DELTA
software
technology

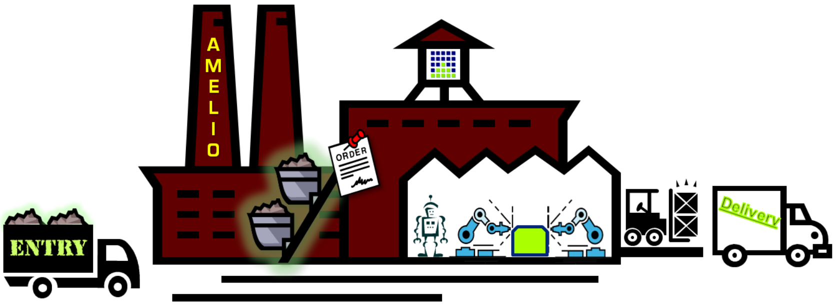
AMELIO
Logic Discovery

Everything new in Spring?

Many companies often regard legacy applications as a burden and their maintenance and further development as error-prone and cost-intensive. But re-writing the application is too expensive and too risky due to size and complexity. By cleaning-up, re-structuring and modernization this burden becomes a valuable legacy that can be efficiently and continuously used. With the tools from the AMELIO family these modernization steps can be executed tailor-made and automated. As a result, the effort and the risk will be manageable.

1 Legacy – Burden or valuable heritage?

Many companies have developed their business-critical core applications in COBOL or PL/I over decades. Large and complex applications are the result of further maintenance and development. These applications are often inflexible, their maintenance and further development is error-prone and time-consuming. That is why legacy application are often seen as a burden, although they contain much knowledge about the company processes. There are many reasons for this: The application has grown over decades, has been passed from developer to developer, has been developed according to different paradigms, methods and versions of a language, it uses old technologies and much more.



Factory for the modernization of legacy applications

Writing the applications in a modern language according to current paradigms is an enormously expensive and risky endeavour due to the size and complexity of the task. But the profit to be achieved is comparatively low and "only" the old functionality will be recovered and no new one would be added. Instead of "throw away" the existing applications or to re-implement them, it makes more sense to clean up, refactor and modernize the code and to improve the readability and maintainability and to enable an efficient further use.

With the tools from the AMELIO family the cleaning and modernization tasks can be made step-by-step and 100% automated. As a result, effort and risk for such a project can be minimized.

2 Fit for the future

There are many possibilities to add flexibility and maintainability to a legacy application. They range from clean-ups over refactoring up to modernization. In some cases it is sufficient to take individual measures, in other cases several should be carried out stepwise one-by-one or in combination.

2.1 Regaining knowledge – Understand your application

Before significant changes to an application can be planned and performed, it is indispensable to understand what the application is doing, how it is doing it and which connections and dependencies exist. The documentation if it is complete and currently available serves only as memory aid for the original developer of the application. In most cases, however, it is unsuitable as a basis for a modernization project.

In such cases AMELIO Logic Discovery helps to analyse the applications and to perform abstractions which facilitate the understanding of the application. Additionally, dispensable ballast, proposals for a refactoring and the cruxes for a modernization can be identified automatically.

2.2 Getting rid of dispensable ballast

Decades of application development, change and maintenance often result in dispensable ballast. Code which is actually no longer required but which is not recognized with the naked eye and is therefore maintained as well. With the automatic analyses such code, statements and data definitions can be identified and deleted. In COBOL and PL/I copybooks and includes play an important role, so it must also be determined whether the code comes from such a module and whether it is always dead or only in some of the programs.

The used programming languages were also developed over time. As a result, the applications often contain a mix of new language constructs and the old less easy to read constructs. These should be standardized.

Maintenance and extensions also change interfaces. However, these changes were not always followed throughout the application. If the application still works correctly, it is more or less a coincidence. Therefore it is important to analyse which interfaces are not defined correctly and take action to correct that.

Even if these are just a small changes at first glance, the impact on the readability is enormous.

2.3 New Structures

Decades of maintenance and further developments have changed the originally clean design. Besides that, programming paradigms have evolved. If monolithic applications were previously state-of-the-art, today they are considered to be hardly maintainable. Refactoring can be used to create better maintainable structures that meet today's standards. In COBOL and PL/I there is often the necessity to combine several small paragraphs under certain conditions or to outsource independent parts from particularly large programs to separate sub-programs. The implementation of a service layer for accesses to databases and files not only improves the readability of the programs, it also increases the flexibility in the exchange or restructuring of the underlying database and file systems.

2.4 Modernization

Over the years new technologies, e.g. database systems, were often implemented without replacing the old ones completely. Data must then be kept consistent in different systems, making the application unnecessary slow and inflexible. Moreover, the maintenance effort is unnecessarily high. By standardizing the technologies or replacing them with a new technology, performance and flexibility can be gained.

3 Fully automatically and step-by-step

Even small clean-up and modernization tasks require many and often deep interventions in the application. AMELIO performs these steps 100% automated, based on a factory-based approach and enables a minimization of effort and risk. AMELIO works model- and rule-based. First, all sources of an application are read and transferred into models by using formal and logic abstraction. The models are then the basis for further analyses or the actual transformations which are specified by rules. In this way it is possible to provide a range of standard analyses and transformations specifically for the code clean-up. On the other hand additional project-specific analyses and transformation can be realized. But, above all, the rule-based and automatic approach offers benefits in terms of security and quality. Analyses and transformations can be understood and reproduced at any time. All transformations are absolutely uniform. Once a rule transforms the code correctly, it always does. This means that not all changed code locations must be tested, only the correctness of the rule must be tested. In this way, the testing effort is drastically reduced. All changes made are automatically recorded in the code and in additional external documents in an audit-proof way. An automated, factory-based approach also makes it possible to perform a step-by-step clean-up and modernization of an application. By doing so, you can first clean-up the application before you decide and implement further measures. Instead of a "big bang", the application is made fit for the future stepwise. At the end of each step you have a working system and you can use the profit immediately.

4 Everything new in Spring?

Yes and no! With the redevelopment of an existing application, the profit is usually out of all proportion to risk and effort. Instead, it is much cheaper to analyse, clean-up, re-factorize and modernize the legacy application by using automated procedures. In this way, the knowledge implemented in the application is efficiently made usable again. If subsequently functional requirements force a rewriting, it is sufficient to re-develop individual components instead of the entire application.

AMELIO Logic Discovery

AMELIO Logic Discovery helps to understand the existing COBOL- and PL/I-applications and thus reduces the costs for re-implementation of the existing functions and for the modernization of the applications.

Further information can be found here: **delta-software.com/amld**

Order number: MT 21088.01 – June 2018