JANUARY 31, 2017

SUBTASK 4.6 REPORT

EVALUATION OF THE CODING OF THE MODELS

One of Three 18-Month Reports

IN-DEPTH TECHNICAL REVIEW OF PENSION BENEFIT GUARANTY CORPORATION'S MULTIEMPLOYER AND SINGLE-EMPLOYER PENSION MODELS

Prepared for:

Social Security Administration
6401 Security Blvd.
Baltimore, MD 21235
Contract SS00-15-30598

Prepared by:

FTI Consulting, Inc.
3 Times Square
New York, NY 10036

CRITICAL THINKING
AT THE CRITICAL TIME™

Contents

Introduction

In July 2015, the Social Security Administration (SSA) engaged the FTI Consulting team (FTI) to conduct an 18-month, in-depth technical review of the Pension Benefit Guaranty Corporation's (PBGC) single-employer (SE) and multiemployer (ME) Pension Insurance Modeling System (PIMS). Task 4 of the Statement of Work (SOW) consists of 10 subtasks required for this in-depth review - nine specific areas of review and a final report.[1] Three of the subtask reports are due at the end of each of the six-, 12- and 18-month periods. This report for Subtask 4.6, along with those for Subtasks 4.4 and 4.8, is one of the three reports due at the end of the 18-month period (with approved extension). As a part of our review of PIMS, this report documents our evaluation of the coding of the models and the associated documentation.

While most of the subtask reports thus far have been devoted to the design and theory behind PIMS, this report is focused on the actual implementation. This report addresses the following key questions raised in Subtask 4.6:

1. Does the coding of algorithms in the model appropriately reflect the documentation?
2. Are the assumptions, algorithms, and procedures reasonable?

Further, as was noted in the Subtask 4.5 report, since the code review was one of the last portions of our evaluation, the examination of coding accuracy was deferred until this subtask. As a result, we have also addressed coding documentation within this report.

[1] Social Security Administration, Evaluation of the Pension Benefit Guaranty Corporation's Pension Models, Description/Specification/Work Statement, page 14.

Authors and Contributors

Authors

Alex Arnote
FTI Consulting, Inc.

Liaw Huang, Ph.D., FSA, MAAA, EA, FCA
The Terry Group

Contributors

Jeff Leonard, FSA, EA, MAAA
FTI Consulting, Inc.

John Moore, FSA, MAAA, EA, FCA
The Terry Group

Findings

During our review of the PIMS coding, we found that basic system design and descriptive documentation was out-of-date or non-existent. For example, neither SE- nor ME-PIMS has a process flow diagram, module library, or variable dictionary. Of particular concern was the lack of design documents for ME-PIMS, which was only recently developed. The only design documentation we received for ME-PIMS was a series of change requests sent to the developer. Because there was little documentation with which to compare the code, much of our review was focused on identifying the design of the model and generating our own process documentation.

Based on our review we recommend the following actions be taken by PBGC, in priority order:

1. Provide an addendum to the PIMS 2010 Guide and the "Key Differences" documents (as referenced below) to include the changes listed based on our module review (under Algorithm Implementation Review Findings, beginning on page 10).
2. Update documentation to reflect methodologies and parameters which appear in the Statistical Analysis Software (SAS) code or as comments in PIMS C++ code but are not currently documented.
3. Refrain from using hard-coded values within the code for values that are not static.
4. Utilize industry best practices for system documentation when designing and implementing future versions of PIMS.
5. Consider creating guidelines around the use of descriptive variables in future programming development.
6. Although a major undertaking, consider rewriting the program from scratch rather than modifying the legacy code, which would result in improved performance and reduced maintenance costs (due to code reduction).

Scope and Methodology of Our Evaluation

This report addresses the accuracy of the implementation of PIMS. Our recommendations are directed towards the Policy, Research and Analysis Department (PRAD) as well as the PBGC Office of Information Technology (OIT) with the goal of ensuring that PIMS has been implemented as designed. This report does not address the adequacy of PIMS modeling techniques, model assumptions, or the change management process, which are covered in other reports in our in-depth technical review.

As part of this evaluation we identified and reviewed the following documents provided by PBGC:

1. **Pension Insurance Modeling System, PIMS System Description, Version 1.0, 2010 ("PIMS 2010 Guide"):** This is a comprehensive description of PIMS modeling methodology, data, assumptions, equations, parameters, and simulation process and code organization. The system description is written for SE-PIMS. ME-PIMS follows similar code organization but has significant differences due to a different pension insurance program design and regulatory framework.
2. **Multiemployer PIMS, System-Validation Documentation, Key Differences Between SE-PIMS and ME-PIMS, 2011 ("Key Differences"):** This document is intended to be read together with the PIMS 2010 Guide. It sets forth, in considerable technical detail, additional algorithms used in ME-PIMS, including the modeling of the Pension Protection Act of 2006 (PPA), PPA statuses, funding improvement and rehabilitation plans, plan sponsor contributions, and mass withdrawals.
3. **Verification and Quantification of Buck's Recommended Changes, 2014:** This document discusses PBGC's research in validating Buck's recommended changes to ME-PIMS. It provides changes to ME-PIMS subsequent to the Key Differences document.
4. **2014 Projections Report:** The appendix to the 2014 Projections Report contains up-to-date assumptions and methods not available in the other documents (e.g., the modeling of the Multiemployer Pension Reform Act of 2014 (MPRA)).

The SOW contemplated that our work would include a comparison of code to documentation. However, the lack of existing system documentation required that we first generate our own module libraries and process flow diagrams. Once created, this FTI-generated documentation was used to determine which of the thousands of modules needed to be included in our review. Inactive code and standard/purchased code libraries were excluded from our review.[2]

Once identified, we conducted a cursory review of the relevant modules focusing on the following questions:

1. Which model (ME or SE) used the module and if used by both, were they the same versions for both models?
2. Does the module contain calculations or formulas of an actuarial or economic nature?
3. Does the module contain any hard-coded values?

This resulting information was then used to identify which modules would be targeted for a detailed review and comparison to the system documentation.

[2] For the purposes of these reports, "inactive code" refers to modules which are never called, either directly or indirectly, by the main process.

Code Review

Identification of Inactive and Third Party Code

During the initial project kickoff meeting, FTI was informed by PBGC staff that SE-PIMS included over one million lines of code. Given the extensive age of the application and the lack of provided documentation, we assumed that a great deal of this code would be inactive and therefore not subject to the review. Our first step was to utilize automated tools to work through the code and identify which modules were never called by the main process and which code was provided by third parties, as these modules were not subject to our review. As shown in the table below, 31% of SE-PIMS code and 41% of ME-PIMS code modules were identified as inactive.

	SE-PIMS	ME-PIMS
Total Functions		
Total Class-Level Functions	1,195	1,246
Total File-Level Functions	1,935	230
3rd Party Functions		
3rd Party Class-Level Functions	315	0
3rd Party File-Level Functions	1,925	2
Total Functions (Excluding 3rd Party)	890	1,474
Functions Called by Main Process (Excluding 3rd Party)	614	873
Functions Not Called by Main Process (Excluding 3rd Party)	276	601
Percent of Functions Inactive	**31%**	**41%**

In addition to the inactive functions, there are numerous variables and commented-out sections of code that appear to be obsolete or that were only intended for temporary use. For example:

```
//oIndBenFlat.basic_ben = 0.00001;  // JPWxTemporary REMOVE THIS

// Per Jeff Lane (e-mail from March 28, 2014 10:35 AM), this should never be called

// NOTHING IS DONE WITH THESE VALUES!!! // SQLS
//long ebc_age = SQLS::comm_xra.fld_ebc_age;
//long eubc_age = SQLS::comm_xra.fld_eubc_age;
//long xra = SQLS::comm_xra.fld_xra;

// VALUES ARE NOT USED!!! AFC // SQLS
//double ir_t = SQLS::comm_bc_age.fld_ir_t;
//double avg_red = SQLS::comm_bc_age.fld_avg_red;
//long tvbc_age = SQLS::comm_bc_age.fld_tvbc_age;
```

Review of Code Modules

Once the active, PBGC-built modules had been identified, each was given a cursory review to identify if it contained any hard-coded values or calculations. There were many modules that appeared in both ME- and SE-PIMS, and these were compared programmatically to determine if any differences existed. If no differences were identified, only the SE-PIMS module was reviewed. Otherwise, only the differences in the ME-PIMS version were reviewed. A complete list of the modules reviewed is included as Exhibit 1 with additional detail available upon request.

Process Flow

FTI then created process flow diagrams using automated tools in order to identify the order in which modules were called and to identify which modules were the key drivers of the process. The resulting process flow diagrams of the main event loops for SE- and ME-PIMS have been included as Exhibit 2 and Exhibit 3, respectively.

Code Review Findings

Because very little coding documentation exists for PIMS, we found no discrepancies of note. However, during our review, FTI did note several items that PBGC should consider addressing:

1. FTI identified hard-coded values throughout the PIMS code. While many of these are appropriate, there are many cases where the values used could change in the future, either permanently or due to, for example, an ad hoc request. For future program changes PBGC should consider using values stored in a SQL server table in place of hard-coded values whenever the value has the potential to change. For example, the below code in PIMS is currently assigned static values and would need to be manually updated every time the values changed:

```
oAct_flat_cashflow.aNewHireHeadcountDist[20] = 0.24870;
oAct_flat_cashflow.aNewHireHeadcountDist[25] = 0.16640;
oAct_flat_cashflow.aNewHireHeadcountDist[30] = 0.14240;
oAct_flat_cashflow.aNewHireHeadcountDist[35] = 0.11600;
oAct_flat_cashflow.aNewHireHeadcountDist[40] = 0.10560;
oAct_flat_cashflow.aNewHireHeadcountDist[45] = 0.08830;
oAct_flat_cashflow.aNewHireHeadcountDist[50] = 0.06580;
oAct_flat_cashflow.aNewHireHeadcountDist[55] = 0.06680;
```

2. The code contains many variables that drive calculations throughout the process. There should be a dictionary that lists every variable used within the program and describes what it represents and how it is used. Barring this, the code should consistently include a short description when the variable is created. Such documentation is even more important given the long life and frequent updating experienced by the PIMS models. As an example, here is a comment from one of the asset modules which illustrates the type of comments we recommend be used more frequently and consistently:

```
// Assume plan uses three-year moving average method
// for asset smoothing.
asset_av = double(asset_mv - (2.0/3.0) * aAsset_capital_gain[ year ]
```

C++ Code Optimization Recommendations

Code optimization for a program of this size and complexity would require its own project as the most effective optimization begins with algorithm design. However, during our review of the code we noticed several items where post-algorithm changes should result in performance gains, particularly when performing large numbers of runs. These changes will be most beneficial when applied to functions which are called multiple times; however, the results will be minor when compared to algorithm improvements.

Loop Optimization

When using counters in loops, it is faster to compare to zero than to compare two numbers. To illustrate, here is an example from the ACT_FLAT_cashflow.cpp file:

As written: `for (int a = min_age; a <= last_retire_age; a++)`

Optimized: `for (int a = last_retire_age - min_age; a--)`

By eliminating the number comparison, the loop counter portion of the code will run faster. Special care would need to be taken to ensure that subsequent calls using the "a" variable account for the change but in many cases this is an easy change to make.

Specialized Functions

Complex programs present a balancing act between optimizing for performance versus reusability. Re-using standardized functions makes maintenance easier but often results in performance sacrifices. When a particular function is called multiple times by a loop, the code will run faster if the loop is moved into the function with values passed to it.

As written:

```
 for( cycle = 1; cycle <= num_cycles; cycle++)
{
   oRes.write_pbgc_details( scenario, cycle, oPBGC );
}
```

Optimized:

```
 oRes.write_pbgc_details( scenario, num_cycles, oPBGC );
```

The new oRes.write_pbgc_details function would need to be updated to include the loop and code which calls the function to account for the migration of the loop.

Array Merge

There are numerous times throughout the code where array values are initialized one item at a time via a loop. As of C++11 this operation can be replaced by the STD::COPY which will result in performance improvements for some compilers.

As written:

```
for (int a = min_age; a <= last_retire_age; a++)
{
        jsFormConv_m[a] = ptrLiabilityData->Male_JNS_Factors[a];
        jsFormConv_f[a] = ptrLiabilityData->Female_JNS_Factors[a];
}
```

Optimized:

```
std::copy(jsFormConv_m[a], jsFormConv_m[last_retire_age], ptrLiabilityData-
>Male_JNS_Factors[a]);

        std::copy(jsFormConv_f[a], jsFormConv_f[last_retire_age], ptrLiabilityData->
Female_JNS_Factors[a]);
```

Array Pointers

Using pointers for array manipulation can cause slight performance improvements. As an example, when filling an array with values:

As written:

```
for (v = 0; v <= vMax - 1; v++ ) bol_calc_newHire_CFs[v] = false;
```

Optimized:

```
for (int* ptrInt = nArray; ptrInt<bol_calc_newHire_CFs+vMax; ptrInt++) *ptrInt= false;
```

Algorithm Implementation Review

Background

Based on the code review described above, we focused our detailed review on the key modules which appear to correspond to the entities described within the documentation.

Modules Reviewed

This section lists the main modules FTI has reviewed for the purpose of assessing whether PIMS C++ coding adequately reflects program documentation. Chapter 7 of the PIMS 2010 guide provides a description of various program components used by PIMS to simulate the pension insurance system.[3] These components are the Economy, Firm, Plan, IRS, and PBGC. An object in PIMS C++ coding represents each of these components.

An object in an object-oriented language such as C++ is an encapsulated program unit that contains data and "methods" that can be used to access and manipulate the data. For example, the Economy object contains the interest rate and stock market return data for each of the 500 macroeconomic scenarios. Inside the Economy object, there are methods that read macroeconomic scenarios from input tables. There are also methods that allow other program units to access information contained in the Economy object. For example, if a program unit needs the interest rate and stock return for a particular year in a particular scenario, it sends a request to the Economy object. A method inside the Economy object handles the request and returns the requested information to the requestor.

Because PIMS uses a sample of firms and plans to represent the entire insured plan universe, similar calculations need to be performed for each firm and for all plans sponsored by that firm. An object-oriented language allows different instances of firms and plans to be created, each processed in a similar fashion. Thus, an object-oriented language provides a natural way for PIMS to organize similar calculations. Furthermore, C++ is generally regarded as a computationally-efficient programming language.

[3] PIMS 2010 Guide, pages 112 – 123.

FTI has reviewed the main C++ modules that correspond to the objects described above (summarized below).

Component	Description	SE-PIMS Module	ME-PIMS Module
Economy	Interest rate and stock market return stochastic processes	Economy.cpp	Economy.cpp
Firm	Generate financial and employment information that affects bankruptcy probability	Firm.cpp Industry.cpp Bankrupt.cpp	Firm.cpp
Plan	Project demographics, calculate liabilities and assets, calibration of liabilities	Plan.cpp Assets.cpp Assump.cpp Benefit.cpp Partic.cpp Annuity.cpp Mortality.cpp	Plan.cpp Plan_calc.cpp Plan_calc_contr_mass_with.cpp CFM_Fore_Cashflow.cpp PPA_Fore.cpp Assump.cpp
IRS	Determines minimum and maximum contributions	IRS.cpp	IRS.cpp
PBGC	Processes bankruptcy, models PBGC's assets and liabilities, collects premiums and pays administrative expenses	Pbgc.cpp	Pbgc.cpp
Running the simulation	Run the simulation by stepping through macroeconomic scenarios, cycles and partner firms	Main.cpp Run.cpp	Main.cpp Run.cpp

From the database manager, the main input tables reviewed are the following:

SE-PIMS / ME-PIMS Tables
Economy
Firm
FirmCom
Firmmem
Industry
IRS
PBGC
Plan
PlanCom
Run
Sample
Sponsor
Stocpath (and the SAS coding)

We have also reviewed the SAS code associated with the macroeconomic scenarios.

Algorithm Implementation Review Findings

PIMS codes are generally consistent with the documentation. However, there are inconsistencies due to certain parts of the documentation being out-of-date. We also note that there are aspects of PIMS codes which have been omitted from the documentation which should be included in the future.

1. The algorithms and parameters in PIMS codes and databases are consistent with the 2014 Projections Report. However, the 2014 Projections Report does not have all the details that are available in the PIMS 2010 Guide and the Key Differences document. There are inconsistencies of PIMS code with certain parts of the PIMS 2010 Guide and the Key Differences documents because certain parts of those documents are no longer current. PBGC should provide an addendum to the PIMS 2010 Guide and the Key Differences document that includes at least the following:

    a. New legislative actions that are reflected in the PIMS coding, but not in the respective documentation (e.g., the funding interest rate stabilization provision of MAP-21 and HATFA, and the provision of MPRA). A description of the modeling methodology and assumptions should be provided. To the extent that certain sections of the documentation are rendered obsolete, they should be noted.
    b. The updates due to "Verification and Quantification of Buck's Recommended Changes, 2014" should be noted in the Key Differences document.
    c. Calculations that are performed on PIMS output (i.e., not as part of PIMS C++ code) should be noted (e.g., the adjustment to PBGC variable premiums).
    d. Parameters that are updated more frequently than the documentation should be noted (e.g., real interest rate (currently 0.64% versus 1.48% in PIMS 2010 Guide) and productivity (currently 1.65% versus 1.07% in the PIMS 2010 Guide)).
    e. Program features or parameters that are not being used should be noted (e.g., the implementation of parameter uncertainty).

2. Certain methodologies and parameters that appear in the SAS code or as comments in PIMS C++ code should be noted in the documentation:

    a. The ultimate corporate bond yield spread is 1.1% over the Treasury yield.
    b. The spread of PBGC's annuity purchase rate over Treasury yield is 30% of the corporate bond yield spread (i.e., 30% of 1.1%, or 0.33% percent).
    c. 106% of the corporate bond yield is used as a proxy for the third segment rate.
    d. Salary merit increase scale is imputed from current age/service/salary distribution.
    e. For calibration purposes, PIMS assumes base wage rates increase by 3% per year.
    f. With respect to the description of the benefits valued, PIMS models different retirement eligibilities, early retirement reduction, supplemental benefits and social security integration.

3. PBGC should investigate the following items, which appear to be inconsistent between PIMS coding and the documentation:

    a. As mentioned in the Subtask 4.2 report, PIMS modeling of PBGC's investment policy (i.e., the extent interest risk is hedged) is unclear from the documentation.
    b. As mentioned in the Subtask 4.4 report, for distressed terminations, PIMS coding suggests that the 80% funded ratio threshold is determined on a plan-by-plan basis,

while the PIMS 2010 Guide[4] suggests that the threshold is determined on an aggregate basis.

c. For the purpose of variable rate premium only, the amount of employer contributions in excess of the minimum required contribution is 21% of the funding target, per the input data. The PIMS 2010 Guide states this is 15%.[5]

Best Practices

As with any large programming task, in-the-code comments and documentation within the PIMS models are important tools for the ongoing maintenance and updating of the model. These comments have taken on an even larger importance due to the lack of formalized system documentation. Lynchval's programs have done an excellent job maintaining version history notes in the comment sections of the coding modules. However, additional descriptions around some of the variables are necessary given the lack of outside documentation.

Some functions simply have placeholders for the missing documentation. For example:

```
//=Desc: Array error handling redirecter. Calls general error handler.
//
// Called by: ???
```

Although comments are frequent throughout the source code, they are generally minimal. Comments with more robust context would be valuable, particularly when referencing unique cases such as special run settings. Microsoft's guide to programming best practices[6] recommends using "complete sentences when writing comments. Comments should clarify the code, not add ambiguity." And it suggests one "use comments to explain the intent of the code. They should not serve as inline translations of the code." Few of the comments in the PIMS source code are complete sentences, and many of the comments appear to be direct translations of the code as opposed to contextual explanations.

Overall Conclusions

Much of the documentation around the coding of PIMS is incomplete, out-of-date, or non-existent. Additionally, PBGC should consider placing additional emphasis on documenting the design process for future code development and updating the existing documentation to reflect the models as they currently exist.

[4] PIMS 2010 Guide, page 121.

[5] PIMS 2010 Guide, page 79.

[6] Coding Techniques and Programming Practices, https://msdn.microsoft.com/en-us/library/aa260844(v=vs.60).aspx