An Empirical Study of Maintenance Activities in an Object Oriented System

Wei Li and Sallie Henry

TR 93-13

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

April 18, 1993

An Empirical Study of Maintenance Activities in an Object Oriented System

Wei Li

Kollmorgen Industrial Drives
201 Rock Road
Radford, VA 24141
(703) 639 - 2495
li@vyopus.cs.vt.edu

Sallie Henry

Computer Science Department
Virginia Tech
Blacksburg, VA 24060
(703) 231-7584
henry@vtopus.cs.vt.edu

Abstract

Decades of research on maintenance activities in the procedural paradigm has produced several conclusions. Among these conclusions are recommendations that a reduction in maintenance cost could be achieved by a more controlled design process, by more rigorous testing of potential problem areas earlier in the life cycle. With the increased emphasis on the object oriented paradigm, the authors performed an empirical study of the maintenance patterns in a commercial object oriented system. Although this is a preliminary study, intuition is presented as insight into the object oriented maintenance activities.

I. Introduction

Software maintenance has been recognized as the most costly and difficult phase in the software life cycle [MUNJ81] [SCHN87]. Some of the causes of software maintenance problems are 1) software maintainability is not a major consideration at the software design and implementation phases [BENS87] [MUNJ81] [SCHN87], 2) maintenance has been largely ignored in software life cycle models [BENS87] [MUNJ81], and 3) poor understanding of maintenance activities.

Various studies have attributed to the understanding of maintenance activities in the procedural paradigm [MUNJ81] [SCHN87] [BENS87] [KAFD85]. There have been very few studies addressing the maintenance activities in the object oriented paradigm. This paper addresses one aspect of the maintenance activities in the object oriented paradigm - the maintenance activities at different inheritance levels. In particular, different types of maintenance are studied at each inheritance level.

Different events can trigger a maintenance request for software. Swanson classifies software maintenance activities into three categories : *corrective*, *perfective*, and *adaptive*. "Corrective maintenance is an activity which would not be performed at all, were it not for the occurrence of failures...Maintenance performed in response to changes in data and processing environments may be termed adaptive maintenance...Maintenance performed to eliminate processing inefficiencies, enhance performance, or improve maintainability may be termed perfective maintenance" [SWAB76].

This research uses Swanson's classification of maintenance activities and make observations of the maintenance activities in a commercial software system.

II. Basic Concepts in Object Oriented Programming

Terminologies vary among object oriented programming languages. However, all object oriented languages share some common concepts. Different classifications of the object oriented paradigm exist [COXB86] [WEGP87] [KORT90]. However, the characteristics of the object oriented paradigm considered in this research include the concepts of *object*, *class*, *attributes*, *inheritance*, *method*, and *message passing*.

Objects are the basic program components in the object oriented paradigm. An object is

an instance of a class. An object consists of some data attributes and operation attributes which are defined in the class. When an object is created from a class, the memory space allocated for all the data attributes can be used and the methods defined in the class can then be invoked. The attributes are encapsulated within the object and should be accessed through the operations (methods). Objects sharing the same definitions of attributes and operations (methods) are grouped into classes.

A *class* encapsulates data structures and operations defined for the manipulations of the data structures. The data structures are the data attributes of a class. The operations are the operation attributes of a class. A class defines the template for *objects* which are the basic system components in the object oriented paradigm.

Attributes represent the properties that a class owns. There are normally two types of attributes in a class: *data attributes* and *method attributes*. Data attributes are the data structures a class encapsulates. Method attributes are the operations (methods) defined in a class. There is only one data attribute defined in the object at line 5: an integer variable.

Inheritance allows the definition of a class to use that of an existing class. Classes are connected through an inheritance hierarchy. The inheritance level (inheritance depth) indicates where the class is in the hierarchy. A class may be a sub-class of another class. A sub-class may inherit some data attributes as well as operation attributes from all its superclasses.

Methods are also known as operations in a class. Methods are defined to provide a means for other classes to access the attributes defined in a class. The set of methods forms the interface (services) that a class provides to other classes.

Message passing is a common communication mechanism among objects. Whenever an

object requests a service that another object provides, it sends a message to the other object.

III. Classic-Ada Design/Programming Language

This research analyzed one object oriented system, UIMS, designed and developed using Classic-Ada. Classic-Ada is an object oriented design/programming language. It brings the capability of object oriented programming to Ada by providing object oriented constructs in addition to the Ada constructs.

Classic-Ada supports all the standard constructs defined in ANSI/MIL-STD-1815A. In addition to the standard Ada constructs, Classic-Ada supports the object oriented features with nine new constructs. The nine new constructs are class, method, instance, superclass, send, self, super, instantiate, and destroy. Classic-Ada supports the following principles:

Information Hiding: Hides the state of a software component (class object) in variables visible only within the scope of that component.

Data Abstraction: Supports abstract data types that define an internal representation plus a set of operations used to access and manipulate the internal representation. Data abstraction encapsulates the abstract data type with its operations within an object.

Dynamic Binding: Determines which operation is invoked for a specific object dynamically, at run-time, depending on the object being manipulated.

Inheritance: Enables the extension or combination of less complex objects into more complex ones.

IV. Statistical Analyses

Some empirical observations of the maintenance activities in the UIMS system are presented. UIMS™ (User Interface System) was designed and developed with Classic-Ada [1]. UIMS is a system providing basic user interface classes. UIMS was developed in 1988 and released in 1989. UIMS has since had two additional releases. The three releases of UIMS are referred to as v1.0, v1.1, and v1.2.

The motivation of this empirical study is to remedy the lack of knowledge of maintenance activities in object oriented systems. The inheritance hierarchy of classes is a unique feature of the object oriented paradigm. In general, there is very little knowledge of maintenance activities at each inheritance level. The distribution of the three types of maintenance activities in object oriented systems is not well known. This paper attempts to address those concerns.

The observations made in this paper of the maintenance activities in an object oriented system may help researchers and practitioners better understand how and what maintenance activities are performed in object oriented systems -- for example, what types of maintenance activities are likely to happen at what release, and if a particular type of maintenance occurs, where it is likely to be in the inheritance hierarchy.

Some interesting questions concerning the maintenance in object oriented systems include 1) which groups of classes are likely to be affected by maintenance? 2) which inheritance levels have the most maintenance activities? and 3) what are the differences of maintenance patterns in various releases?

---

[1] SPS, Classic-Ada, and UIMS are trademarks of Software Productivity Solutions, Inc.

The observations in this section address the following concerns: 1) what proportion of classes are affected by the perfective maintenance at each inheritance level, 2) what proportion of classes are affected by the corrective maintenance at each inheritance level, 3) what proportion of classes are affected by the adaptive maintenance at each inheritance level, 4) what extend do maintenance activities differ in various releases, and 5) what is the distribution of the three types of maintenance over inheritance level in the object oriented systems.

The Perfective, Corrective, and Adaptive Maintenance at Each Inheritance Level

Figures 1 and 2 display the proportion of the classes affected by perfective maintenance at each inheritance level in the UIMS v1.0, v1.1. The horizontal axis in the figures represents the inheritance level. The vertical axis represents the proportion of classes affected by the perfective maintenance. The figures reveal the relation between the inheritance level and the perfective maintenance. The inheritance level starts from zero. The vertical axis value is calculated as follows:

> proportion at each level = (number of classes affected by perfective maintenance at that level) / (number of classes at that level)

Figure 1 shows a roughly even proportion at each level. This relative evenness indicates that classes at all inheritance levels are effected approximately evenly by perfective maintenance. This trend may reflect the immaturity of the classes at all levels in the original release.

Figure 2 indicates that a higher proportion of classes are affected by perfective maintenance at higher inheritance levels. This observation leads to the speculation that as time progresses, more and more perfective maintenance is performed in the classes at higher

inheritance levels. The hypothesized reason for this to happen is that a perfective maintenance may affect several lower level classes or one or two higher level classes. Maintaining fewer higher level classes may take less effort.

Figures 1 and 2 show that a higher proportion of classes are affected by perfective maintenance at higher inheritance levels. The trend may cause potential problems as the life of software progresses such that the higher level classes may become more and more complex due to overloaded maintenance after each subsequent release.

Figures 3 and 4 show the proportion of the classes affected by corrective maintenance at each inheritance level in the UIMS v1.0 and v1.1. The horizontal axis in the figures represents inheritance level. The vertical axis represents the proportion of classes affected by the corrective maintenance. The figures indicate the relation between the inheritance level and the corrective maintenance in UIMS. Each vertical axis value is calculated as follows:

> proportion at each level = (number of classes at that level affected by corrective maintenance) / (number of classes at that level)

Figure 3 suggests a tendency for the maintenance to be distributed over all inheritance levels among the classes in the UIMS original release. This may indicate once again the immaturity of classes at all inheritance levels in the early software release.

Figure 4 shows that as time progresses, corrective maintenance concentrates on the classes at higher inheritance levels in UIMS. This may reflect the fact that bugs were introduced due to earlier perfective and corrective maintenance among classes at the higher inheritance level, since a higher proportion of classes at these levels were affected by corrective maintenance in the first release.

Figures 3 and 4 demonstrate that a higher proportion of classes are affected by corrective maintenance at higher inheritance levels.

Figures 5 and 6 exhibit the proportion of the classes affected by adaptive maintenance at each inheritance level in the UIMS v1.0 and v1.1. The horizontal axis in the figures represents the inheritance level. The vertical axis represents the proportion of classes affected by the adaptive maintenance. The figures display the relation between the inheritance level and the adaptive maintenance in UIMS. Each vertical axis value is calculated as follows:

> proportion at each level = (number of classes at that level affected by adaptive maintenance) / (number of classes at that level)

The overall trends shown in Figures 5 and 6 indicate that adaptive maintenance concentrates on the classes at higher inheritance levels in UIMS. These trends lead to the speculation that adaptive maintenance is pushed to the classes at higher inheritance levels intentionally. The hypothesized reason for this occurrence is that higher level classes are more environment-sensitive than lower level classes. The figures also show that adaptive maintenance is less frequently performed than both the perfective and the corrective maintenance.

Figures 5 and 6 demonstrate that a higher proportion of classes are affected by adaptive maintenance at higher inheritance levels.

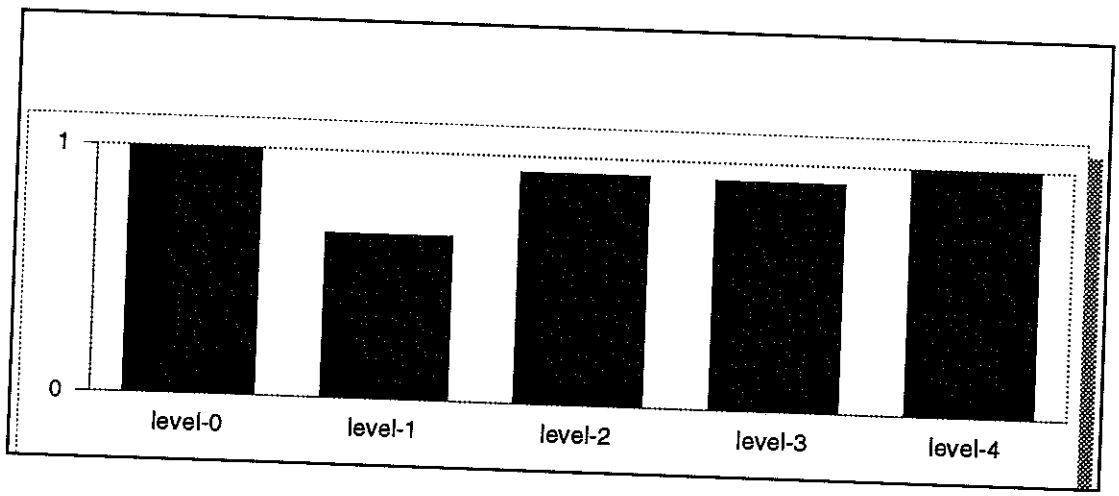

Figure 1 : Proportion of Classes Affected by Perfective Maintenance in UIMS v1.0

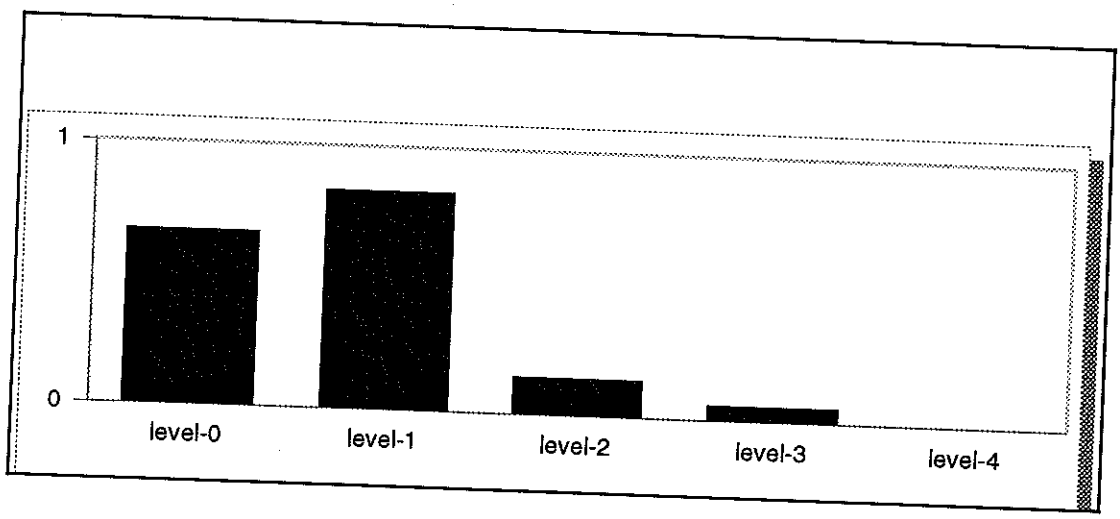

Figure 2 : Proportion of Classes Affected by Perfective Maintenance in UIMS v1.1

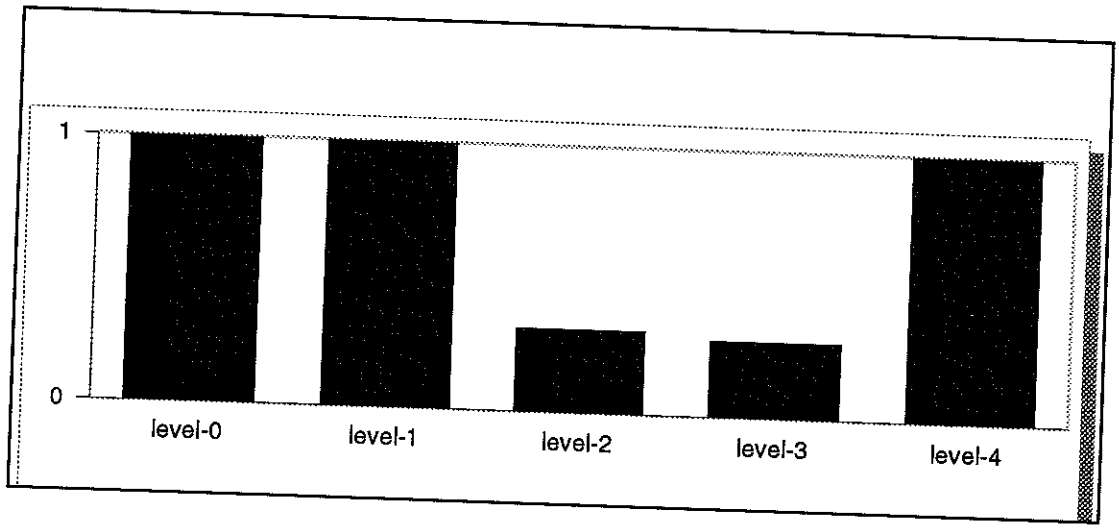

Figure 3 : Proportion of Classes Affected by Corrective Maintenance in UIMS v1.0

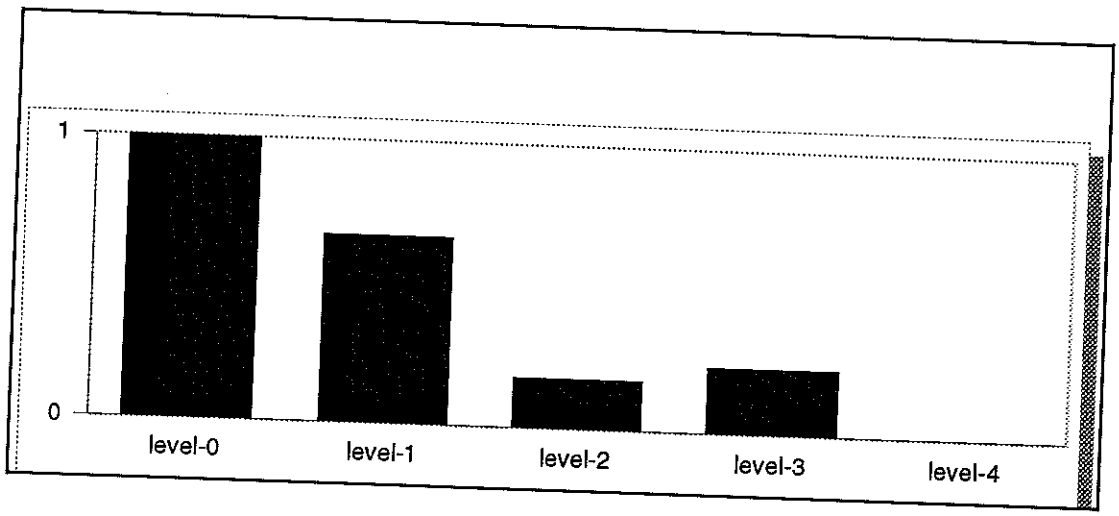

Figure 4 : Proportion of Classes Affected by Corrective Maintenance in UIMS v1.1

The Distribution of Perfective, Corrective, and Adaptive Maintenance

Figures 7 and 8 show the proportion of classes affected by the three types of maintenance in the UIMS v1.0 and v1.1 respectively. The horizontal axis represents the types of maintenance. The vertical axis represents the proportion of each type of maintenance in the system. Each vertical axis value is calculated as follows:

> proportion of classes affected by a particular type of maintenance = (number of classes affected by that type of maintenance in a system) / (number of classes affected by maintenance in the system)

Figure 7 reveals the distribution of the three types of maintenance among the population of the maintained classes in the UIMS original release v1.0. The perfective maintenance dominates the maintenance activities in the first release. This distribution may reflect the immaturity of most classes in the first release.

Figure 8 demonstrates the distribution of the three types of maintenance activities among the population of the maintained classes in the UIMS second release. The corrective maintenance dominates the maintenance activities in the second release. This distribution may indicate the ripple effects of the previous maintenance activities.

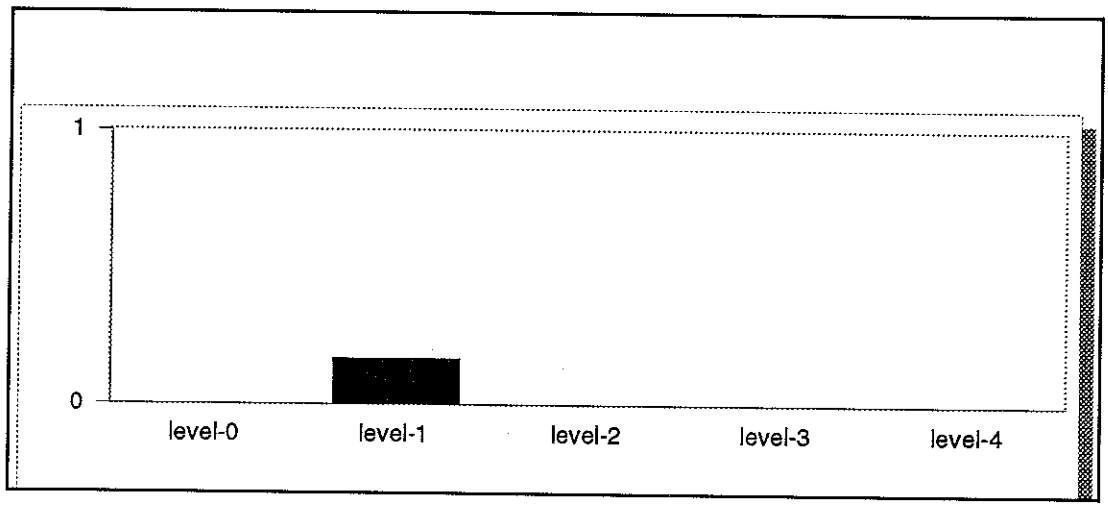

Figure 5 : Proportion of Classes Affected by Adaptive Maintenance in UIMS v1.0

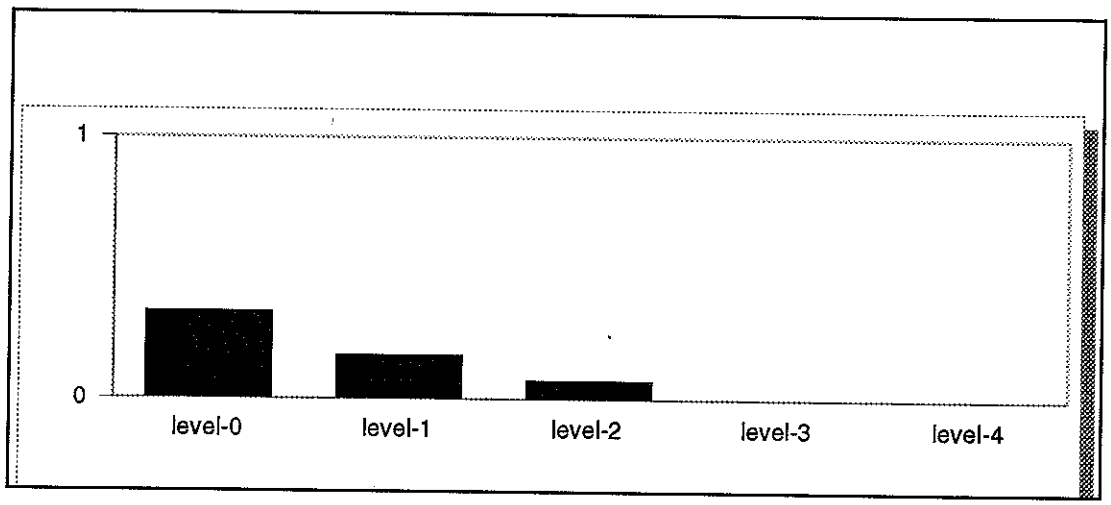

Figure 6 : Proportion of Classes Affected by Adaptive Maintenance in UIMS v1.1

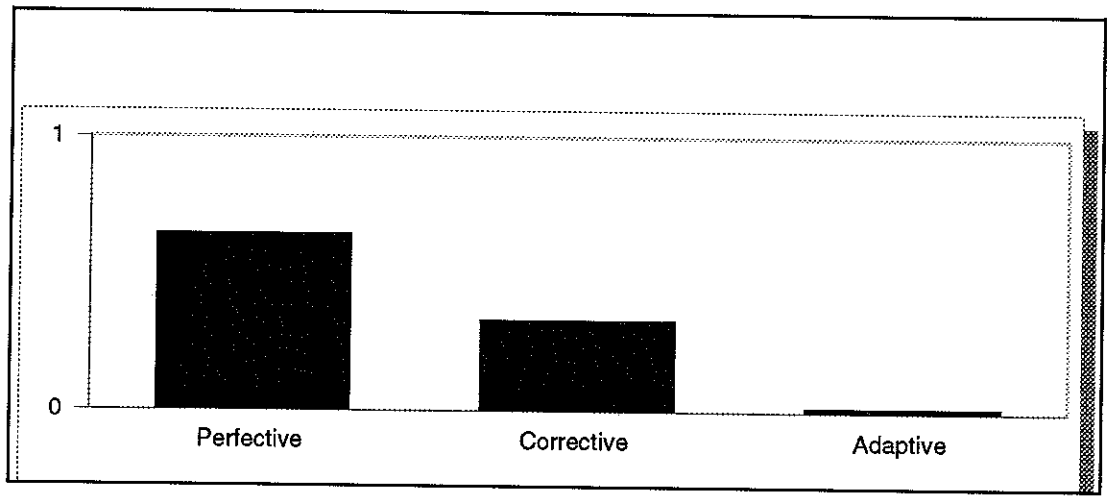

Figure 7 : Proportion of Different Types of Maintenance in UIMS v1.0

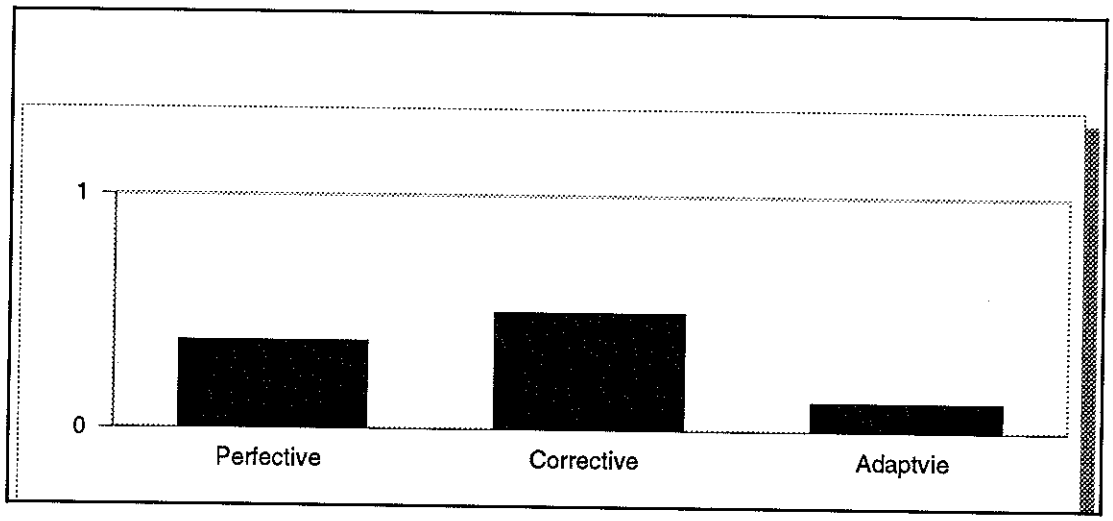

Figure 8: Proportion of Different Types of Maintenance in UIMS v1.1

Summary

Several empirical observations were presented. The observations show that 1) a higher proportion of classes in higher inheritance levels were affected by perfective, corrective, and adaptive maintenance activities; 2) there is a significant difference in terms of maintenance patterns in the original release and the second release in UIMS; 3) the perfective and corrective maintenance are more active than the adaptive maintenance; and 4) perfective maintenance tends to dominate the maintenance activities in the original release, while the corrective maintenance tends to dominate the maintenance activities in the subsequent releases.

The maintenance patterns presented in this section shows that there are observable patterns of maintenance activities in the object oriented systems.

Bibliography

[BENS87] Bendifallah, Salah and Walt Scacchi, "Understanding Software Maintenance Work," *IEEE Transaction on Software Engineering*, Vol. SE-13, No. 3, March 1987, pp.311-323.

[COXB86] Cox, Brad J. "Object Oriented Programming," *Addison-Wesley*, Massachusetts, 1986.

[KAFD85] Kafura, Dennis and James Canning, "A Validation of Software Metrics Using Many Metrics and Two Resources," *Proceedings: 8th International Conference on Software Engineering*, August 1985, pp. 378-385.

[KORT90] Korson, Tim and John D. McGregor, "Understanding Object-Oriented: A unifying Paradigm," *Communication the ACM*, September 1990, Vol.33, No.9, pp.41-60.

[MUNJ81] Munson, John B. "Software Maintainability: A Practical Concern for Life-Cycle Costs," *IEEE Computer*, November 1981, pp.103-109.

[SCHN87] Schneidewind, Norman F., "The State of Software Maintenance," *IEEE Transactions on Software Engineering*, Vol. SE-13, No.3, March 1987, pp.303-310.

[SWAB76] Swanson, E. Burton, "The Dimensions of Maintenance," *Proceedings of the Second International Conference on Software Engineering*, May, 1976, pp.492-497.

[WEGP87] Wegner, Peter, "Dimensions of Object-Based Language Design," *Proceedings: OOPLSA'87*, October, 1987, pp.168-182.