AGENT BASED SOFTWARE CODE COMPREHENDING

Ram Gopal Gupta[1], Bireshwar Dass Mazumdar[2],Kuldeep Yadav[3]
[1]Research Scholar, Uttarakhand Technical University,
[2]Associate Professor, Institute of Engineering & Rural Technology
[3]Associate Professor, Department of CSE, COER
[1]rgmail@rediffmail.com,
[2]emailrgg@gmail.com

Abstract-**A result, software systems remain subject to changes and maintenance throughout their lifetime. It is crucial to manage such changes, as a lot of effort and time are required in order to keep software systems operational and fit for purpose. Software comprehension is essential for developing, maintain, and improving software. This is particularly true of agent-based systems, in which the actions of autonomous agents are affected by numerous factors, such as events in dynamic environment, local uncertain beliefs, and intentions of other agents. The best alternative for software maintainers is to comprehend source code, which is both costly and time consuming. More specifically, 50–90% of the maintenance engineers' time is reported to be spent on software code comprehension. This paper focusing on agents based software comprehension parameters and tools that support the discovery and synthesis of information found in both source code and software documents are an important issue for the software maintenance research community**

Keyword: **software maintenance, cognitive parameters, code comprehension, data mining, agent, visualization tools, code comprehensions tools.**

I. Introduction:-

Today major amount of programming work is accomplished on sophisticated s/w 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 Textproduced 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:-

#1) programming language expertise.

#2) adjustment with the IDE.

#3) personal preference.

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 and combination of strategic knowledge with his or, her knowledge by the coordination of appropriate visualizations and functional tools of the IDE.

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 software code comprehension the main emphasis is on understanding the code 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.

Text comprehension (Just & Carpenter, 1992)[38] is important in research activities because of reading and understanding the code whereas Text and Diagram comprehension offers a cognitive strategies and resulting mental representations.

II. Methods:-

There are two key strands of software code comprehension research:-

(a) The first is Empirical Research which strives for an understanding of Cognitive processes that programmers use when understanding programs.

(b) The second involves Technology Research with a focus on developing semi-automated tool support to improve software code comprehension.

It provides a meta-analysis of how two strands 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.These plans and rules of programming could support in developing cognitive model.

At the end A.I. BASED TECHNOLOGICAL RESEARCH FOR CODE COMPREHENSION was evolved from mental model.

The mental model encodes the programmer's current understanding of the program. It consists of a specification of the program goals and the implementation in terms of the data structures and algorithms used.

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

(a)Top-down comprehension.
(b)Bottom-up comprehension.
(c)Systematic and as –needed comprehension.
(d)Integrated comprehension.

(a)Top-down comprehension:-

In case of Top-down comprehension (Brooks, 1983)[4] process starts with a hypothesis about the general nature of the program. This initial hypo is then refined subsidiary hypothesis. Subsidiary hypothesis are refined and evaluated in a depth first manner. Top-Down comprehension (Soloway, 1984)[49] is used when the code is familiar. It follows following steps:-

❖ *Knowledge Base* is related to gathering information from different servers connected within a Network or, WAN (Ducassé, M., & Emde, A. -M. (1988)) [15].
❖ *Situation Model* is related to situation arises during code decoding process.

- In case of Normal way Reading of source code, the code decoding and comprehension process fluency is good.
- In case of Learning (Lexical Analysis) of source code i.e. Dyslexic, the code decoding fluency is poor whereas the comprehension process is good.
- In case of Learning without training i.e. Hyperlexic, the code decoding fluency is good whereas the comprehension process is poor.
- In case general program or, module learning difficulties code decoding and comprehension process fluency are both poor.

III. *Program Model* is inter-related with Program Assessment, Capacity, Planning, Implementation and Evaluation.

- Assessment of the program counts it's importance and valuation of code.
- Capacity of program means it's impact and scope.
- Planning of the program is used to give it a proper structure and sequence of steps.
- Implementation of the program is to decide area to implement, training and size.
- Evaluation of the program is related to program nature.

(b) *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. Itfollows reverse process of Top Down comprehension. These abstractions are aggregated further until a high-level understanding of the program is attained (Shneiderman, 1979)[26],

Shnaiderman and Msyer's cognitive framework differentiates between syntactic and semantic knowledge of programs.

AccordingtoPenington (Penington,1987)[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.

(c) Systematic and As-needed comprehension:-

Littman et al.[61] describes two comprehension strategies –

(i) *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 thourough understanding of the code.

(ii) *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 1986)[61] observed that programmers either systematically read the code in detail, tracing through the control-flow and data -flow abstraction in the program to gain a global understanding of the program or, that they take an as needed approach focusing only on the code relating to a particular task at hand.

Subjects using a systematic strategy acquired both static knowledge (information about the structure of the program) and casual knowledge (interactions between components in the program when it is executed). This enabled them to form a mental model of the program.

This strategy is considered as knowledge base strategy.

*(d) Integrated comprehension***:-**

Von Mayrhauser and Vans integrated the Top-Down, Bottom-Up, Systematic and as needed Comprehension strategies.

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. Basic activities in reverse engineering process includes:-

- Extraction.
- Analysis.
- Presentation.

Extraction tools include parsers and data gathering tools to collect both static and dynamic data. Static data is obtained by extracting facts from the source code. A **Fact Extractor** should be able to determine what **Artifacts** the program defines, uses, imports and exports as well as relationship between those artifacts. The technologies underlying fact extractors are based on techniques from compiler construct- ion (Aho, 2000)[1] e.g. Modern Fact Extractors include CAN , a fast C/C++ extractor, from the Columbus reverse engineering tool (Ferenc, 2004) and CPPX (Dean, 2001).

Dynamic data is obtained by examining and extracting data from the run time behavior of the program. Such data can be extracted through a wide variety of trace exploration tools and techniques (Hamou – Lha dj, 2004).

Analysis tools support activities such as clustering, concept assignment, feature identification (Eisenbarth, 2003) transformations, domain analysis, slicing and metrics calculations. There are numerous software techniques that can be used during reverse engineering to identify software components (Kosch Ka, 2000).

Dynamic analysis only a subset of the program may be relevant but dynamic traces can be very large posing significant challenges during the analysis of the data.Static analysis can be used to prune the amount of information looked at during dynamic analysis (Systa, 2001).

IV. *Presentation tools include Code editors,*

Browsers, Hypertext viewers and Visualizations.

Methods for evaluating comprehension tools→

In many cases the comprehension tools researchers using case studies. There have been some usability experiments conducted to evaluate program comprehension tools (Storey, 2000). Two different types of available tools, inspection and visualization.

V. Tools for Comprehension:-

The visualization tools are created for OOPs. Both inspection and visualization tools may have features that can help to support cognitive strategies for program and code comprehension.

Note :-
Both inspection and visualisation tools may have features that can help to support cognitive strategies for program comprehension. Each tool

will compared to the criteria defined by Linos to see if they offer comprehensive facilities.

(i) 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.

Note:- This inspection process is developed by Fagan[14] in 1972 and then, updated by himself in 1986[15].

VI. *Inspection Tools and their features*:-

Sl. NO.	**NAME**	**TYPE**	**FEATURES**
1	ASSIST (Asynchronous or,Synchronous Software Inspection Tool) [63],[64]	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
3	ICICLE (Intelligent Code Inspection in a C Language Environment) [70]	Individual	It supports mainly C language constructs through two phase inspection like ; individual inspection and meeting.
4	Collaborative Software Inspection (CSI) [65]	Distributed	It provides an online inspection environment by favouring 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.

(ii) Visualization Tools:-

The visualization tools are created for OOPs. It acts as an interface between two powerful information processing systems i.e.

- The Human Mind.
- The Modern Computer.

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.

Note:- This Visualisation process is developed by Gershon et al.[17] in 1972 and then, updated by himself in 1986[15].

Visualization Tools and their features:-

Sl. NO.	**NAME**	**FEATURES**
1	EasyCODE (C++) [72]	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 to select from several OO methods. It includes unified method, Runbaugh 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 + [73]	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 [56]	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! [68]	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

VII. SCOPE OF RESEARCH:-

The cognitive models of software code comprehension and debugging imposes several questions from which we selected a few of these questions for experimental investigation.

VIII. 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.

REFERENCES:-

[1] Aho A.V., Sethi R., and Ullman J.D. 2000. Compilers : Principals, Techniques and Tools. Addison – Wesley. BallT. And Eick S.G. 1996. Software Visualization in the large. *IEEE Computer 29(4):* 33-43.

[2] Creswell J.W. 1994 , *Research Design, Qualitative and Quantitative Approaches,* SAGE Publications.

[3] Brooks F.P. 1987. No Silver Bullet :Essence and accidents of software engineering. *Computer,* 20(4):10-19

[4] Brooks R. 1983. Towards a thoery of the comprehension of computer programs. *International Journal on Man-Machine Studies, 18,* 543-554

[5] A.S. Rao and M.P. Georgeff(1993) "A model-theoretic approach to the verification of situated reasoning systems" Proc. Of the 13th International Joint Conference on Artificial Intelligence, 318-324, Chambery, France.

[6] Blackwell, A., Jansen, A., & Marriott, K. (2000). Restricted Focus Viewer: A Tool for Tracking Visual Attention. In M. Anderson, P. Cheng, & V. Haarslev, Theory and Application of Diagrams (pp. 575-588).

[7] Brooks, R. (1983). Towards a Theory of the Comprehension of Computer Programs. International Journal Man-Machine Studies , 18, 543-554.

[8] Busemeyer, J. R., & Diederich, A. (2010). Cognitive modeling. Los Angeles: Sage.

[9] Carney, R., & Levin, J. (2002). Pictorial Illustrations Still Improve Students' Learning from Text. Educational Psychology Review , 14 (1), 5-26.

[10] Cheng, P.-H., Lowe, R. K., & Scaife, M. (2001). Cognitive Science Approaches to Understanding Diagrammatic Representations. Artificial Intelligence Rev. , 15, 79-94.

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

[12] Crane, H. D. (1994). The Purkinje image eyetracker, image stabilization, and related forms of stimulus manipulation. Visual science and engineering: Models and applications , 15-89.

[13] Cross, J. H., Hendrix, D., Umphress, D., Barowski, L., Jain, J., & Montgomery, L. (2009). Robust Generation of Dynamic Data Structure Visualizations with Multiple Interaction Approaches. ACM Transactions on Computing Education , 9 (2).

[14] Cutrell, E., & Guan, Z. (2007). What are you looking for?: an eye-tracking study of information usage in web search. ACM conference on Human factors in computing systems, (pp. 407-416). New York, NY, USA.

[15] Ducassé, M., & Emde, A. -M. (1988). A review of automated debugging systems: Knowledge, strategies and techniques. International Conference of Software Engineering, (pp. 162–171). Singapore.

[16] Duchowski, A. (2007). Eye tracking methodology: Theory and practice (Second ed.). Springer.

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

[18] Romero, P., Cox, R., du Boulay, B., & Lutz, R. (2002a). Visual attention and representation switching during Java program debugging: a study using the Restricted Focus Viewer. Diagrammatic Representation and Inference: Second International Conference, Diagrams (pp. 221-235). Callaway Gardens, GA, USA: Springer Verlag.

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

[20] Romero, P., Lutz, R., Cox, R., & Du Boulay, B. (2002b). Co-ordination of multiple external representations during Java program debugging. Empirical Studies of Programmers symposium of the IEEE Human Centric Computing Languages and Environments Symposia. Arlington, VA.

[21] Schnotz, W., & Bannert, M. (2003). Construction and interference in learning from multiple

representation. Learning and Instruction. (13), 141–156.

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

[23] Shah, P., & Carpenter, P. (1995). Conceptual limitations in comprehending line graphs. Journal of Experimental Psychology , 124, 337-370.

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

[25] Shah, P., Mayer, R. E., & Hegarty, M. (1999). Graphs as aids to knowledge construction. Journal of Educational Psychology , 91, 690-702.

[26] Shneiderman, B., & Mayer, R. (1979). Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. International Journal of Computer and Information Sciences , 8 (3), 219-238.

[27] Sime, J. (1996). An investigation into teaching and assesment of qualitative knowledge in engineering. European Conference on Artificial Intelligence on Education , 240-246.

[28] Soloway, E., Adelson, B., & Ehrlich, K. (1988). Knowledge and Processes in the Comprehension of Computer Programs. The Nature of Expertise , 129-152.

[29] Soloway, E., Lampert, R., Letovsky, S., Littman, D., & Pinto, J. (1988, November 31). Designing documentation to compensate for delocalized plans. Communications ACM , pp. 1259-1267.

[30] Green, T. R. (1989). Cognitive dimensions of notations. People and Computers .

[31] Carney, R., & Levin, J. (2002). Pictorial Illustrations Still Improve Students' Learning from Text. Educational Psychology Review , 14 (1), 5-26.

[32] Cheng, P.-H., Lowe, R. K., & Scaife, M. (2001). Cognitive Science Approaches to Understanding Diagrammatic Representations. Artificial Intelligence Rev. , 15, 79-94.

[33] Gernsbacher, M. A., Varner, K. R., & Faust, M. (1990). Investigating differences in general comprehension skill. Journal of Experimental Psychology: Learning, Memory, and Cognition (16), 430-445.

[34] Gilmore, D. J. (1991). Models of debugging. Acta Psychologica , 78 (1-3), 151-172.

[35] Grubb, P., & Takang, A. (2003). Software Maintenance: Concepts and Practice. Singapore: World Scientific Publishing.

[36] Johnson-Laird, P. N. (1983). Mental Models: towards a cognitive science of language, inferences and consciousness. Cambridge: Cambridge University Press.

[37] Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension:. Psychological Review (98), 122–149.

[38] Katz, I., & Anderson, J. (1987). Debugging: An analysis of bug location strategies. Human-Computer Interaction , 3 (4), 351–399.

[39] Kintsch, W., & Van Dijk, T. A. (1975). Comment on se rappelle et on resume des histoires,. In Language (40), 98-116.

[40] Mautone, P. D., & Mayer, R. E. (2007). Cognitive aids for guiding graph comprehension. Journal of Educational Psychology , 99 (3), 640-652.

[41] Mayer, R. E. (1996). Learning strategies for making sense out of expository text: The SOI model for guiding three cognitive processes in knowledge construction. Educational Psychology Review (8), 357-371.

[42] Narayanan, H., & Hegarty, M. (1998). On designing comprehensible interactive hypermedia manuals. International Journal of Human Computer Studies , 48 (2).

[43] Narayanan, N. H., & Hegarty, M. (2002). Multimedia design for communication of dynamic information. International Journal of Human Computer Studies. , 57, 279-315.

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

[45] Pennington, N. (1987a). Comprehension strategies in programming. Comprehension strategies in programming , 100 – 113.

[46] Pennington, N. (1987b). Stimulus Structures and Mental Representations in Expert Comprehension of Computer

a. Programs. Cognitive Psychology , 19, 295-341.

[47] Romero, P., Cox, R., du Boulay, B., & Lutz, R. (2003a). A survey of representations employed in object-oriented programming environments. Journal of Visual Languages and Computing , 14 (5), 387-419.

[48] Soloway, E., Adelson, B., & Ehrlich, K. (1988). Knowledge and Processes in the Comprehension of Computer Programs. The Nature of Expertise , 129-152.

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

[50] Van Oostendorp, H., & Goldman, S. R. (1998). The Construction of Mental Representations During Reading. Mahwah, N.J.: L. Erlbaum Associates.

[51] G. Canfora,L. Mancini and M. Tortorella, *A workbench for program comprehension during software maintenance* , 4th Workshop on Program Comprehension, IEEE Computer Society Press, pp. 30-39, 1996.

[52] J.W. Gintell, J.Arnold , M. Houde, J. Kruszelnicki, R. McKenney and G. Memmi, Scrutiny : A Collaborative Inspection and Review System, in *Proceedings of the Fourth European Software Engineering Conference*, Garwisch – Partenkirchen, Germany, September 1993.

[53] L. Harjumaa and I. Tervonen, A WWW –based Tool for Software Inspection, *in 31st Hawaii International Conference on Systems Sciences*, Volume III , pp. 379-388, 1998.

[54] IBM, *Jinsight*, http://www.alphaworks.ibm.com

[55] D.F. Jerding, *ISVis*, http://www.cc.gatech.edu/morale/tools/isvis/isvis.html.

[56] P.K. Linos, *A Preliminary Report on Program Comprehension Tools (PCT's)* , Tennessee Technological University , http://www.csc.tntech.edu/~linos/pcts.html.

[57] D.C. Dennett (1987) "The Intentional Stance" The MIT Press.

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

[59] M. Wooldridge (1997) "Agent –Based Software Engineering " IEEE Proc. On Software Engineering, 144(1) 26-37.

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

[61] F. Macdonald, J. Miller, A. Brooks, M. Roper and M. Wood. A Review of Tool Support for Software Inspection, In *Proceedings of the Seventh International Workshop on Computer Aided Software Engineering*, pages 340-349, July 1995.

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

[63] F. Macdonald and J. Miller, A Software Inspection Process Definition Language and Prototype Support Tool, *Software Testing, Verification and Reliability*, Vol. 7, No. 2, pp. 99-128, June 1997.

[64] V. Mashayekhi, J.M. Drake, W. Tsai and J. Riedl, Distributed Collaborative Software Inspection, *IEEE Software* , Vol. 10, No. 5, September 1993.

[65] A. Von Mayrhauser and A. Marie Vans, Program Comprehension During Software Maintenance and Evolution, *IEEE Computer*, Vol. 28, No. 8, August 1995.

[66] MicroGold Software Inc., *With Class 98*, http://www.microgold.com.

[67] Objective Software Technology Ltd., Dynamic Visualization of Programs written in C++ , http://www.objectivesoft.com.

[68] D.J. Robson, K.H. Bennett, B.J. Cornelius and M. Munro, Approaches to Program Comprehension, *Journal of Systems Software*, Vol. 14, No. 2, pp. 79-84, February 1991.

[69] V. Sembugamoorthy and L. R. Brothers, ICICLE: Intelligent Code Inspection in a C Language Environment. In *Proceedings of the 14th Annual Computer Software and Applications Conference,* pages 146-154, October 1990.

[70] B. Shneiderman, Software Psychology : Human Factors in Computer and Information Systems, Wintrop Publishers, Inc.,1980.

[71] Siemens AG Austria, EasyCODE, http://www.siemens.at/~easy/easy/en/easycode.htm .

[72] SNiFF+, Release 2.4, User's Guide, TakeFive Software, http://www.takefive.com , January 27th , 1998.

[73] P. Young, Program Comprehension, Visualisation Research Group, Center for Software Maintenance, University of Durham, 1996.