# A Case Study of Generative Engineering-Enabled Software Development

## Description

### OVERVIEW

In the previous post, I discussed how employing accelerated delivery strategies, specifically generative software engineering, can reduce overall project risks.  This post recounts the history of a project in which generative methods were employed to great benefit.

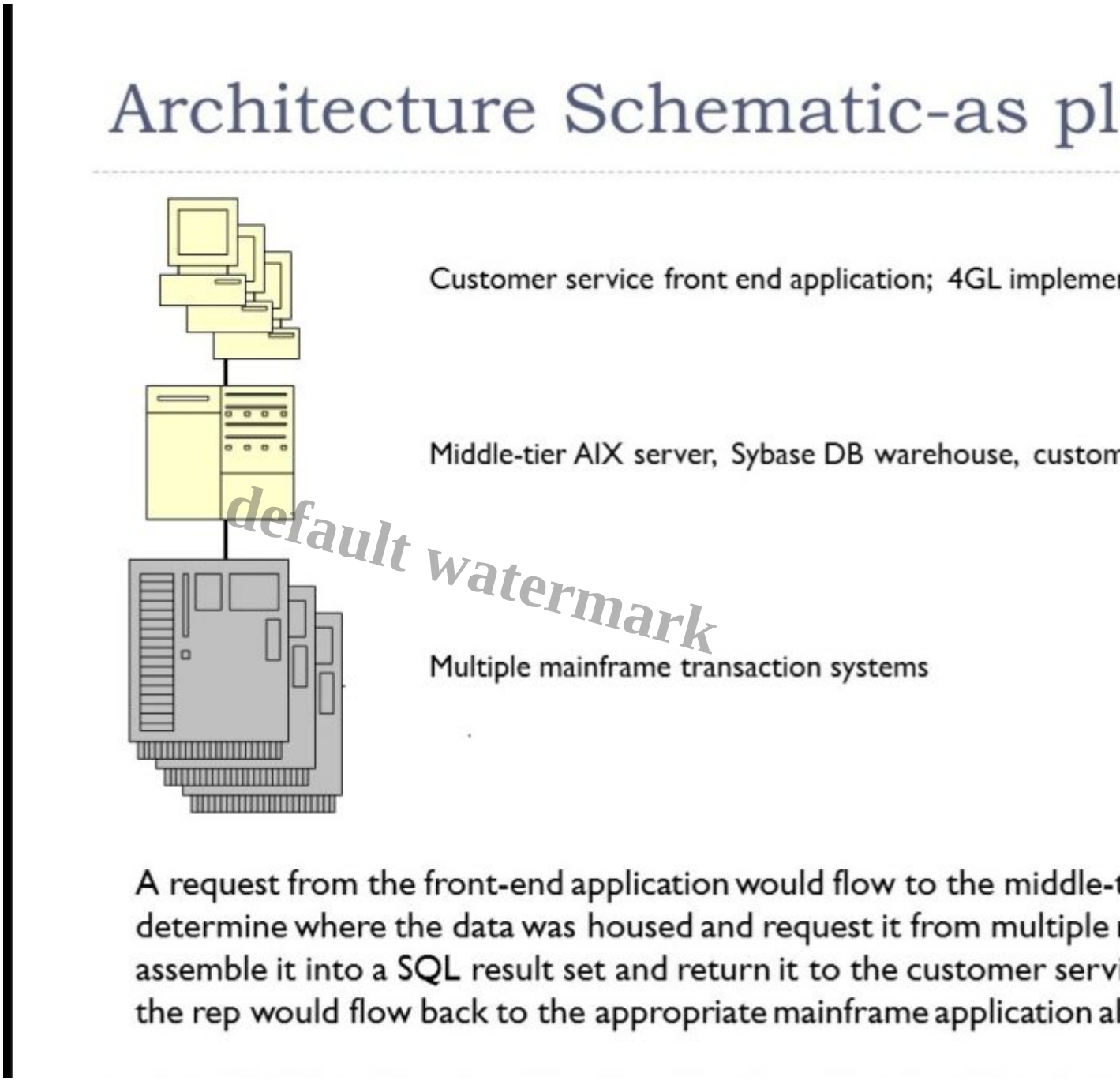
### THE PROJECT

This project was undertaken in the mid-90s, at which time commercial middleware and relational database replicators were just emerging.



The product was a system serving hundreds of customer service representatives in a call center operated by a prominent mutual funds management company.  It was designed to address the problem that information required to service a particular shareholderâ€™s needs might reside in multiple back-end systems, of which a rep could only access a limited number simultaneously.  This inability had caused a number of embarrassing incidents resulting from reps trying to service shareholders with incomplete or unavailable information.

The initial implementation architecture was multi-tiered, as depicted, below:




**INITIAL IMPLEMENTATION AND CHALLENGES**

One of the major challenges was staging the data from the source systems into the middle tier warehouse and packaging it into SQL result sets, with which the 4GL front-end application could communicate. In many cases, sequencing multiple requests to different source systems and performing significant data transformation was required to package the data so that it would be usable. Hundreds of transactions were built to support the overall application functionality. All of them were dependent on mapping data from source to warehouse to front-end application and the structure of any one of them could change as the application evolved.

An information mapping spreadsheet was designed to track the data from source to target through the application tiers and additional information was included to control sequencing and data transformations. A C program was built to manage each transaction, including both reads and writes to the source systems. These hundreds of programs were generated from the application metadata in the spreadsheet and basic templates for the required C functions.

The advantages of this approach included:

- drastically reduced effort to build and maintain the middle-tier code,
- consistent structure across the C modules and
- almost error-free implementation, once the templates had been perfected.

**FURTHER CHALLENGES**

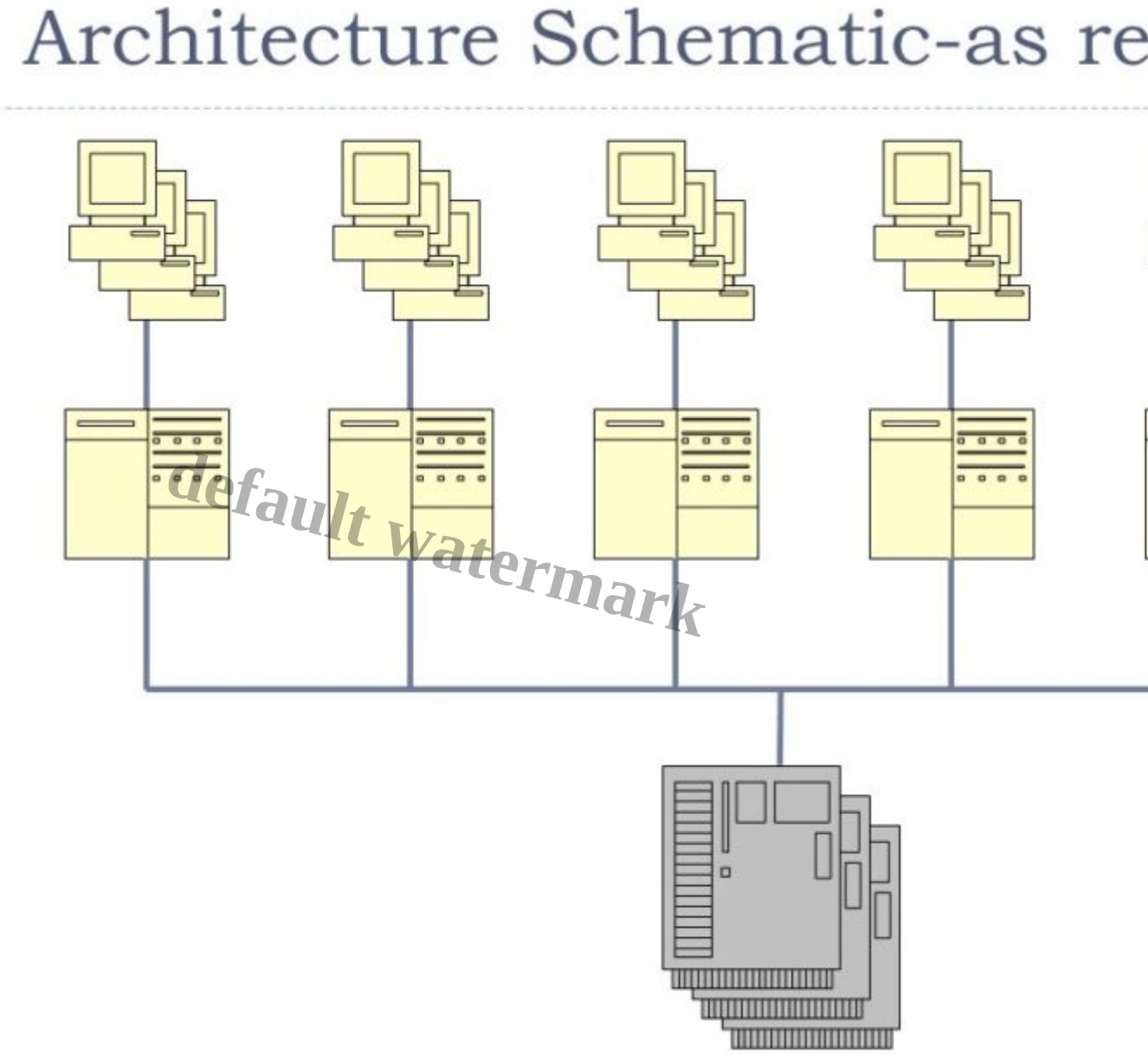
After enough of a cross section of the application had been implemented, testing revealed that a single instance of the middle tier would not provide adequate performance for the number of users that solution was intended to service. Two changes were made sequentially to address this shortfall:



- Relatively static data, such as lists of state abbreviations, were read into and cached within the middle tier, reducing the number of calls to the backend systems and the load on the middle tier server. This required restructuring of the entire set of C modules, which would have been prohibitively time-consuming if not for the ability to modify the metadata and re-generate them. Unfortunately, this by itself did not produce the required performance enhancement.

- The middle tier components were replicated, with each instance dedicated to a group of users that could be adequately served.

The revised implementation architecture is depicted, below:




Replicating the middle-tier components, which now contained cached application data, brought new requirements:

- The cached static data, called *reference data,* needed to be updated periodically and synchronized across all middle tier warehouse instances.
- Any programming or data structure changes required would have to be propagated to each instance of the middleware and warehouse and reference data in the warehouse would have to be preserved to the degree possible.

**MEETING THE SUBSEQUENT CHALLENGES**

Ultimately, generative software engineering was used to solve these problems, as well. An automated process to make data structure changes to the warehouse, propagate them to all instances of the middle tier server and unload, transform and reload cached reference data was built. In addition, a custom-built data replicator was implemented from the metadata contained in the spreadsheet to manage the process of keeping the reference data up to date in one master instance of the warehouse and then replicating the updates to the other instances.

**SUMMARY**

This case study illustrates the benefits of generative software engineering:

- accelerated delivery,
- architecture-focused, rather than implementation-focused, development,
- significant error reduction and
- higher quality software resulting from increased ability to iterate and refactor the code base as it evolves.



In the context of reducing the project risk in the case study, generative software engineering enabled the project team to:

- deliver a working cross-section of the project quickly, which promoted early detection of the performance issue while there was still an opportunity to address it,
- implement and then iterate two solutions to the performance issue; in the first case, requiring substantial changes to the initial middle tier design and in the second, requiring design and implementation of unplanned-for components,
- deliver an architecturally-consistent and relatively error-free solution.

**Date Created**
2015/03/31
**Author**
howardmwiener