Keyhole Software White Paper

COBOL MODERNIZATION STRATEGY WITH SPRING BATCH

Uncover a strategy for modernizing legacy COBOL batch processing applications by leveraging the powerful capabilities of Spring Batch.

KEYHOLE SOFTWARE

asktheteam@keyholesoftware.com
Phone: 877-521-7769

https://keyholesoftware.com
Kansas City, St. Louis, Denver, USA

> *"This strategy positions Spring Batch as an irresistible proposition for organizations ready to bid farewell to their COBOL legacy."*

CONTENTS

COBOL Modernization Strategy with Spring Batch

Introduction 3
White Paper Background 5
COBOL & Batch Processing 5
Modern Challenges For COBOL The Enterprise 6
COBOL Modernization Strategies 7
Code Generation 7
Pros of Code Generation 7
Cons of Code Generation 7
Rewriting COBOL from Scratch 8
Pros of Rewriting from Scratch 8
Cons of Rewriting from Scratch 8
Keyhole Advice 8
Modern Batch Processing 9
Introduction to Spring Batch 9
How Spring Batch Works 10
Chunking Is The Key 11
The Rewrite Experience 13
Rewrite Cost Savings Proposition 13
AI Assistance Unleashed 14
Example Generating Elements with ChatGPT or CoPilot 14
Generating an Entire Application 16
Key Team Roles and Responsibilities 19
Key Personnel 19
Spring Batch Developers 19
Domain Analyst 19
Testers 20
Project Management / Scrum Master 20
Establishing a Test Environment 20
Job Scheduling 21
Conclusion 22

INTRODUCTION

Introduction

In the fast-paced landscape of modern technology, the longevity of COBOL stands as a testament to its enduring relevance in enterprises today. The transformation from legacy COBOL systems to contemporary frameworks is a critical journey for enterprises aiming to enhance efficiency, scalability, and maintainability.

This white paper focuses on COBOL modernization, specifically in batch processing, proposing a strategy with Spring Batch. This paper provides a comprehensive guide based on tangible experience for enterprises navigating the realm of COBOL modernization.

Key sections include:

The Legacy Challenge:
Fear of disrupting critical processes and the absence of a clear migration path have led to the reluctance to remove legacy COBOL systems. We explore two distinct approaches to COBOL modernization—code generation and rewriting applications from scratch. This section sheds light on the complexities associated with COBOL and sets the stage for exploring how Spring Batch emerges as a compelling solution.

Introduction to Spring Batch:
Enter Spring Batch—an influential framework built atop the Spring Framework. Widely adopted for its scalability, performance optimization, and robust error-handling mechanisms, Spring Batch is a modern alternative to COBOL. This paper explores the features that make Spring Batch the preferred choice for batch processing applications, detailing how it facilitates the automation of large-scale data processing tasks.

The Rewrite Experience:
Keyhole's expertise in modernizing COBOL batch implementations highlights the importance of rewriting applications from scratch. While seemingly time-consuming, it offers a chance to eliminate redundant COBOL programs, optimize batch windows, and enhance system efficiency.

Leveraging AI Assistance:
Integrating advanced AI tools, such as ChatGPT or large language models (LLMs), emerges as a transformative factor. This technology proves invaluable, accelerating the modernization of legacy systems by simplifying complex tasks.

This white paper navigates through the intricacies of modern batch processing, focusing on the paradigm-shifting capabilities of Spring Batch within the Java ecosystem and its role in rewriting COBOL batch programs.

Renowned for our expertise, Keyhole Software consultants act as development "change agents," driving clients to success by leveraging cutting-edge software technologies and innovative approaches to gain a competitive edge.

ABOUT KEYHOLE SOFTWARE

Keyhole Software, a premier software development consulting firm, delivers customized technology solutions to clients nationwide. Our expert consultants, with an average of over 15 years of experience and five years of tenure, ensure unparalleled expertise in every project.

Keyhole specializes in enterprise software development, including web and mobile, microservices, cloud-based solutions, and legacy systems modernization, delivering tailored solutions that empower businesses to thrive in the dynamic landscape of software development.

Gold Microsoft Partner

Microsoft

aws partner network

Standard Consulting Partner

RELATED SERVICES

Assessment & Strategy Development:

Keyhole Software analyzes existing COBOL systems, identifying challenges and dependencies. Their experts formulate strategic modernization plans, integrating Spring Batch for efficient batch processing within the Java ecosystem.

Code Refactoring and Integration:

Keyhole Software specializes in refactoring COBOL code for enhanced maintainability. They seamlessly integrate systems with modern technologies, databases, and cloud services, ensuring streamlined batch processing with Spring Batch and related tools.

Spring Batch Development:

Keyhole Software leverages Spring Batch for efficient and scalable batch processing, optimizing reading, transforming, and writing of data within the Java ecosystem. They provide tailored solutions for enhanced performance and maintainability, covering job configuration to error handling.

CONTACT KEYHOLE SOFTWARE

keyholesoftware.com
asktheteam@keyholesoftware.com
877-521-7769

White Paper Background

COBOL & Batch Processing

COBOL stands for **CO**mmon **B**usiness-**O**riented **L**anguage. It was widely used for batch processing in mainframe applications due to several reasons.

1. Firstly, COBOL was designed specifically for business applications, with a strong focus on data processing and report generation. It provided a high-level programming language that allowed programmers to write code that closely resembled the natural English language, making it ***easier to read and understand***. This made COBOL a preferred choice for business-oriented applications, such as those found in mainframes.

2. Secondly, mainframe systems were primarily used for large-scale, high-volume data processing. COBOL offered excellent ***performance and efficiency*** in handling these large datasets. It could efficiently process extensive amounts of data in batch mode, where jobs are executed sequentially without user interaction. This made COBOL an ideal language for mainframe batch processing applications, which often involved processing massive amounts of data overnight or during off-peak hours.

3. Thirdly, COBOL had ***extensive support for file handling and data manipulation***. In batch processing applications, data is typically read from input files, processed, and written to output files. COBOL's rich set of file-handling features made it easy to read and write data to and from different file types, such as sequential files, indexed files, and databases. This flexibility allowed COBOL programmers to create efficient and streamlined batch processing applications.

4. Furthermore, the ***stability and reliability*** of COBOL played a crucial role in its adoption for mainframe batch processing. Mainframe systems were known for their robustness and uptime, and COBOL was well-suited to handle these critical applications. COBOL programs were reliable and provided strong error-handling capabilities, ensuring that batch jobs ran without interruptions and could recover from errors gracefully.

5. Lastly, the extensive knowledge and experience in COBOL programming were also contributing factors. During the time when mainframes and batch processing were prevalent, COBOL ***was a widely taught and used language***. Many programmers and IT professionals were well-versed in COBOL, making it easier to find and maintain a skilled workforce to develop and support mainframe batch processing applications.

In summary, COBOL's suitability for business applications, performance, file handling capabilities, reliability, and the availability of skilled programmers contributed to its widespread use in mainframe batch processing applications. These factors made COBOL a go-to language for building efficient and reliable data processing systems on mainframes.

Modern Challenges For COBOL The Enterprise

One key reason for its enduring popularity is its widespread usage in legacy systems. Many large organizations have heavily invested in COBOL applications over the years, creating vast codebases that continue to power critical business processes.

Quantifying the exact number of large enterprises still utilizing COBOL for batch processing proves challenging. Nevertheless, the continued significance of COBOL is unmistakable, particularly within pivotal sectors such as banking, insurance, government, and healthcare. These industries boast extensive legacy systems entrenched in COBOL, adept at managing substantial datasets through the indispensable mechanism of batch processing.

While modernization efforts are underway to replace or modernize legacy COBOL systems, the transition is a complex and expensive process. As a result, many large enterprises opt to continue to rely on COBOL batch processing to ensure the stability and reliability of their core business operations. Despite its enduring prevalence, COBOL faces distinctive challenges.

- ✔ A pivotal issue is the ***scarcity of COBOL programmers***, primarily stemming from the age demographic of professionals versed in this language. Developed in the late 1950s, COBOL gained prominence in the 1960s and 1970s, leading to a current workforce predominantly comprising individuals from an older generation who learned the language early in their careers.
- ✔ Another challenge is the ***lack of new talent*** entering the field. Many universities and educational institutions have phased out COBOL from their curriculum in favor of more modern programming languages. This means that younger generations of developers are not being exposed to COBOL or acquiring the necessary skills to work with it.

As a result, companies relying on COBOL systems face a scarcity of skilled professionals. They often struggle to find qualified candidates to maintain and update their legacy COBOL codebase. This shortage poses a significant risk, as these systems play a critical role in various industries such as banking, insurance, and government.

Addressing this problem requires a multi-faceted approach. Companies can invest in training programs or partnerships with educational institutions to bridge the knowledge gap and encourage new talent to learn COBOL. Additionally, efforts can be made to modernize COBOL systems and integrate them with more contemporary technologies, making them more appealing to younger developers.

Overall, the scarcity of COBOL programmers represents a significant challenge for organizations that rely on legacy systems. Finding innovative solutions and fostering a new generation of COBOL programmers is crucial to ensuring these systems' continued functionality and success. Alternatively, modernization.

COBOL Modernization Strategies

In the realm of COBOL applications, the predominant nature is batch processing, involving executing a series of jobs or tasks without user interaction, typically scheduled to run at a specific time or triggered by certain events. COBOL's strength in processing large amounts of data makes it well-suited for batch operations.

There are two primary ways to modernize batch COBOL applications. One is to rewrite applications from scratch using a modern application stack, and another is to utilize a code conversion application that converts the legacy COBOL source code to a modern application stack.

Both approaches have pros and cons, so let us review them briefly before giving the Keyhole Software recommendation.

Code Generation

There are many applications that will transform COBOL source code into a targeted modern language. Here are the pros and cons of this approach.

Pros of Code Generation

Time and cost-efficiency:
Automated code converters convert COBOL code to a more modern programming language, reducing the need for manual rewriting. This approach can significantly speed up the modernization process, resulting in cost savings.

Preservation of business logic:
Code converters often maintain the original functionality and business logic of the legacy COBOL application. This ensures that the modernized version performs the same tasks and meets the existing functional requirements.

Cons of Code Generation

Limited modernization potential:
While code converters transform COBOL code into a modern language, they may not fully leverage the benefits of newer technologies. Organizations may miss out on advancements in performance, scalability, and other modern features.

Debugging complexities:
Converted code can be difficult to debug, as it may lack proper documentation or adhere to different coding standards. This can make it more challenging for developers to identify and fix issues, adding complexity and time in the long run.

Rewriting COBOL from Scratch

Pros of Rewriting from Scratch

Enhanced flexibility and scalability: Rewriting applications from scratch allows organizations to leverage the latest technologies and frameworks, providing opportunities for enhanced flexibility, scalability, and performance.

Increased maintainability: By rewriting the application, developers can create a clean and modular codebase, making it easier to maintain, update, and add new features as the system evolves.

Cons of Rewriting from Scratch

Time and cost constraints: Rewriting from scratch can be time-consuming and costly, especially for large and complex applications. The development process may require extensive resources, including skilled developers and project management efforts.

Business risks and disruption: Rewriting applications may introduce risks, including compatibility issues, data migration challenges, and potential disruption to the business during the transition period. This requires careful planning and potential downtime.

Keyhole Advice

Based on our extensive hands-on experience at Keyhole Software, where we have engaged with both code conversion and rewriting approaches, we firmly advocate for the rewrite option as the most cost-efficient and business-supportive strategy.

- ✔ Despite the initial allure of code conversion's apparent speed and cost-effectiveness, our experience highlights higher long-term expenses due to technical debt and maintainability issues.

- ✔ Our experience has demonstrated that ***rewriting legacy batch programs yields tangible advantages when compared with code generation***. Inevitably, rewriting results in identifying and eliminating redundant legacy jobs whose original purposes have become obsolete. Many legacy jobs are culled since their purpose is no longer required.

While organizations may hesitate to remove outdated jobs, our experience shows that doing so streamlines processing, efficiently utilizes resources, and expedites the entire batch cycle. This streamlined process is easier to maintain, reducing complexity and enhancing system reliability.

Essentially, the rewrite approach acts as a transformative tool, eliminating operational noise and aligning batch processes with current organizational needs. While code conversion may seem faster and cheaper initially, companies must factor in the technical debt and maintainability burden in the long run.

Modern Batch Processing

Introduction to Spring Batch

Spring Batch is a popular framework used for batch processing applications in the Java ecosystem. Built on top of the Spring Framework, it provides a powerful and flexible infrastructure to develop, execute, and manage batch jobs efficiently.

There are several reasons why Spring Batch applications are widely used for batch processing:

Scaling and Performance:
Spring Batch enables parallel and distributed processing, allowing applications to handle large volumes of data and process them efficiently. It supports various optimization techniques, such as multithreading, partitioning, and clustering, to enhance performance and scalability.

Robust Error Handling:
Batch processing involves handling a large number of records, and it is crucial to handle exceptions and errors effectively. Spring Batch provides robust error-handling mechanisms, allowing developers to define retry policies, skip policies, and customize error-handling strategies. It ensures that batch jobs can handle errors gracefully and continue processing without manual intervention.

Transaction Management:
Batch processing often involves complex data operations, and maintaining data consistency is essential. Spring Batch integrates smoothly with Spring's transaction management capabilities, enabling developers to define transaction boundaries properly. It ensures that data modifications within a batch job are atomic and consistent.

Job Scheduling and Monitoring:
Spring Batch integrates with various scheduling frameworks, such as Quartz, to schedule batch jobs at specific intervals. Additionally, it provides comprehensive monitoring and reporting facilities. Developers can configure job status listeners, implement custom metrics, and visualize job execution details using Spring Batch's monitoring and management APIs.

Extensive Job Processing Features:
Spring Batch offers functionality that simplifies common batch processing tasks. It provides comprehensive data readers and writers for various data sources, including files, databases, and web services. It also supports complex transformation and validation operations through its item processors. Furthermore, Spring Batch includes features like chunk-based processing, partitioning, and restartability, making it easy to build efficient and reliable batch applications.

Spring Batch is widely used for batch processing applications due to its scalability, performance, error-handling capabilities, transaction management, job scheduling, and extensive job processing features. It empowers developers to create robust and efficient

batch applications and provides a foundation for managing complex data processing workflows.

How Spring Batch Works

Spring Batch is a lightweight framework that provides developers with the necessary tools to design, configure, and execute batch-processing workflows.

Spring Batch is designed to handle the demands of batch processing that are traditionally handled by “Big Iron” mainframes. The COBOL language is built for the mainframe's ability to process large data sets. This is the main feature of Spring Batch that allows it to be a suitable replacement for COBOL.

The framework also offers various features like parallel processing, chunk-based processing, transaction management, and error handling, which enable developers to build robust and scalable batch processing applications. By leveraging Spring Batch, developers can simplify the implementation of complex batch jobs, enhance performance, and ensure reliability in data processing tasks.

- ✔ The role of a Spring Batch **`job`** is to facilitate the automation of large-scale data processing tasks. A Spring Batch job typically consists of one or more steps, where each step represents a specific task or operation, such as reading data from a source, processing or transforming it, and writing it to a target destination.

Jobs are defined using elements provided by the framework that represent “batch” programming nomenclature. These elements are shown in the diagram below.

Job — **The entry point for configuring and orchestrating the job execution. This provides a way to configure listeners and various job parameters that can be accessed during runtime.**

Step — **The Step element in Spring Batch is a logical unit of work that represents a single phase within a batch job. It encapsulates a sequence of one or more batch processes that need to be executed in a specific order.**

Task — **In Spring Batch, the Task element is a fundamental component representing an individual work unit within a batch job.**

These elements are composed to define a batch execution `job`.

- ✔ The **`Step`** level in the Spring Batch is where the fundamental operations of reading, transforming, and writing data are orchestrated within a job execution.

Steps are structured with three essential parts: `reader`, `processor,` and `writer`.

- ✔ **Reader:** The `Reader` is responsible for fetching or reading data from diverse sources, such as databases, files, or external systems. It defines how data is input into the batch processing.

- ✔ **Processor:** The `Processor`, a crucial part of the `step`, is implemented to apply transformation and business logic to data flowing from reading to writing. It applies business logic, data manipulations, or any required transformations to the input data before passing it to the next stage.

- ✔ **Writer:** The `Writer` component handles the output part of the process. It defines how the processed data is written or persisted to the desired destination, such as databases, files, or external systems.

Spring Batch provides numerous reading and writing implementations, which saves a lot of time. Reading and writing implementations are provided for many common data sources; you can read or write from flat files, delimited or in SDF format, relational data sources, XML, and more. They are applied and customized using configuration parameters, which saves significant implementation time.

Additionally, step execution provides a place to apply transaction demarcation, ensuring data integrity throughout the batch processing workflow.

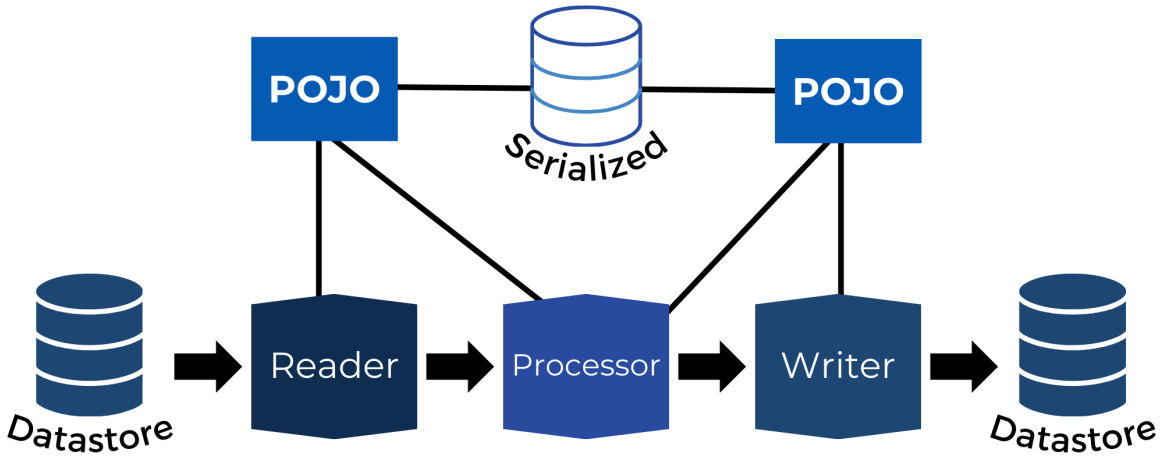
Chunking Is The Key

Ensuring robust error handling and recovery mechanisms during the execution of large-scale jobs, particularly within enterprises managing extensive data records, is a critical challenge. The complexity arises when errors occur mid-job, requiring a seamless rollback of changes and a restart from the point of failure.

Developing a custom solution for error recovery can be an intimidating endeavor. The intricacies of crafting a reliable mechanism to handle such scenarios pose significant challenges, making the "roll-your-own" approach a formidable task. Spring Batch addresses this complexity seamlessly, with its intrinsic support for error recovery and restarts, leveraging a fundamental concept known as "chunking."

- ✔ **Chunking** occurs at the step level—where readers, writers, and processors are defined—and where configuration parameters exist to define transaction management. Steps are intelligently configured to execute upon one or more designated records (chunks) at a time. In the event that an error or exception occurs, the job can be gracefully restarted from this point.

The data exchange between steps' reading, processing, and writing elements relies on Plain Old Java Objects (POJOs). These POJOs, encapsulating the data, are serialized within a chunk and persistently stored by Spring Batch for seamless deserialization during a restart. The diagram below illustrates step chunking behavior.



The Spring Batch framework provides a whole arsenal of `Readers` and `Writers` for multiple data types that are commonly used in enterprise batch applications. The framework also offers mechanisms for monitoring, control flow, parallelism, error handling, and launching of batch jobs.

The Rewrite Experience

Keyhole has helped numerous organizations modernize their COBOL batch implementations using the Spring Batch framework. We've also helped maintain platforms that were built using code generation tools, and through this experience, our recommendation remains steadfast to rewrite applications into Spring Batch applications from scratch.

Rewriting may seem like a time-consuming endeavor. However, with the ready-to-use and pre-built framework elements and the robust features of a programming language like Java, it requires significantly less code than the original COBOL program.

As a positive side effect, there's a good chance, during the rewrite process, that you will find COBOL programs that are no longer needed and can be eliminated from the batch window. It seems common for organizations over time to not remove these programs for fear of them breaking something unknown; it was safer just to let them run.

Rewrite Cost Savings Proposition

While IBM doesn't publicly disclose the monthly leasing costs of a mainframe, it is widely understood that the investment is substantial, particularly for large entities like banks, insurance companies, or government agencies utilizing advanced Z series mainframes. Generally, the cost can range from hundreds of thousands to millions of dollars annually, depending on contract negotiation, the specific model and configuration of the mainframe, the scale of computational power required, and the usage patterns.

Calculating the "real" cost of migrating a batch workload to cloud-native is difficult to do in a vacuum. Obviously, there are direct costs associated with running cloud-native workloads – things like compute instances, storage, networking, and data transfer, for instance – and those costs can be estimated and planned for. The major cloud vendors provide pricing details for each service offering, which can be used for fairly accurate cost estimation based on the scale required.

AWS publishes pricing sheets for all of their services – for example, compute capacity pricing for their on-demand, reserved, and spot instance pricing can be seen here:

- ✔ **https://aws.amazon.com/ec2/pricing**

Using the published pricing, it's straightforward to see that the cost for cloud-computing resources required to replace a legacy system is significantly less, even at large scale. Simply in terms of computing costs, you can see that moving the application to cloud-native will quickly pay for itself.

It's important to point out that a batch environment is not simply a virtual server with a JDK installed. Batch processing often needs to be choreographed and orchestrated in order to

mirror the throughput that a mainframe environment provides. But, the key takeaway is that pure computing power is relatively inexpensive.

There are other “costs'' that aren’t as straightforward to calculate but should be considered. Data migration, skill development and training, and operational costs of process change are among the shortlist that need to be accounted for.

Spring Batch provides a sufficient architecture for replacing COBOL applications and JCL, but other scheduling software, such as Quarkus, will have to be implemented in the cloud environment. This is discussed more in detail in the subsequent Job Scheduling section.

- ✔ AWS does offer a ready-to-use configured batch environment; information can be found at https://aws.amazon.com/batch.

It’s also important to point out that a cloud-based batch environment can be established using other top-tier cloud computing offerings, namely Microsoft Azure and Google. Cost differences with this approach would be negligible.

AI Assistance Unleashed

If you haven't yet explored ChatGPT’s[1] generative AI platform or other implementations like Bard[2] from Google, you have been missing out. This technology is a game-changer, especially for developers. It surpasses traditional Google search by providing precise solutions to prompts instead of a mere list of possibilities. What sets it apart is its contextual understanding, allowing users to fine-tune responses with ongoing prompts.

AI is causing—and will continue to cause—significant disruptions across industries. The software development field is currently experiencing this transformation. Reports highlight increased productivity as developers leverage AI, particularly features like GitHub CoPilot[3], which anticipates and suggests code improvements during active software development and problem-solving.

In the context of producing Spring Batch Jobs from COBOL listings, the GPT client, facilitated by tools like GitHub Copilot, proves to be a valuable asset.

One approach involves feeding sections of a COBOL program to ChatGPT, prompting it to convert them into Spring Batch elements. Alternatively, developers can seek human-readable explanations for specific sections of COBOL code from GPT, streamlining the modernization process.

Example Generating Elements with ChatGPT or CoPilot

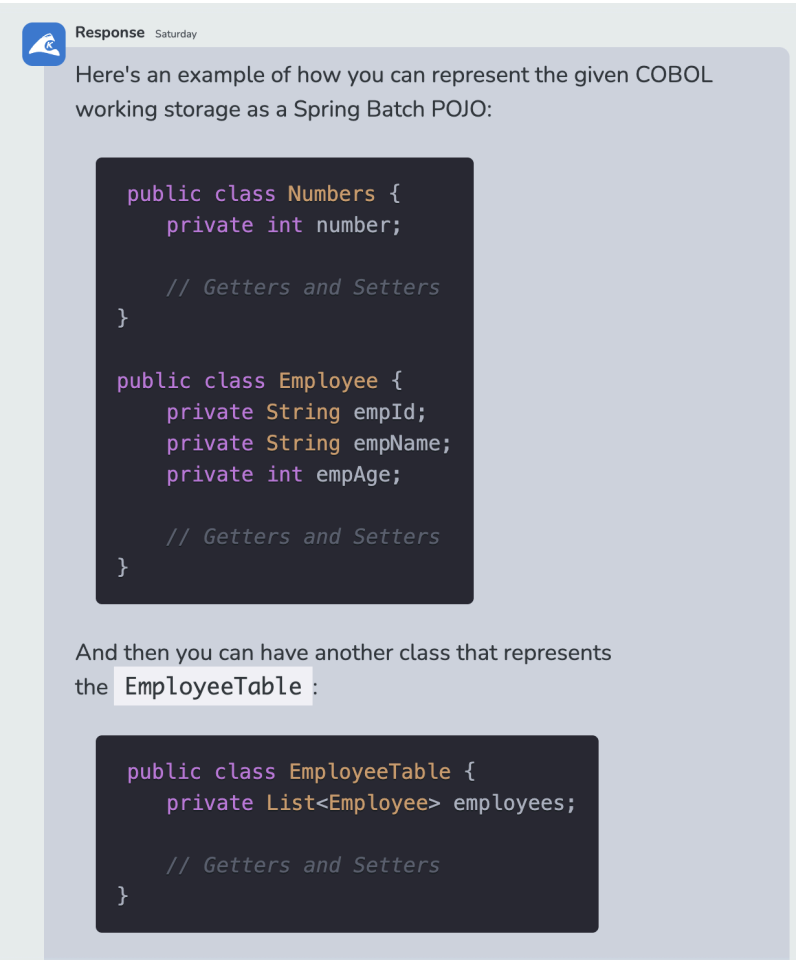
The example below demonstrates how ChatGPT can “assist” the developer in generating elements of a Spring Batch application based upon the corresponding COBOL source code.

[1] ChatGPT, https://chat.openai.com/
[2] Bard, https://bard.google.com/
[3] GitHub Copilot, https://github.com/features/copilot

In this example, we will use ChatGPT or CoPilot to transform a COBOL working storage structure into a POJO for Spring Batch. The provided working storage structure defines a table of employees with ID, name, and age fields.

```
WORKING-STORAGE SECTION.
      01 NUMBERS OCCURS 10 TIMES.
            05 NUMBER PIC 9(3).
      01 EMPLOYEE-TABLE.
            05 EMPLOYEE OCCURS 100 TIMES.
                  10 EMP-ID PIC X(5).
                  10 EMP-NAME PIC X(20).
                  10 EMP-AGE PIC 99.
```

Prompting ChatGPT to turn this into a Plain Old Java Object (POJO) results in the following:



Response Saturday

Here's an example of how you can represent the given COBOL working storage as a Spring Batch POJO:

```
public class Numbers {
    private int number;

    // Getters and Setters
}

public class Employee {
    private String empId;
    private String empName;
    private int empAge;

    // Getters and Setters
}
```

And then you can have another class that represents the `EmployeeTable`:

```
public class EmployeeTable {
    private List<Employee> employees;

    // Getters and Setters
}
```

As you can see, an entire COBOL program can be fed into ChatGPT and prompted to return an equivalent Spring Batch solution.

Generating an Entire Application

To illustrate ChatGPT's capabilities, let's also utilize it to generate an entire Spring Batch solution from COBOL source code. Our example will use a simple COBOL application which involves reading and writing from a person database, and outputting address information to a text file in SDF format.

```
IDENTIFICATION DIVISION.
     PROGRAM-ID. ADDRESS-REPORT.
     DATA DIVISION.
       WORKING-STORAGE SECTION.
          01 WS-DB-ERROR PIC X(100).
          01 WS-ADDRESS-REC.
                 05 WS-FIRST-NAME PIC X(20).
                 05 WS-LAST-NAME PIC X(20).
                 05 WS-STREET-ADDRESS PIC X(50).
                 05 WS-CITY PIC X(30).
                 05 WS-STATE PIC X(20).
                 05 WS-ZIP PIC X(10).
       01 WS-REPORT-LINE PIC X(80).
       01 EOF-FLAG PIC X VALUE 'N'.

PROCEDURE DIVISION.
       OPEN INPUT PERSON-DB.
       OPEN OUTPUT ADDRESS-REPORT.
              PERFORM UNTIL EOF-FLAG = 'Y' READ PERSON-DB INTO         WS-ADDRESS-REC AT END SET
       EOF-FLAG TO 'Y' END-READ
       MOVE WS-FIRST-NAME TO WS-REPORT-LINE
        MOVE WS-LAST-NAME TO WS-REPORT-LINE(21:20)
       MOVE WS-STREET-ADDRESS TO WS-REPORT-LINE(41:50)
       MOVE WS-CITY TO WS-REPORT-LINE(91:30)
       MOVE WS-STATE TO WS-REPORT-LINE(122:20)
       MOVE WS-ZIP TO WS-REPORT-LINE(143:10)

       WRITE ADDRESS-REPORT FROM WS-REPORT-LINE

END-PERFORM
CLOSE PERSON-DB, ADDRESS-REPORT.
STOP RUN.
```

Upon requesting ChatGPT to convert this COBOL program to a Spring Batch application, it produced the following Java Spring Batch source code.

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ItemWriter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
```

```
@Configuration
@EnableBatchProcessing
public class AddressReportBatchConfig {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public ItemReader<WsAddressRec> itemReader() {
        FlatFileItemReader<WsAddressRec> reader = new FlatFileItemReader<>();
        reader.setResource(new ClassPathResource("person-db.txt")); // Update file path
        reader.setLineMapper(new DefaultLineMapper<WsAddressRec>() {{
            setLineTokenizer(new FixedLengthTokenizer() {{
                setNames("wsFirstName", "wsLastName", "wsStreetAddress", "wsCity", "wsState",
"wsZip");
                setColumns(new Range(1, 20), new Range(21, 40), new Range(41, 90), new Range(91,
120), new Range(121, 140), new Range(141, 150));
            }});
            setFieldSetMapper(new BeanWrapperFieldSetMapper<WsAddressRec>() {{
                setTargetType(WsAddressRec.class);
            }});
        }});
        return reader;
    }

    @Bean
    public ItemProcessor<WsAddressRec, WsReportLine> itemProcessor() {
        return new AddressReportItemProcessor();
    }

    @Bean
    public ItemWriter<WsReportLine> itemWriter() {
        FlatFileItemWriter<WsReportLine> writer = new FlatFileItemWriter<>();
        writer.setResource(new ClassPathResource("address-report.txt")); // Update file path
        writer.setLineAggregator(new DelimitedLineAggregator<WsReportLine>() {{
            setDelimiter("");
            setFieldExtractor(new BeanWrapperFieldExtractor<WsReportLine>() {{
                setNames(new String[] { "wsFirstName", "wsLastName", "wsStreetAddress", "wsCity",
"wsState", "wsZip" });
            }});
        }});
        return writer;
    }

    @Bean
    public Step step(ItemReader<WsAddressRec> itemReader, ItemProcessor<WsAddressRec, WsReportLine>
itemProcessor, ItemWriter<WsReportLine> itemWriter) {
        return stepBuilderFactory.get("step")
                .<WsAddressRec, WsReportLine>chunk(10)
                .reader(itemReader)
                .processor(itemProcessor)
```

```
                .writer(itemWriter)
                .build();
    }

    @Bean
    public Job job(Step step) {
        return jobBuilderFactory.get("addressReportJob")
                .incrementer(new RunIdIncrementer())
                .start(step)
                .build();
    }
}

public class WsAddressRec {

    private String wsFirstName;
    private String wsLastName;
    private String wsStreetAddress;
    private String wsCity;
    private String wsState;
    private String wsZip;

    // Getters and setters

}

public class WsReportLine {

    private String wsFirstName;
    private String wsLastName;
    private String wsStreetAddress;
    private String wsCity;
    private String wsState;
    private String wsZip;

    // Getters and setters

}
```

ChatGPT not only effortlessly translated the COBOL logic into Java Spring Batch but also provided crucial guidance. It highlighted the use of Spring Batch's modular abstractions for reading and writing data, making the code more maintainable. Moreover, it emphasized the need for customization, reminding developers to handle exceptions appropriately and address specific requirements like file paths and delimiters.

You'll notice that the example reads and writes from a text file. As a specific data source wasn't defined, ChatGPT intelligently inferred the use of a text file for reading and writing data, showcasing its adaptability. However, for a more refined conversion, specifying the type of data source is recommended.

It's essential to acknowledge that the generated program does have some issues, namely referencing structures it didn't provide. However, it offers ***a substantial 90% solution with good naming conventions and structure***, resembling a human developer's. This makes

sense since the programming style is machine-learned from human-produced source code. It's important to note that compiling and making the program work will require an experienced developer familiar with Spring Batch.

Nevertheless, leveraging this generated code significantly reduces development time. This streamlined conversion process showcases how ChatGPT simplifies complex tasks, accelerating the modernization of legacy systems with precision and efficiency. As organizations embark on their modernization journey, the partnership with ChatGPT and its competitors emerges as a game-changer, unlocking new possibilities in code transformation and software development.

Key Team Roles and Responsibilities

If you go down the path of a "rewrite," there are key roles and responsibilities that must be assigned to your team.

Offshoring the development of the rewrite is questionable at best. There is a high amount of communication required between the analyst, developer, and other members of the team. Having to do this in different time zones and languages could be a barrier.

Key Personnel

Spring Batch Developers

It's essential to have developers well-versed in Spring Batch. Avoid the pitfall of assigning developers experienced in Java/Spring but unfamiliar with Spring Batch intricacies.

Keyhole has seen—***and rescued***—many projects where the lack of expertise in utilizing the Spring Batch framework led to issues. In such instances, despite initial appearances of functionality, these applications faltered under actual load due to improper implementation.

It's noteworthy that Spring Batch has no significant learning curve. Suppose you have an experienced Spring Batch developer. In that case, it is entirely feasible that this individual can mentor other experienced developers with a significant background in Java and Spring.

Domain Analyst

Another important team role is a domain analyst. The domain analyst focuses on the COBOL batch programs, understanding the current applications. The individual must understand—and preferably be currently working with—the COBOL applications. They can answer questions regarding the "what" and "where" of a given program.

It is important to note that COBOL is not a complex language to learn. We at Keyhole Software have found that the requirements documents are the actual COBOL source code listings. Spring Batch developers, with the help of the domain analyst, will quickly learn how to read them and will be able to rewrite the batch job in Spring Batch.

Testers

In this project, testing can primarily be conducted by the development team itself. The supporting test environment is described in the section below, emphasizing that the output from any given batch run will generate data in various formats.

Developers, intimately familiar with the output as they are responsible for writing the batch job that produces it, take the lead in validating and testing their work.

- ✔ A key validation approach involves comparing the production batch output from the previous day with the test data the team is currently working on. This validation process is straightforward, aiming for a perfect match between the output from the ongoing job and the previous night's production run.

Developers can leverage free data compare tools and utilities to facilitate this validation. These tools assist in meticulously assessing the consistency and accuracy of the modernized batch processes. This comprehensive validation process ensures that the modernized system aligns seamlessly with the established production standards and meets the expectations of reliability and precision.

Project Management / Scrum Master

You will need a Project Manager or Scrum Master to oversee the entire modernization project. They will be responsible for planning, coordinating, and meeting project milestones. The role ensures adherence to requirements and facilitates seamless communication among team members. The Project Manager or Scrum Master will meticulously track and monitor all jobs undergoing rewriting.

While we generally suggest an agile approach, whether adopting a waterfall or agile task assignment approach depends on your project preferences, as the requirements are largely black and white.

In summary, a well-rounded team comprising skilled developers, a domain analyst, meticulous testers, and an efficient project manager is integral to the success of the COBOL batch program rewrite. Clear communication and expertise in Spring Batch are central to overcoming potential pitfalls and ensuring a smooth transition.

Establishing a Test Environment

In the preceding section, we outlined the approach to testing rewritten batch programs, involving a comparison with production output from the previous day's batch run. To facilitate this process, it is essential to clone production data before executing the production batch jobs. This ensures that the test environment mirrors the conditions of the live production environment, allowing for a thorough and reliable evaluation of the modernized batch processes.

The cloned data source allows developers to run their new Spring Batch jobs against the same data that the COBOL production jobs were executed against.

Job Scheduling

An IBM mainframe constitutes a centralized, homogeneous system encompassing hardware, operating system, programming language, application lifecycle management, and job scheduling. Mainframe operating systems are equipped with Job Control Language (JCL), a defined language for scheduling and managing job execution.

Multiple solutions exist for job scheduling in Spring Batch, catering to different applications' varied requirements and constraints.

A popular and widely adopted option involves leveraging the intrinsic scheduling features offered by the Spring Framework. Spring's scheduling capabilities empower developers to configure and schedule tasks effortlessly through annotations or XML configuration. This approach is often suitable for simple scheduling needs, where tasks need to be executed periodically or at fixed intervals. It delivers flexibility and simplicity, addressing basic scheduling needs without the requirement for extra dependencies.

Another alternative is to use external schedulers like Quartz or cron expressions.
Quartz[4] is a popular open-source scheduler for Java applications that provides more advanced scheduling features, such as cron-like expressions for defining complex schedules. It integrates well with Spring Batch and allows developers to schedule batch jobs based on various conditions, such as specific dates, times, or intervals. Quartz can be configured and managed programmatically or through configuration files, giving developers greater flexibility and control over job scheduling.

Cloud-based solutions like **AWS Batch**[5] or **Google Cloud Scheduler**[6] present additional options for scheduling Spring Batch jobs. These services provide highly scalable and reliable job scheduling and execution capabilities within a cloud environment. They efficiently manage large volumes of data and processing tasks, proving beneficial for applications with elevated processing demands or unpredictable workloads. Additionally, these cloud-based solutions often include supplementary features like monitoring, fault tolerance, and auto-scaling, substantially enhancing the overall reliability and performance of batch job scheduling.

In summary, Spring Batch provides multiple solutions for job scheduling, ranging from built-in scheduling capabilities to external schedulers like Quartz or cloud-based services. The choice of solution depends on the application's specific requirements, such as the complexity of the scheduling needs, scalability requirements, and the desired level of control and flexibility.

[4] Quartz Job Scheduler, https://www.quartz-scheduler.org/
[5] AWS Batch, https://aws.amazon.com/batch/
[6] Google Cloud Scheduler, https://cloud.google.com/scheduler

CONCLUSION

Conclusion

In the face of 'zombie' COBOL programs haunting legacy systems, organizations are presented with an opportunity for strategic transformation. This white paper advocates rewriting COBOL batch programs using Spring Batch as a catalyst for change.

The transition to Spring Batch is not just a technological upgrade; it's a calculated move toward operational excellence. With its foundation in the Java ecosystem, Spring Batch integrates seamlessly with modern databases, cloud services, and other enterprise technologies, providing agility and interoperability that COBOL simply cannot match. An investment in COBOL modernization pays dividends in reduced operational costs and enhanced performance.

We've shown that Spring Batch significantly reduces batch processing windows, achieving efficiencies previously unattainable with COBOL. Optimized data handling, parallel processing, and harnessing modern hardware lead to faster data processing, reduced resource consumption, and increased throughput. We also introduced how ChatGPT and LLM tools like GitHub CoPilot can assist in the modernization process.

These technical improvements have real business implications, such as better decision-making through timely data availability, enhanced customer experiences through quicker processing times, and cost savings.

Rewriting in Spring Batch offers an escape from the cycle of dependency on outdated COBOL programs. This transition is pivotal for modernizing IT infrastructure, aligning with technological trends, and ensuring responsiveness to today's dynamic business environment. The enduring benefits—reduced operational costs, enhanced performance, and improved maintainability—solidify Spring Batch as an irresistible proposition for organizations ready to bid farewell to their COBOL legacy.

White Paper Author: David Pitt, Founder, Keyhole Software
Publishing Date: February 2024

Contact Keyhole Software

Company Website: https://keyholesoftware.com
Phone: 877-521-7769
Email: asktheteam@keyholesoftware.com
Headquarters: 11205 W 79th Street, Lenexa, KS 66214