A Tool to Support Knowledge Based Software Maintenance: The Software Service Bay

Jonathan I. Maletic

Robert G. Reynolds

Computer Science Department
Wayne State University
431 State Hall
Detroit, MI 48202

Abstract: *A software maintenance methodology, The Software Service Bay, is introduced. This methodology is analogous to the automotive service bay which employs a number of experts for particular maintenance problems. Problems in maintenance are reformulated so they may be solved with current AI tools and technologies.*

1. Introduction.

Software Maintenance costs are undeniably the major cost in the life time of any evolving software system. The cost of maintenance is dependent on the application domain, system age, hardware stability, and development environment [10,14]. Normally maintenance costs are between two and four times the costs of development, but can be as high as 130 times development costs [4]. Much of the work done to reduce the cost of maintenance is to invest in the production of software development methodologies used to construct large software systems. The key assumption here is if a system is constructed with maintenance in mind, then actual maintenance will be easier and therefore less costly. Even though this work is improving the quality of software and reducing the cost of maintenance, there still exists difficult and costly problems related to maintenance [10]. Little work has focused on the actual maintenance phase and how to make it more efficient to do maintenance on, even, a well constructed software system [6]. Much of the difficulty in doing maintenance on a software system is that the maintenance is being done in an environment intended for the development, and not for the maintenance of software.

1.1. An Environment for Software Maintenance.

What is Maintenance? Maintenance is a term used to refer to changes made to a software system after the system has been delivered to the user [7]. Maintenance is performed for a variety of reasons including: correcting errors, design improvements, platform changes, interface changes, database changes, system enhancements, etc. These involve exceedingly different types of problem solving activities. For example, correcting errors (bug fixes) and system enhancements certainly cannot be categorized as the same type of problem solving activity. Even so, both of these activities are viewed as Software Maintenance. In general, software maintenance problem solving is commonly defined as one of four distinct types [10]:

- Corrective
- Perfective
- Adaptive
- Preventive/ Preservation

Corrective maintenance is the correction of error or bugs found in the software system. Perfective Maintenance are those changes to the system designed to improve its current performance. Adaptive maintenance relates to program modifications done in response to changes in the runtime environment of the software system. The terms *perfective*, *adaptive*, and *corrective* were originally used by Swanson [15]. *Preventive* maintenance is done to improve the future maintainability of the software system [10]. This type of maintenance is also refereed to as *preservation* of the software system [5], where preservation involves using development resources to improve an aging system that often has a poor design and is therefore hard to maintain.

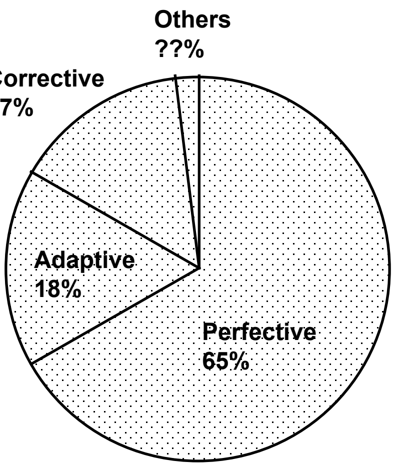

Figure 1.1. Maintenance Types and Distribution.

Of the four types of maintenance, perfective maintenance is cited as the most frequently occurring (see figure 1.1) type of maintenance [10,8,14,12] and preventive maintenance is rarely done. However, it is assumed that a good software development methodology implicitly deals with this issue. Corrective and adaptive maintenance are done much less frequently than perfective maintenance. This is a particular problem since the maintenance activity is taking place in a software development environment. Therefore corrective maintenance can be done easily in a development environment due to the fact that debugging is a typical part of the development process. Generally many tools exist within a development environment to find and fix bugs. Tools intended to assist in the other types of maintenance are rarely (if ever) included in a development environment.

Each of the four types of maintenance problem solving require different goals, methods, and expertise employed to solve problems within their respective domains. Software Development environments do not reflect these differences in their support of the maintenance process. An environment focused on doing overall maintenance problem solving is needed to help in reducing the costs and problems encountered during maintenance of large software systems.

Before an environment focused on maintenance is described, a methodology must be defined that governs the realization of such an environment. To give perspective to such a methodology, a survey of current maintenance methodologies is presented in the following section.

1.2. Maintenance Methodology Trends.

Much of the literature pertaining to maintenance deals with the design of maintainable software, the cost of maintenance, and the organizational management of the maintenance task [10,8,7,14,12]. The concept of a comprehensive maintenance problem solving methodology is noticeably absent in the literature dealing with maintenance and software engineering even though the need is apparent [9,13]. The general principle used to deal with the huge maintenance problem is to build more maintainable software. Maintainable software supposedly has qualities that allow it to be modified, understood, and adapted more easily. That is, a software system built with the a priori knowledge that it will be enhanced, adapted, or debugged will be easier to maintain. This view is based in common sense, and this approach has made great strides in reducing the cost of the maintenance of such systems [3].

The current informal frameworks and methodologies for maintenance are described as modifications of widely used software development methodologies [2] (i.e., waterfall model, spiral model, etc.) or guidelines for the maintenance process given varying amounts of system documentation [10]. These frameworks and guidelines still view maintenance as a one part of the development process and not as a separate methodology.

The following section introduces a formal methodology that focuses on maintenance as the central principle and not just as one of the development phases of the software life cycle. The methodology proposed in this paper is called the *Software Service Bay*. It is motivated by an analogy with the automotive service bay in which there is a different set of tools, problem solving knowledge, and problem solver, for each type of maintenance problem concerning a vehicle.

1.3. The Software Service Bay Methodology.

The Software Service Bay Methodology is akin to the concept of the automotive service bay at one's corner automotive service station. A car is taken in for service because of one or another reason, the engine is running poorly, a drive train problem, a brake problem, or routine/preventive maintenance. In the automotive service bay, with each of these maintenance problems goes a special set of tools and a specialized mechanic trained in the particular maintenance problem. The Software Service Bay Methodology works in much the same way. If a software component no longer meets the needs of the user (e.g., specification change, platform change, uncovered bug) then the component is sent in for "service". But, instead of the typical maintenance methodology, which is generally little more then a modified software development methodology, the Service Bay Methodology supports a host of special tools and experts in each of the maintenance types. Doing software maintenance within a software development methodology is analogous to doing automotive service on the shoulder of the highway instead of at the service station. The place where the vehicle broke down generally is not conducive to the support of any maintenance problem solving activities and in general is a harmful environment to do such activities.

The maintenance problem solving process within the Software Service Bay Methodology involves the following phases:

1) Pose specific maintenance problem(s).
2) Determine service rational or general maintenance type.
3) Acquire specific domain knowledge.
4) Develop maintenance strategies and plans.
5) Implement strategies and plans.
6) Validate solution.

These phases are performed in the order given but may cycle back to a previous phase if necessary, such as when the solution is invalidated by the final phase. Each of these phases are now be described in more detail.

1.3.1. Pose Specific Maintenance Problem(s).

This phase encompasses defining the specific problems needed to be solved within the specific maintenance type. A description of the error, enhancement, platform change, or adaptation is needed for this phase to determine what type of maintenance problem solving is taking place. For example, the particular maintenance problem can possibly be transformed so that it can be solved by some current technology or a special purpose problem solving activity may have to be developed to tackle the problem. The concerned in this phase is with the input and output of the software system rather than the internal design and structure of the system. The intent here is to do an analysis of the maintenance problem without dealing with the low level details of the system. The next phase, assessing the current state of the system deals with these design and structure issues relative to operationalizing a solution for the problem.

1.3.2. Determine Service Rational or General Maintenance Type.

The first phase of the Service Bay Methodology involves determining the problem solving rational behind the service. Just as there is a definite reason why one takes a car in for service, e.g., it stalls at every stop light or the car's ride is very rough, there must be a rational behind the need for maintenance. This rational can be defined explicitly or implicitly in terms of the four types of maintenance: corrective, perfective, adaptive, and preventive. A maintenance type provides the focal point for the problem solving process to follow. Each phase is customizable, using domain knowledge, to reflect the type of maintenance to be done there. The service rational is expressed relative to a maintenance schedule or set of constraints that is established with the generation of the software product. Within that schedule certain generic maintenance tasks are specified and provide temporal landmarks for the realization of other tasks. Identification of new problems can spawn auxiliary schedules for future use.

1.3.3. Acquire Specific Domain Knowledge.

In assessing the current state of the system and its environment, an understanding of the software system's design, structure, and data flow must be acquired relative to the current problem. The depth of knowledge required is dependent upon the rational for the service and the problem at hand. For example, localizing a bug or determining where program changes for an enhancement will occur connote very different levels of knowledge about the software system. The corrective maintenance problem may require general knowledge of the entire system structure and only a very detailed understanding of maybe one or two routines. However, in a perfective or adaptive maintenance problem detailed knowledge of the entire system is often needed because this type of maintenance often requires many code changes throughout the system.

1.3.4. Develop Maintenance Strategies and Plans.

This phase of the maintenance methodology concerns the selection of a set of general problem solving strategies and plans that address the problems elucidated previously in terms of the current knowledge of the system acquired previously. The maintenance strategies developed here will then be operationalized later on in order to implement specific solutions to the particular maintenance problems at hand. For instance, problem solving strategies are used to suggest possible side effects resulting from the change and specify ways to deal with them. In the case of preventive maintenance, the actual problem to be solved is dictated by the strategies developed. That is, the strategies developed suggest ways to make a system more maintainable or point out areas in the system that are poorly constructed. Plans developed in this phase direct what fixes and modifications must be done to certain portions of the software system and the nature of the systems future maintenance schedule. These plans include "canned" fixes or routines known to fit the change specification. This type of planning knowledge and information are retrieved from a software reuse library.

1.3.5. Operationalize Strategies and Plans.

Once the maintenance problem solving strategies and corresponding plan are selected they are implemented, and the changes and modification dictated by them are made to the software system. The plans may involve code changes, code rewrites, or the creation of new code however, they are not limited to modification of code. Changes also occur to the corresponding documentation in order to reflect the modification of the code. This step in the methodology is analogous to the auto mechanic doing the actual work on the vehicle (i.e., get their hands greasy). All of the canned fixes suggested by the previous phase is implemented in this phase. Therefore, a reuse system that allows easy access and indexing of such solutions is needed.

1.3.6. Validate Solution.

The last phase of this methodology involves the validation of the solution. Validation of these changes and modification are made with the knowledge gleaned from the previous phases and any testing history of the system and associated documentation being maintained. The testing of a modified system only entails testing part of the system versus the entire system. A modification may only effect a subset of the entire system. Knowledge of the modifications and where they occurred in the system are useful information in this phase. This type of knowledge is useful to the vehicle mechanic in the same way. For instance, if a new alternator (generator) is installed into a vehicle, the mechanic would check to see if the battery is being charged correctly. And the mechanic will not test the transmission in this case because the alternator has nothing to do with the transmission.

1.4. Service Bay Agents.

Each of the problem solving phases within the Service Bay Methodology is supported by one or more specialists and agents. These specialists and agents are experts in one particular area of the maintenance problem. The Service Bay specialists and agents are functionally independent but work in conjunction with one another in an opportunistic fashion. This topic will be discussed in more detail in section 3.

2. Realization of a Maintenance Methodology.

In section one a methodology for maintenance problem solving, The Software Service Bay, is presented that performs in all types of maintenance situations. This method for doing maintenance is different from current methods for maintenance in that the Service Bay is not embedded in a development methodology whereas most current methods are. In order to implement the Service Bay approach, a number of tools and technologies must be integrated together to support some of the diverse problem solving done in maintenance.

Many maintenance situations can be reformulated into problems that are solved by other existing tools and technologies. Such tools and technologies include: software reuse, re-engineering of software, and KBSE technologies. These techniques must be examined to determine how they may be applied to specific types of maintenance problems.

The next step is to operationalize the Software Service Bay Methodology in terms of problem transformation and reformulation. The general problem solving mechanism is augmented by a special purpose problem solver. Each special purpose problem solver has a set of preconditions that must be satisfied, by a particular problem, for it to be activated. In this manner traditional problem solving approaches are integrated into the maintenance problem solving process.

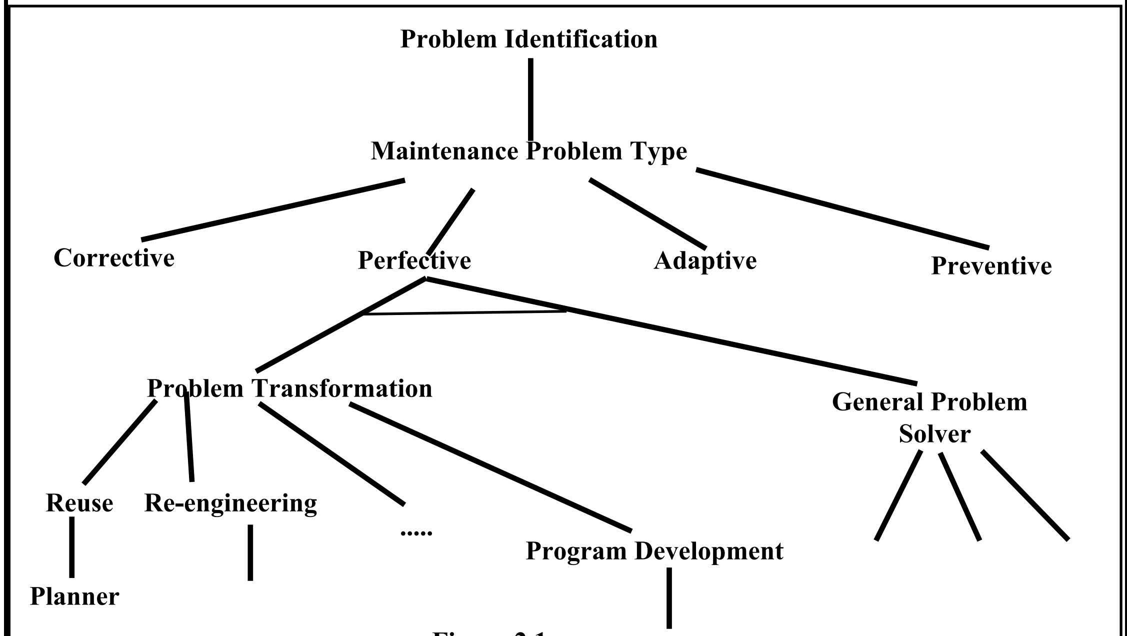

Figure 2.1.

And/or description of the Software Service Bay.

2.1. A Knowledge Based Maintenance Assistant.

The maintenance problem solving process within the Software Service Bay Methodology involves the following phases:

1) Pose the specific maintenance problem(s).
2) Determine the maintenance type for the problem. Within that maintenance type can the problem be transformed in terms of an alternative methodology. Identify the problem solver (special purpose or general purpose) that is appropriate to the original or transformed problem
3) Acquire specific domain knowledge needed to solve that type of problem.
4) Develop maintenance problem solving strategies and plans.
5) Implement problem solving strategies and plans.
6) Validate solution.

These phases are now operationalized (see figure 2.1) with reference to transforming particular maintenance problems into problems can be solved opportunistically in terms of other tools and technologies associated with problem solving situations.

The overall problem solving procedure is described in a task oriented fashion using and/or tree above. The first task in the operationalization scheme (figure 2.1) is Problem Identification. This relates to the first phase of the Service Bay methodology namely, posing the specific maintenance problem. The next task is to determine the general maintenance type. Next, the appropriate problem solver is selected. Each subproblem is then either reformulated so that it can be solved by some special purpose problem solver, or the subproblem is handed over to a general problem solver for that maintenance type. For example, if the subproblem can be transformed into a problem in software reuse, an associated planner is activated to construct a plan or strategy to solve this particular subproblem. If the problem is initially given to the general problem solver for a given maintenance type it can eventually solve enough of a subproblem so that parts or all of its subproblem can be handed over for problem transformation. Each time a problem transformation is applied, a reduction in the search space of possible solutions for the entire problem is made. The transformations are applied in an opportunistic way so that each time the conditions are satisfied for a special purpose problem solver to be used, it will be invoked upon that particular subproblem.

The special purpose problem solvers are represented in the Software Service Bay as *specialists* and generic tools are represented as *agents*. These specialists and agents are experts or tools in one particular area of the maintenance problem solving process and are functionally independent but work in conjunction with one another in an opportunistic fashion. For example, the problem reformulation specialist may determine that a particular subproblem meets the condition to be solved by the reuse specialist, the reuse specialist would then attempt to solve that problem. In another type of interaction between agents: the Static Structure agent, which computes and displays the static structure of a given software component, produces information used by the Program Understanding agent, which needs the static structure information to construct a high level description of the component.

The following is a suggested list of possible (Semi-) Automated Service Bay agents and specialists:

- Reuse specialist- Classifies, retrievals, and storage of software components.
- Re-engineering specialist- Applies re-engineering problem solving to maintenance problems
- Reformulation agent- Transforms problem for solution by specialists.
- Maintenance Strategy/Plan agent- Develops plans and strategies.
- Modification agent- Planner, organizer of modification to component.
- Static and Dynamic Structure agent- Compute and display static/dynamic structure.
- Description agent- Fact sheet, and viewing information about components.
- Understanding agent- At different levels.
- Testing agent- Testing of a component, history of testing.

The reuse specialist is a particularly important part of the Software Service Bay Methodology. By incorporating a reuse library and a reuse based problem solving paradigm into the maintenance problem solving process, a programmer is able to solve certain maintenance problem by reusing existing software. Going back to the analogy of the automotive service bay, the reuse library is much like the parts department at a automotive service bay. An inventory of new (and often times used) parts for a variety of vehicles are kept on hand so the mechanics may fix the malfunctions found. The importance of the reuse activity to the maintenance process is discussed in the next section.

2.3. Conclusion and Future Work.

In order to operationalize the Software Service Bay Maintenance Methodology the following issues must be addressed:

- What information is needed in general to describe the maintenance problem? How can a special case of the maintenance problem be identified? The information needed to describe the maintenance problem has been partially addressed in terms of a task oriented description relating the methodology to current tools and technologies for problem solving. There are several issues yet to be directly addressed for each task in this framework. The issues include the following: what type of information is needed for the problem identification task?, and how is the specific maintenance type to be determined?

- Can the Service Bay methodology be expressed in terms of the blackboard model of problem solving? To implement this methodology with a black board architecture a detailed system analysis is needed. What basic types of specialists, such as reuse based problem solver, are needed to implement a useful system? What types of general problem solving paradigms are to be used? How do the specialists interact with the software system (blackboard) and with the other agents of the system needed?

- How can existing problem solving technologies such as reuse based approaches be embedded in this framework as special purpose problem solvers. In a prototype of the system, as proposed, the issues of integrating traditional problem solving approaches (i.e., software reuse) into the maintenance activity will be examined. How these problem solving approaches enhance the performance of implementing maintenance tasks will also be studied. In doing so, the maintenance problem solving process in general can be better understood.

A prototype of a knowledge based software maintenance problem solver is now under construction at Wayne State University. This prototype employs the Software Service Bay methodology and incorporates into the framework, a reuse oriented maintenance specialist based upon the PM System architecture [11].

3. Bibliography.

1. Barr, A., Feigenbaum, E.A., (1982) *The Handbook of Artificial Intelligence Volume II*, William Kaufmann, Inc.
2. Basili, Victor R., (1990) "Viewing maintenance as Reuse-Oriented Software Development", *IEEE Software*, January, pp. 19-25.
3. Boehm, B., (1977), "Seven Basic Principles of Software Engineering", in *Infotech State of the Art Reports: Software Engineering Techniques* (Maidenhead, England: Infotech International), pp. 77-113.
4. Boehm, B.W., (1975), "The High Cost of Software", in *Practical Strategies for Developing Large Software Systems,* Horowitz, E, (editor), Addison-Wesley Pub. Co.
5. Booch, G. (1994), *Object-Oriented Analysis and Design with Applications 2nd Ed.,* Benjamin/ Cummings Pub. Co. Inc. california.
6. Lamb, David A., (1988), *Software Engineering Planning for a Change*, Prentice Hall.
7. Martin, James, McClure, Carma, (1983), S*oftware Maintenance, The Problem and its Solutions,* Prentice-Hall Inc.
8. Mills, Harlan D., (1993), "Zero Defect Software: Cleanroom Engineering", Advances in Computers, Vol. 36, pp. 1-41.
9. Osborne, Wilma M., Chikofsky, Elliot J., (1990), "Fitting Pieces to the Maintenance Puzzle", *IEEE Software*, January, pp. 11-12.
10. Pressman, Roger S., (1992), *Software Engineering, A Practitioner's Approach 3rd Ed.*, McGraw-Hill Book Co.
11. Reynolds, R.G., Maletic, J.I., Porvin S.E., (1992), "Stepwise Refinement and Problem Solving", IEEE Software September, pp. 79-88.
12. Schach, Stephen R., (1993), *Software Engineering Second Edition*, Aksen Associates Incorporated Publishers.
13. Schneidewind, Norman F., (1987), "The State of Software Maintenance", IEEE Transactions on Software Engineering, SE-13, No. 3, March, pp. 303-310.
14. Sommerville, Ian, (1989), *Software Engineering 3rd Ed.*, Addison-Wesley Publishing Co.
15. Swanson, E.B., (1976), "The Dimensions of Maintenance", *Proc. 2nd Intl. Conf. Software Engineering*, IEEE, pp. 492-497.