Dynamics of Software Maintenance

Pankaj Bhatt, Gautam Shroff
Tata Consultancy Services, New Delhi, India
bhattp@acm.org, gautam.shroff@tcs.com

Arun K. Misra
National Institute of Technology, Allahabad, India
arun_kmisra@hotmail.com

Abstract

As the information technology industry gains maturity, the number of software systems having moved into maintenance is rapidly growing. Often these systems are also potential candidates for outsourcing. However, adequate information regarding size, complexity, reliability, maintainability etc. of these systems is often missing. This makes the task of estimating maintenance efforts for any such system difficult for the organization owning the systems as well as for a software services vendor bidding to outsource maintenance of the system. This is further compounded by human and management factors related to maintenance activities such as management focus, client attitude, engineers' attitude, the need for multi-location support teams etc. These factors make the problem of objectively estimating software maintenance effort almost intractable.

We submit that software maintenance is of equal if not more fundamental importance to the software industry, and has not received the attention it deserves, especially in the context of estimation models.

This paper describes a holistic approach towards a study of the factors affecting the effort involved in maintenance of existing software systems. It describes how one could build a systems dynamics model to predict the effort involved to maintain a software system, based on qualitative and qualitative inputs.

Keywords: Software Estimation, Software Maintenance

Introduction and Motivation

Software as an engineering discipline has been around for many years. Though the growth of the software industry has been significant, its maturity has not been commensurate with its growth. One of the most important abilities still lacking is the ability to estimate the size (and therefore the effort) of the software to a reasonable degree of accuracy. Though a number of techniques have been developed for effort estimation [29], most have primarily focused on software development rather than maintenance. Even though the size of the software is known by the time it reaches the stage of maintenance (also referred to as sustenance in this paper), the maintenance estimation problem is compounded by several more variable factors which make it more difficult than estimating development efforts. Invariably software maintenance is carried out by teams who had nothing to do with the design and development of the software. This is evident from the fact that a significant portion of the outsourced IT work pertains to sustenance of existing software systems. Software systems are also getting increasingly complex, both functionally and technically and this complexity further adds to complications, related to effort estimation. Further, the quality and maturity of the software being taken over for maintenance has a large bearing on the maintenance efforts that will be spent on it.

It is acknowledged that only 25% to 33% of the total effort put in during the complete life cycle of a software system goes in actually building the system [1]. The rest is consumed by effort expended towards the operational maintenance of this system.

This highlights the importance of the maintenance phase in the software life cycle. One approach could be to statistically calculate the maintenance effort based on the effort expended in the other phases of the software life cycle; however, in reality this is usually not possible due to a lack (or absence) of such data for the development phases [9]. It is also important to appreciate that software development and maintenance are also different types of activities and have different inherent characteristics [10]. Software maintenance is an evolutionary process [20] and we believe that it is of equal if not of more fundamental importance to the software industry, and has probably not received the attention and respect that it deserves, especially in the context of estimation models.

Software Maintenance

IEEE [11] defines software maintenance as:

> *The 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.*

While outsourcing of software maintenance has been growing, the field is also getting increasingly competitive due to new players entering the arena. Earlier one saw customers engaging IT services companies on time-and-material basis – where the services were paid for on the basis of number of engineers engaged in the maintenance activities. Now one can see a definite shift in this trend with customers asking for fixed-price bids for sustenance of software over a specified period. Often the services company bidding for the work has no prior insight into the software. It is therefore important to come up with a competitive estimate based on measurable factors that have a bearing on the software maintenance activity.

The software industry has been depending on estimation models for estimation for many years. These models are to quite some extent dependent on quantitative measurements that are still immature due to one or more of the following reasons:

- Incorrect implementation
- Improper usage

Therefore, despite significant research activity in the area of effort estimation, predicting effort for software project remains an elusive and challenging goal, and this is especially true in the case of maintenance projects. Collection of the data required by most of these models is a tedious and time-consuming activity. It is also dependent on human interpretation and therefore prone to errors due to a host of technical, social and political reasons [9].

Software maintenance effort estimation is an important aspect of software maintenance planning [10]. Early and accurate estimates can significantly reduces risks and assist project managers in:

- Improving maintenance process efficiency
- Maintain vs. reengineer/reacquire decisions
- Informed decision-making
- Budgeting staff allocation and rotation

Software maintenance broadly includes error corrections, changes (amendments or enhancements) and improvements to operational software. Lienz and Swanson [21], [22] categorize software maintenance as corrective, adaptive, perfective and preventive. While traditional maintenance often means restoring something to its original shape, software maintenance deals with fixing problems in original system, and bridging the gaps between the operational system and the specifications (which may mean additional efforts in the beginning of roll-out). With real end-users using the software, there is always scope for improvements in the operational system, or technical improvements from a software efficiency perspective. Changes to the operational environment (e.g. software or hardware platform) also make it necessary to make appropriate changes to operational software. We also need to distinguish between maintenance changes made to software in a development environment as opposed to those made while the software is in operation (in production environment), often referred to as 'production fixes'.

Issues and Factors Affecting Software Maintenance

Some of the key factors related to software maintenance activities are described here. These issues have been divided these into four major categories:

Programmer

These issues relate to the engineering team working on the maintenance work. Often, programmers adopt an indifferent or detached attitude to maintenance assignments, and therefore focus only on a purely technical approach to maintenance without relating it to the business needs of the customer. This is linked to the perception that maintenance jobs lack glamour [12], and are perceived as less creative by programmers who often do not see career path through maintenance work. These factors can lead to a lack of motivation leading to a disgruntled team, with possibly higher turnover [12], [16].

There are other issues related to programmer aptitude, which may be due to lack of technical experience, domain knowledge, application knowledge and skills in programming languages used. These get compounded, as often there is a steep learning curve to gain expertise in these areas.

In situations where multi-tiered support teams may be decentralized and people handling different levels of support may not be co-located, communication gap between different tiers often adds to overall effort.

Management Attitude

It is observed that management attitude plays a major role in setting the tone for the maintenance environment. Frequently it has been observed that only a limited study of the program under maintenance is normally carried out, often due to time/budget pressures. This leads to managers setting unreasonable targets for carrying out the changes. Other de-motivating factors include assigning project development assignments to programmers as rewards for good work (i.e. an 'escape' from maintenance), larger payouts to development teams, understaffing of maintenance teams and not providing training opportunities to maintenance teams. Organizational commitment in terms of making adequate programming and support tools and (in cases where the same organization has been involved with development and maintenance of a software system) person(s) involved in the development team also plays a significant role in the overall health of the maintenance project.

Code Baseline

As the code baseline is the primary work item for any maintenance project, quality of code has a direct impact on the maintenance effort and efficiency. Poor software architecture and program structure, poor documentation (either not available or almost unreadable), lack of standards and guidelines for maintenance activities etc., make studying the impacts of any change extremely difficult, if not impossible. Regression test suites and data also are often not available.

Users

It is very important to have active user involvement in the overall maintenance project. In some situations, lack of maintainability is linked to job security of the existing maintenance programmer(s). There may be user turnaround making the software system prone to repetitive misinterpretation of specifications. This may lead to having errors reported due to misunderstood specifications by the users.

Proposed Estimation Model

With this background our end goal is to:

> *Define, calibrate and validate a holistic model to predict the effort expended in maintenance activities over a specified period, for a software system in light of the impact of various dynamic factors influencing it.*

Dynamics of Maintenance

Figure 1 highlights the important fact that unlike a development project, the effort expended in maintenance of software is not a linear relation with respect to time. The normalized unit effort per day expended on maintenance of a system is dependent on many external factors as described later in this paper. Therefore, it is necessary to understand the behavior of this curve as closely as possible to estimate the total effort that may be required to be expended in maintenance of a software system over a specified time period. It is also important to note that these external factors are dynamic in nature and it is therefore necessary to bring these in a dynamic model to realistically simulate system behavior.

Application of Systems Dynamics

Systems dynamics [6], [7], [8] is a method for studying the world in a holistic manner because it looks at systems as a whole, rather than small fragments. The interaction of objects within a system, is extremely important in systems dynamics. Systems dynamics attempts to understand the basic structure of a system, and thus un-

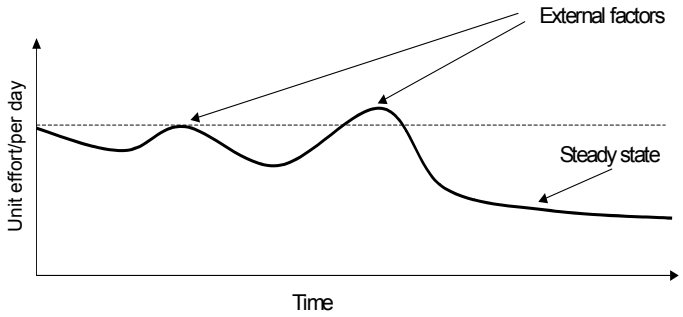

Figure 1: Dynamics of Software Maintenance

derstand the behavior it can produce. System dynamics also takes advantage of the fact that a computer model can be of much greater complexity and can carry out more simultaneous calculations than the mental model of the human mind can. Systems dynamics applies the concept of feedback control systems principles and techniques to organizational and managerial problems. The systems dynamics models tend to be much more accurate than the mental models, and are also fundamentally more powerful than a closed form mathematical formulation (such as can be expressed in a spreadsheet).

The overall research objective is to eventually produce a system dynamics model to predict the effort required to maintain a software system over a specified period. The model's prediction of effort will be based on various factors, such as, type of software, team experience, domain expertise, customer location, customer involvement etc. to name a few.

The systems dynamics approach has been chosen for the following reasons:

- It was conceptualized to simulate social and industrial phenomena that are close to the problem domain.
- It has been found to be effective in reproducing dynamic behavior of software development [2], [3], [4], [5], [13].
- There are inexpensive system dynamics modeling tools such and Vensim [17], [18], [19] available on personal computers and these can be used effectively, to build the proposed model. It has been found to be useful in determining the dynamic behavior of software engineering process.

High Level Interaction Model

A systems dynamics model applies the concept of feedback and control systems, to social and industrial phenomena, using a quantitative representation. Systems dynamics models are made up of interacting equations that produce dynamic behavior, which would otherwise be difficult to reproduce using static techniques.

As mentioned earlier, our aim is to develop a model with the goal of analyzing the effect of environmental factors, on the total effort required, for maintenance of a software system over a specified period. The model will lay special emphasis on the fine-grained aspects of software maintenance activities.

Figure 2 presents a high-level interaction model based on the authors' experience with the software maintenance process.

In this model, we view the maintenance effort being primarily influenced by two categories of parameters – capability and attitude. These parameters are described in the following text.

Capability

Capability can be viewed from two distinct angles, namely, the capability of the software under maintenance, and the capability of the team involved in the maintenance of the software. The capability of software includes metrics related to the product namely size, complexity, maintainability, technological maturity etc. The capability of the team includes the experience (domain as well as technology), team motivation, team attitude, multi-location aspects etc.

Attitude

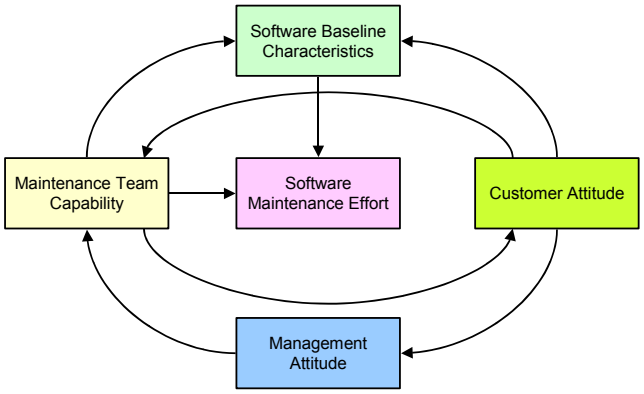

Figure 2: High Level Model

While it is often ignored in calculation of effort, it is believed that the soft factors, namely the customer attitude and management attitude towards the maintenance team, plays a vital role in motivation and consequently the productivity of these teams, which directly influences the effort and quality of work

Figure 2 shows the relationships between the factors that influence software maintenance effort. The factors relating to capability, namely, *Software Baseline Characteristics* (SBC) and *Maintenance Team Capability* (MTC) are the key contributors.

Software Baseline Characteristics

Though it is difficult to define SBC as such; one still needs to look at attributes of a software product that have a direct impact on the maintenance effort. Some of these attributes are:

Baseline Size – size of the code baseline is an important factor. This size could be measured in LoC (lines of code), number of forms, database tables, SQL statements, shell scripts, or anything else that eventually contributes to an executable in the operational system. Alternatively, one can also calculate the size in Function Points (FP) [23], [24], [25].

This can be measured directly by using the code baseline size as input.

Complexity – measurement of program complexity has been a serious topic of discussion. For simplifying this discussion, it is proposed to use a measure of algorithmic complexity, to begin with.

This can be derived by carrying out analysis of the available baseline.

Maintainability – though there is no clear way of measuring maintainability, it can be termed as the ease with which the soft-

ware can be corrected, adapted or perfected [15]. Maintainability can also be measured in terms of time oriented metric, for example Mean Time to Change (MTTC), a cost oriented metric such as *Spoilage* [14], or Software Maturity Index (SMI) as defined by IEEE Std. 982.1-981.

This can be derived by using the available past data supplemented with a questionnaire.

Maintenance history – problems found (functional and non-functional) so far in the life cycle.

This can be derived using the past maintenance data.

Documentation – quality and currency of documentation, both inline and others accompanying the software product such as architecture, high-level design, low-level design, test plans etc.

This can be derived by carrying out qualitative analysis of the available documents.

Maintenance Team Capability

MTC represents the overall capability of the team involved in the maintenance of the software system. Some of the attributes that define the capability of a team are:

Technical Expertise – understanding of the technical environment around the system under maintenance. This includes the operating system, databases and middleware products, application servers, programming language, interactive development environment and any other tool.

This can be derived by analyzing the expertise and experience of team members in related technology areas.

Domain Expertise – functional knowledge of the business domain the system under maintenance relates to. For example for a telecom-billing product, this would correspond to understanding of telecom service provider business.

This can be derived by analyzing the expertise and experience of team members in related business domain.

Application Knowledge – specific knowledge related to the software system under maintenance. An example of this would be its architecture and design.

This can be derived by analyzing the expertise and experience of team members in system under maintenance.

Programmer Attitude – often ignored this is one of the most important attributes that affects the quality and productivity of maintenance to a significant extent.

This can be captured by administration of questionnaires to the maintenance team.

Having described the *Capability* parameters, one needs to look at the *Attitude* related parameters and how they influence the software maintenance effort. Two such key factors are *Customer Attitude* (CAT) and *Management Attitude* (MAT).

CAT is an exogenous factor and primarily concerns with the customers view of software under maintenance. Some of these factors are:

Team Capability – technical and functional knowledge of the customer team interacting with the maintenance team.

This can be derived by administration of questionnaires to the customer team.

Customer Attitude

Business IT Alignment – at times the IT teams of the organization are not completely in sync with the business users. This leads to changes due to a) misinterpretation of features as problems and b) generation of change requests to meet business users requirements not catered to by the operational system.

This can be derived by administration of questionnaires to the customer IT and business teams.

Team Stability – many movements in the customer teams are detrimental to the maintenance process as they make the software system prone to repetitive misinterpretation of specifications.

Involvement – how responsive the customer team is in responding to any queries raised by the maintenance team.

Scope Creep – tendency to include additional items in scope of maintenance by leveraging the competitive market conditions.

The aforementioned factors can be captured only over time. Where the customer vendor relationship is mature, this can be substantiated by administering a questionnaire to the maintenance team.

Management Attitude

MAT is an endogenous factor and relates to attributes related to internal management view of the project team carrying out the product maintenance. Some of the key factors are:

Assignment Priority – projection of development assignments as rewards, lower bonuses to development teams, understaffing of maintenance teams, lack of training opportunities to maintenance teams.

Investment Motivation – training, tools, and environment provided to the team.

Target Definition – how realistic are the target dates set for the maintenance team. Unrealistic targets often lead to high rework leading to poor quality and effort (cost) overruns.

Developer Involvement – in cases where the system was developed by the same organization involved in the maintenance, it is important to involve people from the original development team, in the initial phases of maintenance. This helps in effecting proper knowledge transfer while enforcing the sense of ownership on the part of the development team.

These aforementioned factors can be captured by administration of questionnaires to the maintenance team. Maintenance organization's maturity on model such as CMM [26], CMMI [27] and PCMM [28] can also be used to derive these.

Further Work

This model needs to be expanded to include parameters that describe these high level constituents to a better granularity. These sub models will then need to be integrated to achieve the complete model as depicted in Figure 2.

The complete set of environmental factors to be modeled can be determined during the course of further research. Broadly, the factors can be selected keeping the following criteria in mind:

- These factors should not be organization and project specific, to ensure the usefulness of this research to the software community in general.
- There should be a possibility to collect data relating to these factors from real projects, to validate the behavior of the model.

Disclaimer

The information contained in this document represents the views of the authors and not of the company.

References

[1] Zelkowitz, Marvin V., Perspectives in Software Engineering, ACM Computing Surveys (CSUR) archive, vol. 10, issue 2, June 1978, pp197-216.

[2] Abdel-Hamid, Tarek K., Madnick, Stuart E., The dynamics of software project scheduling, Communications of the ACM archive, vol. 26 , issue 5, May 1983, pp340-346.

[3] Abdel-Hamid, Tarek K., Madnick, Stuart E., Lessons learned from modeling the dynamics of software development, Communications of the ACM archive vol. 32, issue 12, December 1989, pp1426-1438.

[4] Abdel-Hamid, Tarek, Madnick, Stuart E., Software Project Dynamics – an Integrated Approach, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[5] Abdel-Hamid, Tarek K., Adapting, Correcting, and Perfecting Software Estimates: A Maintenance Metaphor, IEEE Computer, March 1993, pp20-29.

[6] Forrester, Jay W., Industrial Dynamics", The M.I.T. Press, Cambridge, MA, 1961.

[7] Forrester Jay W., System Dynamics and Lessons of 35 Years, Sloan School of Management, Massachusetts Institute of Technology, 1991.

[8] Forrester, J. W., Systems Dynamics, Systems Thinking and Soft OR, Systems Dynamics Review, vol 10, no. 4, 1994, pp245-256.

[9] Shepperd, Martin, Cartwright, Michelle, Predicting with Sparse Data, Empirical Software Engineering Research Group, School of Design, Engineering & Computing, Bournemouth University, Talbot Campus, Poole, UK, August 2000.

[10] De Lucia, Andria, Pompella, Eugenio, Stefanucci, Silvio, Effort es-timation for corrective software maintenance, Proceedings of the 14th international conference on Software engineering and knowl-edge engineering, Ischia, Italy, 2002, pp409-416.

[11] IEEE Std. 1219: Standard for Software Maintenance, IEEE Computer Society Press, 1993.

[12] Perry, William E., Managing Systems Maintenance, Q.E.D. Infor-mation Science Inc., 1981.

[13] Rodrigues, Alexandre G, Williams, Terry, System Dynamics in Software Project Management: towards the development of a for-mal integrated framework, Research Paper No. 1996/5. Strath-clyde Business School, United Kingdom.

[14] Tajima, D., Matsubana, T., The Computer Software Industry in Japan, IEEE Computer, May 1981, pp 96.

[15] Pressman, Roger S., Software Engineering A Practitioner's Approach, McGraw-Hill International Edition, 2000.

[16] Parikh, Girish, *Handbook of Software Maintenance*, John Wiley & Sons, 1986.

[17] Vensim 5 Reference Manual, Ventana Systems, Inc., January 25, 2003.

[18] Vensim 5 Modeling Guide, Ventana Systems, Inc., January 25, 2003

[19] Vensim, Ventana Simulation Environment, User's Guide Version 5.

[20] Bennett, K. H., Rajlich, V. T., *Software Maintenance and Evolution: a Roadmap*, Proceedings of the conference on The future of Software engineering table of contents, Limerick, Ireland, 2000, pp 73-87.

[21] Lientz, B. P. and Swanson, E. B., Software Maintenance Management, Addison Wesley, Reading MA, 1980

[22] Swanson, E. Burton, The dimensions of maintenance, Proceedings of the 2nd international conference on Software engineering, San Francisco, California, United States, 1976, pp492-497.

[23] The International Function Point Users' Group (IFPUG), http://www.ifpug.org/

[24] The United Kingdom Software Metrics Association (UKSMA), *MK II Function Point Analysis Counting Practices Manual Version 1.3*, http://www.uksma.co.uk/

[25] COSMICON -The Common Software Measurement International Consortium, *COSMIC-FFP Measurement Manual, Version 2.1*, http://www.cosmicon.com/

[26] Paulk, Mark C., Curtis, Bill; Chrissis, Mary Beth, Weber, Charles, Capability Maturity Model for Software, Version 1.1, Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.

[27] CMMI Product Team, Software Engineering Institute Capability Maturity Model Integration (CMMI), Version 1.1, CMU/SEI-2002-TR-012, March 2002.

[28] Curtis Bill, Hefley, William E., Miller, Sally A., Software Engineering Institute People Capability Maturity Model (P-CMM) Version 2.0, CMU/SEI-2001-MM-01, July 2001.

[29] Boehm, Barry W., Abts, Chris, Wisnor Brown, A., Chulani, Sunita, Clark, Bradford K,, Horowitz, Ellis, Madachy, Ray, Reifer, Donald, Steece, Bert, Software Cost Estimation with COCOMO II, Prentice Hall PTR, 2000.