The secure way to remove Mainframe Assembler

Until the end of the 1980s insurances, financial institutes, government authorities, and airlines had no choice but to develop their core applications in Mainframe Assembler. Changed and extended over and over, the knowledge about the inner detail logic of these systems died away.

While the old-time programmers retired, thousands of these systems are still alive – difficult to maintain and blocking strategical moves.

Promises

How often have you heard about Assembler projects failing because they believed in this "*we'll convert 70% automatically*" or "*our offshore Assembler experts will solve the problem*".

Honestly: The error-free transformation of Assembler code to a "real" computer language is one of the most challenging tasks in IT.

We know, because we accomplished it: the latest extension to our Njema product offers the solution.

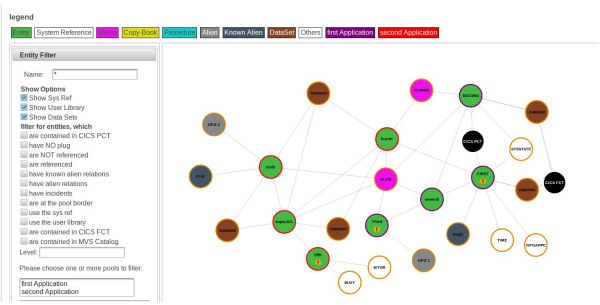
Full Automation in the Assembler-Transformation

- **brings calm** to the project
- **lowers the risk** dramatically
- makes **cost predictable**
- **shortens duration** significantly
- **frees** project from **dependency** on regular program maintenance
- **eliminates human errors**
- allows **repeating** the transformation hassle-free **as often as desired**
- lets you **go live within a few hours or days**

Methodology

The project's success depends on knowing in advance all the details of every single Assembler statement in the whole application.

- Hence Njema examines system and user macros, coding conventions, as well as programmers' tricks and habits. It analyzes addressing, register usage, finds self-modifying code, and so forth and stores this into the *Njema Metadata Repository*.
- Njema also creates a complete inventory of all data fields, their usage properties and - very important – their interrelations.
- This massive amount of data allows to apply **AI algorithms** for smart transformation and a final **re-engineering across program boundaries.** Thus the transformation rules do their job with respect to all other statements within the the whole application.
- Based on this knowledge a project-specific Njema *Transformation Engine* is built by choosing hundreds of transformation rules from the *Njema Rule Repository*.
- This Engine is capable to transform all of the Assembler entities in one processing cycle. Every cycle will reveal new possibilities for more automation, which are then added to the rule repository.
- Unique code constructs which may occur seldom or just once are covered by singular rules.
- In the end, the Transformation Engine has learned to convert the complete Assembler application into a functionally identical Cobol incarnation: 100% automatically.



The Reward

This methodology provides a low-risk and plannable flow of the transformation project.

- Your dedicated assembler Transformation Engine is built and tested while regular development and application maintenance is not affected at all.
- *Continuous Code Monitoring* makes sure that the code to be transformed and tested is updated regularly to the productive program versions.
- Transformation errors discovered in a test cycle are never corrected in the Cobol code. They are corrected in the Transformation Rules, instead.
- The inter-connection between programs, macros, fields, data structures, and files can be presented graphically from the stored metadata (s. a.).

- Once the final test is successful, an *Integration Test* will prove that the new Cobol code can go live, because it matches exactly the functionality of the former Assembler programs.

How can all this be done, when there is no documentation?

Well, in fact **there is reliable documentation**: it is the hexadecimal so-called *Object Code* which Njema uses as the ultimate directive.

The Object Code is what drives the Assembler program's logic. So being able to use it will always lead to generating the correct Cobol equivalent.

- The resulting Cobol code is well structured and easy to read, so regular program maintenance can continue with the Cobol programs right away.

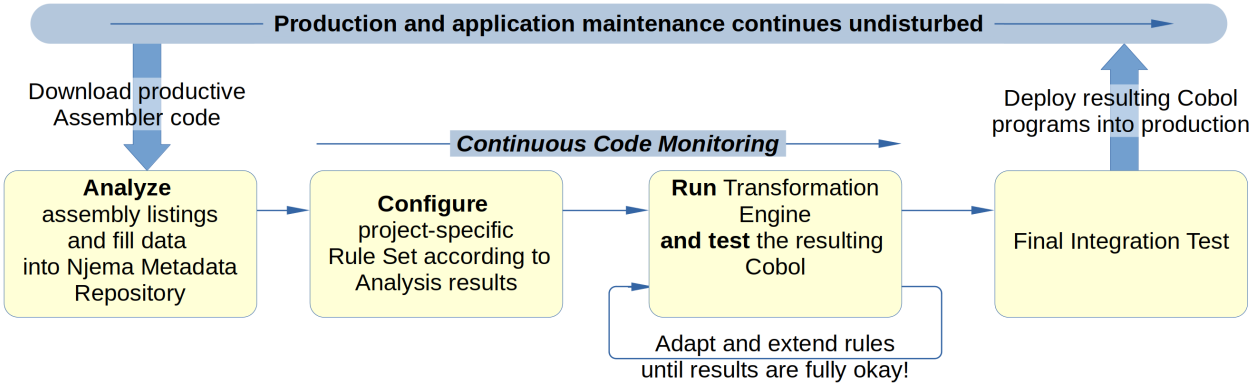
Supported platforms

Currently Njema supports Assembler transformations for z/OS, TPF, and VSE.

Terms and Conditions

Transformation is provided as a service.

Alternatively your individual Modernization Engine is deployed into a cloud as a *Continuous-Build Package*. This means you will use the Engine and ITM will extend and maintain it.



Here is a simple and unbinding way to prove why this is the safest way to get rid of the Assembler

ITM offers free of charge:

1. Transformation of a few thousand lines of source code

2. Analysis of a complete Assembler application resulting in an estimate of cost and project duration