The State of Art within Evolution and Maintenance of Web Services

Mira Kajko-Mattsson
Department of Computer and Systems Sciences
Stockholm University and Royal Institute of Technology
Forum 100
SE 164 40 KISTA, SWEDEN
mira@dsv.su.se

2004-10-05

Document Identifier: SERV-FORV-9

21 pages

Abstract

In this report, we study the state of art within the evolution and maintenance of traditional and web service systems. The study results show that the domain of traditional software evolution and maintenance is neglected. So far, little has been researched on and published about this domain. Regarding the evolution and maintenance web services, there is no literature available whatsoever. Hence, we can not provide any overview of this domain. In an attempt to elicit some knowledge about it, we suggest an international panel debate to be run at International Conference on Software Maintenance in September 2004 in Chicago, USA.

Table of Contents

1 Introduction

Maintenance is our “Software Cinderella” – a neglected stepsister to Software Development. We do not appreciate it, but we cannot live without it. The real Cinderella, however, dropped her shoe only once, and soon after became a queen. Our “Software Cinderella”, on the other hand, has dropped her shoe(s) many times. Her voyage towards the “royal” recognition has not gained much success yet.

In recent years, however, the software community has started recognising software maintenance as a crucial discipline within software engineering. This is due to the fact that we do maintain more than develop today. Maintenance has become the dominating cost factor in most of the software organisations, and the majority of IT-professionals work within maintenance. Despite this, most of the present process models and most of research are still dedicated to software development. Even at universities, one does not teach maintenance. One teaches development instead.

In this report, we study the state of art within the evolution and maintenance of traditional and web service systems. The study results show that the domain of traditional software evolution and maintenance is neglected. So far, little has been researched on and published about this domain. Regarding the evolution and maintenance web services, there is no literature available whatsoever. Hence, we can not provide any overview of this domain.

Due to the fact that nothing has been published about the evolution and maintenance of web services, we provide an overview of the evolution and maintenance of traditional software systems. We believe that this overview will provide understanding for why nothing has been done so far in the context of web services.

The outline of this report is the following. Sections 2-11 present the state of art of traditional evolution and maintenance. Section 12 provides a historical background of the domain, lists the problems currently experienced, and makes suggestions for their resolution. Finally, Section 13 describes the state of art within the evolution and maintenance of web services.

2 Software life-cycle

Maintenance is just as old as development. It is one of the major phases of software life cycle, where a software life cycle begins when a software product is conceived and ends when the software is no longer available for use. Let us have a look at a coarse-grained outline of a software life cycle as apprehended by the majority of software community today. It is depicted in Figure 1. We divide it into three major phases: (1) *Software Development*, (2) *Operation and Software Evolution and Maintenance*, and (3) *Retirement*. During the development phase, a software organisation develops a software product from scratch using the requirements stated by the customer. After being developed and delivered to the customer, the system steps into two parallel phases, the operation phase and evolution and maintenance phase. While the customer organisation operates the system, the software organisation supports it with its daily operation. Meanwhile, the software organisation further develops (evolves) and maintains the system. It attends to all the customer demands for extending the system with new functionality and for correcting all problems encountered by the customer during the operation. Finally, when the system is no longer of use, it gets retired. This means that the customer decides to terminate it and possibly substitute it with some other newer and better system.

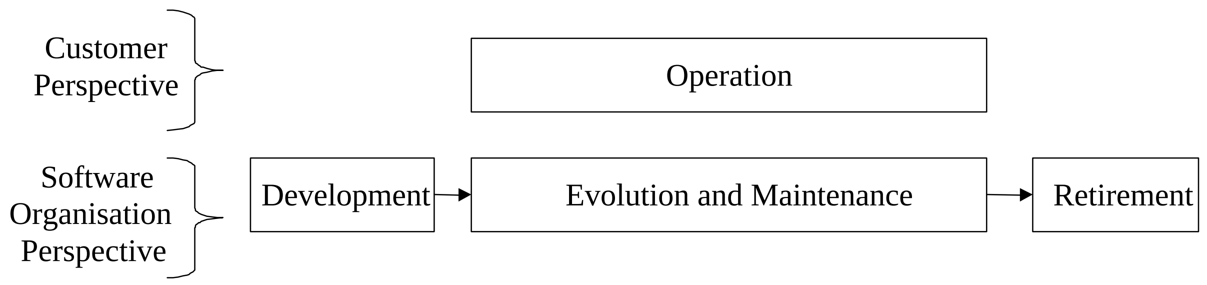

Figure 1. Major phases of the software life cycle

3 Definition of maintenance

Evolution and maintenance are regarded merely as synonyms today. However, the original term for this discipline was maintenance. For this reason, let us first concentrate on its definition first. The subject of evolution will slowly immerse when discussing software maintenance.

Table 1 Definitions of software maintenance

• "…changes that have to be made to computer programs after they have been delivered to the customer or user (Martin and McClure, 1983). • "… the performance of those activities required to keep a software system operational and responsive after it is accepted and placed into production" (FIPS, 1984). • "Maintenance covers the life of a software system from the time it is installed until it is phased out" (von Mayerhauser, 1990). • "…software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity (ISO/IEC 12207, 1995). • "Modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment" (IEEE Std. 1219-1994). • "The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment" (IEEE Std. 610.12-1994).

To define software maintenance has not been an easy task. As depicted in Table 1, various attempts have been made by individual researchers and standardisation organisations. These definitions may vary in their contents and wording, however, they advocate one common thing, the changes made to software systems after they have been delivered to the customer. According to these definitions, maintenance starts as soon as one has delivered the product to the customer. All activities before the delivery are not classified as maintenance.

4 What do we maintain?

What exactly do we evolve and maintain? Software, you say. What do we mean by software then? Do we mean software programs or do we mean all kinds of software artefacts specifying a software system? Software is not only source code and object code. It is more than that. It relates to any system artefact such as:

- Software system encompassing all kinds of requirements-, design-, code-, testing- and other specifications.
- Operating procedures containing instructions to set up and use the software systems and instructions on how to react to system failures.

All these artefacts are evolved and maintained during the software evolution and maintenance phase.

5 Maintenance organisation

The customer expects some organisation to provide changes to the system during the operation phase. But, which organisation? This depends. In one case, the original developer continues to provide all types of support to the customer. In another case, a separate maintenance organisation takes over the evolution and/or maintenance responsibilities. It may also be the case that the original developer may continue to enhance the product with new functionality, and outsource all bug-fixing to some maintenance organisation. Irrespective of who does what, in this report, we call all these organisations *maintenance organisations.*

6 Why do we need maintenance?

Why do we need maintenance? If we concentrated on developing high quality products, would we then need to conduct maintenance? The answer is “no, we would not, but only under the following conditions:

- **Condition 1**: The software product that is delivered to the customer is correct.
- **Condition 2**: The customer is satisfied with the delivered system. He will use it till he gets tired of it, or till he no longer needs it.

These conditions are unfortunately very utopian today. Several studies have shown that there is no such thing as a bug-free software (*Condition 1*). All software that is delivered to the customers contains defects, the defects that one has not managed to discover when testing or inspecting software during development. During operation, these defects may or may not show themselves. If they do, then they must get attended to. If not, then they will stay in the system. Probably it will never be discovered.

Concerning the *Condition 2*, the customer never gets satisfied with the delivered system. Due to various changes within the organisation, such as hardware/software changes, changes to business rules, and evolving or declining maturity of the organisation, the customer will feel the need for modifying his products to suit the new needs. The solution to these needs would of course depend on the product. If it is a mobile phone, the customer would buy another phone. If it is a product supporting, for instance, business operation, the customer would prefer to evolve and maintain the extant product instead of developing or purchasing a new one.

The need for changes described above helps us identify the main goal of software evolution and maintenance, which is, to make the software products continuously provide the required service to their customers. Let us illuminate some of the typical scenarios of providing this continuous service:

- **Fixing bugs**: While operating the software system, the customer encounters problems. For instance, the calculating function delivers wrong results when executing certain functions. To be able to continue using the product, the problems encountered during operation must get attended to.
- **Adapting to environmental changes**: Regularly, the organisations change their hardware and/or software portfolios, such as servers, operating systems. This may affect the organisations' software portfolios. Some changes will have to be made to the software system so that it will be able to run on or co-operate with the new hardware or software.
- **Upgrading**: Changes within the organisation, changes to its working patterns or changes to work deliverables require modifications of the present functionality of the supporting software system. These modifications may mean removing some functionality and replacing it with some new one, and/or adding some new functionality.
- **Improving Performance**: a number of system users may have increased in time. This has strongly affected the system's performance. One needs to modify the system so that it executes faster.
- **Facilitating future maintenance**: Being continuously modified, software systems age. They grow in size and complexity and their architecture gets gradually undermined. This, in turn, leads to the fact that maintainers loose their familiarity with the system, and maintenance becomes a difficult, demanding and costly task. To decrease the maintenance effort and to make the system more maintainable (easy to change), the maintenance organisation restructures the whole of or some parts of the system.

7 Maintenance categories

The scenarios depicted in Section 6 show evidence that maintenance is a very complex phase consisting of diverse activity types. This diversity has lead to the division of maintenance work into four different maintenance types, usually referred to as maintenance categories. As presented in Table 2, the IEEE has designated four maintenance types. They are corrective, perfective, adaptive and preventive. Let us have a look at the IEEE definitions of maintenance categories and map them onto the meanings attached to them by the software community today.

- *Corrective maintenance* is defined as maintenance performed to correct faults in hardware or software. It is mainly understood as the process of resolving software problems reported by the users of the affected software systems. An example of a software problem might be the fact that an arm in a robot turns to the right instead of to the left.

Table 2. IEEE definitions of software maintenance (ANSI/IEEE STD-610.12, 1990)

Corrective maintenance: Maintenance performed to correct faults in hardware or software.
Adaptive maintenance: Software maintenance performed to make a computer program usable in a changed environment.
Perfective maintenance: Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program.
Preventive maintenance: Maintenance performed for the purpose of preventing problems before they occur.

- *Adaptive maintenance* is defined as maintenance performed to make a computer program usable in a changed environment. It is mainly understood as the process of adapting a software system to the changed environment. For instance, some hardware/software parts of the industrial robot might be upgraded, and hence, some modification of software might have to be made in order to accommodate to the new hardware/software upgrades.
- *Perfective maintenance* is defined as maintenance performed to improve the performance, maintainability (ease of change), or other attributes of a computer program. It is mainly understood as the process of adding new features to the extant software system. An example of new features could be a new sorting function in an industrial robot or a new algorithm making this robot execute faster.
- *Preventive maintenance* is defined as maintenance performed for the purpose of preventing problems before they occur. This usually implies discovering software problems and attending to them prior to customer complaints. An example might be the case when a maintenance engineer notices a defect in an already delivered software system. He is sure that this defect, if executed, will cause operational problems to customers. He decides to correct the defect and deliver the defect free software to the customers as soon as possible.

8 Maintenance categories within the industry

The maintenance categories presented in Section 7 are the types suggested by the IEEE standardisation group. They are mainly followed by researchers and students within the academia. However, they are not easily recognised by software practitioners within the industry. Within the industry, one mainly distinguishes between two types of activities: development and maintenance.

By maintenance, the software organisations mean all kinds of corrective changes made to the software system. This corresponds to the IEEE' view of corrective maintenance. By development, the software organisations mean:

- *New development*: Development of a new software system from scratch. This activity corresponds to the development phase as depicted in Figure 1.
- *Continuous development*: Further development of an extant software system by extending it with new functionality or by adapting it to new environment. This corresponds to the IEEE's view of perfective and adaptive maintenance, or the term evolution.

9 Why is it important to categorise maintenance?

It is very important for software organisations to know what exactly happens within maintenance. This knowledge helps the organisations assess their development and maintenance activities and the quality of their products. It also helps them understand the distribution of the resources and make priorities amongst different maintenance tasks.

Making priorities is not easy. Usually, the case is such that corrective and preventive maintenance are delivered for free, while adaptive and perfective maintenance are additionally charged for. For this reason, the software organisations should strive to conduct more perfective and adaptive activities than the corrective and preventive ones. Automatically, our reader would like to draw the conclusion that all the perfective and adaptive tasks should achieve higher priority than the corrective and preventive ones. In an ideal situation, the software organisation should neglect them. The case however is not always such.

Prioritisation must be based on a compromise between the severity of the problems encountered by the customer, the need and benefits of the new modifications or adaptations, the easiness to modify the system, and so on. If a customer has encountered a severe problem, a problem that hinders him from using the system, then of course, he cannot utilise the new extensions. Every minute of a system failure costs the customer organisation a lot of money. At its worst, it may jeopardise its business or even human life. The maintenance organisation must then urgently attend to the software problem. If, on the other hand, the customer has encountered minor problems, the so-called nuisances, that might be lived with for the time being, the software organisation may prioritise the requests for enhancements and attend to minor software problems when it is convenient. Finally, if the software system is highly unmaintainable, that is, every single change to the system may make system invalid or behave strangely, the software organisation may have no other choice then to reengineer the whole system and hope that during this time the customer may wait with extensions and will not encounter any serious problems.

The task of adding functional requirements to existing systems can be likened to the architectural work of adding a new room to an existing building. The design will be severely constrained by the existing structure, and both the architect and the builders must take care not to weaken the existing structure when additions are made. Although the costs of the new room usually will be lower than the costs of constructing an entirely new building, the costs per square foot may be much higher because of the need to remove existing walls, reroute plumbing and electrical circuits and take special care to avoid disrupting the current site.

Figure 2. The constraints of modifying an existing systems

10 Maintenance versus development

Maintenance is regarded as a continuation of development. During development, one builds a completely new system. During maintenance, one changes an existing system. These two activities seem to have much in common. There are however some fundamental differences between them. These differences are due to the constraints of the existing system. To understand these constraints, we invite our reader to follow Jones's line of reasoning as described in Figure 2.

When designing new changes, the maintainer must always consider the existing system in the following situations:

- **Study the modification requirement**: The maintenance engineer must understand what the customer wants. This activity is both common for development and maintenance. The difference is that during maintenance, the maintenance engineer must understand the modification in the context of the extant software system.
- **Learn the system or system part(s) to be changed**: The maintenance engineer must investigate the extant system in order to understand its architecture and design. This activity is typical for maintenance. During development, the engineers do not need to study any existing system.
- **Suggest one or several change designs**: Usually, the software engineers must create several designs. This activity is both common for development and maintenance. The difference is that during maintenance, one must pay heed to the already extant design.
- **Evaluate the design**: Evaluate the resources needed to implement the change, the risks that might appear if the change gets implemented, the side effects of the change and so on. This activity is both common for development and maintenance. The difference is that during maintenance, one must not only evaluate the suggestion for change, but also how it affects other extant parts of the system. For instance, what happens if I change *Component X*, containing common functionality utilised by many other components? First of all, one must know that this component is utilised by other components. Not always, the maintainer knows it, either because this fact is not documented or because of other reasons. If changes are to be made to this component, then the maintainer must assess the ripple effect (or side-effect) of this change. The ripple effect means that changes made to *Component X* imply changes to other components, and so on.
- **Determine the skills and knowledge to implement the change**: Not everybody within the organisation may conduct all sorts of changes. Hence, one must check whether specific skills and knowledge are required to manage some types of changes. This activity is both common for development and maintenance. The difference is that during maintenance, one must not only possess the expert skills required for the implementation, but also one must have a good knowledge of the system to be changed.
- **Test the change**: All the changes must be tested. It is not enough however to only test the changes. One must also test other parts of the system that have not been changed. One must do so in order to ensure that the new changes have not introduced new defects into the system. In other words, one must repeat all the former tests that have been made during development and former stages of evolution and maintenance. This type of testing is called "regression testing":

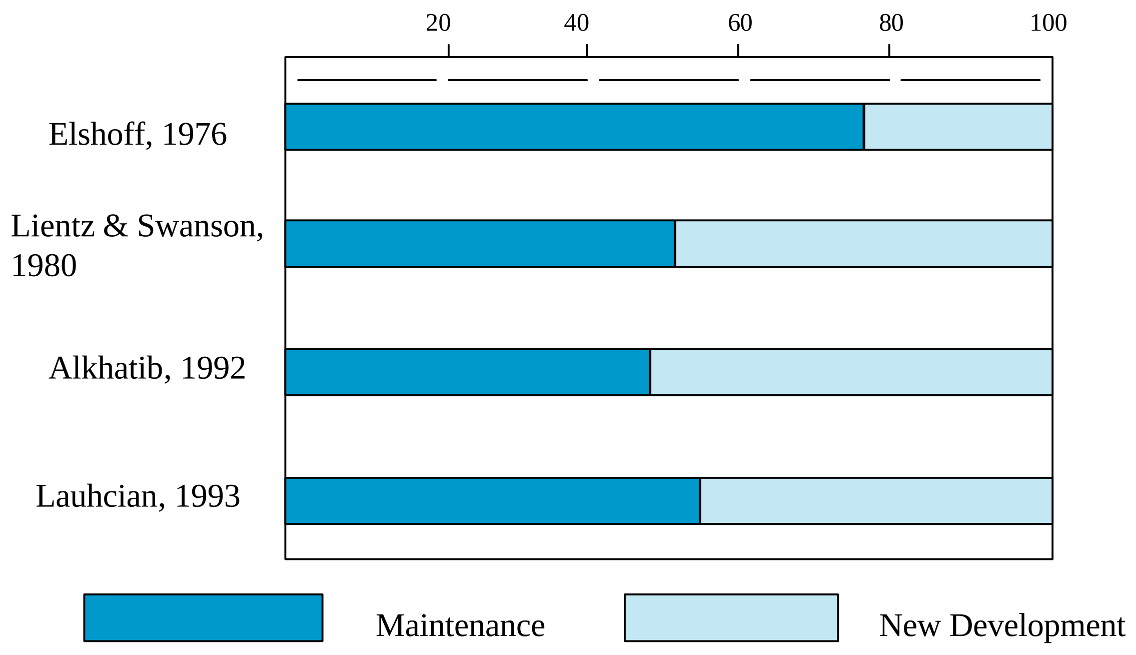

Figure 3. Cost of maintenance (Figure 1.6 in Pigoski, 1997)

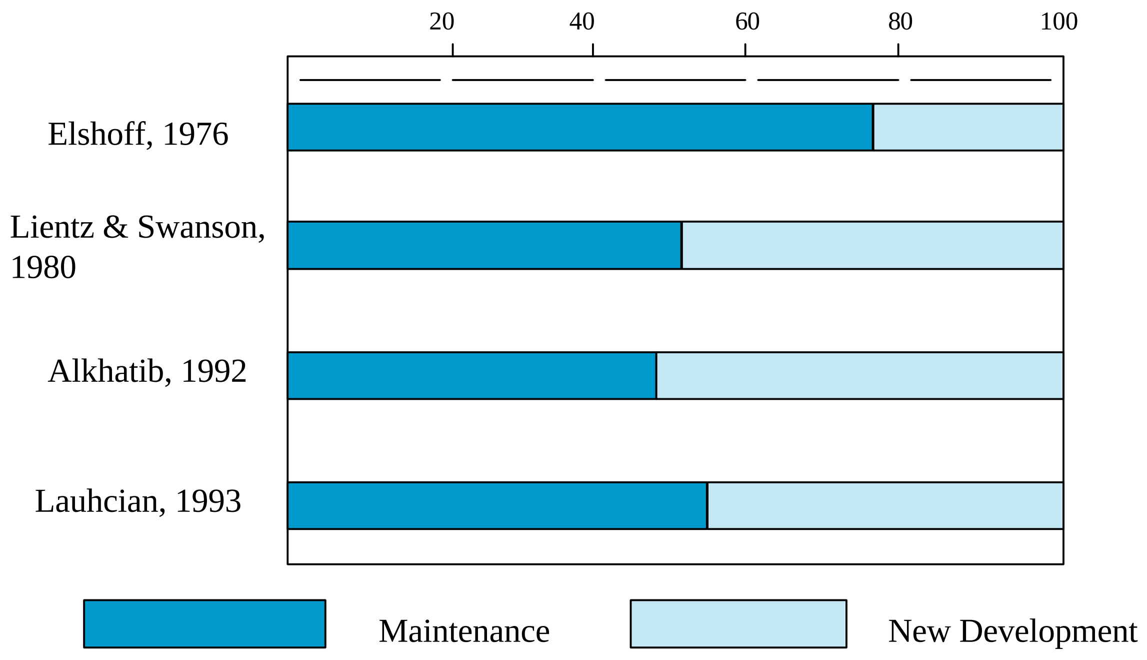

Figure 4. Cost of maintenance (Table 3.1 in Pigoski, 1997).

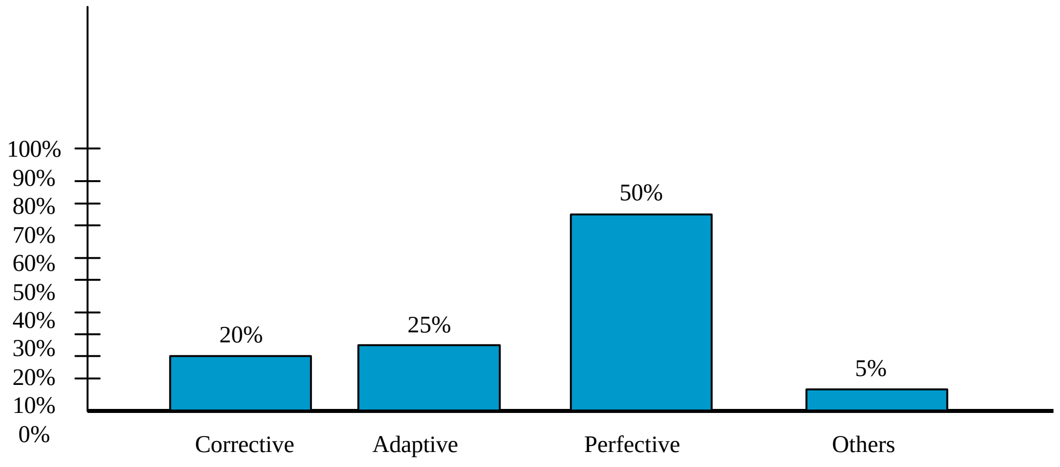

Figure 5. One of the studies of the distribution of maintenance cost (Swanson, 1978)

11 Maintenance is Expensive

There is sufficient data indicating that maintenance is an expensive activity and that its cost is continuously increasing. There is however no agreement on the proportion of the overall software life cost it takes, and how the cost is distributed across the maintenance types. As depicted in Figures 3 – 4 published numbers point out that maintenance costs between 40% to 90%.

There are few publications reporting on the cost of each individual maintenance category. The reported ones are the following (Lientz and Swanson, 1980):

- Corrective maintenance: 16-22% .
- Perfective maintenance: 55%.
- Adaptive maintenance: 25%.
    - Preventive maintenance: No data has been published about the cost of this maintenance category.

12 Status within software evolution and maintenance

Historical Background of Software Maintenance.

For many years, maintenance has not been regarded as glamorous work (Mamone 1994, Schneidewind 1987). It has been looked upon as a low-prestige and "dirty" activity (Rose 1988). It has been unpopular and hated by many software engineers (Parikh 1988). Software engineers preferred to develop new software systems instead of maintaining the old ones. If they started their career within maintenance, then they quickly changed to development. To work with maintenance has been akin to having bad breath (Schneidewind 1987). This attitude has caused maintenance to become a neglected area (Boehm 1976, Boehm 1988, Swanson 1990, Weinberg, 1988). Maintenance has become a complex, expensive and little understood domain within software engineering both within the industry and academia (Mamone 1994, Rombach 1992).

In the 70's and 80's, some researchers raised protests against this negligence; however, to little or to no effect (Boehm 1976, Boehm 1988, Parikh 1985, Parikh 1988, Schneidewind 1987). Software maintenance has been and still is an under-researched activity. For many years, there has been no common maintenance process framework to follow. Research has been mostly concentrated on creating development models. The development models, however, are not easily transferred to a maintenance environment (Lientz 1983). Moreover, maintenance tasks vary a great deal, which makes it hard to find a common maintenance process model (Mayrhauser 1994). The lack of maintenance process models is one of the main reasons that most of the changes have been done in an ad hoc manner (Rombach 1992). The engineers had no instructions for how to infuse changes into the software system.

There has been and there still is deterioration of the quality of the software systems (Chapin 1985, Mamone 1994, Rombach 1992). To some extent, this inadequacy has been due to the fact that during development, the software systems have rarely been designed for maintenance (to be readily modified), and that during maintenance the maintenance work has been done in an ad hoc manner (Parikh 1986, Pari1988). This has contributed to increased program complexity, which, in turn, has caused difficulties in determining the side effects of changes (Schneidewind 1987). A typical side-effect happens when you change one line in a common function, and this change invalids the execution of major parts of the system using this common function.

Maintenance has been viewed as a second class activity, with an admixture of on-the-job training for beginners and low-status assignments for the outcasts and the fallen (Gunderman 1988). The original developers have not usually maintained their systems. Other less qualified and less experienced personnel have been assigned to maintenance tasks instead. As a result, it has been difficult to understand someone else's code (alien code) (Mamone 1994, Schneidewind 1987). To add zest to it, many maintenance engineers have lacked formal training in performing software maintenance. Usually, they had to learn on their own. Hence, even simple changes have become complex ones, often causing severe damage to the system maintained (Freedman et.al. 1988, Lientz 1983).

There has been and there still is high turnover of maintenance staff (Lientz 1983, Takang 1996). It has been difficult to get good people to work with maintenance (Glass et.al. 1981, Jones 1988). The opinion has prevailed that maintenance is a dull, inferior, non-creative, non-challenging activity requiring no more than average intelligence (Foster 1989, Liu 1988). In the past two decades, however, protests have been raised that maintenance should not be a place for new hires, trainees, or misfits (Landsbaum et.al. 1992). On the contrary, maintenance requires a great deal of creativity and ingenuity (Foster at.al. 1989, Jones 1988). Hence, a maintenance engineer should be a highly skilled, intelligent, and creative diagnostician.

All this points out that software maintenance has been and still is a highly neglected topic. In the past decade, however, maintenance has been recognised as a crucial discipline within software engineering. This is mainly due to the fact that maintenance has become the dominating cost factor in most software organisations. It consumes more than half of the life-cycle resources of software systems (Arthur 1988, Boehm 1973, Cashman et.al. 1980, Glass et.al. 1981, Jones 1994, Mills 1976, Moad 1990, deRoze et.al. 1978). Some authors even claim that maintenance consumes more than 90% of the life-cycle cost (Pigoski 1997).

The cost of maintenance does not decrease. On the contrary, it does and it will increase. There are many reasons for this. Some of them have already been presented above. The main reason, however, is the fact that we live in the age of a shift from software development to software evolution and maintenance. Rarely does product development start from scratch today. Instead, emphasis in industry is on further development (evolution) of extant system portfolios.

Problems

In comparison to software development, software maintenance is still an immature area. The reader who is not sufficiently acquainted with the subject, may not easily distinguish the problems reigning with this domain (Kajko-Mattsson 2001c). For instance, he may accept the name "software maintenance", the IEEE definitions of maintenance and its categories without batting an eyelid and conclude that they are clear, well-limited, and informative enough. The experienced reader, on the other hand, may feel somewhat confused and would probably ponder on the following questions?

- Is the name "maintenance" the right name for this engineering domain?
- When does software maintenance begin and when does it end?
- Where does it fit in the software life cycle?
- How does software maintenance relate to software development?
- What exactly happens during software maintenance?
- Does the term maintenance cover all types of post-delivery changes?
- Aren't the definitions of maintenance categories too general?
- Are they exhaustive and exclusive?

The experienced reader is definitely right to feel confused. Being still immature, the domain of software maintenance suffers from many uncertainties and problems today. Some of them are (1) choice of an appropriate name, (2) definition of maintenance, (3) categorisation of maintenance activities, (4) disagreement in the use of terminology, (5) lack of maintenance process models, and (6) measurement problems. These uncertainties and problems greatly contribute to a chaos presently reigning within the software maintenance domain. In the following sections, we shortly describe them.

Choice of an Appropriate Name

The term "maintenance" has been regarded as an unpopular name. There has been a debate going on whether it should be changed or not. This debate started as early as in the eighties, and it has not ended yet. Should we or should we not call maintenance maintenance?

During this sustained debate on the appropriateness of the term "maintenance", different terms have been suggested (Chapin 1985, Ghezzi et.al. 1991, McGregor 1988, Lehman 1980, Schneidewind 1987). They were the following:

software support, software evolution, co-evolution, continuation engineering, production monitoring, systems control, post implementation development, system tuning, application software support.

These terms have come up in response to the opinion that the term maintenance is a misnomer. According to Mills, maintenance connotes restoring a device to its original **correct** state, but the program is never correct to begin with. All programs have defects. Hence, we cannot claim that we can restore the system to the original correct state. This view has been supported by Yourdon, who has shown that large software systems after being delivered still contain a large number of defects.

Some authors are of the opinion that maintenance should be looked upon as continued development (Ghezzi et.al. 1991, McGregor 1988, Lehman 1980, Mills 1988, Parikh 1988, Schneidewind 1987, Schneidewind 1999, Takang 1996). They believe that software systems always evolve; hence, they are never completed. This could be justified by the fact that the work of maintenance includes adding features not originally designed in the system, the so-called enhancements. According to them, enhancements are not maintenance in the conventional sense. Enhancements involve additional development work to fit into the framework of the existing software system. Therefore, a more appropriate term for this activity would be “evolution”.

Definition of Maintenance

There has been a disagreement about the definition of software maintenance. Still, there prevails a controversy on the choice of maintenance scope, its constituents, time span, and on drawing a dividing line between software development and software maintenance (Pigoski 1997, Martin 1983, Schneidewind 1987). No wonder that there have been so widely varying estimates of costs of software maintenance, spanning between 40%-90% of the total software life cycle cost. The major critique of the IEEE definition was its statement that maintenance is typically a postdelivery activity (see IEEE’s definition of maintenance in Table 1). According to some authors, maintenance should start earlier than at the delivery Martin 1983, Pigoski 1997). It should run in parallel with development. During this time, the maintainers should follow the development of the system and continually evaluate its maintainability.

Maintenance Categorisation

To add zest to the problem of defining maintenance, there is also a great deal of confusion concerning the choice and definition of maintenance categories. Protests have been raised against the IEEE definitions (see IEEE definitions of maintenance categories in Tables 1 and 2). The IEEE has designated four maintenance categories. They are corrective, perfective, adaptive and preventive. This categorisation, however, is not mutually exclusive (Chapin 2000b). One study has revealed that definitions of perfective and preventive maintenance overlap and are differently understood by the software community (Kajko-Mattsson 2000). For instance, the IEEE suggests that the improvement of maintainability belongs to perfective maintenance. The software community, on the other hand, attributes the improvement of maintainability to preventive maintenance.

There has also come some critique that the IEEE’s definition of maintenance categories is too general. (Chapin 2000a, Chapin 2000b, Foster et.al. 89). The definitions are not explanatory enough and may lead to misunderstanding and overlap. For instance, the IEEE definition of software maintenance states that it is a *“modification of a software product after delivery, to correct faults, to* *improve performance or other attributes, or to adapt the product to a modified environment"*. Out of this definition, we may clearly distinguish corrective modifications and adaptive modifications. However, "other attributies" are not easily understood. Do they concern perfective maintenance?

Chapin claims that the IEEE types of maintenance are not exhaustive and exclusive enough. They are too much intention based. This means that the choice of the maintenance category rather depends on the intentions of the classifier than on the objective basis of the changes made to the system. One and the same maintenance task can be differently classified (e.g., either as corrective or perfective). Example: Assigning categories to the maintenance tasks can be, for instance, political. Some major corrective tasks may be classified as perfective maintenance. The managers may deliverately do this choice. Within perfective maintenance, they do business, within corrective maintenance they loose business.

Regarding the definition of preventive maintenance, the IEEE does not define preventive maintenance as a member of an exhaustive set of maintenance types. It rather defines it as a type that overlaps other types such as perfective, adaptive and corrective. This means that a maintenance request can be understood as an instance of both perfective and preventive (not-either-of), or adaptive and preventive, or corrective and preventive.

Disagreement in the Use of Maintenance Terminology

We do not use common maintenance concepts uniformly. We seem to take it for granted that everyone understands the concepts we are using. However, such is not always the case. Terminology used by the academia and industry differs greatly (Oman 1998). Even within one and the same organisation, different terms may be in use for one and the same concept. A good example of this is the concept of *maintenance*. For the academia and a few industrial organisations, maintenance relates to the management of both software problems and enhancements. For the majority of industrial organisations, however, the management of software enhancements equates to development (Schneidewind 1987). Another term being frequently abused is *error*. Many engineers use it for either a "software problem" or a "defect", while others use it for totally different phenomena.

Shortage of Maintenance Process Models

Still today, we do not possess detailed maintenance process models. To the knowledge of the author of this book, there are only two internationally recognised maintenance standard models – IEEE 1219 and the newly introduced models such as ISO/IEC FDIS 14764 (ISO 1999) and the IT Infrastructure Library Service Support Model (ITIL Service Support) (OCG, 2004). The first tow models however, are very general. The third model is the most detailed model today.

Despite the fact that maintenance categories strongly differ, they propose generic process models for all the categories. Being too general, these models do not help in acquiring a deep understanding of the scope of each maintenance category. They do not offer the visibility required to each type of maintenance work.

The process models are for the most part capability-oriented. They view maintenance mainly from the perspective the capabilities that the organisations should achieve. An example of a capability might be "*The project follows a written organisational policy for managing the system requirements allocated to software*".

Listing capabilities, however, is not enough. Such models do not suggest the process activities and their order. They do not give enough guidance to the organisations on why to implement or improve their processes. On the contrary, being too terse in their contents and attempting to cover all maintenance categories with one and the same process model, they may mislead the organisations in their process implementation and improvement efforts. Organisations need suggestions for process activities and thorough explanations of and motivations for implementing them.

Measurement of Maintenance Cost

The software community unanimously agrees that software maintenance costs are very high today. Agreement, however, stops at this point. On checking the cost of maintenance in comparison to development as presented in Chapter 1.10, we arrive at widely varying results. As already noted above, published numbers point out that maintenance costs between 40% to 90%.

Why does the cost of maintenance vary between 40-90%? Why are there so few reports on the costs of each individual maintenance category? There may be many answers. One of them is the already mentioned fact that we do not share a common, deep and objective understanding of maintenance, of maintenance categories and of their inherent processes. We are not even unanimous on what maintenance is, on its scope, and on its relationship to development.

How Can We Remedy the Problems?

One way to remedy most of the aforementioned uncertainties and problems is to construct separate process models for each maintenance category. Maintenance categories do differ too much in order to be lumped together under one and the same model. We believe that each maintenance category deserves its own process model (a specialised model). This would offer a good opportunity to scrutinise the definition of each maintenance category, its name, goal, and most importantly, this would lead us towards objective understanding and objective measurement. Specialised models would also aid in agreeing on a single classification system for each maintenance category and its inherent activities. They should be fine-grained enough in order to allow maximal visibility into the process. To the knowledge of the author of this thesis, there is no process model explicitly defined for a particular maintenance category.

Visibility into the maintenance process can be achieved only in cases when the maintenance work is perceptible, distinguishable and measurable enough throughout the whole life cycle phase (Kajko-Mattsson 2001b, Swanson 1999). This requires that all of our life-cycle processes are fine-grained defined, so that insight can be made into the process steps taken, maintenance effort and resources put in to conduct them, and the effects of building in and preserving maintainability during different life cycle phases (Kajko-Mattsson 2001a, Kajko-Mattsson 2001b, Swanon 1999).

Fine-grained process models are not enough, however, to aid in maximising the visibility and objective understanding of maintenance. Each such process must be thoroughly explained and motivated for. It is only in this way that our processes and their inherent activities may become perceptible, distinguishable, objectively understood, and objectively measurable.

13 Evolution and Maintenance of Web Services

Web services are highly vulnerable and subject to constant change. Hence, they offer a novel challenge to software engineering. This challenge has not yet been investigated. From the evolution and maintenance perspective there are many things that must be examined. These include the issues of evolution and maintenance processes, products and roles involved within the processes, and the organisational changes required for adopting to the web service application mode.

When browsing through the current research literature, mainly in the IEEE, ACM, Wiley&Sons databases (KTH 2004), we did not succeed to find any information about this domain. For this reason, we have decided to suggest a panel debate to International Conference of Software Maintenance (http://www.cs.iit.edu/~icsm2004). The panel would discuss the following questons:

- Does evolution and maintenance of web service applications differ from the evolution and maintenance of traditional software systems? The answer should be given from the product-, process-, role- or organisational perspective.
- What are the problems encountered during the evolution and maintenance of web services that are not encountered in the traditional evolution and maintenance?
- Can we remedy these problems and if yes, how?

The panel got accepted and it was lead by the author of this report. The panellists came from the academia and industry, mainly from Europe and United States. During the panel, one came to an agreement that little was known about the evolution and maintenance of web services today. However, some problems were identified and suggestions for how to remedy these problems were made. These are described in the SERVIAM report: SERV-FORV-10 (Kajko-Mattsson 2004).

References

Arthur L J, 1988, *Software Evolution: The Software Maintenance Challenge*, John Wiley & Sons.

Boehm B W, 1976, *Software Engineering*, IEEE Transactions on Computers, Vol. C-25, No. 12.

Boehm B W, 1988, *Software Maintenance*, In Parikh G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., Wellesley, Massachusetts, pp. 51-54.

Cashman P M, Holt A W, 1980, *A Communication-Oriented Approach to Structuring the Software Maintenance Environment*, Software Engineering Notes, ACM SIGSOFT, Vol. 5, No. 1, pp. 4-17.

Chapin N, 1885, *Software Maintenance: A Different View*, In Proceedings, National Computer Conference, AFIPS Press, Reston, Virginia, Vol. 54, pp. 507-513.

[Chapin N, 2000, *Software Maintenance Types – A Fresh View*, In Proceedings, IEEE International Conference on Software Maintenance, pp. 247-252.

Chapin N, 2000, *Do We Know What Preventive Maintenance Is?*, In Proceedings, IEEE International Conference on Software Maintenance, pp. 15-17.

Foster J R, Jolly A E P, and Norris M T, 1989, *An Overview of Software Maintenance*, British Telecom Technology Journal, Vol. 7, No. 4, pp. 37-46.

Freedman D P, Weinberg G M, 1988, *Maintenance Reviews*, In Parikh, G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., pp. 89-92.

Ghezzi C, Jazayeri M, and Mandrioli D, 1991, *Fundamentals of Software Engineering*, Prentice-Hall International, Inc., New Jersey,.

Glass R L, Noiseux R A, 1981, *Software Maintenance Guidebook*, Prentice-Hall.

McGregor B, 1988, *Program Maintenance*, In Parikh G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., pp. 149-155.

Gunderman R E, 1988, *A Glimpse into a Program Maintenance*, In Parikh G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., pp. 55-59.

ISO/IEC 14764: Information technology – Software Maintenance, Reference Number: ISO/IEC 14764:1999(E).

Jones R R, 1988, *Creativity Seen Vital Factor, Even in Maintenance Work*, In Parikh, G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., pp. 83-85.

Kajko-Mattsson M, 2000, *Preventive Maintenance! Do We Know What It Is?*, In In Proceedings, IEEE International Conference on Software Maintenance (ICSM 2000), pp. 12-14.

Kajko-Mattsson M, Westblom U, Forssander S, Andersson G, Medin M, Ebarasi S, Fahlgren T, Johansson S-E, Törnqvist S, Holmgren M, 2001a, *Taxonomy of Problem Management Activities*, In Proceedings, IEEE Conference on Software Maintenance and Reengineering, 2001, pp. 1-10.

Kajko-Mattsson M, Forssander S, Olsson U, 2001b, *Corrective Maintenance Maturity Model (CM^3): Maintainer's Education and Training*, in Proceedings, IEEE International Conference on Software Engineering, 2001, pp. 601-619.

Kajko-Mattsson, M, 2001c, *Motivating the Corrective Maintenance Maturity Model (CM^3)*, In Proceedings, Seventh IEEE International Conference on Engineering of Complex Computer Systems, 2001, pp. 112-117.

Kajko-Mattsson M, 2004, Evolution and Maintenance of Web Services, SERVIAM Report (SERV-FORV-RAPP-10).

KTH, 2004, Royal Institute of Technology Library, www.lib.kth.se.

Landsbaum J B, Glass R L, 1992, *Measuring and Motivating Maintenance Programmers*, Prentice Hall, Englewood Cliffs, NJ 07632.

Lehman M, 1980, *Programs, Life Cycles, and Laws of Software Evolution*, Proceedings of the IEEE, Vol. 68, No. 9.

Lientz B P, 1983, *Issues in Software Maintenance*, ACM Computing Surveys, Vol. 15, No. 3, pp. 272-278.

Liu C, 1988, *A Look at Software Maintenance*, In Parikh G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., pp. 61-71.

Mamone S, 1994, *The IEEE Standard for Software Maintenance*, Software Engineering Notes, Vol. 19, No. 1, pp. 75-76.

Martin J, McClure C, 1983, *Software Maintenance, The Problem and Its Solutions*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

v. Mayrhauser A, 1994, *Maintenance and Evolution of Software Products*, Advances in Computers, Academic Press, Inc., Vol. 39, pp. 1-49.

Mills H, 1988, *From Development to Maintenance*, In Parikh, G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., p. 86.

Moad J, 1990, *Maintaining the Competitive Edge*, DATAMATION. 61-6.

OGC, 2004, Office of Government Office, The IT Infrastructure Library Service Support (ITIL Service Support), HMSO, Licensing Division, St Clements House, 2-16 Colegate, Norwich, NR3 1BQ, United Kingdom.

Oman P W, 1998, *Hitting the Moving Target: Trials and Tribulations of Modeling Quality in Evolving Software Systems*, Panel on IEEE International Conference on Software Maintenance, pp. 66-67.

Comment: We refer to prof. Munson's statement raised during this panel debate on our inability to define common concepts within software engineering.

Parikh G, 1986, *Handbook of Software Maintenance A Treasury of Technical and Managerial Tips, Techiques, Guidelines, Ideas, Sources, and Case Studies for Efficient, Effective, and Economical Software Maintenance*, John Wiley & Sons.

Parikh G, 1988, *Techniques of Program and System Maintenance*, QED Information Sciences, Inc., Wellesley, Massachusetts.

Pigoski T M, 1997, *Practical Software Maintenance*, John Wiley & Sons.

DeRoze B, Nyman T, 1978, *The Software Life Cycle – A Management and Technological Challenge in the Department of Defense*, IEEE Transactions On Software Engineering, Vol. SE-4, No. 4, pp. 309-318.

Rombach H D, Ulery B T, Valett J D, 1992, *Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL*, Systems Software, Vol. 18, pp. 125-138.

Rose L A, 1988, *Management Considerations and Techniques*, In Parikh, G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., p. 123.

Schneiderwind N, 1987, *The State of Software Maintenance*, IEEE Transaction on Software Engineering, Vol. SE-13, No. 3, pp. 303-310.

Schniedewind N, 1999, *Software Maintenance is Nothing More Than Another Form of Development*, Panel 1, In Proceedings, IEEE International Conference of Software Maintenance, pp. 63-64.

Swanson B E, 1999, *IS Maintainability: Should It Reduce the Maintenance Effort?*, ACM SIGCPR, New Orleans LA, USA.

Takang A A, Grubb P A, 1996, *Software Maintenance – Concepts and Practice*, International Thomson Computer Press.

Weinberg G M, 1988, *Worst First Maintenance*, In Parikh, G, Techniques of Program and System Maintenance, QED Information Sciences, Inc., pp. 131-133.