Trends in Software Reverse Engineering

Dr. Rehman Arshad

Research Associate, The Hut Group

M90 3DP, Manchester, United Kingdom

rehman.khan@thehutgroup.com

Abstract

The polymorphic domain of software reverse engineering varies since 90s due to multiple reasons. Some of the primary reasons include the acceptance of new programming languages, underlying technique of reverse engineering and the desired output notation of the reverse engineering that varies with evolution of software. The purpose of this paper is to provide a trend-based taxonomy of reverse engineering that can classify the differences and similarities in the reverse engineering throughout the years.

Key Words —Reverse Engineering, Softwrae Comprehension[1]

1. Introduction

"Reverse engineering can be viewed as a process of analysing a system to identify the system's components and their interrelationships" [14]. This process is used to comprehend, or analyse a system in order to extract an architecture/notation for various purposes e.g., re-structuring of legacy systems [2, 6, 31], understanding the code traces [23], find out the feature locations [37, 51] and developing understanding of the systems with poor documentation [19].

The purpose of this paper is to analyse the pros and cons and to reason about the changes in the nature of reverse engineering throughout the years. This paper can be used as a reference document to understand the factors and reasons that change the nature of software reverse engineering since 90s.

There are few factors that are responsible for the change in nature of reverse engineering throughout the years i.e., targeted programming language, underlying technique to reverse engineer software and the output notation of reverse engineering. Most reverse engineering approaches try to tackle the programming language that is considered *legacy* for a specific domain OR better upcoming options are on the horizon for that particular domain. Therefore, most of the approaches from 90s to early 2000s were applied on ***COBOL*** and ***C*** code bases [18, 21] whereas, most of the current ones are being applied on ***Java*** [6, 44]. Secondly, the notation of retrieved output after reverse engineering started as system documentation/comprehension [12] and went through the concept-lattices/graphs and code traces [9, 13] that can help in understanding the composition of a system. Nowadays, reverse engineering is moving towards architectural retrieval that can enable reusability of resources [2, 29].

This paper only covers those approaches that deal with transformation of source code from one notation to another. The succeeding sections will discuss the framework of classification and discuss the reverse engineering approaches with respect to the timeline since 90s.

2. Framework of Classification

The proposed framework and the approaches that we cover are presented in Table. 1[2]. The stated table has following parameters of classification.

- ***Timeline:*** Timeline has been classified into five intervals. Starting from an interval of ten years[3] followed by the four intervals of five years each.
- ***Approaches:*** This column shows the total number of reverse engineering approaches that we have covered during each phase.
- ***Approach Name and Reference:*** As the name suggests, these columns will show all the appraoches along with their references.

[1]DOI reference number: 10.18293/SEKE2020-061

[2]**Table 1 Legend:**
F: Feature Model, **C**: Re-structured Code, **NEC**: Non-Explicit Components
EC: Explicit Components, **VB** : View Based **G**: Concept-Lattices/Graphs, **CT**: Rank-Based Mapping/Code Trace
CO: COBOL, **LI**: Language Independent, **J**: Java, **OO**: Object-Oriented

[3]First interval consists of a decade rather than five years due to a smaller number of approaches proposed in that specific timeframe.

Timeline	Approaches	Approach Name	Reference	Technique				Retrieved Notation						Programming Language					
				Parsing Based	Plan Based	Transformational	Translational	F	C	NEC	EC	VB		CO	C	C++	LI	J	OO
												G	CT						
90s-2000	5	RECAST	[18]	✓								✓		✓					
		Sub-System Identification	[36]	✓							✓				✓				
		Ward and Bennet's Approach	[48]	✓					✓								✓		
		Burd and Munro's Approach	[12]			✓						✓		✓					
		Design Components	[29]	✓							✓					✓			
2001-2005	9	Concern Graphs	[40]				✓					✓						✓	
		Favre et al.	[25]	✓						✓									✓
		Systematic Method Approach	[31]		✓					✓							✓		
		Dynamic Feature Traces	[23]				✓						✓					✓	
		Concept Analysis	[21]				✓						✓		✓				
		Trace Dependency Analysis	[19]		✓								✓			✓			
		Software Evolution Analysis	[27]				✓					✓							✓
		Locating Features in Source Code	[22]				✓						✓						✓
		Automatic Generation	[39]	✓									✓					✓	
2006-2010	21	Dependence Graph	[40]				✓					✓			✓				
		Javacompext	[6]	✓							✓							✓	
		Chouambe at al.	[15]	✓							✓							✓	
		Antoun et al.	[2]		✓						✓							✓	
		L2CBD	[30]								✓						✓		
		CORE	[32]		✓					✓									✓
		Bunch Tool	[35]	✓						✓							✓		
		Natural Language Parsing	[1]		✓								✓					✓	
		Source Code Retrieval	[34]		✓								✓			✓		✓	
		Combining FCA with IR	[38]				✓					✓						✓	
		STRADA	[20]										✓					✓	
		Call-Graph	[9]		✓							✓			✓	✓			
		Focused-view on Execution	[10]				✓						✓		✓	✓			
		Scenario-Driven Dynamic Analysis	[42]				✓						✓			✓			
		Featureous	[37]				✓						✓					✓	
		Static and Dynamic Analysis	[41]	✓									✓					✓	
		Heuristics Based Approach	[8]		✓								✓					✓	
		Landmark and Barriers	[47]		✓							✓						✓	
		SNIAFL	[52]				✓						✓		✓				
		Cerberus	[17]		✓								✓					✓	
		Concern Identification	[45]	✓								✓						✓	
2011-2015	13	RecoVar	[51]	✓								✓					✓		
		Semi-Automatic Approach	[46]			✓							✓					✓	
		Archimetrix	[16]	✓							✓						✓		
		Quality-centric Approach	[28]		✓						✓								✓
		Memory-constrained Environment	[49]		✓						✓								✓
		Erdemir et al.	[24]	✓						✓									✓
		Product Variants	[50]			✓			✓						✓				
		Evolutionary Algorithms	[33]			✓		✓									✓		
		Software Configurations using FCA	[3]				✓	✓											✓
		Reverse Engineering Feature Models	[44]		✓			✓								✓			
		Component Oriented Architecture	[4]	✓							✓							✓	
		MoDisco	[11]	✓								✓							
		Language Independent Approach	[53]	✓					✓									✓	
2016-Current	3	Shatnawi et al.	[43]		✓					✓									✓
		Alshara et al.	[5]		✓						✓							✓	
		RX-MAN	[7]	✓							✓							✓	

Table 1: Trends in Software Reverse Engineering

- ***Technique:*** The parameter *Technique* is further classified based on the Gannod's and Cheng's framework as follows [26]:
  - ***Parsing Based:*** approaches use parsers like AST (Abstract Syntax Tree) to capture a code base for reverse engineering without losing any detail.
  - ***Plan Based:*** approaches use heuristics and define abstraction models to capture the source code.
  - ***Transformational:*** techniques transform one notation of semantics into another by specifying a formal context.
  - ***Translational:*** ones translate a program into an equivalent formal specification e.g., creation of a directed graph from source code.
- ***Retrieved Notation:*** shows us the output that each approach offers. This parameter has been classified into feature models, restructured code, non-explicit components (no defined composition), explicit components (components that follow a component model) and view based output (further classified into code traces and concept-lattices/graphs).
- ***Programming Language:*** represents the targeted language of a reverse engineering approach. The languages are classified into ***COBOL***, ***C***, ***C++***, ***Java***, ***OO*** (work on any object-oriented code) and ***LI*** (language independent approaches).

3. Trends in Software Reverse Engineering

Based on the proposed framework and 51 approaches that we have covered in this paper (Table. 1), there is a clear pattern that shows the variation of software reverse engineering since 90s.

Almost all the covered approaches from 90-2000s are parsing-based i.e., heuristics and plan-based reverse engineering was not developed enough to conduct the process of reverse engineering. All the approaches were applied and designed for ***COBOL*** and ***C/C++*** code bases i.e., ***Java*** was becoming popular in late 90s and was not considered because its code bases did not reflect legacy code in that era. The popular notations of the output of reverse engineering were components or graphical representations that can help in code comprehension although, the components' notations were not specified by some component model. Components in that era were usually defined as loosely coupled code chunks without proper definition of composition.

From 2001-2005, translation-based reverse engineering was the preferred way instead of parsing-based reverse engineering. Most of the approaches targeted object-oriented legacy code[4]. An important change in this phase is the preferred output of reverse engineering i.e., code trace that can help in finding the feature locations in a code base. Most approaches (e.g., [23] [27]) conducted dynamic reverse engineering to map legacy code bases in terms of the features of software.

From 2006-2010, the domain of reverse engineering was all about heuristics i.e., reverse engineering was based on plan-based or translation-based (translation via heuristics) methodologies. This was the era of visualisation-based reverse engineering i.e., all the approaches produced either code traces or concept lattices to visualise the dependencies e.g., [9] [40]. Many established domains like ***IR*** (information retrieval) and ***NLP*** (natural language parsing) were involved in reverse engineering to produce better results from heuristics (e.g., [1] [38]). In this phase, 61% of the covered approaches targeted Java i.e., the trend moved specifically towards Java rather than general ***OO***. It was justified due to the fact that by the end of this phase, many enterprise java code-bases were started to be considered legacy code.

From 2011-2015, the parsing-based reverse engineering was again on the rise i.e., 42% approaches from this phase were based on parsing. It was due to the fact that most approaches extracted ADL-based components/architectural-notations from the legacy code-bases. Such architectural notations demand preservation of the functionality of an original code base and heuristics cannot guarantee lossless extraction of architecture. 21% of the covered approaches were plan-based and a few of the approaches extracted features/feature models (e.g., [3] [44]). 53% approaches considered object-oriented code-bases.

The current phase (2016-current) of software reverse engineering still revolves around architectural re-usability. Software reverse engineering is moving away from graphs and code traces towards components. ***OO*** code in general and ***Java*** in particular is the favourite choice of current approaches. Table. 1[5] shows the overall statistics of the trends in software reverse engineering.

4. Conclusion

In this paper, we have covered more than 50 approaches to determine the trends and variations in software reverse engineering since 90s. Our framework shows that reverse engineering is moving from code comprehension and graphs towards components and architectural notations.

The adaptation in the techniques of reverse engineering went through phases of parsing-based, translational and plan-based reverse engineering whereas, most of the recent reverse engineering approaches are again in favour of parsing with an aim of architectural retrieval that requires the preservation of syntactic code structure.

References

[1] Surafel Lemma Abebe and Paolo Tonella. Natural language parsing of program element names for concept extraction. In *18th International Conference on Program Comprehension (ICPC), 2010 IEEE*, pages 156–159. IEEE, 2010.

[4] Such approaches can be applied on any object-oriented code though most of them chose ***Java*** as the targeted language.

[5] MoDisco is a framework and L2CBD is a methodology rather than concrete approaches therefore, no programming language/Technique is specified for them respectively.

[2] Marwan Abi-Antoun, Jonathan Aldrich, and Wesley Coelho. A case study in re-engineering to enforce architectural control flow and data sharing. *Journal of Systems and Software*, 80(2):240–264, 2007.

[3] R Al-Msie'Deen, Marianne Huchard, A-D Seriai, Christelle Urtado, and Sylvain Vauttier. Reverse engineering feature models from software configurations using formal concept analysis. In *CLA 2014: Eleventh International Conference on Concept Lattices and Their Applications*, volume 1252, pages 95–106, 2014.

[4] S. Allier, S. Sadou, H. Sahraoui, and R. Fleurquin. From object-oriented applications to component-oriented applications via component-oriented architecture. In *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, pages 214–223, June 2011.

[5] Zakarea Alshara, Abdelhak-Djamel Seriai, Chouki Tibermacine, Hinde Lilia Bouziane, Christophe Dony, and Anas Shatnawi. Migrating large object-oriented applications into component-based ones: Instantiation and inheritance transformation. *SIGPLAN Notices*, 51(3):55–64, October 2015.

[6] Nicolas Anquetil, Jean-Claude Royer, Pascal Andre, Gilles Ardourel, Petr Hnetynka, Tomas Poch, Dragos Petrascu, and Vladiela Petrascu. Javacompext: Extracting architectural elements from java source code. In *16th Working Conference on Reverse Engineering, 2009. WCRE'09.*, pages 317–318. IEEE, 2009.

[7] Rehman Arshad and Kung-Kiu Lau. Reverse engineering encapsulated components from object-oriented legacy code. In *Proceedings of The 30th International Conference on Software Engineering and Knowledge Engineering, 2018*. KSI Research Inc., 2018.

[8] Fatemeh Asadi, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. A heuristic-based approach to identify concepts in execution traces. In *14th European Conference on Software Maintenance and Reengineering (CSMR), 2010*, pages 31–40. IEEE, 2010.

[9] Johannes Bohnet and Jürgen Döllner. Analyzing feature implementation by visual exploration of architecturally-embedded call-graphs. In *Proceedings of the 2006 international workshop on Dynamic systems analysis*, pages 41–48. ACM, 2006.

[10] Johannes Bohnet, Stefan Voigt, and Jurgen Dollner. Locating and understanding features of complex software systems by synchronizing time-, collaboration-and code-focused views on execution traces. In *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008.*, pages 268–271. IEEE, 2008.

[11] Hugo Bruneliere, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032, 2014.

[12] Elizabeth Burd and Malcolm Munro. Investigating component-based maintenance and the effect of software evolution: a reengineering approach using data clustering. In *Proceedings of International Conference on Software Maintenance, 1998.*, pages 199–207. IEEE, 1998.

[13] Kunrong Chen and Václav Rajlich. Case study of feature location using dependence graph, after 10 years. In *18th International Conference on Program Comprehension*. IEEE, 2010.

[14] Elliot J. Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17, 1990.

[15] Landry Chouambe, Benjamin Klatt, and Klaus Krogmann. Reverse engineering software-models of component-based systems. In *12th European Conference on Software Maintenance and Reengineering, 2008. CSMR 2008.*, pages 93–102. IEEE, 2008.

[16] Markus Detten, Marie Christin Platenius, and Steffen Becker. Reengineering component-based software systems with archimetrix. *Software Systems Model.*, 13(4):1239–1268, October 2014.

[17] Marc Eaddy, Alfred V Aho, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008.*, pages 53–62. IEEE, 2008.

[18] Helen M. Edwards and Malcolm Munro. RECAST: Reverse engineering from COBOL to SSADM specification. In *Proceedings of 15th International Conference on Software Engineering, 1993*, pages 499–508, May 1993.

[19] Alexander Egyed. A scenario-driven approach to trace dependency analysis. *IEEE Transactions on Software Engineering*, 29(2):116–132, 2003.

[20] Alexander Egyed, Gernot Binder, and Paul Grunbacher. Strada: A tool for scenario-based feature-to-code trace detection and analysis. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 41–42. IEEE Computer Society, 2007.

[21] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of feature component maps by means of concept analysis. In *Fifth European Conference on Software Maintenance and Reengineering, 2001.*, pages 176–179. IEEE, 2001.

[22] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, 2003.

[23] Andrew David Eisenberg and Kris De Volder. Dynamic feature traces: Finding features in unfamiliar code. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM 2005.*, pages 337–346. IEEE, 2005.

[24] Ural Erdemir, Umut Tekin, and Feza Buzluca. Object oriented software clustering based on community structure. In *2011 18th Asia-Pacific Software Engineering Conference*, pages 315–321, Dec 2011.

[25] Jean Marie Favre, Frederic Duclos, Jacky Estublier, Remy Sanlaville, and Jean Jacques Auffret. Reverse engineering a large component-based software product. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, pages 95–104, 2001.

[26] Gerald C Gannod and Betty HC Cheng. A framework for classifying and comparing software reverse engineering and design recovery techniques. In *Proceedings of Sixth Working Conference on Reverse Engineering, 1999.*, pages 77–88.

IEEE, 1999.

[27] Orla Greevy, Stéphane Ducasse, and Tudor Girba. Analyzing feature traces to incorporate the semantics of change in software evolution analysis. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM 2005.*, pages 347–356. IEEE, 2005.

[28] S. Kebir, A. D. Seriai, S. Chardigny, and A. Chaoui. Quality-centric approach for software component identification from object-oriented code. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 181–190, Aug 2012.

[29] Rudolf K Keller, Reinhard Schauer, Sébastien Robitaille, and Patrick Pagé. Pattern-based reverse-engineering of design components. In *Proceedings of the 21st international conference on Software engineering*, pages 226–235. ACM, 1999.

[30] Haeng-Kon Kim and Youn-Ky Chung. Transforming a legacy system into components. In Marina Gavrilova, Osvaldo Gervasi, Vipin Kumar, C. J. Kenneth Tan, David Taniar, Antonio Laganá, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, pages 198–205, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[31] Soo Dong Kim and Soo Ho Chang. A systematic method to identify software components. In *11th Asia-Pacific Software Engineering Conference*, pages 538–545, Nov 2004.

[32] Sameer Kumar and Promma Phrommathed. *Research methodology*. Springer, 2005.

[33] Roberto Erick Lopez-Herrejon, José A Galindo, David Benavides, Sergio Segura, and Alexander Egyed. Reverse engineering feature models with evolutionary algorithms: An exploratory study. In *Search Based Software Engineering*, pages 168–182. Springer, 2012.

[34] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. Source code retrieval for bug localization using Latent Dirichlet Allocation. In *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*, pages 155–164. IEEE, 2008.

[35] Brian S. Mitchell and Mancoridis Spiros. On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, March 2006.

[36] Hausi A Müller, Mehmet A Orgun, Scott R Tilley, and James S Uhl. A reverse-engineering approach to subsystem structure identification. *Journal of Software: Evolution and Process*, 5(4):181–204, 1993.

[37] Andrzej Olszak and Bo Nørregaard Jørgensen. Featureous: a tool for feature-centric analysis of java software. In *18th International Conference on Program Comprehension (ICPC), 2010*, pages 44–45. IEEE, 2010.

[38] Denys Poshyvanyk and Andrian Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *15th IEEE International Conference on Program Comprehension, ICPC 2007.*, pages 37–48. IEEE, 2007.

[39] Martin P Robillard. Automatic generation of suggestions for program investigation. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 11–20. ACM, 2005.

[40] Martin P Robillard and Gail C Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *Proceedings of the 24th international conference on Software engineering*, pages 406–416. ACM, 2002.

[41] Abhishek Rohatgi, Abdelwahab Hamou-Lhadj, and Juergen Rilling. An approach for mapping features to code based on static and dynamic analysis. In *The 16th IEEE International Conference on Program Comprehension, ICPC 2008.*, pages 236–241. IEEE, 2008.

[42] Maher Salah, Spiros Mancoridis, Giuliano Antoniol, and Massimiliano Di Penta. Scenario-driven dynamic analysis for comprehending large software systems. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 10–pp. IEEE, 2006.

[43] Anas Shatnawi, Abdelhak-Djamel Seriai, Houari Sahraoui, and Zakarea Alshara. Reverse engineering reusable software components from object-oriented apis. *Journal of Systems and Software*, 131:442–460, 2017.

[44] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 461–470. IEEE, 2011.

[45] Mircea Trifu. Improving the dataflow-based concern identification approach. In *13th European Conference on Software Maintenance and Reengineering, CSMR 2009.*, pages 109–118. IEEE, 2009.

[46] Marco Tulio Valente, Virgilio Borges, and Leonardo Passos. A semi-automatic approach for extracting software product lines. *IEEE Transactions on Software Engineering*, 38(4):737–754, 2012.

[47] Neil Walkinshaw, Marc Roper, and Murray Wood. Feature location and extraction using landmarks and barriers. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 54–63. IEEE, 2007.

[48] MP Ward and KH Bennett. A practical program transformation system for reverse engineering. In *Reverse Engineering, 1993., Proceedings of Working Conference on*, pages 212–221. IEEE, 1993.

[49] Hironori Washizaki and Yoshiaki Fukazawa. Extracting components from object-oriented programs for reuse in memory-constrained environments. 2014.

[50] Yinxing Xue, Zhenchang Xing, and Stan Jarzabek. Feature location in a collection of product variants. In *19th Working Conference on Reverse Engineering, (WCRE). 2012*, pages 145–154. IEEE, 2012.

[51] Bo Zhang and Martin Becker. Recovar: A solution framework towards reverse engineering variability. In *4th International Workshop on Product Line Approaches in Software Engineering, (PLEASE), 2013*, pages 45–48. IEEE, 2013.

[52] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. Sniafl: Towards a static noninteractive approach to feature location. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(2):195–226, 2006.

[53] Tewfik Ziadi, Christopher Henard, Mike Papadakis, Mikal Ziane, and Yves Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1064–1071. ACM, 2014.