A Knowledge Transaction Approach for the Request Lifecycle in Application Maintenance

Himanshu Tyagi
Software Engineering and Technology Labs
Infosys Technologies Limited
Manikonda Village, Lingampally
Rangareddy District
Hyderabad, India
91- 9703016128
Himanshu_Tyagi@infosys.com

Kapil Shinde
Software Engineering and Technology Labs
Infosys Technologies Limited
Plot no 1, Rajiv Gandhi InfoTech Park
Phase I Hinjawadi, Taluka Mulshi
Pune, India
91-9881135508
Kapil_Shinde@infosys.com

ABSTRACT

Software engineering, especially software maintenance, is a knowledge intensive task. Detailed studies in the past have delved on - program understanding, comprehension mental models, application domain knowledge and maintenance ontology as some of the ways in which knowledge can be effectively represented. Significant insights have been extracted through these studies but they remain discrete. Our study puts Request for Change/problem ticket raised by a business user at the center stage and dwells upon the interplay of different knowledge elements in the lifecycle, under the request context. By treating the request lifecycle as a transaction, forming a complete unit of work, knowledge accumulation and reuse is demonstrated. We also discuss the results of a survey of the software maintenance practitioner community. We proposed and implemented a solution, where we demonstrated, through the interplay of the problems tickets, application domain knowledge, application code and test assets, a maintenance process which enables creation and leveraging of multi-dimensional knowledge on-the-fly.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments – Interactive Environments.

D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement – Documentation

General Terms

Management, Documentation, Experimentation

Keywords

Application Maintenance, Application Domain, Knowledge Reuse, Maintenance Request Lifecycle, Knowledge Repository

1. INTRODUCTION

Application maintenance as a knowledge intensive activity is very well recognized. Studies report that 40% to 60% of the software maintenance effort is devoted to understanding the system [6, 7]. The knowledge aspect in software maintenance has been studied from multiple perspectives. A significant work in this area has been focused on understanding the comprehension process of application code by the maintenance engineer. Work on the program understanding aspect [11, 12] highlights the multi-level switching approach between program, situation and top down models. It also reveals that the current practice of documentation and coding does not encourage understanding as it puts the knowledge in silos and does not support the cognitive needs.

Additionally, in the program comprehension studies [1], knowledge is classified in 3 domains: Domain knowledge, FORTRAN/language knowledge and programming knowledge. A wider scope study by Dias et al [5] has been on devising ontology for maintenance: System Sub-ontology, skills sub-ontology, modification process sub-ontology, organizational structure sub-ontology and application domain sub-ontology.
There have been contradicting studies on the usefulness of domain knowledge. One of the studies [9] points out low reuse of domain knowledge. But in another study [10] the role of domain knowledge in program understanding is deeply emphasized. The interesting aspect here is the emphasis on having a relation between the why (domain knowledge) and what (program implementation) and the duo forming a coherent whole [10].

The work in the above mentioned studies has been significant and encapsulates the knowledge needs for maintenance well. We, with the experience of working on application maintenance across diverse domains in a large IT consultancy firm, extend some of the work and offer the practitioner's perspective. Via a survey of maintenance engineers and managers working on live applications running in production, we have captured the point of view on the utility of domain knowledge for application maintenance. Targeting some of the current issues plaguing maintenance (as highlighted in section 2 below), we have also created and implemented a knowledge based solution that uses the "Request for Change" (also referred to as "maintenance request" or "request") in the change life cycle as the focal point. It then tracks and records the knowledge created in the form of domain knowledge, impacted code and test cases, within the context of the

"Request for Change". This knowledge is then made available for reuse for similar requests for change. To illustrate, an error reported by a business user in the booking creation of a domestic courier management application is taken as the request for change. The maintenance lifecycle for this request for change will span across multiple knowledge types: the domain knowledge related to booking creation, the programs and screens for creation of booking in the application and the test cases for validation of the fix carried out by maintenance team.

Each request for change is treated as a transaction which then is pushed to a database resulting in a composite knowledge repository that provides a powerful enabler for the maintenance community. In effect, it provides a platform to leverage the past experiences to solve the problem at hand.

The focus of our work has been adaptive and corrective maintenance that are carried out to meet the expected and changed requirements of the end user [4]. The work is initiated by briefly highlighting the issues with the currently prevalent methodology to execute adaptive and corrective maintenance. We then share the result of a survey carried out with the participation of twenty five practitioners involved in maintenance of applications from health care, banking, telecom, manufacturing and retail domains. This survey documents the value of domain knowledge in application maintenance, as judged by the practitioner community. We then introduce and elaborate on the "Business Process Driven Maintenance- the transaction based approach" and with the resolution of maintenance requests through the solution built on this approach; we validate the vital role of persisting multi-dimensional knowledge (maintenance request, application code, application domain and test assets) in software maintenance. Towards the end we highlight some of the problem areas that are not covered by this approach and work that needs to be done further.

2. CURRENTLY PREVALENT PRACTICES AND ISSUES

Currently application maintenance is executed through well designed quality processes and framework, supported by problem management and engineering products that support the request lifecycle from creation till closure. But despite a robust process framework, there are challenges when it comes to providing an efficient and effective solution. The robust quality process framework ensures that the maintenance lifecycle is followed consistently and repeatedly, deliverables are created, and reviews are carried out. The engineering solutions ensure that there is automation in impact analysis, in test management and in test execution. But is the solution aligned to the need of the business user and is the valuable learning, which the support team accrues while working on the request, being captured? *Why* does a user need the requested change, *what* is the business need, *where* are the program and domain connected? [10]. Are these addressed, and eventually recorded and leveraged for future use? Unfortunately, the answer is that these aspects are not well addressed.

The learning, which is combination of application domain knowledge and business rules, is absorbed effectively during the course of problem resolution, but it remains volatile and once the request lifecycle is complete, this knowledge is lost. Since program comprehension consumes more than 50% of resources of software maintenance and evolution, the knowledge thus acquired is a very valuable commodity [8]. Yet in current practice, that value is lost. Despite high value offered by the application domain knowledge, the current processes and associated tools do not provide effective and efficient mechanism for capture of this knowledge. Unstructured knowledge dominates this space, thus creating knowledge silos. This also fosters dependency on Subject Matter experts which is not a healthy practice.
Documentation and maintenance of domain knowledge is not mandated as part of software maintenance lifecycle and thus is not a focus area for the maintenance team.

Additionally, the programmer switches context between program, situation and top down models [11, 12] and thus warrants the need of a solution that provides a smooth transition between these contexts.
Before we elaborate on the solution that attempts to address some of the challenges, we share the results of the survey conducted across the application maintenance community at our organization on their viewpoint on the utility of application domain knowledge for application maintenance.

3. SURVEY FINDINGS

A survey of 25 application maintenance practitioners was conducted. The experience level of these practitioners in maintaining their respective applications ranges from 1 year to 9 years. This sample space represents diverse domains viz., insurance and health care, retail, telecom, banking, manufacturing etc with 23 out of 25 that is 92% respondents have been working on the application maintenance for more than 2 years.

A total number of 11 questions were asked covering aspects related to documentation, utility, usage and maintenance of business process knowledge as applicable in application maintenance. Apart from predefined multiple choice of answers, some subjective responses were also elicited.

The survey questions are listed below.

1) How many years have you been working on the maintenance of the current application?

2) What is the percentage of the tickets/requests (Bug fixes) that require comprehension/knowledge of the business process to fix the issue in the application code? (This is to eliminate request fixes which are non-business intensive in nature for example performance issue, change error description etc)- Corrective Maintenance

3) What is the percentage of the enhancements (Major/Minor) that require comprehension/knowledge of the business process to deliver the enhancement? (Adaptive Maintenance)

4) Please rate the following phases based on the extent of business process/functionality knowledge required to successfully execute the phase. The ratings can be repeated.(1-Always required, 2-Mostly required,3-Required,4-Partially Required,5-Never Required)

   Initial Analysis, Impact Analysis, Build (Code Fix and Unit Testing), Functional Testing

5) Please rate the following phases on extent of business process/functionality knowledge acquired/gathered/assimilated/imbibed in executing the phase. So in which of the application lifecycle phase is the

learning of business processes at its peak. The ratings can be repeated.

(1-Always required, 2-Mostly required, 3-Required, 4-Partially required, 5-Never Required)

Initial Analysis, Impact Analysis, Build/Code Fix and Unit Testing, Functional Testing, Any other

6) What is the current process/mode to document the business knowledge?

7) Currently, which is the most often used source of business knowledge for you?

8) Is documentation of business knowledge as models that are workflows and activities (business process modeling) beneficial in maintenance?

9) What is the biggest challenge in documentation of business/functional knowledge?

10) How can contribution to business knowledge be simplified?

11) What level of granularity, either through a business model or text based, would best help assimilation of business knowledge?

Key findings from the survey are listed below:

- 76% respondents believe that more than 40% bug-fix requests (i.e. corrective maintenance) require knowledge of the underlying business processes to fix the issue in the application code. Hence underscoring that business knowledge is useful even in corrective maintenance.

- Adaptive maintenance, by definition, is a manifestation of changing or changed business needs. Not surprisingly, 80% of the responses said that more than 40% of enhancement requests require knowledge of the underlying business to deliver the enhancement

- It is obvious that in the initial stages (post transition) of application maintenance (i.e. initial 1-2 yrs) , the familiarity with the application is less and hence there is a greater perceived need for business knowledge during maintenance. This is also brought out by the survey, where we observed that almost 89% of respondents from the 1-2 year experience category believed that more than 40% of adaptive maintenance work required knowledge of the underlying business process.

- Even when maintenance engineers gain familiarity with the applications with more years of working with it, the need for application domain knowledge does not seem to diminish, according to our survey. 75% of the respondents , who have been working with the application for more than 2.5 years still believed that business process knowledge was important for more than 40% adaptive maintenance activities

- When it came to the mode of documentation of the business knowledge, 84% respondents said they updated the system appreciation documents. System appreciation documents are usually created during the time of application transition and are updated often during the lifecycle.

- 60% respondents also said that the business knowledge is also documented in request level deliverables. This also emphasizes the importance of the request context in a maintenance scenario, wherein documenting the (change in) business functionality at this level of granularity is probably more relevant and useful in a maintenance scenario.

- Although the above two formats (i.e. system appreciation documents and request level deliverables) form a very useful repository of business knowledge, it must be observed that both are an unstructured knowledge format.

- During the maintenance activity, all the respondents said they depended mainly on a subject matter expert (SME) or self-investigation as their primary source of business knowledge. This, along with the presence of unstructured knowledge, highlights the human dependency for knowledge retention.

- Owing to the Service Level Agreement (SLA) driven turn-around times for maintenance activities, time constraint is the most reported (80%) reason for not documenting the (change in) business knowledge.

- 56% respondents said that that the lack of a SME hampers the business knowledge documentation. This could point to either incomplete transition or loss of business knowledge through resource churn. This is a good case for a formal, structured documentation of business knowledge in a manner and format which would be useful for maintenance.

- 36% respondents said that the lack of suitable tools is hampering the business knowledge documentation activity. This raises an important issue of how the current methods of knowledge capture are either inadequate or ineffective for maintenance.

- Considering that 80% of the respondents cited lack of time as the main reason why business knowledge is not documented, it is not surprising that almost 72% of the respondents feel that a facility which would allow 'knowledge creation on- the-go' would be most effective for business knowledge acquisition and maintenance.

- 60% respondents believe that making knowledge creation a mandatory activity would help. This suggests an inertia , which may be have been induced due to the fact that there is an utter lack of proper knowledge documentation tools for a severely time constrained activity such as application maintenance.

- Another interesting response says that knowledge maintenance should also be made a factor while estimating, so that adequate time is allocated. This gains importance in the light of the fact that 80% respondents have cited time constraint as the main reason for not documenting business knowledge.

4. BUSINESS PROCESS DRIVEN MAINTENANCE – THE TRANSACTION BASED APPROACH

4.1 Introduction

For a support engineer, working on a maintenance request (bug-fix or an enhancement), the focus is primarily centered on the piece of code to be written/amended, service level agreement(SLA) to be met and providing the right solution. For the IT department, "lights on" is the areas to be concerned of. For an end user, early access to the desired business feature is the prime demand. While business user is concerned with the availability of system to enable him to execute the intended business process correctly, IT department is concerned with understanding this need and providing a solution in shortest possible time while meeting the desired functional and non-functional requirements. With apparently different key objectives, both are committed to excellence in business.

Our approach, in business process driven maintenance, is centered on the basic theme of improving the value to the business by providing enabler to the maintenance team to perform their tasks effectively.

Considering the multi-dimensional knowledge [11, 12] and the issue of volatile knowledge [8], the solution attempts to tackle these, by treating *the problem resolution lifecycle as a "knowledge transaction"*.

Similar to the program comprehension studies [1], we have narrowed down four critical key elements, "knowledge atoms" of an application under maintenance. These *four critical elements are: Maintenance request, Application Code under maintenance, Application domain knowledge represented as process workflows and business rules; and the test asset repository*. The key in the approach is to enable documentation of this knowledge in a minimally intrusive way and connect these knowledge elements to build a navigable, dense knowledge repository. This can be enabled by providing multiple medium to "document while you browse" which implies wherever the user is, provide the user a quick means to seamlessly transfer his learning to a repository.

The solution thus enables accruing the vital domain knowledge and additionally provides navigation across the "knowledge atoms", supporting the cognitive needs of the user.

4.2 Knowledge Atoms

The four "knowledge atoms" of an application under maintenance are:

Maintenance Request/Problem – Maintenance request or problem (used interchangeably) is the starting point of the lifecycle in corrective maintenance. It carries information of what is expected by the end user to ensure business as usual and/or to cater to changed/additional business needs.

Application Code - This is the implementation of the business process rules and flow in the programming languages usually accompanied with a database at the backend.

Application domain– In the current approach, the application domain knowledge is primarily created as business processes via a process modeling solution.

Test Asset Repository- These are the test cases, to validate that the solution delivered for the maintenance request meets the business requirement.

Each day the support engineer, as part of the request examines the application code, comprehends the business rules from application code or inherits from the Subject Matter Experts, makes the code changes and runs the test cases. But all the knowledge remains tacit except for mandated deliverables.

The approach and solution to enable effective documentation and utilization of this tacit knowledge distributed across the knowledge atoms is *"Business process driven maintenance"*. This is a bottom up approach. The lifecycle of a maintenance request is treated as a *"knowledge transaction" similar to a database transaction*. This request can be treated as a transaction as there is a logical knowledge capsule that is built from the starting point when the user begins the analysis, till the user commits and releases the problem resolution. Similar to a database transaction, there are checks built-in to ensure the completeness of transaction

So, aligned to a database transaction, the widely accepted ACID (Atomicity, Consistency, Isolation, and Durability) [2, 3] property of a knowledge transaction has been implemented as follows:-

Atomicity- Following an "all or nothing" rule, the system, marks the transaction as complete once all the knowledge atoms to be created are available. So a request for change can be closed if and only if the impacted application domain, code and test assets are connected to the maintenance request.

Consistency - ensures that the knowledge repository remains in a consistent state before the start of the transaction and after the transaction is over.

Isolation- refers to the fact that till the problem resolution is complete and hence the knowledge transaction is fit to use, other transactions cannot access the same.

Durability- refers to the guarantee that once the user has completed the problem resolutions, the transaction will persist thus resulting in a repository rich with historical data.

By applying ACID property to the knowledge transaction, sanity of the knowledge, which spans across multiple knowledge atoms, has been ensured. It guarantees that the knowledge persisted in the repository is complete and hence fit to use.

4.3 Master Data and Transaction Data

This is another database concept that has been effectively deployed in the approach.

The "Master Data" comprises of application domain (business processes), test case and application code elements and their relationship, which is largely static.

The problem resolution produces transaction data which comprises of relationship between maintenance request and other knowledge atoms.

The master data and transaction data together provide transactional knowledge, which drives this approach.

4.4 Implementation Scenarios

The solution is implemented through a proprietary maintenance platform. Listed below are two problem resolution scenarios that help illustrate the solution.

"Scenario 1" is a transaction that results in creation of knowledge.

In the “Scenario 2”, the transaction reuses the knowledge created in “Scenario 1” and updates the knowledge, if necessary.

The maintenance lifecycle here has been simplified to follow the phases Request for change, Planning phase (Program comprehension, Change impact analysis), Change implementation, Verification and validation [13].

4.4.1 Scenario 1-Knowledge Generation

The transaction (problem resolution) begins with a maintenance request. The transaction owner is provided with the ability to document the expected business behavior through the modeling solution.

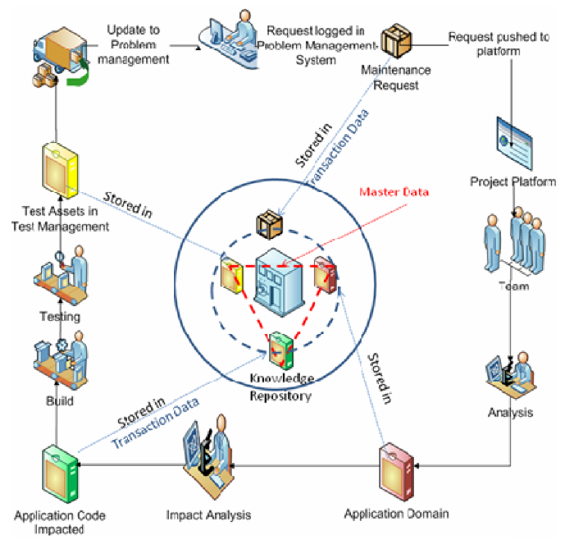
After comprehension of the problem from the application domain perspective, the user will then move ahead with identification of application code components that need to undergo change.

Then, the user will carry out build and proceed with functional testing. For this, the user will create a test plan with test case and execute, resulting in test results. The Figure 1 depicts the lifecycle as executed on the solution platform.

For the user to be able to commit the transaction (close the maintenance request), the knowledge constraints should be satisfied. A user will be allowed to close the maintenance request (commit transaction) if and only if the integrity constraints, for the transaction type, are satisfied and the repository is in a consistent state. In this scenario, establishment of a context between the application domain and the various technology components constitutes the constraint.

At the end of transaction, a knowledge mesh is created with the knowledge atoms, viz. maintenance request, application domain, application code and test cases.

The problem resolution, through a combination of tool based enablers for each of the maintenance phase and a transaction driven approach, results in the creation of a multi-dimensional knowledge linking together maintenance request, application code, application domain knowledge and test assets, hence offering a comprehensive knowledge unit for problems in the same space.



Update to Problem management
Request logged in Problem Management System
Maintenance Request
Request pushed to platform
Stored in Transaction Data
Master Data
Project Platform
Test Assets in Test Management
Stored in
Team
Testing
Knowledge Repository
Build
Stored in
Analysis
Stored in Transaction Data
Application Code Impacted
Impact Analysis
Application Domain

Figure 1. Master and transaction data in maintenance lifecycle

4.4.2 Scenario 2-Knowledge Reuse

The basic assumption here is that the knowledge created during problem resolution scenario 1 can be reused partially or completely during the scenario 2. The reusability factor might vary across maintenance landscape. The reuse is auto-determined based on matching context between the problem in scenario 2 and the context in the repository from previous transactions.

The transaction begins with the availability of maintenance request in the platform. The system then searches through the repository for knowledge that can be reused. The search result provides user a narrowed down turf for further investigation. The various knowledge atoms constitute the search results. The search can be carried out on any of the knowledge atoms mentioned earlier, since searching on one knowledge atom automatically reveals the other related knowledge atoms due to the linkages already established.

The access to historical transactions allows a user to arrive at the probable list of components. Through ready access to solution record of problems solved in past, the user can effectively leverage experiential knowledge.

The access to application domain knowledge through this approach enables the support engineer to understand the “Why” of the problem from the business user need.

By narrowing down the application code components, the user is able to narrow down on the problem area faster and deliver changes with greater accuracy.

The user can reuse this knowledge and based on the additional learning, amend and update the knowledge repository.

The user can also search the knowledge repository based on user-defined criteria and also browse though the knowledge repository.

Thus in scenario 2, the problem resolution was simplified by pin-pointing the vital knowledge atoms, ready to use resolution data, all with seamless modes of knowledge access. With domain knowledge as the driver, it helps address the vital “Why” of application maintenance [10].

4.5 Validation

The solution has been implemented as part of a Proof of Concept in a COBOL/DB2 based maintenance project. One such scenario, (keeping the real customer request context hidden) has been listed below

Request 1- Error in booking creation for hazardous cargo

1) To initiate, a critical business process, booking creation, as identified by the Subject Matter Expert, was modeled.
2) One of the requests related to the business process identified above was picked up and a context was established with the model as also with the application code and test case components.
3) The complete cycle was executed by a resource familiar with the part of the application selected for this validation. Interview with the resource revealed that via the transaction based approach creation of “knowledge atoms” was simplified, as the knowledge could be created ’on-the-fly’.

Request 2- Booking creation for domestic cargo errors out

1) For the second problem, in the same domain, a support engineer who was aware of the project set up but was not familiar with the relevant application domain was chosen.

2) This user was then asked to use the knowledge transaction based approach for resolving the problem. This user, through searching for similar requests for the starting point, identified the relevant business model to develop a better understanding of the entire problem domain. Based on the linkages created earlier for solving request 1, the affected technology components were broadly identified. To make the change, the user did have to resort to studying these components in further detail.

At the end of problem resolution, the interview with the user confirmed that the knowledge transaction approach helped in narrowing down the turf of investigation to few objects. The advantages, as highlighted by the user, were better comprehension of complete problem domain and ease in identification of affected components for investigation. The user also validated the prevention of knowledge loss acquired during analysis via this approach.

5. CHALLENGES AND FUTURE WORK

The utility of the approach is primarily limited to corrective and adaptive maintenance space. A key factor here is the extent of application domain breadth that can be covered because the creation of knowledge is limited to areas where the problems are reported. The repeatability of problem within the problem domain is also a factor for the success of this approach.

During implementation we noticed maintenance teams putting lesser emphasis on modeling the application domain and thus there is a need to educate and encourage them to invest effort in doing so. The granularity of the domain knowledge also needs to be made flexible. Since the current validation was carried out on a legacy system, in a limited way, the future work will also require carrying out more validations, for a longer period, across web technologies and open systems, for varied application sizes.

6. CONCLUSION

The usefulness of domain knowledge for application maintenance has been strongly expressed by the maintenance teams. The domain knowledge, along with other knowledge elements of maintenance request, application code and test assets forms a comprehensive knowledge unit and thus a system which allows seamless creation and usage of this knowledge as a coherent whole has been found effective for carrying out maintenance. But as a future work, there is need to establish efficient ways to document the domain knowledge and identify the granularity level for this knowledge accumulation. There is also a need to carry out extended validation across diverse application and technology domains.

7. ACKNOWLEDGMENTS

We would like to express sincere gratitude to Girish Rama, Dr. Srinivas Padmanabhuni, Haresh Khemani and Virendra Paliwal for their guidance and support and also the practitioners at our organization for sparing their valuable time to respond to the survey.

8. REFERENCES

[1] Clayton R., Rugaber S., Wills L. On the Knowledge Required to Understand a Program. Proc. Working Conference on Reverse Engineering. Oct.1998, 69–78.

[2] Gray. J. The transaction concepts: Virtues and limitations. In Proceedings of the International Conference on Very Large Data Bases, 1981, 144-154.

[3] Haerder T. and Reuter. A. Principles of transaction-oriented database recovery. ACM Computing Surveys, 15(4), December1983, 287-317.

[4] Lientz B. P., Swanson E. B. Software Maintenance Management. Addison Wesley, Reading, MA. 1980.

[5] Márcio Greyck Batista Dias, Nicolas Anquetil, Káthia Marçal de Oliveira. Organizing the knowledge used in Software Maintenance Journal of Universal Computer Science, vol. 9, no. 7 (2003), 641-658

[6] Pfleeger, S. L. Software Engineering: Theory and Practice. 2nd Edition.New- Jersey, Prentice Hall, 2001.

[7] Pigoski, T. M. Practical software maintenance: best practices for managing your software investment. John Wiley & Sons. Dec. 1996, 87-102.

[8] Rajlich V., Varadajan S., Using the Web for Software Annotations, Int. Journal of Software Engineering and Knowledge Engineering vol. 9, 1999, 55 – 72.

[9] Ramal, M. F., Meneses, R. d., and Anquetil, N. 2002. A Disturbing Result on the Knowledge Used during Software Maintenance. In Proceedings of the Ninth Working Conference on Reverse Engineering (Wcre'02) (October 29 - November 01, 2002). WCRE. IEEE Computer Society, Washington, DC, 277

[10] Rugaber, S. 2000. The use of domain knowledge in program understanding. Ann. Softw. Eng. 9, 1-4 (Jan. 2000), 143-192

[11] Vonmayrhauser, A.& Vans, A. (1993b) . From code understanding needs to reverse engineering tool capabilities. Proceedings of the 6th International Workshop on Computer-Aided Software Engineering (CASE99), Singapore. July, 1993, 230-239.

[12] Vonmayrhauser, A. & Vans, A. (1994). Comprehension processes during large scale maintenance. Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, 39-48.

[13] Yau, S.S., Collofello J.S., MacGregor T., Ripple effect analysis of software maintenance, In proceedings of Compsac, IEEE Computer Society Press, Los Alamitos, CA. 1978.40,60–65