Mapping of Concepts in Program Comprehension

Anas Ali

Department of Computer Science & IT.
University of Lahore, Pakistan

Ahmad Salman Khan

Department of Software Engineering.
University of Lahore, Pakistan

Summary
Software system's understanding is the essential part of the software maintenance. Documentation and program source code could be the first aspect in software system's understanding and its inner work-flow. This understanding of code is known as Program Comprehension (PC). With the advancement in technology and rapid growth in industry, information systems require frequent changes. This change need to be implement a specific idea or behavior which is named as "Concept". Concept (variable, class, method) is referred as a typical behavior of code performing specific functionality like a student enrollment in student portal. In large scale systems, to locate the concept in the code is a very challenging task which require massive amount of time if performed in conventional ways. The conventional ways mean start finding the location by going through each and every file of the code. Programmer tries to locate these concepts with the help of domain knowledge using concept location techniques like static, dynamic and textual concept location technique.
In this paper, we focused on static concept location technique and proposed a hybrid model for concept location in program comprehension.

Key words:
Program comprehension PC, Concept location CL, Concept location technique CLT, Cognitive model CM.

1. Introduction

Software system undergoes many changes for correction, updation and enhancements. The rapid changes in code make it difficult for maintainer to understand the program. . Software quality effects when a system comes into maintenance phase. Almost 58% effort of developer spent on this task in the software maintenance process [1]. Understanding the existing system is the first step in the maintenance phase. This understanding of the system is known as "program comprehension" (PC). Program comprehension is one of the basic pillar of software engineering which is aimed to give techniques, methods, strategies and processes those facilities developer in clear understanding of the system. Many researchers work on this topic for decreasing maintenance cost by reducing the time of understanding. According to different researches, program comprehension is a time-consuming task [1, 2].
Comprehension is necessary for change in software without comprehension no change can be performed [3]. In the past few years, researchers have paid more attention towards program comprehension. In result, two classic theories have been presented with respect to program comprehension named as Top-down and Bottom-up theories [4]. In top-down theory a programmer creates particular hypothesis, its acceptance and rejection are based on bacon which acts as evidence derived from its code. The false hypothesis is declined while the true hypothesis becomes a part of program comprehension. Moreover, top-down theory is used in the early stages of comprehension [5]. Chunking (collection of code statements into a code block) is considered as the base for program comprehension in a bottom-up theory [3]. In this, the programmer splits code into different chunks and combine these chunks to get high-level knowledge of the Code, which leads to a systematic view instead of a localized view of the program. However, the bottom up approach is used in maintenance purpose. Furthermore, the combination of both approaches leads to integrated model [6].
In program comprehension the location of a specific idea (method, function, data member, class or an object performing some activity) is known as "Concept". Sometimes this term is also refereed as a feature. Concept or feature is a high level term, used to address a specific functionality of the system. For instance, the module about payment method in a large scale application is referred as "payment concept". Comprehension is not limited to top down and bottom approach, concepts play a significant role in program comprehension. In software maintenance and evolution, change request occurs when we have to add or modify a specific concept in the program. Concept location is the process in which software engineer tries to locate specific concept (feature) according to their domain knowledge. However, as "needed strategy" is used to understand the concept reflected from the code [7, 8]. The size of software causes an increase in complexity of concept location. Concept location technique goal is to identify the concept by analyzing the code. Different techniques are defined for such purposes. Some techniques for concept analysis, such as static analysis, dynamic analysis, and textual analysis [9]. Dynamic analysis is runtime analysis in which they perform analysis on execution trace. Static analysis is pre-execution analysis in which focus is on code structure, dependencies, and control flow. Textual analysis is basically string matching

Manuscript received May 5, 2019
Manuscript revised May 20, 2019

pattern in which programmer provide a string to match a string or to locate concept name [9].
In this paper, we focused on static analysis in concept location to comprehend a program, because we are more interested in pre execution activity. We proposed a model to map the location of concept in large scale applications. Our focus is how individual programmer performs a comprehension with help of static concept analysis. Our goal is to capture programmer's activities and the process to map a concept in program comprehension.
This paper further categorized into four parts. In section 2 we discussed our research method, section 3 is about related work in program comprehension and concept location in program comprehension. In section 4, we discussed our proposed model as in result and section 5 covers the conclusion of out paper.

2. Research Method

Systematic Literature Review (SLR) methodology has been implemented for taxonomy of program comprehension and concept mapping or locating approaches in program comprehension. In [10] presented classification of program comprehension and a taxonomy of concept location presented in [11]. We use this SLR methodology to find the common factors, which use to locate the concept in program comprehension. Furthermore, in this study we are going to develop a model using literature about comprehension techniques and concept location methods.
SLR is a process of critically documenting each step used in research [12]. The aim of documenting each process is to keep track of each and every step that could be easily traceable to others researchers. SLR methodology suggests a roadmap towards interested research area and related questions by examining, exploring and classifying the present literature according to the domain [13]. According to Kitchenham [12] [13] SLR classified into three levels (i) Planning, (ii) Conducting and (iii) reporting the review. We followed Kitchenham et al methodology in this paper. Moreover, we followed three basic level of SLR described in [3] [4]. In figure 1 we have shown the overall process of SLR.

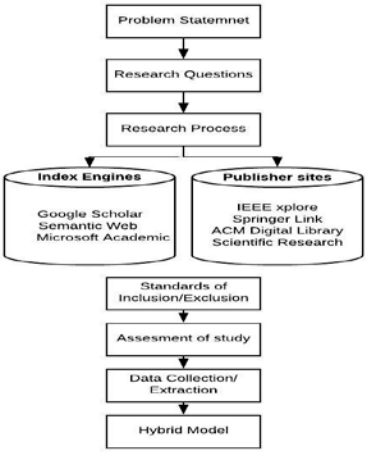

Fig. 1 SLR Process

2.1 Research Problem

Software engineers are the most valuable resource of the any software organization. Almost 58% time of the software engineer spent their time on comprehending the code or understanding the code [14] [15] [16]. Although many researchers introduced techniques, model and strategies for code comprehension but they are effective to small scale system. In large scale system, we have to implement the related concept in code [3]. The aim of our study is to find the factors in the present literature for locating the concept in program comprehension and provide a concept location model to reduce the comprehension time.

2.2 Research Question

RQ1: What type of methods for static analysis are used for locating the concepts?
RQ2: What are the common factors used in static concept location techniques?

2.3 Research Process

In this SLR, we chosen Electronic Data Sources (EDS) according to our previous research experience and we followed suggestion of Chen at al provided in [17]. In this research, we chosen index engine and publisher's site as databases. In publisher's site we get publication of their own database, whereas in index engine databases contains citation and other cited data which is published by other publishing websites. In Table 1 we presented list of selected databases.

Table 1: EDS

Publisher Indexing	Index Search Engine
IEEE xplore	Google Scholar
ACM Digital Library	Semantic web
Springer Link	Microsoft Academia
Scientific Research	

We used different search strings in databases mentioned in table 1. These search strings are constructed on major keywords and related words to research questions. These keywords are preferred on the base of existing literature in context of program comprehension, concept location and static analysis in concept location. In table 2 we have shown the major keywords.

We used 'OR' and 'AND' operators for concatenated the alternative words and the main keywords to generate valuable search strings shown in table 2. These search strings used in selected databases: ('Program comprehension' OR 'Code comprehension' OR 'Program understanding' OR 'System understanding' OR 'Code understanding') AND ('Program comprehension strategy' OR 'Program comprehension approach' OR 'Program comprehension techniques' OR 'The code comprehension strategy' OR 'Strategy of understanding the code) AND ('Program comprehension model' OR 'The code comprehension model' OR ' Code understanding model' OR 'Program understanding model') AND ('Concept Location in Program Comprehension' OR 'Concept location in code understanding' OR 'Formation of concept in program comprehension' OR 'Feature location in program comprehension') AND ('Static Concept analysis in Program Comprehension' OR 'Static feature location in Program comprehension' OR 'Static concept location in code comprehension'.

Table 2: Key Strings and Alternatives

Main Keywords	Alternative Keywords
Program Comprehension	Program understanding, The code comprehension, The code understanding, System understanding,
Program Comprehension Strategy	Program comprehension approach, The code Comprehension strategy, Strategy of understanding the code
Program Comprehension Model	Code comprehension model, code understanding model, program understanding model
Concept Location in Program Comprehension	Concept location in code understanding, Formation of concept in program comprehension, Feature location in program comprehension
Static Concept analysis in Program Comprehension	Static feature location in Program comprehension, Static concept location in code comprehension

2.4 Selection of Publication

In below sections we provide the detail of Inclusion and exclusion standards

Inclusion Standards: We considered those studies which emphasis on activities performed in program comprehension and locating the concepts in program comprehension. These studies must be from reputed journals, workshops, conferences and related books. High weightage is given to the articles those having empirical studies with case studies. Major selection has been made on the basis of relevance to the selected domain and work provided in this domain.

Exclusion Standards: Those article which did not clearly discussed program comprehension factors, classifications, techniques, models and concepts importance in program comprehension have been excluded. Additional exclusions made on the base of static concept locating approaches, those approaches which are not covering the static analysis in program comprehension. We exclude the articles those are duplicate because of different electronic source databases.

Primary Selection: In the primary selection process we found many research articles, we applied tollgate technique studied in [18] and refined our selection process shown in figure 2. The tollgate technique comprised on five levels explained below.

In first level, we collected total 195 articles from different search engines. These article were selected on the base of mentioned inclusion standards.

In second level, we removed duplicate papers collected from different search engines. We made selection of 130 papers after exclusion.

In third level, selection was made upon the title of the article and the abstract relevant to the keywords used to search the paper according to our research questions. We sort out 72 paper after implementation of this level.

In fourth level, we read 'introduction and conclusion' of the selected articles and made further exclusion or inclusion. The articles those have empirical studies relevant to program comprehension in the context of concepts mapping selected into our study. After implementation of this level, 55 articles preceded for the final level.

In the final level, selection of the articles were made upon full text reading. Articles primarily related to program comprehension or concept mapping (location) in program comprehension were included in this level. However, we selected 30 articles for primary study in our research.

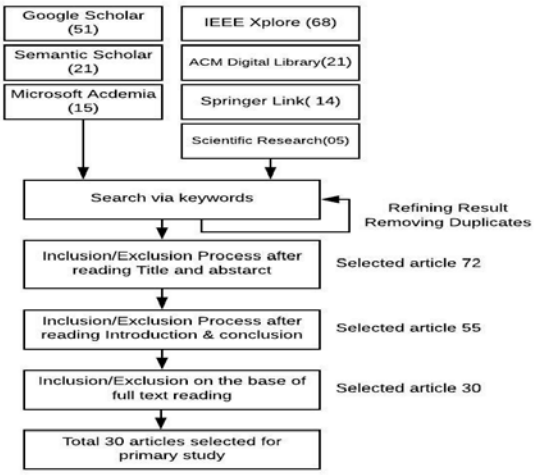

Fig. 2 Primary Article Selection Process

3. Literature Review

As technology growing with the passage of time, new technologies are emerging and more complex and comprehensive systems are being developed. When such comprehensive and large systems are developed, their maintenance becomes a challenging and critical task. It is very difficult for a system designer to understand the previous version of the code without having a useful method of system understanding. Program comprehension means understanding the existing system (code and documentation) [19]. For this purpose, many researchers tried to contribute some approaches for better understanding of the system. In [10] program comprehension is classified into different categories,
(i)Cognitive model (ii) Software visualization (iii) Information Extraction.
In this section, we describe these categories in detail and presents the comprehension methods used for concept location with a special focus on static analysis.

3.1 Cognitive Model

In Cognitive Model (CM), programmers understand the code in the context of program comprehension. It is an important task because it allows different mental process, techniques, strategies, and concepts that a programmer use to understand the program.
Concept and terminology: In [6, 10], Store et al discussed about cognitive model and its impact. He also discussed terminologies of program comprehension shown in figure 2.1. Like, Mental model is the developers mental illustration of the program that how he understands the system by utilizing temporary information and reasoning [10]. Furthermore, a piece of code, which shows a typical behavior in programming known as programming plans. Like, in sorting code we use a loop for comparing two numbers in every iteration [10, 20]. According to brooks [20, 21] bacons are familiar, recognizable point in the code that represent a specific structure. Like, function name can represent the implementation in the code. Moreover, the programming style name convention could be used as bacon.
Top down approach: Top down comprehension in which developers create a hypothesis and then look into code and try to understand the code. Developer search for bacons in top down manners. Verification or rejection of this hypothesis dependent on the existence of bacons [3, 6, 20, 22].

Bottom-up approach: In Bottom-up comprehension developer firstly read the code statement by statement and check the control flow of the code to form different chunks (referred as a code block) of high level abstraction. Combination of these chunks helps in understanding of full system [3, 6, 20, 23].
Integrated Model: The Integrated model is consists on the combination of top-down and bottom-up [20]. This model also represents the program model and the situational model. Program model refers to chunking the code and situational model are developed with the abstract level of the functions. In [6, 22] Mayrhauser et al discussed integrated meta model, in which he explained integrated meta model consist on four steps; situational model, program model, top-down domain model, and knowledge base. First, three steps used to create a mental model and knowledge base used for code comprehension.
Knowledge base: A knowledge base model is dependent upon programmer's personal knowledge and domain knowledge related to the system [6, 23]. This knowledge helps him to make an understanding of the system and to create a tool and method for better comprehension. Moreover, some comprehension strategies discussed. Which are (i) Browsing support (ii) Search Strategy (ii) multiple views [6].
• **Browsing support** helps the developer in top down approach to navigate from higher level to lower level with the help of bacons [3, 6, 20] for example scrolling up and down to find a specific code.
• **Searching strategy** is a tool support which helps developer in iterative code search [3, 6]. They search the code with their knowledge and experience with the help of this tool. For example search through keywords, these keywords are generated through their knowledge.
• **Multiple aspects:** In this strategy programmer have different aspects. For example, one aspect for programming goal (with the intentions to add, or modify

code) and other is to understand its relationships (e.g. 10 quality assurance dept. to understand the core functionalists of the system) [6].

Systematic and As needed strategy: A systematic strategy is about the understanding of whole system during maintenance. Aim of this strategy is to get design of program created by real programmer which helps them in maintenance (modification or bug correction) of the system. Whereas, as needed strategy is used at the current situation in which a developer understood specific chunk of code which is going to be modified [5, 25].

3.2 Software Visualization

In the process of software visualization, a programmer analyzes the system properties e.g. the architecture of its source code or metrics of their runtime behavior. Visualization can be classified into three groups as; (i) Algorithm visualization (ii) Program Visualization (iii) system Visualization [10].

Algorithm visualization: In this type of visualization main goal is to teach data structure and algorithms This software visualization (SV) method concerned with the student, this model is used in a learning process [10].

Program visualization: The aim of this method is to realize the functionality of program and analyze the relation in its component e.g. classes, hierarchy (inherit ice, polymorphous, and composition), objects, methods performing specific actions, and data members. The main concern of this method is to study the behavior of the program [10, 26].

System visualization: This method is concerned to study large program which consists on different modules. Therefore, visualize technique will be applied on every module to understand the system. However, algorithm visualization and program visualization can be used in this method [10].

3.3 Information Extraction

In this Information Extraction, a technique code instrumentation is used to extract system information [10]. In this technique, the useful statements are inserted into source code e.g. checkpoints, or temporary variables. Start and end of the function are used as a checkpoint of the code and these checkpoints are used as an Inspection function. These inspection function prints parameters, names and function name which can be used by a programmer to form a strategy for the system.

Concept Location in Program Comprehension

Concept location is the process of implementing the software requirements into source code which maps those requirement [28]. In program comprehension, typically programmer has knowledge about domain concepts. He understands them very well but unaware of their presence in the program code. Fetching knowledge about domain concepts is easier than code knowledge because the use of program gives a lot of knowledge to comprehend domain concepts [27]. Moreover, user manual also provides information for domain concepts. The aim of gathering domain knowledge is the correct implementation of concepts in the code. All relevant concepts should be implemented on the piece of code or different piece of code. The concept location process helps to find out the formulation of these relevant concepts in the code [3, 27] According to object-oriented programming, the object is referred as a concept. Each class in object oriented programming mentions its concept e.g. salary or bonus. Many classes represent different concepts or may be distinct from each other. Like this, many programmer uses design patterns in which different classes collaborates to implement a single concept [29].This kind of concept needs to be identified in the source code and we need to change this unique concept because we have to make a change in all the classes those have an impact on this concept. Furthermore, concept location process is highly dependent upon programmer's personal skills. However, there are multiple ways to locate concepts (feature) in the program. We discussed concept location techniques and their use how and where these techniques are implemented. Many researchers [3, 29–31] have work into this for making it more comprehend. These techniques are (i) Dynamic concept location (ii) Static concept Location (iii) Textual concept Location [3, 29, 30]. In this literature we focused on static concept location techniques.

Static Concept Location

Static concept location approaches do not need any information about the execution of software system. In this technique, developer statically analyzed the source code. However, its structure and dependencies can be explored automatically or manually [3]. Some other static concept location techniques (CLT) are involved in data dependencies and different control 13 types. Whereas, other static concept location techniques use structural dependencies of software system [30]. Furthermore, static CLTs is not limited to dependency graph but also on software artifacts set e.g. developers feedback and historical information [30, 31]. These set of artifacts works as an initial point which helps in the analysis to obtain program element related to initial set [30]. However, the developer specified these initial artifact set.

In [32] chen et al presented a method named as Abstract System Dependence Graph (ASDG). Abstraction of the system dependence graph (SDG) performed in this method. Global variables or function used as a node in ASDG and control dependencies between different function or data

flow among variables used as the edges between different nodes [32]. The static CLTs which implements ASDG's needs statically developed a dependence graph as a starting node and input as well [30, 32]. However, this input is chosen by the developer. A program element which is relevant to the concept could be the starting node, which might be known by the programmer or he selected any node randomly as a starting node or it can be related to the main method. Developers feedback required in each step of CLTs. Like, the selected node is relevant to the concept or irrelevant. Moreover, CLTs keep track of visited node and search gaps on the basis of developer's feedback and try to fill the search gaps and try to keep the relevant nodes. This process remains to continue till all the program element relevant to the maintenance find by the developer. This technique supported by a tool named as Ripples [30]. These ripples help in generating ASDG from the code of programming language (C) and helps the developer to visualize the graph and select the node which is relevant to the concept.

Another method named as Concern Graph Representation.

Concern graph representation is developed by Robillard et al. this graph representation allows an abstract image of a concept or concern which helps in creating mapping and it's storing between source code and feature. Program elements subsets and relation between these sets are encapsulated in the concern graph representation. Static dependencies among the program elements are the base of this relation. The supporting tool of concern graph representation is named as Feature Exploration and Analysis Tool (FEAT) [33] . However, through this tool developers can visualize the concern graph, analyze the course code and its relation to the graphs program elements and give permission to the developers to change the concern graph.

In [34] Saul et al presented an approach name as Finding with Random walks (FRAN). This approach suggests relevant program elements on the basis of some input as a starting point. FRAN uses systems structural information and simplifies Robillards method [34]

. It take an input (Program 14 element e) of developer's interest and build a graph of program which has dependency to the neighbor of e. Moreover, this approach uses large number of correlated program elements and uses Kleinberg et al [30] algorithm on the dependency graph for ranking program elements.

Trifu el al [35] presented a feature location technique which is based on static analysis of data. This technique takes many variables as an input known as information sinks. However, these variables are chosen by the developers and utilized as initial points to recognize the all chunks of the code where variables values propagate. Dependencies in data flow are used to track them into code [30].

Trifu et al used concept information base to improve this approach named as data flow based concern identification. However, this conceptual information defines the limitation of the concern.

Zhao et al [36] introduced an approach named as a static, non-interactive approach to feature location (SNIAFL). This approach uses combination of information retrieval (IR) and branch reserving call graph (BRCG). Moreover, this approach uses branch information with call graph expanded version. Information retrieval is used to locate initial program elements to locate a specific feature and BRCG is used to find other related elements.

In [37] Ratiu et al presented an approach, which recovers an abstract picture of relevant chunks of the code and real world concepts. Although, this approach not exactly for concept location but it helps to locate the concept with the help of developers point of view according to their domain knowledge [30, 37]. Moreover, they introduced a schema that discussed logical error generated by inappropriate variable naming style and developed an algorithm that used to retrieve the concepts between program elements and ontology elements. This algorithm exploits graph matching for mapping program elements and concepts. However, concepts are formulated in ontology and graph are used for program abstraction.

In table 3, we tried to compare some of above static concept location techniques and shown their detail with respect to different factors.

Table 3: Comparison of Static Concept Location Techniques

Static Analysis	**Input type**		**Programming Language**		**Data Sources**		**Output**	
Papers	Query	Program element	JAVA	C++	Dependency Graph	Compilable Source Code	Class	Method
K.Chen [32]	✕	✓	✕	✓	✓	✓	✕	✓
Robillard [33]	✕	✓	✓	✕	✓	✓	✓	✓
M.Saul [34]	✕	✓	✕	✓	✓	✓	✕	✓
Trifu [35]	✕	✓	✓	✕	✓	✕	✕	✓
Zhao [36]	✓	✓	✕	✓	✓	✕	✕	✓

This table addressed our first research question answer and conclude the detail comparison of the static concept location techniques. Moreover, with this literature we identify some factors which used in static analysis for concept location (RQ2). These factors are listed below.

- Dependencies (Abstract system dependencies)
- Ranking (Concern Graph)
- Point of view (Developer understanding view)
- Intention (maintenance or development)
- Concept name (domain knowledge specific or experience)

4. Result

The studied literature lead towards a preliminary model for CL with the help of static analysis which shown an activity diagram in Figure 3. We proposed a hybrid model for concept location. In this model programmer starts the activity and perform these step respectively (i)Cognitive Model (ii) Create the chunk of code (iii)Categorize the

Chunk (iv) Comprehend the code (v) Formulate the Concept.

• **Cognitive Model** In this activity, programmer start with a cognitive model e.g. mental model [6, 11]. This model helps the programmers to understand the code in context of program comprehension. It is an important task because it allows different mental process, techniques, strategies and concepts that a programmer use to understand the program.
• **Create the Chunk of Code** In this activity, programmer uses bottom up approach and perform action of reading the code statement by statement in order to obtain group of code or chunks of code as an output [3, 6, 10, 12, 23].
• **Categorize the Chunk** In this activity, programmer has many different chunks .These chunks categorized into a group on the basis of relationships and dependencies between them [1, 3, 5, 6, 10]. These dependencies and relationship is given as input to categorized the chunk activity to get high level abstraction and a comprehended code as an output. After this step, static analysis will be performed on comprehended code. In this analysis programmer gives initial input and checks data sources to get output as an program element(Class, function) [23].
• **Concept Formulation** In this activity, we have a comprehended code in which static analysis is already performed. Programmer will investigate and formulate the needed concept. After formulation of concept, this activity will be ended

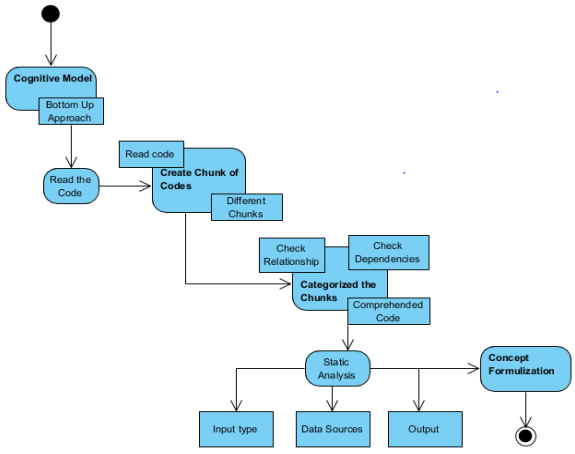

Fig. 3 Activity Diagram of Hybrid Model for CL

5. Conclusion

Concept location is an important task in program comprehension. Many techniques and methods are proposed in this context. But still there is no existing model for concept location. We proposed a hybrid model concept location model by combining the approaches and implementing the basic factor of static analysis in concept location model. In future, we will validate our model through case study and industrial survey. We will try to improve our model with the help of developer's feedback.

References

[1] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," IEEE Transactions on Software Engineering, vol. 44, no. 10, pp. 951– 976, 2018.
[2] I. Schr¨oter, J. Kr¨uger, J. Siegmund, and T. Leich, "Comprehending studies on program comprehension," in Program Comprehension (ICPC), 2017 IEEE/ACM 25th International Conference on. IEEE, 2017, pp. 308–311.
[3] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in Program Comprehension, 2002. Proceedings. 10th International Workshop on. IEEE, 2002, pp. 271–278.
[4] J. Siegmund, "Program comprehension: Past, present, and future," in 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 5. IEEE, 2016, pp. 13–20.
[5] A. Karahasanovi´c, A. K. Levine, and R. Thomas, "Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study," Journal of Systems and Software, vol. 80, no. 9, pp. 1541–1559, 2007.
[6] M.-A. Storey, "Theories, tools and research methods in program comprehension: past, present and future," Software Quality Journal, vol. 14, no. 3, pp. 187–208, 2006.
[7] N. Alhindawi, J. Alsakran, A. Rodan, and H. Faris, "A survey of concepts location enhancement for program comprehension and maintenance," 2014.
[8] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev, "Static techniques for concept location in object-oriented code," in Pro28 BIBLIOGRAPHY 29-gram Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on. IEEE, 2005, pp. 33–42.
[9] M. Revelle and D. Poshyvanyk, "An exploratory study on assessing feature location techniques," in Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. IEEE, 2009, pp. 218–222.
[10] M. H. P. P. M. J. &. U. R. Berón, "Program inspection to interconnect behavioral and operational view for program comprehension," 2007.
[11] B. R. M. G. M. &. P. D. Dit, "Feature location in source code: a taxonomy and survey.," Journal of software: Evolution and Process, vol. 25(1), pp. 53-95, 2013.
[12] B. B. O. P. B. D. T. M. B. J. &. L. S. Kitchenham, "Systematic literature reviews in software engineering–a systematic literature review.," Information and software technology, vol. 51(1), pp. 7-15, 2009.

[13] B. Kitchenham, "Procedures for performing systematic," Keele, UK, Keele University, vol. 33(2004), pp. 1-26, 2004.
[14] B. Z. A. V. D. A. M. L. &. K. R. Cornelissen, "A systematic survey of program comprehension through dynamic analysis," IEEE Transactions on Software Engineering, vol. 35(5), pp. 684-702., 2009.
[15] S. M. Dekleva, "The influence of the information systems development approach on maintenance," MIS quarterly, , pp. 355-372, 1992.
[16] R. T. K. WALID MAALEJ, "On the Comprehension of Program Comprehension," ACM Transactions on Embedded Computing Systems, vol. 9(4), March 2014.
[17] L. A. B. M. &. Z. H. Chen, "Towards an evidence-based understanding of electronic data sources," 2010.
[18] A. A. &. K. J. Khan, "Systematic review of success factors and barriers for software process improvement in global software development," IET Software, vol. 10(5), no. IET, pp. 125-135, 2016.
[19] S. Letovsky, "Cognitive processes in program comprehension," Journal of Systems and software, vol. 7, no. 4, pp. 325–339, 1987.
[20] M. P. Obrien, "Software comprehension–a review & research direction," Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report, 2003.
[21] R. Brooks, "Towards a theory of the comprehension of computer programs," International journal of man-machine studies, vol. 18, no. 6, pp. 543–554, 1983
[22] A. Von Mayrhauser and A. M. Vans, "Program comprehension during software maintenance and evolution," Computer, no. 8, pp. 44–55, 1995.
[23] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," International Journal of Computer & Information Sciences, vol. 8, no. 3, pp. 219– 238, 1979.
[24] S. Letovsky and E. Soloway, "Delocalized plans and program comprehension," IEEE Software, vol. 3, no. 3, p. 41, 1986.
[25] J. Koenemann and S. P. Robertson, "Expert problem solving strategies for program comprehension," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 1991, pp. 125–130.
[26] M. Ben-Ari, "Program visualization in theory and practice," 2001.
[27] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in Reverse Engineering, 2004. Proceedings. 11th Working Conference on. IEEE, 2004, pp. 214–223.
[28] A. Armaly, J. Klaczynski, and C. McMillan, "A case study of automated feature location techniques for industrial cost estimation," in 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2016, pp. 553–562.
[29] E. Gamma, Design patterns: elements of reusable object-oriented software. Pearson Education India, 1995.
[30] 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.
[31] H. E. Salman, A.-D. Seriai, and M. Hammad, "Quality-driven feature identification and documentation from source code," Journal of Theoretical and Applied Information Technology, vol. 84, no. 2, pp. 183–195, 2016.
[32] K. Chen and V. Rajlich, "Case study of feature location using dependence graph," in Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on. IEEE, 2000, pp. 241–247.
[33] M. P. Robillard and G. C. Murphy, "Concern graphs: finding and describing concerns using structural program dependencies," in Proceedings of the 24th international conference on Software engineering. ACM, 2002, pp. 406–416.
[34] Z. M. Saul, V. Filkov, P. Devanbu, and C. Bird, "Recommending random walks," in Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2007, pp. 15–24.
[35] M. Trifu, "Using dataflow information for concern identification in object-oriented software systems," in Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on. IEEE, 2008, pp. 193–202.
[36] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "Sniafl: Towards a static noninteractive approach to feature location," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 15, no. 2, pp. 195–226, 2006
[37] D. Ratiu and F. Deissenboeck, "From reality to programs and (not quite) back again," in Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on. IEEE, 2007, pp. 91–102.