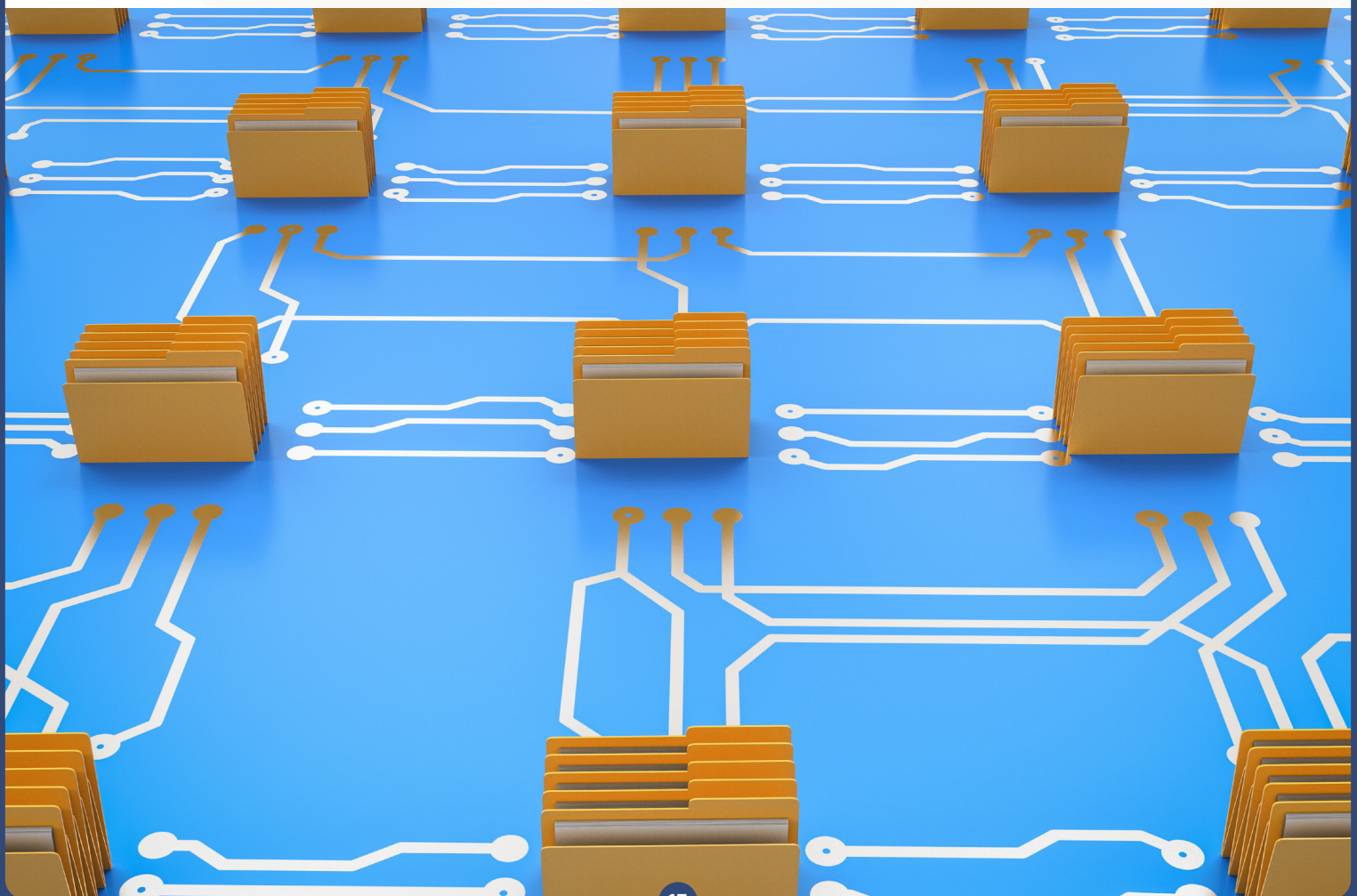


AI IN PORTFOLIO MIGRATION:

WHAT WORKS TODAY – AND WHAT COMES NEXT

BY **BARTLOMIEJ MACIAGA, ARNO RASCH** AND **SVEN GRÖNEWÄLLER**

Portfolio migrations in life insurance are lengthy, costly and error-prone – yet they follow clear actuarial premises. That is what makes them an ideal testing ground for explainable AI. The AI Lab of the DAV (German Association of Actuaries) working group on portfolio migration shows what is possible today and which role actuaries play in it. >



MOTIVATION

Portfolio migrations are among the most demanding undertakings in life insurance. Large policy portfolios with contract histories that often reach back decades must be migrated from a complex legacy administration system into a modern one. The reasons for migration are diverse: increasing operating costs of mainframe systems and new demands driven by digitalisation are key factors behind insurers' extensive efforts to modernise their core IT systems. Moreover, mergers between insurers and portfolio acquisitions by run-off carriers typically lead to migration projects.

In the past, migration projects – depending on portfolio size, tariff diversity and product complexity – could take several years. The DAV working group on portfolio migration therefore set out to investigate the use of new methods, particularly from the field of artificial intelligence, for migrations.

WHITE BOX INSTEAD OF BLACK BOX

The guiding decision was to focus on explainable AI rather than black box AI. For actuaries, it is not only the result that matters but the traceable path to it. There are useful applications for black box models, e.g., classification tasks, but in the actuarial context, those are often difficult to sign off from a regulatory perspective and are hard to handle in professional validation. White-box approaches, by contrast, produce traceable and reproducible results.

POTENTIAL APPLICATIONS OF AI IN MIGRATION PROJECTS

In this context, we examined the actuarially relevant subtasks, that typically arise in migration projects, for their potential to benefit from AI, and derived the following use cases.

- Analysis of product documentation, such as general policy conditions, technical business plans and regulatory filings.
- Analysis and preparation of the portfolio to be migrated, for example by detecting inconsistencies and applying automated corrections.
- Support in calculation-engine development through AI-assisted code generation, documentation and onboarding of new staff.
- Support in implementing product data, especially for target systems with heavily parameterised tariff frameworks, by intelligent combination of existing calculation logic.
- Support for data transformation through semantic analysis of data-field descriptions combined with machine-based evaluation of the portfolio data.
- Automation of testing the ETL (Extract – Transform – Load) pipeline through self-learning methods that detect recurring patterns in the data transformation.
- Providing reference values through the automated generation of reference calculators from existing tariff information and pricing tools.
- Generation of test portfolios, e.g., based on tariff descriptions, and assessment of test coverage.
- Support in migration controlling through Machine Learning-based plausibility checks, clustering of tolerance breaches, and root-cause analysis. >

DEEP DIVE: CALCULATION-ENGINE DEVELOPMENT

The use case concerning calculation-engine development was examined in greater depth. The prototype was the porting of an existing Excel tariff calculator to Python. Such calculators usually are hard to maintain, have no version control, and can barely be integrated into a modern IT landscape. Because of their complexity – multiple worksheets with formulas, defined names, references and Visual Basic for Applications (VBA) code – porting is generally a non-trivial task and hence a suitable case to evaluate the practical use of AI.

The first approach was hands-on: we manually decomposed the Excel tariff calculator into its individual components, such as formulas, mortality tables and VBA code, and translated these into Python step by step in dialogue with ChatGPT. The resulting Python calculator worked as intended, but it is difficult to reproduce the process.

In the next step, we therefore developed an industrial, scalable approach in the form of a pipeline: first, all contents of the Excel file are automatically extracted into text; then, utilising several calls to the Large Language Model (LLM), the corresponding Python code is generated and automatically tested using golden-master comparisons. Every step of this pipeline is traceable for humans and safeguarded by test gates. If one of the test gates fails, a new iteration is triggered, or a human intervenes to correct it.

Meanwhile, the capabilities of current LLMs have risen to the point where they can now port a complete Excel calculator on their own. Hence, we are placing further and more stringent demands: In addition to demonstrable correctness of the generated code, certain architectural guidelines must be obeyed during generation. This is particularly important when additional code is generated for existing target calculators. Beyond code, further sources – >

FIGURE 1 – Current architecture: the agentic system

