Key Elements Extraction and Traces Comprehension Using Gestalt Theory and the Helmholtz Principle

Raphaël Khoury
and Lei Shi
Department of Computer Science and Mathematics
Université du Québec à Chicoutimi
Chicoutimi, Canada
{lei.shi1, raphael.khoury}@uqac.ca

Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering
Software Behaviour Analysis Research Lab
Concordia University
Montreal, Canada
abdelw@ece.concordia.com

Abstract—**Trace analysis techniques are used by software engineers to understand the behaviour of large systems. This understanding can facilitate various software maintenance activities including debugging and feature enhancement. However, traces usually tend to be very large, which makes it difficult for software engineers to unveil the key logic and functionalities embedded in a program's execution. Hence, it is necessary to develop methods and tools that can efficiently identify the important information contained in a large trace. In this paper, we propose an approach that builds on the concept of trace segmentation to extract the major components of a traced scenario. Our approach is based on Gestalt theory and the Helmholtz principle. We show the effectiveness of our approach by applying it to a dataset of large traces.**

I. Introduction

Trace analysis is essential to many software activities including debugging [1], performance analysis [2], feature enhancement [3], and a host of other tasks that require an understanding of the system behaviour. However, execution traces could be considerably large[1], in which case some form of abstraction must be used in order to analyse their content. Many trace abstraction techniques (see [4] for a survey) exist in the literature. They mainly focus on reducing the size of traces by eliminating (manually or automatically) utilities and other low-level components.

Recently, another class of trace abstraction methods has emerged. It seeks is to segment large traces into short and meaningful trace segments that capture the various execution phases of the program. An execution phase is defined as a coherent set of functions that implement a specific functionality of a program. For instance, a trace that is generated during the execution of a compiler could contain the following phases: initialisation, preprocessing, lexical analysis, syntactic analysis, semantic analysis, etc. The main difficulty lies in pinpointing the boundary between each segment in the trace. This is because there is no support at the programming language level that helps programmers indicate the beginning and end of each phase.

Despite the fact that trace segmentation research is a relatively new field of study, several techniques have already been developed (see [5], [6], [7], [8], [9]). Recently, Pirzadeh and Hamou-Lhadj [5], [10] proposed a trace segmentation approach inspired by Gestalt laws [11] (more precisely laws of similarity and continuation), which are used in psychology to describe the operational principles of the human brain, particularly the ability of humans to visually recognize objects and shapes as a whole and not just as points and lines. Pirzadeh and Hamou-Lhadj designed a mathematical model based on these laws using traces as objects and trace events as points and lines. They were thus able to derive a method that automatically segments large traces into phases by grouping coherent events together using this mathematical model. Once the phases are identified, [5], [10] showed how a "summary" of each phase, containing its most relevant events, can be constructed by treating each phase as a document and applying *Term Frequency-Inverse Document Frequency* (TF-IDF) to rank the functions of each trace according to its relevance.

In this paper, we seel to improve the ranking of the most relevant functions[2] using another one of Gestalt theory principles, the Helmholtz Principle. The Helmholtz Principle is the quantitative version of Gestalt's general grouping law and is usually used for image processing. In [12], the Helmholtz Principle is used to automatically extract the keywords for document processing. Following this study, we applied the Helmholtz principle on execution traces, treating the trace file as a type of document. By using this approach, we make two important improvements to the study proposed by Pirzadeh and Hamou-Lhadj [5], [10]:

1. We propose an alternative method to extract key elements from trace segments, based on the Helmholtz Principle rather than on TF-IDF. This alternative method has a stronger theoretical foundation and allows us to present an end-to-end approach that is completely based on Gestalt theory.

2. We apply our new algorithms to the problem of program comprehension and report the results of the experiment. We show that after traces have been segmented, we can use the key elements extracted from each segment to reconstruct the probable behaviour of the original sequence. It is interesting to note that the use of two alternative and independent key element extraction algorithms allows for a more precise reconstruction since the results of the two algorithms can be combined or contrasted.

[1]For example, the traces used in section 4 capture the functions calls performed by JHotDraw for short scenarios, but can contain up to 50 000 function calls.

[2]In this paper, we focus on traces of function calls.

This paper is organized as follows: in Section II, we discuss in more detail the trace segmentation approach proposed by Pirzadeh et al., on which our method is built. In Section III, we analyse the use of the Helmholtz principle for key trace elements extraction, while experimental results are given in Section IV. Related work is presented in Section V. Concluding remarks and insights for future work are presented in Section VI.

II. The Gestalt theory and the trace analysis

Gestalt (a German word for form or shape) psychology is a theory of perceptual understanding. "*Gestalt psychology seeks to understand the laws that govern our ability to acquire and maintain meaningful perceptions in an apparently chaotic world*", and the way our perceptual systems follow certain grouping principles (e.g., good continuation, proximity, and similarity properties of the elements) [11].

In previous research, [5], [10], Pirzadeh et al. proposed an approach for analysing execution traces and extracting key information of each trace phase. This approach operates in two stages and is inspired by the Gestalt theory. The first stage, called trace segmentation, deals with the automatic division of the trace into phases with re-grouped and clustered trace elements. In the second stage, called key trace elements extraction, a "summary" is extracted from each of the segments, generated from stage 1, consisting of the most representative and meaningful trace elements according to their weighted value. In [10], this step was accomplished using a TF-IDF algorithm for content prioritization. In this paper, we propose an alternative method to accomplish this task, based on Gestalt theory.

The trace segmentation process itself operates in two steps namely 1) phase detection and 2)phase boundary identification. In the first step, trace elements are grouped into candidate phases using gravitational schemes that apply the technique based on Gestalt laws of perception (principle similarity and good continuation). The second step consists in identifying the boundary of each phase using a k-means clustering algorithm.

In [10], the phase detection and segmentation algorithm was successfully applied to large traces captured from two object-oriented systems.

III. The Helmholtz Principle for Key Trace Elements Extraction

After the trace has been divided into several meaningful segments using the approach described above, it is still too large to be understood by a programmer. There is thus a need for an algorithm that can easily identify the major program functionalities present in each phase. In this paper, we develop an approach inspired by Helmholtz Principle, that can automatically extract the key trace elements, i.e., the elements that are most meaningful to the execution phase.

According to the Helmholtz principle, "*we [humans] immediately perceive whatever could not happen by chance*"[13]. This means that a structure is easily recognized when it exhibits a large deviation from randomness. For example in Figure 1, a group of five aligned dots appears in both images. These dots can be easily perceived in the right hand side image because it exibits a large deviation from randomness, which is unlikely to happen by chance. By contrast, the same aligned dots are difficult to identify in the left hand side image, where they are submerged by a large quantity of random information.

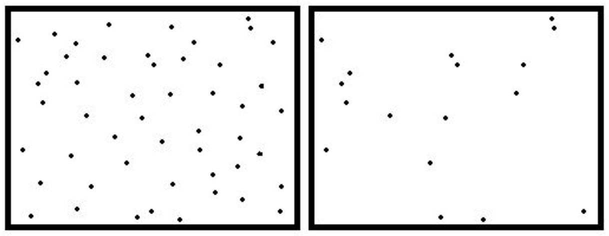

Fig. 1. The Helmholtz principle in human perception (from [12])

In order to evaluate the degree of deviation of a trace event, we introduce a universal variable, called Number of False Alarms (NFA), which represents the expectation of the number of occurrences of an event. When NFA is less than ε, we say that this event is "ε-meaningful". Intuitively, ε fixes an upper bound on how likely it is that a pattern being observed—in our case the unusual frequency of a given function call—is actually due to chance.

With regard to the analysis of function calls in a trace, we designate as *a meaningful event* any function call that occurs in a given phase at a rate that exhibits a large deviation from randomness, as compared with its rate of occurrence in the entire trace. Dadachev et al. [12] proposed a method that relies upon a qualitative measure of these deviations in order to automatically extract keywords from a document. Their method thus allows a user to extract the most meaningful keywords from a text written in natural language, with no preprocessing or parametrization of the algorithm. We present their method in the remainder of this section, then show how it can be applied to the trace segmentation problem in Section IV.

Consider a trace T partitioned into P phases (T_1, T_2, …, T_P), using the method given in stage 1. Let M be a function call that is present in one or more of these P phases. Assume that the method call M appears a total of K times in all P phases. These occurrences of M are collected in a set $S_M = \{M_1, M_2, ..., M_K\}$.

Now consider the possibility that the function call M occurs n times in some phase T_P. The verification of whether the number of occurrences of M in T_P is either an unexpected or expected event proceeds as follows:

Assume that the number of occurrences from S_M are uniformly and independently distributed into the P phases and let C_n (with $1 \leq n \leq K$) be an n-tuple capturing the number of occurrences of the function calls in S_M appearing in each phase. For n-tuples of the method calls $i_1, i_2, ..., i_n$ between 1 and $K(1 \leq i_1 < i_2 < ... < i_n \leq K)$, the random variable:

$$\mathcal{X}_{i_1,i_2,...,i_n} = \begin{cases} 1 & \text{if } M_{i_1,i_2,...,i_n} \text{ are in the same phase} \\ 0 & \text{otherwise} \end{cases}$$

It follows that the definition of the variable C_n is:

$$C_n = \sum_{1=<i_1<i_2<\cdots<i_n<=K} \mathcal{X}_{i_1,i_2,...,i_n}$$

Therefore the expected number of occurrences of n-tuples of function calls is the sum of all expected values of $\mathcal{X}_{i_1,i_2,...,i_n}$:

$$E(C_n) = \sum_{1=<i_1<i_2<\cdots<i_n<=K} E(\mathcal{X}_{i_1,i_2,...,i_n})$$

As $\mathcal{X}_{i_1,i_2,...,i_n}$ takes only values of zero or one, $E(\mathcal{X}_{i_1,i_2,...,i_n})$ is equal to the probability that all $M_{i_1}, M_{i_2}, ..., M_{i_n}$ belong to the same phase, i.e.

$$E(\mathcal{X}_{i_1,i_2,...,i_n}) = \frac{1}{P^{n-1}}$$

From the identities given above it can be concluded that:

$$E(C_n) = \binom{K}{n} \cdot \frac{1}{P^{n-1}}$$

where $\binom{K}{n} = \frac{K!}{n!(K-n)!}$ is a binomial coefficient.

According to [13], we can define $E(C_n)$ as the number of false alarms (NFA) needed to measure the “meaningfulness”of an event with the following expression:

$$NFA(n, K, P) = \binom{K}{n} \cdot \frac{1}{P^{n-1}}$$

Here K is defined as the sum of occurrences of M in trace T, n is the sum of occurrences of M in each divided phase, and P is the number of the total phases.

If the function call M appears n times in the same phase of the trace, then this function call is designated as a meaningful trace element if and only if its NFA is smaller than 1. If the NFA is less than the threshold ε, we say that M is $\varepsilon-$meaningful. Likewise, the set of $\varepsilon-$key elements for a trace is the set of events for which NFA $< \varepsilon$. The smaller ε becomes, the more meaningful the event is. However, calculating NFA in practice is not easy since the values K, n, and P could be very large. Indeed, a small change of n can lead to a large fluctuation in NFA. For this reason, we use the following expression, described in [13], to measure the meaningfulness of an event:

$$\text{Meaningful}(n, K, P) := -\frac{1}{n} logNFA(n, K, P)$$

In this case, a set of meaningful events is defined by: Meaningful$(n, K, P) > 0$. If Meaningful(n, K, P) is greater than ε, we call it $\varepsilon-$meaningful.

This algorithm is presented as Algorithm 1.

Algorithm 1 The Helmholtz principle algorithm

```
Require: Program segments $T_1$ to $T_P$, as divided in Stage 1
1: for all trace segments $T_1$ to $T_P$ do
2:     calculate the number of times K the method call M appears
3: end for
4: for each trace segment $T_1$ to $T_P$ do
5:     calculate the number of times $n_i$ the method call $M$ appears in the segment $T_i$
6:     calculate Meaningful $(n_i, K, P)$
7:     if Meaningful $(n_i, K, P) > \varepsilon$ then
8:         add M to the set $KM(T_1, ..., T_P)$ and mark $M$ as a $\varepsilon-$meaningful event for $T_i$
9:     end if
10: end for
```

IV. Case Study

We now conduct a case study in order to assess the effectiveness of using the Helmholtz Principle in identifying the key elements of a trace. We also compare the results with the TF-IDF as proposed by Pirzadeh and Hamou-Lhadj [10].

The traces used in this study are generated from JHotDraw[3] (version 5.2), the target system. JHotDraw is a framework implemented in Java for technical and structured graphics. It consists of 11 packages, 171 classes, 1414 methods and 9419 lines of codes. We generated 40 traces from JHotDraw[4] using TPTP[5], an eclipse plug-in. Each trace contained between 2 and 4 phases, with each phase consisting of drawing a single one of 6 possible shapes namely: a rectangle, a triangle, a polygon, a rounded-rectangle, an ellipse, or a diamond. Moreover, each trace begins with a start-up phase, (in which the program was launched) and ends with a shutdown phase. The traces were then segmented using the algorithm described in [10]. In total, the 40 traces were partitioned into 191 segments, of 8 possible segment-types (the six shapes, plus the start-up and shutdown phases).

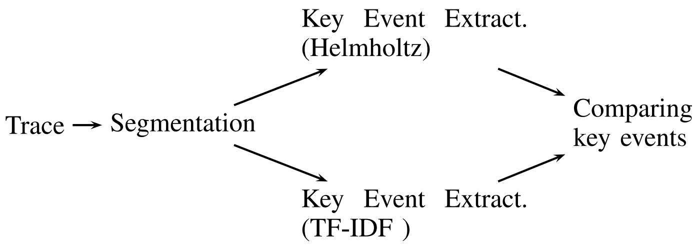

Fig. 2. The process followed in the case study

We generated the key events for each segment using both TF-IDF and Helmholtz algorithms. A single occurrence of each segment-type was chosen randomly and used as a reference to construct a library of 8 key event sets. We then attempted to match each of the remaining 183 key event sets with its corresponding library entry. Let K_a be the set of key events in a segment a. We compute the similarity between segment a and segment b from the library as follows:

$$similarity_{a-b} = K_a \cap K_b/(K_a - (K_a \cap K_b)).$$

This formula was derived at experimentally after trying a number of alternatives. In Section VI, we explore alternative metrics for segment assignation. Figure 2 depicts the process we followed in the case study.

We computed the similarity of each of the 183 segments with each of the 8 segments present in the library. A segment was assumed to be of the type for which the evaluation of the similarity function was the highest. The value returned by this function also serves as a measure of the confidence we have in this assignation. These results are shown in Table I. As the the table shows the key event sets generated by the Helmholtz algorithm were correctly matched 62% of the time (115 out of 183), while those generated using the TF-IDF approach were correctly matched 59% of the time (108 out

[3]www.jhotdraw.org

[4]The traces used in this experiment are available at: https://datahub.io/dataset/jhotdraw-sample-traces

[5]www.eclipse.org/tptp/

of 183). The success rate increases to 80% for Helmholtz and 83% for TF-IDF when we consider the top two matches of each segment. We also observe that correct matching reaches a high degree of confidence (on average 74% for TF-IDF and 90% for Helmholtz) whereas incorrect matchings average 60% and 50% respectively. These results indicate that erroneous matches could be easily identified and suppressed if a cutoff point for the validity of results is established. Figure 3 exhibits a Receiver operating characteristic graph that illustrates the improvement in accuracy as the threshold for accepting valid results increases, peaking at 87% accuracy for Helhmoltz.

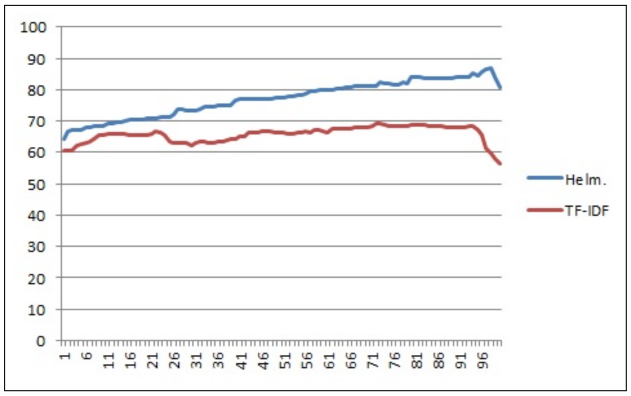

Fig. 3. Receiver operating characteristic graph

TABLE I. EXPERIMENTAL RESULTS

	Correct	confidence for correct assignation	Incorrect	confidence for incorrect assignation
TF-IDF	108	60%	75	60%
Helmholtz	115	90%	68	50%

Results are even more encouraging when broken down by event type, as shown in table II, which reports the number of correct (true positives) and incorrect identifications (false negatives). Note that the most distinctive segment types, namely start-ups and shutdowns, are almost always correctly detected. This is an interesting result since these program phases are the most distinctive from a software engineering or program maintenance perspective. Two other segment-types, polygon and diamond, are also typically correctly identified. By manual inspection of the results (using the source code comments and JHotDraw website), we were able to detect a number of patterns in mistaken assignations, which we seek to correct in future work. First, the algorithm seems to have had much difficulty in differentiating rectangles and round rectangles. It could be that JHotDraw mainly uses the same functions in drawing both figures. When either rectangle (resp. round rectangle) was misidentified by the algorithm, the second choice was often a round rectangle (resp. a rectangle), with a confidence only slightly lower.

The algorithm seems to have particular difficulty in recognizing triangle segments. This could result from the fact that JHotDraw does not use any unique functions when drawing a triangle, but rather uses the same functions as for polygons.

Table III and IV show the results for Precision (TP/TP+FP), Recall (TP/TP+FN) and Accuracy (TP+TN/total) for Helmholtz and TF-IDF approach.

TABLE II. EXPERIMENTAL RESULTS BY EVENT-TYPE

Event type	Correct TF-IDF	Incorrect TF-IDF	Correct Helmholtz	Incorrect Helmholtz
Start-up	37	2	38	1
Shutdown	23	16	38	1
rectangle	11	16	11	6
round rectangle	6	11	4	13
polygon	20	0	15	5
triangle	3	15	2	16
ellipse	6	11	5	12
diamond	2	14	2	14

TABLE III. PRECISION, RECALL AND ACCURACY (HELMHOTLZ)

Event type	Precision	Recall	Accuracy
Start-up	100%	97.4%	99.5%
Shutdown	88.4%	97.4%	96.7%
rectangle	47.8%	64.7%	90.2%
round rectangle	20.0%	23.5%	84.2
polygon	60.0%	75.0%	91.8%
triangle	50.0%	11.1%	90.2%
ellipse	27.8%	29.4%	86.3%
diamond	28.6%	12.5%	89.6%
Average	52%	51%	90%

TABLE IV. PRECISION, RECALL AND ACCURACY(TF-IDF)

Event type	Precision	Recall	Accuracy
Start-up	100%	94.9%	98.9%
Shutdown	92.0%	59.0%	90.2%
rectangle	52.4%	64.7%	91.3%
round rectangle	42.9%	35.3%	89.6
polygon	41.7%	100.0%	84.7%
triangle	27.3%	16.7%	87.4%
ellipse	37.5%	35.3%	88.5%
diamond	19.2%	12.5%	87.4%
Average	51%	51%	89%

A. *Causes of misidentification*

As discussed above, some segments, namely triangle, rectangle and round rectangle, were identified at lower than the average rate. A possible method to improve the detection in this case is to increase the value of ε, thus producing a larger set of key-events.

By manual inspection of the data, we have found that a major source of miss-assignation was a flaw in the program segmentation phase of the algorithm. We had assumed that the segmentation phase would divide the trace in a manner that roughly matches the beginning and ending of each of the program actions we had performed. While this was largely the case, the segmentation algorithm sometimes suggested very short sequences of a few function calls as distinct segments. In fact, some of these segments were so short that their key-element summaries were empty. Naturally, very short segments cannot be matched. When such aberrant segments are taken out of the analysis, the rate of correct attribution increases to 77% for Helmholtz and 80% for TF-IDF.

V. RELATED STUDIES

Several studies have sought to identify the most important elements of a trace, in an attempt to solve the feature location problem. A survey of these is presented in Dit et al. [14]. Most of these techniques, however, are not designed to recover the key elements of each execution phase of the traced scenario. They look at the trace as a whole and not as a sequence of different phases.

Poshyvanyk et al. [15] introduced an information-retrieval-based technique for feature location, which was later followed by many other studies such as the work of Asadi et al. [16]. Their approach combines source code information and trace content to identify parts of the trace that are most relevant to the implementation of the traced scenario. Our approach uses Gestalt theory to identify the significance of the trace elements in each phase. We also do not rely upon source code to keep our approach lightweight.

Greevy et al. [17] introduced a compact feature-driven approach to investigate the relationship between features and classes. Medini et al. proposed a concept location technique that relies on trace segments [18]. In [7], Medini et al., proposed SCAN (Segment Concept AssigNer) an approach to assign labels to sequences of methods in execution traces and to identify relations among execution traces segments. (FCA). Pirzadeh et al. [19] presented a phase detection approach in which a trace is divided into phases by measuring the distance between calls to the same methods.

Other trace segmentation approaches include that of Watanabe et al. [20] which use a LRU cache for phase detection. In [21], the authors proposed an analogy between signal processing and dynamic analysis to identify similar phases. Neither of these approaches extract key information about phases. Visualization techniques [22], [1], [23] have also been adapted by some researchers to reduce the vast amounts of trace information.

VI. Conclusion

In this study we present an approach to identify key elements of a trace using Gestalt laws and the Helmholtz Principle. We show that this method is effective for identifying the main elements of behaviour imbedded in large execution traces, a development that has multiple applications in software maintenance and reverse engineering.

There are many ways to improve this method. As previously mentioned, the similarity between segments is computed in this study by a simple function involving the intersection of the sets of key elements in both segments. More elaborate methods (i.e., involving patterns learned from data mining) could improve the efficiency of the approach. Furthermore, data about the possible ordering of program phases relative to one another discovered through this algorithms could be used to refine the segment assignation process. For instance, in our example, it would quickly become obvious that every segment always begins with a start-up segment, and ends with a shutdown segment.

References

[1] D. Jerding and J. Stasko, "The information mural: a technique for displaying and navigating large information spaces," *IEEE Trans. on Visualization and Computer Graphics*, vol. 4, no. 3, pp. 257–271, Jul 1998.

[2] D. Becker, F. Wolf, W. Frings, M. Geimer, B. Wylie, and B. Mohr, "Automatic trace-based performance analysis of metacomputing applications," in *Parallel and Distributed Processing Symposium, 2007*, March 2007, pp. 1–10.

[3] T. Systa, "Understanding the behavior of java programs," in *7th Working Conference on Reverse Engineering*, 2000, pp. 214–223.

[4] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, 2009.

[5] H. Pirzadeh and A. Hamou-Lhadj, "A novel approach based on gestalt psychology for abstracting the content of large execution traces for program comprehension," *16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, pp. 221–230, 2011.

[6] S. Medini, V. Arnaoudova, M. D. Penta, G. Antoniol, Y. Guéhéneuc, and P. Tonella, "SCAN: an approach to label and relate execution trace segments," *Journal of Software: Evolution and Process*, vol. 26, no. 11, pp. 962–995, 2014.

[7] S. Medini, G. Antoniol, Y. Guéhéneuc, M. D. Penta, and P. Tonella, "SCAN: an approach to label and relate execution trace segments," in *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012*, 2012, pp. 135–144.

[8] O. Benomar, H. A. Sahraoui, and P. Poulin, "Detecting program execution phases using heuristic search," in *Search-Based Software Engineering - 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings*, 2014, pp. 16–30.

[9] H. Pirzadeh and A. Hamou-Lhadj, "A software behaviour analysis framework based on the human perception system," in *In Proc. of the 33rd International Conference on Software Engineering (ICSE'12 NIER Track)*, 2011, pp. 948–951.

[10] H. Pirzadeh, A. Hamou-Lhadj, and M. Shah, "Exploiting text mining techniques in the analysis of execution traces," *IEEE International Conference on Software Maintenance, ICSM*, pp. 223–232, 2011.

[11] K. Koffka, *Principles of gestalt psychology*. Harcourt, 1935.

[12] B. Dadachev, A. Balinsky, H. Balinsky, and S. J. Simske, "On the helmholtz principle for data mining," in *2012 Third International Conference on Emerging Security Technologies, Lisbon, Portugal, September 5-7, 2012*, 2012, pp. 99–102.

[13] J.-M. M. Agnes Desolneux, Lionel Moisan Jean-Michel, *From Gestalt Theory to Image Analysis*, 2006, vol. 34, no. July.

[14] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[15] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 420–432, June 2007.

[16] F. Asadi, G. Antoniol, and Y. Gueheneuc, "Concept location with genetic algorithms: A comparison of four distributed architectures," in *2nd International Symposium on Search Based Software Engineering*, Sept 2010, pp. 153–162.

[17] O. Greevy and S. Ducasse, "Correlating features and code using a compact two-sided trace analysis approach," in *9th European Conf. on Software Maintenance and Reengineering*, March 2005, pp. 314–323.

[18] S. Medini, P. Galinier, M. D. Penta, Y. Guéhéneuc, and G. Antoniol, "A fast algorithm to locate concepts in execution traces," in *Search Based Software Engineering - Third International Symposium, SSBSE 2011, Szeged, Hungary*, 2011, pp. 252–266.

[19] H. Pirzadeh, A. Agarwal, and A. Hamou-Lhadj, "An approach for detecting execution phases of a system for the purpose of program comprehension," in *Proc. of the 8th International Conference on Software Engineering Research, Management & Applications, Montreal, Canada*, 2010, pp. 207–214.

[20] Y. Watanabe, T. Ishio, and K. Inoue, "Feature-level phase detection for execution trace using object cache," in *Proceedings of the 2008 International Workshop on Dynamic Analysis*.

[21] A. Kuhn and O. Greevy, "Exploiting the analogy between traces and signal processing," in *22nd IEEE International Conference on Software Maintenance, 2006. ICSM '06*, Sept 2006, pp. 320–329.

[22] S. P. Reiss, "Visual representations of executing programs," *J. Vis. Lang. Comput.*, vol. 18, no. 2, pp. 126–148, Apr. 2007.

[23] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. van Wijk, and A. van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *15th IEEE International Conference on Program Comprehension (ICPC '07)*, June 2007, pp. 49–58.