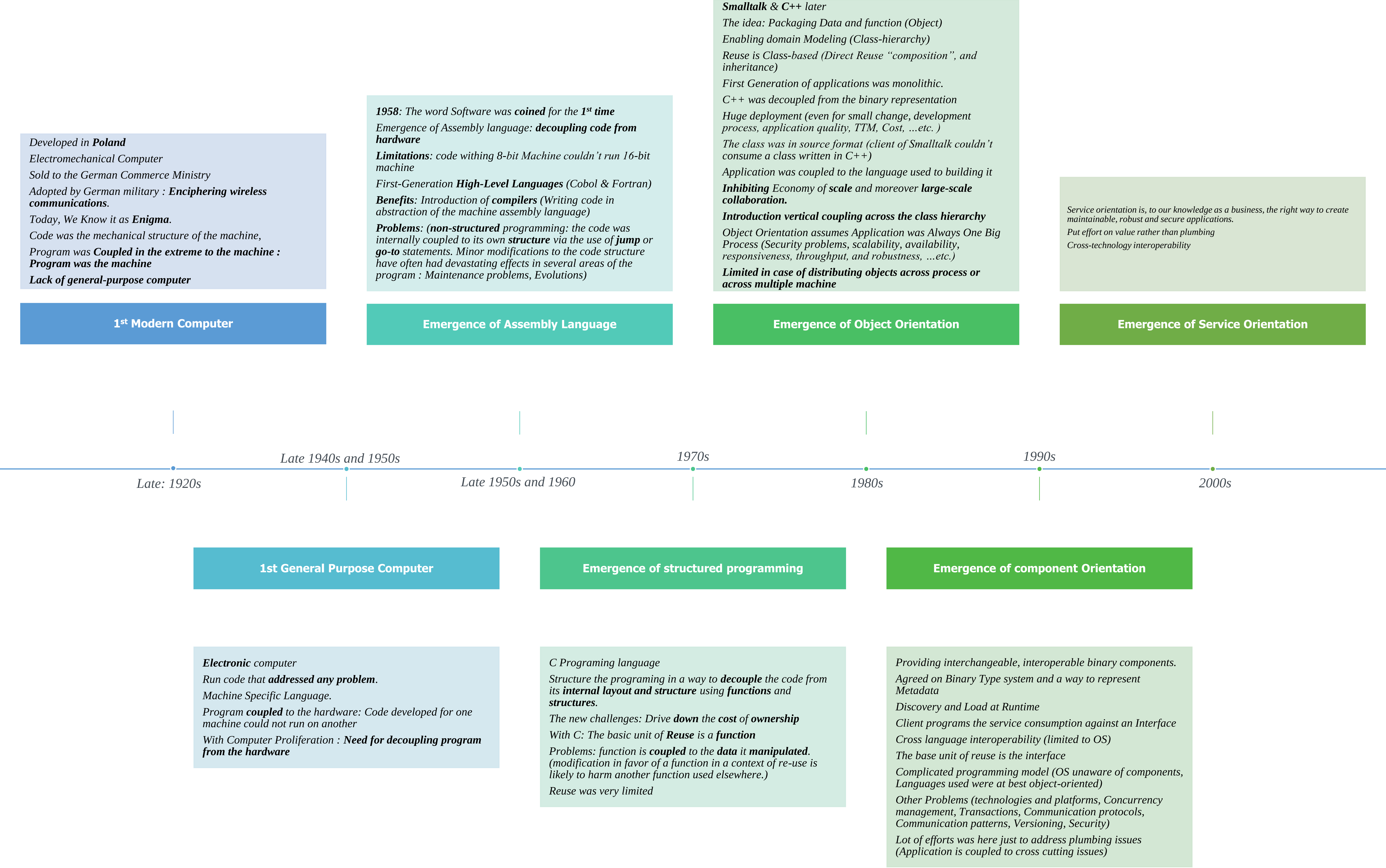

Late: 1920s

1st Modern Computer

Developed in **Poland**

Electromechanical Computer

Sold to the German Commerce Ministry

Adopted by German military : **Enciphering wireless communications**.

Today, We Know it as **Enigma**.

Code was the mechanical structure of the machine,

Program was **Coupled in the extreme to the machine : Program was the machine**

Lack of general-purpose computer

Late 1940s and 1950s

1st General Purpose Computer

Electronic computer

Run code that **addressed any problem**.

Machine Specific Language.

Program **coupled** to the hardware: Code developed for one machine could not run on another

With Computer Proliferation : **Need for decoupling program from the hardware**

Late 1950s and 1960

Emergence of Assembly Language

1958: The word Software was **coined** for the **1st time**

Emergence of Assembly language: **decoupling code from hardware**

Limitations: code withing 8-bit Machine couldn’t run 16-bit machine

First-Generation **High-Level Languages** (Cobol & Fortran)

Benefits: Introduction of **compilers** (Writing code in abstraction of the machine assembly language)

Problems: (**non-structured** programming: the code was internally coupled to its own **structure** via the use of **jump** or **go-to** statements. Minor modifications to the code structure have often had devastating effects in several areas of the program : Maintenance problems, Evolutions)

1970s

Emergence of structured programming

C Programing language

Structure the programing in a way to **decouple** the code from its **internal layout and structure** using **functions** and **structures**.

The new challenges: Drive **down** the **cost** of **ownership**

With C: The basic unit of **Reuse** is a **function**

Problems: function is **coupled** to the **data** it **manipulated**. (modification in favor of a function in a context of re-use is likely to harm another function used elsewhere.)

Reuse was very limited

1980s

Emergence of Object Orientation

Smalltalk & **C++** later

The idea: Packaging Data and function (Object)

Enabling domain Modeling (Class-hierarchy)

Reuse is Class-based (Direct Reuse “composition”, and inheritance)

First Generation of applications was monolithic.

C++ was decoupled from the binary representation

Huge deployment (even for small change, development process, application quality, TTM, Cost, …etc.)

The class was in source format (client of Smalltalk couldn’t consume a class written in C++)

Application was coupled to the language used to building it

Inhibiting Economy of **scale** and moreover **large-scale collaboration.**

Introduction vertical coupling across the class hierarchy

Object Orientation assumes Application was Always One Big Process (Security problems, scalability, availability, responsiveness, throughput, and robustness, …etc.)

Limited in case of distributing objects across process or across multiple machine

1990s

Emergence of component Orientation

Providing interchangeable, interoperable binary components.

Agreed on Binary Type system and a way to represent Metadata

Discovery and Load at Runtime

Client programs the service consumption against an Interface

Cross language interoperability (limited to OS)

The base unit of reuse is the interface

Complicated programming model (OS unaware of components, Languages used were at best object-oriented)

Other Problems (technologies and platforms, Concurrency management, Transactions, Communication protocols, Communication patterns, Versioning, Security)

Lot of efforts was here just to address plumbing issues (Application is coupled to cross cutting issues)

2000s

Emergence of Service Orientation

Service orientation is, to our knowledge as a business, the right way to create maintainable, robust and secure applications.

Put effort on value rather than plumbing

Cross-technology interoperability