Deloitte.

Refactoring the Approach to Legacy Application Modernization

GSAW 2023

Content

Legacy Space System Challenges

The time to migrate is now. Organizations are plagued with workforce aging, low reskilling in legacy code, and a small pool of cleared personnel. There is a growing demand for system modernization.

Legacy Space System Challenges

- High Costs
- Prolonged Reporting
- Security & Risk
- Inefficiency
- Mainframe Brain Drain
- System Misalignment
- Dependency on few SMEs

Benefits of Modernization

- Reduced Total Cost
- Enhanced Quality
- Reduced Risk
- Repeatable Approach
- Return on Investment
- Enhanced Interoperability
- Wider labor pool available

Application Modernization Solutions

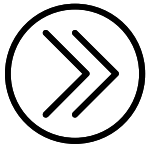
The integrated approach to refactor existing systems comprises of an end-to-end service to modernize systems with minimal risk and disruption, creating opportunities for the adoption of new services and analytical opportunities.

DISCOVERY

Know your legacy code

MINING

Understand the business functions of code



TRANSFORMATION

Transform legacy code into modern languages

LEGACY DEVOPS

Use DevOps in legacy and modern environments

MODERNIZATION

Adopt cloud, mobile, microservices, analytics and more

Transformation Approach

Refactoring and modernization enhancements occur in parallel tracks to deliver early and incremental progress over the life of the project.

TRANSFORMATION

Discovery

- Deliver system inventory report (code, data, interfaces)

Transformation Plan

- Develop plan (sequence, timeline) for refactoring

Refactor code, data, and user interface*

- Implement 100% automated process to refactor code, data, user screens
- Integrate code and data with interfaces and third-party software
- Automate testing to validate new system is functionally equivalent
- Acclimate development team with Java programming language, tools, frameworks

MODERNIZATION

Future-state roadmap

- Identify priority modernization areas (e.g., analytics)
- Create plan for subsequent modernization steps

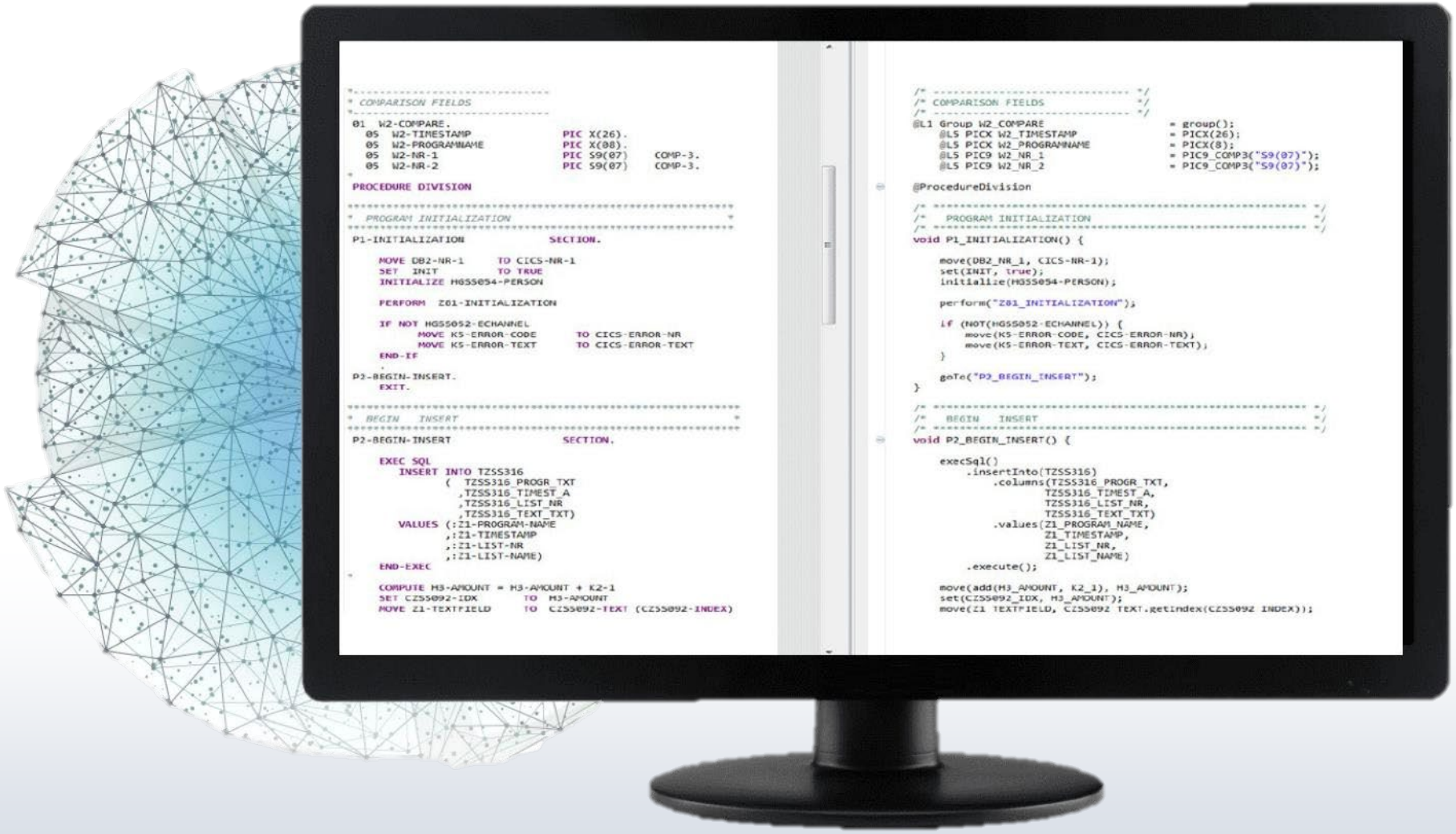
Modernization enhancements

- Analyze legacy system to derive requirements
- Implemented enhancements may include:
    - Analytics
    - Business Rules Extraction
    - Service-oriented architecture
    - Cloud deployment
    - Enhanced user interfaces

- **Additional opportunities to initiate DevSecOps and/or Cyber activities**

Transformation Example | From Cobol To Java

One-to-one conversion of all code to a modern language.



```
*-------------------------------
* COMPARISON FIELDS
*-------------------------------
01  W2-COMPARE.
  05  W2-TIMESTAMP              PIC X(26).
  05  W2-PROGRAMNAME            PIC X(08).
  05  W2-NR-1                   PIC S9(07)    COMP-3.
  05  W2-NR-2                   PIC S9(07)    COMP-3.
*
 PROCEDURE DIVISION

*-------------------------------------------------------*
*  PROGRAM INITIALIZATION                               *
*-------------------------------------------------------*
P1-INITIALIZATION            SECTION.

    MOVE DB2-NR-1      TO CICS-NR-1
    SET  INIT          TO TRUE
    INITIALIZE HGSS054-PERSON

    PERFORM  Z01-INITIALIZATION

    IF NOT HGSS052-ECHANNEL
         MOVE K5-ERROR-CODE       TO CICS-ERROR-NR
         MOVE K5-ERROR-TEXT       TO CICS-ERROR-TEXT
    END-IF
    .
P2-BEGIN-INSERT.
    EXIT.

*-------------------------------------------------------*
*  BEGIN   INSERT                                       *
*-------------------------------------------------------*
P2-BEGIN-INSERT              SECTION.

    EXEC SQL
       INSERT INTO TZSS316
              (  TZSS316_PROGR_TXT
                ,TZSS316_TIMEST_A
                ,TZSS316_LIST_NR
                ,TZSS316_TEXT_TXT)
       VALUES (:Z1-PROGRAM-NAME
               ,:Z1-TIMESTAMP
               ,:Z1-LIST-NR
               ,:Z1-LIST-NAME)
    END-EXEC
*
    COMPUTE M3-AMOUNT = M3-AMOUNT + K2-1
    SET CZSS092-IDX       TO  M3-AMOUNT
    MOVE Z1-TEXTFIELD     TO  CZSS092-TEXT (CZSS092-INDEX)
```

```
/* ------------------------------- */
/* COMPARISON FIELDS               */
/* ------------------------------- */
@L1 Group W2_COMPARE                = group();
    @L5 PICX W2_TIMESTAMP           = PICX(26);
    @L5 PICX W2_PROGRAMNAME         = PICX(8);
    @L5 PIC9 W2_NR_1                = PIC9_COMP3("S9(07)");
    @L5 PIC9 W2_NR_2                = PIC9_COMP3("S9(07)");

@ProcedureDivision

/* ------------------------------------------------------- */
/*   PROGRAM INITIALIZATION                                */
/* ------------------------------------------------------- */
void P1_INITIALIZATION() {

    move(DB2_NR_1, CICS-NR-1);
    set(INIT, true);
    initialize(HGSS054-PERSON);

    perform("Z01_INITIALIZATION");

    if (NOT(HGSS052-ECHANNEL)) {
        move(K5-ERROR-CODE, CICS-ERROR-NR);
        move(K5-ERROR-TEXT, CICS-ERROR-TEXT);
    }

    goTo("P2_BEGIN_INSERT");
}

/* ------------------------------------------------------- */
/*   BEGIN   INSERT                                        */
/* ------------------------------------------------------- */
void P2_BEGIN_INSERT() {

    execSql()
        .insertInto(TZSS316)
            .columns(TZSS316_PROGR_TXT,
                     TZSS316_TIMEST_A,
                     TZSS316_LIST_NR,
                     TZSS316_TEXT_TXT)
            .values(Z1_PROGRAM_NAME,
                    Z1_TIMESTAMP,
                    Z1_LIST_NR,
                    Z1_LIST_NAME)
        .execute();

    move(add(M3_AMOUNT, K2_1), M3_AMOUNT);
    set(CZSS092_IDX, M3_AMOUNT);
    move(Z1_TEXTFIELD, CZSS092_TEXT.getIndex(CZSS092_INDEX));
```

Overall Modernization Approach

An integrated approach to transforming legacy code into efficient, modern apps.

STEP 1 *Refactor legacy code & data*	STEP 2 *Assess current state after refactoring*	STEP 3 *Determine cloud modernization strategy*	STEP 4 *Make apps cloud-ready*	STEP 5A/5B *Building Iaas & Cloud-native apps*
• Convert old code and data to a modern language/datatypes* • Integrate code & data • Validate system functionality • Acclimate developers to Java • Migrate code to IaaS	• Select applications based on functionality & priorities • Identify apps that address specific business needs • Assess apps for cloud sustainability	• Refine: Identify apps that can be modernized • Replace: Move apps to cloud & retire existing applications • Enhance: Identify apps that can be made cloud-native	• Design app as services, then combine services • Decouple data & separate components • Revise APIs between apps • Design for scaling & performance	• 5a: Build Iaas apps • 5b: Build Cloud-native apps

* Modern user experiences can be achieved as early as step 1 with UI redesign

Transformation/Modernization Benefits

Transformation can help boost efficiencies, reduce risks and costs, and prepare your organization to implement new technologies.

Reduced risk

No functional requirements, no changes in functionality, minimal end-user training

Reduced total cost

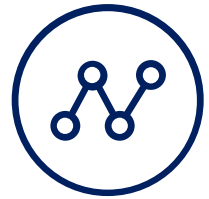
Refactoring prior to modernization reduces need for mainframe developers; rapid migration can quickly cut operational costs

Enhanced quality

1-to-1 test scenarios deliver highly accurate releases, without interruption to business

Enhanced interoperability

Mobile and Web services apps streamline interoperability and relational databases result in easier future changes



Repeatable approach

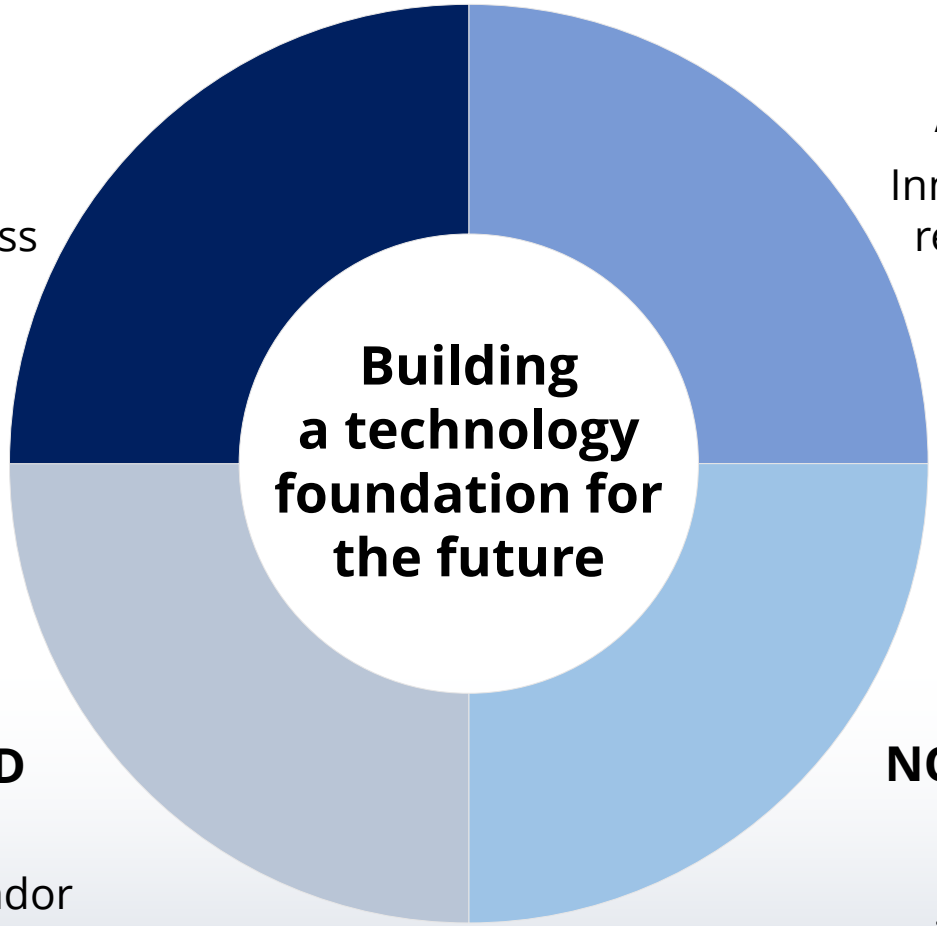
Agile application development life cycle provides a controllable process to coordinate releases

Staff Transition

Refactoring supports legacy developers' transition into modern Java developers by building on their legacy skillsets

Approach Differentiators

An end-to-end, 100% automated application modernization results in a fully functional updated system with no performance disruption.



EXPANSIVE NETWORK

Highly skilled employees across geographies

FULLY AUTOMATED

Innovative software refactors all code and preserves functionality

END-TO-END SOLUTION

One trusted vendor for the entire project

NO CODE FREEZE

Update legacy systems, often without disruption