Software Maintenance Types—A Fresh View

Ned Chapin
Information Systems Consultant
InfoSci Inc., Box 7117
Menlo Park CA 94026–7117, USA
NedChapin@acm.org

Abstract

Compatible with the recently proposed ontology of software maintenance, the paper proposes a fresh view of the types of software maintenance. The paper offers a classification based not on people's intentions, but upon objective evidence of activities based on documentation that may include the source code. The classification includes taking into account evidence of: 1) changes in the character and use of the documentation, 2) changes in the properties of the software, and 3) changes in the in the functionality of the software. The paper provides a hierarchic summary guide to the proposed fresh view of the types of software maintenance.

Keywords: software maintenance; enhancive maintenance; adaptive maintenance; corrective maintenance; performance maintenance; evaluative maintenance; empirical studies; maintenance terminology

Introduction

The significant new work proposing an ontology of software maintenance [1], done by a multinational mostly European team, has stimulated a next step from the other side of the Atlantic—proposing a fresh view of the types of software maintenance.

The motivations include those of the ontology team, namely:

- providing a context for the study by researchers of specific questions about software maintenance,
- helping in the interpretation of the reports of empirical studies of maintenance,
- offering a common ground for a cogent reporting of empirical studies of maintenance, and
- supplying a framework for organizing and categorizing the reports of empirical studies of maintenance.

The additional motivations are:

- providing a realistic and practical terminology to facilitate communication about and the management of software maintenance among researchers, practitioners, and their managers,
- basing the terminology on objective evidence ascertainable even in the absence of the personnel who may have been involved with the maintenance,
- recognizing that software maintenance efforts typically encompass a variety of tasks and activities that both practitioners and their managers, and researchers, want to capture in order to record them as meaningful data, and
- focusing on the more commonly encountered software maintenance situations and circumstances.

Practitioner concerns were the focus of early published work. Thus, in 1972, Canning [2] summarized these in his well known piece, "That Maintenance 'Iceberg'." Canning reported that practitioners saw maintenance narrowly as correcting errors, and broadly as expanding and extending software functionality. Some practitioners also included accommodating to changes in the underlying system software or hardware. For the benefit of practitioners and researchers alike, E. Burton Swanson in 1976 offered a typology to rationalize the practitioners' varied recognition of types of software maintenance [3]. His typology was based on the system's owners'/users' dominant objective or intention in requesting or undertaking the maintenance work—i.e., what was seen as the cause or purpose of the maintenance, or why was the maintenance to be done [1,4]. On a mutually exclusive and exhaustive basis [5], Swanson pointed to three intentions for software maintenance:

- to perfect the system in terms of its performance, processing efficiency, or maintainability ('perfective maintenance'),

- to adapt the system to changes in its data environment or processing environment ('adaptive maintenance'), and
- to correct processing, performance, or implementation failures of the system ('corrective maintenance').

These intentions, like beauty, exist primary in the eye of the beholder, and unless specifically recorded by the personnel involved at the time of the event, cannot be reliably and consistently determined after the fact from available objective evidence. Intentions reflect the organizational and political context in which the system operates as well as the character of the maintenance work. The requester may consider something as "corrective" but the person with approval authority may regard the requested change as "perfective". If approved and done, the Information Technology manager may treat the change as part of an "adaptive" maintenance project. However, an intentions basis for a maintenance typology is appropriate for and can be used in some kinds of valuable research, as has been well demonstrated, for instance, by Lientz and Swanson [5].

Current state of the field

Among researchers, Swanson's typology has been influential [6]. Many researchers have adopted the terminology ('corrective', 'adaptive', 'perfective'); however, few researchers have used the typology with Swanson's definitions. Instead, they have redefined the terms, and not agreed among themselves on the meanings. While this diversity has made for variety in the reported research, the lack of agreement on the meaning of commonly used terms has added to the chaos in the literature making it difficult for researchers to build upon each other's work. The IEEE put out a glossary [7] that includes the terms 'corrective maintenance', 'adaptive maintenance', and 'perfective maintenance'. Interestingly, the IEEE *Glossary* definitions are partially inconsistent with Swanson's definitions [3,7], and with the definitions in the IEEE *Standard for Software Maintenance* [3,7,8] . Most researchers have made modifications, especially to the definitions. Some have done so explicitly, such as [9,10,11,12] but most implicitly or tacitly such as [13,14,15,16], even when retaining the use of the terms 'corrective', 'adaptive', and 'perfective'. Non-researcher authored materials have followed a similar pattern, such as [17,18].

Such extensively observable variations in definitions and terminology suggest that the time is ripe for a fresh classification of the types of software maintenance to meet the needs of researchers, and of practitioners and their managers. The next section summarizes this paper's objectives in proposing a fresh classification of software maintenance types. The third section presents the proposal, and the closing section provides some discussion.

Objectives

The objectives in proposing a fresh classification for types of software maintenance have been to make the classification be:

- based on objective evidence of activities ascertainable from documentation including the source code, rather than be based on intentions,
- independent of hardware platform, operating system choice, design methodology, implementation language, organizational practices, and the availability of the personnel who were or are involved in the maintenance or its management,
- applicable to a large proportion of the situations encountered by researchers and practitioners and their managers, but not try to fit all of the situations of every researcher, practitioner or manager,
- practical to use for a large proportion of researchers and of practitioners and their managers, but not try to meet all of the needs of every researcher, practitioner or manager,
- an evolutionary refinement and/or redefinition of existing terminology and practice, and
- an extention and enrichment the contributions from the proposed ontology of software maintenance [1].

Proposed classification

Criteria decisions

Changes made, observed or detected on three aspects of the software maintained or the being maintained, serve as the criteria for groups or clusters of types:

- the software as a whole,
- the source code, and
- the customer-experienced functionality.

The classification is based on changes in these entities as a result of the activities of doing software maintenance. Evidence is obtained by comparing the relevant parts of the software as of *before* the alleged maintenance, with the relevant parts of the software as of *after* the alleged maintenance. None of the evidence requires or depends upon personnel's statements or recollections of intentions about, or the causes of, or the reasons for, or the purposes of, the software maintenance. For personnel using intention-based classifications, the objective evidence can sometimes be used to find out or infer causes or purposes or intentions or reasons or stated requirements for work done.

As pointed out in the ontology [1], software maintenance

commonly involves several to many processes or activities that may result in from none to many changes in the software. Within each grouping of processes or activities noted from applying the criteria, detailed types of software maintenance can be distinguished by applying type decisions. From the set of all changes made, observed, or detected as attempted from the evidence, a dominant process or activity, or a grouping of processes or activities, typically emerges, with supporting or associated processes or activities usually also present. Relevant management and quality assurance activities are considered to be included within each type. The classification proposed here is defined to be exhaustive with mutually exclusive types, where any instance of maintenance work may involve a mix or aggregate in any order of some or all of the types, even though one type may be deemed or observed as dominant.

Type decisions

The use of the criteria decisions to identify the grouping or clusters of types, is followed by the use of the type decisions to identify the type within the group or cluster. Table I summarizes this in a hierarchic pattern. Note that the groups or clusters and the types within the groups or clusters increase in their software maintenance significance from top to bottom in Table I.

The first column in Table I lists the three criteria, with their associated criteria decisions shown immediately to the right. The second column indicates the response choices for the associated criteria decisions. The third column lists the decisions, type and criteria, to be made from the evidence, or in order to provide the hierarchic structure, directs the nesting of the applicability of the criteria decisions. The fourth column provides two kinds of data. At the left margin, it lists the group or cluster identifying names. Indented under those names it lists the types within the group or clusters. The types apply only when the response to the associated type decision to the left is "yes".

To use Table I, the starting point is criterion decision # 1: "Did the work change the software?" The choice ("yes" or "no") dictates the type decisions to ask or provides direction to the next criteria decision. Within the choice, each type decision is asked in turn from top within the choice to bottom within the choice. A "yes" response to any type decision leads to the cluster and type shown to the right. Note that typically software maintenance work involves many activities. In some situations, the type for each should be identified. In other situations, only a single overall type is wanted, even when many types have been identified. In such cases, the applicable type that is lowest in Table I is used.

Consider an example of using Table I. Suppose that the available evidence indicates that the software maintenance work in question resulted in changing a small part of a user manual. The assigned programmer-analyst made the changes after studying the existing user manual and the associated other documentation, and conferring with the user. The first criteria decision is "Did the work change the software?" Conferring with the user did not change the software, but gives a "yes" on the type decision "Was the software the basis for consultation?" identifying the *consultive* type. Studying the existing user manual and associated documentation did not, but gives a "yes" on the type decision "Was the software evaluated?" identifying the *evaluative* type. Making changes to the user manual did change the software, and hence directs us to the second criteria decision "Was the source code changed?" Since the response here is "no", the first type decision in the documentation cluster is "Did non-code documentation change make it meet stakeholder needs?" Since the response is "yes", the type is *reformative*. Hence, three types of software maintenance were done, *consultive*, *evaluative*, and *reformative*. Even though most of the time may have been spent on activities of the evaluative type, if a single type is wanted, then *reformative* is the type, since it the lowest in Table I of the relevant types.

In general, most maintenance work examines and tests more source code and other documentation than gets changed. The changes that do get made in source code may change its characteristics or properties, many of which are transparent to the user or customer. The ones that are not transparent are usually instances of the *performance* type, like increasing the response speed. The most important category of changes to the source code that are visible to the user or customer or other stakeholders, are the changes in functionality. Such software changes may causes the system to do more for the customer, or alter what the customer sees or does or can do. These almost always involve implementing changes to the business rules embedded in the software.

Type summary

Each group or cluster of types is distinct and different. The **support interface cluster** concerns changes in how information systems or technology personnel interact with stakeholders and others with respect to the software. In the *training* type, common activities are presenting training to stakeholders about the system implemented by the software. In the *consultive* type, common activities are making time and cost estimates for proposed maintenance work, serving on a help desk, assisting a customer in preparing a maintenance work request, and making specialized knowledge about the software or resources available to

Table I. Summary of proposed evidence-based types of software maintenance

Criteria	Choice	Decisions to be made from the evidence, or directions	Cluster and Type
1	Did the work change the software?		
			Support interface
	No	Was the software the subject of stakeholder training?	Training
		Was the software the basis for consultation?	Consultive
		Was the software evaluated?	Evaluative
	Yes	Ask criterion decision # 2.	
2	Was the source code changed?		
			Documentation
	No	Did non-code documentation change make it meet stakeholder needs?	Reformative
		Did non-code documentation change make it conform to implemented?	Updative
	Yes	Ask criterion decision # 3.	
3	Was the customer-experienced functionality changed?		
			Software properties
	No	Did change in software properties improve elegance or security?	Groomative
		Did change in software properties facilitate maintainability or future use?	Preventive
		Did change in software properties alter performance characteristics?	Performance
		Did change in software properties utilize different technology, resources?	Adaptive
	Yes		Business rules
		Was customer-experienced functionality removed, restricted or reduced?	Reductive
		Was customer-experienced functionality made more correct or fixed?	Corrective
		Was customer-experienced functionality replaced, added to or extended?	Enhancive

others in the organization. In the *evaluative* type, common activities are very diverse and include studying the source code and other documentation, tracing how a proposed change might ripple, preparing and running tests, examining the interactions between operating system features and the software to be maintained, searching for needed data, and debugging.

The **documentation cluster** concerns changes in the documentation other than the source code or its machine language equivalents. In the *reformative* type, common activities are improving the readability of the documentation, modifying the documentation to incorporate the effects of changes in the local standards manual, preparing training materials, and adding entries to a data dictionary. In the *updative* type, common activities are replacing obsolete documentation with accurate current documentation, preparing UML models to document existing source code, and incorporating test plans into the documentation.

The **software properties cluster** concerns changes in the properties and characteristics of the software, but not changes in the functionality of the software. In the

groomative type, common activities are replacing algorithms or components with more elegant ones, changing data naming conventions, doing backups, modifying access authorizations, and recompiling source code. In the *preventive* type, common activities are making modifications to improve maintainability, and putting in place a base for making a future change to some new technology. In the *performance* type, common activities have results that affect the user (but not the functionality) and are such activities as improving system up time, and replacing algorithms or components with faster ones. In the *adaptive* type, common activities are porting the software to a new platform, increasing COTS utilization, and moving to object-oriented technologies.

The **business rules cluster** concerns the functionality experienced by the customer—i.e., how the software and the customer personnel work together to get the customer's work done. In the *reductive* type, common activities are eliminating data from output received by the customer, reducing data flows into or out of the software, and reflecting a customer's narrowed business plan. In the *corrective* type, common activities are fixing detected bugs, adding more defensive programming, and changing the handling of exceptions. In the *enhancive* type, common activities are adding or replacing business rules to extend or expand the system's functionality accessible to the customer, and adding data flows into or out of the software.

Discussion and conclusion

Evidence for many of the twelve types of software maintenance can be observed for most software maintenance work. This is such a common situation that the question is, what type of maintenance is dominant for any specific maintenance task?

On a person-hours spent basis, the *evaluative* type is usually dominant, because it includes comprehending what the software does and how it does it, and preparing and using diagnostic tests. Practitioners and their managers are usually loath to report or advertise *evaluative* as their dominant type of software maintenance because that fails to communicate clearly about the contribution information systems or technology makes to the stakeholders. The more obviously significant work is nearer the bottom of the list in Table I.

Consider an example. A programmer-analyst was assigned to work a change request. The programmer-analyst examined some software seeking the parts that would be relevant, and making an estimate of the time and cost to do the work. After some conferences with the customer personnel and the programmer-analyst's own supervisor, the programmer-analyst got the go ahead. The first task was to build a protected test environment, a regression test suite and associated test plan, and run the tests diagnostically. Then came designing the data layouts, the patches to add the new functionality, and the revised database access. To forestall a warned about possible future complaint, the programmer-analyst replaced an existing procedure with a faster one before attempting to test run with the compiled patches in place. After debugging and having to add a defensive trap, the programmer-analyst prepared the software for regression testing. After passing that testing, and updating both the user manual and the UML, the programmer analyst secured SQA's signoff, and with the supervisor's approval, scheduled and put in some on-site time to help transition the customer personnel to using the changed software when it was put into production.

Question: what type of software maintenance was this? The evidence indicates that nine types were done, but with *preventive*, *adaptive*, and *reductive* apparently not done. Even though comprehending the code and testing appeared to have accounted for the most person-hours, what appeared to stand out most is the increase in functionality gained by the customer. Therefore, within the business rules cluster, we seek the process that yields the best fit with the evidence that is lowest in Table I. In this example, that puts *enhancive* in the position of being the dominant type. Note, however, that both practitioners and their managers, as well as researchers, may find value in knowing about all of the types individually identifiable in order to gain insight and management opportunities.

The choice of names for the types of maintenance has been a matter of concern. Following the lead of Swanson [3], words ending in ...ive have been usually used. Only one freshly coined term, "groomative", has been proposed as an adequately descriptive adjective for grooming the documentation. "Perfective" has been dropped in order to make the proposed classification more precise and useful both to practitioners and their managers, and to researchers. Previous attempts at terms redefinition have not been warmly received and widely used, as for example [7,9,19]. Nonetheless, two terms used by Swanson [3], "adaptive" and "corrective", have been redefined by deleting the intention component from both and by narrowing their applicability to software properties and business rules respectively. These two redefinitions are urged for three reasons:

- the likely acceptance of two newly coined terms for these types of maintenance would likely gain even less acceptance,
- the terms help provide some continuity with popular usage, as noted earlier, and
- the two types, *corrective* and *adaptive*, along with *enhancive*, get the most attention.

The proposed types of software maintenance are

independent of the "scenarios" concerns addressed in the ontology report [1]. The types proposed here bypass not only the scenarios discussed in that report, but also nearly all others. The possible exceptions are total outsourcing of software maintenance, or the combination of total retirement and replacement.

In closing, a look at a major limitation is appropriate. As a detailed analysis can easily show, the use of the proposed criteria decisions in their three levels can be followed by far more than just a dozen type decisions to yield more than a dozen types of software maintenance. Some persons (more likely managers than researchers) may argue that the field does not need to recognize even a dozen types of software maintenance at all. Just two, they say, are and will be sufficient: "too much" and "none".

Acknowledgments

The author thanks the ICSM reviewers for their advice to shorten and condense the paper as submitted. This has been done here.

References

1. Kitchenham BA, Travassos GH, Mayrhauser Av, Niessink F, Schneidewind NF, Singer J, Takada S, Vehvilainen R, Yang H. Toward an ontology of software maintenance. *Journal of Software Maintenance* 1999, **11**(6):365–389.
2. Canning RG. That maintenance 'iceberg'. *EDP Analyzer* 1972, **10**(10):1–14.
3. Swanson EB. The dimensions of maintenance. In *Proceedings 2nd International Conference on Software Engineering*. IEEE Computer Society Press: Long Beach CA, 1976; 492–497.
4. Chapin N. Do we know what preventive maintenance is? In *Proceedings International Conference on Software Maintenance*. IEEE Computer Society Press: Los Alamitos CA, 2000; in press.
5. Lientz BP, Swanson EB. *Software Maintenance Management*. Addison-Wesley Publishing Co.: Reading MA, 1980; 214 pp.
6. Swanson EB, Chapin N. Interview with E. Burton Swanson. *Journal of Software Maintenance* 1995, **7**(5), 303–315.
7. IEEE. *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990*. Institute of Electrical and Electronics Engineers: New York NY, 1991: 83 pp.
8. IEEE. *IEEE Standard for Software Maintenance, IEEE Std 1219–1998*. Institute for Electrical and Electronic Engineers: New York NY, 1998; 47 pp.
9. Kemerer CF, Slaughter SA. Determinants of software maintenance profiles: an empirical investigation. *Journal of Software Maintenance* 1997, **9**(4):235–251.
10. Martin J, McClure CL. *Software Maintenance: The Problem and Its Solution*, Prentice-Hall, Inc.: Englewood Cliffs NJ, 1983; 472 pp.
11. Parikh G. *Techniques of Program and System Maintenance, 2nd Edition*. QED Information Sciences, Inc.: Wellesley MA, 1988; 463 pp.
12. Perry WE. *Managing Systems Maintenance*. Q.E.D. Information Sciences, Inc.: Wellesley MA, 1981; 371 pp.
13. Arthur LJ. *Software Evolution*. John Wiley & Sons, Inc.: New York NY, 1988; 254 pp.
14. Bendifallah S, Scacchi W. Understanding software maintenance work. *IEEE Transactions on Software Engineering* 1987, **SE-13**(3):311–323.
15. Gustafson DA, Melton AC, An KH, Lin I. *Software Maintenance Models*, Technical Report. Department of Computing and Information Sciences, Kansas State University: Manhattan KS, 1990; 14 pp.
16. Martin RJ, Osborne WM. *Guidance on Software Maintenance, NBS Special Publication 500–106*. National Bureau of Standards: Washington DC, 1983; 66 pp.
17. Grady RB. 1987. Measuring and managing software maintenance. *IEEE Software* 1987, **4**(9):35–45.
18. Reutter J. Maintenance is a management problem and a programmer's opportunity. In *AFIPS Conference Proceedings of the 1981 National Computer Conference, Vol. 50*. AFIPS Press: Reston VA, 1981; 343–347.
19. Chapin N. Software maintenance: a different view. In *AFIPS Conference Proceedings of the 1985 National Computer Conference, Vol. 54*. AFIPS Press: Reston VA, 1985; 328–331.