DE GRUYTER OPEN

Information Technology and Management Science doi: 10.1515/itms-2015-0005

2015 / 18

Decomposition of Enterprise Application: A Systematic Literature Review and Research Outlook

Inese Supulniece[1], Inese Polaka[2], Solvita Berzisa[3], Egils Meiers[4], Edgars Ozolins[5], Janis Grabis[6]
[1–3,6]*Riga Technical University*, [4–5]*Visma Enterprise*

***Abstract* – Enterprise applications are aimed at managing enterprise operational data and improving business efficiency. Many enterprise applications have been developed over the past three decades, therefore, known as legacy systems. Usually, they are monolith, inflexible, poorly documented and hard to maintain. The purpose of this paper is to describe best practices and limitations for enterprise application decomposition based on the results of the systematic literature review in order to introduce an approach for enterprise application decomposition. The paper focuses on decomposition of large-scale systems using clustering methods. The investigation is performed as part of the university-industry collaboration research project.**

***Keywords* – Component identification, decomposition, enterprise application, literature review, object-oriented, software clustering.**

I. Introduction

Despite the well-known disadvantages, such as being inflexible and hard to maintain, enterprise applications are still vitally important to enterprises as they support complex core business processes; they cannot simply be removed as they implement and store critical business logic. Many enterprise applications have been developed over the past three decades, therefore, known as legacy systems. The knowledge contained in these systems is of high value to the enterprises. On the other hand, proper documentation, skilled manpower and resources to evolve are scarce [1]. The need to preserve enterprise applications is motivated by multiple aspects commonly associated with the advantages of reuse: taking advantage of software that has been extensively tested in real life, reducing risk, preserving domain knowledge, and speeding up the process for reaching current business objectives. It becomes vital for the enterprises to reuse their legacy systems as application front-ends and back-ends and do it in a gradual manner [2].

Maintainability and reuse of the enterprise applications can be improved by decomposing them into modules. Modularization re-organises a software system so that the related parts are collected together [3]. The heralded advantages of a modular architecture includes [4]: handling complexity of a large system; designing and developing different parts of the same system by different people; testing a system in partial fashion; repairing defective parts of a system without interfacing with other parts; controlling defect propagation; or, reusing the existing parts in different contexts.

The literature review about legacy system evolution towards service-oriented architecture [5] reports wrapping as the most popular migration technique. Wrapping will not solve problems already present, such as problems in maintenance and upgrading. It is a known fact that software maintenance is the most expensive activity over the software life cycle [6]. In many cases, studying the internals of the legacy system is important and white-box modernisation tools are required [5]. Industrial migration approaches do not use reverse-engineering techniques to understand the legacy systems. The required knowledge is elicited from the stakeholders who own the knowledge [2]. This means that they start from semi-formal domain business models and produce domain software components. This constitutes an important shortcoming like the inability to apply these approaches when domain business models are missing. Even if these artifacts are available, most of the times, they do not express the true reality of the system due to the erosion phenomenon [7]. The reality of the system is reflected by its source code and this latter is the only artifact always available for legacy systems [8]. Architectural understanding also helps to locate a suitable area to implement changes. It has been observed that more than 50 % time effort is spent on program comprehension before an actual change is made [3].

The systematic literature review is an initial stage of the university-industry collaboration research project. The industry partner is an IT company developing and supporting a legacy enterprise resource planning system over 20 years. This enterprise application is developed in the Delphi environment, has around 4 million lines of code and around 10,000 classes. It is used by many customers from different business domains.

The main problems are: 1) complex maintenance and exponentially growing maintenance costs; 2) difficulties to analyse and calculate change impact; 3) increasing number of bugs; 4) none of the employees understands the whole system – management would like to create a set of teams, where each team would own one or more modules and would be responsible for its development. The literature review sets the stage for further research on developing a methodology for decomposition of large-scale enterprise applications by taking into account both business consulting and application development perspectives.

The rest of the paper is structured as follows: Section 2 describes the research method; Section 3 presents our findings and best practices, open research issues and agenda. Conclusion and future work is presented at the end of the paper.

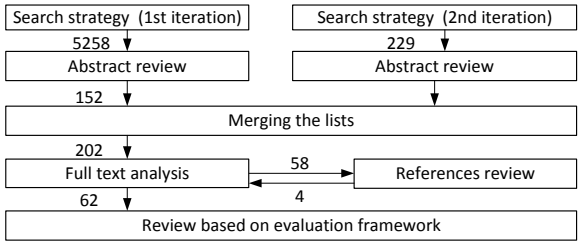

Fig. 1. The review process with a number of papers.

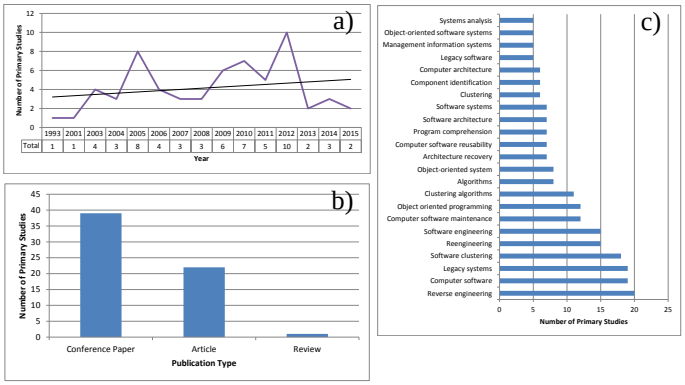

Fig. 2. Descriptive statistics: a) a number of primary studies per year; b) a number of primary studies by type; and c) a number of primary sources by keywords used.

II. Research Method

Guidelines for performing a systematic literature review in software engineering [9] are applied to our research to summarise the existing contributions, identify the gaps in the current research and avenues for future research.

The following research questions are formulated:

- What methods and techniques are used for decomposition of a large monolith application?
- What are the existing research issues and what should the future research agenda be in the area decomposition of the large monolith legacy enterprise applications?

Firstly, the systematic literature review about the decomposition of source code driven legacy systems is performed. The full systematic literature review protocol is described in [10]. Fig. 1 presents the process for conducting this literature review and the number of primary sources per activity. Descriptive statistics of the selected data sources is available in Fig. 2.

Research about applicability of decomposition methods in the area of enterprise applications is limited. In total, case studies on 187 software systems are reported. 13 of them are business systems (see Table I) and 3 of them are related to enterprise resource planning systems. The rest are software development tools and scientific applications.

This paper focuses on large enterprise application decomposition to collect best practices, limitations and research agenda. Three data sources are not sufficient to draw any conclusions. Therefore, all data sources from [10] are reviewed to collect any knowledge that could be applied for large enterprise application decomposition.

III. Best Practices and Research Agenda of Enterprise Application Decomposition

Analyses of the identified methods, according to ISO/IEC 24744 [11] framework, yield four main stages or phases: fact extraction, pre-processing, component identification and post-processing. The fact extraction phase prepares data about system's structure. Additional data gathering and processing are performed in the pre-processing phase. In the component identification phase, different clustering algorithms are mostly used for determination of system's components, but rules can also be used [12]–[14]. The component identification result improvement and evaluation activities are carried out in the post-processing phase.

Fact extraction and clustering are present in almost all methods analysed during the literature review (see Table II). However, pre-processing and post-processing are poorly described and usually fused with the clustering process.

TABLE I

Business Systems in Decomposition Case Studies

Data Source	System Description	Impl. Language	LOC	Classes
[6]	Industrial system	NA	NA	600
[14]	E-commerce project	Java	NA	main 4
[15]	Software repository of medical imaging product	C/C++	several million	NA
[16]	A proprietary CRM system	Java	2681573	5063
[16]	FinApp – a proprietary financial application	Java	153824	551
[17]	TOBEY – a proprietary industrial system that is under continuous development	NA	250000	NA
[18]	Base Station Management Centre – a management suite for commercial GSM base stations	Java	72000	NA
[19]	Business logic layer of the insurance software – manages the coverage information of insurance policies offered by an international insurance company	Java	51000	436
[20]	E-PaperML – personal finance application	Java	12655	84
[21]	Message routing system – a subsystem of an enterprise financial system	NA	9000	61
[22]	ATM system	C++	1400	16
[23]	A small banking application	Java	NA	21
[24]	MFG-PRO – a commercially available, enterprise resource planning (ERP) system, with a warehousing management module extension called AIM	Progress 4GL	NA[6200 procedure files]	NA

TABLE II

SUMMARY OF DECOMPOSITION METHODS

Primary Source	Decomposition Phases				Iterative Process	Tool
	Fact Extraction	**Pre-processing**	**Clustering/Component Identification**	**Post-processing**		
[3], [25]	Static code analysis	Similarity evaluation	Clustering	Evaluation	No	No
[6]	Dynamic code analysis	Trace compression	Clustering	No	No	Yes
[12]	No	Rules	Rules	No	No	Yes
[13]	Static code analysis	No	Rules	No	No	Yes
[14]	Static code analysis and behaviour analysis	No	Rules	Rules and meta-model	No	No
[18]	Static code analysis	Classification, weighting	Clustering	No	No	No
[20]	Dynamic SQL analysis	Formalisation	Clustering	No	No	No
[21]	Static code analysis	Business process data	Clustering	Evaluation, manual refinement	No	No
[22]	Static code analysis	Grouping, removing needless classes	Clustering	No	No	No
[26]	Dynamic code analysis	No	Clustering	Optimisation, Refinement	No	Yes
[27]	Static code analysis	Responsibility tree	Clustering	Layer identification, optimisation	Yes	Yes
[28]	Static code analysis	No	Clustering	Evaluation	No	Yes
[29]	Dynamic code analysis	Omnipresent element identification	Clustering	No	No	No
[30]	Static code analysis	No	Clustering	Manual refinement	No	No
[31]	Static code analysis	No	Clustering	Evaluation	Yes	No
[32]	Static code analysis	No	Clustering, optimisation	No	No	No
[33]	Static code analysis	Concept assignment	Clustering	Manual refinement	Yes	No
[34],[35]	Static code analysis and semantic analysis	No	Identification	Evaluation	No	No
[36]	Static code analysis	Class dependency identification	Clustering	No	No	No
[37]	Static code analysis and semantic analysis	Similarity evaluation	Clustering	No	No	Yes
[38]	Dynamic code analysis	Annotate feature, entry points	Features	No	Yes	No
[39]	Static code analysis	No	Clustering	No	No	Yes
[40]	Static code analysis	Authority calculation	Clustering	No	No	Yes
[41]	Static code analysis	No	Clustering	Interface identification	No	Yes
[42]	Static code analysis	Library class elimination	Clustering	No	No	No
[43]	Static code analysis	No	Clustering	Interface identification	No	No
[44]	Static code analysis	Design patterns	Clustering	No	No	No
[45]	Static code analysis	No	Clustering	No	No	Yes
[46]	Static code analysis	Omnipresent class identification	Clustering	No	Yes	No
[47]	Static code analysis	No	Clustering	No	No	Yes
[48]	Static code analysis	No	Clustering	Refinement	No	No
[49]	Static code analysis and dynamic analysis	No	Clustering	Refinement	No	No

The pre-processing (e.g., omnipresent class identification, data optimisation) and post-processing (e.g., cluster interface identification) phases are vital for large enterprise applications because of sizeable computations and lack of manual process control and verification.

In the literature, iterative decomposition methods are also considered. Iterative means that input parameters and algorithms are tuned before the next iteration. There are no methods, where additional datasets or types could be added before the next iteration. Iterative or layered approach could improve performance of large application decomposition. For example, [15] reports large enterprise application decomposition experience. They have introduced a levelled approach to cope with the large number of concepts.

A. Fact Extraction

Facts are data used as input for software decomposition. Fact extraction is legacy architecture and implementation language specific. Usually general information about this step is provided; however, realisation details should be analysed and applied to each particular programming language.

Most reverse engineering approaches either rely on static code analysis or dynamic analysis approaches [50]. In the area of software decomposition, most of the studies describe a static source code analysis and only two studies have applied both static and dynamic source code analyses. Dynamic source code analysis presents business process implementation. Business processes are the basis for enterprise applications; therefore, a dynamic source code analysis should be part of the enterprise application decomposition. However, full coverage of business process variations within a large enterprise application is not cost efficient; therefore, a dynamic source code analysis should be combined with a static source code analysis.

The most popular input data for object-oriented system decomposition is a system class/entity/object dependency model. It consists of classes/entities/objects and their relations. It

might also include relation weights and features. Relations might be aggregated on the class level. Usually this model is created in the form of graph, which can be directed or undirected. The nodes are classes/entities/objects, and edges are their relations. Note that nodes and also relation types differ according to a programming language; therefore, key concepts should be defined for each individual system according to the meta-model (Fig. 3). Applying a particular standard (e.g., UML class diagram) to a class/entity/object dependency model would avoid different interpretations of its content.

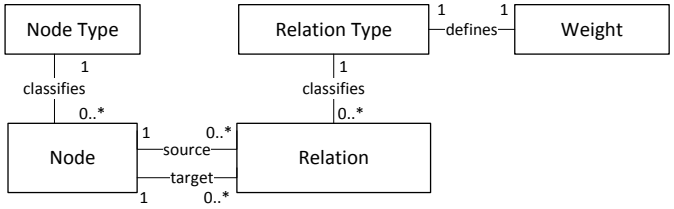

Fig. 3. The meta-model for class/entity/object dependency model.

B. Pre-processing

The data pre-processing step is motivated by reduction of search space (a number of objects that are to be merged into components) and by improving quality of the initial data. The approaches that are not framework/language dependent and are mentioned in multiple articles are the following:

- Exclusion of omnipresent classes or utilities – classes that are heavily used by other classes can be misinterpreted by decomposition algorithms and create noise as being important members of several components [29], [42], [46].
- Incorporation of non-structural information from file names, comments etc. [15], [27], [37], [41], [51], [52].
- Dividing classes into layers based on their functionality [15], [31], [42], [49], [53].
- Using a software system meta model to assign class functionality according to business processes [21], [44].

Other pre-processing approaches from the analysed sources are based on the chosen component search algorithm to make the initial data easier to process.

C. Clustering

The decomposition process is model based or clustering based. Model-based approaches (e.g., decomposition is applied starting from business process or UML models) are not applicable to large applications because of a large number of atomic objects (e.g., classes, tasks, entities), which cannot be managed by a single business expert.

Clustering algorithms dominate within component identification methods (Fig. 4), especially hierarchical clustering and divisive clustering. Although hierarchical clustering is the most popular approach and many modifications have been presented, the best parameters for clustering (weights, metrics, inter-cluster linkage, algorithm) are still unknown and have to be fine-tuned for each task at hand. Prerequisite for divisive clustering is to define the number of clusters before the clustering process. For large enterprise applications, it is impossible to define all cluster centres manually. However, a business architect or expert could specify some of the cluster centres and this information should be utilised in the clustering process.

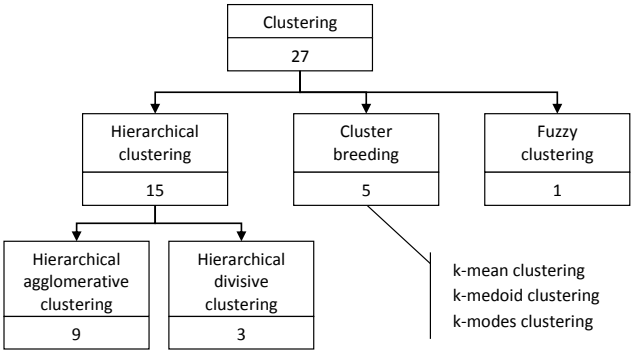

Fig. 4. Number of data sources per clustering type.

The most commonly used classification approach (hierarchical clustering) is the most robust one due to its modest requirements towards data and system. This algorithm is universal and adjustable using parameters like similarity measure or distance metric, inter-cluster similarity, steps of each iteration etc. The shortcomings of this approach also stem from its simplicity and universality – too simple heuristics (merging objects that have similar connections) – that possibly are not based on the real features of the objects, because an object that belongs to a specific component might share just one type of edge with other objects and have completely different other features. Another problem arises from determining the number of clusters (components) and their limits. If the desirable number of components is known prior to clustering, the best option would be to use k-means clustering (or some of its modifications) using this number of clusters and few more and less clusters for adjustment purposes. But, in the cases where the number of components in a legacy system is not known, it is advisable to use a hierarchical cluster analysis and make a cut in the built dendrogram at the most suitable distance (the analysed research papers do not agree on the best approach to determine the distance for the cut because it is highly dependent on the nature of a system and most often is determined experimentally).

A decomposition method for large enterprise applications should also consider business domain areas, besides cohesion and coupling, because these applications are usually delivered and configured by business domain modules.

D. Post-processing

The post-processing phase includes activities related to component refinement [21], [26], [33], [48], [49], [53] and evaluation [3], [21], [28], [31]. During refinement, the identified components are modified by moving the border classes to other components with the target to improve component quality. In this phase, it is also possible to perform layer identification [27], component interface identification [41], [43], outliner class processing [49] and cluster merging [48], [49].

Many of the considered studies adopt the step of migrating border objects among clusters because the limit of each cluster is not strict and objects close to the borders might be very similar. This is one of the problems considered in the post-processing step. It is usually addressed using some optimisation methods or evolutionary algorithms.

E. Approbation and Evaluation

Enterprise application data are usually confidential; therefore, open source systems dominate in the described case studies. Decomposition is mainly performed for systems written in Java. Large legacy enterprise systems are usually built in other programming languages.

Most of the studies have tested one system. It is difficult to compare the size of these systems because some studies report the number of classes, others – the lines of code (LOC) and some do not report any of them. Small (10K-100KLOC) and micro systems (<10KLOC) are mostly tested. Large systems (>1 million LOC) are tested in only 8 papers.

The best evaluation approach would be to identify criteria related to the initially defined problem (e.g., the number of bugs or the amount of development time) and compare those criteria before and after the decomposition. In the literature, this kind of evaluation is applied very rarely (only three papers [43], [48], [54]). Alternative is to evaluate the decomposition process (e.g., performance) or outputs (e.g., component/module quality).

The most common evaluation methods in the area of software decomposition are: authoritativeness [55], stability [56], extremity of module size distribution [18] and execution time. Some studies use only one evaluation method, others use several evaluation methods (Table III). A manually created full reference decomposition model is hardly achievable in the case of larger enterprise applications; thus, alternative approaches for authoritativeness assessment or modifications are expected.

TABLE III

EVALUATION SUMMARY

Evaluation Method	**Data Sources**
Decomposition Quality Evaluation (Authoritativeness)	[3], [15], [18]–[21], [31], [37], [42], [44], [46], [49], [57]
Decomposition Quality Evaluation (Stability)	[18], [31]
Decomposition Quality Evaluation (Extremity of Module Size Distribution)	[18], [31], [37], [46]
Decomposition Quality Evaluation (Other)	[13], [25], [28], [32], [38], [41], [45], [8], [58]
Decomposition Process Evaluation (Usually Execution Time)	[18], [23], [26], [31], [32], [38], [42], [46], [49], [57]

Enterprise application decomposition evaluation should be performed by applying several decomposition methods to the same enterprise application. Results should be discussed with business architects/experts. Performance and scalability of the decomposition process should be tested.

Evaluation criteria depend on the evaluation of the approach/method. Measurements for authoritativeness are MoJo distance [13], [42], [44], [46], [49], MoJoSim [18], [31], [37], MoJoFM [3], [25], Precision and Recall [18], [21], [44], EdgeSim, MeCl, EdgeMoJo. Stability is measured by stability criteria [18], [31]. Extremity of module size distribution is measured by NED [18], [31], [37] criteria. Decomposition process can be evaluated by seconds [23], [26], [42], [57], milliseconds [18], [31], minutes and hours.

Most of the researchers perform internal evaluation, which cannot be generalised. External evaluation is more objective. Comparative evaluation of multiple methods using the same criteria is performed only in a few papers: [8], [18], [19], [28], [31], [32], [44], [49]. The most popular benchmarks are BUNCH [47], ACDC [19] and LIMBO [52].

The statistical significance of improvements achieved is evaluated in three papers: t-test and Mann–Whitney statistical test [18]; Wilcoxon signed ranked test [56]; k-fold cross validation and Pearson and Spearman correlation coefficient [48].

IV. CONCLUSION AND FUTURE WORK

This paper reports a systematic literature review on source code driven decomposition of large object-oriented enterprise applications. We will use the obtained results in the collaboration project between industrial partners and the university to create a decomposition method for large enterprise applications. Application of the decomposition methods is expected to lead towards an application design, which is easier to maintain, and development process suitable for autonomous teams.

The main requirements for the enterprise application decomposition method are: 1) it should be scalable and computationally feasible for large applications (several million LOC and >10000 classes); 2) it should be source code driven; 3) it should consider business domain knowledge; 4) it should automatically produce the list of loosely coupled modules; 5) it should be tractable for software architects; 6) it should be a parameterised process.

Various subjective measures have been involved in this systematic literature review, e.g., the selection of the primary studies (search keywords and search strategy), data extraction process, and evaluation framework. Such subjective measures can bias the overall result of the findings. The following actions have been performed to reduce the possibility of bias:

- Possible synonyms and related terms for each keyword have been included.
- The search process has been organised in two iterations; the search strategy has been adapted based on the first search iteration results.
- Inclusion and exclusion criteria have clearly been specified.
- The selection process has been distributed among three researchers.

ACKNOWLEDGMENT

The research has been conducted within the framework of European Regional Development Fund project “Information and Communication Technologies Competence Centre” No.

KC/2.1.2.1.1/10/01/001 (Contract No. L-KC-11-0003, www.itkc.lv), activity 1.3 "The Method of Monolithic System Decomposition According to SOA Principles."

REFERENCES

[1] S. Ali and S. Abdelhak-Djamel, "Evolution approaches towards a Service oriented architecture," in *Proc. of 2012 Inte. Conf. on Multimedia Computing and Systems, ICMCS 2012*, 2012, pp. 687–692. http://dx.doi.org/10.1109/icmcs.2012.6320243

[2] M. Razavian and P. Lago, "A survey of SOA migration in industry," in *Proc. of 9th int. conf. on Service-Oriented Computing, ICSOC'11*, 2011, pp. 618–626. http://dx.doi.org/10.1007/978-3-642-25535-9_48

[3] S. Muhammad, O. Maqbool, and A.Q. Abbasi, "Evaluating relationship categories for clustering object-oriented software systems," *IET Softw.*, vol. 6, no. 3, 2012, pp. 260–274. http://dx.doi.org/10.1049/iet-sen.2011.0061

[4] N. Anquetil and J. Laval, "Legacy software restructuring: Analyzing a concrete case," in *Proc. of 15th European Conf. on Software Maintenance and Reengineering, CSMR 2011*, 2011. pp. 279–286. http://dx.doi.org/10.1109/CSMR.2011.34

[5] A.A. Almonaies, J.R. Cordy, T.R. Dean, "Legacy System Evolution towards Service-Oriented Architecture," in *Proc. of Int. Workshop on SOA Migration and Evolution, SOME*, 2010. pp. 53–62.

[6] P. Dugerdil, and J. Repond, "Automatic generation of abstract views for legacy software comprehension," in *Proc. of India Software Engineering Conf., ISEC'10*, 2010. pp. 23–32. http://dx.doi.org/10.1145/1730874.1730881

[7] C. Riva, "Reverse architecting: an industrial experience report," in *Proc. of 7th Working Conf. on Reverse Engineering*, 2000. pp. 42–50. http://dx.doi.org/10.1109/WCRE.2000.891451

[8] S. Kebir, A.-D. Seriai, S. Chardigny, A. Chaoui, "Quality-centric approach for software component identification from object-oriented code.," in *Proc. of Joint Working Conf. on Software Architecture and 6th European Conf. on Software Architecture, WICSA/ECSA 2012*, 2012. pp. 181–190. http://dx.doi.org/10.1109/WICSA-ECSA.212.26

[9] S.C.B. Kitchenham, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Technical report, 2007.

[10] I. Supulniece, S. Berzisa, I. Polaka, E. Meiers, E. Ozolins, and J. Grabis, "Source Code Driven Decomposition of Object-Oriented Legacy Systems: A Systemic Literature Review and Research Outlook," In Press, 2015.

[11] International Organization for Standardization, "ISO/IEC 24744 :2014, Software Engineering: Metamodel for Development Methodologies," 2014.

[12] K.S. Hwang, J.F. Cui, and H.S. Chae, "An automated approach to componentization of java source code," in *Proc. of the IEEE 9th Int. Conf. on Computer and Information Technology, CIT 2009*, 2009, pp. 205–210. http://dx.doi.org/10.1109/CIT.2009.19

[13] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico, "An approach for architectural layer recovery," in *Proc. of the ACM Symposium on Applied Computing*, 2010, pp. 2198–2202. http://dx.doi.org/10.1145/1774088.1774551

[14] S. Alahmari, E. Zaluska, and D. De Roure, "A Service Identification Framework for Legacy System Migration into SOA," in *Proc. of the IEEE Int. Conf. on Services Computing*, 2010 pp. 614–617.

[15] M. Glorie, A. Zaidman, A. van Deursen, and L. Hofland, "Splitting a large software repository for easing future software evolution—an industrial experience report," *J. Softw. Maint. Evol. Res. Pract.*, vol. 21, no. 2, pp. 113–141., 2009. http://dx.doi.org/10.1002/smr.401

[16] S. Sarkar, A.C. Kak, and G.M. Rama, "Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software," *IEEE Trans. Softw. Eng.*, vol. 34, pp. 700–720., 2008. http://dx.doi.org/10.1109/TSE.2008.43

[17] M. Shtern and V. Tzerpos, "Factbase and decomposition generation," in *Proc. of the European Conf.on Software Maintenance and Reengineering, CSMR*, 2011, pp. 111–120.

[18] U. Erdemir and F. Buzluca, "A learning-based module extraction method for object-oriented systems," *J. Syst. Softw.*, vol. 97. pp. 156–177., 2014. http://dx.doi.org/10.1016/j.jss.2014.07.038

[19] H.H. Kim and D.-H. Bae, "Object-oriented concept analysis for software modularisation," *IET Softw.*, vol. 2, no. 2, pp. 134–148, 2008. http://dx.doi.org/10.1049/iet-sen:20060069

[20] C. Del Grosso, M. Di Penta, and I. G.-R. de Guzman, "An approach for mining services in database-oriented applications," in *Proc. of the European Conf. on Software Maintenance and Reengineering, CSMR*, 2007, pp. 287–295. http://dx.doi.org/10.1109/csmr.2007.11

[21] Z. Cai, X. Yang, X. Wang, and Y. Wang, "A systematic approach for layered component identification," in *Proc. of the 2nd IEEE Int. Conf. on Computer Science and Information Technology*, 2009, pp. 98–103. http://dx.doi.org/10.1109/ICCSIT.2009.5234763

[22] E. Lee, B. Lee, W. Shin, and C. Wu, "A reengineering process for migrating from an object-oriented legacy system to a component-based system," in *Proc. of the IEEE Computer Society's Int. Computer Software and Applications Conf.*, 2003, pp. 336–341. http://dx.doi.org/10.1109/CMPSAC.2003.1245362

[23] C. Matos and R. Heckel, "Legacy transformations for extracting service components," SENSORIA Project, *LNCS* vol. 6582, 2011, pp. 604-621. http://dx.doi.org/10.1007/978-3-642-20401-2_29

[24] A. Le Gear, J. Buckley, B. Cleary, J.J. Collins, and K. O'Dea, "Achieving a Reuse Perspective within a Component Recovery Process: An Industrial Scale Case Study," in *Proc. of the 13th Int. Workshop on Program Comprehension, IWPC'05*, 2005, pp. 279–288. http://dx.doi.org/10.1109/WPC.2005.4

[25] S. Muhammad, O. Maqbool, and A.Q. Abbasi, "Role of relationships during clustering of object-oriented software systems," in *Proc. of the 6th Int. Conf. on Emerging Technologies (ICET)*, 2010, pp. 270–275. http://dx.doi.org/10.1109/icet.2010.5638477

[26] S. Allier, H.A. Sahraoui, and S. Sadou, "Identifying components in object-oriented programs using dynamic analysis and clustering," in *Proc. of the 2009 Conf. of the Center for Advanced Studies on Collaborative Research - CASCON '09*, 2009, pp. 136–148. http://dx.doi.org/10.1145/1723028.1723045

[27] A.B. Belle, G. El Boussaidi, and H. Mili, "Recovering software layers from object oriented systems," in *Proc. of the 9th Int. Conf. on Evaluation of Novel Approaches to Software Engineering, ENASE 2014*, 2014, pp. 78–89.

[28] J.F. Cui and H.S. Chae, "Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 601–614, Jun. 2011. http://dx.doi.org/10.1016/j.infsof.2011.01.006

[29] P. Dugerdil, "Using trace sampling techniques to identify dynamic clusters of classes," in *Proc. of the 2007 Conf. of the Center for Advanced Studies on Collaborative Research, CASCON '07*, 2007, pp. 306–314. http://dx.doi.org/10.1145/1321211.1321254

[30] G. El Boussaidi, A.B. Belle, S. Vaucher, and H. Mili, "Reconstructing architectural views from legacy systems," in *Proc. of the Working Conf. on Reverse Engineering, WCRE*, 2012, pp. 345–354. http://dx.doi.org/10.1109/wcre.2012.44

[31] U. Erdemir, U. Tekin, and F. Buzluca, "Object Oriented Software Clustering Based on Community Structure," in *Proc. of the 18th Asia-Pacific Software Engineering Conf.*, 2011, pp. 315–321. http://dx.doi.org/10.1109/apsec.2011.33

[32] I. Hussain, A. Khanum, A. Q. Abbasi, and M. Y. Javed, "A novel approach for software architecture recovery using particle swarm optimization," *Int. Arab J. Inf. Technol.*, vol. 12, no. 1, pp. 32–41, 2015.

[33] J. Jahnke, "Reverse engineering software architecture using rough clusters," in Proc. of the *Annu. Meet. NORTH Am. FUZZY Inf. Process. Soc. NAFIPS 2004*, 2004, pp. 4–9. http://dx.doi.org/10.1109/nafips.2004.1336239

[34] J. Cha and C. Kim, "MaRMI-RE: Systematic componentization process for reengineering legacy system," *Comput. Sci. ITS Appl. – ICCSA 2005, PT 3*, vol. 3482, pp. 896–905, 2005.

[35] H. Kim and Y. Chung, "Transforming a legacy system into components," *Comput. Sci. ITS Appl. – ICCSA 2006, LNCS*, vol. 3982, pp. 198–205, 2006.

[36] S.K. Mishra, D.S. Kushwaha, and A.K. Misra, "Creating reusable software component from object-oriented legacy system through reverse engineering," *J. Object Technol.*, vol. 8, no. 5, pp. 133–152, 2009. http://dx.doi.org/10.5381/jot.2009.8.5.a3

[37] J. Misra, K. M. Annervaz, V. Kaulgud, S. Sengupta, and G. Titus, "Software Clustering: Unifying Syntactic and Semantic Features," in *Proc. of the 19th Working Conf. on Reverse Engineering*, 2012, pp. 113–122. http://dx.doi.org/10.1109/wcre.2012.21

[38] A. Olszak and B. Nørregaard Jørgensen, "Remodularizing Java programs for improved locality of feature implementations in source code," *Sci. Comput. Program.*, vol. 77, no. 3, pp. 131–151, 2012. http://dx.doi.org/10.1016/j.scico.2010.10.007

[39] X. Peng, W. Zhao, Y. Wu, and Y. Xue, "Research on support tools for object-oriented software reengineering," in *Proc. of the 7th Int. Conf. on Enterprise Information Systems, ICEIS 2005*, 2005, pp. 399–402.
[40] G. Scanniello and U. Erra, "Software entities as bird flocks and fish schools," in *Proc. of the First IEEE Working Conf. on Software Visualization , VISSOFT*, 2013, pp. 1–4. http://dx.doi.org/10.1109/vissoft.2013.6650544
[41] S. Budhkar and A. Gopal, "Component identification from existing object oriented system using Hierarchical clustering," *IOSR Journal of Engineering*, vol. 2, no.5, pp. 1064–1068, 2012. http://dx.doi.org/10.9790/3021-020510641068
[42] I. Sora, G. Glodean, and M. Gligor, "Software architecture reconstruction: An approach based on combining graph clustering and partitioning," in *Proc. of the International Joint Conf. on Computational Cybernetics and Technical Informatics*, 2010, pp. 259–264. http://dx.doi.org/10.1109/icccyb.2010.5491289
[43] X. Wang, J. Sun, X. Yang, C. Huang, Z. He, and S.R. Maddineni, "Reengineering standalone C++ legacy systems into the J2EE partition distributed environment," in *Proc. of the Int. Conf. on Software Engineering*, 2006, pp. 525–533. http://dx.doi.org/10.1145/1134285.1134359
[44] L. Wang, Z. Han, J. He, H. Wang, and X. Li, "Recovering Design Patterns to Support Program Comprehension," in *Proc. of the 2Nd Int. Workshop on Evidential Assessment of Software Technologies*, 2012, pp. 49–54. http://dx.doi.org/10.1145/2372233.2372248
[45] H. Washizaki and Y. Fukazawa, "A technique for automatic component extraction from object-oriented programs by refactoring," *Sci. Comput. Program.*, vol. 56, no. 1–2, pp. 99–116, Apr. 2005. http://dx.doi.org/10.1016/j.scico.2004.11.007
[46] L. Zhang, J. Luo, H. Li, J. Sun, and H. Mei, "A biting-down approach to hierarchical decomposition of object-oriented systems based on structure analysis," *J. Softw. Maint. Evol. Res. Pract.*, vol. 22, no. 8, pp. 567–596, Dec. 2010. http://dx.doi.org/10.1002/smr.417
[47] B.S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool," *IEEE Trans. Softw. Eng.*, vol. 32, no. 3. pp.193–208., 2006. http://dx.doi.org/10.1109/TSE.2006.31
[48] R. Islam and K. Sakib, "A Package Based Clustering for enhancing software defect prediction accuracy," in *Proc. of the 17th Int. Conf. on Computer and Information Technology ICCIT*, 2014, pp. 81–86. http://dx.doi.org/10.1109/iccitechn.2014.7073117
[49] B. Andreopoulos, A. An, V. Tzerpos, and X. Wang, "Multiple layer clustering of large software systems," in *Proc. of the 1th Work. Conf. Reverse Eng, WCRE*, 2005. pp. 79–88. http://dx.doi.org/10.1109/wcre.2005.24
[50] K. Krogmann, Reconstruction of Software Component Architectures and Behaviour Models Using Static and Dynamic Analysis. KIT Scientific Publishing, 2012, p. 371.
[51] O. Maqbool and H. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Trans. Softw. Eng.*, vol. 33, no. 11, pp. 759–780, 2007. http://dx.doi.org/10.1109/TSE.2007.70732
[52] P. Andritsos and V. Tzerpos, "Information-theoretic software clustering," *IEEE Trans. Softw. Eng.*, vol. 31, no. 2, pp. 150–165, Feb. 2005. http://dx.doi.org/10.1109/TSE.2005.25
[53] G. El Boussaidi, A. B. Belle, S. Vaucher, and H. Mili, "Reconstructing Architectural Views from Legacy Systems," in *Proc. of the 19th Working Conf. on Reverse Engineering*, 2012, pp. 345–354. http://dx.doi.org/10.1109/wcre.2012.44
[54] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, "Class Level Fault Prediction using Software Clustering," in *Proc. of the 28th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 640–645, 2013. http://dx.doi.org/10.1109/ase.2013.6693126
[55] F. Beck and S. Diehl, "On the impact of software evolution on software clustering," *Empirical Software Engineering*, vol. 18. 2013, pp. 970–1004. http://dx.doi.org/10.1007/s10664-012-9225-9
[56] V. Tzerpos and R. C. Holt, "On the stability of software clustering algorithms," in *Proc. of the 8th Int. Workshop on Program Comprehension, IWPC 2000*, 2000, pp. 211–218. http://dx.doi.org/10.1109/WPC.2000.852495
[57] M. McKenna, J. Slonim, M. McAllister, and K. Lyons, "Identification of software system components using semantic models and graph slicing, " in *Proc. of the 4th Int. Conf. on Software and Data Technologies, ICSOFT 2009*, 2009, pp. 5–12.
[58] K. Mahdavi, M. Harman, and R. M. Hierons, "A Multiple Hill Climbing Approach to Software Module Clustering," in Proc. of the *IEEE Int. Conf. on Software Maintenance, ICSM*, 2003, pp. 315–324. http://dx.doi.org/10.1109/icsm.2003.1235437

Inese Supulniece holds a Doctoral Degree and is a Researcher at the Institute of Information Technology of Riga Technical University (Latvia). Her main research fields are ERP system's usability, user-adaptive systems, user modelling, business process modelling and process personalisation in business applications. Her professional activities are related to business and system analyses.
E-mail: Inese.Supulniece@rtu.lv

Inese Polaka holds a Doctoral degree and is a Lecturer at the Institute of Information Technology of Riga Technical University (Latvia). Main research interests include data mining, machine learning, classifiers, evolutionary algorithms and their applications, as well as bioinformatics and biostatistics.
E-mail: inese.polaka@rtu.lv

Solvita Berzisa holds a Doctoral degree (2012) and is a Lecturer and Researcher at the Institute of Information Technology of Riga Technical University (Latvia). She obtained her *Dr. sc. ing.* (2012), *Mg. sc. ing.* (2007) and *B. sc. ing.* (2005) degrees in Computer Science and Information Technology from Riga Technical University. Her main research fields are IT project management, project management information systems implementation and application as well as project data analytics. Also she works as an IT Project Manager at Exigen Services Latvia. She holds PMP certificate and is awarded the IPMA Outstanding Research Contribution of a Young Researcher 2013. She is a member of PMI, IIBA and Latvian National Project Management Association.
E-mail: Solvita.Berzisa@rtu.lv

Egils Meiers holds a Master degree in Management and is the Development Project Manager at VISMA Enterprise. His main field of research lies in system architecture and business process optimisation. Over the past 5 years he has led a number of research and development projects which resulted in solutions for new business opportunities.
E-mail: Egils.Meiers@visma.com

Edgars Ozolins holds a Master degree from the Institute of Industrial Electronics and Electrical Engineering of Riga Technical University (Latvia). His research interests lie in the complex and multidisciplinary systems engineering and workflows. Edgars has over 19 years of experience in commercial software development, software architecture, systems and business analysis and management. He has worked on the international projects in Germany, UK and Australia as well as collaborated with partners in Scandinavian countries, Belarus and Azerbaijan.
E-mail: Edgars.Ozolins@visma.lv

Janis Grabis holds a Doctoral degree and is a Professor at Riga Technical University (Latvia) and the Head of the Institute of Information Technology. His main research interests lie within the application of mathematical programming methods in information technology, enterprise applications and system integration. He has published more than 60 scientific papers, including a monograph on supply chain configuration. He has led a number of national projects and has participated in five projects in collaboration with the University of Michigan-Dearborn (USA) and funded mainly by industrial partners, such as SAP America and Ford Motor Company.
E-mail: Grabis@rtu.lv