International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 9 • Number 45 • 2016

Review- Program Comprehension

Raj Singh[a] B.D. Mazumdar[b] and A.K. Vyas[c]

[a]*Jodhpur National University Jodhpur (Raj.) Lovely Professional University*

E-mail: er.rajsingh@gmail.com

[b]*Department of Computer Applications School of Management Studies Varanasi, U.P*

[c]*Department of Mathematics,Faculty of Engineering and Technology Jodhpur National University*

Abstract: Program appreciation research has been described by both the speculations that give rich clarifications about how developers understand programming and additionally the apparatuses that are utilized to help with perception assignments. A Cognitive Model portrays the subjective procedures and interim data structures in software engineer's head .We have audited a percentage of the key psychological hypotheses of system perception that have developed in project appreciation. Utilizing these speculations we investigate what number of apparatuses that is well known to bolster program appreciation. In particular, we have examined how the subjective hypotheses and supporting devices are connected and think about the exploration strategies that were utilized to build the speculations and assess the devices. The checked on speculations and devices are further separated by qualities, program attributes, and the setting for the different perception errands. At long last, we anticipate how these subjective components will influence in the project appreciation devices and techniques. The software maintenance task is very time consuming and tedious job. The industries spend 60-70 % of the time in maintenance. We have proposed the cognitive model, which can help in reducing the cost program comprehension during software maintenance.

Keywords: *Software maintenance, program comprehension, cognitive models, agents, program comprehension approaches.*

1. INTRODUCTION

Today major amount of programming work is accomplished on sophisticated software applications which we called Integrated Development Environment (IDE). IDE are commonly favored by programmers because of Rapid Application Development (RAD). It provides programmers some special tools like; Source Code Editor, Build Tools, Debugger, Compiler or Interpreter, Version Control System etc.

These functionalities present more than one perspectives of the same program, which is in development process. These representation forms are known as program visualizations. It provides programmers not to treat programs as Code Text produced as Program Entities, Which are executed in conditions. Program visualizations are presented either in textual or, graphical form and presents different information about the program *e.g.* If there is simultaneous use of both Unified Modeling Language (UML) diagram and Flow control diagram to

tackle different perspectives of single software project. These visualizations are used by the programmer to debug a program. Different programmers use these functionalities (Tools) according to their interest, which depends on factors like:-Programming language expertise adjustment with the IDE and personal preferences. It means that effective usage of visualizations depends over the skill of a programmer. These skills are in generating and testing hypothesis from the program output and visualization. Novice programmers having no knowledge of IDE faces problem of understanding and using IDE in skilled way. It is necessary to develop a platform and training process for guiding these novice programmers. In case of program comprehension the main emphasis is on understanding the programs written by others. Majority of program or, code comprehension research is focused on capturing the logical (thinking) ways of programming through comprehension models, instead of Eye Tracking Methodologies or, Models. Recently researches are mainly focused on Visual Attention Tool, which is called **Restricted Focus Viewer (RFV).** It may be called Eye Tracker. For this purpose researchers are working on studying the psychology of the programmers.

The main focus of the present research is centralizing on investigation through theoretical hypothesis and empirical methods of Cognitive Processes active during the time of programming. First develop cognitive model of program comprehension and debugging methods. Second empirical study of programmers, it was designed and controlled to explore the processes involves in program comprehension and debugging.

2. COGNITIVE MODEL

It is worried with comprehension of procedures that the human cerebrum uses to handle complex undertakings including seeing, learning, recollecting, and considering, anticipating is moving around the framework. Essential objective of an intellectual model is to logically clarifying more than one of the above subjective procedures and their collaboration (8). They uncover data identified with subjective and perceptual imperatives.

It shows up in numerous fields that arrangement with insight, going from recognition to critical thinking and deciding. It fuses mental models which are as indicated by Johnson – Laird's hypothesis (37). It gives essential structure.

Mental Model (46), (47) plays a focal and bringing together part in speaking to protests, situation, groupings of occasions far and wide, social and mental activities of regular schedule. Mental model are improved forms of complex situation made in the working memory. It is less demanding to imagine, decipher and foresee activities. Built Mental model depend on:

1. Perception.
2. Comprehension.
3. Imagination.

Some of the cognitive models which are proposed and studied is in the areas of Text comprehension, Graph and picture comprehension, Program comprehension and human Computer Interaction.

Text comprehension (38) is important in research activities because of reading and understanding the code. Text and diagram comprehension offers a cognitive strategies and resulting mental representations. A cognitive model portrays the subjective procedures and impermanent data structures in software engineer's head. An intellectual element incorporates the accompanying:

1. Knowledge Level
2. Social Level
3. Co-operation
4. Co-ordination
5. Belief

6. Commitment
7. Goal to Achieve
8. Capacity

3. PROGRAM COMPREHENSION METHODS

Program comprehensions have two major key strands:

1. The first is observational examination which makes progress toward a comprehension the project structure, control stream and working i.e. mental model that software engineer's utilization when comprehension programs.
2. The second includes device based methodology, which focuses on creating semi-computerized apparatus support to enhance program understanding.

It provides analysis of how two approaches of research are related. During 1970's various non-technical and random methods were applied for cognitive based code comprehension. Some technical methods are evolved for cognitive based code comprehension. To understand and describe developer's mental representation mental model was used. This mental module was evolved from a cognitive module. As per shown in the Figure 1.

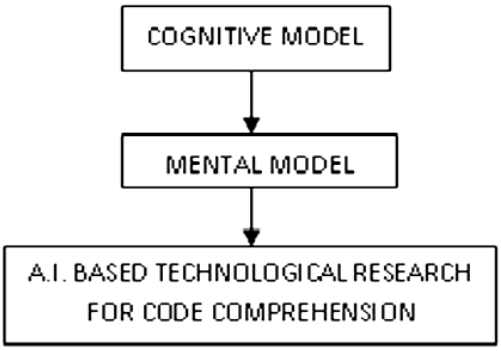

Figure 1: Program comprehension methods

These plans and rules of programming could support in developing cognitive model.

At the end we have Artificial intelligent based technical research for code comprehension was evolved from mental model. The mental model encodes the software engineer's present comprehension of the system. It comprises of a detail of the system objectives and the usage regarding the information structures and calculations utilized. As per shown in the Figure 2.

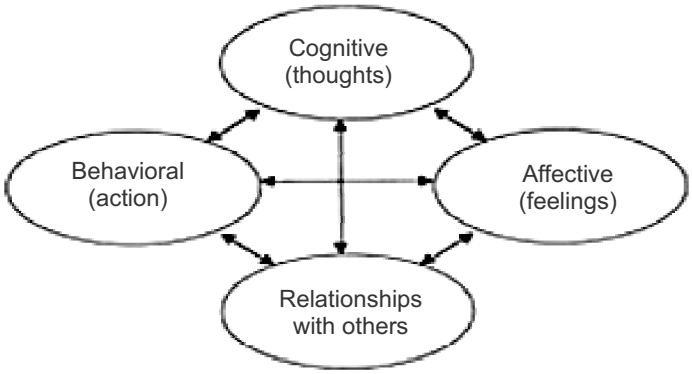

Figure 2: Program comprehension model

4. PROPOSED COGNITIVE MODEL

When a person involved in studies to investigate debugging strategies with multiple ways of visualizations in IDE's, this limited the use of representations. We have to select a few strategies among them during the time of experiment. But restricting the strategies gives not a proper solution to the professional programmers. For this a special type of IDE (jGRASP) is used, which offers a combination of visualizations (it is used performance wise and professionally both). It gives programmers unrestricted access to many static and dynamic visualization aids with program code.

It could generate a problem when question get arise. Defining such problem is called **"Problem Statement".**

A cognitive model has 3 (three) main components, As per shown in Figure 7:

1. Cognitive Aids / Representations used while debugging.
2. A cognitive process is either primed by a cognitive aid or, a process that is inherently evoked.
3. Mental Representations are derived from the cognitive processes and cognitive aids. Programmer constructs and manipulates anybody's mental representations in case of interacting with the programming environment and understanding the information presented.

In case of program plans three types of comprehension process were used:

4.1. Top-down comprehension.

4.2. Bottom-up comprehension.

4.3. Systematic and as needed comprehension.

4.4. Integrated comprehension.

4.1. Top-down comprehension

If there should be an occurrence of Top-down perception (4) process begins with a speculation about the general way of the project. This introductory hypo is then refined auxiliary speculation. Auxiliary theory is refined and assessed in a profundity first way. Top-Down understanding (49) is utilized when the code is commonplace. It takes after steps:

1. **Knowledge Base** is related to gathering information from different servers connected within a Network or, WAN (15). As per shown in the Figure 6.
2. **Situation Model** is related to situation arises during code decoding process. As per shown in Figure 3.
   a) In case of Normal way Reading of source code, the code decoding and comprehension process fluency is good.
   b) In case of Learning (Lexical Analysis) of source code *i.e.* Dyslexic, the code decoding fluency is poor whereas the comprehension process is good.
   c) In case of Learning without training *i.e.* Hyperlexic, the code decoding fluency is good whereas the comprehension process is poor.
   d) In case general program or, module learning difficulties code decoding and comprehension process fluency are both poor.

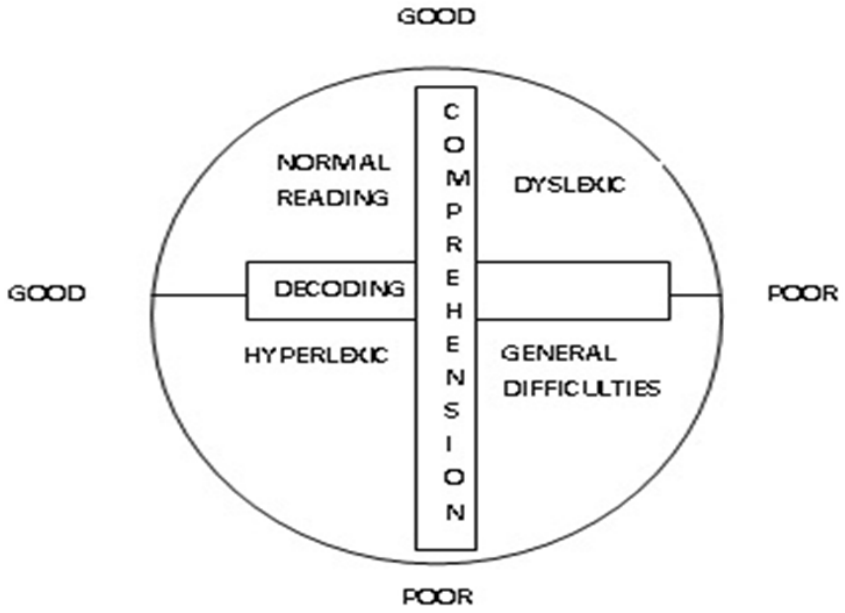

Figure 3: Situation model

3. **Program Model** is inter-related with Program Assessment, Capacity, Planning, Implementation and Evaluation. As per shown Figure 4.
   - **a)** **Assessment** of the program counts its importance and valuation of code.
   - **b)** **Capacity** of program means its impact and scope.
   - **c)** **Planning** of the program is used to give it a proper structure and sequence of steps.
   - **d)** **Implementation** of the program is to decide area to implement, training and size.
   - **e)** **Evaluation** of the program is related to program nature.

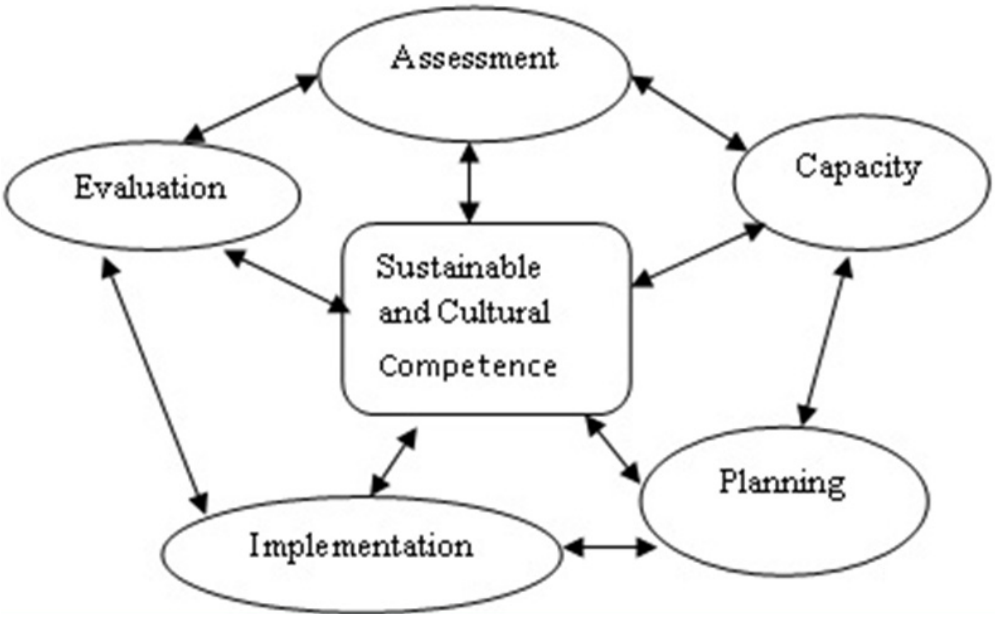

Figure 4: Program model

4.2. Bottom-up Comprehension

In case of Bottom-Up comprehension assume that programmers first read code statements and then, mentally chunk or, group these statements into higher level abstractions. It follows reverse process of Top Down comprehension. These abstractions are aggregated further until a high-level understanding of the program is attained (26), Shnaiderman and Msyer's cognitive framework differentiates between syntactic and semantic knowledge of programs. According to Pennington (46), (47) describes a Bottom up model. She observed that programmers first develop control-flow abstraction of a called Program Model.

Once the program model is fully assimilated the situation model is develop. It encompasses knowledge about data -flow abstraction and functional abstraction. The assimilation process describes how the mental model evolves using the programmer's knowledge base together with program so user code and documentation. It may be top-down or, bottom-up depending on programmer's initial knowledge.

4.3. Systematic and As-needed Comprehension

Littman et al. (59) describes two comprehension strategies

4.3.1. Systematic Comprehension

Systematic is where a programmer systematically reads through code in detail, looking at both the control-flow and data-flow abstractions is used to obtain a thorough understanding of the code.

4.3.2. As-needed Comprehension

As-needed comprehension is the method where the programmer only looks at the code related to a particular task. Parts of the code are looked at only when the programmer needs to understand them. As-needed comprehension description could be thought of as describing both checklist and scenario defect detection methods gets highlighted.

Littman (59) in 1986 watched that developers either deliberately read the code in subtle element, following through the control-stream and information stream reflection in the system to pick up a worldwide comprehension of the project or, that they take an as required methodology concentrating just on the code identifying with a specific current workload.

Subjects utilizing a precise system procured both static learning (data about the structure of the project) and easygoing information (connections between segments in the project when it is executed). This empowered them to frame a mental model of the system. This technique is considered as learning base procedure.

4.4. Integrated Comprehension

Von Mayrhauser and Vans coordinated the Top-Down, Bottom-Up, Systematic and as required Comprehension techniques. An integrated metamodel created by Von Mayrhauser and Vans expands on four noteworthy segments (models) like, as per shown Figure 5.

1. Top-Down Model
2. Program Model
3. Situation Model
4. Knowledge Base

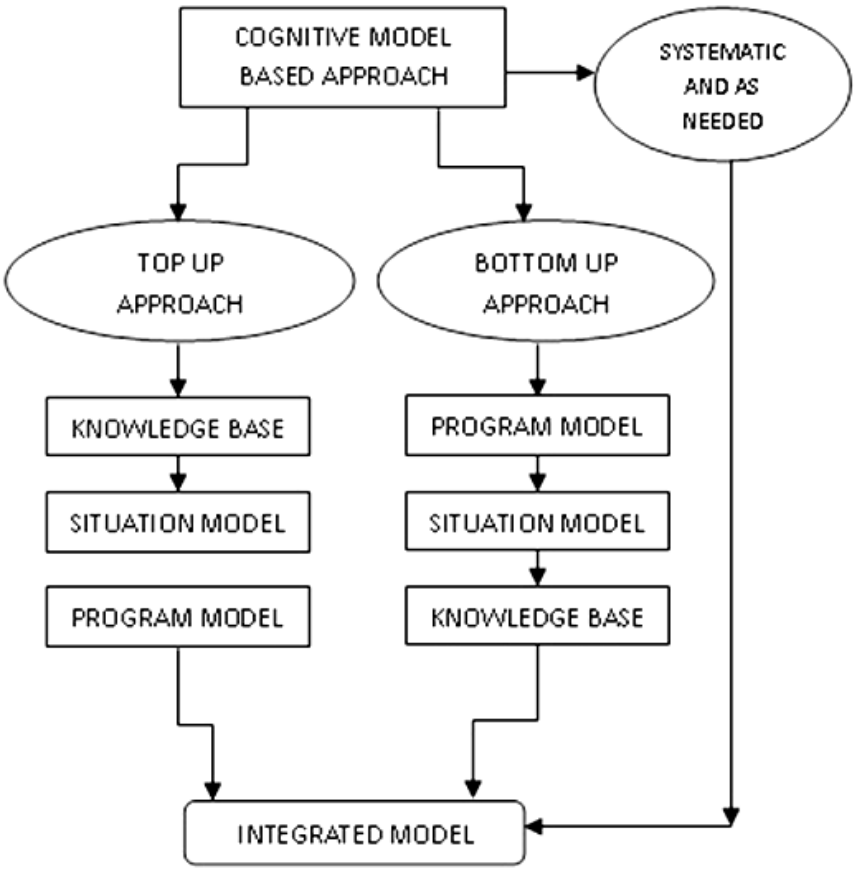

Figure 5: Integrated model

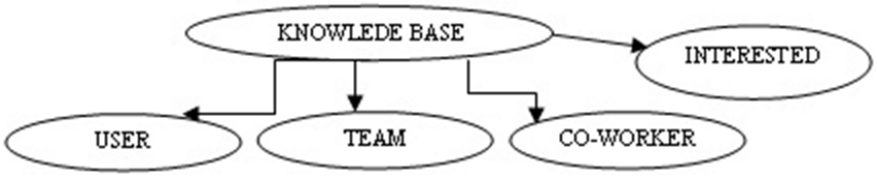

Figure 6: Knowledge-base

The initial three models portray the cognizance forms used to make mental representation at different levels of reflection. The fourth segment depicts the learning base expected to perform a cognizance process. Program that is precisely planned and very much reported will be simpler to comprehend, change or, reuse in future.

Pennington's trial (46) demonstrates that decision of dialect affects cognizance process. There are 3 segments to his model. The information base encodes the software engineer's aptitude and foundation learning. COBOL software engineer's reliably fared better at noting questions identified with information stream than FORTRAN developers though, FORTRAN software engineers reliably fared superior to anything COBOL developer's for control stream questions. Sorts of methodology such as Modular, Structured or, Object Oriented writing computer programs are utilized. Tremendous specialists built up the conventional subjective hypotheses for system perception. Examines the ramifications of the created speculations on instrument outline and at times. Likewise, how instruction and project outline could be enhanced to address program understanding difficulties. Program (by Curtis) perception research gives numerous counsel on how apparatuses can be made strides. The device planners use current speculations to comprehend the elements that are required.

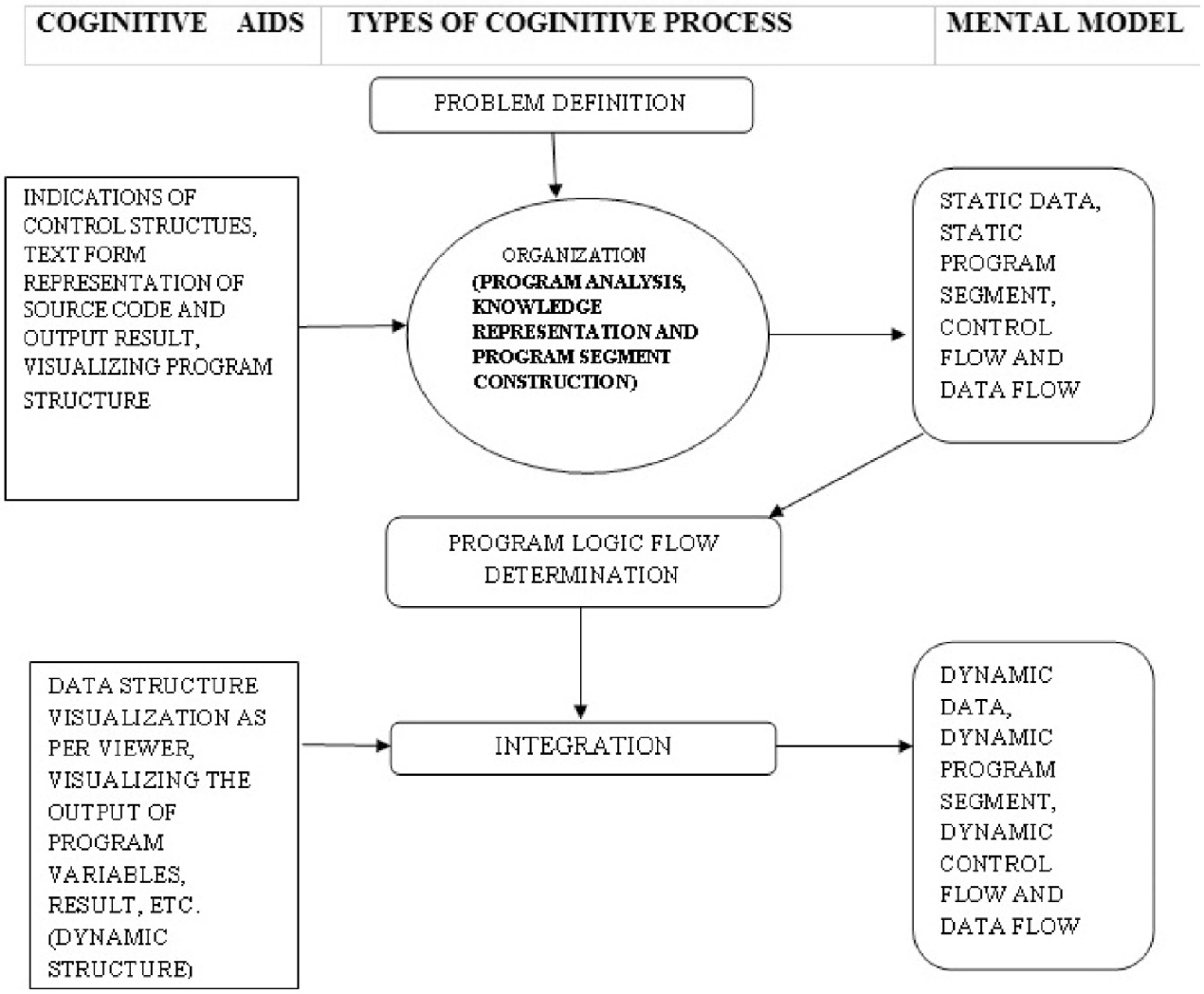

Figure 7: Proposed cognitive model

Cognitive Models and Tool implications

1. Documentation.
2. Browsing and navigation support.
3. Searching and Querying.
4. Multiple Views.
5. Context Driven Views
6. Cognitive Support.

5. TOOL REQUIREMENTS EXPLICITLY IDENTIFIED

Several researchers studied expert programmers in industrial settings and consequently recommended specific requirements for improving tools to support comprehension like;

1. Concept assignment problem.
2. Reverse engineering tools needs.
3. Importance of search and history.
4. Information needs for maintainers.
5. Software visualization tool needs.

6. METHODS FOR DETERMINING PROGRAM COMPREHENSION TOOL REQUIREMENTS

6.1. Program Comprehension Research Tools

The field of program comprehension research has resulted in many diverse tools to assist in program comprehension. Program comprehension tools generally implement a reverse engineering process. As per shown in Figure 8, basic activities in reverse engineering process includes:

1. Extraction.
2. Analysis.
3. Presentation.

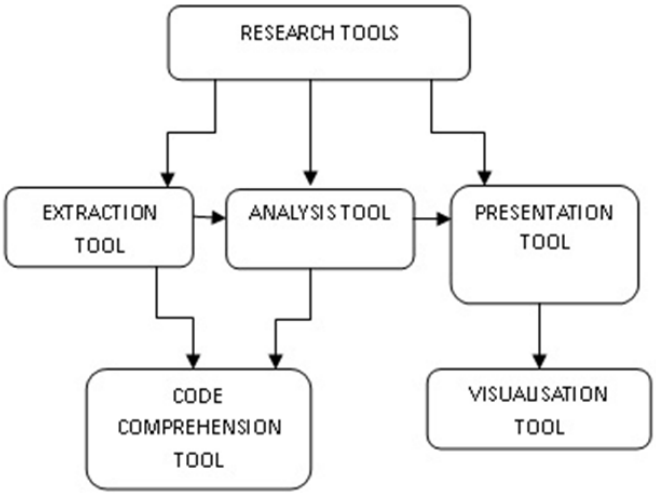

Figure 8: Program comprehension tool

Extraction apparatuses incorporate parsers and information gathering devices to gather both static and element information. Static information is acquired by extricating actualities from the source code. A Fact Extractor ought to have the capacity to figure out what Artifacts the system characterizes, uses, imports and fares and also relationship between those curios. The advancements fundamental truth extractors depend on procedures from compiler build particle (1) e.g. Present day Fact Extractors incorporate CAN, a quick C/C++ extractor, from the Columbus figuring out apparatus.

Dynamic information is acquired by analyzing and separating information from the run time conduct of the system. Such information can be removed through a wide assortment of follow investigation instruments and strategies.

Investigation apparatuses bolster exercises, for example, grouping, idea task, highlight distinguishing area examination, cutting and measurements figuring's. There are various programming procedures that can be utilized amid figuring out to distinguish programming parts.

Dynamic investigation for the most part includes instrumentation of the source code. With element investigation just a subset of the system might be applicable however dynamic follows can be huge posturing noteworthy difficulties amid the examination of the information. Static examination can be utilized to prune the measure of data took a gander at amid element investigation. Presentation instruments incorporate Code editors, Browsers, Hypertext viewers and Visualizations.

7. TOOLS FOR COMPREHENSION

The visualization tools are created for object oriented programming. Both inspection and visualization tools may have features that can help to support cognitive strategies for program and code comprehension. Tools will be compared to the criteria defined by Linos.

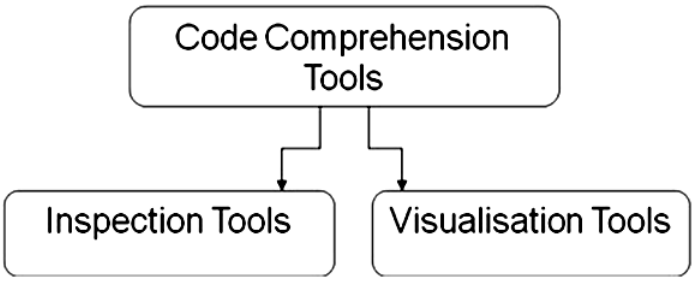

Figure 9: Code comprehension tools

Both inspection and visualization tools may have features that can help to support cognitive strategies for program comprehension. As per shown in Figure 9.

7.1. Inspection Tools

Inspection Process:

Step 1: Getting an overview of the project description.

Step 2: In the preparation step, each member of the group works on their own and attempts to gain an understanding of the documents which is being provided.

Step 3: In this step, it is used to check that all problems that were raised in the inspection process have been dealt with.

This inspection process is developed by Fagan (14) in 1972 and then, updated by himself in 1986 (15). As per the comparison shown for inspection tools and their features in the table1.

Table 1
Inspection tools and their features

S. No	*Name*	*Type*	*Features*
1.	ASSIST (Asynchronous or, Synchronous Software Inspection Tool) (61),(62)	Distributed	Defect finding Aids, Enhanced Document representation, Facility for metric collection and analysis, provision of facilities for distributed inspection, provides online checklists, Generic software inspection template
2.	Scrutiny (53)	Distributed	It mainly supports documents. It inspect by following the steps …Initiation -> Preparation -> Resolution -> Resolution -> Completion

S. No	*Name*	*Type*	*Features*
3.	ICICLE (Intelligent Code Inspection in a C Language Environment) (67)	Individual	It supports mainly C language constructs through two phase inspection like ; individual inspection and meeting.
4.	Collaborative Software Inspection (CSI) (63)	Distributed	It provides an online inspection environment by favoring four types of collaborative inspection meeting such as; same time and place , same time and different place , different time and same place , different time and place. It supports both synchronous (group meeting) and asynchronous (individual checking) activities.
5.	WiP (54)	Distributed	It attempts to solve the problem of having a scattered inspection team by utilizing www and is designed to distribute the documents to be inspected. It allows document marking, search documents, allow selection of checklists and gather inspection statistics. It provides access to users to find source documents and checklists.

7.2. Visualization Tools

The visualization tools are created for Object oriented programming. It acts as an interface between two powerful information processing systems *i.e*. The Human Mind and The Modern Computer.

Table 2
Program visualization tools

S.No	*Name*	*Features*
1.	Easy CODE (C++) (69)	It is a PC based commercial windows package from Siemens AG Austria. It uses structured program techniques to visually display programs. It is a improved version XperCASE.
2.	With Class 98 (67)	It is an Object oriented CASE tool developed by MicroGold software for Windows on PC. The program allows the construction of graphical model in an Object Oriented methodology and allows selecting from several OO methods. It includes unified method, Run Baugh method, Coad Yourdon method, Booch method, etc. With the use of this designing of class diagrams, detailing class attributes and methods are possible.
3.	SNiFF + (70)	It supports C, C++, Java, Fortran program developing environment. It provides features including version and configuration management, project management, code comprehension and debugging, browsing document and document building management. It contains filtering and visualization techniques.
4.	ISVis (55)	It helps to visualize interaction patterns in executing program on Sun Solaris, SunOS and IRIX platforms This program is carry out large amount of real information and able to carry out abstractions, data simplifications. It leads to "Visualize interaction patterns in program execution"
5.	Look! (65)	It is C++ debugging and visualization tool available for Windows, SunOS, Solaris and AIX. It provides views of Object creation relationship, class clusters , Object Networks , message Flow and dynamic class views

This visualization process is developed by Gershon et al. (17) in 1972 and then, updated by himself in 1986 (15). It involves manipulating information, data and knowledge and converting it into a visual representation in more than one dimension, which utilizes the human visual system. A comparison is shown in table 2, various visualization tools. Visualization tool has become very common practices for identify the relationship between the various program. As graphical representations of code is much more correct and beneficial for novices and programmer, also more understanding as compared to normal textual information.

8. CHALLENGES IN PROGRAM COMPREHENSION

We have concluded from the review that there is need to develop agent based code comprehension models (72,73). The human factors like experience, knowledge and intension are the basic factors which can affect the cost of program comprehension (74). The source code contains various relationships in variables, classes and functions, the tools have implications to identify the relation dependencies. The software visualization tools can perform specific tasks only, but the relations within the code is always remains a challenge. So there is no such tool or model which can reduce comprehension task. The review also reveals the need for higher-level abstractions and visualizations Semantic and visual support required for software maintainers during routine maintenance tasks (75). Cognitive agents execute a decision cycle in which they process events and derive a choice of action from their beliefs and goals. Current state-of-the-art debuggers for agent programs provide insight in how agent behavior originates from this cycle but less so in how it relates to the program code (76).

9. CONCLUSION

Code comprehension process is an approach of understanding the cognitive and social aspects of program comprehension using conventional methods of agents as well as technical support. Code comprehension plays a remarkable role for software re-engineering. It is an A.I. Based technique using the automated support of software tools. It replaces any multi agent with computer based multi-agent system**.** In future we need to develop the agent based model for code comprehension. This can automate the code comprehension process during software maintenance.

REFERENCES

[1] Aho AV, Sethi R, Ullman JD. Compilers : Principals, Techniques and Tools. 2000.

[2] Ball T, Eick SG. Software visualization in the large. Computer. 1996 Apr;29(4):33-43.

[3] Creswell JW. Research design: Qualitative, quantitative, and mixed methods approaches. Sage publications; 2013 Mar 14.

[4] FP Jr B. No Silver Bullet Essence and Accidents of Software Engineering. Computer. 1987 Apr 1(4):10-9.

[5] Brooks R. Towards a theory of the comprehension of computer programs. International journal of man-machine studies. 1983 Jun 1;18(6):543-54.

[6] Rao AS, Georgeff MP. A model-theoretic approach to the verification of situated reasoning systems. In Proceedings of the 13th international joint conference on Artifical intelligence-Volume 1 1993 Aug 28 (pp. 318-324). Morgan Kaufmann Publishers Inc..

[7] Blackwell AF, Jansen AR, Marriott K. Restricted focus viewer: a tool for tracking visual attention. In Theory and application of diagrams. 2000 Sep 1; (pp. 162-177). Springer Berlin Heidelberg.

[8] Blackwell AF, Jansen AR, Marriott K. Restricted focus viewer: a tool for tracking visual attention. In Theory and application of diagrams . 2000 Sep 1 ; (pp. 162-177). Springer Berlin Heidelberg.

[9] Busemeyer JR, & Diederich A. Cognitive modeling. Los Angeles: Sage. 2010.

[10] Carney RN, Levin JR. Pictorial illustrations still improve students' learning from text. Educational psychology review. 2002 Mar 1;14(1):5-26.

[11] Cheng PC, Lowe RK, Scaife M. Cognitive science approaches to understanding diagrammatic representations. Artificial Intelligence Review. 2001 Mar 1;15(1-2):79-94.

[12] Cox R, Brna P. Analytical reasoning with external representations: Supporting the stages of selection, construction and use. Journal of Artificail Intelligence in Education. 1995; 6 (2/3), 239-302.

[13] Crane HD. The Purkinje image eyetracker, image stabilization, and related forms of stimulus manipulation. Visual science and engineering: Models and applications. 1994 Mar 30:15-89.

[14] Cross II JH, Hendrix TD, Umphress DA, Barowski LA, Jain J, Montgomery LN. Robust generation of dynamic data structure visualizations with multiple interaction approaches. ACM Transactions on Computing Education (TOCE). 2009 Jun 1;9(2):13.

[15] Cutrell E, Guan Z. What are you looking for?: an eye-tracking study of information usage in web search. InProceedings of the SIGCHI conference on Human factors in computing systems 2007 Apr 29 (pp. 407-416). ACM.

[16] Ducasse M, Emde AM. A review of automated debugging systems: knowledge, strategies and techniques. InProceedings of the 10th international conference on Software engineering 1988 Apr 1 (pp. 162-171). IEEE Computer Society Press.

[17] Duchowski A. Eye tracking methodology: Theory and practice. Springer Science & Business Media; 2007 Sep 14.

[18] Gentner D. The mechanisms of analogical learning. In S. Vosniadou, & A. Ortony, Similarity and Analogical Reasoning. 1989; (pp. 197-241). Cambridge: Cambridge University Press, England.

[19] Romero P, Cox R, du Boulay B, Lutz R. Visual attention and representation switching during java program debugging: A study using the restricted focus viewer. In Diagrammatic Representation and Inference 2002 Apr 18 (pp. 221-235). Springer Berlin Heidelberg.

[20] Romero P, du Boulay B, Cox R, Lutz, R. Java debugging strategies in multi representational environments. 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG). 2003b. Keele University, UK.

[21] Romero P, Lutz R, Cox R, Du Boulay B. Co-ordination of multiple external representations during Java program debugging. In Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on 2002; (pp. 207-214). IEEE.

[22] Schnotz W, Bannert M. Construction and interference in learning from multiple representation. Learning and instruction. 2003 Apr 30; 13(2):141-56.

[23] Schnotz W, Bannert M. Support and interference effects in learning from multiple representations. In European Conference on Cognitive Science 1999 Oct 30 (pp. 447-452).

[24] Shah P, Carpenter PA. Conceptual limitations in comprehending line graphs. Journal of Experimental Psychology: General. 1995 Mar;124(1):43.

[25] Shah P, Hoeffner J. Review of graph comprehension research: Implications for instruction. Educational Psychology Review. 2002 Mar 1;14(1):47-69.

[26] Shah P, Mayer RE, Hegarty M. Graphs as aids to knowledge construction: Signaling techniques for guiding the process of graph comprehension. Journal of Educational Psychology. 1999 Dec;91(4):690.

[27] Shneiderman B, Mayer R. Syntactic/semantic interactions in programmer behavior: A model and experimental results. International Journal of Computer & Information Sciences. 1979 Jun 1;8(3):219-38.

[28] Sime J. An investigation into teaching and assesment of qualitative knowledge in engineering. InEuropean Conference on Artificial Intelligence on Education 1996 Sep (pp. 240-246).

[29] Soloway E, Adelson B, Ehrlich K. Knowledge and processes in the comprehension of computer programs. InChi et al 1988 Jun (Vol. 123, pp. 129-152).

[30] Soloway E, Lampert R, Letovsky S, Littman D, Pinto J. Designing documentation to compensate for delocalized plans. Communications of the ACM. 1988 Nov 1;31(11):1259-67.

[31] Green TR. Cognitive dimensions of notations. People and computers V. 1989 Sep 5:443-60.

[32] Busemeyer JR, Diederich A. Cognitive modeling. 2010. Los Angeles: Sage.

[33] Carney RN, Levin JR. Pictorial illustrations still improve students' learning from text. Educational psychology review. 2002 Mar 1;14(1):5-26.

[34] Cheng PC, Lowe RK, Scaife M. Cognitive science approaches to understanding diagrammatic representations. Artificial Intelligence Review. 2001 Mar 1; 15(1-2):79-94.

[35] Gernsbacher MA, Varner KR, Faust ME. Investigating differences in general comprehension skill. Journal of Experimental Psychology: Learning, Memory, and Cognition. 1990 May; 16(3):430.

[36] Gilmore DJ. Models of debugging. Acta psychologica. 1991 Dec 1;78(1-3):151-72.

[37] Grubb P, Takang AA. Software maintenance: concepts and practice. World Scientific; 2003 Jul 7.

[38] Johnson-Laird PN. Mental models: Towards a cognitive science of language, inference, and consciousness. Harvard University Press; 1983.

[39] Just MA, Carpenter PA. A capacity theory of comprehension: individual differences in working memory. Psychological review. 1992 Jan;99(1):122.

[40] Katz IR, Anderson JR. Debugging: An analysis of bug-location strategies. Human-Computer Interaction. 1987 Dec 1;3(4):351-99.

[41] Kintsch W, Van Dijk TA. Comment on se rappelle et on résume des histoires. Langages. 1975 Dec 1(40):98-116.

[42] Mautone PD, Mayer RE. Cognitive aids for guiding graph comprehension. Journal of Educational Psychology. 2007 Aug;99(3):640.

[43] Mayer RE. Learning strategies for making sense out of expository text: The SOI model for guiding three cognitive processes in knowledge construction. Educational psychology review. 1996 Dec 1;8(4):357-71.

[44] Narayanan NH, Hegarty M. On designing comprehensible interactive hypermedia manuals. International journal of human-computer studies. 1998 Feb 28;48(2):267-301.

[45] Narayanan NH, Hegarty M. Multimedia design for communication of dynamic information. International journal of human-computer studies. 2002 Oct 31; 57(4):279-315.

[46] Nathan MJ, Kintsch W, Young E. A Theory of Algebra-Word-Problem Comprehension and Its Implications for the Design of Learning Environments. Cognition & Instruction. 1992; 9 (4), 329.

[47] Pennington N. Comprehension strategies in programming. InEmpirical studies of programmers: second workshop 1987 Dec 1 (pp. 100-113). Ablex Publishing Corp..

[48] Pennington N. Stimulus structures and mental representations in expert comprehension of computer programs. Cognitive psychology. 1987 Jul 31;19(3):295-341.

[49] Romero P, Cox R, Du Boulay B, Lutz R. A survey of external representations employed in object-oriented programming environments. Journal of Visual Languages & Computing. 2003 Oct 31;14(5):387-419.

[50] E. Soloway, B. Adelson, and K. Ehrlich, "Knowledge and Processes in the Comprehension of Computer Programs," in The Nature ofExpertise, M. Chi, R. Glaser, and M. Farr, eds.A. Lawrence Erlbaum Associates, Hillsdale, N.J., 1988, pp.129-152.

[51] Trabasso T, Van den Broek P. Causal Thinking and the Representation of Narrative Events. Journal of Memory and Language. 1985; (24), 612-630.

[52] Van Oostendorp H, Goldman SR, editors. The construction of mental representations during reading. Psychology Press; 1998 Nov 1.

[53] Canfora G, Mancini L, Tortorella M. A workbench for program comprehension during software maintenance. InProgram Comprehension, 1996, Proceedings., Fourth Workshop on 1996 Mar 29 (pp. 30-39). IEEE.

[54] Gintell J, Arnold J, Houde M, Kruszelnicki J, McKenney R, Memmi G. Scrutiny: A collaborative inspection and review system. In Software Engineering—ESEC'93. 1993 Sep 13 (pp. 344-360). Springer Berlin Heidelberg.

[55] Harjumaa L, Tervonen I. A WWW-based tool for software inspection. InSystem Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on 1998 (Vol. 3, pp. 379-388). IEEE.

[56] Jerding DF. ISVis [Internet]. Cc.gatech.edu. 2016 [cited 31 March 2016]. Available from: http://www.cc.gatech.edu/morale/tools/isvis/isvis.html

[57] Dennett DC. The Intentional Stance. The MIT Press. 1987

[58] Weiss G. Multi- Agent Systems. MIT Press. 1999.

[59] Wooldridge M. Agent-based software engineering. InSoftware Engineering. IEE Proceedings-[see also Software, IEE Proceedings] 1997 Feb (Vol. 144, No. 1, pp. 26-37). IET.

[60] Littman DC, Pinto J, Letovsky S, Soloway E. Mental models and software maintenance. In *Empirical Studies of Programmers , 1986;* pp. 80-98. Ablex Publishing Corporation.

[61] Macdona F, Miller J, Brooks A, Roper M, Wood M. A review of tool support for software inspection. InComputer-Aided Software Engineering, 1995. Proceedings., Seventh International Workshop on 1995 Jul 10 (pp. 340-349). IEEE.

[62] Macdonald F, Miller J. Automated Generic Support for Software Inspection, 10th International Quality Week, San Francisco, 1997 May 27-30.

[63] MacDonald F, Miller J. A software inspection process definition language and prototype support tool. Software Testing, Verification and Reliability. 1997 Jun 1;7(2):99-128.

[64] Mashayekhi V, Drake JM, Tsai WT, Riedl J. Distributed, collaborative software inspection. Software, IEEE. 1993 Sep;10(5):66-75.

[65] Von Mayrhauser A, Vans AM. Program comprehension during software maintenance and evolution. Computer. 1995 Aug; 28(8):44-55.

[66] Cain J, McCrindle R. Software visualisation using C++ lenses. Proceedings Seventh International Workshop on Program Comprehension. 1999 May: 20-26.

[67] Robson DJ, Bennett KH, Cornelius BJ, Munro M. Approaches to Program Comprehension. Journal of Systems Software. 1991 February; Vol. 14, No. 2, pp. 79-84,

[68] Sembugamoorthy V, Brothers L. ICICLE: Intelligent code inspection in a C language environment. InComputer Software and Applications Conference, 1990. COMPSAC 90. Proceedings., Fourteenth Annual International 1990 Oct (pp. 146-154). IEEE.

[69] Shneiderman B. Software psychology: Human factors in com. Computer and Information Systems. 1980.

[70] Siemens EasyCODE (C++) [Internet]. http://siemens-easycode-c.software.informer.com/ 2016 [cited 31 March 2016]. Available from: . Österreich S. Siemens EasyCODE (C++) [Internet]. Software Informer. 2016 [cited 31 March 2016]. Available from: http://siemens-easycode-c.software.informer.com/

[71] Software Development: Product review: SNiFF+ for Linux [Internet]. Linuxfocus.org. 2016 [cited 31 March 2016]. Available from: http://www.linuxfocus.org/English/March2000/article140.shtml

[72] Young P. Program comprehension. Visualisation Research Group, Centre for Software Maintenance, University of Durham. 1996 May 28.

[73] Tiarks R, Röhm T. Challenges in Program Comprehension. Software technik-Trends. 2012; 32(2):19-20.

[74] Raj Singh D. Agent Based Code Comprehension Model Using Semantic Knowledge Base. International Journal of Engineering Research and Technology [Internet]. 2014 [cited 1 April 2016];Vol. 3 - Issue 5 (May - 2014)(Vol. 3 - Issue 5 (May - 2014). Available from: http://www.ijert.org/view-pdf/9575/agent-based-code-comprehension-model-using-semantic-knowledge-base.

[75] Siegmund J, Schumann J. Confounding parameters on program comprehension: a literature survey. Empirical Software Engineering. 2014;20(4):1159-1192.

[76] Gonen B, Fang X, El-Sheikh E, Bagui S, Wilde N, Zimmermann A. Ontological Support for the Evolution of Future Services Oriented Architectures. TMLAI. 2014;2(6).

[77] Koeman V, Hindriks K. Designing a Source-Level Debugger for Cognitive Agent Programs. PRIMA 2015: Principles and Practice of Multi-Agent Systems. 2015;:335-350.