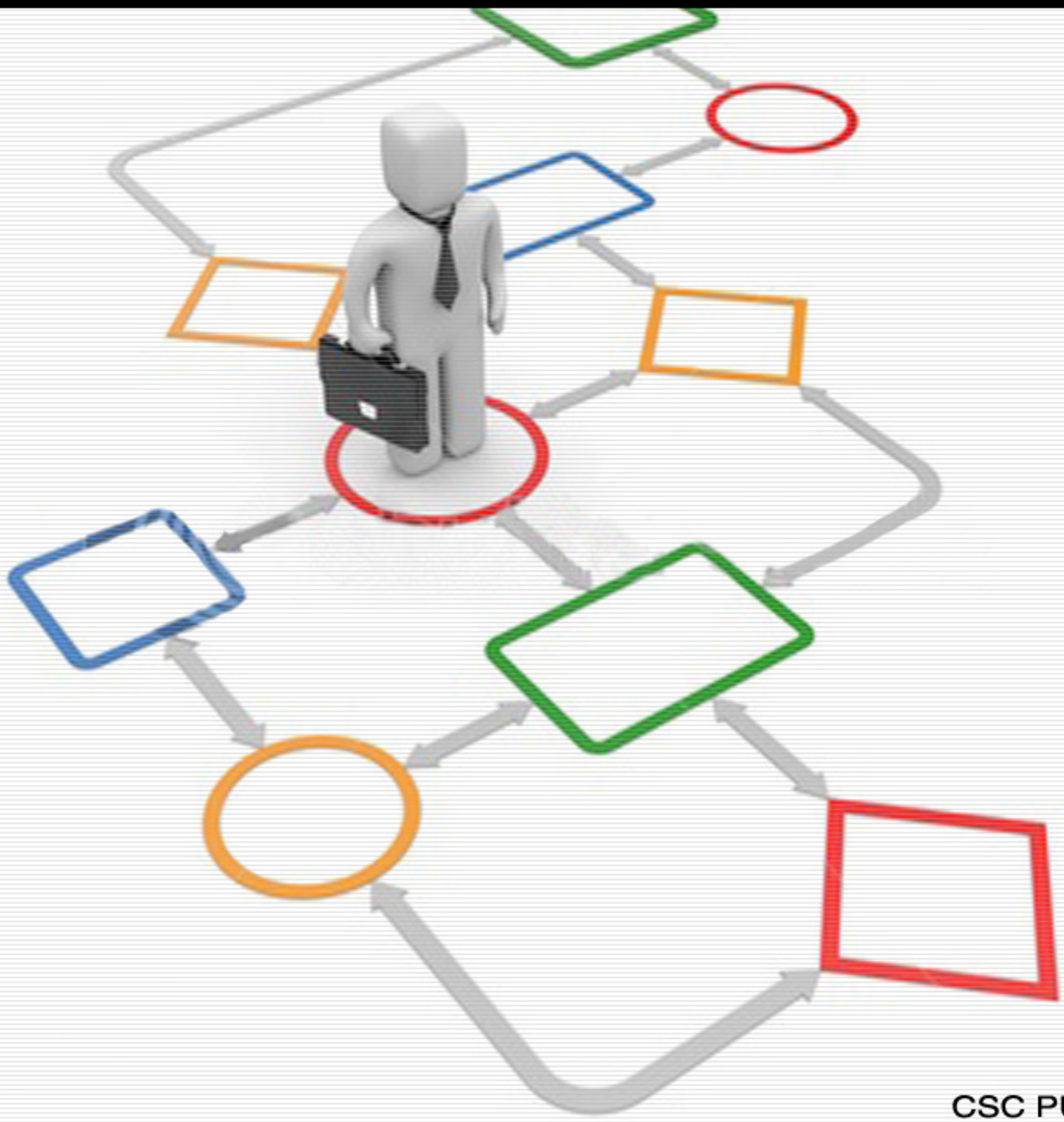
Volume 3 • Issue 2 • April 2012

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

ISSN : 2180-1320

Publication Frequency: 6 Issues / Year



CSC PUBLISHERS
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

VOLUME 3, ISSUE 2, 2012

EDITED BY
DR. NABEEL TAHIR

ISSN (Online): 2180-1320

I International Journal of Software Engineering (IJSE) is published both in traditional paper form and in Internet. This journal is published at the website http://www.cscjournals.org, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJSE Journal is a part of CSC Publishers
Computer Science Journals
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

Book: Volume 3, Issue 2, April 2012
Publishing Date: 16 - 04 - 2012
ISSN (Online): 2180-1320

IJSE Journal is a part of CSC Publishers
http://www.cscjournals.org

Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2012

EDITORIAL PREFACE

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. It is the second issue of third volume of IJSE and it is published bi-monthly, with papers being peer reviewed to high international standards.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Starting with volume 3, 2012, IJSE appears in more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering, software science, theoretical software engineering, computational intelligence, and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. Some important topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

IJSE gives the opportunity to researchers and practitioners for presenting their research, technological advances, practical problems and concerns to the software engineering. IJSE is not limited to a specific aspect of software engineering it cover all Software engineering topics. In order to position IJSE amongst the most high quality journal on computer engineering sciences, a group of highly professional scholars are serving on the editorial board. IJSE include empirical studies, requirement engineering, software architecture, software testing, formal methods, and verification.

International Editorial Board ensures that significant developments in software engineering from around the world are reflected in IJSE. The submission and publication process of manuscript done by efficient way. Readers of the IJSE will benefit from the papers presented in this issue in order to aware the recent advances in the Software engineering. International Electronic editorial and reviewer system allows for the fast publication of accepted manuscripts into issue publication of IJSE. Because we know how important it is for authors to have their work published with a minimum delay after submission of their manuscript. For that reason we continue to strive for fast decision times and minimum delays in the publication processes. Papers are indexed & abstracted with International indexers & abstractors.

EDITORIAL BOARD

TABLE OF CONTENTS

Volume 3, Issue 2, April 2012

Pages

Contributors to Reduce Maintainability Cost at the Software Implementation Phase

Mohammed Abdullah H. Al-Hagery
Faculty of Computer/Department of Computer Science,
Qassim University, Boridah, KSA

Abstract

Software maintenance is important and difficult to measure. The cost of maintenance is the most ever during the phases of software development. One of the most critical processes in software development is the reduction of software maintainability cost based on the quality of source code during design step, however, a lack of quality models and measures can help asses the quality attributes of software maintainability process. Software maintainability suffers from a number of challenges such as lack source code understanding, quality of software code, and adherence to programming standards in maintenance. This work describes model based-factors to assess the software maintenance, explains the steps followed to obtain and validate them. Such a method can be used to eliminate the software maintenance cost. The research results will enhance the quality of the source code. It will increase software understandability, eliminate maintenance time, cost, and give confidence for software reusability.

Keywords: Maintainability Time, Software Maintenance, Standard Code, Quality of lines of Code, Understandability, Maintainability Factors.

1. INTRODUCTION

Software maintenance is an important phase in the software life cycle. It focuses on keeping the software fully functional and up to date. Maintenance engineers used different approaches and methods to gain understanding of software systems so maintenance tasks can be performed effectively. A lot of efforts have been put into finding a way to measure maintainability of software [1].

Maintainability cannot be seen as an attribute of the software system alone, because it depends a great deal on who maintains it, a team that has a lot of experience with a particular system will maintain it more easily. Both the software and the team have internal attributes that influence maintainability, for example, structural complexity of the software and skill of the team members. We want to survey the factors that lead to low or high maintainability [2].

A change request can be due to a failure, changing requirements, prevention or any other reason. The activities by the maintenance team include actually performing the change, but also documenting, testing, and reporting, depending on the maintenance procedures. When a system is changed extensively a new team is formed to implement the changes that are not regarded as a change. Such a situation is more like a new system being developed [2]. There are many factors that influence maintainability can be assembled and adapted from [3], [4], [5], [6], [7], [8], [9]. Measuring the maintainability of source code revisions presents some challenges [10].

This work concentrates on quality of source code rather than code defects. Code defects are defects attributable to coding errors such as branching to a wrong location. These defects are found throughout the coding process as well as in final test of changes and enhancements to an application.

1.2 Survey of Related Works

The largest cost associated with any software product over its life-cycle is the software maintenance cost. One approach to controlling maintenance costs was to utilize software metrics during the development phase [11]. A number of studies is examining the link between Object Oriented software metrics and maintainability have found that in general these metrics

can be used as predictors of maintenance effort [12],[11],[13],[14], and [15], which can be measured in working hours.

Yuming and Hareton, presented an empirical study that sought to build object-oriented software maintainability prediction models using a novel exploratory modeling technique, MARS. To build the MARS models, they made use of the Li and Henry's data sets, UIMS and QUES, obtained from two different object-oriented systems. The prediction performances of the MARS models were assessed and compared with those of the multivariate linear regression models. These models are the artificial neural network models, the regression tree models, and the support vector models, but their focus was not on the implementation phase and data set used was not enough to prove the suggested model [16].

Mari et al. introduces the framework of maintainability and the techniques that promote maintainability in three abstraction levels; system, architecture and component. In system dimension, the maintainability requirement is considered from a business related point of view. In architecture, maintainability means a set of quality attributes e.g. extensibility and flexibility. At the component level, maintainability focuses on modifiability, reusability, integration, and testability [17]. Ardimento et al. in [18] reports the results of their empirical study aimed at understanding how characterizations of components affect the maintenance effort of the system components. They have made the assessment that:

(i) Functionality of each component should be as concentrated as possible over a single aspect of the application domain,
(ii) The training time offered by the component's producer usually indicates the complexity of understanding it and if a component is difficult to understand, then it is difficult to maintain; and
(iii) A deep knowledge of the component is necessary for the organization before its adoption.

Van Koten and Gray, make the first use of the BBNs in building software maintainability prediction models. They use a special type of Bayesian networks called Naïve–Bayes classifier, which assumes no expert knowledge about the prior probability distribution but learns it from data by batch learning. The results show that the prediction accuracy of the BBN model is more accurate than regression-based models for one system but is less accurate than regression-based models for another system. Accurate software metrics-based maintainability prediction is desirable first because it reduces future maintenance efforts by enabling developers to better identify the determinants of software quality and thereby improve design or coding, and second because it provides managers with information for more effectively planning the use of valuable resources. Although a number of maintainability prediction models have been developed in last decade, they have low prediction accuracies according to the criteria suggested in [15], [19].

Maintainability metrics are commonly language dependent, and computing them requires tools that typically assume access to the full definitions of the software entities [10]. It was found that a number of metrics such as the lines of code changed, and the number of operators changed are strongly correlated to maintenance efforts [1]. Heitlager et al. discussed several problems with the maintainability index (MI), and they identified a number of requirements to be fulfilled by a maintainability model to be usable in practice. they sketched a maintainability model that alleviates most of these problems, and discussed their experiences with using such as system for IT management consultancy activities [20].

Bertoa et al. have been reported that they presented a set of measures to assess the maintainability of software components. Furthermore, they described the process followed to obtain and validate them. Such a process can be maintained for defining and validating measures for other quality characteristics [21]. Wu et al. proposed a technique for maintaining evolving component based system by utilizing a static analysis to identify the interfaces, events and dependence relationship that would be affected by the modification in the maintenance activity [22], [23]. The maintainability of a software system can be measured in different ways. Currently and in past studies, maintainability has been defined as ‘‘time required to make changes’’ and ‘‘time to understand, develop, and implement modification’’[24]. As well as, Yuming and Hareton measured the maintainability of a software system as the number of changes made to code during a maintenance period. They employed a novel exploratory

modeling technique, multiple adaptive regression splines (MARS), for building maintainability prediction models using the metric data collected from two different object-oriented systems [16].

1.3 Motivations and Objective

One of the most critical processes in software development is the reduction of software maintainability cost accordingly the quality of code design, however, a lack of quality models and metrics can help asses the software maintainability process. Software maintainability suffers from many challenges such as lack of source code quality, and source code understanding, adherence to programming standards in maintenance. The main objective of this work is to define and establish a Criteria-Based-Model that can be used to assess S/W quality characteristics, and that can assist in implementation phase. Such criteria could reduce the maintenance cost; these criteria will be created as three or one group. This objective can be detailed in the following points:

1. Create a group of criteria that support writing a standard software programs(proposed criteria)
2. Construction of a mathematical model for applying the proposed criteria to reduce the final S/W cost.
3. Increase S/W understandability, readability and flexibility.
4. Participation of undergraduate students in the research work through the formation of work groups to study the code standardization, to write some programs and then execute software maintenance on several software programs. These programs help to ensure acceptance of the model and the proposed factors or criteria.

2. SOFTWARE MEASUREMENT

Software measures can be classified into three types; derived measures, base measures, and indicators. Base measures do not depend upon any other measure (e.g., the number of tables in the manuals). A derived measure is derived from other base or derived measures (e.g., the ratio of methods per interface). An indicator is a measure that is derived from other measures using an analysis model according to decision criteria. The objective of that is to obtain a measurement result that satisfies an information need (e.g., the size of a sub-system is “medium” if it has more than 30 assemblies, provides more than 45 interfaces, and its manuals have more than 7,000 Line of Code (LOC).

Measures relate a defined measurement approach and a measurement scale. A measurement approach is the logical sequence of operations, described generally, used in quantifying an attribute with respect to a specified scale [25]. A measure is expressed in units, and can be defined for more than one attribute. Examples of measures for software component attributes include the number of provided interfaces, the ratio of methods per required interface, or the throughput of video frames emitted per input video frame (they correspond, respectively, to possible measures for the aforementioned attributes size, interface complexity, and performance)[21].

Accurate software metrics-based maintainability prediction can not only enable developers to better identify the determinants of software quality and thus help them improve design or coding, it can also provide managers with useful information to help them plan the use of valuable resources[16].
The act of measuring software is a measurement, which can be defined as the set of operations that aims at determining a value of a measurement result, for a given attribute of an entity, using a measurement approach [21].

The term metric is not present in the measurement terminology of any other engineering disciplines, at least with the meaning it is commonly used in software measurement. Therefore, the use of the term “software metric” seems to be imprecise, while the term “software measure” `seems to be more appropriate to represent this concept. Accordingly, in the following the term measure will be used. This is also consistent with ISO/IEC and IEEE Computer Society positions which, in order to ensure both consensus and consistency with other fields of sciences, made a decision in the year 2002 to align their terminologies on measurement with the internationally accepted standards in this field. In particular, ISO-JTC1-SC7 is trying to

follow as much as possible the ISO international vocabulary of basic and general terms on metrology [26]. A number of software metrics measuring maintainability has been proposed by means of theoretical and empirical studies. However, component based system presents a unique maintenance challenges. Unlike the traditional software systems, one cannot be done by viewing or changing the source codes of the component, but are restricted to reconfiguring and reintegrating components [27].

3. MAINTAINABILITY

Maintainability [28] is "The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment". The seminal research work by Basili and Turne in 1975 has identified different characteristics of software system that effect software maintainability. Effective maintenance involves detailed observations of the behavior of a system and is driven by software complexity [29]. Voas in 1998 provided an overview of the maintenance challenges raised by Component Based Software Development by identifying reasons including frozen functionality, incompatible upgrades, unreliable components and complex middleware [27].

The "understandability" of a source code is related directly to the maintainability, because understandability is one of the dominant factors affecting software maintainability [30]. For example, let us assume a perfect source code that does not have any faults or logical errors. Nevertheless, if a source code is difficult to understand, an increase of costs and/or of failure potential during maintenance is then inevitable. Several factors such as complex logic, the many variables included in a code and lengthy codes could interfere with the understanding of the program context by maintenance personnel [31].

3.1 Maintainability Attributes

The software maintainability affects by a number of criteria such as: understandability, reusability, learnability, readability, and operability. It can be defined as follow:

- Understandability: the capability of the component to enable the user to understand whether the component is suitable, how it can be used for particular tasks and conditions of use. System developers should be able to select a component suitable for their intended use, for example, component elements (e.g. interfaces, operations) should be easy to understand [21].
- Reusability: the capability of the software to enable the developer or the maintainer to modify its functions easily.
- Readability: the ability of the software to enable the developer or maintainer to understand the software functions by reading its lines of source code.
- Learnability: the capability of the software component to enable the user or system developer to learn its application. For example, the user manual and the help system should be completed, the help should be context sensitive and explain how to perform common tasks, etc.
- Operability: the capability of the software component to enable the user (system developer) to operate and control it. An Operability measure should be able to assess whether system developers can easily operate and control the component. Operability measures can be categorized by the dialogue principles described in ISO/IEC-9241-10 [21]. Figure 1 illustrates the relation between maintainability and source of code.

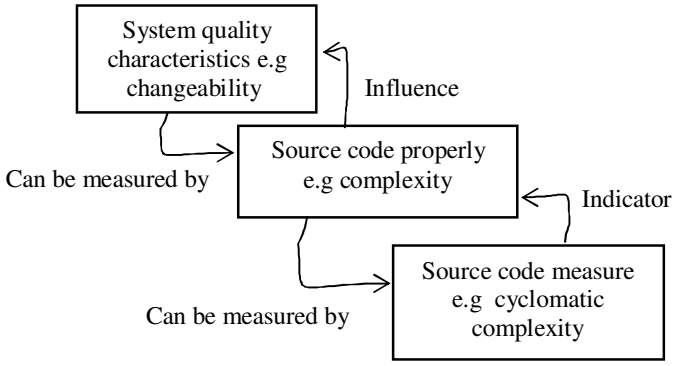


FIGURE 1: The relation between maintainability and source of code

3.2 Factors Affecting Maintainability

In [32], four main factors for software maintainability are included in ISO/IEC 9126 such as analyzability, changeability, stability, and testability. These factors are defined clearly in [33]. First, analyzability is attributes of software that bear on the effort needed to diagnose of deficiencies or causes of failure and to identify parts to be modified. Second, changeability is some attributes of software that bear on the effort needed to make modifications, eliminate faults or change the system in response to environmental change. Third, stability that can be represented by attributes of software that bear on the risks associated with unexpected effects of modifications. Finally, Testability attributes of software that bear on the effort needed to validate modifications.

Studies that take the application development view of software seem to address maintenance as an afterthought of development rather than a critical and expensive part of the total system life-cycle. For example, Dekleva [34] evaluates how the choice of development approach will influence maintenance. Perry in [35] addresses maintenance quality in the context of development quality. The maintenance phase of the life-cycle is a natural and necessary part of the system operation [36]. Software evolves over time primarily due to changes in requirements and technologies. As a result, Information systems development is typically acknowledged as an expensive and lengthy process, often producing code that is of uneven quality and difficult to maintain. Software reuse has been advocated as a means of revolutionizing this process. The claimed benefits from software reuse are reduction in development cost and time, improvement in software quality, increase in programmer productivity, and improvement in maintainability [37]. Prasanth et al., proposed a model for improving software maintainability based on risk analysis, they identified a set of metrics that affects the external and internal complexity [38].

4. QUALITY OF SOURCE CODE

There are two main types of software quality, Quality of process and quality of products. In general, there is a lack of consensus about how to define and categorize software quality characteristics. Quality of system documentation includes quality of external documentation and quality of internal documentation [39].

The development of high-quality software must satisfy both the users' requirements and the software firm's budget [40]. Program restructuring is a key method for improving the quality of ill-structured programs, thereby increasing the understandability and reducing the maintenance cost [41]. Our concentration is on some important rules of code design. Quality is one of the most sought after dimensions of the business software applications that organizations depend on today. Despite this high demand for quality, very few studies have been done that evaluate the ongoing quality of software applications during the maintenance portion of the system life-cycle [42]. Quality is also measured objectively as number of failures and defects per month [42] and also quality can be supported by a standard implementation of code which, will result in quality software maintenance.

5. METHODOLOGY STEPS

The research methodology includes; establishment of some criteria related to standard code design, construction of a suitable model for measuring the values of the proposed criteria, maintain of construction groups (BSc students team), and results comparison. The following steps are representing the research methodology in details

1- Construction of documentation criteria and evaluation formula as shown in Table 1 and Formula 1.

2- Preparation of code segments (two sets, each one contains 18 programs) by two ways

a) Undocumented code, denoted by g1

b) Documented code, denoted by g2

3- Execute a short training course in the international documentation standards, to train two groups of code maintainers (four Bsc students)

4- Apply the criteria of Table 1 on g1 and g2 separately, the calculated results are shown in Table 2.

5- Calculate the total satisfaction for each set.

6- Maintain the software code (g1 & g2) depends on adaptive maintenance, then calculate the maintainability time for each program in g1 and g2.

7- Results comparison

5.1 Coding Factors

The proposed factors selected depend on three groups [43], these factors increase the code understandability; this will reduce the maintainability time of software. The proposed factors are thirteen factors, can be classified in three groups; first associated with general code, second associated with methods, third associated with classes. Each factor can be assigned to any of the following values {0,1,2,3,4}. Where, 0 indicates that the factor effect is absent, 1 means factor satisfaction is low, 2 means factor satisfaction is medium, then 3 is high and 4 means factor is completely satisfied (very high), kindly see Formula (1), that was created by Al-Hagery [43], the values of any factor FR in Table 1 can be estimated by Formula (1).

$$FR\_measure = \begin{cases} 0 : iff\ satisfaction \geq o\ \&\ < 10\% \\ 1 : iff\ satisfaction > 10\%\ \&\ \leq 25\% \\ 2 : iff\ satisfaction > 25\%\ \&\ \leq 50\% \\ 3 : iff\ satisfaction > 50\%\ \leq 75\% \\ 4 : iff\ satisfaction > 75\%\ \leq 100\% \end{cases} \quad (1)$$

The proposed factors were extracted from three groups of factors implemented in [43]. These factors produce a high quality code to reduce the maintainability cost. These factors are shown in Table 1.

Index	Factor name	Factor rank (FR)				
		0	1	2	3	4
1	Variables scope and role are defined clearly			○		
2	Code describes what is being done		○			
3	Understand the code by reading the comments					○
4	Preface comments defined clearly		○			
5	Use nouns or noun phrases for naming					○
6	Use alignment to enhances readability			○		
7	End of lines comments				○	
8	The meaning of return values		○			
9	Use verbs for Function names, Get, Find, …					○
10	The purpose of each method/function		○			
11	Variables declarations should be left aligned				○	
12	Use correct spelling in names		○			
13	Avoid using names that differ only by letter			○		
	Total Satisfaction = 29	**0**	**5**	**3**	**2**	**3**

TABLE 1: Maintenance Based Factors

Our model-based factor (MBF) is proposed to find the degree of documentation based on some standard criteria, as shown in formula (2).

$$MBF = \sum_{i=1}^{n} (\text{Factor } i \times \text{Factor\_Rank}), \quad n=13 \qquad (2)$$

For the example, the value of MBF obtained in Table 1 is 29, this value gives an indicator of the documentation level, the minimum value of MBF is 0 and the maximum value is 52, so the value of this example classified as medium.

6. WORKING GROUPS

Two teams are selected for maintenance purpose, each team consists of two students, the development strategy used is the "extreme programming". Team members are a final year students at the Computer Science department. On the other hand, the teams studied and practiced the concepts of writing standard code and they created some documented code as a result of their training, but this is not included within the research data, because they were maintain a code written by another people.

7. RESEARCH DATA SETS

The maintenance task performed by using eighteen software programs designed in C & C++ programming languages. This software constitutes the research data set that was used to prove the research validity. This data set was prepared as two groups, the first group prepared as a documented code, its documentation level graduated from 66% to 12% as partially documented code. Second group is prepared as undocumented code as shown in Table 2 column 3.

8. EXPERIMENTAL RESULTS

Table 2 displays summary results in this research. It includes some important attributes such as Complexity level, level of documentation (g1 and g2), total time1 for group 1 and total time2 for group2 and indicators. All these attributes are selected to be used for results evaluation and interpretation. The table contents are organized in ascending order depends on the value of indicator of the last column. The indicator value is assigned as follows:

Time1 > Time2 → - (the results are negative)
Time1< Time2 → + (the results are positive)
Time1 equal Time2 → ≡ (the results are equal)

program no	Complexity level	Level of Documentation		Time1	Time2	Indications *
		g1	g2			
9	15	15	1	6	5	-
16	65	18	1	10	7	-
1	60	40	1	5	4	-
14	80	36	1	22	20	-
11	20	42	1	5	3	-
7	20	21	2	7	7	≡
3	50	26	1	2	2	≡
6	20	12	1	3	5	+
5	40	21	1	8	16	+
10	10	25	1	17	24	+
4	35	34	1	3	10	+
8	35	35	1	3	4	+
18	70	35	2	4	6	+
2	45	36	1	1	2	+
17	70	40	1	14	23	+
13	75	45	1	3	4	+
12	60	46	1	2	5	+
15	50	44	2	8	14	+
Average				**6.83**	**8.94**	

TABLE 2: Summary of Experimental results

9. RESULTS DISCUSSION

Based on the results shown above in Table 2, these results show the rate of time that was measured during the maintenance of 18 programs applied in this research. The maintenance time was measured in two separate cases. First case, contains programs classified as partially documented. The second case contains undocumented programs, In the first case and second case, the average rate of time for maintenance was equal to (6.38.3) and (8.94) units of time, respectively.

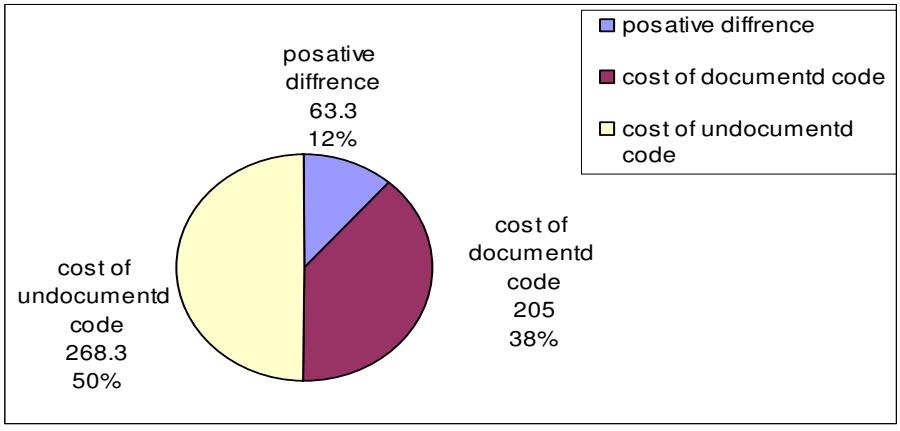


FIGURE 2: Maintenance cost results

Based on the previous values, found that the difference in time is equal to (2.11) unit of time, if we supposed that the cost of each maintenance hour is equal to 30 US$. So based on this value, the average cost of case 1 was 268US$ ≡ 50% of the total cost, and the average cost of case 2 was 205US$ ≡ 38% of the total cost, as well as the difference of the average in cost was 63.3 US $ ≡ 12% of the total cost, as presented in figure 2. Although there is a positive difference supports the principle of documented code which applied in this research. The total results showed three types of values, first gives a negative results, the second gives positive results, then the last gives a balanced results, as illustrated in figure 2.

Eleven programs of eighteen supports the principle of code documentation in a positive and shows important difference in the results, and based on this relative difference can present positive results for the proposed model, this obtained clearly how much cost can be reduced by building a complete documented code. Finally, it is important to mention that the average level of documentation of all applied programs was equal to 61% depends on both problem complexity and code size, and the average level of complexity of the used programs was equal to 46.

The more documentation process within large and complex programs, would contribute to the maintenance process required in the future, in addition to reducing the cost to do so. Also by comparing the results shown in Table 2, it is clear that small programs are not affected by documentation because its ideas is simple, easy, and the required time for maintenance is very short.

10. CONCLUSION

After discussing the results presented in this work, we found that applying the international quality standards on the code contents is very important to reduce its cost. In addition to that, it enables developers to reuse the source code. This code also will be more flexible, readable, easy to understand, and then S/W development organizations can do future development at a lower cost and better results depends on the results of this research. For programs that are small, simple, and well documented, they have negative results because the maintainers spend a lot of time and effort to understand the idea of the program by reading its documentation, although they can understand the idea directly without documentation of the Source code.

The presented results gave in general a positive effect of applying standard documentation process on software code, especially for long life software projects. The impact of this process is positive to support reducing the cost of software maintenance. By the proposed model we predicted that the medium level of software documentation reduces the cost of long-term maintenance by 12% and high level of software documentation (full documented code with complex programs) reduces the total maintenance cost by 24% at least, depending on the results comparisons presented above. This value is increasing with large, complex, and full documented projects/software. This also will encourage organizations to support the software quality by improving the developer's culture in this side, so any other S/W teams in future can enhance and improve documented legacy systems by adding new features or new functions.

11. FUTURE WORKS

There are some points can be taken into account to extend and modify this work from different points; firstly, increase the proposed factors to cover all quality factors. Secondly, improve the research results by increasing the number of maintenance teams. Thirdly, expanding the testing data to be more than 18 projects depends on big sizes, and complex projects that are completely documented.

12. REFERENCES

[1] M. Reformat, A. Kapoor, and N. J. Pizzi. “Software Maintenance: Similarity and Inclusion of Rules in Knowledge Extraction”, Proc of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), 2006, 723-731.

[2] W. Hordijk, and R. Wieringa. "Surveying the Factors that Influence Maintainability", In: Proc of the 10th European software engineering conference held jointly with 13th ACM

SIGSOFT international symposium on Foundations of software engineering, 5-9 Sep. 2005, pp. 385-388.

[3] N. E. Fenton, and S. L. Pfleeger. "Software Metrics: A Rigorous and Practical Approach", PWS Publishing Co, 1998.

[4] M. A. CÔTÉ, W. Suryn, C. Y. Laporte, and R. A. Martin. "The evolution path for industrial software quality evaluation methods applying ISO/IEC 9126: quality model: Example of MITRE's SQAE method. Software Quality Journal", Elements of Software Science, vol. 13, pp. 17-30, 2005.

[5] Y. Ahn, J. Suh, S. Kim, and H. Kim. "The software maintenance project effort estimation model based on function points", Journal of Software Maintenance, vol. 15, Issue 2, pp. 71-85, March/April 2003.

[6] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice", Addison-Wesley, 2nd edition, 2003.

[7] T. L. Graves, and A. Mockus. "Inferring change effort from configuration management databases", In METRICS '98: Proc of the 5th International Symposium on Software Metrics, IEEE Computer Society, 1998, pp 267-273.

[8] M. Lehman. "Laws of Software Evolution Revisited", Software Process Technology (EWSPT 96), 1996, vol. 1149, pp 108-124.

[9] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto. "Software quality analysis by code clones in industrial legacy software", In IEEE METRICS '02: Proceedings of the 8th International Symposium on Software Metrics, 2002, pp. 87-87.

[10] H. Abram, W. G. Michael, and C. H. Richard. "Reading beside the lines: Using indentation to rank revisions by complexity", journal of Science of Computer Programming,
Vol. 74, Issue 7, pp. 414-429, May 2009.

[11] R. K Bandi, V. K. Vaishnavi, and D. E. Turk. "Predicting maintenance performance using object-oriented design complexity metrics", IEEE Transactions on Software Engineering, vol. 29, no.1, pp.77-87, 2003.

[12] W. Li, and S. Henry. "Object-oriented metrics that predict maintainability", Journal of Systems and Software, vol. 23, no. 2, pp.111-122, 1993.

[13] S. C. Misra. "Modeling design/coding factors that drive maintainability of software systems", Software Quality Journal, vol. 13, no. 3, pp.297-320, 2005.

[14] M. T. Thwin, and T. S. Quah. "Application of neural networks for software quality prediction using object-oriented metrics", Journal of Systems and Software, vol. 76, no.2, pp.147-156, 2005.

[15] K. V. Coten, and A. Gray. "An application of Bayesian network for predicting object-oriented software maintainability", Information and Software Technology, vol. 48, no.1, pp. 59-67, 2005.

[16] Y. Zhou, and H. Leung. "Predicting object-oriented software maintainability using multivariate adaptive regression splines", The Journal of Systems and Software, vol. 80, pp. 1349-1361, 2007.

[17] M. Mari, and N. Eila. "The impact of maintainability on component-based software systems", In Proc of 29th Euromicro Conference, 2003, p. 25-32.

[18] P. Ardimento, A. Bianchi, and G. Visaggio. "Maintenance-oriented selection of software components", In Proc of Eighth European Conference on Software Maintenance and Reengineering, 2004, p. 115-124.

[19] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. "Software Engineering Metrics and Models", Benjamin-Cummings Publishing, Redwood City, CA, USA,1986.

[20] I. Heitlager, T. Kuipers, and J. Visser. "A Practical Model for Measuring Maintainability", Proc of the 6th International Conference on the Quality of Information and Communications Technology, IEEE, 2007, pp. 44-49.

[21] M. F. Bertoa, J. M. Troya, and A. Vallecillo. "Measuring the usability of software components", The Journal of Systems and Software, vol. 79, pp.427-439, 2006.

[22] Y. Wu, and J. Offutt. "Maintaining evolving component-based software with UML", In Proc of Seventh European Conference on Software Maintenance and Reengineering, 2003, p. 133-142.

[23] Y. Wu, D. Pan, and M.H. Chen, "Techniques of maintaining evolving component based software", In Proc of International Conference on Software Maintenance, 2000, p. 236-246.

[24] L. S. Rising. "Information hiding metrics for modular programming languages", PhD dissertation, Arizona State University, 1992.

[25] ISO/IEC 15939, "Software Engineering-Software Measurement Process", 2002.

[26] ISO VIM, second ed. "International Vocabulary of Basic and General Terms in Metrology", International Standards Organization, Geneva, Switzerland, 1993.

[27] J. Voas. "Maintaining component based systems", IEEE Software, vol. 15, no. 4, pp. 22-27, 1998.

[28] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Std 610-1990, The Institute of Electrical and Electronics Engineers, New York, NY, 1990.

[29] G. T. Heineman, and W. T. Councill. "Component-Based Software Engineering: Putting the Pieces Together", Addison-Wesley, 2001, pp. 741-753.

[30]S. S. Yau, R. A. Nicholl, J. J. Tsai, and S.S. Liu. "An integrated life-cycle model for software maintenance", IEEE Transactions on Software Engineering, 1988, vol.14, no .8, pp.1128-1144.

[31] J. Park, W. Jung, and J. Ha. "Development of the step complexity measure for emergency operating procedures using entropy concepts", Journal of Reliability Engineering & System Safety, vol. 71, pp. 115-130, 2001.

[32] ISO/IEC 9126. "Software Engineering-Product Quality-Part 1: Quality Model", International Standards Organization, Geneva, Switzerland, 2001.

[33] C. Chen, C. Lin, C. Wang, and C. Chang. "Model for measuring quality of software in DVRS using the gap concept and fuzzy schemes with GA", Journal of Information and Software Technology vol. 48, pp.187-203, 2006.

[34] S. M. Dekleva. "The influence of the information systems development approach on maintenance", the journal of MIS Quarterly. Vol.16.issue.3, pp.353-372. 1992.

[35] W. E. Perry. "Quality concerns in software development", the challenge is consistency, Journal of Information Systems Management, vol. 9, Issue 3, pp. 48-50, 1992.

[36] M. A. Cusamano, and C. F. Kemerer. "A quantitative analysis of U.S. and Japanese practice and performance in software development", Journal of Management Science, vol. 36, issue 11, pp. 1384-1406, 1990.

[37] D. L. Nazareth, and M. A. Rothenberger. "Assessing the cost-effectiveness of software reuse: A model for planned reuse", The Journal of Systems and Software, vol. 73, pp. 245-255, 2004.

[38] P. Narayanan, S. P. Raja, X. Birla, K. Navaz, and S. A. Abdul Rahuman. "Improving Software Maintainability through Risk Analysis", International Journal of Recent Trends in Engineering, vol. 2, issue. 4, pp. 198-200,November 2009.

[39] J. A. Hoffer, J. F. George, and J. S. Valacich. "Modern Systems Analysis and Design", Third Edition, 2005.

[40] R. A. DeMillo, R. J. Liption, and A. J. Perlis. "Software Project Forecasting", Software Metrics, MIT Press, Cambridge, MA, p. 77, 1981.

[41] C. Lung, X. Xu, M. Zaman, and A. Srinivasan. "Program restructuring using clustering techniques", The Journal of Systems and Software, vol. 79, pp.1261-1279, 2006.

[42] M. Ghods, and K. M. Nelson. "Contributors to quality during software maintenance", Journal of Decision Support Systems, vol. 23, issues 4. pp. 361-369, 1998.

[43] M. A. Al-Hagery. "Model-based factors to extract quality Indications in software lines of code", International Journal of Computer Science & Information Technology (IJCSIT), vol. 3, issue 2, pp. 112-121, April 2011.

Distributed Agile Development: Practices for Building Trust in Team Through Effective Communication

Dr. Satya Prasad Ravi *profrsp@gmail.com*
Associate Professor, Dept. of Computer Science & Engg.,
Acharya Nagarjuna University , Guntur, Andhra Pradesh, India

Lakshmi Sridhar Movva *ml.sridhar@gmail.com*
Certified Scrum Master and Industry Consultant in Information Technology.

B.Reddaiah *b.reddaiah@in.com*
Assistant Professor, Department of computer Applications,
Yogi Vemana University, Kadapa, Andhra Pradesh, India

Abstract

Agile methods have been now widely popular and have been proved to be delivering high-quality software to the global users in shorter time frames and are effectively handling the continuous change on the requirements from the users. However, due to various reasons such as technical expertise scarcity, functional expertise scarcity, cost effectiveness, resource availability, globalization, necessity to work to the fullest taking the advantage of time zone variations and other factors; the teams can be geographically dispersed. We can call them Distributed Agile teams[1]. Given this globally distributed nature of the Agile team, the major challenge lies with the team communication and building trust across the team;

It is difficult to foster team bonding and collaboration with the distributed teams with few or no face-to-face interaction. The difficulties in communication and lack of trust in Distributed agile teams would have an impact on the productivity .Our ob-ejective is to suggest usage of some of the existing practices and propose a new practice KYTE to overcome the barriers of communication and building trust in Distributed Agile Teams ,which would contribute to the increase in productivity.

Keywords: Agile Methodologies, Distributed Agile, Team Communication, Building Trust, KYTE

1. INTRODUCTION

[6]Software organizations constantly need to react to market dynamics, new customer requirements and technological innovations (Beck 2000; Lycett et al. 2003). The degree of market dynamics and needs has increased over the past decades creating a number of fast moving software organizations (Börjesson and Mathiassen 2004). The experiments and surveys on agile methods promise faster development thus improving the communication and collaboration inside agile teams and within the teams, customers and business units (Anderson 2003). Many organizations regard agile methods as a way of addressing key problems in software development; namely, the software takes too long to develop, costs too much and has quality issues upon delivery (Holström et al. 2006). Thus, agile methods (e.g. Extreme Programming [XP]; Beck and Andres 2004) and SCRUM (Schwaber and Beedle 2002) have been suggested as a way of responding to the changes, shortening the development time and improving communication and collaboration, especially in situations in which timing is a critical competitive advantage for an organization (Anderson 2003; Karlstrom and Runeson 2006). Communication and trust between team members is an important factor in software development and, thus, a relatively common success factor. These factors are even more important if the team is distributed.

In a Distributed Agile team environment the team members do not work in close proximity. They might belong to different Countries, Regions, Cit-is, organizations, Cultures, Race, Origin and can be at different levels of expertise. For example the Product Owner sits in London, Business might be at China and the Scrum Master can be located in Bangalore, India along with few team members and some team members might be working from Hyderabad, India. Given this diversity; when working as a team; it would be difficult to effectively communicate and build trust among the team members without a face-to-face interaction. [1]The more Distributed the team is ,the more challenging it becomes in terms of communication.Working with a distributed team means actively working on communication,making sure teams have right tools, addressing the issues head on and always seeking ways to improve[1]Communication can impact team members understanding of what they should be doing.

The types of Distributed teams in the order of increased distribution of team members is

- Collocated part time,
- Distributed with overlapping hours and
- Distributed with no overlapping hours.

The challenge of communication increases in the same order.
However; the existing communication methods given below can break these barriers to an extent. In this paper we propose a few more methods that would enrich the communication enabling to build the trust between the team members and lead to a productive Distributed Agile team. The various methods that are existent to overcome these barriers are as below.

2. EXISTING COMMUNICATION METHODS

The Existing Communication Methods; few of them:

Share your screen Using Remote Desktop, VNC, Net Meeting, Team Viewer etc for Screen sharing .This is very useful for demonstrating new features, reproducing bugs, working with customers, sharing Ideas / understanding, assisting in installations and much more.

Screen casts are video/audio recordings of the computer screen and a person talking; they are very useful for explaining a feature or module to another developer. Recording then viewing a screen cast is not as effective as sitting alongside someone explaining in real-time, but it has the tremendous advantage of re-playability. A series of screen casts explaining important parts of a system will get new team members up to speed quickly. The Knowledge transfer activity for the new joiners can be given with these screen casts.

Screen Shots. Don't just tell; show. Show another developer on your team what you mean; by taking a screenshot and showing them up.

Mockups. A distributed team leaves more room for misunderstanding of desired results. Counteract this by building a mockup (text files, paper, Excel, etc.) of what you want. These mockups can be very well shared across the distributed team by taking a photograph and sending them across or they can be shared via screen sharing too.

Issue Tracking. Mantis, Trac, Jira,MKS is a few of them. These tools manages and maintains lists of issues, their priorities, severities and status as on date as needed by an organization.

Source Control Systems. CVS, SVN, Clear Case, proforce,vss are a few of them and are helpful to have the project version history available locally to all the team member though they are dispersed geographically.

. Phone calls Phone calls are cheap now a days. VOIP is the best for long distance calls. However; usage of Skype, IM, Office communicator which are available for online voice conversations are at their best.

- Instant Messaging. Do not type longer conversation until and unless you require the chat transcripts to be recorded. The key value in IM is as a substitute for the awareness of who is available and working and for shorter and quick conversations/clarifications.

- Email is probably the most important tool. Used to keep a record of the communication. Use it to summarize discussions, MOM, conclusions arrived at, communication to group. Each of these methods/tools help in communicating the information across the team.However;Usage of these tools/methods without actually knowing the other person might not be that effective when compared to the usage of the same with a known team member personally. This gap comes with many apprehensions, assumptions we have about the other team member sitting on the other side of the globe or in the other city elsewhere whom you have never met. As we are dealing with the Distributed Agile team; we propose a few new methods to minimize the ineffective communication between the team members and hence build the trust between them to an extent.

3. ADDITIONAL PRACTICES

3.1 KYTE: Know Your TEam Better

FIGURE:1

KYTE (Know Your TEam better) is a new process created keeping in view the challenges in distributed environment particularly lessened trust among the team members because of poor communication.

Teams tend to work for years together without knowing much about the other team members.There were instances the team members feel that they should have know the other team members better to communicate with them better. Effective communication would be possible with lessened apprehensions about the other members and having a oneness amongst the team.

KYTE is one step towards overcoming these problems to an extent. As the name says it is all about knowing the team better; better in terms of knowing the team members; their thought process; their culture; their expertise; their thoughts about us; their interest and to be crisp knowing anything about them which makes us communicate better with them.

The KYTE template provides the details of all the team members viz Team members name; their birthday; about them; where they are from (add a hyperlink to their native pointing to any of the website that provides the details); area of interests; favorites etc which helps the other team members to know a bit of what he/she is.

The details of the new team members should be added on need basis.The Template should be stored in a common folder enabling ease of access to do any updates on it .This should be shared across every quarter.Any recent photographs, photographs taken during festivals in last

quarter and the festival details (provide a hyperlink for the festival, any of your photos) in their country would be good things to add any point in time.Below is the sample KYTE format.

Any New team member who is on-boarded to project will be asked to fill in those details and update KYTE template should be shared across team.

Details of the following people can be added to KYTE.

- Product owner
- Scrum Master
- Team Members
- Stake holders

Sample KYTE template for one team member

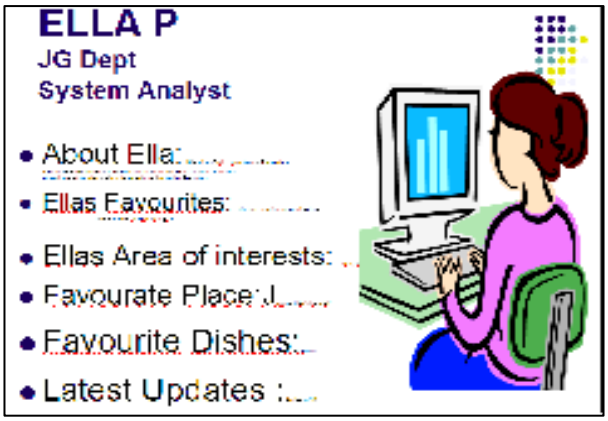


FIGURE 2: KYTE Template with filled in Details of all team members

FIGURE 3

3.2 Use Pinup Boards

Also as a part of the KYTE; Prepare a poster with simplest details of the team members photos, their name, location, designation, Company and pin it up to the pin boards at the desks. Below is a sample. You will eventually end up having a look at the team members once in a while and this would definitely help in giving you a feel that you know your team better.

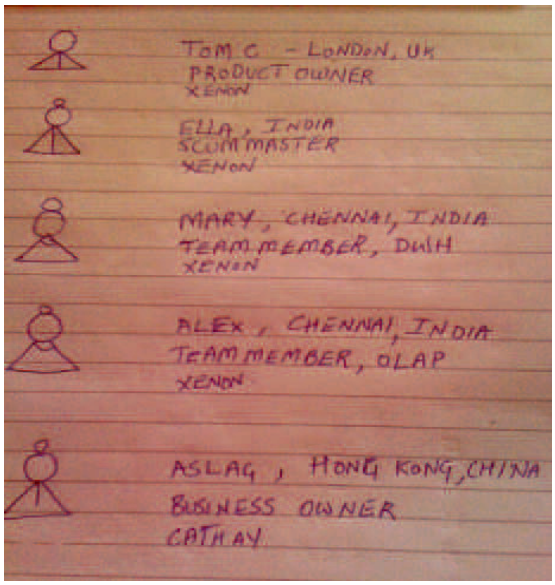


FIGURE 4: A sample of team member details for putting on Pin up boards.

3.3 Usage of Photograph in IM:

Practice to use the photograph of self in IM. This would definitely prove better when you are chatting with the team member. Usage of a webcam along with teleconference via phone or IM,skype etc which would be more beneficial.

3.4 WWU

WWU is we wish you.WWU is about wishing the other team members with a card at least once or twice a year. Pick up a nice card with a team motivating quotation on any occasion and send it across signed by all the team members. Though it looks simple; but it would definitely work wonders in conveying team members that we are with you. It would boost up the team morale. Pin the cards received on to the pin boards.

4. ADOPTION AND CONCLUSION

The practices described in this paper are proposed to a Distributed agile team.The team has not been using these practices before and there is scope for some improvement in the team as it is totally distributed.Further;the plan for analyzing the impact of the new practices on comfort in communication and trust in team is measured based on a feedback from the team. The team is given a questionnaire and asked to answer them before and after usage of the KYTE and suggested practices in this paper.

The set of questions shared with the team is as follows.

1. Do all team Members know the other team members and their interests?
2. How well do members of your team share responsibility for tasks?
3. Do team members interact not only in meetings but also through other means like IM,Phone more effectively?
4. Do team members argue even it is not productive to do so?
5. Does the team engage in complex analysis, including listening and asking questions?
6. Does the team collectively own the situation in difficulties.
7. What do you feel is the overall comfort in communication across team?
8. Do you trust your team members?

The team were asked to select one of the five options as answers as below

- Extremely Well

- Very Well
- Moderately Well
- Slightly Well
- Not at all Well

The Answers were given rating as below

- Extremely Well -5
- Very Well -4
- Moderately Wel-3
- Slightly Well-2
- Not at all Well-1

The feedback received from the team is rated and summed up for each question and is as below Here Before (B) is before implementing this practices and After (A) is after implementing them Below is the data calculated from the feedback ratings.

TABLE:1

Sno	Feedback	Before (B)	After (A)
1	FB1	13	15
2	FB2	11	16
3	FB3	15	17
4	FB4	14	16
5	FB5	12	15
6	FB6	12	16
7	FB7	12	16
8	FB8	14	15

Here FB1 is Feedback for Question 1
Each value in the cell in Before (B) column is the sum of ratings given by the team members before implementing KYTE and suggested features

For example; in the above table Before(B) data related to FB1 is 13 which is sum of the ratings given by five team members 2+3+3+2+3= 13

Each Cell in After(A) is the sum of ratings after implementation

C_b: Comfort in communication and trust within team before the adopting the new features are

$$C_b = \frac{\text{Sum(B)}}{\text{FB cnt*highest score for FB*team member cnt}}$$

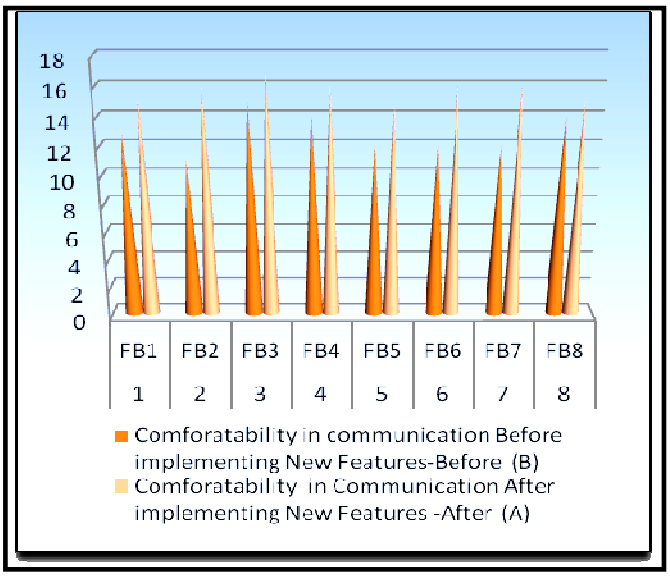
$C_b = 103/8*5*5 = 103/200 = 0.515$

C_a : Comfort in communication and trust within team after adopting the new features is

$$C_a = \frac{\text{Sum(B)}}{\text{FB cnt*highest score for FB*team member cnt}}$$

$C_a = 126/8*5*5 = 126/200 = 0.63$

FIGURE 5 : Below is the comfort chart representing the data in Table:1



After implementing the suggested practices and KYTE, it is evident that there is some information exchange happening within the team on the details of the other members. Though the information shared about each team member might look simple; after the KYTE and other practices are implemented, the effectiveness of having a feel that the team knows each other well has increased considerably and the feel of oneness among the team has been expressed by the team members.Also, a few team members who never had an opportunity or situation to speak with each other members have opened up a communication channel discussing about their interests. Also it is observed that some of the less interactive team members sending wishes to others on some occasion or the other. Now the team members are more open for discussion and have less/fewer apprehensions about the others in the team when compared with the earlier situation in the team.

To asses the comfort on communication and increase in trust front, a questioner was framed. We have gauged the rating of comfort in communication from 0 to 1; which is 'Not Comfortable at all' to 'Highly comfortable'. Before implementing the suggested practices, based on the feedback on the questioner from the the team members, and the rating given to them, we have calculated the comfort in communication and increase in trust and is at 0.515. The same has elevated to 0.63 after implementing the KYTE and practices suggested,which means we have the members in team knowing each other much better ,communicating much better and trusting each other.

Based on the data gathered; the total percentage of increase in comfort of communication and increase in trust by implementing these suggested and new features is as below
Increase in comfort $C_\theta = C_a - C_b = 0.63-0.515 = 0.115$
Percentage of Comfort Increase $C_i = C_\theta * 100 = 0.115 * 100 = 11.5\ \%$

Further; Productivity of a team is based on various factors and one factor being effective communication between the team and trust between team members. The productivity of team would be high if there is an open communication between team members and they trust each other.If there is a C_i increase in the communication comfort and trust between team members; then it would definitely cater to the overall increase in productivity of team. Thus implementing the new KYTE practice and following the suggested practices proved to have a positive impact and boost up the comfort in communication and trust, which directly caters to the overall increase in productivity of the distributed teams.

5. REFERENCES

[1] A practical Guide to Distributed Scrum: Elizabeth Woodward,Steffan Surdek and Matthew Ganis.

[2] Coaching Agile Teams-Lyssa Adkins

[3] Agile Product Management with Scrum –Roman Pichler.

[4] Experience Report: Distributed agile: project management in a global environment Seiyoung Lee & Hwan-Seung Yong

[5] Agile Alliance, Principles behind the Agile Manifesto, http://agilemanifesto.org/principles.html.

[6] Highsmith J, Cockburn A. Agile software development: The business of innovation

[7] Lycett M, Macredie RD, Patel C, Paul RJ. Migrating agile methods to standardized development practice

[8] A Paper named Mira Kajko-Mattsson 'A Problems in Agile Trenches-08'

[9] A paper named 'The impact of agile practices on communication in software development' by M. Pikkarainen & J. Haikara & O. Salo & P. Abrahamsson & J. Still

[10] http://www.agilealliance.com

[11] http://www.controlchaos.com

[12] http://www.scrumalliance.org

Java-centered Translator-based Multi-paradigm Software Development Environment

Xiaohong (Sophie) Wang *xswang@salisbury.edu*
Department of Mathematics and Computer Science
Salisbury University
Salisbury, MD 21801, USA

Abstract

This research explores the use of a translator-based multi-paradigm programming method to develop high quality software. With Java as the target language, an integrated software development environment is built to allow different parts of software implemented in Lisp, Prolog, and Java respectively. Two open source translators named *PrologCafe* and *Linj* are used to translate Prolog and Lisp program into Java classes. In the end, the generated Java classes are compiled and linked into one executable program. To demonstrate the functionalities of this integrated multi-paradigm environment, a *calculator* application is developed. Our study has demonstrated that a centralized translator-based multi-paradigm software development environment has great potential for improving software quality and the productivity of software developers. The key to the successful adoption of this approach in large software development depends on the compatibility among the translators and seamless integration of generated codes.

Keywords: Software Development Environment, Translator, Multi-paradigm.

1. INTRODUCTION

Improving the quality of software products and the productivity of software developers has been an enormous challenge for the software industry. To respond to the challenge, many new design and development methodologies and programming paradigms have been introduced. The availability of modeling tools and rich sets of libraries and the adoption of design patterns and application frameworks all contribute to produce better software systems today. Another rapid evolving frontier in this campaign is the development of programming languages based on different paradigms. In the context of computer science, a programming paradigm is defined as a computational model [1] that a programming language is based on, i.e., the style or approach a programming language uses to express problem solving plans. In the past forty years, several generations of programming languages have been introduced based on the following four dominant programming paradigms: imperative, functional, logic and object-oriented. Since real world problems are much diversified, it is not surprising that some styles are better suitable to solve some problems than others. Another observation is that for large sophisticated software, it is likely that a single paradigm may not be enough to develop all parts of the system. This naturally led to the pursuit of software development using programming languages with different paradigms, i.e., multi-paradigm programming. The overall objective of multi-paradigm programming is to allow developers to choose a paradigm best suited for the part of the problem to be solved. As for how multiple paradigms can be deployed to build a single application, many different routes have been taken to try to answer this question.

The translator-based multi-paradigm programming was first proposed in [6]. This approach allows multi-paradigm programming by translating the source code written in different paradigms into a target language code before they are integrated. [7] has demonstrated the feasibility of this approach by developing a compiler for a functional programming language. However, there are still some questions remain to be answered. How feasible and realistic is it to use this approach in

large scale real world application development? What are the main obstacles of deploying this approach in real world?

The rest of this paper is organized as following. In the second section, the background of the translator-based multi-paradigm is discussed. In the third section, we describe our experiment with the translator-based multi-paradigm programming by implementing a centralized software development platform *SourceMerge*, which allows for program development with logic, functional and object-oriented programming languages. Using this platform, a *calulator* application is developed with expression validation, evaluation and GUI components written in Prolog, Lisp and Java respectively. Issues encountered during this experiment are also discussed in this section. The final section summarizes our current work.

2. TRANSLATOR-BASED MULTI-PARADIGM PROGRAMMING

A programming paradigm is often defined as a computational model ([1]) that a programming language is based on. In general, a programming language implements only a single paradigm. For example, the imperative paradigm, with C as an example language, is identified by the use of variables, assignment statements and explicit flow of control. The functional paradigm, with Lisp as a representative, distinguishes itself in function definitions, recursion and the ability to create high-order functions. Logic paradigm depends on rules and logic, and a main language supporting this paradigm is Prolog. The object-oriented paradigm, featured by Java, uses class inheritance hierarchy and polymorphism to create applications with dynamically reusable code. Some modern programming languages can support more than one paradigm. For example, C++ supports both imperative and object-oriented paradigm and SICStus Prolog supports both logic and object-oriented paradigm. Each paradigm has its strength and weakness in representing the concepts and carrying out the actions of a specific application. Multi-paradigm programming is to explore different ways to integrate the best features of each paradigm in software development.

Generally speaking, multi-paradigm programming can be accomplished either inside the same programming language (i.e., language extension using multi-paradigm languages) or in a system that assures a certain way of integration and interaction among separate processes or modules. [2] and [3] proposes to use a multi-paradigm language. Rather than deciding what the correct paradigm is to use, using a language that implements every paradigm can easily solve the issue theoretically ([2]). The problem with this approach is that a language with many paradigms intertwined would be too difficult for most programmers to learn and would be rather hard to understand and debug applications written in such language. An emerging theme is the ability to access one programming language from another ([4]). A good practice would suggest keeping paradigms separated to allow for understandable code. [6] and [7] approach the problem by translating a single language program into a target language such as C. While the approach is credible, it restricts the abilities of multi-paradigm programming to only allowing source and target paradigms to be used together. As one can see, all those approaches are limited in either the number of paradigms can be combined and the degree of integration can be achieved.

Translator-based programming has been discussed in [1], [6], and [9]. The main idea is that different parts of an application can be written in different programming languages; later those different parts are translated into one target language; finally the translated source code in the target language are compiled into a final executable by the target language compiler. A similar approach surfaced after [6] allows for two languages to be written together in the same source file(s) and to be translated/interpreted during compile time ([4]). This approach is much like translator-based multi-paradigm programming except the code are written as if part of the same language instead of being written in separate files as separate programs. [1] argues that after comparing with all others, the translator-based multi-paradigm approach appears to be the most viable and expandable solution since theoretically it allow any number of paradigms to be integrated in a natural way and it is a much better compromise between ease of use and degree of integration.

3. JAVA-CENTERED TRANSLATOR-BASED MULTI-PARADIGM PLATFORM

To answer the questions such as how feasible and realistic is it to use translator-based multi-paradigm approach in large scale real world application development and discover the main obstacles of deploying this approach, we developed a Java-centered translator-based multi-paradigm platform *SourceMerge*. *SourceMerge* provides a simple interface for selecting source files written in different paradigms and translated them into the target paradigm. In this case, Java is selected as the target language and programs written in Lisp and Prolog representing logic and functional paradigm are prime candidates to be translated. Once the selected source files written in Prolog and/or Lisp are translated by their corresponding compilers respectively, *SourceMerge* can collect them into a single location and generate all required libraries and packages and use Java compiler to generate a single, standalone Java application.

3.1 Design Considerations

Java is selected as the target language in *SourceMerge*. Java's object-oriented paradigm is much more suitable for large-scale applications due to its capability for abstraction, inheritance and polymorphism, easy-to-use language interface and its portability to any system that runs a Java Virtual Machine. Java's strength in describing real world objects and their behaviors with class structure makes it natural to represent the concepts and actions to be carried in other paradigms (such as functions in functional paradigm and predicate logic in logic paradigm). Java's sandboxing provides a security blanket that can protect a user from system crashes caused by errors in translated code.

We choose logic and functional paradigms as the two candidate paradigms in *SourceMerge*. However, as can be seen from Figure 1, the design of *SourceMerge* allows the inclusion of a new paradigm to the system can be done easily (as demonstrated by the dashed line portion in Figure 1) since the translation process for each paradigm is independent from the rest of the system. *SourceMerge* acts as an adapter to translate multiple paradigms into a single target paradigm. As long as a language translator follows the rules for generating output defined by *SourceMerge*, it can be easily integrated into the system.

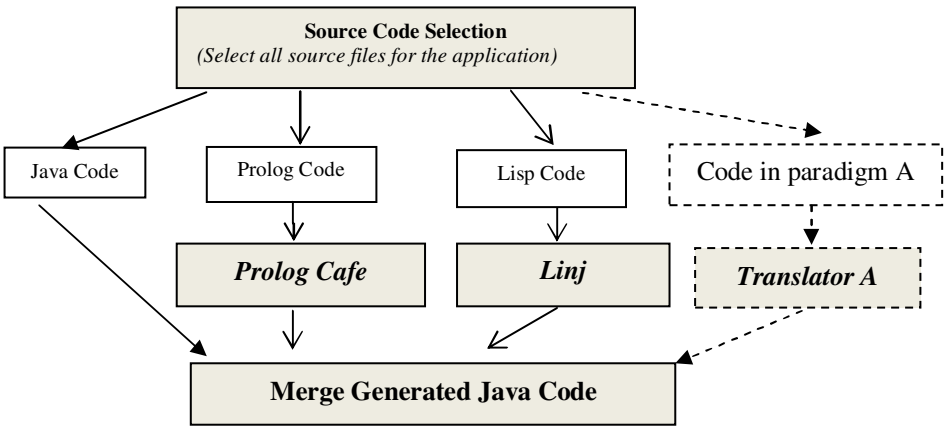


FIGURE 1: *SourceMerge*'s Select, Translate, Merge and Compile Process.

Figure 2 show the GUI of *SourceMerge* application. Three major tasks can be completed on this interface: selection of source files, translation of the source code and merging of the final executable. The execution status of translation and compilation is also displayed on the interface.

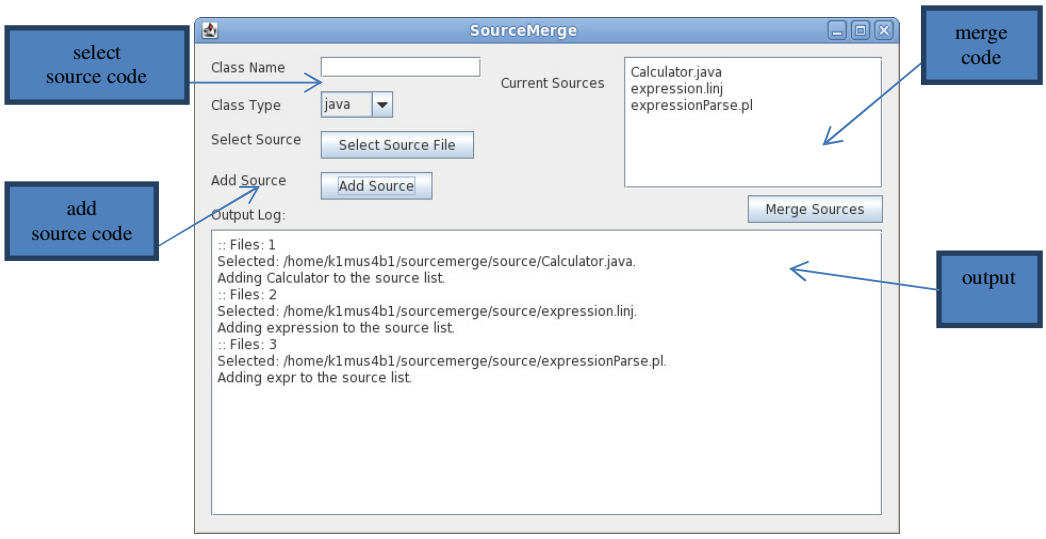


FIGURE 2: *SourceMerge* Main Interface.

3.2 Translation Tools

Due to the time constraint for this study, we decided to use existing open source Lisp and Prolog translators. After a thorough research over the Internet, we find that *Prolog Cafe* [9] and *Linj* [10] are the only two freely available tools exist today that specifically meet the needs of translating the Prolog and Lisp languages to Java.

The Prolog to Java translator *Prolog Cafe* is built on the *de facto* standard for Prolog compilers, the Warren Abstract Machine (WAM) ([5], [11]). After translating Prolog sources to Java, the execution model of the generated Java classes are also based on the WAM. Though this produces difficult-to-read code and layers of terms to sort through, the translated output executes cleanly and, in the Prolog aspect, quickly. *Prolog Cafe* is written in Java and therefore portable to any platform with Java compiler and runtime environment. The generated source code requires the inclusion of the *Prolog Cafe* Java libraries that implement the WAM algorithms. Additionally, the compiled program still depends on a standalone interpreter within *Prolog Cafe*. This makes it unsuitable to be embedded into other Java programs and we will address this development issue further later.

Linj, the Lisp to Java translator, is open source and it translates from one source paradigm to Java. To make the translation algorithm efficient and allow the programmer to follow the Lisp programming conventions and still have the translated source follow all of Java's rules, *Linj* comes with its own language, respectively named *Linj* ([10]). The *Linj* language is syntactically the same as Common Lisp except for packages, something that is ignored for this study, and the existence of a null-term as opposed to an empty list. The *Linj* translator is written in Common Lisp and the translated source code is purposed to be human-readable and efficient. There are no specially required Java libraries to include so the individual classes generated can be compiled as standalone programs, or embedded into other Java applications.

3.3 Encountered Issues

A major hindrance towards the research for this project is that: both of the translators, *Prolog Cafe* and *Linj*, are no longer supported by their creator. Also, due to the limited user base of the tools, community support and resources are scarce, if not nonexistent.

The first obstacle is that *Prolog Cafe* generates Java source code as a standalone application running through a command-line interpreter that comes with *Prolog Cafe*. This execution model does not fit into our design of the centralized environment that merges Java classes generated

from different paradigms and compiles them into a single Java application. Since the command-line interpreter for *Prolog Cafe* is also open source, and it also includes the same Java libraries that are required to be included in the generated source code, this became the starting point of tackling this issue. After stripping away the bells and whistles of the command-line interface of the interpreter, it became visible that the same procedures are called to execute any translated code each time. Therefore, for each Prolog source code, a specially defined template class, which gets dynamically modified by *SourceMerge*, is used to produce the source suitable for integration with other Java classes.

Along with no longer being supported, the documentation for *Linj* was never completed. There are also no instructions as to how to install *Linj* nor the system requirements for installation. The *Linj* translator uses direct Linux commands and the translation process must be performed under a Linux system. This information is missing from the unfinished documentation. Another challenge is that all required Lisp packages are not specified in the document either. So significant amount of effort were made to determine the system requirements, write a parser to search through the *Linj* translator to identify all packages used before the final *Linj* to Java translator running Linux with the Steel Bank Common Lisp compiler was installed successfully.

Two similar issues affected both the *Prolog Cafe* and *Linj* translators from producing usable code and neither were ever hinted at in documentation or other sources. The required Java libraries by the source generated from *Prolog Cafe* don't exist, until one translates them from Prolog to Java. Since the interpreter provided with *Prolog Cafe* has a set of pre-compiled libraries required, initially no errors were encountered when it was used alone. However, to include them into other sources, they needed to be in their raw source and the errors started to show up. With *Linj*, the translator successfully translates all basic Lisp programs to Java without an issue. However, whenever non-basic expressions were used, such as a built-in function, compilation would silently fail without any error messages. After significant time and effort spent on debugging, we found that *Linj* requires a Lisp-package for each of the types being translated (such as mathematical expressions, or vectors). Both of the issues were resolved eventually by letting *SourceMerge* automatically supply the required Java libraries during translation.

3.4 Application Development Demonstration

To demonstrate the functionalities of *SourceMerge*, an arithmetic calculator program was created under the *SourceMerge* environment. The *Calculator* application was written using all three distinct paradigms allowed by *SourceMerge*: functional, logic, and object-oriented. Each paradigm was used to write the part of the application that highlights the best features of this paradigm.

The *Calculator* can perform input validation and evaluate arithmetic expression. Java, the object-oriented paradigm with rich GUI libraries, was used to create the *Calculator*'s GUI (see Figure 3-4).

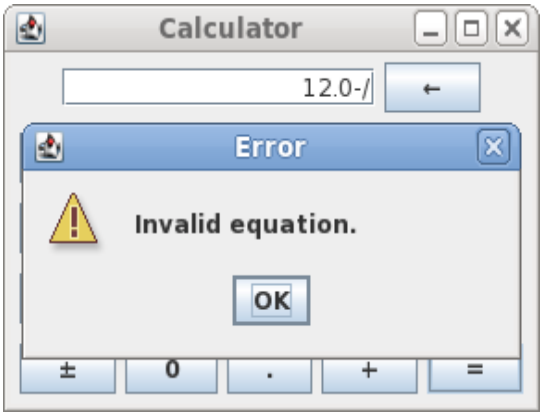


FIGURE 3: Expression Validation.

During the development of the *Calculator*, Prolog, the logic paradigm language, was used to implement the validation function for the *Calculator* (Figure 3). Using predicate logic, the validation of a proper arithmetic expression can be achieved by the following block of Prolog code:

```
expr(L) :- num(L).
expr(L) :- append(L1, [+|L2], L), num(L1), expr(L2).
expr(L) :- append(L1, [-|L2], L), num(L1), expr(L2).
expr(L) :- append(L1, [*|L2], L), num(L1), expr(L2).
expr(L) :- append(L1, [/|L2], L), num(L1), expr(L2).
num([D]) :- number(D).
```

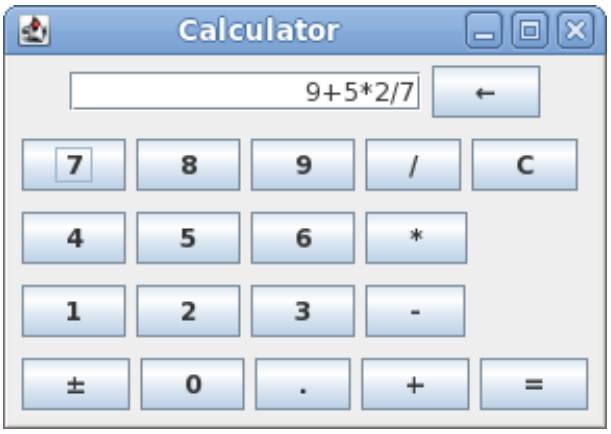


FIGURE 4: Expression Evaluation.

The expression evaluation of the *Calculator* was implemented using Lisp, the functional paradigm language (Figure 4). Avoiding the easy-way of using Lisp's eval, the source was designed to use the prefix notation to operate on two numbers. The used method is as follows:

```
(defun expression(x/float operator/string y/float)
        (cond ((string-equal operator "+") (+ x y))
                ((string-equal operator "-") (- x y))
                ((string-equal operator "*") (* x y))
                ((string-equal operator "/") (/ x y))
                (t 0)))
```

As discussed earlier, during the implementation of the *Calculator*, each of the three involved paradigms was used to implement a component that showcases the paradigm's features most suitable for the functionality of the component. The three kinds of source files were sent into *SourceMerge* (see Figure 2) and the Prolog and Lisp files were successfully translated into Java classes; these Java class files were all merged together; and a compiled *Calculator* application was presented.

4. SUMMARY AND DISCUSSIONS

The Java-centered multi-paradigm software development environment *SourceMerge* built in this study confirms that translator-based multi-paradigm software development approach is, theoretically feasible for producing good quality software efficiently. *SourceMerge* demonstrates a way to take source code from Prolog and/or Lisp, translates them into Java classes and has them merge together to form a single application. This is an important and exciting step. What is even important is to know what factors make this approach deployable in real world software development. The experience in this study has shed some lights in answering these questions.

First of all, our experience shows that a centralized development similar to *SourceMerge* is crucial to the success of translator-based multi-paradigm programming. It would be very difficult and frustrating if a user has to go through many complex processes to accomplish the translation and deal with the inconsistent behaviors and interfaces of the generated code. All the productivity increase and quality improvement promised by the translator-based multi-paradigm programming will be diminished by this difficulty.

Secondly, when building a centralized environment for multi-paradigm software development, many important factors should be taken into considered. Multi-paradigm means not just one, two or three paradigms to be used. It means that a centralized environment should be scalable enough to accommodate as many paradigms as possible. To achieve this goal, the design of the system should follow the Open-Closed design pattern, i.e., a new paradigm should be added to the system easily and the update on the current paradigm translation process should not affect other translation processes at all.

Thirdly, the selection of translators is the key to building a centralized multi-paradigm system. In this study, two open source translators were selected. This decision was made due to our time constraint. During our development process, we have encountered many unforeseen obstacles, such as limited documentation, unknown system requirements and missing features. Another obvious drawback with using existing translators is that the output code generated by the translators of different paradigm may not be consistent. This will definitely make the integration of the final executable very different if not impossible. We recommend that standards for the translated code should be established and the design for each translator should base on the pre-defined standards so that they can have consistent behaviors and interfaces. Although this approach requires an upfront investment to build the translators, this will make the integration of translated sources and the future extension of the system easier.

5. REFERENCES

1. R. Horspool and M. Levy. “Translator-Based Multiparadigm Programming”. Journal of Systems and Software, 25, 39-49, 1993.

2. T. Budd and R. Pandey. “Never Mind the Paradigm, What About Multiparadigm Languages?” SIGCSE Bulletin, 27, (2), 25-30, 1995.

3. D. Spinellis. “Programming Paradigms as Objective Classes: A Structuring Mechanism for Multiparadigm Programming,” Ph.D. Thesis, University of London, 1994.

4. M. Carlsson et al. “SICStus Prolog User’s Manual”. Swedish Institute of Computer Science, 2011.

5. H. Ait-Kaci. “Warren’s Abstract Machine: A Tutorial Reconstruction”. MIT Press Cambridge, 1991.

6. P. Codognet and D. Diaz. “wamcc: Compiling Prolog to C”. In 12th International Conference on Logic Programming, MIT Press, 317 – 331, 1995.

7. M. Levy and R. Horspool. “Translating Prolog to C: a WAM-based approach”. In Proceedings of the Second Computing Network Area Meeting on Programming Languages, and the Workshop on Logic Languages, 1993.

8. X. Wang, “Compiling Functional Programming Languages Using Class Hierarchies”. M.Sc. Thesis, Department. of Computer Science, University of Victoria, 1992.

9. M Banbara, N. Tamura, and K. Inoue. “Prolog Cafe: A Prolog to Java Translator System”. INAP, 45-54, 2005.

10. A. Leitão. “Migration of Common Lisp Programs to the Java Platform -The Linj Approach Linj”. 11th European Conference on Software Maintenance and Reengineering, 243 – 251, 2007.

11. D. Warren. “An abstract Prolog Instruction Set”. Technical Note 309, SRI International, Menlo Park, CA, 1983.

Using Meta-modeling Graph Grammars and R-Maude to Process and Simulate LRN Models

Nardjess Dehimi *ndehimi@yahoo.fr*
Computer Science Department/MISC Laboratory
University Mentouri Constantine
Constantine, 25000, Algeria

Allaoua Chaoui *a_chaoui2001@yahoo.com*
Computer Science Department/MISC Laboratory
University Mentouri Constantine
Constantine, 25000, Algeria

Abstract

Currently, code mobility technology is one of the most attractive research areas. Numerous domains are concerned, many platforms are developed and interest applications are realized. However, the poorness of modeling languages to deal with code mobility at requirement phase has motivated researchers to suggest new formalisms. Among these, we find Labeled Reconfigurable Nets (LRN) [9], This new formalism allows explicit modeling of computational environments and processes mobility between them and It allows, in a simple and an intuitive approach, modeling mobile code paradigms (mobile agent, code on demand, remote evaluation). In this paper, we propose an approach based on the combined use of Meta-modeling and Graph Grammars to automatically generate a visual modeling tool for LRN for analysis and simulation purposes. In our approach, the UML Class diagram formalism is used to define a meta-model of LRN. The meta-modeling tool ATOM3 is used to generate a visual modeling tool according to the proposed LRN meta-model. We have also proposed a graph grammar to generate R-Maude [22] specification of the graphically specified LRN models. Then the reconfigurable rewriting logic language R-Maude is used to perform the simulation of the resulted R-Maude specification. Our approach is illustrated through examples.

Keywords: Code Mobility, Modeling Mobility, Labeled Reconfigurable Nets, Mobile Agent, Graph Transformation, R-Maude, ATOM3 Tool, Maude.

1. INTRODUCTION

Indeed, the formal tools which have been used to model and analyze classical systems are unable to deal with code mobility system properties [8]. Many studies on formal tools have attempted to extend classical tools to deal with code mobility properties. Among these studies we can mention process algebra based models (π-calculus [13], join-calculus [7], HOπ-calculus [16], for example) and state transition based models such as Petri nets. Different works tend to propose methodologies and approaches ([6], [14], [17]…), to prevent ad-hoc development for code mobility software. In effect, these approaches are mostly informal, to deal with this problem, some high level Petri Nets have been proposed. The most known are Mobile Nets (variant of colored Petri nets) [1]. To fit mobile Petri nets to specific mobile systems, various extensions have been proposed such as Elementary Object Nets [18], labeled reconfigurable nets [9], Nested Petri Nets [10] [11], HyperPetriNets [4], … In [3], the authors proposed an approach for transforming mobile UML Statechart diagrams to Nested nets models for analysis purposes. It produces highly-structured, graphical, and rigorously-analyzable models that facilitate early detection of errors like deadlock, livelock, etc … Their approach is based on graph transformation since the input and output of the transformation process are graphs. the meta-modeling tool ATOM3 is used. In this study we will deal with the transformation of LRN models into their equivalent R-Maude specification for analysis purpose.

The rest of the paper is organized as follows. In section 2, we introduce the key concepts that are dealt with in our research. In section 3 we describe our approach which consists of transforming labeled reconfigurable nets models into their equivalent R-Maude specifications. In section 4, we illustrate our approach using examples. Section 5 presents the prototype R-Maude using one of the 2 examples given in section 4. In section 6, we will round off our paper by suggesting some concluding remarks and putting forward the future perspectives for further research.

2. Background

In the following subsections we consider the main concepts and tools used in this paper and give further references for the reader.

2.1 Labeled Reconfigurable Nets

The formalism "labeled reconfigurable net" [9] is dedicated to model code mobility systems [8]. The authors proposed to model mobility in an intuitive and explicit way. Mobility of code (a process or an agent) will be directly modeled through a reconfiguration of the net. The formalism allows adding and deleting places, arcs, and transitions at run time. For more details, the reader can refer to [9].

2.2 Rewriting Logic and Maude

Rewriting logic [12] has been introduced by José Meseguer as logic for describing concurrent systems. It is implemented by several languages such as Maude [12]. The latter is a specification and programming language. It is simple, expressive and has a high-performance implementation. Maude offers three types of modules: Functional modules, System modules and Object-Oriented modules. It offers full Maude to support that; furthermore, it has its own model-checker that is used in checking system's properties. For more details the reader can refer to [12].

2.3 Graph Transformation and ATOM3

Graph grammars [15] can be used to describe graph transformation or to generate sets of valid graphs or to specify operations on them. Graph grammars are composed of production rules, each of them, having graphs in their left and right hand sides (LHS and RHS). Several tools for graph transformation have been proposed in the literature. Among these tools, we can cite ATOM3 "A Tool for Multi-formalism and Meta-Modeling" [2]. The two main tasks of ATOM3 are meta-modeling and model transformation. For more details, the reader can refer to [15].

2.4 Reconfigurable Maude

Maude [19] is a high-level language and high performance system supporting executable specifications and declarative programming in rewriting logic [20].

Maude has been extended to deal with some aspects not considered in former version. Maude [21] is a system to specify and analyze real time and hybrid systems. Mobile Maude [6] is an extension of Maude for mobile systems specification.

An encoding of Labeled Reconfigurable Nets in a Maude-based language has been proposed [22]. The inspired language is called "Reconfigurable Maude" (R-Maude). It profits from the power of Maude (as a meta-language). Maude was extended to support the translation of LRN and their simulation. R-Maude enriches Maude with new kind of rewriting rules. These rules are called Reconfigurable rules (R Rules). The semantic of these rules is similar to that of Reconfigurable transition in LRN. When a rule is executed, the R-Maude specification will be updated in different ways, this will depend on the label associated with this rule.

A specification in R-Maude is a set of Reconfigurable rewrite theories (R-theories). An R-Theory RT is a triple (Ω, E, R) as like a rewrite theory. The difference resides in the set R. R will contain two kinds of rules: standard Rules S-Rules (well known rules of Maude) and Reconfigurable rules R-Rules. An R-Rule rλ is composed of a label λ=<d, RT1, RT2, S> and a rule t□t'. In the label λ, RT1 and RT2 are two R-Theories; S is a segment of a theory, and d a specific parameter. The segment S can be a set of sorts, rules, variables, operators that can be an R-theory or not. When rλ is fired, the specification can be updated in several ways.

Updating specification means that their R-theories will be changed. This change depends on λ. In general, when rλ is fired, the segment S will move from RT1 to RT2 or the inverse. The d parameter can be used to express direction of this move. For more details the reader can refer to [22].

3. OUR APPROACH

The steps of our approach are as follows.

3.1 Meta-Modeling Of LRN (Labeled Reconfigurable Nets)

To build models of LRN formalism in AToM3, we have to define a meta-model for LRN. The meta-formalism used in our work is the UML Class Diagrams and the constraints are expressed in Python code. Since LRN consists of places, transitions, and arcs from places to transitions and from transitions to places, we have proposed to meta-model LRN two Classes to describe Places and Transitions, and two associations for arc-in and arc-out as shown in FIGURE 1. We have also specified the visual representation of each class or association. Given our meta-model, we have used AToM3 tool to generate a visual modeling environment for LRN models. FIGURE 2 shows the generated LRN tool and a dialog box to edit a transition. Each transition has two attributes (label and nom, the attribute label is defined in the case of reconfigurable transition).

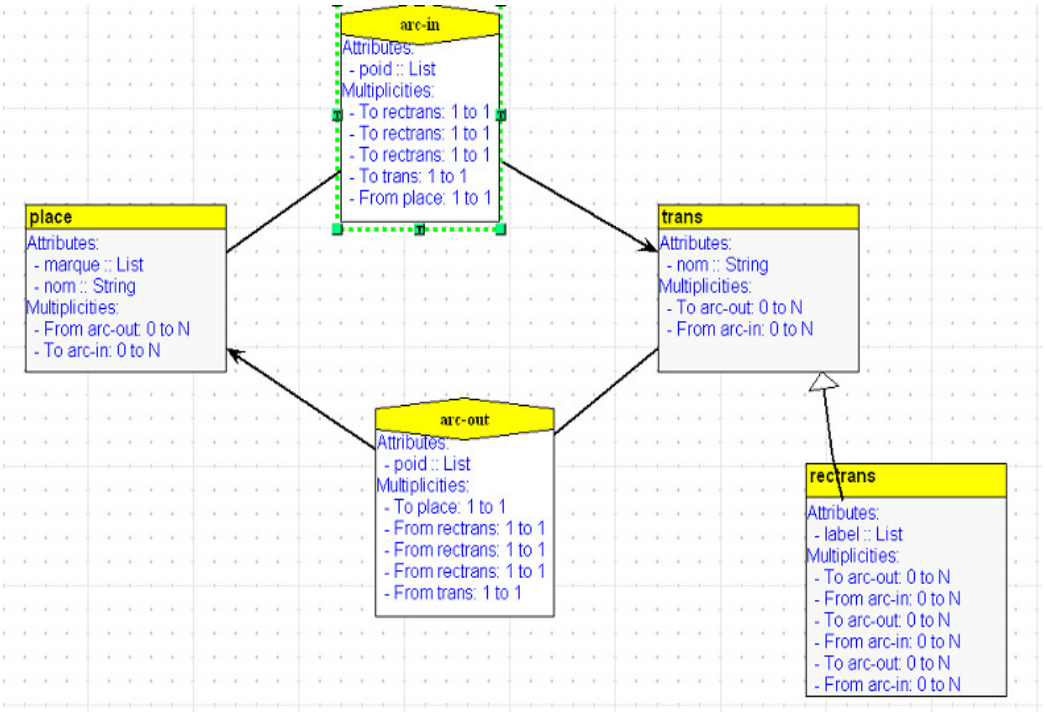


FIGURE 1: LRN Meta-Model

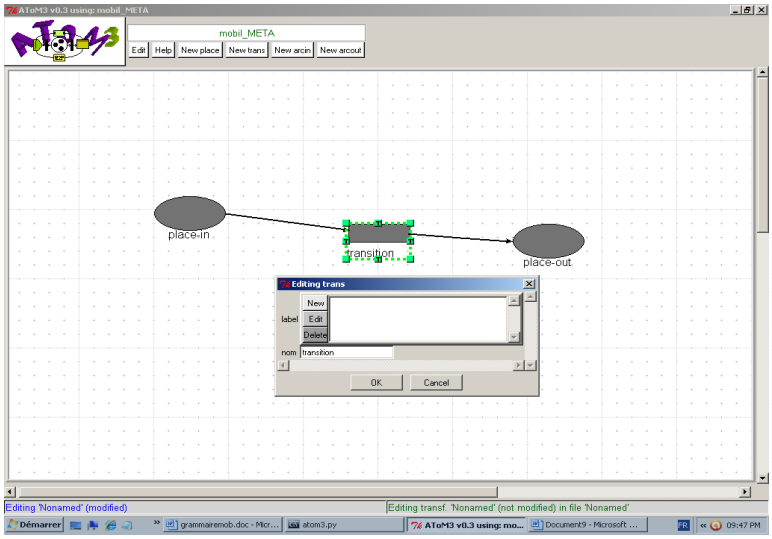


FIGURE 2: Tool bar of the Generated tool to process LRN models

3.2 The Graph Grammar Generating R-Maude Specifications from LRN Models

In order to simulate LRN models, we translate them into their equivalent representations in R-Maude syntax. In this section we show how to use the modeling environment generated in the previous section to generate R-Maude specification. We do this by defining a Graph Grammar to traverse the LRN model and generates the corresponding code in R-Maude. The graph grammar has an initial Action that is used to create the file where the code will be generated. This action decorates also all the Transition and Place elements in the model with temporary attribute according to the conditions specified in the rules. In Transition elements, we use two attributes: current and visited. The current attribute is used to identify the transition in the model whose code has to be generated, whereas the visited attribute is used to indicate whether code for the transition has been generated or not. In Place elements, two attributes are used: fromVisited and toVisited. The fromVisited attribute is used to indicate whether this place is processed as input place whereas the toVisited attribute is used to indicate if this place is processed as output place.

In our graph grammar, we have proposed seven rules (see FIGURE 3) which will be applied in ascending order by the rewriting system until no more rules are applicable. We are concerned here by code generation, so the grammar rules will not change the LRN models. These rules are described as follows:

Rule1: lefthandside (priority 1): is applied to locate a place (not previously processed) related to current transition with an input arc, and generates the corresponding R-Maude specification.
Rule2: separate (priority 2): is applied to generate R-Maude code which separates LHS and RHS of the equivalent rewriting rule.
Rule3: righthandside (priority 3): is applied to locate a place (not previously processed) which is related to current transition with an output arc, and generate the corresponding R-Maude specification.
Rule4: condition (priority 4): is applied to generate the appropriate R-Maude code depending on the condition of the transition, and decorates the transition as visited.
Rule5: InitPlace (priority 5): is applied to locate and initialize temporary attributes in places for processing the next transition.
Rule6: SelectTransition (priority 6): is applied to select an LRN transition not previously processed and generates its equivalent rewriting rule in R-Maude.

Rule7: terminate (priority 7): is applied to perform the writing of the generated R-Maude file and closes it.

The graph grammar has also a final action that erases the temporary attributes from the entities and closes the output file.

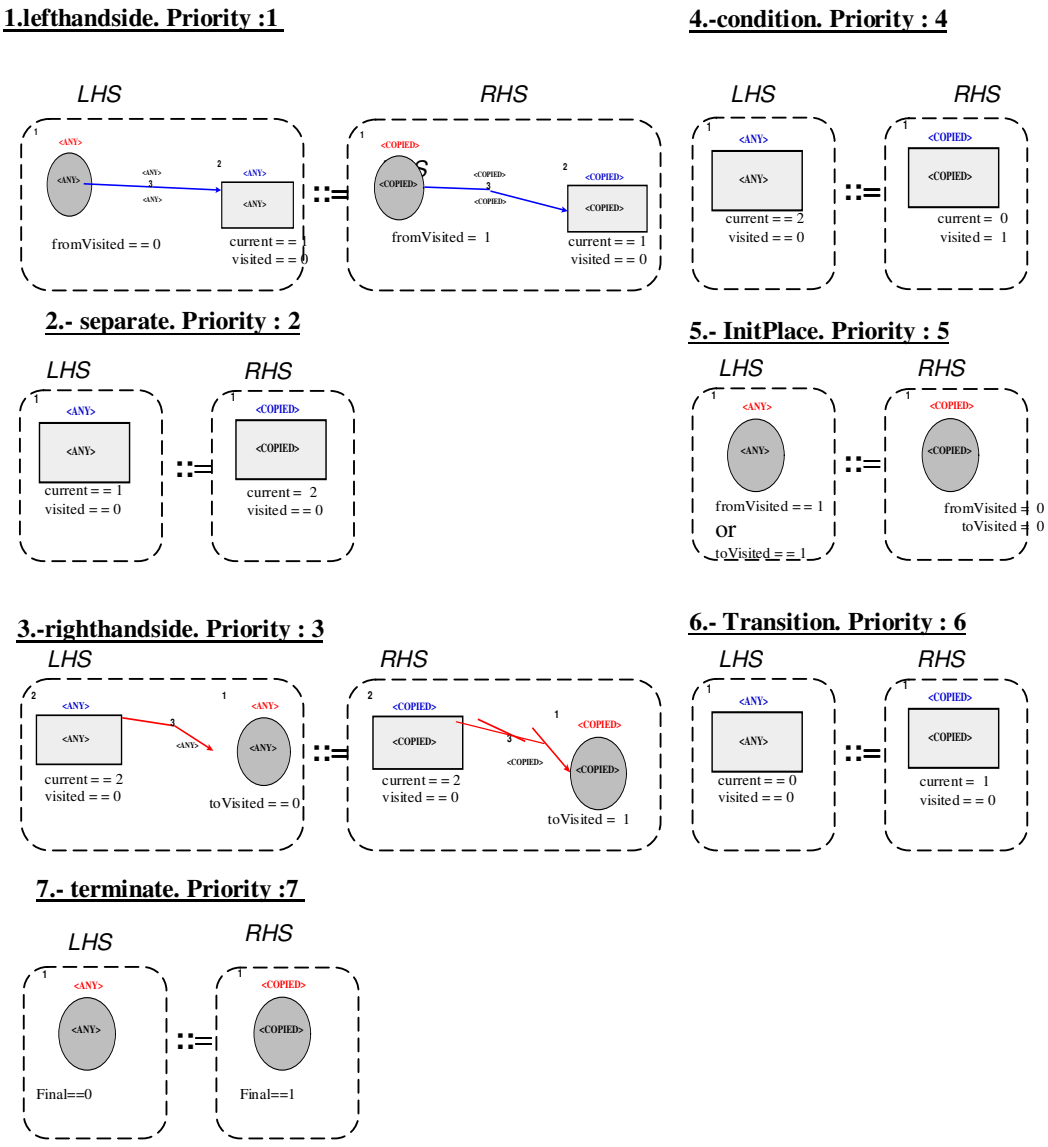


FIGURE 3: The graph grammar rules

4. Examples

4.1 Example 1: LRN With a Static Agent

This example is about a computational environment E1. E1 contains a unique static agent SA1 that execute infinite loop. SA1 requires a non-transferable resource bound by type PNR2

to execute a32. The system will be modeled as a labeled reconfigurable net LRN. FIGURE 4 presents the graphical model of this example created in our tool.

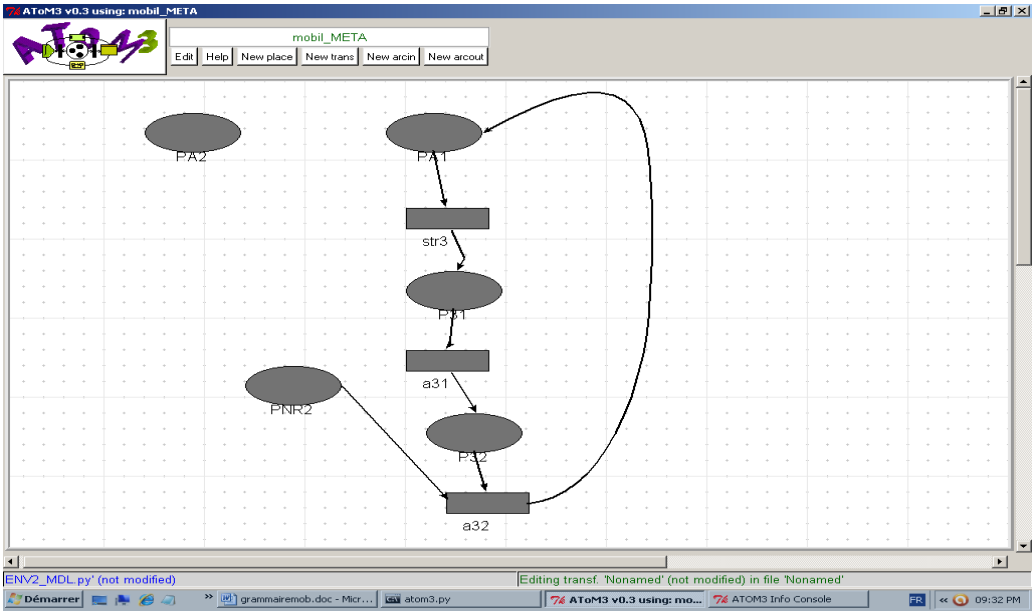


FIGURE 4: The LRN model of the example

Translating LRN Model to R-Maude Specification

This step has graphical representation of an LRN model as input. It consists of translating this graphical representation into its equivalent R-Maude specification using the graph grammar defined in the previous section. To realise this translation, the user have to execute the graph grammar previously defined.

The result of this translation given in FIGURE 5, is the file env2-system.Maude which contains the specification of the LRN model of FIGURE 4.

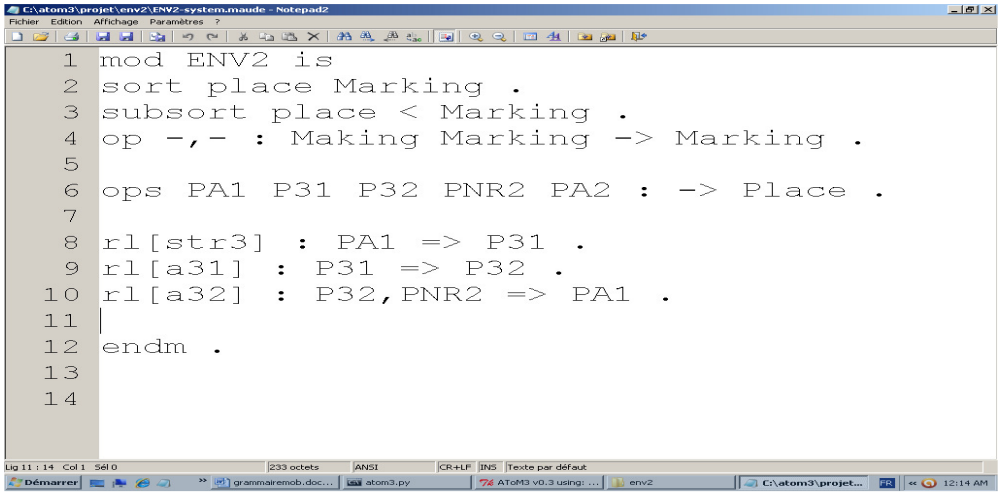


FIGURE 5: Generated R-Maude specification env2-system.maude

4.2 Example 2: LRN With a Mobile Agent

In this example, E1 and E2 are two computational environments. E1 contains two agents, a mobile agent MA and a static agent SA1; E2 contains a unique static agent SA2. The three agents execute infinite loops. MA executes actions {a11, a12, a13 }, SA1 executes actions {a21, a22, a23}, and SA2 executes actions {a33, a32}. To be executed, a11 require a transferable resource TR1 and a nontransferable resource bound by type PNR1 which is shared with a21. a12 and a22 share a transferable resource bound by value, and a13 and a23 share a non-transferable resource NR1. In E2, SA2 requires a non-transferable resource bound by type PNR2 to execute a32. PNR2 has the same type of PNR1. The system will be modeled as a labeled reconfigurable net LRN. LRN contains two environments E1 and E2 that model the two computational environments. In this case the unit A that models the mobile agent A will contain a reconfigure transition rt < A, E1, E2, ψ, β >; such that:

1. E1 =(RP1, GP1, U1, A1); RP1 contains at least four places that model the four resources. Let TR1, NR1, PNR1 and VTR1 be these places. GP1 contains at least a free place PA1 modeling that A can be received, and U1={A}.
2. E2=(RP2,GP2, U2, A2); RP2={PNR2}, GP2={PA2}.
3. ψr={TR1}, ψc={VTR1};
4. β={(PA2, str1), (PNR2, a11), (NR1, a13)}.

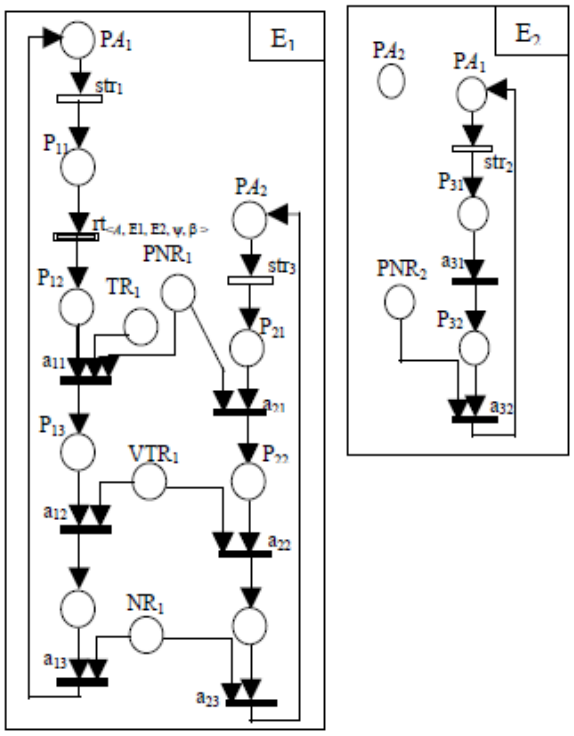


FIGURE 6: LRN MA-model before firing rt

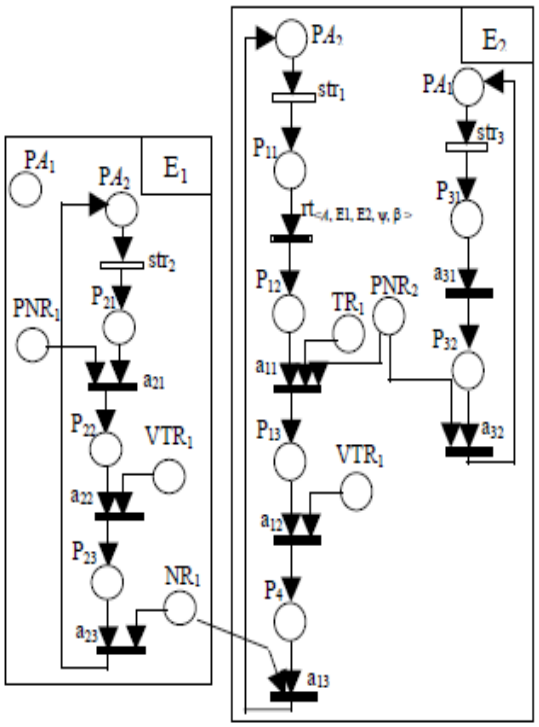


FIGURE 7: LRN MA-model after firing rt

Translating LRN Model to R-Maude Specification

The model of the LRN associated to E1 of FIGURE 6 is too huge in ATOM3, so because of the lack of space, only its R-Maude specification generated by the execution of the graph grammar on the ATOM3 tool and saved in the file env1-system.Maude,is given by FIGURE 8.

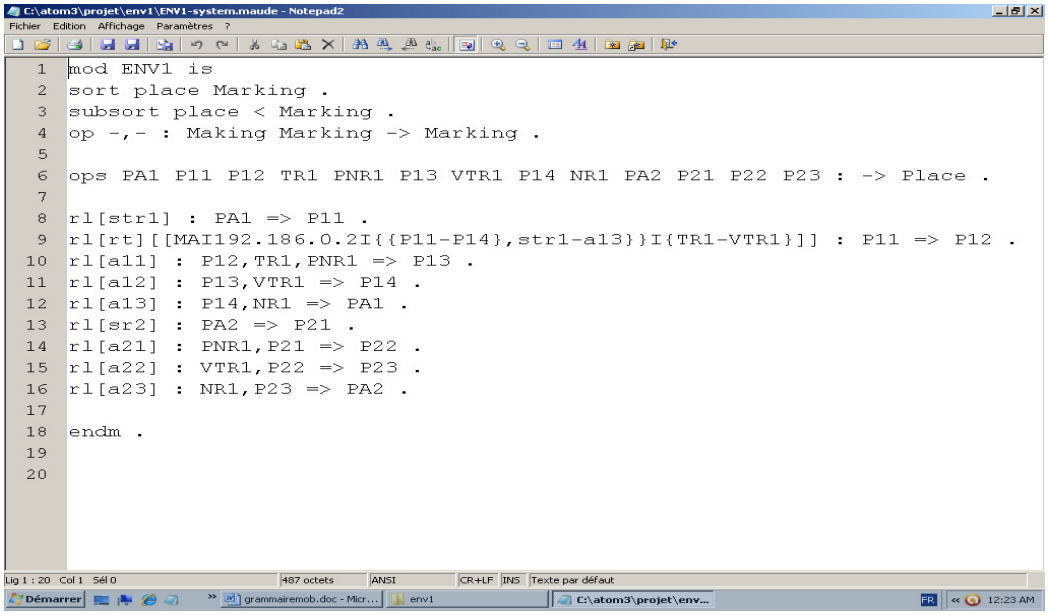


FIGURE 8: Generated R-Maude specification env1-system.maude

5. PROTOTYPING R-MAUDE

R-Maude has been prototyped [22]. The prototype is a system composed from a text editor and an interpreter. The editor is used to enter the specification and commands which have been automatically generated by the execution of the graph grammar on the models to experiment, using ATOM3 tool. The interpreter executes commands and updates specifications. The system was experimented on a LAN (Local Area Network), consisting of a few machines. The system is installed on all hosts. So the specifications are edited everywhere. On every host, commands can be executed. The execution of commands will create the system dynamic. This dynamic can be shown as migration of specification's part (or else the whole) through the LAN.

The specifications (or their parts) are transferred in messages between machines, using UDP protocol. The interpreter realized for R-Maude can be used to interpret Maude specifications. The major different is that in this newest interpreter, the interpretation of R-Rules is added. The label of an R-Rule precedes the rule, and it has the form [MT|L| IP@| S]. Semantics of these parameters is : MT: mobility type (MA, COD, REV, …), L: a multi-set of operations and rules to be moved, cloned or removed from or to the local host, IP@: IP address of distant host, S: sources to move or to remove from or to the local host. When specifications (or part of them) are moved, some resources (R) necessary to firing some rules become far (on another host). IP address of the far host appears with the concerned resource in the form: R[IP@].

The newest is that R-transition will be translated in R-Rules.
We consider that the two environments E1, E2 are specified as two specifications on two hosts (Host1 and Host2). Host1 has the IP address: 192.168.0.1, and Host2 has the IP address: 192.168.0.2.
On Host1, the specification is given by FIGURE 8.
and on Host2, we have the specification :

```
mod E2
sort Place Marking .
subsort Place << Marking .
op _,_ : Marking Marking ->Marking .
ops PA1,PA2,P31,P32,PNR2 : -> Place.
rl [str3] : PA2=>P31 .
rl [a31] : P31=>P32 .
rl [a32] : P32, PNR2=>PA2 .
endmod
```

As an example of a command, we have "rw PA1" on Host1. The execution of this command will produce respectively on Host1, and Host2 the two specifications:

```
mod E1
sort Place Marking .
subsort Place << Marking .
op _,_ : Marking Marking ->Marking .
ops PA1,PA2,P21,P22,P23,VTR1,PNR1,NR1:->Place.
rl [str2]: PA2=>P21 .
rl [a21] : P21, PNR1=>P22 .
rl [a22] : P22, VTR1=>P23 .
rl [a23] : P23, NR1=>PA2 .
endmod.
```

And in Host2 the produced specification s
```
mod E2
sort Place Marking .
subsort Place << Marking .
op _,_ : Marking Marking ->Marking .
ops PA1,PA2,P31,P32,PNR2 : -> Place.
ops VTR1, TR1:-> Place.
ops P11,P12,P13,P14:->Place.
rl [str3] : PA2=>P31 .
rl [a31] : P31=>P32 .
rl [a32] : P32, PNR2=>PA2 .
rl [str1] : PA1=>P11 .
rl [rt][MA|192.186.0.2|{{P11-P14},{str1-a13}}|{TR1,VTR1}] :P11=>P12 .
rl [a11] : P12, TR1, PNR2=>P13 .
rl [a12] : P13, VTR1=>P14 .
rl [a13]: P14,NR1[192.168.0.1]=>PA1.
Endmod
```
Finally, the state of the marking will be : “P12” on the Host2. At this point, the two specifications continue their execution on the two hosts where they reside.

6. CONCLUSION

In this paper, we have proposed an approach and a tool in dealing with the transformation of LRN models to their equivalent R-Maude specifications automatically. This transformation aims to make it possible to achieve the verification of properties of systems modeled using LRN, since the latter do not have tools for analysis and verification. Our approach is based on the combined use of meta-modeling and graph grammars wherein ATOM3 is used as a graph transformation tool.

This study opens up new perspectives for future research. In our forthcoming studies, we plan to hide the steps of the Simulation and thereby spare the user from invoking R-Maude language manually and manipulating the textual version of the simulation result. Thus, the latter will be returned in graphical way to LRN model structure. We plan also, to focus on modeling and analyzing aspects. In the modeling aspects, our concern will be much more on handling problems such as those of modeling multi-hops mobility, process states during travel, birth places and locations. As for the analysis aspect, we are currently working on a denotational semantics for LRN. It is to be underlined that the current R-Maude can be used only to simulate Models. Future works will handle specification analyzing, so we plan to integrate the LTL Maude Model checker in our tool to perform some verification of mobile systems properties, so that Maude model-checker will be adapted to reconfigurable Maude. For LRN, many extensions have been proposed, in [23], authors have proposed “Temporal LRN” and in [24] they have suggested “Coloured LRN”. In line with these suggestions, we focus on using R-Maude to simulate models of these extensions,

7. REFERENCES

[1] A. Asperti, N. Busi. “Mobile Petri Nets”. Technical Report UBLCS-96-10, Department of Computer Science University of Bologna, May 1996.

[2] AToM3 Home page, version 3.00, http://atom3.cs.mcgill.ca.

[3] M. R. Bahri, A. Hettab, A. Chaoui, E. Kerkouche. “Transforming Mobile UML Statecharts Models to Nested Nets Models using Graph Grammars: An Approach for Modeling and Analysis of Mobile Agent-Based Software Systems“. In Proccedings of IEEE SEEFM2009, the 2009 Fourth South-East European Workshop on Formal Methods. Thessaloniki, Greece,
[4] Dec 5th, 2009. pp. 33-39,

[5] M.A. Bednarczyk, L. Bernardinello, W. Pawlowski, L. Pomello. “Modeling Mobility with Petri Hypernets”. 17th Int. Conf. on Recent Trends in Algebraic Development Techniques, WADT’04. LNCS vol. 3423, Springer-Verlag, 2004.

[6] D. Xu, Y. Deng. “Modeling Mobile Agent Systems with High Level Petri Nets”. 0-7803-6583-6/00/ © 2000 IEEE.

[7] F. Dur, N. Steven, E. P. Lincoln, J. Meseguer. “principles of mobile maude”. In D.Kotz and F.Mattern, editors, Agent systems, mobile agents and applications, second international symposium on agent systems and applications and fourth international symposium on mobile agents, ASA/MA 2000 LNCS 1882, Springer Verlag. Sep 2000.

[8] C. Fournet, G. Gonthier. “The Join Calculus: a Language for Distributed Mobile Programming”. In Applied Semantics. International Summer School, APPSEM 2000, Caminha, Portugal, Sep00, LNCS 2395, Springer-Verlag. Aug 2002, pp. 268-332.

[9] A. Fuggetta, G. P. Picco, G. Vigna. “Understanding Code Mobility”. IEEE transactions on software engineering, vol. 24, no. 5, may 1998.

[10] L. Kahloul, A. Chaoui. “Labeled reconfigurable nets For modeling code mobility“. In proceedings of ACIT 2007, Lattakia, Syria.

[11] K. M. van Hee, I. A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve. “Nested Nets for Adaptive Systems”. 14 EE. ICATPN, 2006, pp. 241-260.

[12] 11, I.A. Lomazova. “Nested Petri Nets”; Multilevel and Recursive Systems. Fundamenta Informaticae vol.47, pp.283-293. IOS Press, 2002.

[13] J. Meseguer. “A Logical Theory of Concurrent Objects and its Realization in the Maude Language”. Agha G., Wegner P. and Yonezawa A., Editors, Research Directions in Object-Based Concurrency. MIT Press, 1992, pp. 314-390.

[14] R. Milner, J. Parrow, D. Walker. “A calculus of mobile processes”. Information and Computation, 100:1–77, 1992.

[15] R. Berger, I. Dori, S. Katz.”Modeling code mobility and migration: an OPM/Web approach”, Int. J. Web Engineering and Technology, Vol. 2, No. 1, pp.6–28, 2005

[16] G. Rozengerg, "Handbook of Graph Grammar and computing Graph Transformation", World Scientific, 1999.

[17] D. Sangiorgi, D. Walker. “The π-Calculus: A Theory of Mobile Processes”. Cambridge University Press, 2001.

[18] L. Athie, S. A. DeLoach. “Designing and Specifying Mobility within the Multiagent Systems Engineering methodology ” Special Track on Agents, Interactions, Mobility, and Systems (AIMS) at the 18th ACM Symposium on Applied Computing (SAC 2003). Melbourne, Florida, USA, 2003.

[19] R. Valk. “Petri Nets as Token Objects: An Introduction to Elementary Object Nets”. Applications and Theory of Petri Nets, LNCS vol.1420, pp.1-25, Springer-Verlag, 1998.

[20] M. Clavel, F.Durán, S.Eker, P.Lincoln, N. Marti-Oliet, J.Meseguer, J. Quesada. “Maude:specification and programming in rewriting logic”.SRI International, http://maude,.csl.sri.com, Januray 1999.

[21] J. Meseguer: “Conditional rewriting logic as a unified model of concurrency”. Theoretical Computer Science, 96 (1):73-155, 1992.

[22] P. C. Ölveczky, J. Meseguer:” Real-Time Maude : A tool for simulating and analyzing real-time and hybrid systems”. In K. Futatsugi, editor, Third International Workshop on Rewriting Logic and its Applications, volume 36 of Electronic Notes in Theoretical Computer Science. Elsevier, 2000.

[23] http://www.elsevier.nl/locate.entcs/volume36.html.

[24] L. Kahloul, Allaoua Chaoui. ”LRN/R-Maude Based Approach For Modeling And Simulation Of Mobile Code Systems”. Ubiquitous Computing and Communication Journal, vol. 3, No. 6, Dec. 2008.

[25] L. Kahloul, Allaoua Chaoui. ”Temporal Labeled Reconfigurable Nets for Code Mobility Modeling”. The International Workshop on (Trustworthy Ubiquitous Computing (TwUC 2007)

[26] associated to the iiWAS2007 conference.

[27] L. Kahloul, Allaoua Chaoui. “Coloured reconfigurable nets for code mobility modeling”. In Proc World Academy of Science, Engineering and Technology. Vol. 25, International Conference Venice, Italy. Nov 2007.

INSTRUCTIONS TO CONTRIBUTORS

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. The general topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Starting with volume 3, 2012, IJSE appears in more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJSE LIST OF TOPICS

The realm of International Journal of Software Engineering (IJSE) extends, but not limited, to the following:

- Ambiguity in Software Development
- Application of Object-Oriented Technology to Engin
- Architecting an OO System for Size Clarity Reuse E
- Composition and Extension
- Computer-Based Engineering Techniques
- Data Modeling Techniques
- History of Software Engineering
- IDEF
- Impact of CASE on Software Development Life Cycle
- Intellectual Property
- Iterative Model
- Knowledge Engineering Methods and Practices
- Licensing
- Modeling Languages
- Object-Oriented Systems
- Project Management
- Quality Management
- Rational Unified Processing
- SDLC
- Software Components
- Software Deployment
- Software Design and applications in Various Domain
-
-
- Software Engineering Demographics
- Software Engineering Economics
- Software Engineering Methods and Practices
- Software Engineering Professionalism
- Software Ergonomics
- Software Maintenance and Evaluation
- Structured Analysis
- Structuring (Large) OO Systems
- Systems Engineering
- Test Driven Development
- UML
-

CALL FOR PAPERS

Volume: 3 - **Issue:** 4 - August 2012

i. Paper Submission: May 31, 2012 **ii. Author Notification:** July 15, 2012

iii. Issue Publication: August 2012

CONTACT INFORMATION

Computer Science Journals Sdn BhD
B-5-8 Plaza Mont Kiara, Mont Kiara
50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6207 1607
006 03 2782 6991

Fax: 006 03 6207 1697

Email: cscpress@cscjournals.org

COMPUTER SCIENCE JOURNALS SDN BHD
M-3-19, PLAZA DAMAS
SRI HARTAMAS
50480, KUALA LUMPUR
MALAYSIA

PHONE: 006 03 6207 1607
006 03 2782 6991

FAX: 006 03 6207 1697
EMAIL: cscpress@cscjournals.org