Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Optimizing Object Design Model in Software Engineering

Zeynab Rashidi
Master Student in Department of Mathematics and Computer Science,
AmirKabir University of Technology,
Tehran, Iran

Abstract: One of the modern paradigms to develop a system is object oriented analysis and design.The analysismodel focuses on the functionality of the system and does not take into account system designdecisions. During object design, we transform the object model to meet the design goalsidentified during system design. A major challenge, here, is that the direct translation of a design model into source code is often inefficient. In this paper, we make a survey on optimizations used in software design and architecture. Then, we present several techniques that can be used in optimizations of design, consisting of revising class design, maximizing inheritance, adding associations tooptimize access paths, collapsing objects into attributes, delaying expensive computations, andcaching the results of expensive computations. These techniques certainly improve efficiency in software design.

Keywords: Optimization, Class, Object, Design, Object-Oriented, Software Engineering

I. INTRODUCTION

One of the modern paradigms to develop a system is object oriented analysis and design. In this paradigm, there are several objects and each object plays some specific roles. The object model produced during an analysis object model describes the application domain concepts that the system manipulates and the user-visible interfaces of the system. After identifying objects, the various relationships among objects must be identified.

Object design is an activity during which developers define custom objects to bridge the gapbetween the analysis model and the hardware/software platform. This includes specifying object and subsystem interfaces, selecting off-the-shelf components, restructuring the object model to attain design goals, and optimizing the object model for performance. Object design results in the object design model.Moreover, there are many popular design modeling processes and guidelines such as GRASP [5] and ICONIX [4] for assigning responsibility to classes and objects in object-oriented design.

One the major solutions that often used in object design is to use design patterns (see [11], [12]). If the design pattern selection and the specification of class interfaces are done carefully,most design issues should now be resolved. We could implement a system that realizes the use cases specified during requirements elicitation and system design. However, as developers start putting together the individual subsystems developed in this way, they are confronted with many integration problems such as minimization of response time, execution time, or memory resources. To address these problems, different developers have probably handled contract violations differently.

During design, some parameters may be added to the application program interface (API) to address requirement changes.Additional attributes possibly be added to the object model, but are not handled by thepersistent management system, possibly because of a miscommunication. As the delivery pressure increases, addressing these problems results in additional improvised code changes and workarounds that eventually yield to the degradation of the system. The resulting code would have little resemblance to our original design and would be difficult to understand. For example, in the case of a Web browser, it might be clearer to representHTML documents as aggregates of text and images. However, if we decided during system design to display documents as they are retrieved, we may introduce a proxy object to represent placeholders for images that have not yet been retrieved.

The main motivation of this paper is to present some guidelines and techniques for optimizing object design model . The structure of remaining sections is as follows. In Section 2, the related works are described. Section 3 presents major techniques in optimization object design model. Finally, Section 4 is considered to summary and future works.

II. RELATED WORKS

Majority of system and application softwares are not designed with performance as a primary consideration[10]. Even if profiling and monitoring tools that can detect various problems are used, these tools do not provide a solution to optimize performance. It takes a very experienced designer or developer to deal with such problems efficiently. The recent researches in design practice areusing design patterns, disusing anti-patterns, doing refactoring,optimizing software architecture and product line architecture. In this section, we review these researches.

Gabor and Murphy (2002) paper present a framework that provides a way to automatically correct design problems preserving the functionality of the system and improving its performance[10]. The approach is to use a hybrid of patterns, anti-patterns and refactorings that produces an optimized system. The proposed framework uses patterns as the basis for

an expert system to detect performance problems and use refactoring to solve these problems in the system (See [14]). The research propose a tool that will discover the patterns associated with a given software system. The input to this tool is a UML representation of the system, possibly annotated with known problems. The output is a set of patterns associated with the software, indicating what improvements can be made.It is very interesting and would be desirable to have a system that would automatically refactor software based on the patterns detected.

Aleti et al. (2013) make a systematic literature review on software architecture Optimization methods in the last decades[9]. This research identified that software architecture optimization methods, which aim to automate the search for an optimal architecture design with respect to a (set of) quality attribute(s), have proliferated. However, the reported results are fragmented over different research communities, multiple system domains, and multiple quality attributes. To integrate the existing research results, this research have analyzed the results of 188 research papers from the different research communities. Based on this survey, a taxonomy has been created which is used to classify the existing research. Furthermore, the systematic analysis of the research literature provided in this review aims to help the research community in consolidating the existing research efforts and deriving a research agenda for future developments.

Aleti et al. (2013) made awider survey of literature on architectural optimization techniques[9].They have performed a systematic literature review and analyzed the results of 188 research papers from the different research communities. Based on this survey, a taxonomy has been created which is used to classify the existing research. Furthermore, the systematic analysis of the research literature provided in this review aims to help the research community in consolidating the existing research efforts and deriving a research agenda for future developments.

Automatic architectural optimization can be used to assist decision process, enabling designers to rapidly explore many different options and evaluate them according to specific criteria. To address the problem, Architecture Description Languages (ADLs) and EAST-ADL are developed. EAST-ADL is an architecture description language (ADL) intended for use in the automotive domain. Walker et al. (2013) present a multi-objective optimization approach based on EAST-ADL with the goal of combining the advantages of ADLs and architectural optimization[15]. The approach is designed to be extensible and leverages the capabilities of EAST-ADL to provide support for evaluation according to different factors, including dependability, timing/performance, and cost. The technique is applied to an illustrative example system featuring both hardware and software perspectives, demonstrating the potential benefits of this concept to the design of embedded system architectures.

In literature, several approaches have been introduced tospecify and detect code smells and anti-pattern (See [13], [23], [24], [25]). They range from manual approaches, based on inspection techniques [26], to metric-based heuristics ([27], [28]), using rules and thresholds on various metrics [29] or Bayesian belief networks [30]. An anti-pattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences. Performance anti-pattern support the interpretation of performance analysis results and help to fill the gap between numbers and design alternatives. Arcelli et al. (2015) present a model-driven framework that enables an early detection of performance anti-pattern, i.e., without generating performance models. Specific design features (e.g., the number of sent messages) are monitored while simulating the specified software model, in order to point out the model elements that most likely contribute for performance flaws. To this end, the research proposes to use fundamental UML (fUML) models instrumented with a reusable library that provides data structures (as Classes) and algorithms (as Activities) to detect performance anti-pattern while simulating the fUML model itself. A case study is provided to show the framework at work, its current capabilities and future challenges.

Potena et al. (2014)analyzes the challenges that quality decisions represent to software project managers[18]. This research indicates that projects goals are normally determined by the paradigm of the Iron Triangle of project management. In this triangle, managers need to know:(a) which are the effects of a quality assurance (QA) decision,(b) which effects in quality they can get but at what cost and (c) which effects may appear in terms of schedule. This decision problem is clearly related to existing disciplines like search-based software engineering (SBSE), multi-objective optimization and methods for return on Investment (ROI) calculation and value-based software engineering. This survey paper critically reviews the contributions of these disciplines to support QA decisions together with basic information from a pilot survey carried out as part of the developments of the Iceberg project funded by EU Programme Marie Curie.

To develop similar softwares from a shared set of software assets, software companies take advantage from software product line. A flexible andreusable Product Line Architecture (PLA) enables largescalereuse[31]. Colanzi and Vergilio (2014) did a comparative analysis of two multi-objective evolutionary algorithms in PLA design optimizations[19].To optimize PLA design, usually a multi-objective optimization problem is made and solved properly with search-based algorithms. The research indicates that search-based PLA design is an incipient research field. Due to this, works in this field have addressed main points to solve the problem: adequate representation, specific search operators and suitable evaluation fitness functions. Similarly what happens in the search-based design of traditional software, existing works on search-based PLA design use Non-dominatedSorting Genetic Algorithm (NSGA-II), without evaluating the characteristics of this algorithmsuch as the use of crossover operator. This research reports results from a comparative analysis of two algorithms, NSGA-II and Pareto Archived Evolution Strategy (PAES) to the PLA design problem. PAES was chosen because it implements a different evolution strategy that does not employ crossover. An experimental study was carried out with nine PLAs and results of the conducted study attest that NSGA-II performs better than PAES in the PLA design context.

Gabrel et al. (2014) propose a model based on 0-1 linear programming for automatically determining a transactional composite web service (CWS) from a service dependency graph that optimizes quality of service (QoS) measure[20]. The QoS measure used in this model can be either a classical weighted sum of QoS criteria or a MinMax-type criterion

such as response time. The transactional properties are a set of rules that ensures a reliable execution of the resulting CWS. The proposed 0-1 linear programmingmodel is solved using a standard solver (CPLEX). The experiments show that this new exact model surpasses two main related approaches: an approximate one based on transactional requirements and an exact one, based on 0-1 linear programming, but not dealing with transactional properties. In a large majority of the test sets used for our experiments, our model finds a better solution more rapidly than both related approaches and is able to guarantee its optimality. Moreover, our model is able to find the optimal solutions of big size test sets, as the ones proposed by the Web Service Challenge 2009.

Etemaadi (2014) did a dissertation, entitled as "Quality-driven multi-objective optimization of software architecture design: method, tool, and application"[21]. In the dissertation, the author mentions that Software architecting is a non-trivial and demanding task for software engineers to perform. In this research, an automated approach for software architecture design is proposed that supports analysis and optimization of multiple quality attributessuch as performance, safety, and cost.The research demonstrates an optimization approach for automated software architecture design. It reports the results of applying our architecture optimization framework to an automotive sub-system that was conducted based on a large-scale real world case study. Moreover, the research introduces two novel degrees of freedom which demonstrate how the number of processing nodes and their interconnecting network can be codified to fit into a genetic algorithm. The studies show that these extra degrees of freedom lead to better overall software architecture optimization. Additionally, the research proposes a new search-based approach for generating a set of optimal software architectural solutions for use in software product lines. The new approach analyses the commonality of the found optimal solutions and proposes a set of solutions which are suitable for the range of products defined by various feature combinations.

Mehiaoui et al. (2013) focus on three stages of the deployment methodologies at the same time[22]. These stages of the deployment consist of placement of functions on a distributednetwork of nodes, the partitioning of functions in tasks and thescheduling of tasks and messages. This research present a staged approach towardsthe efficient deployment of real-time functions based on geneticalgorithms and mixed integer linear programming techniques.Application to case studies shows the applicability of the methodto industry-size systems and the quality of the obtained solutionswhen compared to the true optimum for small size examples. The staged presented in this paper is very useful for complex real-time systems in industry and academia applicationsthat functions are deployed onto an execution platform.

Fokaefs et al. (2012) describe a method and a tool designed to fulfill exactly the extract class refactoring [6]. The method involves three steps: (a) recognition of extract class opportunities, (b) ranking of the opportunities in terms of improvement to anticipate which ones to be considered to the system design, and (c) fully automated application of the refactoring chosen by the developer. Bavota et al. (2014) propose an approach for automating the extract class refactoring [7]. This approach analyzes structural and semantic relationships between the methods in each class to identify chains of strongly related methods. The identified method chains are used to define new classes with higher cohesion than the original class, while preserving the overall coupling between the new classes and the classes interacting with the original class.

Senin et al. (1999) used Distributed Object-based Modeling Environment (DOME) and Genetic Algorithm (GA) to optimize a product design system[8]. DOME is a software framework for product design system modeling where designers are distributed geographically and make use of different software tools. In the framework, designers develop their own local software components, and distributed object technology is used to integrate their services via the Internet to form an overall system model. Designers can then explore alternatives by making changes to local models or remote services while observing how the entire model responds. This exploration is amenable to automated search, which involves both continuous parameters, changing the value of services, discrete changes and selecting different objects to substitute entire local models. In this research, a genetic optimization object and appropriate direct representation genomes and operators were developed for this purpose. The effectiveness of several genetic algorithms was compared and a new variation of restricted tournament selection (RTS) was developed. The RTS variation, called the Struggle GA, most reliably located the global optima and the most local optima. Other crowding algorithms reliably located the global optima but were less successful identifying multiple local

III. MAJOR TECHNIQUES IN OPTIMIZATION DESIGN MODEL

The previous section reviewed the latest research around software architecture optimization and the frameworks. The majority of the approaches considered the problem of optimizing conflicting quality attributes simultaneously[16]. Other approaches focus on effectively searching for better software architectures by either using smart problem-dependent heuristics or by combining the expression power of Architecture Description Languages (ADLs) with architecture optimization.If we look at a lower level of design or detailed design, we can observed that identifying performance problems is critical in the software design[17]. It is mostly because the results of performance analysis such as mean values, variances, and probability distributions are difficult to be interpreted for providing feedback to software designers. In this section, the major techniques in optimization design model are presented to which mainly related to minimization of response time, execution time, or memory resources.

3-1-Optimizing through Revising Class Design

During analysis, we focused on the logical structure of the information that is needed tobuild a business solution. During design, we need to look at the best way to implementthe logical structure that helps optimize the application performance. Many of the effectiveimplementation structures that we need are instances of container classes; examplesare arrays, lists, queues, stacks, sets, bags, dictionaries, associations, and trees. Mostobject-oriented languages

already have libraries that provide such classes. Although we have defined business algorithms for building a business solutionduring analysis, we may need to optimize the algorithms for implementation. Duringoptimization, we may add new classes to hold intermediate results and new low-levelmethods. These new classes are usually implementation classes that are not mentioneddirectly in the analysis model. They are usually service domain classesthat support the building of the application classes. When new methods are added,some have obvious target objects as their owner. However, some methods may haveseveral target objects for their owner. Assigning responsibility for the latter kind of servicecan be very frustrating. This is the fundamental problem when we invent implementationobjects; they are somewhat arbitrary and their boundaries are more a matterof convenience than of logical necessity"

3-2-MaximizingUse of Inheritance

In object-oriented software design, sometimes the same services defined across several classes and can be easily inherited from a common super-class. However, often the services in different classes are similar but not identical. By slightly modifying the prototype of the service, the services can be made to match so that they can be handled by a single inherited service. When this is done, riot only must the name and the signature of the service match, but they should all have the same semantic meaning. The following guidelines are commonly made to increase inheritance:

(a) When some services have fewer arguments than other services, we can add the missing arguments, but ignored in the method.
(b) When a service has few arguments because it is a special case of a more general service, we can implement the special service by calling the general services with all of the arguments.
(c) When attributes in different classes have the same semantic meaning, we choose one name for the attribute and move it to a common super-class.
(d) When services in different classes have the same semantic meaning, we must choose one name for the service and apply the guideline (a) or (b) to take advantage of polymorphism.
(e) When a service is defined on several different classes, but not in other classes that semantically should be in one group, we define the service in the super-class and declare it as no-op method in the class that does not care about providing this service.
(f) When a common behavior is recognized, we can create a common super-class to implement the shared behavior, leaving the specialized behavior in the subclasses. Usually, this new super-class is an abstract class.

It is strongly recommended that software engineers do not use inheritance as purely an implementation technique. This happens when developers find an existing class that has implemented a large number of the services needed by a newly-defined class, even though the two classes are different semantically. The developer may then want to use inheritance to achieve partial implementation of the new class. This can lead to side effects because some of the inherited methods may provide unwanted behaviors. It can also lead to brittle inheritance hierarchies that are difficult to change as the analysis model evolves to reflect changing requirements. A better technique is to use delegation which allows the newly formed class to delegate only the appropriate services.

3-3-Optimizing through Avoiding Recomputation

During the software execution, some computations are performed repeatedly. For example, in the education systems of each university, semester/session grade-point average and cumulative grade-point average are calculated repeatedly to represent numerically a student's quality based on his/her courses marks.To avoid recomputation to improve performance, we should define newobjects/classes to hold these derived attributes (data). We must remember that derivedattributes must be updated when base values change. This can be done by:

- **Explicit Code**: Because each derived attribute is defined in terms of one or more attributes of base objects, one way to update the derived attribute is to insert code in the update attribute method of the base object(s). This additional code would explicitly update the derived attribute that is dependent on the attribute of the base object. This is synchronizing by explicit code.
- **Periodic Recomputation**: When base values are changed in a bunch, it may be possible to recompute all the derived attributes periodically after all the base values are changed. This is called periodic recomputation.
- **Triggers**: An active attribute has dependent attributes. Each dependent attribute must register itself with the active attribute. When the active attribute is being updated, a trigger will be fired that will inform all the objects containing the dependent attributes that the active attribute has a changed value. Then it is the responsibility of the derived object to update its derived attribute. This is called updating by triggers.

3-4-Optimizing Multiplicity and Access Paths

One of the most relationships among objects is association. In this relationship, there are three common sources of inefficiency related to multiplicity and access paths[1]: (a) repeated traversal of multiple associations; (b)traversal of associations with “many” multiplicity, and (c) the misplacement of attributes.

- **Repeated association traversals:**To identify inefficient access paths, software engineers should identify operations that are invoked often and examine, with the help of a sequence diagram, the subset of these operations that requires multiple association traversal. Frequent operations should not require many traversals, but should have a direct connection between the querying object and the queried object. If that direct connection is missing, software engineers should add an association between these two objects. In interface and

reengineering projects, estimates for the frequency of access paths can be derived from the legacy system. In greenfield engineering projectsin which a development project starts from scratch, the frequency of access paths is more difficult to estimate. In this case, redundant associations should not be added before a dynamic analysis of the full system—for example, during system testing—has determined which associations participate in performance bottlenecks.

- **Many associations:**For associations with 'many' multiplicity, software engineers should try to decrease the search time by reducing the 'many' to 'one'. This can be done with a qualified association. For example, in a hierarchical file system each file belongs to exactly one directory in which each file is uniquely identified by a name. Many files can have the same name in the context of the file system; however, two files cannotshare the same name within the same directory. If it is not possible to reduce the multiplicity of the association, software engineers should consider ordering or indexing the objects on the 'many' side to decrease access time.
- **Misplaced attributes:**Another source of inefficient system performance is excessive modeling. During analysis many classes are identified that turn out to have no interesting behavior. If most attributes are only involved in set() and get() operations, software engineers should reconsider folding these attributes into the calling class. After folding several attributes, some classes may not be needed anymore and can simply removed from the model. The systematic examination of the object model using the above questions should lead to a model with selected redundant associations, with fewer inefficient many-to-many associations, and with fewer classes.

3-5-Optimizing via Collapsing Objects

After the object model is restructured and optimized a couple of times, some of its classesmay have few attributes or behaviors left. Such classescan be collapsed into an attribute, when they are associated only with one otherclass. This technique reduces the overall complexity of the model.We can considerFigure 1, as an example, a model that includes Persons identified by a 'SocialSecurity'object. During analysis, two classes may have been identified. Each Person is associated with a'SocialSecurity'class, which stores a unique social security number identifying the Person.Now, we assume that the use cases do not require any behavior for the 'SocialSecurity 'object andthat no other classes have associations with the 'SocialSecurity'class. In this case, the'SocialSecurity'class should be collapsed into an attribute of Person.

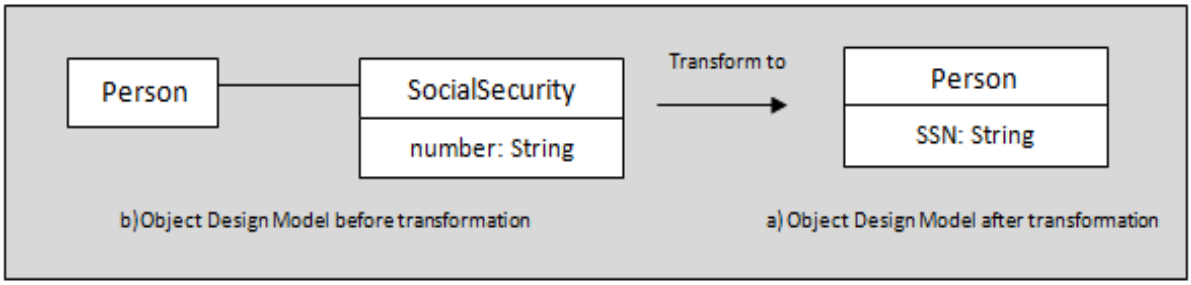

Figure 1: The Initial Object Design Model and its transformation (collapsing) without interesting behavior into an attribute

The decision of collapsing classes is not always obvious. In the case of a social securitysystem, the 'SocialSecurity'class may have much more behavior, such as specialized routinesfor generating new numbers based on birth dates and the location of the original application. Ingeneral, developers should delay collapsing decisions until the beginning of the implementation,when responsibilities for each class are clear. Often, this occurs after substantial coding hasoccurred, in which case it may be necessary to refactor the code.The refactoring equivalent to the model transformation of Figure 1 is Inline Classrefactoring [2]. The procedure of refactoring can be performed as below:

(a) Declare the public fields and methods of the source class (e.g., 'SocialSecurity') in the absorbing class (e.g., 'Person').
(b) Change all references to the source class to the absorbing class.
(c) Change the name of the source class to another name, so that the compiler catches anydangling references.
(d) Compile and test.
(e) Delete the source class.

3-6-Optimizing through Delaying Expensive Computations

In dynamic memory allocation, creation some specific objects are often expensive. However, their creation can often bedelayed until their actual content is needed. For example, consider an object representing animage stored as a file. Loading all the pixels thatconstitute the image from the file is expensive. However, the image data need not be loaded untilthe image is displayed. We can realize such an optimization using a *Proxy design pattern*[3]. An ImageProxy object takes the place of the Image and provides the sameinterface as the Image object (Figure 2). Simple operations such as 'width()'and 'height()'are handled by ImageProxy. When Image needs to be drawn, however, 'ImageProxy'loads thedata from disk and creates a 'RealImage'object. If the client does not invokes the 'paint()'operation, the 'RealImage'object is not created, thus saving substantial computation time. Thecalling classes only access the ImageProxy and the 'RealImage'through the Image interface.

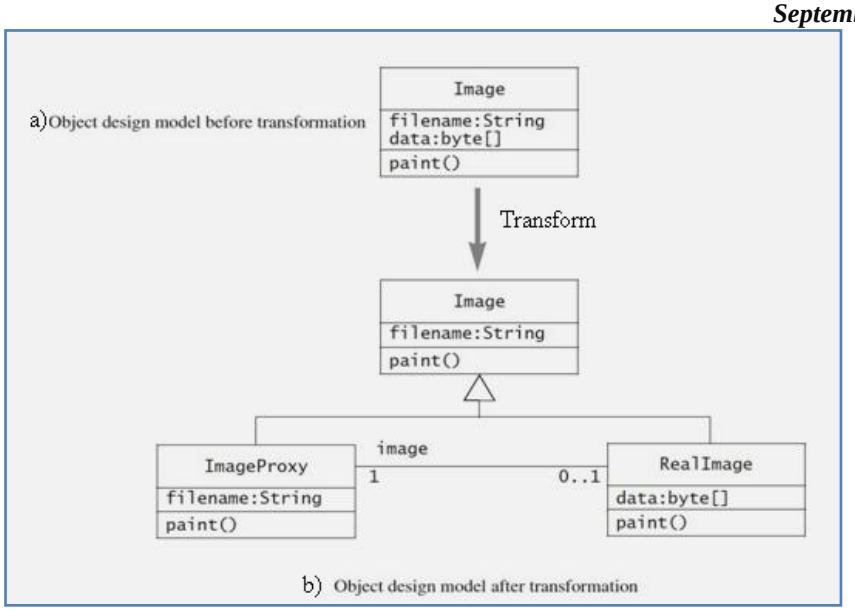

Figure 2: Theinitial Object design models and its transformation after Delaying expensive computations using a Proxy designpattern

3-7-Optimizing through Caching the Result of Expensive Computations

During software execution, some methods are called many times, but their results are based on values that do notchange or change only infrequently. Reducing the number of computations required by thesemethods substantially improve overall response time. In such cases, the result of thecomputation should be cached as a private attribute in the object. We can consider, for example, the method of 'getStatistics()'in the Science Olympiad. This methoddisplays the statistics relevant to allstudents and completions in the tournaments. These statistics change only when a competitionis completed,so it is not necessary to recompute the statistics every time a User wishes to see them. Instead,the statistics for a competitioncan be cached in a temporary data structure, which is invalidated thenext time a completionis completed. This technique includes a time-space trade-off. Although it improves the average response time for the 'getStatistics()'operation, we consumememory space by storing redundant information.

IV. SUMMARY AND CONCLUSION

Due to significant industrial demands toward optimizing software systems with increasing complexity and challenging quality requirements, software architecture design has become an important development activity and the research domain is rapidly evolving.There are many challenges which face designers of complex system architectures to optimize. The introduction of Architecture Description Languages (ADLs) has helped to meet these challenges by consolidating information about a system and providing a platform for modeling and analysis capabilities. However, optimizing this wealth of information can still be problematic, and managing of potential design decisions is still often performed manually. In this paper, we present seven techniques to optimize design model in object-oriented paradigm. These techniques are *Optimizing through Revising Class Design, Maximizing Use of Inheritance, Optimizing through Avoiding Recomputation, Optimizing Multiplicity and Access Paths, Optimizing via Collapsing Objects, Optimizing through Delaying ExpensiveComputations* and *Optimizing through Caching the Result Of Expensive Computation*s. When applying optimizations, developers must strike a balance between efficiency andclarity. Optimizations increase the efficiency of the system but also the complexity of themodels, making it more difficult to understand the system.

REFERENCES

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, & W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[2] M. Fowler, *Refactoring: Improving The Design of Existing Code*, Addison-Wesley, Reading, MA, 2000.

[3] E. Gamma, R. Helm, R. Johnson, & J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1994.

[4] D. Rosenberg, and M. Stephens, "Use Case Driven Object Modeling with UML: Theory and Practice", Apress, 2007.

[5] Larman C., Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edition, Prentice Hall, 2005

[6] Fokaefs M., Tsantalis N., Strouliaa E., Chatzigeorgioub A., "Identification and Application Of Extract Class Refactoring In Object-Oriented Systems ", Journal of Systems and Software, Vol. 85 , pp. 2241–2260, 2012.

[7] Bavota G., Lucia A. D., Marcus A., Oliveto R., Automating extract class refactoring: an improved method and its evaluation, Empirical Software Engineering, Vol. 19, pp. 1616-1664, 2014.

[8] Senin N., WallaceD.R., Borland N., Object-based Design Modeling and Optimization with Genetic Algorithms, In GECCO-99 Proceedings of the Genetic and Evolutionary Computation Conference, USA, 1999.

[9] Aleti A. , Buhnova B., Grunske L., Koziolek A., Meedeniya I., Software Architecture Optimization Methods: A Systematic Literature Review, CSDL Home IEEE Transactions on Software Engineering , Vol.39 (5), pp: 658-683, 2013.

[10] Gabor L., Murphy J., A Framework for Automated Software Design Optimization, Proceeding of IT&T Annual Conference, 2002

[11] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M., Pattern-Oriented Software Architecture - a System of Patterns, John Wiley & Sons, 1996

[12] Gramma E., Helm R., Johnson R., Vlissides J., Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[13] Smith C. U., Williams L. G., Software Performance Antipatterns, 2nd International Workshop on Software and Performance Proceedings, 2000

[14] Tokuda L., Batory D., Evolving Object-Oriented Designs with Refactorings, Kluwer Academic Publishers, 2001

[15] Walker M., Reiser M.O., Tucci-Piergiovanni S., Papadopoulos Y., Lönn H., Mraidha C., Parker D., Chen D.J., Servat D., Automatic optimization of system architectures using EAST-ADL, Journal of Systems and Software, Vol. 86 (10), pp. 2467–2487, 2013.

[16] Grunske L., Aleti A., Quality optimization of software architectures and design specifications, Journal of Systems and Software, Vol. 86 (10), pp. 2465–2466, 2013

[17] Arcelli D., Berardinelli L., Trubiani C., Performance Antipattern Detection through fUML Model Library,Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development, pp. 23-28, 2015

[18] Potena P., Fernández L., Pagés C., Diez T., Creating a Framework for Quality Decisions in Software Projects,Computational Science and Its Applications, Lecture Notes in Computer Science, Vol. 8583, pp. 434-448, 2014

[19] Colanzi T.E, Vergilio S.R., A Comparative Analysis of Two Multi-objective Evolutionary Algorithms in Product Line Architecture Design Optimization, 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), pp. 681-688, 2014

[20] Gabrel V., Manouvrier M., Murat C., Optimal and Automatic Transactional Web Service Composition with Dependency Graph and 0-1 Linear Programming Service-Oriented Computing, Lecture Notes in Computer Science, Vol. 8831, pp 108-122, 2014

[21] Etemaadi R., Quality-driven multi-objective optimization of software architecture design: method, tool, and application, Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden University, 2014

[22] Mehiaoui A., Wozniak E., Tucci-Piergiovanni S., Mraidha C., Natale M.D., Zeng H., Babau J.B., Lemarchand L., Gerard S., "A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems, in Proceedings of the 14th ACM SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems. ACM, pp. 121-132, 2013

[23] N. Moha, F. Palma, M. Nayrolles, B. J. Conseil, Y.-G. Guéhéneuc, B. Baudry, and J.-M. Jézéquel, "Specification and detection of soa antipatterns" in ICSOC, pp. 1–16, 2012.

[24] F. Khomh, M. D. Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," Empirical Software Engineering, Vol. 17(3), pp. 243–275, 2012.

[25] D. Romano, P. Raila, M. Pinzger, and F. Khomh, "Analyzing the impact of antipatterns on change-proneness using fine-grained source code changes," in Working Conference on Reverse Engineering (WCRE), pp. 437–446, 2012

[26] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality", ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 47–56, 1999.

[27] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in International Conference on Software Maintenance (ICSM), pp. 350–359, 2004.

[28] R. Oliveto, F. Khomh, G. Antoniol, and Y.-G. Guéhéneuc, "Numerical signatures of antipatterns: An approach based on b-splines," in European Conference on Software Maintenance and Reengineering (CSMR), pp. 248–251, 2010.

[29] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur, "Decor: A method for the specification and detection of code and design smells," IEEE Trans. Software Eng., Vol. 36(1), pp. 20–36, 2010.

[30] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. A. Sahraoui, "Bdtex: A gqm-based bayesian approach for the detection of antipatterns," Journal of Systems and Software, Vol. 84 (4), pp. 559–572, 2011.

[31] Linden F.V.D., Schmid F., Rommes E., Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. Springer, 2007

