An Evaluation of Maintainability of Aspect-Oriented Systems: a Practical Approach

Avadhesh Kumar avadheshkumar@aiit.amity.edu
Amity Institute of Information Technology
Amity University
Sec-125, Noida, India

Rajesh Kumar rakumar@tiet.ac.in
School of Mathematics & Computer Applications
Thapar University
Patiala, Punjab, India

P.S. Grover groverps@hotmail.com
Guru Tegh Bahadur Institute of Technology
GGS Indraprastha University
Delhi, India

Abstract

Maintenance of software systems is becoming major concern for software developers and users. In software projects/products, where software changes/updates are frequently required to improve software quality, maintainability is an important characteristic of ISO 9126 quality standard to evaluate. Analyzability, changeability, stability, and testability are sub attributes/characteristics of maintainability in ISO 9126. In this paper, changeability is measured by making changes at code level of an Aspect-Oriented (AO) system. The approach taken to evaluate the changeability of an AO system is to compute the impact of changes made to modules of the system. Some projects[1] in aspect-oriented programming (AOP) language, AspectJ, have been taken for testing. The results suggest that the AO system can easily absorb changes and AO design metrics can be used as indicators of changeability as well as of maintainability. The results also suggest that a code level change in AO systems not always cause less change impact to other modules than a code level change in Object-Oriented (OO) systems.

Keywords: Software quality, maintainability, changeability, AO system, AO metrics, AspectJ.

[1]Original three different projects were developed using object-oriented programming language Java/Servlets for a university with logging facility, chat facility, student result, etc. These projects are having 129 classes. Same projects are re-engineered to aspect-oriented programming using AspectJ. AO projects are with 149 modules (classes and aspects).

1. INTRODUCTION

Software quality refers to the conformance of the product to explicitly state functional and performance requirements, documented development standards, and implicit characteristics. Quality of software project/product is characterized by certain attributes, which are highlighted by ISO standards. An example of such standard is ISO 9126.

ISO 9126 is a standard that provides a generic definition of software quality, in terms of six main desirable characteristics: *functionality, maintainability, usability, efficiency, reliability,* and *portability* [1, 2]. Extensive studies have shown that maintenance is one of the major cost concerns, as a matter of fact; a growing cost concern [3]. Maintainability has further four sub attributes, *analyzability, changeability, stability,* and *testability*. Out of these, changeability is the most significant from the point of view of organizations, as most organizations use software, developed by other organization(s). It need not bother about any other attribute, except *changeability.* When we make changes in a program at various levels, such as design, code, and architecture and so on, then how do these affect the quality of the software? Carrying out the impact analysis based on the various changes made can evaluate this.
Maintainability of a software system depends on its design [4], which depends on the software design approach that one uses. Salient design approaches are: Module-Oriented (MO), OO, and AO [5, 6]. MO and OO paradigms have been used quite commonly and well-accepted in industry. Each has its own limitations and range of applicability. One of the major constraints has been the spread of concerns over various modules/classes (cross-cutting concerns). This leads to program codes, which are difficult to maintain and understand [7].
Aspect-Oriented Programming (AOP) is a new approach for separating concerns into units called *aspects*. An aspect is a modular unit of crosscutting concern implementation. It encapsulates behaviors that affect multiple classes into reusable modules [8, 9]. We implement AOP by OO language (e.g. Java), and then we deal separately with crosscutting concerns in our code by implementing aspects. Finally, both the code and aspects are combined into a final executable form using an aspect weaver. As a result, a single aspect can contribute to the implementation of a number of methods, modules, or objects, increasing both reusability and maintainability of the code. The original code need not know about any functionality of the aspect that has been added, it needs only to be recompiled with the aspect to regain the original functionality. It is being argued that AOP will lead to better quality software.

Most research work on change impact assessment has been carried out on MO and OO software [10, 11, 12, 13, 14], whereas AO approach has not been studied to that extent. Zhao [15] did some work in this area based on *program slicing* technique, but has not applied to realistic systems. Avadhesh et, al.[16] have measured changeability characteristics only for operation signature change, not for other members of the module and changes at system level. We have explored this problem incorporating code–level changes for all types of members inside module as well as at system level of AOP. Our technique to assess change impact for AO systems is different. We have used new terminologies for class(s) and aspect(s) as *modules* and for method(s) of class and advice(s)/introduction(s) of aspect as *operations*. A change in access scope, data types, operation signature etc. will impact other modules. We evaluated change impact on modules occurred due to a syntax change in code. We have taken projects developed in AspectJ, as a case study.

2. RELATED WORK

Characterization of design is mostly done through metrics. According to Rombach, architecture is more influencing than algorithmic design on maintainability [17]. For AO design, many design metrics has been proposed and published. Ceccato and Tonella [18] have proposed metrics, which include ten different metrics for AOP: Weighted Operations in Module (WOM), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling on Advice Execution (CAE), Coupling on Intercepted Modules (CIM), Coupling on Method Call (CMC), Coupling on Field Access (CFA), Response for a Module (RFM), Lack of Cohesion in Operations (LCO) and Crosscutting Degree of an Aspect (CDA). Zakaria and Hosny [19], proposed the effects of AO on the C&K metric suite, which are: Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Lack of Cohesion in Methods (LCOM), Coupling Between Objects (CBO), and Response For a Class (RFC). Zhao [20] has defined another set of complexity metrics in terms of program dependence relations to measure the complexity of an aspect-oriented program from various viewpoints. Once the dependence graph of aspect-oriented program is constructed, the metrics can be easily computed in terms of dependence graph. According to Zhao, following are some salient metrics type designed to measure complexity from various viewpoints:

Module-Level Metrics: Module-level metrics are designed based on advice dependence graph (ADG), Introduction dependence graph (IDG) and method dependence graph (MDG).

Aspect-Level Metrics**:** Aspect level metrics can be defined for an individual aspect based on its aspect inter-procedural dependence graph (AIDG).

System-Level Metrics**:** System-level metrics can be defined at the whole system level based on aspect-oriented system dependence graph (ASDG).

Li and Offut [21] proposed algorithms for calculating the complete impact of changes made in a given class. They explored the effects of encapsulation, inheritance, and polymorphism. Hsia et al [22] studied the effect of architecture on maintainability. They measured maintainability and its relationship to architecture, especially broadness of the architecture trees. As a result, they found, that maintainability is better for systems with broader trees. Chaumun et al [23]'s change impact model for changeability assessment in object-oriented software systems is applied to programs in C++. In this work, for each of the possible changes identified in C++, the impact is calculated so that necessary actions may be taken to ensure a successful system compilation after change implementation.

Jingyue Li et al [24] have studied how AOP eases the adding and replacing the components in COTS-based development. When adding or replacing a COTS component, the main benefit of using AOP in a COTS-based system is that fewer classes need to be changed that using Object-Oriented Programming (OOP). However, using AOP does not ensure that less Lines Of Code (LOC) need to be modified when adding or replacing COTS components. It depends on whether the crosscutting concerns in the glue-code are homogeneous. Using AOP when crosscutting concerns are heterogeneous may not be benificial.

3. ASPECTJ

AspectJ [25] is simple general-purpose extension to Java that provides, through the definition of new constructors, support for modular implementation of crosscutting concerns. It enables plug-and-play implementations of crosscutting concerns [26]. AspectJ has been successfully used for modularizing the crosscutting concerns such as synchronization, consistency checking, protocol management and others. AspectJ supports the definition of aspects' *join points, pointcuts, advice and introduction* [27].

Join points: Join points represent well-defined points in a program's execution. Typical join points in AspectJ include method calls, access to class members, and the execution of exception handler blocks. Join points may contain other join points. For example, one method call may result in several other method calls before it returns.

Pointcuts: Pointcut is a language construct that picks out a set of join points based on defined criteria. The criteria can be explicit function names, or function names specified by wildcards.

Advice: Advice is code that executes *before, after,* or *around* a join point. You define advice relative to a pointcut, saying something like "run this code before every method call I want to log."
Introduction: introduction allows aspects to modify the static structure of a program. Using introduction, aspects can add new methods and variables to a class, declare that a class implements an interface, or convert checked to unchecked exceptions.

4. CHANGE IMPACT ANALYSIS

Change impact analysis is the task through which the programmers can assess the extent of the change, i.e. the software component that will impact the change, or be impacted by the change. Change impact analysis provides techniques to address the problem by identifying the likely ripple effect of software changes and using this information to re-engineer the software system design [28].

From the viewpoint of separation of *concerns* in software development, change impact analysis can be performed at many levels of software systems during software evolution, at the specification level, design level, architecture level, code level etc. Our work is focused on code level change impact for AOP and the language for this work we have chosen is AspectJ.

4.1 Code Level Changes in AspectJ

Following are possible code level changes in AspectJ:

4.1.1 System level change

- Add super module
- Delete super module
- Add sub module
- Delete sub module
- Add a module reference
- Delete a module reference
- Add an aggregated module
- Delete an aggregated module

4.1.2 Module level change

- Add member
- Delete member
- Define/Redefine member
- Change member
  - Change member access scope
  - Change operation signature
  - Change data member
  - Operation implementation change
- Change pointcut
  - Add pointcut
  - Delete pointcut
  - Signature change of a pointcut

With a single change, we are interested in knowing which other parts (operations) in the rest of the system will be affected by this change. A specific part may be affected, in case it is 'connected' to the changed component via some link(s) between them. Following are four types of links:

Association (S): One module is referencing data variables of another module.
Aggregation (G): It is established between two modules when a module definition is based on objects of another module.
Inheritance (H): inheritance between two modules means that the derived module can benefit from whatever has already been defined in the base module.
Invocation (I): When operations defined in one module are being invoked by operations in another module.
We have also considered for impact with in the changed module itself. This type link could be represented as '**Local' (L)** link.

4.2 Change impact evaluation

Module change impact is a numeric value used to express the impact level of one module to others. It considers the factor of contaminates type and relationships among impacted module. For example consider a change in the scope of an attribute from *public* to *default*. Modules, which are accessing this attribute from different package, will be impacted but modules, which are accessing from the same package, will not. Similarly, a change in the signature of a pointcut in any module will impact all those modules, which have a join point with matching signature of pointcut. And modules having advice for this pointcut will not be impacted. Adding a new advice, for which there are no join points, will cause no impact to any of the modules and change impact will be zero.

At this point, we are emphasizing on the type of impact and we are looking for some code level in AspectJ systems. A given change is characterized by a transformation of the code somewhere in the system. If the system is successfully re-compiled, then there is no impact. Otherwise, we are faced with an impact, i.e., code modifications that must be done elsewhere in the system to obtain a syntactically correct code that will re-compile. Semantic issues relating to the code transformation are overlooked at this point because they cannot be inferred from the source code alone. For example, if a variable is added but not used later, we may feel that this addition is useless. But, from a syntactic point of view, we are indeed certain that the system will stand good after re-compiling. Furthermore, since our focus is only on system compilation after a change, the appropriate measures we have to apply are based on impact that is only dependent on the static nature of the source code.

To calculate the impact of each identified change, a truth table is set up for that change with the five links appearing in section 4.1. For each row, representing one configuration of these five links, we investigate whether there is impact or not, and the row is marked accordingly. In some cases, it may happen that the state underlying the row cannot exist, and the row is left unmarked. For example, when there is a change in the return type of an abstract method, the rows in which **G** or **I** appear cannot be investigated since neither the abstract class can be instantiated as an object (**G**) nor the abstract method can be invoked (**I**). For each row, the appropriate Boolean expression is derived and reduced, if possible, and the term "***L***" is appended if there is local impact. For example, the change impact formulae for a change to each component type are as follows: (i) *Impact* (Attribute deletion) = **S+*L*,** means impacted modules will be modules associated with attribute or local impact ***(***ii) *Impact* (operation scope change from public to protected) = **IH',** means impacted modules will be the modules invoking this operation and not inherited modules from the module having this operation. (iii) *Impact* (Class deletion) = **H+G+S+I,** means impacted modules will be inherited or aggregated or associated or due to invocation.

In this paper, we have considered changes at system level as well as at module level. The primary goal of the experiment is to analyze empirically whether an AO design metric has any relationship with the impact of a change for the test system. The change considered is the operation signature change; the Boolean expression of its impact is **I**, meaning there is impact in modules where the operation is invoked. The impact is calculated for the operation signature change on every operation defined in a targeted module, summed for all the operations defined in that module, and divided by the number of operations of that module. We will call this average value "change- impact" of the module. The metric chosen to correlate is the WOM metric, which in our experiment, is equal to the number of operations defined in a module.

5. AOP CHANGE IMPACT: A CASE STUDY

5.1 System tested

AOP projects that we have taken for testing, having 149 modules, are re-engineered using AspectJ, which originally were university projects developed using Java/Servlets having 129 modules. We extracted the Ceccato and Tonella metrics [15] with the help of the tool developed and provided by Ceccato and Tonella, for AOP metrics, which computes all the proposed measures for code written in the AspectJ language. The tool exploits a static analyzer developed in TXL [29]. The descriptive statistics of the metrics distribution is given in table-I. Waited Operations per Module (WOM) metric refers to the sum of the complexities of all the operations defined in a module. We have assumed operation complexity to be one for all the operations or we can say WOM have been taken as Number of Operations per Module (NOM).

Table-I: Summary of AOP metrics for the tested systems

	Min	Max	Mean	Median	Std. Dev.
WOM	0.0	17.0	3.2	2.0	0.33
DIT	0.0	2.0	0.36	0.0	0.048
NOC	0.0	26.0	0.31	0.0	0.25
CAE	0.0	3.0	0.45	0.0	0.046
CIM	0.0	1.0	0.34	0.0	0.046
CMC	0.0	6.0	1.39	1.0	0.144
CFA	0.0	3.0	0.22	0.0	0.592
RFM	0.0	4.0	1.06	1.0	1.131
LCO	0.0	7.0	1.04	1.0	0.56
CDA	0.0	39.0	1.0	0.0	0.381

Table-I cont..

We have categorized 149 modules in three groups.

- Group 1: Modules contain 1 to 2 operations (51 modules).
- Group 2: Modules contain 3 to 7 operations (62 modules).
- Group 3: Modules contain at least 8 operations (36 modules).

In our sample projects, Inheritance level is not too high. It is maximum 3 and average inheritance level is less than 1. Average numbers of operations per module are 3.2, which indicates, proper decomposition has been taken care.

We have tested 44 modules from Group 1, 40 modules from Group 2, and 20 modules from group 3 randomly. Most of the changes performed are at module level and few changes are at system level. Total numbers of tested modules that we have performed randomly in this case study are 104. On these 104 tested modules, we have evaluated change impact.

5.2 Impact Results

In all 104 tested modules, the change impact numeric value is given in fig. 1. Minimum change impact is 0.0 and maximum is 7.50. Average change- impact with a single change is 0.77 for all three projects. Table-II summarized descriptive statistics of the impact results for the modules. The mean value of module impact increases from group 1 to group 3. In group 1, majority of the programs are with aj extension (Aspects) and in group 3, majority of the programs are with java extension (classes). We have calculated statistically values like mean, median and standard deviation of the change impact values for the modules.

Table-II: Descriptive statistics of the impact results for the three groups

	Group1 (1-2 operations)	Group2 (3-7 operations)	Group3 (7+ operations)
Total Module Present	51	62	36
Total Module Tested	44	40	20
Impact	Module	Module	Module
Min	0.00	0.00	0.50
Max	7.50	3.00	5.33
Mean	**0.53**	**0.64**	**1.56**
Median	0.00	0.00	1.50
Std.Dev.	0.20	0.13	0.28

We have also evaluated correlation factor between metric WOM and change impact, which is 0.41.

Similarly we tested original OO projects, which were developed with Java/Servlets. Out of 129 modules (classes), we tested 104 modules randomly and evaluated average change impact as a whole for all three projects, and it was found to be 0.87. We also separated above change impact data project wise and evaluated average value of change impact for OO and AO systems. Descriptive statistics of the average impact results for the three projects are given in table-III. In

OO systems, project 1, project 2 and project 3 are with 29, 64 and 36 classes respectively and out of these 20, 52 and 32 classes are tested .In AO systems, project 1, project 2 and project 3 are with 34, 72 and 43 modules respectively and tested modules are same as in OO systems i.e. 20, 52 and 32 respectively.

Table-III: Descriptive statistics of the impact results for the three projects

	Project 1 (20 modules tested)		**Project 2 (52 modules tested)**		**Project 3 (32 modules tested)**	
	OO	AO	OO	AO	OO	AO
Average change impact	0.89	0.75	0.76	0.81	0.91	0.78

6. RESULTS

Interpretation of the result is as follows:

I. Since mean value of change impact is 0.77 for whole systems which is less than 1, which means a single change at code level will impact, on average, not more than one module.
II. Mean value of change impact increases from group 1 to group 3 that means change impact increases with increase in number of operations in the module i.e. in AO systems, if number of operations per module are increasing then such systems' maintainability will increase. Thus such systems are required to decompose properly.
III. Average change impact in AO systems is less than the average change impact in OO systems as a whole that means AO systems are easily maintainable than OO systems. But when we evaluated it project wise, we found that in project 2, OO system mean change impact is less than that of AO system, that means a code level change in AO systems not always cause less change impact to other modules than a code level change in OO system or in other words in some cases OO system is easily maintainable than AO system. This empirical result may be because of aspect mining has not been taken care properly.
IV. Correlation factor with change impact and WOM is found to be 0.41, which is not too high, which means there is not too strong relationship between WOM and change impact. So, WOM metric can be used as an indicator for changeability analysis, but not too strong indicator for changeability characteristic.

7. CONSLUSION & FUTURE WORK

In this paper, we measured the changeability characteristic of AO software projects. We evaluated the change impact with real system. Projects that we have considered for testing are AspectJ projects. The change impact is evaluated for each of the possible code level changes so that required changes should be made to ensure a successful system compilation after change implementation.

Result shows that a single change at code level will cause impact to other modules.On an average change impact value is less than one; this implies that not more than one module is impacted with a single change or we can say a change is easily absorbable in AO system. By increasing in WOM metric value, change impact is also increasing. It indicates that with increase in WOM value, will cause increase in maintainability. Correlation factor between WOM and change impact is found to be 0.41, which is week. It indicates that WOM can be used as an indicator for changeability or maintainability but not as a strong indicator.

Average change impact in AO system was found less than that in OO system, which suggests that AO system can absorb more changes compare to OO system. In other words, AOP are easily maintainable than OOP. But if at the time of reengineering OO system to AO system, concerns which are not crosscutting, are mined to aspect, may cause resultant system more complex. In such cases AO systems maintainability will be more difficult than that of OO systems. In future, this technique may be used to compare maintainability of different AO Systems.

8. REFERENCES

[1]. Jorgen Boegh, Stefano Depanfilis, Barbara Kitchenham, Alberto Pasquini, “A Method for Software Quality Planning, Control, and Evaluation” IEEE Journal, pp-69-77, March-1999.

[2]. Ho-Won Jung; Seung-Gweon Kim; Chang-Shin Chung, “Measuring software product quality: a survey of ISO/IEC 9126”, Software, IEEE,Volume 21, Issue 5, pp-88-92, Sep-Oct-2004.

[3]. T. M. Pigoski. Practical Software Maintenance. John Wiley & Sons, New York, pp-384, 1997.

[4]. H.D. Rombach. “Design measurement: Some Lessons Learned”. In IEEE Software, Vol. 7, No. 2, pp- 17-25, 1990.

[5]. Tzilla Elrad, Robert E. Filman, Atef Bader, “Aspect-oriented programming: Introduction”, Communication of the ACM Volume 44, Issue 10, pp-29-32, October 2001.

[6]. K. Lieberher, D. Orleans, and J. Ovlinger, “Aspect- Oriented Programming with Adaptive Methods,” Communications of the ACM, Vol.44, No.10, pp.39-41, October 2001.

[7]. Avadhesh Kumar, Rajesh Kumar, P.S. Grover, “A Comparative Study of Aspect-Oriented Methodology with Module-Oriented and Object-Oriented Methodologies”, ICFAI Journal of Information Technology, Vol IV, pp- 7-15, Dec 2006.

[8]. Aldrich, J., Open Modules: “A Proposal for Modular Reasoning in Aspect-Oriented Programming”, Carnegie Mellon Technical Report CMU-ISRI-04-108, 2004 (Earlier version appeared in Workshop on Foundations of Aspect-Oriented Languages.).

[9]. M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, & T. Tourw, “A qualitative comparison of three aspect mining techniques”, Proc. of the International Workshop on Program Comprehension (IWPC), 2005. Proceedings. 13th International Workshop on 15-16, pp-13-22, May 2005.

[10]. S. Iorwitz, T. Reps and D. Binkley, " Interprocedural Slicing Using Dependence Graphs", ACM 7~ransoction on Programming Language and System, Vol.12, No.l, pp.25-fiO, 1990.

[11]. D. Hung, J. Gao, P. Hsia, F. Wen, Y. Yoyoshima, and C. Chen, "Change Impact Identification in Object-Oriented Software Maintenance," Prec. Iutervmtional Conference on Software Mointenonce~ pp.202-211, 1994.

[12]. L. D. Larsen and M. J. Harrotd, "Slicing Object-Oriented Software," Proceeding of the 18th International Conference on Software Engineering, German, March, 1996.

[13]. J. P. Loyall and S. A. Mathisen, "Using Dependence Analysis to Support the Software Maintenance Process," Prac. International Conference an SoJimarc Maintenance, 1993.

[14]. J. Zhao, "Slicing Concurrent Java Programs," Prec. Seventh IEEE International Workshop on Program Comprehension, pp.12fi-133, May 1999.

[15]. Jianjun Zhao, “Change Impact Analysis for Aspect-Oriented Software Evolution” Proceedings of the International Workshop on Principles of Software Evolution, pp. 108-112, 2002.

[16]. Avadhesh Kumar, Rajesh Kumar, P.S. Grover, “A Change Impact Assessment in Aspect-Oriented Software Systems”, International Software Engineering Conference Russia 2006 (SECR-2006), pp-83-87, Dec 2006.

[17]. H.D. Rombach. “Design measurement: Some Lessons Learned”, In IEEE Software, Vol. 7, No. 2, pp- 17-25, 1990.

[18]. M. Ceccato & P. Tonella, “Measuring the effects of software aspectization” , Proc. 1st Workshop on Aspect Reverse Engineering in conjunction with the 11th IEEE Working Conf. on Reverse Engineering, Delft University of Technology, Netherlands, November 9th, 2004.

[19]. A. A. Zakaria and H. Hosny. “Metrics for aspect-oriented software design”. In AOM: Aspect-Oriented Modeling with UML, AOSD, March 2003.

[20]. J. Zhao. “Towards A Metrics Suite for Aspect-Oriented Software”, Technical-Report SE-2002, Information Processing Society of Japan (IPSJ), pp. 136-25, 2002.

[21]. L. Li and A. J. Offutt. “Algorithmic Analysis of the Impact of Changes to Object-Oriented Software”, in ICSM96, pp- 171-184, 1996.

[22]. P. Hsia, A. Gupta, C. Kung, J. Peng and S. Liu. “A Study of the Effect of Architecture on Maintainability of Object-Oriented Systems”, In ICSM95, Nice, France, pp- 4-11, Oct 17-20, 1995.

[23]. M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, and François Lustman., “A Change Impact Model for Changeability Assessment in Object-Oriented Systems”**,** Science of Computer Programming, vol.45 pp- 155-174, 2002. Elsevier Science Publishers.

[24]. Jingyue Li, Axel Anders Kvale and Reidar Conradi “A Case Study on Improving Changeability of COTS-Based System Using Aspect-Oriented Programming”, Journal of Information Science and Engineering, Vol. 22 No. 2, pp- 375-390, March 2006.

[25]. The AspectJ Team. The AspectJ Programming Guide. 2003.

[26] V. C. Garcia, E. K. Piveta, D. Lucrédio, A. Álvaro, E. S. Almeida, L.C. Zancanella, & A.F. Prado, “Manipulating crosscutting concerns” , Proc. 4th Latin American Conf. on Patterns Languages of Programming (SugarLoafPLoP), Porto das Dunas, CE, Brazil, 2004.

[27]. Kiczales, G. et al. “Getting Started with AspectJ”. Communication of the ACM, vol. 44, no.10, pp-59-65, October 2001.

[28]. M. A. Chaumun. “Change Impact Analysis in Object-Oriented Systems: Conceptual Model and Application to C++”, Master’s thesis, Université de Montréal, Canada, November 1998.

[29]. J. Cordy, T. Dean, A. Malton, and K. Schneider. “Source transformation in software engineering using tihe TXL transformation system”, Information and Software Technology, 44(13): pp-827–837, 2002.