Implementation of Software Management and Maintenance

Dr. Sumeet Kumar
(Assistant Professor, Computer Science Department,M. M. Modi College, Patiala,ksumeet2012@yahoo.com)

Abstract:

Software maintenance and management means to manage and maintain software for future expansion. Software maintenance also includes debugging of errors. In software engineering , the most important is the management and maintenance of software that software development company gives to their clients. It deals with how to improve the performance by zeroining all the faults that is errors. Most people think ,software maintenance only deals with removing of errors while delivering the software but it actually deals with managing and using the same software for future expansions of organizations. For example , consider a new bank that makes software for their all banking operations or transactions .Initially this software will work very good but with the passage of time as the customer base of bank grows and also government makes new policies etc etc ,may be this old software doesnot work efficiently, so here software management and maintenance is needed. Meir M. Lehman[1] was the one who introduced software management and maintenance concept for organizations in 1969.Making new software is not so expensive but changing whole old software is very expensive and cumbersome .

Keywords:

Software maintenance, Software management , Software developer, testing etc

Introduction:

Maintaining old software is very important for the profit making of any organization. There are number of key software maintenance issues such as customer privacy, to give friendly service to customers ,cost of maintenance of software etc. there are both technical and managerial issues are involved in software management and maintenance. The main aim of software management and maintenance is to use the same software that organization developed during starting of organization, as long as possible by taking care of future growth such as increased customer base ,adjusting new government policies (if any) etc. Let us consider the simple example , few years back most of customers has only landline phone numbers but now a days every customer has mobile phones , even some have two mobile numbers .So , there should be a provision to manage such changes in a software. Organization cannot completely change the old software for this simple change in customer profile."Value of old software should be preserved" must be the aim of software management and maintenance. Maintenance period may be of 25 years or more .Software management and maintenance is the branch of software engineering that's deals with the use of same software for longest period of time by making very few modifications in existing software as low as possible.There are number of common activities [2] of software management and maintenance such as versioning, building, testing,packing , configuring , installing ,distributing etc:

Versioning.

Version control system refers to keep track of the changes to source code . Sometimes there are multiple software engineers working on the same software ,in such cases versioning is required.

Building.

In this process, the software is build or compiled**.** If any kind of error is detected in this stage then that error is removed from the source code.

Testing.

Testing is most crucial before using the software in organization.Number of tests are done using dumy data before introducing software finally in an organization.In testing all the test run results were checked and made necessary changes if required.

So, testing is done after the compilation of the software.

Packaging.

Once software has been built then next step is packaging. Packaging means to share the developed software with others.

Configuring.

Configuring means that the developed software should run on any platform. Sometimes same software behaves differently on different operating systems. This should be carefully handled. So in this process, the software is configured in such a way that it can run in any kind of environment or any specified environment.

Installing.

After software has been built and configured, the next procedure is to install the software into the systems of organization where it actually works.

Distributing.

After the packaging of software, the next procedure is to publish[3] it on the internet.

OBJECTIVES OF THE STUDY:

The objectives of present study are :

1. To study the implementation process of software.
2. To study the maintenance process of software.
3. To study the efficiency of processes involved in software management.

RESEARCH METHODOLOGY

In a survey , it was found that two third around 75% of the software management and maintenance effort was on the versioning and building and testing consumed around about 22%.From most of the studies it was found that the crucial part in software management and maintenance is the involvement of end user. Because ultimately the end users are one that gives the organization profit. There are number of adjustment factors in software management and maintenance such as Maintenance specialists, Y2K and special search engines etc.

Impact of key adjustment factors on maintenance (sorted in order of maximum positive impact)[4]

Maintenance Factors PLUS	Range
Maintenance engineer	32%
Techinical staff	34%
Table-driven variables and data	33%
Low complexity of base code	32%
Y2K and special search engines	30%
Code restructuring tools	29%
Re-engineering tools	27%
High level programming languages	25%
Reverse engineering tools	23%
Complexity analysis tools	20%
Defect tracking tools	20%
Y2K ―mass update‖ specialists	20%
Automated change control tools	18%
Unpaid overtime	18%
Quality measurements	16%
Formal base code inspections	15%
Regression test libraries	15%
Excellent response time	12%
Annual training of 20 days	12%
High management experience	12%

There are number of factors that decreases the performance of software such as suitable maintenance tools, such as error prone functions in a software, program complexity, fixed data ,inexperience staff ,poor handling of software etc. List of these factors is given below-

Impact of key adjustment factors on maintenance (sorted in order of maximum negative impact)

Maintenance Factors	Minus Range
Error prone functions	50%
Fixed variables and data	45%
Inexperience staff	40%
Complexity of software	30%
Poor ease of use	18%

Evidence of activities that produce these changes serves as the criteria; observation and comparison provide the evidence. The comparison is of the relevant parts of the software as of *before* the alleged evolution or maintenance, with the relevant parts of the software as of *after* the alleged evolution or maintenance. The observation is of the activities and their artifacts.As the recent ontology has pointed out, software evolution or maintenance commonly involves several too many processes or activities that may result in from none too many changes in the software. From the set of all changes observed, made, or detected as attempted from the evidence, a dominant process or grouping of processes or activities typically emerges, with supporting or associated processes or groupings usually also present. We define our proposed classification to be exhaustive with mutually exclusive types, grouped into clusters, where any instance of software evolution or software maintenance 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. The order of the

types and their clusters is significant because of their different impacts. One dimension is the
impact of the evolution or maintenance on the customer‘s ability to function effectively using the software implemented system in
meeting its goals i.e., the business process impact on the customer or how the customer does its/his/her business and the customer‘s
satisfaction with the system. This is drawn from left (low impact) to right (high impact). The number of blocks suggests the likely range of the business process impact on the customer. For instance, enhancing the software with new functionality for the customer typically has more effect on the customer‘s ability to meet its goals than does changing the non-code documentation. The other dimension is the impact of the evolution or maintenance on the software itself. This is diagrammed top (low impact) to bottom (high impact). For instance, reformatting a page of a user manual typically has less effect on the software than does correcting (fixing) a bug. With a
variety of activities carried out, many of the types of software evolution and software maintenance can to some degree be observed, even when some particular type appears to be dominant The question then is what are the relationships among the types?

The key to understanding the relationships goes back to the three criteria decisions A, B, and C. All three must always be asked.

Responding ‘No‘ to any one of them identifies a respective cluster of types where at least one of the type decisions may earn a ‘Yes‘ for some type. It also for the A and B criteria decisions takes us on to ask the next criterion decision, B or C respectively. In addition, responding either ‘No‘ or ‘Yes‘ to either the B or C criteria decision is also effectively invoking the ‘No‘ response cluster associated with the A or B criteria decisions respectively. Consider an example. A maintainer, after studying the documentation including the source code, rewrote the source code for one module, and without changing any other documentation,
compiled the revised module, tested it, and installed it in the production library. The only consequence the customer saw was a
faster response time.

To explain , where software management and maintenance is needed, consider the coding of some college class where class initially had 30 students. This code prepares the result of 30 students-

```
#include<iostream.h>
#include<conio.h>
struct stu
{
char n[10];
int r;
int a;
int b;
int c;
int T;
};
void main( )
{
 Stu t[30];
int i;
clrscr();
for(i=0;i<30;i++)
{
cout<<”Enter name”;
cin>>t[i].n;
 cout<<”Enter roll number”;
cin>>t[i].r;
cout<<”Enter marks”;
cin>>t[i].a>>t[i].b>>t[i].c;
t[i].T=t[i].a+t[i].b+t[i].c;
cout<<”Total Marks=”<<t[i].T;
cout<<endl<<endl<<endl;
getch( );
}
```

Now the above code calculates the result of 30 students but the above code fails if number of students in coming year increases or decreases .So here comes software maintenance and management.

DISCUSSION AND CONCLUSION

The term ‘evolution‘ has been used since the early 1960s to characterize the growth dynamics of software. Halpern in 1964 applied it to software implementing programming systems, for example. The term ‘evolution‘ in relation to application systems took hold more slowly, with an early mention being in 1970 and a research being published the next year . In 1985, Lehman and Belady, keying off their study of several major systems, identified five laws of evolution as intrinsic to ‘Esystem‘ software - i.e., software being used in real-world domains and having stakeholders
because of evolutionary pressures present during development and during the continuing processes aimed at maintaining the system and its cost effectiveness and acceptability to its stakeholders within a changing environment.Responding to

these pressures typically results in an asymmetric or clumpy growth in system functionality.

As Bennett and Rajlich have pointed out, software evolution currently has no one widely accepted definition, but some

researchers can use ‘software evolution‘ instead of the term ‘software maintenance‘. Others prefer a narrow view that software evolution occurs when either *enhancive* or *reductive* maintenance is carried out.

The current common view is that software evolution occurs when the software maintenance done is of the *enhancive*, *corrective*, or *reductive* types (any of the business rules cluster), or it changes software properties sensible by the customer, i.e., it is of the *adaptive* or *performance* types. This objective-evidence basis for identifying software evolution is, we believe, easier to apply and generally more useful than an intentions-based typology. The term ‘maintenance‘ has been a term used for making deliberate modifications of the software part of systems implemented at least in part with software, since the early 1960s. The terms ‘change‘ or ‘modification‘ were common if the activities were done by the personnel who had done the original development of the software. ‘Maintenance‘ was associated with change or modification made by other personnel to existing software for which they had not carried out the development. In the 1973 to 1975 time frame, IBM reorganized the way it supported making changes to its software products and helping customers use them, largely freeing the development personnel from subsequent support activities. That model has continued to influence the group of activities that comprise ‘software maintenance‘, although many other terms have been and are being used as substitutes for ‘software maintenance‘, such as ‘operational support‘ or ‘continuous development‘. Both increased competition and the Internet are increasing the need for fast software evolution and software maintenance. As time to market and organizational responsiveness become more important, software modification activities and processes using them are being adapted to gain faster change. However, just substituting new current data into the software supporting the web site (effectively just changing values of the input data for processing) no more qualifies as either maintenance or evolution than does handling batch data arising from sales paid for by credit card tenders. Sometimes quick maintenance is required. For example, an organization active in international politics may have to modify how it presents data on its web site several times during a day. To the webmaster and the maintainer support staff, the situation often appears like continuous change in the software, with releases made on-the-fly. What the customer can access from one hour to the next may be very different, including its form and content, with still or video graphics, sound behind its competition . Our proposed types of software evolution and software maintenance are as applicable to such quick response software evolution and software maintenance situations as they are to the common traditional situations and to embedded software modification.

References:

1. Abdalla M. Elramsisi, Fareed Zaghlool, Tharwat O. S. Ahanafy and Abdou Saad El Din Moustafa, ―Neural Approach Modeling Scheme for the Software Management, New York Science Journal, Vol. 3, No. 12, pp. 142-149, 2010.

2. Akila, M., Suresh Kumar, S., Improving Feature Extraction in Software Management, Proceedings of the International Conference on Sustainable Energy and Intelligent Systems (SEISCON 2011), pp. 891–898, 2011.

3. Ali, H., Wahyudi, Salami, M., ―Essential Components of Software Management, Proceedings of 5th International Colloquium on Signal Processing & Its Applications, pp. 198–203, 2009.

4. https://yourstudent-gemini...

5. Anil Jain, Karthik Nandakumar, Arun Ross, ―Study on Software Management Pattern Recognition, Vol. 38 , pp. 2270 – 2285, 2005.

6. Aqlan. A. M, W. F. Abd El-Wahed and M.A. Abd El-Wahed, ―A Study on Maintaince processes of Software 6th International Conference on Informatics and Systems, Faculty of Computers & Information-Cairo University, Cairo-Egypt, pp. 110-117, 2008.

7. Azevedo, G.L.F., Cavalcanti, G.D.C., Carvalho Filho, E.C.B.,―Analysis of Software Management Processes.