UNIQUE*SOFT™

1530 E. Dundee Road, Suite 100
Palatine, Illinois 60074
(847) 963-1777
info@uniquesoft.com

Migrating to a COTS System through Rules Extraction

Version 1.4

Contents

1 Executive Summary

CMS has issued guidelines to the state Medicaid system on what types of modernization they will support financially. If approved by CMS, they will pay 90% of the cost of legacy system modernization. CMS guidance to states on system modernization includes a suggestion to consider adopting commercial-off-the-shelf (COTS) solutions. The main challenge in using COTS systems is that each state must be compliant with state regulations and policies, requiring a substantial degree of state-specific customization to make the COTS product work correctly.

Business rule extraction is one way to significantly reduce the risk of a COTS system procurement and implementation. By better understanding the business rules implemented in the existing system, a Medicaid agency can create a more complete and accurate system specification. This also allows the option to redesign business processes and refactor business rules prior to the start of the system conversion. By understanding and documenting the business processes in the existing system, the agency can also select a COTS system vendor that most closely supports the desired processes of the new system.

2 Background

Authorized by Title XIX of the Social Security Act, Medicaid was signed into law in 1965 alongside Medicare. All states, the District of Columbia, and the U.S. territories have Medicaid programs designed to provide health coverage for low-income people. Although the federal government establishes certain parameters for all states to follow, each state administers its Medicaid program differently, resulting in variations in Medicaid coverage across the country.[1] This timing coincided with the emergence of large, cost-effective mainframe systems provided by vendors such as IBM. Over the next 50+ years, these systems went through multiple stages of evolution, largely due to changing federal and state policies and regulations on the administration of the Medicaid program.

More recently, rapidly changing IT technology options have presented additional complications. The original mainframe technology has been supplanted by newer, more cost-effective computing engines, and centralized, locally managed IT systems are being replaced by more flexible COTS systems. Serving a rapidly changing set of political and public service priorities in a shifting computing environment is a challenge being faced by all the CIOs supporting state Medicaid services. This paper addresses one common option that agencies are considering: the migration from a homegrown or highly customized mainframe system to a COTS system.

Nicole McNeal, a consultant with Public Knowledge, has done considerable research on the pain points of Medicaid Management Information System (MMIS) implementations across the country. Generally, states either write their own code or hire a contractor to develop a system to their specifications. One common issue she sees is that vendors customize the systems to meet the needs of one state and then try to transfer that system to another state. “The more customizations you make to a particular system, the less flexible it is,” she said. “So, I think the states are sold systems that they expect to be flexible and configurable, but they end up having more spaghetti code than they actually realized.” [2]

When an MMIS developed for one state is ported over to another, the vendor must modify 25 to 40 percent of the system to meet the new state’s specific regulatory requirements. According to Jim Joyce, general manager of health consulting services for Cognosante, which helps states with their MMIS implementations, “That modification becomes the project” because the vendor begins the work and finds the amount of change is larger than it had expected. During a typical three-year implementation process, new legislative changes often are

[1] https://www.medicaid.gov/about-us/program-history/index.html
[2] “Medicaid IT Systems: The Perfect Storm”, Government Technology Magazine, David Raths, April 5, 2015

introduced that add more business rules and complexity. "Many states get caught up in adding new changes, which adds more time to the project," Joyce said. "It can become a vicious circle." [3]

3 Migration Options

The Center for Medicare and Medicaid Systems (CMS) has issued guidelines to the state Medicaid systems on what types of modernization they will support financially. If approved by CMS, they will pay 90% of the cost of legacy system modernization. CMS guidance to states on system modernization strongly suggests that states consider COTS solutions as one element of their modernization strategy.[4]:

The most common options chosen include:

- Purchasing a COTS system and then customizing it
- Subcontracting the operation of the system and letting the contractor do the modernization
- Contracting for MMIS services to be provided by a SaaS vendor
- Working with one or more other state agencies to share a common platform, with customization to address the specific needs of each state

4 Challenges of Migrating to a COTS Solution

The challenges of migrating to a COTS system can be grouped into three categories.

1. Implementation of federally mandated or nationally standardized functionality. These rules are generally well captured in the COTS system and will generally require minimal adjustment.
2. Implementation of state-specific business rules as defined by state legislation and policies. These rules will always require significant custom development. A precise set of requirements is needed for the COTS vendor to correctly implement these business rules.
3. Adaptation of existing business rules, business processes, and work flows. This existing functionality is not standardized and can vary by a significant degree. The agency is faced with a difficult choice: convert to the processes already implemented in the COTS product or have the vendor change the COTS product to conform to the existing processes. The first option has major implications for compatibility and retraining, and the second is difficult and expensive. This is the most challenging category.

The problem faced by all state systems is to uncover the business logic in their legacy system that has been implemented over many decades and to re-implement that logic in a new code base. Each state can adjust certain aspects of its Medicaid program independently and often changes those priorities with every change in administration. This introduces layers of complexity into the system as policy and procedure changes are layered upon past changes. Adopting a COTS system without modifications is guaranteed not to be compliant with that specific state's policies and regulations.

One could attempt to write a complete MMIS system requirements specification from scratch. In theory, this will discard historical baggage and design a new system from the ground up based on the selected COTS framework. Ideally this can be based on a complete set of requirements drawn from a correct and complete set of documentation. Unfortunately, this level of documentation often does not exist for legacy systems. Agencies then must manually inspect their old documentation, try to correct the inevitable errors, and add new specifications based on recent legislation. This set of specifications is an approximation of the new system's requirements. Previous efforts using this approach have met with mixed results. For example, in 2016 California cancelled a multi-year, $179 million modernization project. Unclear and changing system requirements were a significant

[3] "Medicaid IT Systems: The Perfect Storm", Government Technology Magazine, David Raths, April 5, 2015
[4] MEDICAID MODERNIZATION: RESHAPING LARGE REPLACEMENT PROJECTS, HIMSS Annual Conference 2017, Jessica Kahn/Anshuman Sharma, CMS Patricia MacTaggart, ONC, February 22, 2017

contributing factor that caused the vendor to fall behind on deliveries and eventually negotiate a separation agreement.[5]

Incorrect or unclear specifications can cause a system supplier to deliver a new system that appears compliant with the specification but does not actually work correctly. Ideally, one could document exactly what the current system does as a precise, understandable specification by extracting all the business rules implemented in the existing system. These rules will describe the complete system, including a complete description of the handling of corner cases and exceptions that were not completely described in the original requirements or were added over many years of refinement and improvement. A full business rules extraction will enable subject matter experts to organize the resulting rules into categories to determine whether they should be carried forward to the new system, modified in some way, or discarded (for example, if they are obsolete).

5 Automated Business Rules Extraction

5.1 Introduction to Business Rules Extraction

A business rule is a policy or constraint that applies to a set of business decisions or processes. From a business perspective, a business rule is a precise statement that describes, constrains, or controls some aspect of a business. From an IT perspective, business rules are implemented as a set of decisions that link together activities into programs or applications.

Although the concept of automated business rules extraction has been around for over twenty years, it has only gained significant traction in the last few years. The emergence of AI-based tools has improved the ability to automate the identification of business rules in code and to deliver them in a format consistent with the needs of the client organization.

There are several categories of needs that can be addressed with business rules extraction. These include:

- Creation of system documentation.
- Creation of test cases.
- Creation of requirements for a manual rewrite of the system.
- Creation of input requirements to drive the conversion to a business rules engine.
- Creation of requirements to drive the conversion to a COTS system.

We primarily focus here on the creation of requirements to drive the conversion to a COTS system and briefly touch on test case generation as it relates COTS testing and implementation.

5.2 Business Rule Characteristics

While there is no single, universally applicable definition of a business rule, we can describe certain properties that the extracted business rules must have to be useful. The extracted business rules must be correct, complete, understandable, and usable. The next sections discuss these properties, and the process of extracting the rules as implemented in the UniqueSoft D*Code legacy modernization tools is described. The D*Code tool platform is an analysis and transformation engine that uses AI methods to extract, filter and transform business rules into usable form.[6]

[5] https://healthpayerintelligence.com/news/120-million-settlement-ends-medical-claims-processing-failure

[6] : Business Rules Extracted from Code, UniqueSoft White Paper, October 26, 2017
http://www.uniquesoft.com/pdfs/UniqueSoft-Extracting-Business-Rules-From-Code-v11.pdf

5.2.1 "Big B… Little b" Distinction

One important distinction that is useful to point out is between rules one might find in a requirements document, (called "Big B" business rules) and the rules one can extract from code (which we call "Little b" business rules). We will focus here on the latter. That is, we are interested in the decisions and actions that are the functionality of the applications. For example, consider the following top-down, high-level business rule:

> *No loans can be originated to high-risk applicants.*

While this rule states a relatively simple concept, the details are too vague for it to have a direct implementation in code. What is implemented in the code would probably better summarized as the following bottom-up rule:

> A loan can be originated to an applicant if the applicant has zero delinquent accounts and the loan type is home equity or mortgage, or two or fewer delinquent accounts and the loan type is auto. Otherwise, the loan is rejected.

The process of going from the code to this more detailed expression of the business rules will be discussed below.

5.2.2 Correctness and Completeness

Three main characteristics must hold for rules to be considered correct:

1. The code from which the rules are extracted must be complete.
2. The rules that are extracted must be equivalent to the original code.
3. The context of the individual rules must be captured.

We will assume here that the original code is correct in some sense. That is, we must capture what the code does, not what we think it should be doing. To that extent, having all the relevant code is critical. If the code is not complete, rules will be missed or misinterpreted. All the referenced files must be included, and there should be no missing definitions.

The behavior expressed by the extracted rules must be equivalent to the behavior of the original code. All the logic and computation of the original code must be expressed in the conditions or actions of the rules. If they are not, then the rules cannot be equivalent to the original code.

The context of the rules is equally important for correctness. Just as the statements in the code cannot be executed in an arbitrary order, a correct set of rules is not just a collection of all the decision statements in the code. The sequencing of and dependencies between the rules must be captured. For example, consider the following pseudo-code example:

```
IF retiree_years_of_service >= 5 THEN
   benefit_percent = 50
ELSE
   benefit_percent = 25
ENDIF
…
IF retiree_years_of_service >= 5 THEN
   benefit_percent = 75
ELSE
   benefit_percent = 50
ENDIF
```

As individual rules, there is clearly a contradiction as to what the benefit percent should be. However, if we look at the contexts of these decisions in the overall flow of the code, we may see that the second rule is only used when the retiree is at the vice-president level, as is shown below.

```
IF retiree_grade == vice_president THEN
   compute_compensation_b
ELSE
   compute_compensation_a
ENDIF
…
compute_compensation_a {
   …
   IF retiree_years_of_service >= 5 THEN
      benefit_percent = 50
   ELSE
      benefit_percent = 25
   ENDIF
   …
}
…

compute_compensation_b {
   …
   IF retiree_years_of_service >= 5 THEN
      benefit_percent = 75
   ELSE
      benefit_percent = 50
   ENDIF
   …
}
```

The various parts of this logic may be thousands of lines apart in the code and in different methods/paragraphs. To correctly understand a rule, one must know the context in which it is being executed. D*Code can express this context as explicit sequencings of rules, or as combined rules where possible. For example, the following rules table may be produced:

retiree_grade == vice_president	retiree_years_of_service >= 5	*ACTION*
True	True	benefit_percent = 75
True	False	benefit_percent = 50
False	True	benefit_percent = 50
False	False	benefit_percent = 25

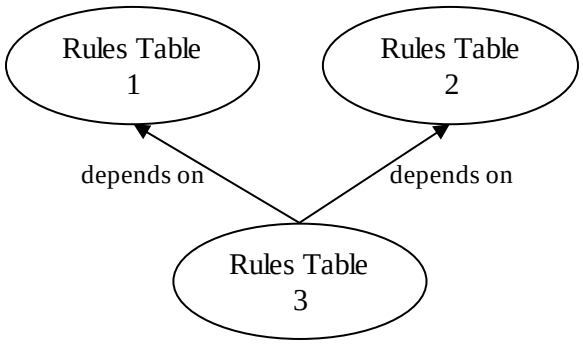
Or, in a more easily understandable English-like notation:

Retiree Grade	Retiree Status	*ACTION*
Vice President	Fully vested	Set benefit level to 75%
Vice President	Partially vested	Set benefit level to 50%
not Vice President	Fully vested	Set benefit level to 50%
not Vice President	Partially vested	Set benefit level to 25%

Similarly, the dependencies between the rules must be fully captured. For example, consider the following pseudo-code example.

```
IF retiree_years_of_service >= 5 THEN
   benefit_multiplier = benefit_multiplier + 0.2
ENDIF
IF retiree_is_vice_president THEN
   benefit_multiplier = benefit_multiplier + 0.5
ENDIF
IF retiree_payment_choice IS annuity THEN
   benefit_percent = 75 * benefit_multiplier
ELSE
   benefit_percent = 50 * benefit_multiplier
ENDIF
```

In this case, the first two decisions can be executed in either order. However, both must be executed before the third rule can be executed. Capturing the dependency of the third decision on the first two, as is shown below, is critical to correctness.



Rules Table 1

Retiree Status	*ACTION*
Fully vested	Increase benefit multiplier by 0.2

Rules Table 2

Retiree Grade	*ACTION*
Vice President	Increase benefit multiplier by 0.5

Rules Table 3

Retiree Payment Choice	*ACTION*
Annuity	Set benefit level to 75% of the benefit multiplier
not Annuity	Set benefit level to 50% of the benefit multiplier

Understanding that the first two decisions are independent, while not necessary for correctness, aids understandability, which will be discussed in the next section.

5.2.3 Understandability

While it is necessary that the extracted rules be correct, the rules must also be understandable to be useful. What is understandable will depend on the target audience, but the following three general principles apply.

1. Unwanted code should be filtered out of the rules.
2. Unnecessary complexity should be removed.
3. English descriptions of the rules should be available.

Filtering out unwanted code has two aspects. First, the business logic in the code may be separable into individual use cases. For example, the logic for creating a new insurance policy may be independent of the rules associated with the logic for processing a claim. Being able to process the business logic for each use case separately may allow smaller and more intuitive rule sets to be extracted.

Second, applications often contain code that is not part of the business logic. This code relates to how the application interacts with its users, what platform error handling it performs, what activities it keeps track of, etc. Code artifacts that are typically not business rules include:

- User input error detection / correction
- User interface design (e.g., the look and feel of the GUI)
- Program initialization and scheduling
- Database interfacing and error handling (i.e., the mechanics of database access)
- System and activity monitoring and logging
- Dead code
- Output formatting and printing
- Input data parsing
- Authentication (unless it is part of the business process itself)

In general, though, one must be careful not to remove too much code from consideration. For example, determining if a user-supplied account number is well formed is probably not business logic, but determining if the account number corresponds to an active account might be. Another consideration is the ultimate use of the extracted rules. If they will be used by a business team to review the business process or will be used as requirements for the system, then error handling, etc., should be removed. If the rules will be directly implemented in a rules engine, removing all the error handling is probably a bad idea.

As mentioned in the previous section, unwanted complexity should be removed. This complexity imposes artificial dependencies between the rules. In the example code above, we saw that the first two rules can really be executed in either order. Which one appears first in the code is based on a programmer choice and does not reflect an inherent constraint. Similarly, it may be possible to reorder the code to combine rules with the same conditions. Especially after code has been maintained for years or decades, simplifying the code though reordering and removing artificial dependencies often provides a significant benefit.

Being able to review the rules in an English-like notation removes the reliance on understanding the programming language and on the algorithms used to implement the logic. For some programming languages, such as COBOL, the use of abbreviations and cryptic identifier names in the code is common, which makes understanding the rules even more difficult. When the rules will be reviewed by a business team, making these rules more English-like may be a necessary step. For example, consider the pseudo-code example.

```
IF comb_score <= 450 THEN
   ln_cnt = 0
   ln_tot = 0
   FOR ln FROM 0 TO max_ln DO
      ln_cnt = ln_cnt + 1
      ln_tot = ln_tot + lns[ln]
   ENDFOR
ENDIF
```

Extracting this rule would result in a row in the table such as the following:

comb_score <= 450	*ACTION*
True	ln_cnt = 0 ln_tot = 0 FOR ln FROM 0 TO max_ln DO

	ln_cnt = ln_cnt + 1 ln_tot = ln_tot + lns[ln] ENDFOR

With some idea of the domain of the code and the way the programmer named the identifiers, this code is not that hard to figure out assuming a basic understanding of coding. Even so, every reviewer would have to figure it out. It would be better to have the rule in the table shown as the following:

Combined Credit Score	*ACTION*
Poor	Calculate the number and total amount of all outstanding loans

Rules such as these are also easily written in a textual form, as in the example below:

given Combined Credit Score is Poor
do Calculate the number and total amount of all outstanding loans

5.3 Advantage of Automating Business Rules Extraction

Business rules can and have been extracted from code manually. For small projects, this approach can work. However, as code size grows, the challenges increase exponentially. First, any manual process will insert human errors into the results. The larger the code base, the greater the number of errors introduced. Second, perception of what a rule is will vary between different analysts. On any large project, many analysts will be operating in parallel, introducing a source of variation. Finally, if errors are made, the process for redoing the work is equally time consuming.

In this context, the advantages of automation are obvious if a subject matter expert can quickly iterate the extraction of rules to find the correct level of business rules granularity and present them in a consistent and understandable format. This output will be uniform in quality and will produce repeatable results.

5.4 Types of Rules Extraction Output

The extracted business rules can be represented in a variety of ways, such as decision tables captured in a spreadsheet or informal text. D*Code can aid in this process by, for example, automatically migrating the extracted rules to a rules engine or by translating the extracted rules to a modern language such as Java.

Once the rules are extracted, the conditions and actions can be summarized in English. The names used in the original program are often cryptic and inconsistent, and the actions are expressed in the implementations of the algorithms in the code. While this is often the format desired when re-implementing the rules in a rules engine, it is not the easiest for a human to review. D*Code provides a mechanism for the decision names, decision results, and actions to be summarized in English, and this English form can be browsed and reviewed in the same way that the original rules can. However, the original code-based rules are also still available so that there is traceability from the English rules back to the code.

5.5 Other Benefits of Business Rules Extraction

5.5.1 Business Process Redesign

One common goal is to map the extracted rules onto the high-level business processes. The extraction of business rules from code can dramatically speed up the process of mapping the detailed implementation details instantiated in the code onto these high-level processes. This process can result in a precise description of the business processes both from a top down and bottoms up perspective. One option to consider is the redesign of business processes. With the information extracted from the code and the mapping of that detail on a high-level description

of the business processes, one has the tools to tackle a business process redesign. This can either be bundled in with the COTS system procurement or be addressed separately prior to the procurement.

5.5.2 Construction of an RFP for a New System

A precise description of the state-specific requirements will dramatically help the COTS vendor get the implementation right the first time. This will lower their risk and implementation effort, possibly resulting in a lower cost. A precise description of the desired process will enable the agency to select a COTS product that most closely conforms to the desired process, minimizing both undesirable changes to the existing processes and COTS vendor customization.

5.5.3 Implementation Risk Mitigation

Managing risk is a very important aspect of a legacy modernization project. The proper extraction of business rules from legacy systems helps reduce implementation risk in three ways:

1. By selecting a COTS vendor that can support the desired business processes and work flows, you reduce the amount of product redesign and process change required after implementation.
2. By specifying the precise business rules to be implemented in the new target system, you reduce the opportunities for error on the part of the COTS vendor.
3. By extracting and documenting the business processes in the legacy system, you can explicitly rationalize the extracted business rules and processes. Making these changes prior to the conversion to the COTS system will remove a significant source of errors.

5.5.4 Validation of the New System

Ideally, the agency already has a regression test suite for the legacy system that can be used to validate the new system. In practice, this is seldom the case. Just like legacy documentation, legacy test suites are not updated as changes are made to the systems and, over a period of years, no longer reflect the functionality of the system.

Manually creating new test cases often takes more effort than the code transformation itself. Manual test case creation also is highly error prone, and every error needs to be analyzed to understand whether the error is in the code or in the test case itself.

Automation of this labor-intensive and expensive part of a project reduces execution cost and cycle time. The effort needed to add test case generation to a project is moderate. Both test case generation and test execution can benefit from automation. Automated test case generation and testing automation reduce project risk, project duration, and cost.

6 Conclusion

Business rule extraction is one way to significantly reduce the risk of a COTS system procurement and implementation. By better understanding the business rules implemented in the existing system, a Medicaid agency can create a more complete and accurate system specification. This also allows the option to redesign business processes and refactor business rules prior to the start of the system conversion. By understanding and documenting the business processes in the existing system, the agency can also select a COTS system vendor that most closely supports the desired processes of the new system.

7 Who We Are

7.1 UniqueSoft LLC

UniqueSoft is a Chicago area based legacy modernization tools vendor that specializes in providing systems integrators with a powerful set of tools and techniques to automate the legacy modernization process. UniqueSoft's D*Code tool platform automates Code Visualization, Code Discovery, Business Rules Extraction, Translation, Transformation of old code to modern structures, and Migration to a new hardware or operating system platform. The D*Code tools platform applies artificial intelligence techniques to improve the quality of translation and business rules extraction in large complex enterprise systems.

UniqueSoft's D*Code platform is currently being used by Select Computing Inc, a systems integrator, to extract business rules from a midwestern Department of Human Services MMIS system.

7.2 UniqueSoft Contact Information

James DeBelina – Director Business Development
Dan Coombes – VP Marketing and Business Development
UniqueSoft LLC
1530 E. Dundee Road
Palatine, IL 60074

www.uniquesoft.com
info@uniquesoft.com