# Expert Maintainers' Strategies and Needs When Understanding Software: A Case Study Approach

Christos Tjortjis and Paul Layzell
Department of Computation, UMIST
P.O. Box 88, Manchester, M60 1QD, UK
Email: {christos, pjl}@co.umist.ac.uk

## Abstract

*Accelerating the learning curve of software maintainers working on systems with which they have little familiarity motivated this study. A working hypothesis was that automated methods are needed to provide a fast, rough grasp of a system, to enable practitioners not familiar with it, to commence maintenance with a level of confidence as if they had this familiarity.*

*Expert maintainers were interviewed regarding their strategies and information needs to test this hypothesis. The overriding message is their need for a "starting point" when analysing code. They also need standardised, reliable and communicable information about a system as an equivalent to knowledge available only to developers or experienced maintainers.*

*These needs are addressed by the proposed "rough-cut" approach to program comprehension. Work underway assesses the suitability of using data mining techniques on data derived from source code to provide high level models of a system and module interrelationships.*

## 1. Background

Program comprehension is a demanding task comprising up to 90% of the total time spent on software maintenance [2], [21], [27], which in turn is the most expensive process in the lifetime of software [2]. This has been attributed to a plethora of problems reported in the literature, such as lack of up-to-date and precise documentation, inadequate communication, and unavailability of the original designers and programmers [11], [13].

Researchers have been trying to improve and accelerate the process of program comprehension in a number of ways. Because of the limited amount of information that a maintainer can assimilate at one time, the *as-needed* strategy suggests maintainers gain an understanding of the application while performing the change process [13]. Similarly *partial comprehension* has been proposed in many cases to be the only feasible approach when systems are large or when deadlines have to be met [5], [20], [24].

A *code comprehension model* has been put forward, through analysing cognition behaviour [16], [17], [18], constructed from three high level models: a top-down model, a program model and a situation model. Other researchers recommended that a *middle-out* approach could be more functional [31]. Finally *program plans, monitoring inter-module function calls, multiple simultaneous views* and *varying focal distance* are considered to be of value when attempting code understanding [7], [12], [19], [22], [25], [26].

These examples illustrate the diversity of approaches to program comprehension, and whilst empirical evidence justifies each one, there is very slow convergence on the best approach. Indeed, it was a apparent outcome of one of the working sessions of IWPC 2000 that: "We do neither have explicit guidelines to help us to perform a given program understanding task, nor do we have good criteria to decide how to represent knowledge derived by and used for program understanding" [3].

No commonly accepted framework exists that can be used to guide comprehension in the absence of familiarity with the code or the domain [29]. Neither there is a well-defined set of metrics for measuring properties of the code such as structure, modularity etc [15]. The danger for the research community will be to fall into the trap of trying to develop the best, single approach to program comprehension, without understanding the pragmatics of everyday maintenance and comprehension. A major research challenge therefore is to understand key objectives in the program comprehension process and to provide the most suitable support for the specific task in hand and at the time it is required.

The remainder of this paper continues with section 2 which describes the motivation and main objectives of the work presented. Section 3 discusses the investigation process and the way it was performed. Section 4 presents the initial findings of the conducted interviews. Section 5 analyses the information gathered during the interviews and its implications. Section 6 presents conclusions and further work.

0-7695-1083-3/01 $17.00 © 2001 IEEE

## 2. Motivation

Investigating ways of accelerating the learning curve of a software maintainer who needs to perform a maintenance task on a system with which they have no particular familiarity was a major motivation for this work. A key issue is therefore the maintainer's ability to quickly comprehend the software in order to make the required change.

Ongoing work seeks to bring together the technological support for maintenance with an awareness of the pragmatic, business parameters and constraints within which such tasks must be delivered. This work revealed the commercial pressures under which maintainers operate and therefore speed of understanding is decisive [11]. As part of this work, a qualitative study of expert software maintainers was undertaken, focusing on the issue of program comprehension, in an attempt to better appreciate its needs and objectives and thereby provide better tool support. This empirical study also aimed at verifying the extent of contemporary tools' deployment in the industrial world.

A working hypothesis for this study was that automated methods are needed to provide a quick, rough grasp of a software system, to enable practitioners who are not familiar with a system, to commence maintenance with a level of confidence as if they had this familiarity.

To test this hypothesis, it was first necessary to get a good understanding of what happens during program comprehension and to identify its support requirements. This involves identifying strategies employed by maintainers when dealing with a 'request for change' and related maintenance tasks. It also requires establishing methods, practices and needs existing in the industrial world and identifying areas where research can enhance these methods- in particular, to explore the potential of any possible automated methods in achieving program comprehension.

A more specific research question to be answered in this context was whether gaining a high-level understanding of a system or identifying specific issues and structures within source code by use of automated or semi-automated techniques was required by practitioners.

## 3. Study Process

Personal interviews, conducted in three steps as described in this section, was the method selected for information gathering. The first step was to identify a possible set of organisations and maintainers to be targeted. As this study was designed to be a qualitative one, diversity was the key criterion to reach well-informed views and to maximise the generic nature of conclusions. Diversity for organisations was sought in business areas, application domains, systems, platforms and programming languages used. Diversity for maintainers was sought in skills, education and degree of involvement in the distinct phases of software lifecycle.

Five organisations were selected for this study and are referred to as organisations A, B, C, D and E respectively for the sake of anonymity and brevity. Table 1 summarises key features of the organisations and profile details of the nine interviewees.

Organisation A has implemented and is currently maintaining a system for administering halls of residence. The system was developed in 1989, written in COBOL, and is still maintained by the original developers, one of whom was interviewed.

Organisation B is a large financial institution, with several software systems of variable age, implemented either in-house or externally, in a variety of languages such as COBOL, IBM Assembler, C++, JAVA, and Visual Basic. These systems are maintained by specially designated teams. Three members of a team, with experience ranging from 3 to 24 years, were interviewed together.

C is a very large multinational corporation providing software and hardware solutions. One of their largest and most complex software products is implemented in a low-level language, and is maintained by a large team. The most experienced, active member of this team was interviewed; his experience includes work in several different systems for the past 15 years.

**Table 1. A summary of selected organisations and interviewees**

| Organisation | Application Features | Interviewee |
|---|---|---|
| A. Halls administration | 11 years old, in COBOL | Original programmer and maintainer ever since |
| B. Financial Institution | 30 years old, in COBOL, JAVA, IBM Assembler, C++ and Visual Basic. | 3 people with varied experience / responsibilities |
| C. Systems Provider | Large and complex software product, in low level language | 15 years experience, supports numerous clients |
| D. Software Provider | Complex and powerful operating system implemented in a variety of in-house developed low level languages | 3 people with varied experience / responsibilities |
| E. Software House | 10 years old, financial/accountancy applications in BASIC | Original programmer and maintainer ever since |

Organisation D is a large software provider. The longest established software support team is run by one of the three people who were interviewed. This team support a complex and powerful operating system implemented in a variety of low-level languages developed in-house.

Finally organisation E is a small-medium sized software house. Two of their products are financial applications written in BASIC and have been supported for the past 10 years by the original developer, who was interviewed

A primary consideration in the selection of interviewees was to ensure that they had extensive maintenance experience, in particular, that they thoroughly understood the systems for which they were responsible. This criterion for selection was vital to the study since to provide any evidence for or against the basic hypothesis, it is necessary to understand how experts work. If successful, the strategies and processes experts employ can then be replicated through tool support and enable maintainers, without extensive system knowledge and familiarity, to operate with the same speed and effectiveness as their experienced counterparts. A focused, qualitative survey as opposed to an extensive quantitative one is well justified, given the requirement for expert maintainers working on mature applications in contrasting organisations.

The second step was to approach the management of the organisations to explain the motivation behind the study and to request them to identify experienced software maintainers, involved in any of their major systems. These maintainers would then be invited to participate in the study. In this case the interview would be arranged and a list of possible topics for discussion would be circulated for them to consider in advance and request clarifications if necessary.

This list consisted of two parts. The first part included specific software maintenance topics regarding their approach, schedule and course of action, team communication, means for predicting the necessary effort and for measuring maintainability and finally the information required for maintenance. The second part investigated the tools and methods used to facilitate program comprehension, locating information and estimating the scale and the impact of changes in code. The effectiveness of partial understanding, the use and recording of mental models, the existence and usefulness of any metrics for comprehension was also examined. The effectiveness and the desirability of automated methods and other relevant facilities to accelerate and enhance program comprehension were investigated. Further empirical evidence was explored regarding possible changes over time in their approach towards comprehension.

The third step was to conduct the interview on-site at the location where the work takes place, during face-to-face meetings allowing for clarification of both questions and answers and for further confirmation of conclusions by examining facilities, tools, documentation and getting a feel of what the activities involve. Other team members were often called upon to provide further information beyond the interviewee's own experience or recall. Shortly afterwards a list of answers and conclusions would be produced and sent back to interviewee for confirmation, additions or modifications as appropriate.

It must be stated here that given the nature of the process and the sample population selected, it is important to recognise the qualitative nature of data collection- that it relates to personal experiences and preferences- and therefore is limited in the extent to which generalisations can be made.

## 4. Results

The results of this study are presented in order of support, starting with issues that the majority of interviewees agreed, followed by ideas that were less common but still used in practice or thought to be of value. Of course one must note that the diversity of the sample in terms of individual personal differences, organisations and policies, systems, domains and level of software maturity was bound to introduce variations in perceptions and procedures. It is challenging however to identify trends and possible new needs that may have arisen but are not fully recognised by the research community.

*The influence of types of maintenance.* As systems mature, all interviewees reported reductions in the time to undertake maintenance arising from (a) elimination of most errors, (b) increased maintainer's experience and (c) increased risk that changes may destabilise the system as the impact cannot be estimated in advance. Corrections tend to take up most of the time allocated to maintenance and are usually approached by attempting to locate the point where the fix needs to be applied as soon as possible. One might regard this as a 'detail-first' strategy, in which the change is made and then, through techniques such as regression testing, the wider impact of the change is assessed. Enhancements on the other hand usually require a different strategy, where a high-level understanding of the system's functionality and modules interrelationships is pursued. Preventative maintenance was deemed rarely to occur and interviewees' only comment was that such activities should be incorporated in development, thus confirming the perceived continuous nature of development and maintenance.

*Consultation of experts.* Although formal team communication is limited, informal meetings take place in order to assist new or inexperienced members of the team seek guidance from senior experienced members, or the original designers and programmers. This model of staff familiar with the system, mentoring less familiar staff is common within all five organisations, indicating the importance of human factors and the degree of dependence of the understanding effort on the maintainer's familiarity with the system. There seems to be no high-quality substitute for experience when it comes to understanding and maintaining a system, as existing methods and tools are not effective enough and documentation tends to be unreliable. In other words there appears to exist a clear need for means of achieving comprehension equivalent to having access to the original developers of the system or to maintainers who are very familiar with it.

*Partial program comprehension.* Whilst many reports suggest very significant effort devoted to program comprehension [2], [21], [27], interviewees indicated that significantly less proportion of time was devoted to comprehension in their experience, typically around 30%. This is partly explained by the fact that interviewees were experts in the systems they maintain. However all interviewees reported that more time was needed, but they were restricted by commercial pressures and delivery dates. Thus only a partial understanding was achievable in most cases and this had to be balanced against the risk of failure in successfully completing a maintenance task.

*Team communication.* The size of projects operated in the selected organisations meant that team-based maintenance was common, although the teams were clearly loosely structured and flexible in nature. However information exchange among team members is sparse, informal and is hardly ever formally recorded. There was only one organisation where formal team meetings are held on a regular basis. Teams are perceived as primarily a management and organisational structure and have little bearing on the maintenance and program understanding process itself. This raises the issue whether communication during maintenance and program comprehension activities is inherently problematic, such as difficult to formally record, or whether maintenance and comprehension is primarily an individual task which does not benefit from interaction with peers, possibly because it is time-consuming for the value it gives.

*High level models.* High-level overviews, abstractions, sequence and localised diagrams of the system, and also some means to estimate the impact of changes is thought to be potentially useful information to facilitate comprehension confirming similar views suggested by researchers [8]. The desirability of automated methods for deriving high level abstractions and module interrelationships was confirmed in 4 out of 5 cases, in order to accelerate and enhance program comprehension. A reluctance to use existing tools was also identified, partly because of the lack of time for learning their use and partly because practitioners were not convinced that the anticipated results would justify the investment. Interviewees from 4 organisations reported that mental models of programs, i.e. high level abstractions of subsystems with related functionality and interrelationships, were implicit in their work, but are hardly ever recorded for future use, even when they are communicated to other people during meetings by use of diagrams and other supportive oral explanatory material. The need for visualising and recording mental models is also identified elsewhere [28]. In one particular case, an interviewee also mentioned his experience in users developing their own mental models of a system and the importance of cross-referencing such models with those of maintainers in order to improve communication and resolve misunderstandings.

*Continuous nature of development and maintenance.* In 3 out of 5 organisations, interviewees were involved in both the development and maintenance of systems. Their view was that development and maintenance is a continuous process, with two interviewees actually asking what was exactly meant by maintenance. This view of a continuous development and maintenance process is confirmed by other surveys [11]. Of particular relevance to future work is the issue of understanding development processes, as much as understanding maintenance processes.

*Source code comments.* Maintenance activities are not documented in 3 out of the 5 organisations, except from extensive changes which are usually reflected on user manuals. Otherwise, detailed comments in code are used to describe changes, their reason and the way they were implemented. This implies that comments in mature systems get accumulated over time and tend to reflect subsequent changes rather than the original implementation ideas. An interesting future research issue here concerns the need to extract information from comments regarding changes and relate this to known functionality of code[1].

*Identification of a starting point.* In three organisations interviewees considered that assistance in locating a starting point for subsequent searching and tracing through programs significantly accelerates

[1] For Java this need is partly addressed by *Javadoc*, a tool from Sun Microsystems for generating API documentation in HTML format from comments in source code [9].

the comprehension process. Such assistance occurred through consultation with experts in the system and maintainers' own developing experience.

*Measuring comprehensibility.* Two experts suggested that the number of key functions, the depth of subroutines, the number of branches, the number of past changes and some other subjective metrics could be used for program comprehension. Keeping track of past changes using comments, history logs or other means thus seems to influence several aspects of program comprehension.

## 5. Analysis

Much of the analysis of the responses of interviewees highlights well-known problems documented in relevant program comprehension literature. However there are a number of issues which arise that have been subjected to less consideration.

The issue of *team communication* highlights that communication within maintenance teams is weak and lacks efficient means to convey information to support maintenance tasks. Oral communication is problematic as it gives rise to misunderstandings and is difficult to keep track of. Furthermore, members of teams are not necessarily physically located in the same area nor are expected to be in the future [6], as was the case in one organisation studied; thus compounding the difficulty in communication and mutual support from maintainers with experience in a given system to those without such experience. This is an area where computer supported co-operative working (CSCW) is of particular relevance [4].

The issue of communication also arises with *mental models* of systems. Models are important factors in assisting program comprehension and the performance of software maintenance, but there are no effective mechanisms to share understanding or experiences. The problem is that they only represent snapshots of a system and can become disconnected from reality if not maintained themselves, which may account for their transient nature.

A lot of information regarding the current state and functionality of software systems is held within source code in the form of comments, and a way to retrieve and record this information in a semi-automated manner could save a lot of time in program understanding. The same also applies in extracting information from identifiers and names used within code.

A broad outcome of this study into how experienced maintainers go about the process of program comprehension is that a great deal of the process, as applied in practice, is ad hoc and largely opportunistic. There is a clear tendency to rely extensively on a maintainer's experience and this experience is hardly ever recorded or documented for future use, possessing a significant risk to the maintainability of a system.

An obvious remedy could be the introduction of a formal, standardised way to represent and record knowledge about a system's current state, its shortcomings and general maintenance needs. However, such an approach is idealistic given the commercial pressures under which maintainers must operate and the psychological issue that practitioners may not wish to publicise their expertise in a way that could undermine their position.

An alternative approach is therefore proposed, based upon our initial hypothesis: that automated methods are needed to provide a quick, rough grasp of a software system, to enable practitioners who are not familiar with a system, to commence maintenance with a level of confidence as if they had this familiarity.

This alternative approach defines and introduces the notion of a "rough-cut approach" to program comprehension in which emphasis is placed on supporting the maintainer sufficiently to start a task, with a tool providing the equivalent of an inexperienced maintainer consulting with an experienced maintainer in order to scope a problem and get started.

It was generally agreed that the most useful pieces of information to facilitate code comprehension when maintaining software are:

a. an easy to navigate multi-layered subsystem abstraction, capturing control flows and modules interrelationships providing an overview of the system and possible impact of changes

b. knowledge derived from past maintenance which can mainly be retrieved from comments

Many tools exist to support these processes, however they fail to fully address the commercial context and time pressures in which source code analysis must be conducted. Therefore an alternative approach to retrieving information about a software structure could be to apply to data mining techniques to source code, using techniques such as clustering or association rules [1], [14], [15], [20], [23], [32]. The information content of comments could also be retrieved by use of text mining techniques and mapped to code descriptions [10].

Such an approach to support program comprehension is termed "rough-cut", as its primary aim is to provide a broad contextual picture of the system under consideration, rather than a refined, detailed model. Such a broad model provides a basic roadmap by which maintainers, who lack a detailed knowledge of a system, can navigate around the code, scoping the change request and solution space in a relatively short period. This in turn will enable more detailed analysis of targeted code to be undertaken, minimising analysis and computation time.

It is clear that the picture of a system automatically retrieved from code and comments should be incremental.

Taking into account the continuous nature of development and maintenance, the fact that changes in software are not always for the better and the reality that different versions of a system can be available, it would be useful if the model to be constructed can capture this dimension as well.

## 6. Conclusions and further work

Although the results of the study presented in this paper cover a number of areas associated with software maintenance and comprehension, the overriding message is the need to provide maintainers with a clearer "starting point" when analysing code as part of a program comprehension task.

Another clear requirement is a means for providing standardised, reliable and communicable information regarding a software system as an equivalent to knowledge available only to developers or experienced maintainers. Capturing knowledge regarding past modifications emerges to be of great importance.

The "rough-cut" approach to program comprehension presents a major research challenge to provide hard evidence that automated methods can achieve what conventional methods and tools are currently able of achieving in less time and by reducing the risks involved.

Work is now currently underway to assess the suitability of using data mining techniques on data derived from source code. Subtasks include populating a database with attributes extracted from code, identifying which data mining methods are more suitable for this domain and acquiring knowledge to be reviewed by experienced maintainers. A parallel step is looking at the utilisation of natural language text stored in the form of comments by extracting meaningful interpretations to be cross-referenced with code excerpts.

The proposed approach involves representing a program as a number of entities that are grouped in clusters representing subsystems, based on their similarity. These clusters can indicate structure amongst functions and also possible interrelationships between them, in a way that the impact of changes can be predicted with an acceptable amount of uncertainty. Central issues are the specification of program entities and their attributes, similarity metrics, and clustering strategy.

A prototype clustering tool using input data extracted from C/ C++ programs of various sizes has been used for experiments. Initial results indicate that a high-level system abstraction as a number of subsystems can be achieved by clustering functions into groups depending on the use and types of parameters. Interrelationships amongst components were identified in a similar manner.

Further work involves retrieving similar information by means of mining databases populated by "low level" data extracted from COBOL source code. For instance, such "low level" data include, but are not limited to variable names, identifiers and key words. Initial exploration of this possibility resulted in database schemas that can be utilised by association rules techniques, normally used for transactional databases [30] and clustering algorithms.

## 7. Acknowledgements

The authors would like to express their thanks to all the staff and companies who contributed their time and experience to the fact finding stage of the study reported in this paper.

## 8. References

[1] N. Anquetil and T.C. Lethbridge, 'Experiments with Clustering as a Software Remodularization Method', *Proc. 6th Working Conf. Reverse Engineering* (WCRE 99), IEEE Comp. Soc. Press, 1999, pp. 235-255.

[2] L.J. Arthur, *Software Evolution. The software maintenance challenge*, John Wiley & Sons, Inc., 1987.

[3] F. Balmas, H. Wertz and J. Singer, 'Understanding Program Understanding', *Proc. 8th Int'l Workshop Program Comprehension* (IWPC 00), IEEE Comp. Soc. Press, 2000, pp. 256.

[4] D. Coleman, *Groupware- Collaborative Strategies for Corporate LANs and Intranets*, Prentice Hall, San Francisco, 1997.

[5] K. Erdös and H.M. Sneed, 'Partial Comprehension of Complex Programs (enough to perform maintenance)', *Proc 6th Int'l Workshop Program Comprehension* (IWPC 98), IEEE Comp. Soc. Press, 1998, pp. 98-105.

[6] A. French and P.J. Layzell, 'A Study of Communication and Cooperation in Distributed Software Project Teams', *Int'l Conf. Software Maintenance* (ICSM 98), IEEE Comp. Soc. Press, 1998, pp.146-155.

[7] N. M. Goldman, 'Smiley—An Interactive Tool for Monitoring Inter-Module Function Calls', *Proc. 8th Int'l Workshop Program Comprehension* (IWPC 00), IEEE Comp. Soc. Press, 2000, pp.109-118.

[8] Y. K. Jang, H. S. Chae, Y. R. Kwon, and D. H. Bae, 'Change Impact Analysis for A Class Hierarchy', *Proc. Asia Pacific Software Engineering Conf.* (APSEC'98), IEEE Comp. Soc. Press, Dec. 1998, pp. 304-311.

[9] Javadoc Tool Home Page http://java.sun.com/j2se/javadoc/index.html (last accessed 8/01)

[10] H. Karanikas, C. Tjortjis, B. Theodoulidis, 'An Approach to Text Mining using Information extraction', *Proc. Workshop Knowledge Management Theory Applications* (KMTA 00), Lyon, Sept. 2000.

[11] P.J. Layzell and L. Macaulay, 'An Investigation into Software Maintenance – Perception Practices', *Journal of Software Maintenance and Practice*, vol.6, no.3, June 1994, pp 105-120.

[12] S. Letovsky, and E. Soloway, 'De-localized Plans and Program Comprehension', *IEEE Software*, vol. 39, no. 3, May 1986, pp. 41-49.

[13] D.C. Littman, J. Pinto, S. Letovsky, and E. Soloway, 'Mental Models and Software Maintenance', *Empirical Studies of Programmers*, Eds. Soloway and Iyengar, c fl1986, Ablex Publishing Corporation, pp. 80-98.

[14] S. Mancoridis, B.S. Mitchell, Y. Chen and E.R. Gansner, 'Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures', *Proc. Int'l Conf. Software Maintenance* (ICSM 99), IEEE Comp. Soc. Press, 1998, pp. 50-59.

[15] S. Mancoridis, B.S. Mitchell, C. Rorres, Y. Chen and E.R. Gansner, 'Using Automatic Clustering to Produce High-Level System Organisations of Source Code', *Proc. 6th Int'l Workshop Program Understanding* (IWPC 98), IEEE Comp. Soc. Press, 1998, pp. 45-53.

[16] A. Von Mayrhauser, A.M. Vans, 'From Program Comprehension to Tool Requirements for an Industrial Environment', *Proc. 2nd Int'l Workshop Program Comprehension* (IWPC 93), IEEE Comp. Soc. Press, 1993, pp. 78-86.

[17] A. Von Mayrhauser and A.M. Vans, 'Program Comprehension During Software Maintenance and Evolution', *IEEE Computer*, vol. 28, no. 8, Aug. 1995, pp. 44-55.

[18] A. Von Mayrhauser and A.M. Vans, 'Program Understanding Behavior During Adaptation of Large Scale Software', *Proc. 6th Int'l Workshop Program Comprehension* (IWPC 98), IEEE Comp. Soc. Press, 1998, pp.164-172.

[19] R. T. Mittermeir, 'Comprehending by varying Focal Distance', *Proc. 8th Int'l Workshop Program Comprehension* (IWPC 00), IEEE Comp. Soc. Press, 2000, pp. 3-4.

[20] C. Montes de Oca and D.L Carver, 'Identification of Data Cohesive Subsystems Using Data Mining Techniques', *Proc. Int'l Conf. Software Maintenance* (ICSM 98), IEEE Comp. Soc. Press, 1998, pp.16-23.

[21] T.M. Pigoski, *Practical Software Maintenance: best practices for managing your software investment*, John Wiley & Sons, New York, NY., 1996.

[22] J. Sajaniemi, 'Program Comprehension through Multiple Simultaneous Views: A Session with VinEd', *Proc. 8th Int'l Workshop Program Comprehension* (IWPC 00), IEEE Comp. Soc. Press, 2000, pp.99-108.

[23] K. Sartipi, K. Kontogiannis and F. Mavaddat, 'Architectural Design Recovery Using Data Mining Techniques', *Proc. 2nd European Working Conf. Software Maintenance Reengineering* (CSMR 2000), IEEE Comp. Soc. Press, 2000, pp. 129-140.

[24] H. Sneed and T. Dombovari, 'Comprehending a Complex, Distributed, Object-Oriented Software System: A Report from the Field', *Proc. 7th Int'l Workshop Program Comprehension* (IWPC 99), IEEE Comp. Soc. Press, 1999, pp. 218-225.

[25] E. Soloway, and K. Ehrlich, 'Empirical Studies of Programming Knowledge', *IEEE Transactions on Software Engineering*, vol. 10, no. 5, Sept.1984, pp. 595-609.

[26] E. Soloway, 'Learning to Program = Learning to Construct Mechanism and Explanations', *Communication of the ACM*, vol. 29, no. 9, June 1986, pp. 850-858.

[27] T.A. Standish, 'An Essay on Software Reuse', *IEEE Transactions on Software Engineering*, vol. 10, no. 5, Sept. 1984, pp. 494-497.

[28] M.-A.D. Storey and F.D. Fracchia, 'Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization', *Proc. 5th Int'l Workshop Program Comprehension* (IWPC 97), IEEE Comp. Soc. Press, 1997, pp. 17-28.

[29] S.R. Tilley, S. Paul, and D.B. Smith, 'Towards a Framework for Program Understanding', *Proc. 4th Int'l Workshop Program Comprehension* (IWPC 96), IEEE Comp. Soc. Press, 1996, pp. 19-28.

[30] H. Toivonen, 'Sampling Large Databases for Association Rules', *Proc. 22nd Int'l Conf. Very Large DataBases* (VLDB 96), 1996, pp. 134-145.

[31] M. Ward, 'Language Oriented Programming', *Software - Concepts and Tools*, no. 15, 1994.

[32] T. A. Wiggerts, 'Using Clustering Algorithms in Legacy Systems Remodularization', *Proc. 4th Working Conf. Reverse Engineering* (WCRE 97), IEEE Comp. Soc. Press, 1997, pp 33-43.