In Medio Stat Virtus: Extract Class Refactoring through Nash Equilibria

Gabriele Bavota[1], Rocco Oliveto[2], Andrea De Lucia[1], Andrian Marcus[3]
Yann-Gaël Guéhéneuc[4], Giuliano Antoniol[4]
[1]University of Salerno, Fisciano (SA), Italy, [2]University of Molise, Pesche (IS), Italy
[3]Wayne State University, Detroit, MI 48202, USA, [4]École Polytechnique de Montréal, Montréal, Canada
gbavota@unisa.it, roliveto@unisa.it, adelucia@unisa.it, amarcus@wayne.edu, yann-gael@gueheneuc.net, antoniol@ieee.org

***Abstract*—Extract Class refactoring (ECR) is used to divide large classes with low cohesion into smaller, more cohesive classes. However, splitting a class might result in increased coupling in the system due to new dependencies between the extracted classes. Thus, ECR requires that a software engineer identifies a trade off between cohesion and coupling. Such a trade off may be difficult to identify manually because of the high complexity of the class to be refactored. In this paper, we present an approach based on game theory to identify refactoring solutions that provide a compromise between the desired increment in cohesion and the undesired increment in coupling. The results of an empirical evaluation indicate that the approach identifies meaningful ECRs from a developer's point-of-view.**

***Index Terms*—Design Quality, Refactoring, Game Theory.**

I. Introduction

Software systems evolve inevitably during their life-time to meet ever-changing user needs and to adapt to changes in their environment. Software evolution is often a non-structured process during which the pressure to reduce time to market may lead to design erosion and the introduction of poor design solutions, such as *design antipatterns*. A design antipattern is "*something that looks like a good idea, but which back-fires badly when applied*" [11]. It generally stems from experienced developers' expertise and describes common pitfalls in Object Oriented (OO) programming, *e.g.*, Brown's 40 antipatterns [9]. An example of antipattern is the *Blob* [9], also called *God Class*. It is represented by a large and complex class that centralizes the behavior of a portion of a system and only uses other classes as data holders, *i.e.*, data classes. Blob classes can rapidly grow out of control, making it harder and harder for developers to understand them, to fix bugs, and to add new features. They defeat the purpose of OO programming and of separation of concerns and responsibilities among classes, and usually have low cohesion.

Extract Class refactoring (ECR) is a widely used technique to remove the Blob antipattern [15], [24]. ECR aims at decomposing a class with several responsibilities into a set of new classes having individually (i) a higher cohesion than the original class and (ii) a better-defined set of responsibilities. Although it is possible to perform ECR manually, ECR might be very challenging and tool support is crucial [24]. Indeed, studies in the literature, *e.g.*, [20], highlighted that in large software systems there might be several Blobs, some of them having a huge size resulting from years of maintenance conducted by several developers. It is then difficult for a developer to gain the right knowledge to refactor them. Thus, having suggestions/feedback from a tool is desirable.

Applying ECR is likely to improve the overall cohesion of the system (*desired effect*). However, it is also likely that the overall coupling of the system will increase because of new dependencies between the extracted classes (*side effect*). Thus, a trade-off between cohesion and coupling must be found. Unfortunately, an optimal balance between contrasting goals such as cohesion and coupling is not easy to identify manually. Indeed, finding the partition of methods of a Blob class that maximizes the cohesion limiting at the same time the increase of coupling is a combinatorial multi-objective problem. Previous approaches addressed this problem (or similar problems, like remodularization) by looking for an approximation of the optimal solution. Search-based approaches reduced the solution space by iteratively selecting a subset of solutions that might lead to an optimal solution [29]. Clustering or graph-based approaches aggregate methods into classes by exploiting dependence or similarity measures between methods [4], [3], [14]. We show in this paper that game theory [13] also provides a suitable solution to the ECR problem. It is a branch of mathematics widely applied in the social sciences, especially in economics [25], to mathematically model the behavior of individuals in strategic situations in which an individual's success in making choices depends on the choices of others [13].

In this paper, we propose the use of game theory to model the ECR problem as a game between n players. Each player represents a class to be extracted from a given class and seeks to maximize its cohesion while maintaining its coupling as low as possible. A naive application of game theory would be to search for the Nash equilibrium [25] on a payoff matrix representing the quality of all possible partitions of methods among the players in terms of cohesion and coupling. However, such an application would have a too large solution space because it would require to compute all the possible partitions. Consequently, we model the ECR problem as an iterative multi-round game. Players begin the refactoring process with one *seed* method each, taken from the original class to be refactored; then they contend for the remaining methods to create new classes. This approach is iterative because, at each step, each player can choose one of

the unassigned methods. The chosen methods correspond to the Nash equilibrium computed on a payoff matrix that uses similarity measures between methods to assess the effect of the players' choice on the cohesion and coupling of their classes.

Similarly to partitioning clustering algorithms, the proposed approach requires as input the number of classes to be extracted, *i.e.*, the number of players, and a seed method for each player. This information might be difficult to derive from the analysis of a class that must be refactored due to its size and–or complexity. Thus, we propose a heuristic based on the analysis of the topics captured in the source code of the class to be refactored, using Latent Dirichlet Allocation (LDA) [8] to identify the candidate number of players and a candidate seed method for each player.

The proposed ECR approach and the new heuristic have been evaluated in two different case studies. First, classes with high cohesion in three open-source systems have been merged to create artificial Blobs and then refactored using our approach with the aim of reconstructing the original classes. We use this study to assess the parameters and the performance of the approach. Second, we performed a study in a real life scenario to refactor seven Blobs of an open-source system, namely GanttProject, to evaluate the actual usefulness of the proposed approach for actual developers. In both studies we compare the performances of the proposed approach with those achieved by using a well-know partitioning clustering algorithm, namely k-means. The results show the superiority of our approach.

The remainder of this paper is organized as follows. Sections II and III discuss the related literature and report essential background information. Section IV details our refactoring approach while Sections V and VI report and discuss the case studies in the artificial scenario and on real Blobs. Section VII concludes the paper.

II. Related work

The approach presented in this paper relates to extract class refactoring and game theory applied to software engineering.

A. Extract Class Refactoring

In the last decade, a lot of effort has been devoted to the definition of automatic and semi-automatic approaches for software refactoring. A survey of the related literature can be found in [24] while a discussion of more recent approaches can be found in [3]. We focus our discussion on ECR techniques.

Bavota *et al.* [4], [3] propose two approaches that support ECR based on graph theory. Both approaches represent the class to be refactored as a weighted graph in which each node represents a method of the class and the weight of an edge that connects two nodes represents a combination of the structural and semantic similarities of the two methods. In both the approaches, the similarity matrix representing the graph is first filtered to remove spurious relations between methods. A MaxFlow-MinCut algorithm is used in [4] to split the graph in two sub-graphs to obtain two sets of methods that should be placed in the same class. This approach always splits the class to be refactored in two classes, while the approach proposed in [3] can split a class in more classes: the transitive closure of the incident matrix is computed to identify sets of methods representing the new classes to be extracted. The game theory based ECR method proposed in this paper exploits the same structural and semantic measures used in these previous approaches. However, while the approaches in [4], [3] mainly aim at maximizing the cohesion of the extracted classes, our game theory approach is designed to find a fair compromise between the increase of cohesion in the extracted classes (desired effect) and the consequent increase in coupling (side effect).

Fokaefs *et al.* [14] use a clustering algorithm to perform ECR. Their approach analyzes the structural dependencies between the instance variables and methods of a class to be refactored. Using this information, they compute the *entity sets* for each instance variable (the set of methods using it) and for each method (all the methods invoked by it and all the instance variables that are accessed by it) of the class. Then, the Jaccard distance between all couples of entity sets of the class is computed to cluster cohesive groups of entities that can be extracted as separate classes. A hierarchical clustering algorithm is used. Differently from our approach, only structural information is taken into account.

B. Game Theory in Software Engineering

Only preliminary studies have been conducted on the application of game theory in software engineering, yet they serve as inspiration and support our decision to use game theory in our work. Grechanik *et al.* [16] model software development activities as a non-cooperative game involving the project stakeholders. In such a scenario, the Nash equilibrium is used to find a compromise between the contrasting goals of the different stakeholders.

Sazawal *et al.* [31] highlight how game theory can model software design decision-making. In fact, games naturally model the sequence of design decisions that must be made throughout the software development lifecycle, finding a compromise between contrasting goals, *e.g*, stand-alone architecture *vs.* layered architecture.

Bavota *et al.* [6] propose a preliminary approach to ECR based on game theory that can only to split a Blob class in two classes and has been only experimented on artificial Blobs. In this paper we generalize the approach to n players and present an experimentation to (i) assess and calibrate the approach and (ii) evaluate it in a real scenario.

III. Background

We provide background information about two fundamental aspects of our work: game theory and the structural and semantic measures that we use to take into account the impact of refactoring on cohesion and coupling.

A. Game Theory

The purpose of this section is to give a general introduction to game theory focusing on non-cooperative games using the prisoner's dilemma [30], a famous game example.

TABLE I
PAYOFF MATRIX FOR THE PRISONER'S DILEMMA

		Tom	
		confess	not confess
Sally	confess	**(5, 5)**	(0, 7)
	not confess	(7, 0)	(4, 4)

In bold the Nash equilibrium

Two brokers, Sally and Tom, are accused of fraudulent trading activities. Both Sally and Tom are being interrogated separately and do not know what the other is saying. Both want to minimize the amount of time spent in jail and here lies their dilemma: Table I represents the payoff matrix for their different possible choices, reporting the sentences for each player in consequence of their selected strategy. In particular (i) if *Sally (Tom) confesses and Tom (Sally) does not confess*, Sally (Tom) will not receive any sentence while Tom (Sally) will stay in jail for seven years; (ii) if *both decide to confess*, both Sally and Tom will receive a sentence of five years; (iii) if *both decide not to confess*, both Sally and Tom will receive a sentence of four years.

Traditional applications of game theory attempt to find equilibria in such games. The most studied equilibrium is the Nash equilibrium [25]. Nash equilibrium is a solution to any game involving two or more players, in which each player is assumed to know the equilibrium strategies of the other players and no player has anything to gain by changing only his own strategy unilaterally. Each finite game, *i.e.*, a game with a finite number of players and actions, have at least one Nash equilibrium in mixed strategies [25].

In Table I, there is only one Nash equilibrium: *(confess, confess)*. Indeed, if one of the players decides not to confess, then the other player can confess to minimize his/her own sentence. However, s/he would also sentence the other to the maximum punishment. Therefore, given the non-cooperative nature of the game, the minimum sentence for both players can be obtained only if both players confess. Of course the best solution for the two players is *(non confess, non confess)*, *i.e.*, a Pareto optimum[1]. The game presents two other Pareto optima, namely *(non confess, confess)* and *(confess, non confess)*, but these optima give the maximum payoff to only one of the two players *i.e.*, minimize only one of the sentences while the other is actually maximized.

Defining an automatic heuristic to choose the best solution for a non-cooperative game is usually hard. In fact, if we choose the solution with the minimum total sentence (maximum total payoff), we should choose between *(non confess, confess)* and *(confess, non confess)*, which both heavily penalize one of the two players. In this situation, the Nash equilibrium gives a solution that it is not necessarily a Pareto optimal solution but represents a compromise between two competing goals. We show in the following that such a compromise strategy is well suited for ECR.

[1]The Pareto optimum is achieved when it is not possible to improve the gain of a player without making worse the gain of another player.

B. Similarity Measures for Refactoring

Current ECR approaches [3], [4], [14] use heuristics to approximate an optimal solution to the problem of decomposing a class into two or more classes by maximizing the cohesion and minimizing the coupling. These heuristics exploit dependency or similarity measures that are the basis of cohesion and coupling metrics, instead of directly using cohesion and coupling metrics because ECR requires to analyze metrics at the method-level rather than at the class-level.

For example, structural cohesion and coupling metrics [10], [18], [21] take into account the amount of common instance variables referenced by methods or of calls among methods, while semantic cohesion and coupling metrics [22], [28] take into account the textual similarity among methods.

Previous work [22] shows that structural and semantic cohesion metrics do not correlate, which indicates that they capture different aspects of cohesion. Consequently, in our approach we use a combination of three structural and semantic measures, namely Structural Similarity between Methods (SSM) [18], Call-based Dependency between Methods (CDM) [3], and Conceptual Similarity between Methods (CSM) [22]. These measures have been previously used to identify ECR opportunities [3], [4] and they do not correlate [4].

The attribute references in methods are captured by Structural Similarity between Methods (SSM), used to compute the cohesion metric ClassCoh [18]. Let I_i be the set of instance variables referenced by method m_i. The SSM of m_i and m_j is calculated as:

$$SSM(m_i, m_j) = \begin{cases} \frac{|I_i \cap I_j|}{|I_i \cup I_j|} & \text{if } |I_i \cup I_j| \neq 0; \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the higher the number of instance variables the two methods share, the higher the likelihood that the two methods should be in the same class. SSM values are in $[0, 1]$.

The Call-based Dependency between Methods (CDM) [4] takes into account the calls performed by the methods. Let $calls(m_i, m_j)$ be the number of calls performed by method m_i to m_j and $calls_{in}(m_j)$ be the total number of incoming calls to m_j. $CDM_{i \to j}$ is defined as:

$$CDM_{i \to j} = \begin{cases} \frac{calls(m_i, m_j)}{calls_{in}(m_j)} & \text{if } calls_{in}(m_j) \neq 0; \\ 0 & \text{otherwise.} \end{cases}$$

$CDM_{i \to j}$ values are in $[0, 1]$. If $CDM_{i \to j} = 1$, then m_j is only called by m_i. Otherwise, if $CDM_{i \to j} = 0$, then m_i never calls m_j. To ensure that CDM represents a commutative measure (like the other two measures), the overall CDM of m_i and m_j is:

$$CDM(m_i, m_j) = \max \{CDM_{i \to j}, CDM_{j \to i}\}$$

Concerning the semantic information impacting cohesion and coupling, we use the Conceptual Similarity between Methods (CSM) introduced in [22] to define the Conceptual Cohesion of Classes and used in [28] to define the Conceptual Coupling between Classes. Two methods are conceptually related if their (domain) semantics are similar, *i.e.*, they perform conceptually

similar actions. To measure CSM, Latent Semantic Indexing (LSI) [12] is used to represent each method as a vector that spans a space defined by the vocabulary extracted from the methods. The conceptual similarity between two methods is then the cosine of the angle between their corresponding vectors:

$$CSM(m_i, m_j) = \frac{\overrightarrow{m_i} \cdot \overrightarrow{m_j}}{\|\overrightarrow{m_i}\| \cdot \|\overrightarrow{m_j}\|}$$

where $\overrightarrow{m_i}$ and $\overrightarrow{m_j}$ are the vectors corresponding to the methods m_i and m_j, respectively, and $\|\overrightarrow{x}\|$ represents the Euclidean norm of the vector x. The higher the value of CSM, the higher the similarity between two methods. Also CSM has values in $[0, 1]$.

IV. The ECR approach

The proposed ECR approach is based on an iterative algorithm that incrementally assigns the methods of a Blob class to n players $P_1, P_2, \ldots, P_n$ representing the classes to be extracted. The algorithm takes as input a class C to be refactored (*e.g.*, a Blob [9]), the number n of classes that should be extracted from C, and n *seed methods* (one for each player). The n players then contend for the remaining $l - n$ methods of C (where l is the number of methods of C) to build the new classes. The assignment of methods to the players made at each iteration is based on the Nash equilibrium computed on a payoff matrix that takes into account the effect of the assignment on the cohesion and coupling of the classes corresponding to the players.

In Section IV-A, we present the incremental ECR algorithm while, in Section IV-B, we detail how the payoff matrix is constructed. Finally, in Section IV-C, we present a heuristic based on Latent Dirichlet Allocation (LDA) [8] to suggest the input information for the ECR algorithm: number of players and seed methods. Our ECR algorithm is independent of this heuristic; we will study other heuristics in future work.

A. The Extract Class Refactoring Algorithm

Each player (*i.e.*, class to be extracted) initially holds only its *seed method*. The n players will then contend for the remaining $l - n$ methods of the class to be refactored. The construction of the classes is incremental. At each iteration, each player takes at most one of the unassigned methods of the original class (two players cannot take the same method). If a player does not take any method, then we say that it plays the *null* move. The purpose of the *null* move is to increase the rationality of a player, *i.e.*, a player takes a method only if there is a clear advantage in taking it. Without the *null* move, the refactoring process might result in a trivial splitting of the class to be refactored, where the original class is split in n new classes having the same number of methods (in case the number of methods of the original class is a multiple of n). We prevent players from all playing the $null$ move at the same time, as this would stop the iterative algorithm without assigning all the methods of the original class. Also, players cannot yield methods that they already got in previous iterations.

Let us consider the case of two players P_1 and P_2, one of the following pairs of moves is allowed during an iteration:

- P_1 takes m_i and leaves m_j to P_2 ($i \neq j$);
- P_1 takes m_i while P_2 plays the $null$ move;
- P_1 plays the $null$ move, while P_2 takes m_j.

The combination of moves to be performed by the n players during an iteration of the algorithm is chosen by finding the Nash equilibrium in the payoff matrix [27][2]. Thus, the payoff matrix represents the core of the approach. Each entry of the matrix corresponds to a combination of possible moves for the n players $P_1, P_2, \ldots, P_n$ and contains a tuple of payoffs $(p_1, p_2, \ldots, p_n)$ for this combination (a payoff for each player). At the first iteration, the size of the matrix is $(l - n + 1)^n$, as each player can take one of the unassigned $l - n$ methods of the original class or play the *null* move. For example, in the case of two players, the matrix will contain $(l - 1)$ rows and columns.

For an entry in the matrix, the tuple of payoffs $(p_1, p_2, \ldots, p_n)$ measures the effect of the corresponding tuple of moves on the cohesion and coupling of the classes under construction (see Section IV-B). The final goal of the refactoring algorithm is to split the original class in n classes by attempting to maximize their cohesion and minimize their coupling. Thus, at each iteration the higher the payoff for all players, the higher the expected increase of cohesion of the classes under construction and the lower the expected increase of coupling among them.

Once the tuple of moves to be performed has been identified, the methods taken by the players are added to the corresponding classes and removed from the payoff matrix that is then recomputed as follows: (i) the vectors corresponding to the assigned methods are removed from the matrix for all the players, as these methods cannot be assigned further (for example, in case of two players the rows and columns of the assigned methods are removed from the matrix) and (ii) the payoffs are re-computed to take into account the changes in the classes being extracted. The algorithm stops when each method of the original class is assigned to one of the n players, *i.e.*, one of the extracted classes.

Finally, the instance variables of the original class are distributed among the extracted classes according to how they are used by the methods in the new classes, *i.e.*, each instance variable is assigned to the new class having the higher number of methods using it. If a private field must be shared among the extracted classes, the implementation of the needed getter/setter methods is left to the developer. This implementation would not change dramatically the cohesion/coupling of the extracted classes because shared instance variables are considered by the SSM measure.

B. Computing the Payoffs

The payoffs in the matrix must take into account the effect of the moves (*e.g.*, adding a method to a class under

[2]If more than one Nash equilibrium is present in the payoff matrix, the exploited algorithm [27] always returns only one Nash equilibrium.

TABLE II
PAYOFFS FOR TWO-PLAYERS EXTRACT CLASS REFACTORING

p_1	p_2	Combination of moves
$Sim(C_1, m_i) - Sim(C_1, m_j)$	$Sim(C_2, m_j) - Sim(C_2, m_i)$	P_1 takes the method m_i and P_2 takes the method m_j
$\epsilon_1 - Sim(C_1, m_j)$	$Sim(C_2, m_j) - \epsilon_2$	P_1 plays the *null* move and P_2 takes the method m_j
$Sim(C_1, m_i) - \epsilon_2$	$\epsilon_1 - Sim(C_2, m_i)$	P_1 takes the method m_i and P_2 plays the *null* move
-1	-1	P_1 and P_2 take the same method or both play the *null* move

construction) on the cohesion and coupling of the new classes. We capture the effect of the moves on cohesion and coupling using a combination of the similarity measures presented in Section III, *i.e.*, SSM, CDM, and CSM. We compute the overall similarity between two methods m_i and m_j as:

$$sim(m_i, m_j) = w_{SSM} \cdot SSM(m_i, m_j) + w_{CDM} \cdot CDM(m_i, m_j) + w_{CSM} \cdot CSM(m_i, m_j)$$

where $w_{SSM} + w_{CDM} + w_{CSM} = 1$ and their values express the confidence (*i.e.*, weight) in each measure. These weights are empirically established and assessed in Section V.

The similarity between two methods is used to compute the similarity between a class under construction and a single method m_h that a generic player P_k can take during an iteration of our approach:

$$Sim(C_k, m_h) = \frac{1}{|C_k|} \sum_{m_l \in C_k} sim(m_l, m_h)$$

where C_k represents the set of methods already assigned to player P_k (*i.e.*, the class).

During an iteration of the game the goal of each player is to take a method having a high similarity with the class it is building and to leave methods having a low similarity with its class to the other players. Each player tries to maximize the cohesion of its class keeping low the coupling with the other classes under construction. The definition of the payoffs for a tuple of moves should balance these goals for the different players. For example, suppose that at a given iteration player P_k takes method m_i and leaves a non empty set of methods M_k to the other players. Then, the higher the similarity between C_k and m_i, the higher the expected increase of cohesion for C_k, the higher the payoff for the player P_k. On the contrary, the higher the similarity between C_k and the methods in M_k, the higher the expected increase of coupling between C_k and the other classes, the lower the payoff for the player P_k. Thus, we define the payoff p_k for player P_k as:

$$p_{k,i} = Sim(C_k, m_i) - \frac{\sum_{m_j \in M_k} Sim(C_k, m_j)}{|M_k|}$$

We also allow a player P_k to play the *null* move. This move is still desirable for P_k if the methods in M_k taken by the other players[3] have low similarity with the class C_k. Therefore, we define the payoff p_k as:

$$p_{k,null} = \epsilon_1 - \frac{\sum_{m_j \in M_k} Sim(C_k, m_j)}{|M_k|}$$

The value of ϵ_1 balances the negative effect of the similarity of class C_k with methods in M_k left to the other players: when the similarity between the class C_k and the methods in M_k is low, the payoff p_k encourages P_k to take the *null* move rather than taking a method with low similarity with its class.

Finally, a player P_k could be the only one taking a method m_i during an iteration, *i.e.*, all the other players play the *null* move. In this case, we cannot compute the average similarity between the class C_k and the methods left to the other players during this move, as the set M_k is empty and then the payoff p_k is defined as:

$$p_{k,i} = Sim(C_k, m_i) - \epsilon_2$$

The value of ϵ_2 balances the positive effect of the similarity of class C_k with method m_i and is used to discourage P_k to take a method with low similarity with C_k when the other players play the *null* move. Heuristics for the values for ϵ_1 and ϵ_2 are determined empirically in Section V.

As said in Section IV-A, we want to avoid that during an iteration two or more players take the same method or that all the players play the *null* move at the same time. For these combinations of moves, the payoff of all the players is -1 (the lowest possible payoff), which guarantees that such a combination of moves does not correspond to a Nash equilibrium and is not selected by the game theory algorithm. Table II shows the payoffs for the possible pairs of moves in case of two players.

C. Identifying the Players

The proposed approach requires as input the number of classes to be extracted, *i.e.*, the number of players, and a seed method for each player. This information is difficult to derive from the analysis of a Blob class due to its size and–or complexity. The problem is not unique to our solution, as other partitioning techniques also require similar inputs, *e.g.*, k-means [19], requires the number of classes and a centroid for each cluster.

The simplest solution is to leave this decision to the developers. However, in a Blob class, a manual identification of the set of different responsibilities of the class is intrinsically tedious and error prone. Thus, we employ a topic analysis technique using Latent Dirichlet Allocation (LDA) [8] to provide the developers with such information. The LDA based heuristic is rather simple, with little overhead, and it leverages the textual information present in the code. In some sense, it mimics the way a developer would read the code and assess the main responsibilities of the methods of a class based on the identifier names and comments, without a deeper structural analysis. LDA is a generative probabilistic model for collections of discrete data, such as text document corpora. In LDA, documents are modeled as a mixtures of various topics and a topic represents a set of words from the document

[3]If player P_k plays the *null* move, then M_k is not empty.

corpus. Our conjecture is that the main topics in the methods might be representative of the main responsibilities in the class to be refactored. These topics are captured by the textual information in the source code (identifiers and comments). Each method corresponds to a document in the corpus under analysis. Given a Blob to be refactored, we parse it to extract its methods and use LDA to extract from the methods a set of topics. LDA associates each method with a set of relevant topics together with a relevance score for each topic.

LDA requires as input the number of topics to be extracted. This might be a problem in large document corpora, where knowing the number of topics a priori is hard. In the context of ECR, we assume that a Blob cannot have more than 10 responsibilities: we ask LDA to extract 10 topics. Our experimental validation showed that this number of topics is reasonable and that, in most Blobs, the number of responsibilities is actually overestimated when using 10 topics. We deal with the overestimation by merging related topics. Indeed, we can extract similar topics, *i.e.*, topics described by a set of similar words, which could be representative of the same responsibility. To obtain a better estimation of the unique topics representing the responsibilities of the class, we identify similar topics and merge them in a single topic. The similarity between two topics T_i and T_j is computed using the Jaccard similarity coefficient:

$$sim(T_i, T_j) = \frac{|W_i \cap W_j|}{|W_i \cup W_j|}$$

where W_k represents the set of words describing the topic T_k. We merge two topics if their similarity is higher than a threshold s. We fix such a threshold at 0.5, *i.e.*, topics that share more than half of the words describing them are merged together. The final set of topics represents the responsibilities of the class to be refactored, which defines the number of players that will take part in the game. In addition, the *seed* method assigned to each player is the method with the highest relevance score with the topic corresponding to the player. The extracted information (set of players) is shown to the developers that can accept the suggestion or propose a different set of players. Note that while LDA could be suitable to identify the main topics embedded in a class, it does not represent a complete ECR solution. In fact, grouping together methods only based on textual overlap would completely miss structural dependencies existing between methods representing important cohesion/coupling indicators (see Section III).

V. Assessment of the Approach

The goal of this assessment study is to analyze the impact of the configuration parameters, *i.e.*, the weights of the similarity measures (w_{SSM}, w_{CDM}, and w_{CSM}) and the balancing parameters (ϵ_1 and ϵ_2), on the performance of our approach. We also assess the LDA-based heuristic to identify the input for the refactoring algorithm. For replication purposes, we posted more details and data online [7]. The study has been conducted on three open-source software systems: ArgoUML 0.16 (1,071 classes and 97 KLOC), GanttProject 1.10.2 (273 classes and 28 KLOC), and JHotDraw 6.0 b1 (275 classes and 29 KLOC). These systems have been previously used for assessing refactoring approaches [3], [4], [5]. An analysis of their quality based on commonly used metrics, namely Lack of Cohesion of Methods (LCOM2) [10], Conceptual Cohesion of Classes (C3) [22], Coupling Between Object classes (CBO) [10], and Message Passing Coupling (MPC) [21] indicates comparable levels of design quality (except for some outlier classes), which suggests, even without a quality model, that the overall quality of the systems is "good", especially as one of them, JHotDraw, has been developed as a design exercise.

A. Planning

To analyze the influence of the configuration parameters, we identified different refactoring solutions on the same classes using different weights for the adopted similarity measures and different values for the balancing parameters. To perform an automated assessment, we artificially created classes with more responsibilities and low cohesion from classes of the original systems having high cohesion. For each system, we randomly selected 150 classes with a cohesion higher than the average cohesion. Then, we randomly created groups of two or three classes and merged together the classes of the same group to create artificial Blobs, *i.e.*, we created a new class containing all the methods (except the constructors) and instance variables of the classes to be merged. For each system, we obtained 50 artificial Blobs merging three classes and 150 artificial Blobs merging two classes, because, from each artificial Blob composed of three classes C_1, C_2, and C_3, we generated three different artificial Blobs composed of two classes, *i.e.*, $B_1 = C_1 + C_2$, $B_2 = C_1 + C_3$, and $B_3 = C_2 + C_3$.

The proposed approach was then applied to split the artificial Blobs and to reconstruct the original classes. Given the observed quality of all the merged classes (higher than average cohesion), we consider them (*i.e.*, the original classes) as a *golden standard*. Hence, to evaluate the results, the refactored classes are compared with the original classes to count the number of methods correctly and incorrectly moved in the split classes. We use the F-measure [2], which is the harmonic mean of recall and precision, to quantify the reconstruction accuracy.

The study was organized in two parts. In the first part, we focus on the assessment of the parameters while in the second part we analyze the accuracy of the LDA-based heuristic to identify the input for the refactoring algorithm. For this reason, in the first part, we provided as input the exact number of players (*i.e.*, the number of original classes merged in the artificial Blob) as well as an exact seed method for each player (*i.e.*, a randomly selected non-constructor method of the original class). We refactored 200 artificial Blobs obtained by merging three classes (50 Blobs) or two classes (150 Blobs) for each system (600 artificial Blobs in total), experimenting all the possible combinations of configuration parameters (starting at 0 and increasing each parameter until 1 by a step of 0.1). This study resulted in 4,792,200 refactoring operations on the three systems. The large number of refactoring operations needed to exercise the many parameter configurations made

a manual evaluation prohibitive and justifies our choice to refactor artificial Blobs and automatically compare the results with the original classes.

In the second part of the study, we analyzed the accuracy of the initial game configuration provided by LDA on the 600 artificial Blobs created in the three systems. Given an artificial Blob B_m created merging m classes, namely $C_1, C_2, \ldots, C_m$, we expected that the LDA-based heuristic identifies m different responsibilities in B_m. Also, the proposed heuristic should assign to the identified responsibilities m different seed methods, each one belonging to a different original class, *i.e.*, $C_1, C_2, \ldots, C_m$. Therefore, we analyzed the accuracy of the LDA-based heuristic measuring the percentage of times that (i) the number of classes to be extracted and the seed method for each class are correctly identified (*LDA_Correct*), (ii) the number of classes to be extracted is correct while the seed methods chosen for them are wrong (*LDA_Wrong_Seed_Methods*), and (iii) the number of classes is wrong (*LDA_Wrong_#_Classes*). In addition, we evaluated the reconstruction accuracy of the proposed refactoring algorithm when the number of players and the seed methods are provided as input by the LDA-based heuristic. We used the parameter configurations resulting in the best reconstruction accuracy, identified in the first part.

B. Results

We analyzed the reconstruction accuracy achieved by our approach with all the possible combinations of configuration parameters. As for the balancing parameters ϵ_1 and ϵ_2, we observed that the performances of our approach are stable across different combinations of these two parameters. In particular, the values of the balancing parameters hardly affect the results when our approach is applied on Blobs composed of two classes. However, slightly better results are achieved when ϵ_2 is smaller than or equal to 0.4. When the number of merged classes is three, the performance of our approach seems to be slightly more sensitive to the values assigned to ϵ_1 and ϵ_2. The best reconstruction accuracy is achieved with any combination of balancing parameters having $\epsilon_1 \geq 0.3$. These observations are similar for all the systems [7].

To analyze the impact of the weights assigned to the similarity measures, *i.e.*, w_{SSM}, w_{CDM}, and w_{CSM}, on the performances of our approach we set $\epsilon_1 = 0.5$ and $\epsilon_2 = 0.2$, *i.e.*, one of the optimal configurations previously identified for both cases of merging two and three classes. The results reveal that the weight for the semantic measure (w_{CSM}) should be higher than 0.3. In fact, any combination of weights having $w_{CSM} \geq 0.4$ has a reconstruction accuracy almost equal to the best one. Moreover, even if our approach is stable across all the configurations of weights having $w_{CSM} \geq 0.4$, it shows slight decrease of performances when one (or both) the structural measures are set to zero.

Table III reports descriptive statistics of the reconstruction accuracy achieved with one of the optimal configurations (*i.e.*, $w_{CSM} = 0.5$, $w_{SSM} = 0.1$, $w_{CDM} = 0.4$, $\epsilon_1 = 0.5$, and $\epsilon_2 = 0.2$) on all the systems. There is no major difference between the results obtained on the three systems, indicating a high reconstruction accuracy as well as a high stability of the identified configuration parameters. We report on all the possible combination of parameters in [7].

TABLE III
RECONSTRUCTION ACCURACY ACHIEVED WITH THE BEST CONFIGURATION PARAMETERS (F-MEASURE)

System	Merging 2 Classes			Merging 3 Classes		
	Mean	Median	Std.dev.	Mean	Median	Std.dev.
ArgoUML	0.89	1.00	0.18	0.82	0.84	0.11
GanttProject	0.85	0.90	0.18	0.82	0.84	0.08
JHotDraw	0.92	1.00	0.13	0.86	0.85	0.08

Regarding the LDA-based heuristic to identify the initial game configuration, we applied LDA using the MALLET tool [23], which is an implementation of Gibbs' sampling algorithm. We run the tool for 100,000 sampling iterations, the first half of which are used for optimization [17]. We set the number of topics to extract to 10 and each topic is described by 20 words. Figure 1 reports the results achieved. The LDA-based heuristic identifies the correct number of classes and a correct *seed* method for each class in 65% of the cases on average. When the information provided by the LDA-based heuristic is correct, the performances of our approach (Game Theory + LDA_Correct in Figure 1) are almost equal to the performances achieved when manually providing to our approach the actual number of players derived from the oracle (Game Theory + Manual_Correct in Figure 1). Figure 1 also highlights that when the heuristic fails, in 31% of the cases on average, it fails in the identification of the correct number of classes because the approach uses the wrong seed method picked from the wrong topics. The heuristic correctly identifies the number of classes but provides the wrong *seed* methods only in 4% of the cases on average. In all these cases, as expected, the performances of the proposed ECR algorithm decrease.

C. Discussion

The approach used to assess and calibrate the proposed refactoring technique implies an important assumption, *i.e.*, the original classes represent a golden standard. Even if the object systems are generally well designed [4], there is the risk that the original classes are not an appropriate oracle, *i.e.*, classes with a high cohesion level and–or low coupling. This is a possible explanation that 100% accuracy was never achieved in average. To mitigate such a threat, we only considered classes having high cohesion (higher than the average cohesion of the classes of the considered system). To avoid biasing the study, the mutation of the original system was performed by a tool that randomly selected the classes to be merged from these chosen quality classes.

Another threat related to the design of our case study concerns the quality of the merged classes, *i.e.*, the artificial Blobs. In particular, the merged classes must have a low cohesion to test an ECR approach, *i.e.*, they approximate well real Blobs. We measured the cohesion, through LCOM2 and C3, for the original classes, the merged classes, and the refactored classes. The analysis reveals that the artificial Blobs

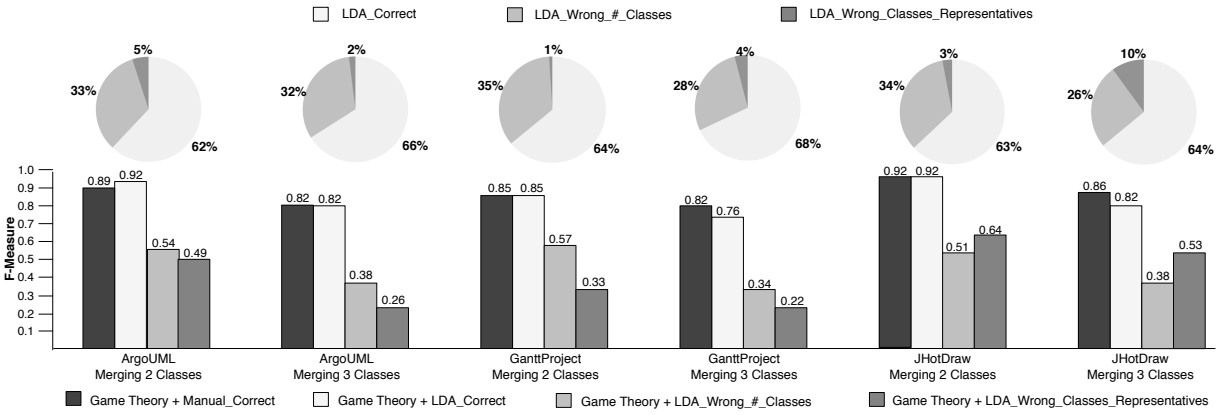

Fig. 1. Performances of Game Theory when using LDA to set-up the players.

used in our case study simulate well the type of classes that may be candidates for the ECR [7].

Another threat that could affect the results is represented by the input provided to the proposed iterative refactoring algorithm. In the first part of the study, we provided the number of classes to be split as well as randomly chosen correct seed methods to start the game. Even if such an approach is acceptable to assess the parameters of the approach in the ideal case, there is the risk that the artificial Blobs are too easy to refactor. For this reason, we compared the reconstruction accuracy of the proposed approach with the ones achieved by using a well-know partitioning clustering algorithm, namely k-means, because it takes similar inputs, *i.e.*, number of clusters to extract (corresponding to the players of our approach) and a method to be used as centroid for each cluster (corresponding to the seed method assigned to each player in our approach). To compare the two approaches exactly in the same conditions, the distance measure used by k-means was based on the same combination of similarity measures used by our approach. For k-means, we repeated the same study made for our approach to assess the weights of the similarity measures [7]. The achieved results indicate that our approach always overcome the reconstruction accuracy obtained with k-means, *i.e.*, on all the object systems, when merging two or three classes, and for all combinations of weights. The average difference of F-measure is about 11%. This result highlights that the refactoring of the artificial Blobs does not represent a trivial task for ECR approaches.

To analyze the practical effect of the LDA-based heuristic on the reconstruction accuracy of the proposed refactoring approach, we provided as input to the refactoring algorithm the information derived by using LDA. The results highlighted that when LDA correctly identifies the number of classes and seed methods, the accuracy of the refactoring algorithm is not affected at all. On the contrary, when LDA fails to identify the correct information, we observed a strong decrease of the reconstruction accuracy. To verify if the decrease of accuracy is related to our refactoring approach, we also used the same information to set the number of clusters and the centroids in the k-means partitioning technique. The achieved results indicated that with k-means there is also a similar decrease when the LDA-based heuristic fails to identify the correct number of classes to extract from the artificial Blobs or the correct seed methods for the classes. When the LDA-based heuristic correctly identifies the classes to be extracted, the reconstruction accuracy of the game theory-based approach overcomes the one obtained with k-means in all cases [7]. Future work includes an exhaustive study of the parameters of LDA, *i.e.*, the number of topics and the similarity threshold used to merge similar topics.

VI. Evaluation

We also evaluated the proposed ECR approach in a real-life usage scenario. Our approach was used to refactor seven Blobs of the GanttProject open-source software system, which have been manually identified and reported previously in the literature [20]. To refactor the Blobs, we used one of the best configurations identified during the assessment of the approach (see Section V), *i.e.*, $w_{CSM} = 0.5$, $w_{SSM} = 0.1$, $w_{CDM} = 0.4$, $\epsilon_1 = 0.5$, and $\epsilon_2 = 0.2$. LDA was used to determine the number of players and the seed methods.

A. Empirical study planning

We formulate the following research questions:

- **RQ**$_1$: Do the extracted classes have a higher cohesion than the original classes?
- **RQ**$_2$: Are the extracted classes meaningful from a functional point of view as assessed by developers?

To respond to our first research question, we compared the cohesion (measured by LCOM2 and C3) of the Blobs and of the refactored classes. We also computed coupling to verify that the proposed refactoring does not increase class coupling dramatically. To this aim, we used the MPC metric because it allows to understand if the extracted classes have a high method interactions.

For the second research question, we analyzed the refactored classes from a functional point of view. Two Ph.D. students and six M.Sc. students evaluated three different refactoring operations for each of the seven Blobs of the GanttProject system: (i) the refactoring suggested by our approach, (ii) the refactoring suggested by k-means, and (iii) a random

TABLE IV
RESULTS OBTAINED REFACTORING SEVEN BLOBS FROM THE GANTTPROJECT SYSTEM.

Blob Class	# Split Classes	Pre-refactoring					Post-refactoring				
		LOC	Methods	LCOM2	C3	MPC	LOC	Methods	LCOM2	C3	MPC
GanttGraphicArea	2	2,160	44	845	0.13	575	2,005	40	733	0.21	525
							166	6	1	0.33	56
GanttOptions	3	513	69	2,100	0.18	472	471	59	1,515	0.23	478
							31	5	8	0.49	3
							41	10	31	0.64	0
GanttProject	2	2,269	92	2,318	0.08	1,528	1,892	80	1,513	0.18	1,379
							409	13	23	0.69	155
GanttTaskPropertiesBean	1	1,685	28	183	0.13	276	*Not Split*				
GanttTree	2	1,730	49	649	0.14	358	1,382	43	493	0.22	262
							423	7	15	0.36	106
ResourceLoadGraphicArea	2	1,060	30	252	0.17	447	633	22	123	0.31	331
							442	9	20	0.41	118
TaskImpl	3	329	48	884	0.27	45	234	32	119	0.31	42
							69	12	58	0.38	6
							44	6	3	0.41	4

refactoring. The latter option was considered only to verify whether participants seriously considered this assignment[4]. We set k-means with its best configuration of parameters identified in our first case study, *i.e.*, $w_{CSM} = 0.5$, $w_{SSM} = 0.1$, $w_{CDM} = 0.4$. Moreover, also for k-means, LDA was used to determine the number of clusters to extract and the centroids for the clusters. For each of the proposed refactoring, the students had to express their level of agreement to the claim "*The proposed refactoring results in an appropriate division of responsibilities*" proposing a score using a Likert scale [26]: 1: *Strongly disagree*; 2: *Disagree*; 3: *Neutral*; 4: *Agree*; 5: *Fully agree*.

B. Results

Table IV compares the original Blobs and the classes refactored using our approach to answer our first research question ($\mathbf{RQ}_1$). The first column of the Table contains the name of the Blob class while the second column contains the number of classes resulting after refactoring (# Split Classes). The table also reports statistics concerning the LOC and the number of methods as well as the cohesion and the coupling for each Blob and for each extracted class. The results are positive. For almost all the classes, the cohesion is sensibly improved. Moreover, the refactoring solutions resulted in a very small increase of the total MPC value.

A particular case is represented by the class *GanttTaskPropertiesBean*, where our approach could not split the original class in two or more new classes. This is an Entity class having 67 instance variables and 28 methods. All the methods share many instance variables with the other methods and the calls among methods are many. Indeed, this Blob can be classified as "Data God Class" or "Lazy Class" [15] because the class holds a lot of the system's data in terms of number of instance variables. In this case, as suggested in [15], other types of refactoring should be applied to improve the quality of the class.

In summary, excluding the case of *GanttTaskProperties-Bean*, we obtained an average improvement of about 72% and 128% in terms of LCOM2 and C3, respectively. The average increase in terms of MPC is about 1%. Considering the significant improvement in terms of cohesion, we believe that the small increment in coupling is acceptable.

Concerning the qualitative assessment of our approach ($\mathbf{RQ}_2$), Table V reports the answers provided by the subjects expressing their level of agreement to our claim. The subjects gave higher scores on the Likert scale to the refactoring proposed by our approach. Indeed, the median of the scores given to our approach is 4 (Agree) against 3 (Neutral) achieved by k-means and 1 (Strongly Disagree) achieved by the random refactoring. Moreover, the refactoring suggested by k-means was never preferred over the refactoring suggested by our approach.

The overall results allow us to conclude that the ECRs identified by our approach lead to classes with higher cohesion than the original classes. The new classes only slightly increases the overall coupling of the system highlighting the ability of our approach to identify refactorings that provide a compromise between cohesion and coupling. The ECRs are also meaningful from a functional point of view as assessed by developers.

C. Threats to Validity and Discussion

The main threat to the validity of our study that could affect the generalizability of the reported results is represented by the subjects, *i.e.*, students, who evaluated the meaningfulness of the identified refactorings. Students had good analysis, development, and programming experience, and they can be considered close to junior industrial analysts. In addition, as highlighted by Arisholm and Sjoberg [1], the difference between students and professionals is not always easy to identify. Nevertheless, there are several differences between industrial and academic contexts. We plan to replicate the experiment with industrial subjects to corroborate the results. Another threat to the generalization of our results is related to the limited number of real Blobs analyzed. We plan in the future to replicate the experiment on a larger number of Blobs.

The resulting evaluation could have some degree of subjectivity, because the subjects did not have an extensive knowledge of the object system, *i.e.*, GanttProject. However, the evaluation aimed at giving us some qualitative insights to confirm the quantitative empirical evidence about the quality of the extracted classes previously measured in terms of cohesion and coupling. The subjects did not know the goal of our experimentation to avoid bias.

[4]The students involved in the experimentation were not aware of the employed refactoring techniques.

TABLE V
ANALYSIS OF THE REFACTORING OPERATIONS.

Class	Game Theory								*k*-means								Random							
	PhD	PhD	Master Students						PhD	PhD	Master Students						PhD	PhD	Master Students					
	I	II	I	II	III	IV	V	VI	I	II	I	II	III	IV	V	VI	I	II	I	II	III	IV	V	VI
GanttGraphicArea	4	4	4	4	3	5	4	5	3	2	2	3	3	2	3	3	1	1	2	1	1	1	1	1
GanttOptions	4	4	5	5	4	4	5	4	2	2	3	4	3	2	2	3	1	1	1	1	1	1	2	1
GanttProject	5	4	5	5	5	5	4	4	3	3	4	3	3	3	2	1	1	1	1	2	1	1	2	1
GanttTree	4	3	4	5	4	4	5	5	2	3	2	1	3	3	1	3	1	1	1	2	1	2	1	1
ResourceGraphicArea	4	4	4	5	4	4	4	4	3	2	3	3	2	3	3	3	1	2	1	1	1	1	1	1
TaskImpl	4	4	5	4	5	5	5	5	4	3	4	3	4	4	3	3	1	1	2	1	1	2	1	2

Finally, GanttProject has been used in both studies presented here, *i.e.*, the former aimed at assessing and calibrating the approach and the latter aimed at evaluating the proposed refactoring solutions. However, we are confident that this does not bias the results for two reasons: (i) the parameters of our approach are stable and not sensible to the different systems used in the assessment study and (ii) the classes used in the first experimentation (good quality classes) are different from the classes used in the second experimentation (Blobs).

VII. CONCLUSION

We presented an iterative algorithm to support the Extract Class Refactoring (ECR) that incrementally assigns the methods of a Blob class to n players representing the classes to be extracted. The assignment of methods to the players made at each iteration is based on the Nash equilibrium computed on a payoff matrix that takes into account the effect of the assignment on the cohesion and coupling of the classes corresponding to the players. To increase the usability of the approach, we also proposed a heuristic based on topic analysis using LDA to identify the candidate number of players and a candidate seed method for each player. Case studies showed that the proposed approach can refactor Blob classes into new meaningful classes with higher cohesion and marginal increment of coupling. Future work will be devoted to further automate our approach, extend its empirical validation, and compare the Nash equilibrium with other approaches within the iterative refactoring algorithms. In addition, we also plan to investigate other heuristics for the initialization step.

REFERENCES

[1] E. Arisholm and D. Sjoberg. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):521–534, 2004.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto. Automating extract class refactoring: an improved method and its evaluation. *Empirical Software Engineering*, page To Appear.

[4] G. Bavota, A. De Lucia, and R. Oliveto. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of Systems and Software*, 84:397–414, 2011.

[5] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto. A two-step technique for extract class refactoring. In *Proceedings of 25th IEEE International Conference on Automated Software Engineering*, pages 151–154, 2010.

[6] G. Bavota, R. Oliveto, A. D. Lucia, G. Antoniol, and Y.-G. Guéhéneuc. Playing with refactoring: Identifying extract class opportunities through game theory. In *Proceedings of the 26th IEEE International Conference on Software Maintenance (ERA Track)*, pages 1–5, 2010.

[7] G. Bavota, R. Oliveto, A. D. Lucia, A. Marcus, Y.-G. Guèhèneuc, and G. Antoniol. www.sesa.dmi.unisa.it/gt-refactoring.html.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[9] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray. *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1st edition, 1998.

[10] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[11] J. O. Coplien and N. B. Harrison. *Organizational Patterns of Agile Software Development*. Prentice-Hall, Upper Saddle River, NJ (2005), 1^{st} edition, 2005.

[12] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[13] M. Dresher. *The Mathematics of Games of Strategy: Theory and Applications*. Prentice-Hall, 1961.

[14] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander. Decomposing object-oriented class modules using an agglomerative clustering technique. In *Proceedings of the 25th International Conference on Software Maintenance*, pages 93–101, Edmonton, Canada, 2009.

[15] M. Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley, 1999.

[16] M. Grechanik and D. E. Perry. Analyzing software development as a noncooperative game. In *Proceedings of 6th International Workshop on Economics-Driven Software Engineering Research*, 2004.

[17] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(Suppl. 1):5228–5235, 2004.

[18] G. Gui and P. D. Scott. Coupling and cohesion measures for evaluation of component reusability. In *Proceedings of the 5th International Workshop on Mining Software Repositories*, pages 18–21, Shanghai, China, 2006. ACM Press.

[19] J. A. Hartigan. *Clustering Algorithms*. Wiley, 1975.

[20] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui. A bayesian approach for the detection of code and design smells. In *Proceedings of the 9th International Conference on Quality Software*, pages 305–314, Hong Kong, China, 2009. IEEE CS Press.

[21] W. Li and S. Henry. Maintenance metrics for the object oriented paradigm. In *Proceedings of the First International Software Metrics Symposium*, pages 52–60, 1993.

[22] A. Marcus, D. Poshyvanyk, and R. Ferenc. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions on Software Engineering*, 34(2):287–300, 2008.

[23] McCallum and A. Kachites. Mallet: A machine learning for language toolkit. 2002.

[24] T. Mens and T. Tourwe. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, 2004.

[25] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[26] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter Publishers, 1992.

[27] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a nash equilibrium. In *Games and Economic Behavior*, pages 664–669, 2004.

[28] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, 14(1):5–32, 2009.

[29] K. Praditwong, M. Harman, and X. Yao. Software module clustering as a multi-objective search problem. *IEEE Trans. Software Eng.*, 37(2):264–282, 2011.

[30] A. Rapoport and A. M. Chammah. *Prisoner's Dilemma*. University of Michigan Press, 1965.

[31] V. Sazawal and N. Sudan. Modeling software evolution with game theory. In *Proceedings of International Conference on Software Process*, pages 354–365, Vancouver, Canada, 2009.