ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20220826124055

Preliminary communication
Received: 2022-08-26, Accepted: 2022-10-19

A Black-Box Computational Business Rules Extraction Approach through Test-Driven Development

Emad Albassam

Abstract: Business rules extraction is an important activity in situations in which a software system becomes obsolete and needs to be replaced by a newer system, since the replacing system needs to satisfy the business rules embedded in the legacy software system. In this paper, we investigate an approach in which the computational business rules of a legacy software system can be extracted given previously generated output of the system and without requiring access to the system's source code. Furthermore, extracted computational business rules are validated automatically with minimal involvement of domain experts through Test-Driven Development (TDD) such that test cases are constructed from historic output of the system. The proposed approach is applied to extract the computational business rules of a large-scale governmental payroll legacy software system. The study results demonstrate that the suggested approach extracted computational business rules van meet a substantial number of test cases. Thus, the efforts involving domain experts can be reduces to analyze such instances.

Keywords: business rules extraction; legacy software systems; software modernization; test cases

1 INTRODUCTION

Software Evolution is regarded as a term that refers to the procedure of developing software initially, then timely updating it for different reasons, for instance, to add new characteristics or to terminate obsolete functionalities, etc. [1-3]. Some legacy software systems cannot be feasibly modified or expanded further [2, 4, 5]. Thus, many organizations evolve their legacy software systems through redevelopment or replacement of these systems to sufficient support business goals. Such goals may include (1) switching from a legacy programming language to a modern language, (2) adopting a flexible software architecture for the software system (such as switching from a monolithic architecture to a microservice architecture), and (3) targeting a new operational environment (such as switching from Mainframe environment to Cloud environment) [6-8]. To accomplish such evolution activities, the replacing software system is often needed to fully or partially satisfy the business rules embedded in the legacy software system. Nevertheless, since multiple legacy software systems are either partially documented or lack any documentation of their development [9-12], the business rules embedded in these systems must be extracted first to accomplish software evolution.

Although there exist approaches in the literature for business rules extraction from legacy software systems, such approaches require (1) access to the source code of the legacy software system for analysis and (2) involvement of domain experts to manually validate the extracted business rules. In this paper, we discuss an approach in extracting computational business rules. The proposed approach does not require access to the source code of the legacy software system. Instead, we rely on collecting and analyzing historic output of the legacy software system to extract such computational rules [13-16]. Furthermore, the extracted computational business rules are validated automatically against test cases that are constructed from historic system output using TDD. The application of the proposed approach is illustrated in which the computational business rules are extracted from a large-scale governmental legacy software system.

The remainder of the paper is organized as follows: section 2 discusses works related to business rules extraction from legacy software systems. Section 3 provides the background of our work. Section 4 describes our approach for business rules extraction. Section 6 contains the discussion and threats to validity. Section 7 contains the conclusion.

2 LITERATURE REVIEW

Although business rules extraction is considered an important activity in legacy software system evolution and replacement, a systematic review by Normantas and Vasilecase [17] concluded that there is a lack in the number of publications in this area suggesting that this field requires more research efforts.

To better position our work with existing research efforts, we classify business rules extraction approaches based on whether they require (1) access to the source code of the legacy software system (i.e. white-box approaches) or not (i.e. black-box approaches) and (2) involvement of domain experts to validate extracted business rules, as discussed next.

2.1 White-Box Business Rule Extraction Approaches

The majority of existing approaches in the literature are considered white-box such that the source code of the legacy software system is assumed to be known in order to accomplish the extraction activity.

Work by Wang et al. proposed a tailored approach for business rules extraction from legacy software systems [4]. Their approach relies on (1) slicing the legacy system's code into smaller related units, (2) identifying domain variables in each slice, (3) analyzing domain variables and integrating

them into business rules with the help of domain experts, and (4) validating extracted business rules. Earls et al. [5] described a method in which the source code of legacy software systems is analyzed to determine the conditions at which error processing sections are invoked. Such conditions are then captured, translated, and recorded as business rules which are then evaluated by domain experts. In their works, Hatano et al. [8] discussed an approach for understanding business rules in software systems in which the source code of the system is analyzed to determine conditional statements that affect the computation of variables used to generate system outputs.

Work by Cosentino et al. [6] discussed a model-driven reverse engineering framework for extracting the business rules in COBOL legacy software systems in which the source code is analyzed and sliced to extract business-relevant statements. Pichler showed how business rules can be automatically extracted from legacy software code through symbolic execution [18,19]

Compared to these approaches, our approach extracts computational business rules of a legacy software system without requiring access to the system's source code (i.e. a black-box approach). Furthermore, we show how the extracted business rules are automatically validated with minimal involvement of domain experts.

2.2 Black-Box Business Rule Extraction Approaches

There have been some research efforts that consider the extraction of business rules from artifacts other than the source code. Chaparro et al. [10] investigated how structural business rules can be recovered from a legacy system's database. Paradauskas and Laurikaitis [9] also proposed an approach in which business knowledge is extracted from the relational database of legacy software systems combined with analysis of their source code. In their approach, a data reverse engineering algorithm is responsible for schema extraction and semantic analysis. Their approach is augmented with source code analysis to identify new business rules that are not explicitly stored in the database. Compared to such approaches, we consider the extraction of computational business rules from previously generated output of the legacy software system.

A research by Jin et al. [18] described an approach in which the business rules of a software system is extracted from historic test cases which had been previously constructed for the system as the software system evolves. Work by Martínez-Fernández et al. [7] proposed an automated process for extracting business rules in the banking industry from unrestricted text via SBVR (Semantic Business Vocabulary and Rules) standard and OWL language, which is considered a domain-specific approach. However, such approaches are considered domain specific.

2.3 Test-Driven Development: Background

In Test-Driven Development (TDD) [20-23], the specifications of a new software system are first captured in test cases followed by incremental implementation of the software system. As the software system is being implemented, test cases are executed to evaluate the correctness of the currently implemented parts of the software system, where the goal is to satisfy the constructed test cases at the end of the implementation process. Therefore, test cases in TDD are used to ensure that the software implementation adheres to the intended software specifications.

In our approach, we incorporate the concept of TDD to ensure that extracted computational business rules of the legacy software system satisfy the observed output that was previously generated by the legacy software system. Specifically, we generate test cases such that the expected results of test cases are derived from historic system output. When computational business rules are extracted, these rules are executed to produce actual results which are then evaluated against the test cases, therefore minimizing the role of domain experts involved in evaluating the extracted business rules.

2.4 A Black-Box Approach for Extracting Computational Business Rules

Our black-box approach to extract the computational business rules consists of 5 phases (see figure 1). As can be seen from the figure, the approach involves (1) analyzing whether black-box extraction of computational business rules is suitable for the project, (2) gathering initial information to facilitate the extraction process (such as historic system output), (3) constructing test cases from gathered information, (4) extracting computational business rules, and (4) evaluating the extracted rules using constructed test cases.

The remainder of this section describes the details of each phase. To illustrate each phase, we applied our approach to extract the computational business rules of a real-world legacy software system used in a large-scale governmental agency. The scope of the project as per stakeholder request is the *payroll* component of their legacy software system which is an information system responsible for generating the payrolls of the agency's employees. The payroll component is developed in-house by the agency through its IT department using the COBOL programming language and incorporates CICS transactional processing and runs on an on-premises Mainframe environment. The agency's business goal for the modernization project is twofold. First, the agency aims to lower its operational costs by switching from the Mainframe environment to operating its services on the cloud. Second, the modernization project aims to ensure continuous maintenance sustainability by rebuilding this component using a modern programming language due to the lack of national practitioners with appropriate qualifications in COBOL programming and CICS technology. We consider the payroll component as large-scale since it is responsible for generating the monthly payrolls of more than 15,000 employees in this agency. Furthermore, the agency is part of the country's higher-education sector and hires employees from different cadres such as the academic cadre, administrative cadre, and health cadre. Therefore, the

computational business rules that are responsible for computing the employee payrolls differ based on their cadres.

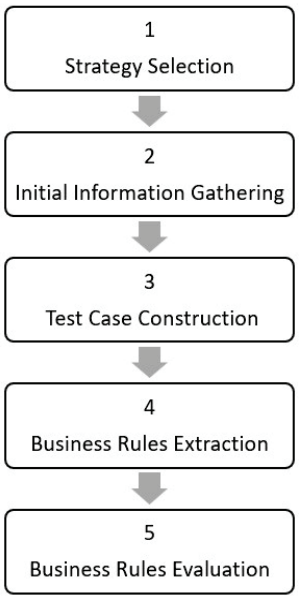

Figure 1 Phases involved in extracting computational business rules

2.5 Strategy Selection

In order to select an appropriate strategy for business rules extraction for this legacy software system, we analyzed and identified risks and challenges of this project. This identification process is accomplished by means of interviews with key stakeholders which revealed the following:

- *Lack of documentation*: as with many legacy software systems, interviews with the agency's IT personnel revealed that the payroll system lacks any documentation of its analysis, design, and implementation. Therefore, the computational business rules must be first extracted from the legacy software system in order to proceed with software replacement.
- *Improper implementation practices*: inspection of the legacy payroll system's artifacts revealed that the source code suffers from many improper implementation practices, such as overuse of GOTO statements (i.e. "spaghetti code"), unnormalized database design and lack of relations, and ambiguous naming convention. Therefore, techniques that rely on reverse engineering of source code may result in incomplete models that require substantial manual efforts to analyze, revise, and validate.
- *Retirement of original programmers*: interviews with the current programmers of the agency revealed that the original programmers of this component have either retired or left the agency. Furthermore, no current programmers have the full knowledge of the complete implementation of the payroll component such that implementing new change requests may require them to manually inspect and analyze unknown parts of the code. Therefore, relying on interviewing the current software system's programmers to obtain the computational business rules may result in requiring the programmers to inspect and analyze the entire source code, which is an error-prone process and may increase the costs and time for the project.
- *Negative previous experiences*: through project interviews, it has been discovered that the organization has gone through a previous, failed attempt to modernize its legacy systems. As a result, key stakeholders, such as HR and finance department personals, have conveyed frustration with such projects during interviews, which may negatively impact their effective participation.

Due to these factors, we consider an approach in which the computational business rules need to be (1) extracted without accessing the source code and (2) validated with minimal domain expert involvement.

2.6 Initial Information Gathering

Next, we gathered preliminary information related to the payroll legacy system that is either readily available or directly observable from the legacy software system, as follows:

- *Historic system output*: we collected observable, previously generated output from the payroll legacy software system representing the monthly employee payrolls. We observe that although the software is complex in terms of its implementation, its output is structural and well-defined. This historic system output is then analyzed to construct (1) an entity-relationship diagram (ERD) of the system output (as shown in figure 2), and (2) a data dictionary table for this ERD. Therefore, the goal of the extraction process is to extract the computational business rules responsible for generating the computed values of the output (e.g., BASIC SALARY and ELEM AMT of employee payrolls in figure 2).
- *Instances and their features*: we analyzed the database of this legacy software system (including tables, relationships, columns, and records) to identify attributes that could be used by computational business rules to compute the system's output. Specifically, we analyze the database to define the set of instances I (such that each instance $i \in I$ represents an employee of the organization) as well as the set of features for each instance f_i such as the status, gender, department, rank, and grade of employees.
- *Initial computational business rules*: we gathered the salary ladders for the various cadres in this agency. Each salary ladder is associated with a particular cadre (e.g. academic cadre, administrative cadre, and health cadre) and defines the monthly *base salary* for an employee given the employee's current rank and grade. Salary ladders are unified across this country's governmental agencies and are publicly available. Fig. 1 shows an example of a salary ladder for the Academic cadre. In the proposed approach, this information serves as the *initial computational business rules* from which additional computational business rules will be extracted.

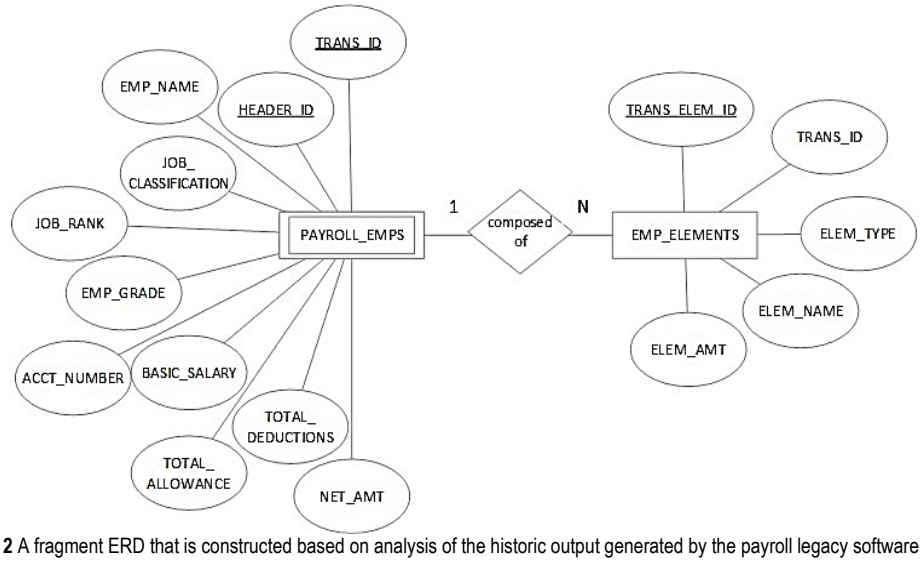

Figure 2 A fragment ERD that is constructed based on analysis of the historic output generated by the payroll legacy software system.

2.7 Test Cases Construction

Given the risks and challenges, our approach relies on automatically validating extracted business rules with minimal involvement of stakeholders and domain experts. To accomplish this, we adopted the Test-Driven Development (TDD) in which test cases are developed from the historic outputs of the legacy software system. The premise is that since the output of the legacy software system represents the results of applying the computational business rules embedded in the system, then we can use previously generated outputs of the legacy software system to automatically validate the correctness of extracted business rules. Therefore, we constructed test cases such that the *expected results* of test cases are mapped to the computed values obtained from historic system's output (e.g., BASIC _SALARY and ELEM AMT in Fig. 2). For example, if the historic output shows that an instance received the amount 6,650 as BASIC_SALARY in a particular month, then a test case is constructed for that instance and month with this amount as the expected result.

2.8 Business Rules Extraction

In this phase, we extract computational business rules by analyzing correlations between the expected results obtained from the historic system output and the actual results computed from initial business rules according to the following steps:

2.8.1 Computing the Actual Result for Each Instance

In this step and given the set of initial business rules and the set of instances $I = i_1, ..., i_n$ obtained previously during phase 2, we computed the *actual results* $A = a_{i1}, \ldots, a_{in}$ for the instances by applying the initial business rules B on each instance as follows:

$\forall i \in I$, $a_i = C(B, f_i)$, where f_i is the features defined for instance i.

As an example of this step, for an Academic employee with the rank *Teaching Assistant* and Grade 1, the actual result for his basic salary is 6,650 based on the initial business rules (see the academic salary ladder shown Tab. 1).

Table 1 Example of the obtained salary ladders. This example shows the Academic salary ladder in which the base salary of an academic employee is determined given his/her current rank and grade.

Rank	Grade					Annual Allowance
	1	2	..	14	15	
Teaching Assistant	6,650	7,065	..	12,045	12,460	415
Lecturer	8,765	9,275	..	15,395	15,905	510
Assistant Professor	12,765	13,335	..	20,175	20,745	570
Associate Professor	16,080	16,745	..	24,725	25,390	665
Professor	18,420	19,155	..	27,975	28,710	735

Note: the salary ladder has been shortened for space consideration.

2.8.2 Computing the Difference Percentage between the Actual Result and Expected Result for Each Instance

For each instance, we compute in this step the difference, in percentage, between the actual result for the instance (as computed in the previous step) and the expected result for this instance (as obtained from the historic output of the system).

Specifically, given the expected results $E = e_{i1}, ..., e_{in}$ which is obtained from historic system output for the set of instances I and (2) the computed actual results, $A = a_{i1}, \ldots, a_{in}$, we computed the difference percentage for the instances as follows:

$$\forall i \in I, d_i = \left| \frac{e_i - a_i}{e_i} \right| \times 100. \quad (1)$$

As an example of this step, if the expected result for the basic salary of an Academic employee (as per historic system output) is 7,065 and the obtained actual result for this employee from the previous step is 3,532.5, then the difference percentage is 50%.

2.9 Correlating Data

We calculate the correlations between the difference percentages computed previously and the instance features, where the goal is to find the association strength between them. To accomplish this, we use the following:

a) Pearson's *R* for correlating continuous data.
b) Correlation Ratio for correlating categorical data with continuous data.
c) Cramer's V for correlating categorical data.

The result is a heat map (see Fig. 3) that depicts the correlation between instance features such that as the correlation value between two features increases, the strength association between the two features becomes stronger indicating a significant and positive relationship. On the other hand, the value 0 indicates no correlation between features.

For example, Fig. 3 shows that there is a significant relationship between the difference percentage and the status of an employee for computing basic salaries of Academic employees, while the correlation between the difference percentage and marital status is insignificant. Strong correlations are analyzed further to extract computational business rules as explained in the next step.

2.10 Analysis of Correlated Data

At this step, strongly correlated features are analyzed to detect patterns of changes between actual results and expected results in instances using *histograms*. The majority of instances have 0% changes (i.e. earned basic salary is equal to the amount indicated by the salary ladder) while approximately 28% of instances have 50% of changes between their actual and expected results (i.e. academic employees earned a monthly basic salary equal to half the amount indicated by the salary ladder). In addition, some instances have 100% difference between their actual and expected results (i.e. no basic salary is earned by these instances).

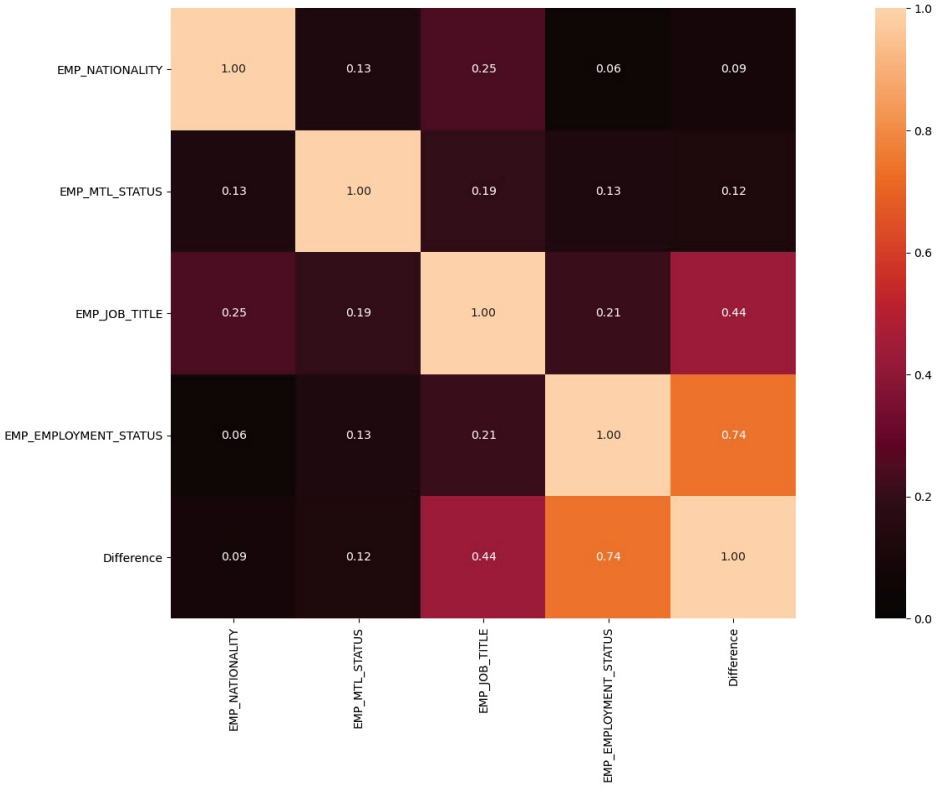

Figure 3 A fragment of the Academic cadre heat map for the Basic Salary

Such patterns are then converted into new computational business rules based on correlation results obtained from the previous step. For instance, the heat map shown in Fig. 3 indicates that the status feature of an academic employee affects his/her basic salary. By analyzing the status feature of each category we define the following computational business rules:

a) If the *status* of an academic employee is *scholar*, then the employee earns 50% of the base salary indicated by the academic salary ladder.
b) If the *status* of an academic employee is *suspended*, then the employee earns 0% as base salary.
c) If the *status* of an academic employee is *Member of the country's Consultative Council*, then the employee earns 0% as base salary.

d) Similarly, other business rules were analyzed and extracted. The following are examples of extracted business rules related to various instance features:
e) If an academic employee is assigned as a department chair, a 1500 allowance is given.
f) If an academic employee is assigned as a vice dean, a 2000 allowance is given.
g) If an academic employee is assigned as a dean, a 2500 allowance is given.
h) A monthly retirement agency deduction is applied equal to %0.09 of the academic employee basic salary.
i) If an academic employee is teaching the full load of credit hours, then the employee earns an allowance equals to the amount of the first grade in his/her current rank from salary ladder.

2.11 Evaluation of Business Rules

Initial and extracted computational business rules are evaluated at this step using the test cases constructed previously in phase 3 for every defined instance. That is, for every instance (i.e., employee), we execute the computational business rules (both initial and extracted) to compute the actual results for that instance (e.g., basic salary, allowances, and deductions) and then compare the actual results with the expected results obtained from historic system output. This evaluation relies on three metrics as follows:

a) Total number of constructed test cases.
b) Number of passed test cases: this metric indicates the number of test cases that passed such that the expected result match the actual result.
c) Number of failed test cases: this metric indicates the number of test cases that failed such that the expected result did not match the actual result.

To track these metrics more easily, we have categorized the test cases based on cadre type (e.g. academic, administrative, etc.) and payroll element type (e.g. base salary, allowances, deductions). Fig. 4 shows a fragment of the evaluation results obtained for the Academic cadre. For example in the department chairman allowance, 223 test cases were constructed of which 217 has passed and only 6 has failed. This indicates that the extracted business rule for computing this allowance has covered 97% of instances in the historic system output.

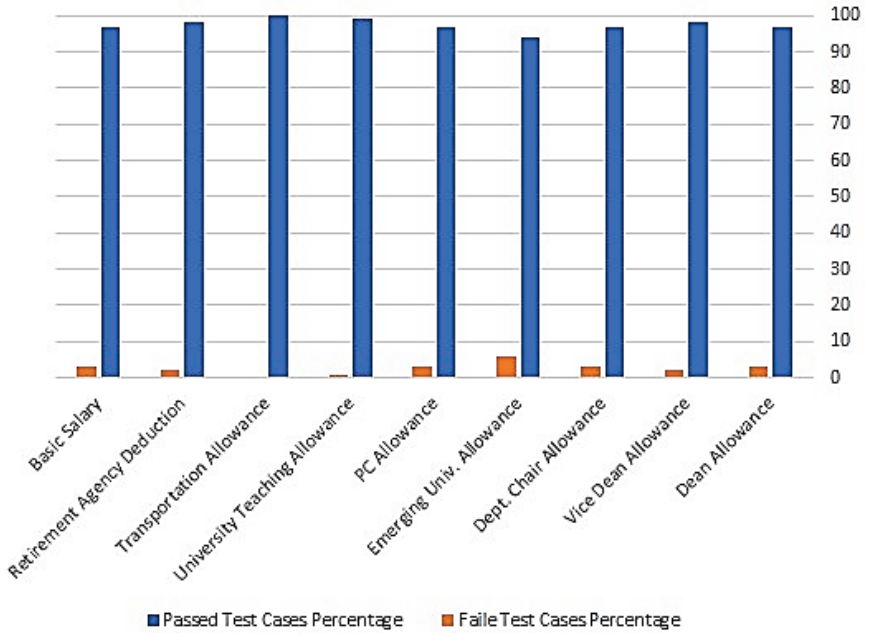

Figure 4 Fragment of evaluation results showing percentages of passed and failed test cases for the Academic cadre for various payroll components

3 RESULTS AND DISCUSSION

In spite of the fact that the extracted computational business rules through the proposed approach have covered a high percentage of the instances obtained from the output of the legacy payroll system, the proposed approach did not cover all business rules due to the existence of failed test cases. Therefore, such failed cases require further analysis by domain experts and stakeholders. Having said that, the approach is able to pinpoint these cases for further analysis. Plus, since the number of failed test cases is limited, then the effort of manual analysis is limited to the instances corresponding to these cases.

It is worth mentioning that the approach did not cover business rules associated with some cases because of factors such as (1) some instance did not follow any business rules embedded in the legacy software system (e.g. some allowances are specified manually by an HR personal such that no business rules for these components are implemented in the legacy software system) and (2) some instances follow very complex business rules in which multiple features may impact the computations.

This studied approach assumes that initial business rules must be gathered in order to extract additional rules. Moreover, the approach relies on the availability of historic system outputs. In addition, the approach requires the historic state of instance features to reproduce generation of historic system outputs. For example, historic information related to the Rank, Grade, and status of an academic employee is required to reproduce the historic basic salary of this instance.

However, it can be concluded that the approach can assist in extracting substantial computational business rules. The results of this study can contribute to decreasing the efforts involving the stakeholders and domain experts by focusing on discovering instances for which no business rules were extracted. In addition, the approach enables automatic evaluation of extracted business rules via test cases with the aid of a Test-Driven approach.

4 CONCLUSION

The present investigation endeavored to discuss a method for extracting the computational business rules of a large-scale governmental payroll legacy software system utilizing Test-Driven Development (TDD). We examine in this approach a black-box computational business rules extraction process in which no knowledge of the system's source code is assumed to be comprehended. To achieve this, we rely on computing correlations derived from the historical output of the legacy software system. Such correlations are then analyzed to extract computational business rules.

Moreover, we regard the evaluation of extracted business rules using Test-Driven development. Evaluation results show that the proposed approach extracted computational business rules that satisfied many test cases. Furthermore, the process helps pinpoint instances where no computational business rules are extracted. Thus, domain experts' efforts can be minimized to analyze such issues.

5 REFERENCES

[1] Godfrey, M. W. & German, D. M. (2008). The past, present, and future of software evolution. In *IEEE 2008 Frontiers of Software Maintenance*, 129-138. https://doi.org/10.1109/FOSM.2008.4659256

[2] Bakar, H., Razali, R., & Jambari, D. I. (2020). A guidance to legacy systems modernization. *International Journal on Advanced Science, Engineering and Information Technology, 10*(3), 1042-1050. https://doi.org/10.18517/ijaseit.10.3.10265

[3] Wolfart, D., Assunção, W. K., da Silva, I. F., Domingos, D. C., Schmeing, E., Villaca, G. L. D., & Paza, D. D. N. (2021). Modernizing legacy systems with microservices: A roadmap. In *Evaluation and Assessment in Software Engineering*, 149-159. https://doi.org/10.1145/3463274.3463334

[4] Wang, C., Zhou, Y., & Chen, J. (2008). Extracting Prime Business Rules from large legacy system. In *IEEE 2008 International Conference on Computer Science and Software Engineering, 2*, 19-23. https://doi.org/10.1109/CSSE.2008.497

[5] Earls, A. B., Embury, S. M., & Turner, N. H. (2002). A method for the manual extraction of business rules from legacy source code. BT technology journal, 20(4), 127-145. https://doi.org/10.1023/A:1021311932020

[6] Cosentino, V., Cabot, J., Albert, P., Bauquel, P., & Perronnet, J. (2013). Extracting business rules from COBOL: A model-based framework. *The 20th IEEE Working Conference on Reverse Engineering (WCRE)*, 409-416. https://doi.org/10.1109/WCRE.2013.6671316

[7] Martínez-Fernández, J. L., González, J. C., Villena, J., & Martínez, P. (2008). A preliminary approach to the automatic extraction of business rules from unrestricted text in the banking industry. In: Kapetanios, E., Sugumaran, V., Spiliopoulou, M. (eds) Natural Language and Information Systems. *NLDB 2008. Lecture Notes in Computer Science, vol 5039*. Springer, Berlin, Heidelberg. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-69858-6_29

[8] Hatano, T., Ishio, T., Okada, J., Sakata, Y., & Inoue, K. (2014, November). Extraction of conditional statements for understanding business rules. *The 6th IEEE International Workshop on Empirical Software Engineering in Practice*, 25-30. https://doi.org/10.1109/IWESEP.2014.14

[9] Paradauskas, B. & Laurikaitis, A. (2006). Business knowledge extraction from legacy information systems. *Information technology and control, 35*(3). https://doi.org/10.5755/j01.itc.35.3.11772

[10] Chaparro, O., Aponte, J., Ortega, F., & Marcus, A. (2012, October). Towards the automatic extraction of structural business rules from legacy databases. *The 19th IEEE Working Conference on Reverse Engineering*, 479-488. https://doi.org/10.1109/WCRE.2012.57

[11] Vavpotič, D., Kalibatiene, D., Vasilecas, O., & Hovelja, T. (2022). Identifying Key Characteristics of Business Rules That Affect Software Project Success. *Applied Sciences, 12*(2), 762. https://doi.org/10.3390/app12020762

[12] Gang, X. (2009). Business rule extraction from legacy system using dependence-cache slicing. *The First International Conference on Information Science and Engineering, IEEE*, 4214-4218. https://doi.org/10.1109/ICISE.2009.373

[13] Krustev, E. (2010). Business rules extraction and management-problems and solutions. *Information Systems & Grid Technologies*.

[14] Hnatkowska, B. & Ważeliński, M. (2016). Extraction of Structural Business Rules from C. In *Asian Conference on Intelligent Information and Database Systems*. Springer, Berlin, Heidelberg, 225-234. https://doi.org/10.1007/978-3-662-49381-6_22

[15] Maneva, N. & Manev, K. (2012). A Case-Driven Approach to Business Rules Extraction. *Proceedings of the 8th International Conference on Computer Science and Education*, Boston, July 6-10, 28-35.

[16] Kopp, A., Orlovskyi, D., & Orekhov, S. (2021). An Approach and Software Prototype for Translation of Natural Language Business Rules into Database Structure. *COLINS*, 1274-1291.

[17] Normantas, K. & Vasilecas, O. (2013). A systematic review of methods for business knowledge extraction from existing software systems. *Baltic Journal of Modern Computing (BJMC), 1*(1-2), 29-51. https://doi.org/10.20334/2197-M

[18] Jin, C., Gu, F., Xu, B., Cai, H., Huang, C., & Zhang, Y. (2017). A Framework of Business Rule Extraction from Historic Testing Cases. *The 14th IEEE International Conference on e-Business Engineering (ICEBE)*, 249-254. https://doi.org/10.1109/ICEBE.2017.47

[19] Pichler, J. (2013). Specification extraction by symbolic execution. *The 20th IEEE working conference on reverse engineering (WCRE)*, 462-466. https://doi.org/10.1109/WCRE.2013.6671323

[20] Fernández-Sáez, A. M., Caivano, D., Genero, M., & Chaudron, M. R. (2015). On the use of UML documentation in software maintenance: Results from a survey in industry. *The 18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 292-301. https://doi.org/10.1109/MODELS.2015.7338260

[21] Fernández-Sáez, A. M., Chaudron, M. R., & Genero, M. (2018). An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles. *Empirical Software Engineering, 23*(6), 3281-3345. https://doi.org/10.1007/s10664-018-9599-4

[22] Chiang, C. C. & Bayrak, C. (2006, October). Legacy software modernization. In *2006 IEEE international conference on systems, man and cybernetics, 2*, 1304-1309. https://doi.org/10.1109/ICSMC.2006.384895

[23] Janzen, D. & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer, 38*(9), 43-50. https://doi.org/10.1109/MC.2005.314

Author's contacts:

Emad Albassam, Assistant Professor
Department of Computer Science,
Building FCIT Building, Room 148,
King Abdulaziz University, Jeddah, Saudi Arabia
ealbassam@kau.edu.sa