Baltic J. Modern Computing, Vol. 1 (2013), No. 1-2, pp. 29–51

A Systematic Review of Methods for Business Knowledge Extraction from Existing Software Systems

Kestutis NORMANTAS, Olegas VASILECAS

Information Systems Research Laboratory, Vilnius Gediminas Technical University, Sauletekio aleja 11, Vilnius 10223, Lithuania

kestutis.normantas@isl.vgtu.lt,olegas.vasilecas@vgtu.lt

Abstract. Software maintenance and evolutions often result in large cost overruns and delayed delivery of required changes or improvements. As numerous studies have shown, adopting software to meet ever-changing business needs constitutes a major part of the software maintenance cost. The demand to facilitate software maintenance has led to the emergence of different methods for automated knowledge extraction from source code and other artefacts of existing software systems. This paper presents a systematic literature review of peer-reviewed conference and journal articles on this topic. The review has been undertaken to summarise the state-of-the-art in the research field, identify any gaps and explore possible directions for the further research. In this review, 7 digital libraries were searched and 24 papers dealing with the topic were identified and classified according to the four dimensions: extracted business knowledge kind, extraction techniques, kinds of software artefacts used as input sources, and extracted knowledge representation forms. The results of this study indicate that the research field is still immature and requires more comprehensive research. The results also show that there is a minority of methods that rely on widely adopted business knowledge classification schemes and only very few of methods employ standards for knowledge representation. It is believed that this review and classification scheme proposed in the paper would serve as a guide for both researches and practitioners in the further studies.

Keywords: business vocabulary, business rules, business processes, knowledge extraction, knowledge discovery

1 Introduction

Software maintenance and evolution are integral parts of software life cycle. According to the first Lehman's law of software evolution, once the software is in production, it undergoes continual change or becomes progressively less useful

(Lehman; 1980). The change or decay process continues until it is judged more cost effective to replace the system with a recreated version.

A comprehensive research on software maintenance and evolution by Bennett and Rajlich (2000) emphasizes the importance of software maintenance as it consumes a large part of the overall life-cycle costs, whereas the inability to change software quickly and reliably means that business opportunities are lost. The recent forecast analysis by Gartner indicates that the software maintenance consumes 38% of total cost of spending for software systems (Tom; 2013). Numerous studies have determined that the most of maintenance activities are caused by the need to adopt the system to ever-changing business requirements. A widely cited study by Lientz (1980) revealed that software enhancements to meet customer needs comprise 41.8% of the total effort spent for software maintenance. The recent study by Glass (2012) points out that enhancement is responsible for roughly 60% of software maintenance costs. One of the main reasons of high software maintenance cost is limited understanding about its initial design and its actual implementation. As the SWEBOK (Abran et al.; 2004) notes, 40% to 60% of the maintenance effort is devoted to understanding the software to be modified. This is due to the fact, that typically software maintainers are not its designers, so they must expend many resources to examine and learn about the system (Chikofsky and Cross; 1990). Moreover, the changes are rarely well documented or even not documented at all (which is often the case) and the comprehension acquired in producing changes often remains with individual developers.

As the recent study on modernizing information systems (Ulrich and Newcomb; 2010) notes, less than 30% of software source code contains business logic, while the remaining code supports infrastructure-related activities. It follows that, if the large part of software changes are due to the need to adopt its functionality to the changed business requirements, then facilitating software comprehension with automated business knowledge extraction methods may significantly reduce the cost of software maintenance and evolution. This hypothesis has been investigated by many researches during the past several decades resulting in numerous methods for business knowledge extraction from existing software systems. This paper presents a systematic literature review (SLR) in order to: summarise the state-of-the-art in this research field, identify any gaps in current research and explore possible directions for the further research, provide a framework in order to appropriately position new research activities in the field of business knowledge extraction from existing software systems.

The System Literature Review (SLR) has been undertaken by following the guidelines proposed by Kitchenham and Charters (2007). In the Section 2 we formulate research questions used to evaluate existing literature. In Section 3 we define research methods used to conduct SLR. Section 4 briefly describes the process of SLR while Section 5 presents exhaustive overview of the SLR results. In Section 6 we provide a discussion on the results. In Section 7 we consider threats to validity of this review. Finally, in Section 8 we conclude the review and discuss on further research.

2 Research Questions

In order to undertake the Systematic Literature Review we formulated the following four research questions:

RQ1: What kinds of business knowledge are being extracted by the research?
RQ2: What analysis techniques are used as a basis for knowledge extraction?
RQ3: What types of software artefacts are used as input sources for knowledge extraction?
RQ4: How the extracted knowledge is being represented?

Business knowledge in this context is considered as a certain understanding of the business domain. The commonly accepted forms for defining and for representing business knowledge are business vocabulary, business rules and business processes. A *business vocabulary* is defined by the Semantics of Business Vocabulary and Business Rules (SBVR) (OMG; 2008) as a collection of terms and facts that are used in a business for communication. A *business rules* is defined by the Business Rules Group (The Business Rules Group; 2000) as "*a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business*". Morgan (Morgan; 2002) indicates that business rules should be concerned only with *the conditions that must apply in a defined state*, and emphasises that it is also important to know who may invoke the rule, when and where the rule must be executed. A *business process* is considered as *"a defined set of business activities that represent the steps required to achieve a business objective. It includes the flow and use of information and resources*" (OMG; 2011). The business vocabulary, rules, and processes are gathered by business analysts into a set of documents that sustains throughout the development of a software system. However, the initially acquired knowledge changes or becomes obsolete after the delivery of the software system to the production.

There are many different formal and semi-formal *analysis techniques* that may be used in the field of business knowledge extraction. In this review, we are attempting to determine which of analysis techniques (such as program analysis, classification, patterns matching, model transformations, etc.) are used as a basis to formulate knowledge extraction algorithms and how they are used. The selection of certain analysis technique in the knowledge extraction process is partially determined by the *software artefacts* being analysed. This involves the production generated during the development process of a software system. Besides the source code, there might be included configuration, resource definitions, various types of software documentation, or data. However, this review is not concerned with studies that investigate information retrieval from documents, or business process mining from system logs. In our opinion, these research fields should be reviewed separately.

The business knowledge extraction process may involve several kinds of *knowledge representation form*: intermediate and output. The former is used during

various analysis activities and may include data, control, and dependency graphs, lattices, dependency tables, or meta-models that formally define the knowledge about the software system. The latter is used to present and validate the knowledge with the end-users, such as business analysts and stakeholders. Although there is no standard format for representing business rules, there are many different approaches to define them using rule patterns, production rule representation format, decision tables and trees, the Object Constraint Language (OCL), the Object-Role Modelling notation, or others. Business processes are typically represented by process models defined with a certain kind of modelling approach, such as activity flow, data flow, workflow, communication and interaction diagrams, or certainly gaining more popularity the Business Process Modelling Notation (BPMN). We believe that formulated research questions should allow conducting an overview of the-state-of-art in the current research, get a better understanding of the general principles of knowledge extraction from existing software systems, and identify existing gaps and potential opportunities for the future research.

3 Review Methods

3.1 Data sources and search strategy

In order to retrieve relevant research studies and to achieve maximal coverage, we chose to search in digital libraries and in search engines using predefined search string. Due to technical limitations, we had to limit the number of data sources. However, our initial study of selected digital libraries, and their particular sections, has shown that they contain significant number of books, journals, peer-reviewed conferences and workshops relevant to the research field. We believe that this list is fair enough to cover sufficiently large amount of relevant studies. Though, it could be extended in the future research by concerning other well-known digital libraries, search engines, or other sources. The following table summarises the data sources used to retrieve the data.

Table 1: A list of data sources used to search for relevant papers

Data source	Subjects	Website
ACM	Software and its engeering; Theory of computation; Information systems; Computing methodologies; Applied computing.	`dl.acm.org`
IEEExplore	Computing & Processing.	`ieeexplore.ieee.org`
SpringerLink	Computer Science.	`link.springer.com`
ScienceDirect	Computer Science.	`www.sciencedirect.com`

Wiley InterScience	General Computing; Computer Science; Information Science and Technology.	`onlinelibrary.wiley.com`
CiteSeerX	-	`citeseerx.ist.psu.edu`
Google Scholar	-	`scholar.google.com`

We followed the general principles proposed by Brereton (2007) to define search string. First, we defined major terms by breaking down the research questions specified in Section 4**.** Then, we searched for alternative spellings, abbreviations, synonyms, and related terms in the thesaurus and search engines. In addition, we checked for the keywords and titles in any relevant resource that could be obtained by combining the major terms. After identification of the major and alternative terms, we construct search string by combining the major terms with the alternatives using Boolean OR operator and by combining resulting predicates with each other using Boolean AND operator. The following table summarizes the search terms being used for searching relevant resources.

Table 3: A list of major terms and alternate terms used to construct the search strings

Major terms	**Alternative terms**
Business	Domain
Knowledge	Vocabulary OR Facts OR Terms OR Concepts OR Rules OR Process OR Logic OR Requirements OR Documentation
Extract	Discover OR Retrieve OR Derive OR Gather OR Reverse Engineer OR Re-Engineer OR Recover
Existing	Legacy
Software System	Information System OR Source Code OR Program

To validate whether the search string is constructed appropriately, we tried to search full-text of publications in several digital libraries and search engines. As we obtained that results are too large to be processed, consisting of hundreds of thousands of records, we limited the search in titles, abstracts, and meta-data (if such feature was provided by digital library/search engine). It should be noted, that not every digital library/search engine supports complex and long search queries. Therefore, we had to break down the search string into smaller pieces without modifying the conjunction between query predicates containing major terms and manually manipulate the search to obtain the results.

3.2 Study selection

Once the potentially relevant primary studies have been obtained, they need to be assessed for their actual relevance (Kitchenham and Charters; 2007). For this reason we formulated inclusion and exclusion criteria. Research works were included in the review list if they met at least one of the following criteria:

- Presents an approach for business knowledge extraction from existing software systems OR
- Presents a case study on applying a certain business knowledge extraction approach OR
- Presents an extension to a certain approach for business knowledge extraction.

Research works were excluded in the review list if they met at least one of the following criteria:

- Is not available in the English language OR
- Is not available as a full version paper, only an extended abstract or presentation OR
- Is a duplicate of the already included paper (we have selected the most recent version) OR
- Only discuss on the possibilities to extract business knowledge rather than presents an approach OR
- Concerns knowledge extraction from data sources other than software artefacts (i.e. only log mining or information retrieval from documents).

In order to make decision whether to include the obtained work or not, we planned several iterations as follows. During the first iteration, the title, keywords, and abstract or summary of the obtained work were reviewed according to the formulated inclusion and exclusion criteria. During the second iteration, the full-text of the obtained work was reviewed considering inclusion criteria. The obtained work was evaluated by one participant (the first author) and another participant (the second author) validated this evaluation. In case of disagreement, the discussion on the work was held between all participants to decide the inclusion of the work in the review list.

3.3 Study quality assessment

Although there is no commonly agreed definition of study "quality" (Kitchenham and Charters; 2007), it could be in some degree assessed by constructing checklists of factors that need to be evaluated for each study. For this reason we defined the following questions:

1. How the proposed approach has been evaluated?
   (a) Industrial case study;
   (b) Research case study;
   (c) An example.
2. What is the automation level of the proposed approach?
   (a) Automatic;
   (b) Semi-automatic;

(c) Manual.

3. How well the approach has been defined?
   (a) Formal enough to be reproduced;
   (b) Informal, but general steps could be reproduced;
   (c) Vaguely, impossible to reproduce.

It should be noted, that the knowledge extraction process in most cases is complex and involves multiple activities (source code parsing, transformations, analysis and refinement, information retrieval from available documentation, and etc.); therefore, it would be unreasonable to expect that all details about the implementation of the approach or its evaluation in a case study could be clearly and particularly specified, especially when there are different limitations, such as the length of paper in journals or conference proceedings. On the other hand, without assessing these criteria, it would be difficult evaluate the quality of selected works.

3.4 Data extraction

In order to accurately record the information obtained by reviewing the works, we defined data extraction form within the references management tool (JabRef). In addition to the default properties of a reference item, such as title, publication type, authors, annotation, etc. we introduced several custom properties to determine quality assessment, and to assign dimensions from the classification scheme, developed according to the research questions, formulated in Section 4. The classification scheme is presented in the Figure 1.

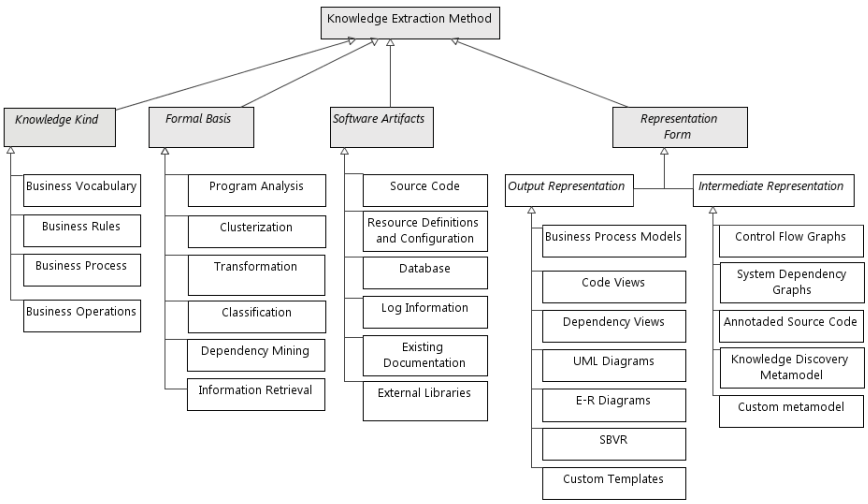

Fig. 1: Classification scheme for evaluating knowledge extraction methods

3.5 Data synthesis

To collate and summarize the results of the included studies we chose the descriptive (narrative) synthesis (Kitchenham and Charters; 2007). Extracted information about the studies was tabulated in consistent with the research questions and the quality evaluation criteria. The summarising table was structured to highlight similarities and differences between selected studies. Section 5 presents detailed discussion and the summarising table.

4 Conducting the Review

The Systematic Literature Review (SLR) was conducted in three phases (Figure 2). At the first phase, over 3500 studies were obtained by searching digital libraries/search engines using the search string or constructing such search string with the functionality provided by the web site of digital library. The total number of selected studies was reduced to 145 by considering their titles and abstracts for relevance. At the second phase, selected studies were reviewed considering inclusion and exclusion criteria and by removing duplicates. Although the number of studies was reduced to 24, it was possible to observe that certain studies were published by the same authors and concerned the same method application in different case studies or its extension. At the third phase, we reviewed in detail

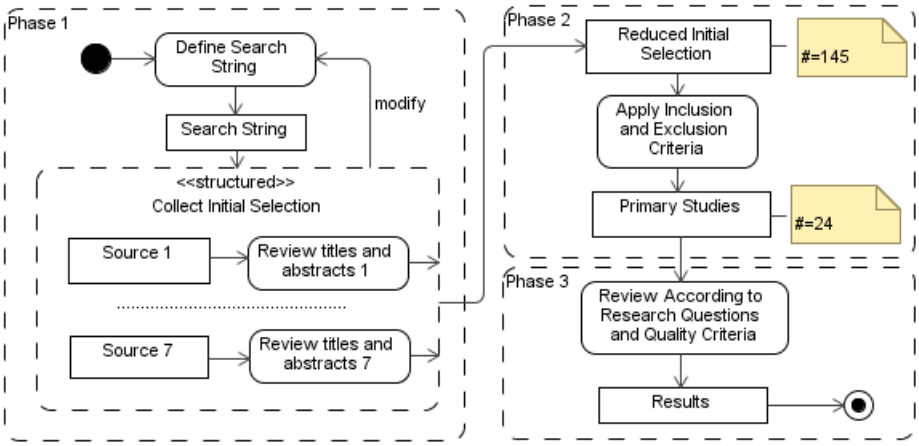

Fig. 2: Conducting the Literature Review

each of the study according to the research questions and quality criteria and summarised the results in tabular form by grouping studies on the same method (that, for example, evaluated in the other case study) and treating them as a single study.

5 Results

The distribution of the studies selected within the systematic literature review is presented in the Table 5. It could be observed that although reverse engineering is relatively old research field, the steady interest in extraction of business knowledge (such concepts as business rules and processes) from software artefacts has begun only in 2001. Moreover, the overall number of selected studies as well as the considerably lower number of the selected studies in journals than in conference proceedings and collections indicate that this research field is still young and immature. We reviewed the selected primary studies in regards with the

Table 5: Distribution of selected primary studies by year and publication type

Year	Journal	Proceedings	Collections	Total
1996	0	2	0	**2**
2001	0	1	0	**1**
2002	1	1	0	**2**
2004	0	3	0	**3**
2006	1	2	0	**3**
2007	0	2	0	**2**
2008	0	0	1	**1**
2009	0	3	0	**3**
2011	2	1	0	**3**
2012	1	1	2	**4**
Total	**5**	**16**	**3**	**24**

research questions defined in Section 2. There were identified 10 research studies that concerns with extraction of business rules, 6 that in addition to business rules extract business vocabulary (business terms and facts), 6 that concerns with extraction of business processes, and only 2 that considers extraction of both the business rules and business processes. In the following we present a detailed overview of the selected studies. At the end of this section we present summarizing table (Table 6)

5.1 Methods for Business Rules Extraction

Extracting Business Rules using Program Slicing

The vast majority of business rules extraction approaches are based on techniques for program analysis. The program analysis concerns static techniques for computing information about the behaviour of software systems (Nielson et al.; 2004). Although primarily intended for optimization of compilers, the program analysis techniques have also been successfully applied in software comprehension. That resulted to the introduction of the concept of program slicing, a set

of techniques to extract specific and rigorous knowledge from source code, to be suitably represented and visualized, and to provide basis for further analysis, classification and reconstruction (Rodrigues and Barbosa; 2010).

Program slicing has been introduced by Weiser 1981 as a technique to understand and debug programs. Since then, program slicing has grown as a field and a great number of researches have been performed resulting in different forms of program slicing, algorithms to compute slices, and applications to software (reverse and re-) engineering (Tip; 1995; De Lucia; 2001). According to its initial definition, a program slice S_C is an executable set of program statements that preserves the execution behaviour of the program P with the respect to a subset of variables V of interest at a given point of program p, i.e. a slicing criterion $C =< p, V >$. Although in other program slicing techniques, this definition is sometimes relaxed to allow a non-executable set of statements that directly or indirectly affects a subset of values of interest. This concept has been borrowed by many business rules extraction methods, because it enables to discover thorough and rigorous view of business logic implementation.

Chiang (2006a; 2006b) proposes program slicing based method for business rules extraction in order to afford possibility for business rules reuse by migrating them from the legacy code to the loosely coupled components (i.e. web services). The paper considers several kinds of business rule: primitive and complex. A primitive business rule is a function that from given set of input parameters produces one parameter, and a complex rule is defined as a set of primitive function. A function represents a program slice computed using static backward slicing (Weiser; 1981). Chiang (2006a) notices that a number of slices becomes extremely large in the case of large software system. For this reason, program code is separated into the three categories: user interface; business logic; and data access. The starting point for slicing user interface is considered as a statement reading or displaying the data. Data I/O statements such as read, write, rewrite, open, or close are considered as candidates for the starting point of slicing data access layer. During the analysis of business logic category, code parts that affect data variables are separated and presented for software developers for validation.

Unfortunately, the method is illustrated with relatively straightforward example where it is applied for a snippet of COBOL code. From this example it is hard to evaluate the feasibility or effectiveness of the method. Moreover, the presented example does not reflect the business rule at all; rather it shows a certain kind of interface to identified data structure.

Huang et al. (1996) propose business rules extraction using several types of program slicing. The method propose a number of heuristic rules for domain variables identification, slicing criteria identification, and slicing algorithm selection. Domain variables are identified considering every input and output variables of the system, arguments and return parameters of procedures. Slicing criteria involves input and output statements of the program, dispatch centres and return statements of procedures. A slicing algorithm is selected according formulated slicing criteria: for input variables and dispatch centres forward slicing is used; for output variables backward slicing is used to extract the relevant

computation logic. Extracted business rules are represented either using a code-view, a formula-view (three parts formulae – left hand side for a variable, right hand side for an expression that modifies variable, and conditions under which modifications may be executed), or input-output dependence view (bidirectional data flows between input and output parameters). The proposed approach has been implemented in the prototype tool called Business Rules Extraction Environment targeting at extraction of business rules from COBOL programs. A small example of application of the tool for a particular scenario has been presented within the paper. Though the approach gives some advices on formulating heuristics for business rules extraction from legacy code, it lacks on clear definition of what kind of business rules it attempts to discover. Rather, the approach equalizes business rules with the program slices that reflect all possible paths of certain variables computation at given point of a program. One would notice that a program slice may represent a particular use case scenario and contain numerous nested rules (i.e. control statements) within it, or the slice may even be meaningless in case of computation of non-domain variable (i.e. not a business term).

The latter issue is considered in extended version of this approach presented by Wang et al. (2004a; 2004b; 2009). In this approach, classification techniques are employed during domain variables identification step. The approach also emphasizes the importance of identification of synonymous variables occurring in different modules. Having extracted domain variables and their dependences, the next step, called data analysis, identifies business items that are actually implemented in the selected slice. According to the obtained information, a set of business rules is extracted and represented using multiple views in order to be validated with stakeholders. However, proposed views representing business rules require deep understanding of technological aspects of the software (i.e. to understand the code or graphs that represents dependencies between code blocks); therefore they are hardly understood by business analysts and stakeholders.

The program slicing as a technique for collecting of relevant information about identified business concepts is also suggested by Paradauskas and Laurikaitis (2006; 2011). The method uses both forward and backward slicing in regards with a subset of variables that are used in program input or output statements, such that forward slicing is used to collect all statements that are dependent on a given input statement (e.g. read user provided data), and backward slicing is used to gather statements that contribute to the variables of output statement (e.g. print data to the user). Sneed (1996; 2001) also suggests use program slicing for rules extraction from legacy systems. However, neither of method above does provide any further information on how to store, refine and represent the slices that correspond to business rules.

Extracting Business Rules Using Pattern Matching

Apart from program slicing, Paradauskas and Laurikaitis (2006; 2011) incorporate pattern matching for business knowledge extraction from legacy information systems. The method involves schema extraction and semantic analysis

techniques to discover conceptual schema from legacy source code and relational database. In addition to explicitly defined relations between tables in the database (i.e. reference constrains), the approach extracts implicit relations by analysing the legacy source code (written in C language) and SQL queries embedded within it. A number of query patterns are proposed to identify candidate keys that would allow establishing relations between tables. Inclusion dependency mining (that is comparison of populations of attributes participating in extracted relations) is used to classify extracted relations into the IS-A, dependent, aggregate, and other kinds of relations. Extracted schema elements are further converted from the intermediate representation defined in XML into the entity-relationship diagram notation. According to the Business Rules Approach (2000), all the extracted information is valuable because it reflects business vocabulary (business terms and facts). However, although the approach claims ability to extract business rules, it seems that it is being capable to extract only static business rules (i.e. cardinality constrains). Extraction of other kinds of business rules is neither investigated, nor discussed in the presented research. Moreover, the presented examples do not convey the feasibility of the approach in real-world scenarios, especially where the large data set are present and inclusion dependency mining is costly activity.

Chapparro et al. (2012) proposes patterns matching based method for extraction of structural business rules from legacy databases. They define a number of heuristic rules that assign certain construct of database schema (columns, table, constraint, dependencies) to a certain kind of business fact type (unary, binary, property, and category) or to the structural business rule. Then, by applying pattern matching technique, extracts corresponding constructs and stores in the relational database. The method has been evaluated in the case study of legacy system implemented using Oracle Forms technology. The results of case study showed only one third of extracted business rules being correct. The method proposed by Normantas and Vasilecas (2012; 2011) combines both patterns matching and program slicing to extract more rigorous details about implementation of business rules. As in Chaparro et al. (2012) , they define a set of patterns for extraction of different kinds of business facts from available resources of enterprise content management system, including database schema, source code, and resource definitions. To refine extracted knowledge and help to identify possible candidates to business terms and facts, the method applies text comparison algorithms on indexed documentation thus retrieving a set of relevant matches for extracted terms. For the internal representation of extracted knowledge the method uses the Knowledge Discovery Meta-model (KDM). The system dependence graph is created within a set of KDM models and further is used to slice business scenarios (i.e. Use Cases) – a behavioural logic that handles various kinds of software events. By applying a set of heuristic rules, the method identifies behavioural business rules within the extracted scenarios. However, the method produces only intermediate representation of business knowledge; transformations to more abstracted forms of knowledge representation (such as

SBVR, decision tables and trees) although are considered but not discussed in more detail in the research.

Putrycz and Kark (2007; 2008) uses patterns to identify code snippets that corresponds to production rules. The method considers extraction of production business rules in the form *<Condition><Action>* by analysing abstract syntax tree (AST) of legacy COBOL application. In order to separate code implementing business logic from setup and data transfer implementations, the approach focus on single statements that embody calculations or branching since they most often represent high level processing. The knowledge extraction process consists of the two following steps: construction of extracted knowledge base and linking the knowledge base items with existing documentation. The knowledge base is created from the production rules, identifiers, dependencies between rules, exception statements. The elements of knowledge base are linked with documents by performing key phrase analysis (in particular by employing Kea algorithm). Although the approach overcomes previously discussed approaches with the ability to refine business relevant knowledge and present it in form acceptable for a business analyst, it lacks on clarifying how the approach helped to extract business rules. The feasibility of the method has been evaluated in a case study; however, from the presented results of the case study it is hard to find out how many of business rules were identified, and how much of them were accepted as relevant after performing key-phrase extraction.

Earls et al. (2002) proposes a method for manual extraction of business rules from source code. Assuming that error handling code parts reflects violations of implemented business rules, the method proposes to identify these parts and scan backwards to collect all related code parts that lead to that exception. The method consists of the following steps. First, the source code is placed into text editor and prepared for analysis by removing irrelevant code parts: comments, working storage declarations, database connectivity management, reporting heading processing, and log-displaying code. Then, each call to procedure is replaced by the procedure body. After, error-processing sections are located and classified and code parts that invoke these sections are recorded. Finally, the conditions that led to the invocation of the error-processing code are translated into business rules and stored in the rule repository for evaluation with domain experts. The proposed method was applied in modernization project of large legacy system resulting in numerous design and business rules extracted from the source code. The design rules revealed certain decisions on the system implementation that have been taken into the consideration during migration to targeting platform. As the authors observe, not every business rule implemented in the system was extracted; however, proposed method allowed modernization engineer to discover business knowledge in faster and more accurate way. Notwithstanding the fact that error handling code sections often reflect violation of certain rules and therefore are reasonable candidates for business rules extraction, the proposed method omits consideration of other kind of business rules, such as structural rules, derivations, or other types of behavioural rules. Moreover, manual extraction of business rules from software systems that

contains multiple heterogeneous artefacts and interacts with external systems would result in extremely labour-intensive activities that would hardly produce any valuable result. Finally, simplification of source code in the proposed manner (i.e. removing irrelevant code parts) disallows the method be used in software maintenance tasks,when changes are quite common.

5.2 Business Processes Extraction

Extracting Business Processes by Slicing Use Cases

Hung and Zou (2004; 2006; 2007) present an approach for recovering workflows from multi-tiered enterprise software systems. The approach uses static tracing to identify all possible execution paths that cover user interface, application logic, and database tiers. As an entry point to start tracing, a procedure that handles certain UI event is selected. Following the flow of control and by identifying code patterns that corresponds to the elements of workflow (e.g. fork, join, task, sequence) the code is transformed into intermediate workflow representation (custom meta-model). In order to refine workflows into more abstracted versions, the approach considers a number of heuristic rules. The result is a set of high level workflows that are further transformed to models supported by IBM WebSphere Business Modeler (WBM).

To demonstrate the effectiveness of the approach, a case study has been performed on existing Enterprise Resource Planning (ERP) system, developed on Apache OFBiz platform. A prototype tool, capable of code preparation, analysis, and transformation has been applied in case study, resulting in high precise and recall factors, ranging from 75% to 100% (meaning that almost all of the extracted workflows are accurate). The results were manually evaluated in order to identify the number of misidentified and missed tasks according to documentation, comments and naming conventions. The evaluation has shown that the main categories of irrelevant tasks include utility functions, including logging and error handling.

Although the results are satisfactory, the approach has some drawbacks from our point of view. First, it considers only user input as main entry points for control flow tracing; though many researches emphasize the importance of analysis of different entry points to the system, such as external calls or internal periodical events. Second, the paper considers only a subset of intermediate representation (i.e. representing only code and UI, though in general it aims at gathering information from other artefacts). Finally, it is neither proposed, nor discussed on how to retain traceability from extracted knowledge to its actual implementation.

Extracting Business Processes Using Pattern Matching

Ricardo Perez-Castillo et al. (2011; 2012) propose a method for business processes extraction from legacy information systems. The method is based on a

framework called MARBLE that is aligned with Architecture-Driven Modernization (ADM) and which uses the Knowledge Discovery Meta-model (KDM) standard for representation and manipulation of the knowledge about the legacy information system. The MARBLE framework spans the following levels of abstraction: L0 – implementation of a legacy information system; L1 – direct representation of software system artefacts using language specific meta-models (e.g. an AST of java code); L2 – various models represented within the KDM; L3 – representation of extracted business process models using Business Process Modelling Notation (BPMN). Representations within each of the level are derived by several kinds of model transformations. The most important transformation from the representation within KDM to BPMN is based on a set of patterns.

The patterns varies from straightforward (e.g. sequencing, starting and termination) to more complex (e.g. branching, conditional sequencing, exceptions, and collaboration). The method formally defines corresponding KDM model structures and uses these definitions to create model-based (i.e. Query-View-Transform language, QVT) transformation rules. The feasibility and effectiveness of the proposed method have been evaluated in the case study of a medium-size real-world legacy system implemented in Java (28 KLOC). During the case study a number of business process models have been identified by discovering interrelated business tasks that corresponds to the predefined patterns. With a help of business experts, discovered tasks were reviewed and refined. As a result, almost half of overall tasks were recognized as relevant (223 from 425) and 10 percent (41 from 425) as unidentified relevant tasks. The effectiveness evaluation has shown reasonable transformation time (i.e. linear with respect to the size of the models) showing the scalability of the proposed method.

This method shows that knowledge discovery problem may be reduced by incorporating model-based software comprehension: the business knowledge extraction is separated from obtaining the intermediate representation (i.e. "as-is" models) of the source code. The method becomes (relatively) independent from the software implementation. Although the method is based on the KDM standard, it seems that the method does not utilize all features provided by this standard. Currently, the method considered the transformation from KDM Code model to BPMN. However, the KDM itself has different abstraction layers enabling model refinement and abstraction within the same facility. The KDM Conceptual model contains elements intended for representation of behaviour and scenario units that are very closely related to the concepts of task and business process investigated within this research. In regard with the purpose of the KDM, the problem of knowledge discovery could be further reduced to KDM Code to KDM Conceptual model transformation and KDM Conceptual to BPMN transformation, allowing reuse of extracted knowledge (i.e. business concepts and tasks) in other kinds of representation, such as business vocabulary and business rules. Unfortunately, this issue has not been considered within the research.

Kalsing et al. (2010) together with Nascimento et al. (2009; 2012) propose a method that extracts business processes by identifying business rules within the

legacy code. For this reason they use pattern matching technique. Patterns are formulated according to Weiden et al. (2004) classification scheme: mathematical calculations, function/procedure calls, data persistence, user interaction, pre-processing, post-processing, and control flow. Using code transformations, the legacy code is augmented with comments denoting corresponding rule class and with invocation of logging function enabling traceability of source code execution.

Having modified and recompiled the legacy code, the approach identifies use case scenarios performed by legacy software users and executes these scenarios to log the identified rules execution. Then, by using heuristics based IncrementalMiner algorithm, it extracts dependency graphs from log information. Extracted graphs are transformed to business process models (BPM) to facilitate further modernization of legacy system. In order to evaluate the proposed approach, the research has performed a case study on the Financial Module of legacy Enterprise Resource Planning (ERP) system written in COBOL language. As a result, the case study has extracted the structure of 7 business processes together with more than 50 business rules implemented within this module, showing that incremental process mining approach allows extraction of partial results (analysis of certain module) and thus reduces the total processing time.

However, from our point of view, this approach has several important drawbacks. First of all, this paper does not discuss on how to separate the code parts implementing business logic from the code parts intended to support infrastructure related activities. It is clear that not every mathematical calculation implements computation of business value (e.g. computing the size of user interface window), or not every control flow statement depends on evaluation of business value (e.g. condition evaluating whether certain object has been instantiated). Moreover, the paper does not consider how to identify business rules that covers nested conditional statements or even multiple procedures or modules. Finally, injection of logging instructions into the code of legacy system may not to be acceptable if it is in usage. Modifying the code of cloned legacy software, on the other hand, may not reflect the real-world scenarios because of employment specific characteristics (platform configuration, interaction with external libraries or other software systems owned by organization), or absence of working data.

Table 6: Summary of Methods for Business Knowledge Extraction

Study	Knowledge Form	Analysis Techniques	Software Artefacts	Representation Form	Automation Level	Evaluation	Clarity
Huang (1996)	Domain variables, business rules	Program slicing	Source code	Intermediate: CFG; Output:Code view, formula view, dependence view	Semi-automatic using prototype tool	Example	Informal, but general steps could be reproduced
Sneed and Erdos (1996)	Business rules	Pattern matching	Source code	Intermediate: N/A; Output:Source code snippets	Semi-automatic using prototype tool	Example	Informal, but general steps could be reproduced

Sneed (2001)	Business rules	Pattern matching, program slicing	Source code	Intermediate: CFG; Output:Source code snippets	Semi-automatic using prototype tool	Research case study	Informal, but general steps could be reproduced
Earls, Embury, and Turner (2002)	Business rules	Manual backward analysis	Source code	Intermediate: Source code; Output:Source code snippets	Manual extraction using text processors	Industrial case study	Informal, but general steps could be reproduced
Fu et al. (2002)	Business rules	Transforma-tions	Source code	Intermediate: BRL; Output:N/A	N/A	Example	Formal enough to be reproduced
Wang et al. (2004b; 2004a)	Business rules	Domain variable classifica-tion, program slicing	Source code	Intermediate: PCG; Output:Slices	Semi-automatic using prototype tool	Research case study	Informal, but general steps could be reproduced
Zou et al. (2004; 2006; 2006; 2007)	Business processes	Pattern matching, control flow analysis	Web pages, source code, con-figuration files, database	Intermediate: Custom; Output:Business process models	Semi-automatic using prototype tool	Reseach case study	Informal, but general steps could be reproduced
Chiang (2006a)	Business rules	Program slicing	Source code	Intermediate: PDG; Output:Slices, source code snippets	N/A	Example	Informal, but general steps could be reproduced
Paradauskas and Laurikaitis (2006; 2011)	Business concepts, business rules	Pattern matching, program slicing, inclusion depen-dency mining	Source code, database schema, data	Intermediate: PDG, Custom; Output:Entity-relationship diagrams	Semi-automatic using prototype tool	Example	Informal, but general steps could be reproduced
Putrycz and Anatol (2007; 2008)	Business rules	Pattern matching	Source code, docu-mentation	Intermediate: N/A; Output:Business rule templates	Semi-automatic using prototype tool	Industrial case study	Informal, but general steps could be reproduced
Cai, Yang, and Wang (2009)	Business processes	Interviews, static analysis, dynamic analysis	Source code	Intermediate: PDG; Output:Business process models	Semi-automatic	Research case study	Informal, but general steps could be reproduced
Gang (2009)	Business rules	Program slicing	Source code	Intermediate: PDG; Output:Slices	Semi-automatic using prototype tool	Example	Informal, but general steps could be reproduced
Nascimento et al. (2009; 2012)	Business rules, business processes	Patterns matching, rewriting	Source code	Intermediate: N/A; Output:Business process models	Manual	N/A	Informal, but general steps could be reproduced
Castillo et al. (2011; 2012)	Business processes	Patterns matching, program slicing, model-based transfor-mations	Source code	Intermediate: AST, KDM; Output:BPMN	Semi-automatic using MARBLE 2.0 framework (tool suite)	Industrial case study	Formal enough to be reproduced
Normantas and Vasilecas (2011; 2012)	Business vocabu-lary, business rules	Patterns matching, program slicing, model-based transfor-mations	Source code, resourse definitions, configura-tion, documen-tation	Intermediate: AST, PDG, KDM; Output:KDM	Semi-automatic using prototype tool	Research case study	Informal, but general steps could be reproduced
Chaparro et al. (2012)	Business vocabu-lary, business rules	Heuristics based clas-sification, transfor-mations	Database schema, source code	Intermediate: Custom; Output:N/A	Semi-automatic using prototype tool	Research case study	Informal, but general steps could be reproduced

Abbreviations

PDG - Program Dependence Graph
PCG - Program Call Graph
AST - Abstract Syntax Tree
KDM - Knowledge Discovery Metamodel
IR - Intermediate Represenation (problem specific)
BPM - Business Process Model
BPMN - Business Process Modelling Notation
E-R - Entity Relationship diagram

6 Discussion

In this review we asked "what kinds of business knowledge are being extracted by the current research?". The most common business knowledge kinds that are being considered in extraction methods are business rules and business processes. Several studies show interest and provide guidance in extracting the business vocabulary as well. Although there are various business vocabulary and rule categorization schemes proposed in the business rules research field, only several methods consider usage of certain scheme to extract and classify the business knowledge. Nonetheless, the result of this study shows that the interest in business knowledge extraction is important and relevant topic.

The next question was "what analysis techniques are used as a basis for knowledge extraction?". This study showed that the methods for business rules extraction have several common characteristics. They are similar in the sense that most of them concern the following general procedure for knowledge extraction: gathering initial information, extracting candidate rules, refining candidates, and transforming to the output representation form. The most common techniques for business rules extraction are program slicing, pattern matching, and transformations. These techniques are used to extract business processes as well. It is important to notice that the most of reviewed studies propose using combination of techniques in order to achieve more accurate results. It is also should be noted, that in this review we did not include the studies that concern usage of information retrieval or mining as primary methods, such as business processes mining from event logs or rules extraction from data or documentation. In our opinion, business knowledge extraction from software artefacts may be treated as a particular research field, therefore above mentioned topics require separate studies.

The third question asked "what software artefacts are used as input sources for knowledge extraction?". There are very few methods that consider knowledge extraction from software artefacts other than source code, for instance, resource (web pages, forms, reports) definitions or configurations. Though, within contemporary software systems, these artefacts may contain much valuable information, including business vocabulary and rules. This could be explained by the fact that the most of business rules extraction methods considers legacy software written COBOL. It is important to notice, that neither of reviewed study propose a method to extract business processes from such software.

The fourth question asked "how extracted knowledge is being represented?". As we already mentioned, there are two types of knowledge representation: intermediate and output. Methods that use program slicing considers intermediate representation intended for slicing, such as control flow graphs, call graphs, or program dependence graphs. Methods that uses patterns matching either perform it directly in code or propose custom intermediate representation (custom meta-models). Only several methods use standard based (i.e. Knowledge Discovery Meta-model) intermediate representation of the extracting knowledge.

We were surprised that very few methods consider widely accepted forms for business rules representation (such as templates, decision tables, etc.). Most of

the reviewed studies suggest using either code snippets, or graph slices as the output representation, though such representation form is not acceptable solution when extracted rules must be evaluated with business analysts or stakeholders. In our opinion, this issue should be investigated in the further research.

However, this issue is irrelevant for the output representation of extracted business processes. The most of reviewed studies consider representing business processes using models defined either with Business Process Modelling Notation (BPMN), or other notation supported by business process management tools.

Considering the quality of the research, the most of reviewed papers provides informal or semi-formal definitions of knowledge extractions algorithms that we believe in certain cases could be reproduced. However, as it was already mentioned, it would be unreasonable to expect that all details about the implementation of the approach or its evaluation in a case study could be clearly and particularly specified, especially when there are different limitations, such as the length of paper in journals or conference proceedings. It is important to notice, that the most of studies propose tool support for the extraction method; however, only very few presents industrial case study on a very large software systems.

7 Threats to Validity

There are three main threats to validity of a Systematic Literature Research (SLR): study selection bias, internal validity (design and execution of the study) and external validity (applicability of the effects observed in the study). As this review has been conducted by two researches, there is potential chance that this research has not obtained a complete coverage of the studies in the research field. This could be due to the selection of primary data sources (digital libraries/search engines) or due to absence of analysis of secondary resources. To ensure the selection of unbiased studies, we formulated research questions, defined the search string from these questions, defined inclusions/exclusion criteria, and conducted multi phase literature review.

The review process was designed by single researcher. This could lead to misinterpretation of the main concepts during development of the research questions, defining search string, or selecting inclusion/exclusion criteria. Therefore, the design of the review process was evaluated with other researcher and with the experts in this field in order to prevent systematic errors in design and execution of the study. In this review we selected publications that are from academic domain, in particular from peer-reviewed scientific publications. It is likely that relevant methods are applied in industry, but not reported in scientific publications. As we had no possibilities to access such resources, these methods were not included in the review.

8 Conclusions

This paper reviews studies on business knowledge extraction from existing software systems. The studies include peer-reviewed scientific papers published in journals, conference proceedings and collections, available at seven digital libraries/search engines. The review was performed following the guidelines for the Systematic Literature Review (SLR) defined by Kitchenham et al. (2007). We believe that the contributions of this review will benefit both the researchers and practitioners addressing the identified research issues.

From this review we can conclude that although the interest in business knowledge extraction from existing software systems is important and relevant, the relatively small number of the publications in scientific journals indicates that this field is not enough explored and requires comprehensive research.

First, there is an absence of rigorous and formal specifications of algorithms that would allow the method to be reproduced in order to evaluate and validate its characteristics: complexity, performance time, and accuracy of the results. Second, a minority of methods rely on standards (such as SBVR, BPMN, UML/OCL, KDM) for producing intermediate and output representation of extracted knowledge. Third, only very few of the methods were evaluated in industrial case studies on large enterprise software systems, considering various software artefacts and other available knowledge sources. These issues should be investigated in the further research in order to develop exhaustive and rigorous methods.

Acknowledgements

We would like to thank anonymous reviewers for providing sound guidance and suggestions for early versions of this paper.

References

Abran, A., Bourque, P., Dupuis, R., Moore, J. W. and Tripp, L. L. (2004). *Guide to the Software Engineering Body of Knowledge - SWEBOK*, 2004 version edn, IEEE Press, Piscataway, NJ, USA.

Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap, *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, ACM, New York, NY, USA, pp. 73–87.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M. and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain, *Journal of Systems and Software* **80**: 571 – 583.

Cai, Z., Yang, X. and Wang, X. (2009). Business process recovery for system maintenance - an empirical approach, *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pp. 399–402.

Castillo, R. P., Cruz-Lemus, J. A., de Guzman, I. G. R. and Piattini, M. (2011). Business process archeology using marble, *Information & Software Technology* **53**(10): 1023–1044.

Chaparro, O., Aponte, J., Ortega, F. and Marcus, A. (2012). Towards the automatic extraction of structural business rules from legacy databases, *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pp. 479 –488.

Chiang, C.-C. (2006a). Extracting business rules from legacy systems into reusable components, *System of Systems Engineering, 2006 IEEE/SMC International Conference on*.

Chiang, C.-C. and Bayrak, C. (2006b). Legacy software modernization, *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, Vol. 2, pp. 1304–1309.

Chikofsky, E. J. and Cross, J. H. (1990). Reverse engineering and design recovery: a taxonomy, *Software, IEEE* **7**(1): 13–17.

De Lucia, A. (2001). Program slicing: methods and applications, *Source Code Analysis and Manipulation, 2001. Proceedings. First IEEE International Workshop on*, pp. 142–149.

Earls, A. B., Embury, S. M. and Turner, N. H. (2002). A method for the manual extraction of business rules from legacy source code, *BT Technology Journal* **20**(4): 127–145.

Fu, G., Shao, J., Embury, S. M. and Gray, W. A. (2002). Representing constraint business rules extracted from legacy systems, *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, DEXA '02, Springer-Verlag, London, UK, pp. 464–473.

Gang, X. (2009). Business rule extraction from legacy system using dependence-cache slicing, *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*, ICISE '09, IEEE Computer Society, Washington, DC, USA, pp. 4214–4218.

Glass, R. L. (2012). A study about software maintenance, *Information Systems Management* **29**(4): 338–339.

Huang, H. (1996). Business rule extraction from legacy code, *Proceedings of the 20th Conference on Computer Software and Applications*, COMPSAC '96, IEEE Computer Society, Washington, DC, USA, pp. 162–168.

Hung, M. and Zou, Y. (2007). Recovering workflows from multi tiered e-commerce systems, *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, pp. 198–207.

Kalsing, A., do Nascimento, G., Iochpe, C. and Thom, L. (2010). An incremental process mining approach to extract knowledge from legacy systems, *Enterprise Distributed Object Computing Conference (EDOC), 2010 14th IEEE International*, pp. 79–88.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering, *Technical Report EBSE 2007-001*, Keele University and Durham University Joint Report.

Lehman, M. (1980). Programs, life cycles, and laws of software evolution, *Proceedings of the IEEE* **68**(9): 1060–1076.

Lientz, B. P. and Swanson, B. E. (1980). *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, Addison-Wesley.

Morgan, T. (2002). *Business Rules and Information Systems: Aligning IT with Business Goals*, Pearson Education.

Nascimento, G. S., Iochpe, C., Thom, L., Kalsing, A. C. and Moreira, A. (2009). A method for rewriting legacy systems using business process management technology, *Proc. 11th Int'l Conf. on Enterprise Information Systems (ICEIS'09), Volumne on Information Systems Analysis and Specification*, pp. 57–62.

Nascimento, G. S., Iochpe, C., Thom, L., Kalsing, A. C. and Moreira, A. (2012). Identifying business rules to legacy systems reengineering based on bpm and soa, *in* B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. Rocha, D. Taniar and B. Apduhan (eds), *Computational Science and Its Applications ICCSA 2012*, Vol. 7336 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 67–82.

Nielson, F., Nielson, H. R. and Hankin, C. (2004). *Principles of Program Analysis*, corrected edn, Springer.

Normantas, K. and Vasilecas, O. (2012). Extracting business rules from existing enterprise software system, *in* T. Skersys, R. Butleris and R. Butkiene (eds), *Information and Software Technologies*, Vol. 319 of *Communications in Computer and Information Science*, Springer Berlin Heidelberg, pp. 482–496.

OMG (2008). Semantics of business vocabulary and business rules v1.0. http://www.omg.org/spec/SBVR/1.0/.

OMG (2011). Business process modeling notation v1.0. http://www.omg.org/spec/BPMN/2.0/PDF/.

Paradauskas, B. and Laurikaitis, A. (2006). Business knowledge extraction from legacy information systems, *Information Technology And Control, Kaunas, Technologija* **35**(3): 214–221.

Paradauskas, B. and Laurikaitis, A. (2011). Extracting conceptual data specifications from legacy information systems, *Electronics and Electrical Engineering* **1**(107): 46–50.

Putrycz, E. and Kark, A. (2008). Connecting legacy code, business rules and documentation, *in* N. Bassiliades, G. Governatori and A. Paschke (eds), *Rule Representation, Interchange and Reasoning on the Web*, Vol. 5321 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 17–30.

Putrycz, E. and Kark, A. W. (2007). Recovering business rules from legacy source code for system modernization, *RuleML*, pp. 107–118.

Ricardo, P. C. e. a. (2012). A family of case studies on business process mining using marble, *Journal of Systems and Software* **85**(6): 1370–1385.

Rodrigues, N. F. and Barbosa, L. S. (2010). Slicing for architectural analysis, *Sci. Comput. Program.* **75**(10): 828–847.

Sneed, H. M. (2001). Extracting business logic from existing cobol programs as a basis for redevelopment, *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pp. 167–175.

Sneed, H. M. and Erdos, K. (1996). Extracting business rules from source code, *Proceedings of the 4th International Workshop on Program Comprehension (WPC '96)*, WPC '96, IEEE Computer Society, Washington, DC, USA, pp. 240–245.

The Business Rules Group (2000). Defining business rules - what are they really?

Tip, F. (1995). A survey of program slicing techniques, *Journal of Programming Languages* **3**: 121–189.

Tom, E. e. a. (2013). Forecast analysis: Enterprise application software, worldwide, 2011-2016, 4q12 update.

Ulrich, W. M. and Newcomb, P. (2010). *Information Systems Transformation: Architecture-Driven Modernization Case Studies*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Vasilecas, O. and Normantas, K. (2011). Deriving business rules from the models of existing information systems, *Proceedings of the 12th International Conference on Computer Systems and Technologies*, CompSysTech '11, ACM, New York, NY, USA, pp. 95–100.

Wang, X., Lai, G. and Liu, C. (2009). Recovering relationships between documentation and source code based on the characteristics of software engineering, *Electron. Notes Theor. Comput. Sci.* **243**: 121–137.

Wang, X., Sun, J., Yang, X., He, Z. and Maddineni, S. (2004a). Application of information-flow relations algorithm on extracting business rules from legacy code, *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*, Vol. 4, pp. 3055–3058.

Wang, X., Sun, J., Yang, X., He, Z. and Maddineni, S. (2004b). Business rules extraction from large legacy systems, *Proceedings of the Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'04)*, CSMR '04, IEEE Computer Society, Washington, DC, USA, pp. 249–258.

Weiden, M. e. a. (2004). Classification and representation of business rules.

Weiser, M. (1981). Program slicing, *ICSE '81: Proceedings of the 5th international conference on Software Engineering*, IEEE Press, Piscataway, NJ, USA, pp. 439–449.

Zou, Y. and Hung, M. (2006). An approach for extracting workflows from e-commerce applications, *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, IEEE Computer Society, Washington, DC, USA, pp. 127–136.

Zou, Y., Lau, T., Kontogiannis, K., Tong, T. and Mckegney, R. (2004). Model-driven business process recovery, *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pp. 224–233.

Received May 9, 2013 , revised June 7, 2013, accepted June 10, 2013