machine code. This trade-off between programmer convenience and execution speed would become a recurring theme in language design.

Conclusion

The evolution of programming languages reflects the continuous effort to make computers more accessible and useful. From Ada Lovelace's notes on the Analytical Engine to modern languages with sophisticated type systems and concurrency support, each advance has built on previous work while addressing new challenges.

Several themes emerge from this history. First, abstraction has been crucial - hiding hardware details has allowed programmers to work at higher conceptual levels. Second, different problems require different approaches, which is why we have multiple programming paradigms rather than one universal language. Third, practical concerns like execution speed, development time, and maintainability have been just as important as theoretical elegance in determining which languages succeed.