Software Lifetime and its Evolution Process over Generations

Tetsuo Tamai

Graduate School of Systems Management
The University of Tsukuba, Tokyo
Tokyo 112, Japan

Yohsuke Torimitsu

Systems Division
Nippon Oil Information Systems Corporation
Tokyo 105, Japan

Abstract

Software evolution process does not end at the death of an individual software system but usually continues its evolution over generations through being replaced by newly built software. To explore this research topic, we conducted a survey collecting data of software lifetimes, replacement practices and factors of replacement.

In this paper, we report the results of the survey and discuss some possible long range strategies for software life cycle planning and control based on our findings.

1 Introduction

Nobody denies every software system is mortal. However, there has scarcely been any data of the distribution of software lifetimes and causes of their deaths or literature discussing the significance of such knowledge and how to exploit it.

The distinguished work by Belady and Lehman on software evolution dynamics [3, 7] is well known and still frequently quoted. Owing to their study, it is now our common knowledge that software keeps on functionally evolving but structurally deteriorating, which eventually terminates life of software. It is usually concluded from this observation that efforts of preserving design structure and curbing system complexity growth (or entropy growth) should be given high priority in the maintenance phase. The assumption behind this principle is that the longer software lives, the better.

This might be one of the reasons that there has been little interest in collecting real data of software lifetime, although there was some argument calling attention to the importance of such statistics [5]. The survey by Lientz and Swanson [8] and similar other studies collected data of the ages of programs but they are the current ages of covered existing software systems rather than total life lengths of those systems.

We note that causes of software death is not restricted to structural deterioration. Change of hardware, operating systems and other execution environments can be a fatal cause. Considerable addition or radical change of users' requirements is clearly another critical factor. These events may induce the decision of throwing away the existing software and rebuilding or purchasing new software for replacement, be the current software system structurally deteriorated or not, which implies that longevity of software is not necessarily an essential property of good software.

Sometimes, a software system is not replaced but just abandoned and ceases to be used. Practically, such cases are not so interesting. What is more important is the case such that a software system once gets life, lives and finally dies but then revives. This long range process of software evolution over generations has not been given enough attention. We try to explore this process based on the concept of software lifetime, which is a span of software life for one generation but its role is succeeded by the next generation and so the evolutional process, as a species, continues over generations.

In the following, we report the results of the survey conducted to explore the current status of software lifetime and evolution process and discuss some possible long range strategies for software life cycle planning and control based on our findings.

2 Background

2.1 Objectives of Survey

We planned to conduct a survey on software lifetime and replacement. Our target software field is business applications, in contrast to Belady and Lehman's study and other similar works treating system software. A system software such as OS/360 studied by Belady and Lehman keeps on growing by version-up but rarely reconstructed as a whole. By contrast, it is not unusual to see an application system totally replaced by a newly constructed system after being maintained over a period of years.

In our survey, we defined a software lifetime as a period of time since the birth of a software system when it is supplied for usage until its death when it ceases to be used and is abandoned, but particularly focused on the case when the death is followed by replacement. Thus, lifetime of a software system is a property of one generation but so long as its demand of usage exists, it will repeat generations. Replacement is realized by reconstruction or by purchasing off-the-shelf software. By reconstruction, we mean an almost total re-design and re-implementation process thus a generation is a larger unit of time than a version in its ordinary usage.

The objectives of the survey were:

1. to get statistics of software life length,

2. to know the state of the practice of software reconstruction and

3. to analyze decision factors of reconstruction or almost equally determining factors of software lifetime.

Our survey approach is cross sectional over a fair number of systems rather than historical, i.e. studying one or few systems intensively over time.

2.2 Preliminary Survey

First, we conducted a preliminary survey in October 1991, taking information systems of the company one of the authors belong to as the target. The survey method was by distributing questionnaires to the personnel who are in charge of system maintenance. The systems surveyed were not exhaustive. Most of the systems were business application software written in Cobol. The average software size was about 30 KLOC (kilo lines of code).

The results can be summarized as follows.

1. There are considerable cases of software replacement. We collected 32 cases of 27 systems (which means 5 cases are repeating replacement).

2. Average software lifetime caused by replacement is about 9 years, the maximum 20 years and the minimum 2 years.

3. Multiple factors are given for the causes of replacement. Typical factors are:

   - hardware replacement or change of system architecture, e.g. change from batch to online system, change of data entry method, installation of page printers and Japanese text input/output functions.

   - change of business procedures or social systems, e.g. introduction of consumption tax system, business tie-up, and other functional enhancement requirements.

   - high maintenance cost caused by structural deterioration.

The results of this preliminary study convinced us of the significance of the study and we proceeded to a survey of wider scope.

3 Results of Survey

3.1 Overview

Our second survey was conducted in December 1991. Questionnaires were sent to information system divisions of 150 organizations in Japan, all mainframe users. 42 answers (28 %) were returned which is a fairy good response rate considering the questionnaire required a good amount of work for collecting data and filling in answers.

The main requirement of the survey was to report software replacement cases at each organization within the past five years. The reason we limited the target period to five years was that we expected to collect reliable data of uniform quality through that condition. In total, 95 valid cases were obtained.

The difference between replacement and large scale maintenance is subtle. In our questionnaire, we left the precise definition of replacement to the judgement of responders but gave a selection question whether the software reconstruction was “total” or “partial” and if the answer was partial, we let them write its percentage. 54 cases (60%) were answered as total reconstruction. In most of the “partial” cases, the percentage is no less than 50 % but there were some that gave the figure under 50%. We deleted those data under 50% out of the following lifetime statistics analysis.

Most of the respondents were system managers. The industry sectors their companies belong to are distributed as shown in Fig.1.

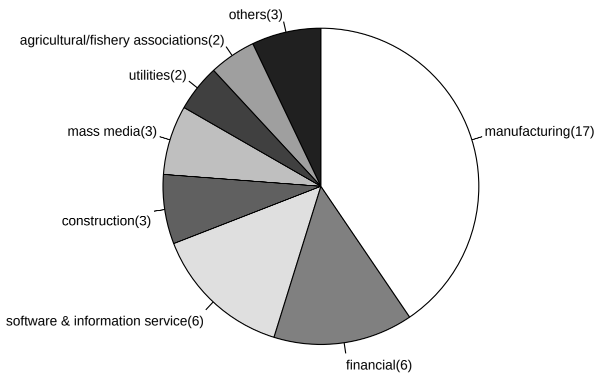

Figure 1: Industry Sectors Distribution

3.2 Findings

We summarize the findings we get from this survey under three categories: software lifetime, software replacement, and replacement causes (= lifetime determining factors).

3.2.1 Software Lifetime

1. *Software lifetime is about 10 years on the average.*

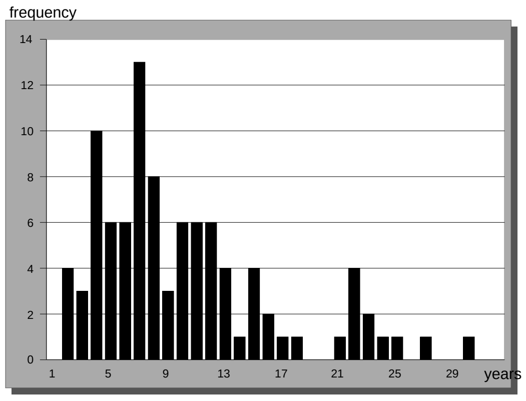

Figure 2: Software Lifetime Distribution

A simple average of software life span calculated over collected 95 samples was 10.1. This is very close to the average of the preliminary survey (8.8 years) and also matches to our common sense (we sometimes hear the range of 6 or 7 to 10 years). However, this is longer than the depreciation term of software, 5 years, stipulated by the tax law of Japan. This is in contrast to hardware as we usually see computers and other devices being replaced before its depreciation term ends.

2. *The variance of lifetime data is large.*

The distribution of software lifetime is as shown in Fig.2. The maximum is 30 years, the minimum 2 years, and the standard deviation is 6.2. The analysis of the causes for this variance is difficult but intriguing, which we shall partly try in the following.

3. *Small scale software tends to have shorter life.*

We collected two kinds of software size data, one measured by lines of code and the other by number of programs. Since the correlation of these two measures is high, we only use KLOC in the following discussion. The average size of all samples including systems both before and after replacement is about 700 KLOC but it ranges from 4 to 15,000 KLOC.

The lifetime and the size have a positive correlation but very low (the correlation coefficient between the lifetime length and the lines of code is 0.19) so that it seems better to say they are irrelevant. However, when we take a subset of samples consisting of size under 100 KLOC (the number of samples = 37), its average lifetime is 6.8, significantly shorter than the remaining samples (see Table 1).

Table 1: Software Lifetime by Size

size(KLOC)	no. of samples	average (years)	max	min	std dev.
< 100	37	6.8	16	2	3.6
< 1000	34	11.1	30	4	7.2
$1000 \leq$	8	12.3	21	8	4.2
unknown	16	14.4	27	7	5.8

4. *Administration systems live longer than business supporting systems.*

   We classified the samples by their application areas and took their statistics. As shown in Table 2, personnel systems and accounting systems live longer than sales support systems and manufacturing systems. This is not surprising because administration type applications should be relatively stable in functions while business supporting type systems must be affected by the frequent change of business environment.

Table 2: Software Lifetime by Application Area

application area	# samples	average (years)	standard deviation
personnel	17	12.1	7.0
accounting	16	12.1	8.8
sales support	39	8.8	4.2
manufacturing	10	8.6	6.9
others	13	10.5	5.5

5. *Some companies are setting software life lengths at the time of release and use them for life cycle management.*

   We asked if the responding company has a rule of predicting or planning software lifetime when a software system is released to the user. To this question, 12 companies (29 %) answered yes. Typical setting length is 5 years or 10 years. Most of them are determined as a milestone rather than a forecast so that when that time approaches, the actions of checking the system and deciding between continuous use or replacement are triggered.

3.2.2 Replacement

6. *Software size grows by replacement.*

   The ratio of software size after and before replacement is 2.66 on the average (measured by KLOC). It implies that replacement is usually accompanied by functional enhancement (but the fact is not that simple as discussed in section 4.2).

   Among the 76 data available of size change, only 5 cases show size reduction. In 4 out of those 5 reduction cases, high level description languages, 4GL and the like (Natural, SAS, and YPS), are adopted in addition to or in place of Cobol.

7. *Cobol is still dominant in business applications but diversification in programming languages is also observed.*

   Cobol is used in 70 % of the systems surveyed; its share is almost constant before and after the replacement.

   In 43 cases (45 %), the language or the set of languages used is the same before and after the replacement. The diversification in the languages is shown by the data that 13 different languages are totally used in the old systems whereas 21 languages are used in the new systems. Most of the newcomers are so called the fourth generation languages in a broad sense including database languages, program generators, and statistics manipulation systems. The number of languages of this kind increases from 3 to 12, its usage cases from 13 to 56. They are mostly used in combination with compiler languages, typically Cobol.

Decrease of usage is seen in assemblers and report generators.

3.2.3 Replacement Factors or Lifetime Determining Factors

8. *Factors that cause replacement are composite.*

In the questionnaire, we asked to list the factors that caused software replacement for each case. We did not give a candidate factor list for selection but let the responder write in freely (some examples were shown to guide the way of description).

Almost all the responses listed more than one factor for each replacement case. On the average, 2.7 factors were given.

This is conceivable, because replacement requires a large amount of investment and when compared to investment on totally new system, its priority tends to be set lower. Therefore, in order to justify the replacement decision, strong and composite reasons are required.

9. *The factor of satisfying user requirements is given as one of the causes in more than half of replacement cases.*

We classified the factors into 8 groups as shown in Table 3. The classification is, of course, not unique but the table seems to show a good whole picture. We counted 58 cases (61 %) that gave factors relating user requirements. This factor can be subclassified into 1) functional enhancement (52 cases), 2) business strategy or environment change (13), 3) cost saving and efficiency (8) and 4) organization or related system change (6) [1] .

[1] As multiple factors are given for each case, the sum of these subclass cases exceeds the total.

Table 3: Classification of Replacement Factors

No.	replacement factors	# cases
1	hardware replacement	22
2	change into distributed system	10
3	change into online system	39
4	handle Japanese text	14
5	enhance maintainability	22
6	adopt DBMS	19
7	integrate into larger system	17
8	satisfy user requirements	58

We may group together some of the factors above to make larger classes as Table 4.

Table 4: Broader Classification of Replacement Factors

replacement factors (subclass numbers)	# cases
hardware & system architecture change (1,2)	32
provide service with new technology (3,4)	42
requirements from software maintenance technology (5,6,7)	45
satisfy user requirements (8)	58

In the software engineering context, the factor of deteriorating maintainability is emphasized as a cause for software death and replacement. It surely is an important factor but not the greatest and is usually combined with other factors to lead to the replacement decision.

10. *Software replaced by the reason of maintainability has a longer life.*

The average lifetime of software replaced by the factor of maintainability is 11.2 years, significantly longer than the whole sample statistics. On the other hand, the average lifetime ended by the user requirements factor is 9.8 and that by hardware change is 9.1, considerably shorter. (The average is calculated over the samples that have been given the corresponding factor as at least one of the replacement causes, thus some cases are duplicately used for these average calculations.)

This may be explained as follows. In general, software whose functions are relatively stable over time can enjoy longer life but minor maintenance is accumulated during the life and eventually the problem of structure deterioration and low maintainability appears. New models of hardware enter market in much shorter interval compared with the average software lifetime and thus such kind of software more likely to be affected by hardware tends to be replaced in a shorter period. Similar argument should be valid for the user requirements factor.

4 Discussions

Findings from this survey can be the basis for forming long range software life cycle management strategies. We discuss some important issues for considering such strategies in this section.

4.1 Management by Lifetime Characteristics

We have seen that although the factors determining software lifetime are not simple, software properties such as application areas, size, etc. affect the length of its life. It might be possible to build a model that forecasts software lifetime taking such factors for explanation variables. A model that can be used not only for forecasting lifetime at the time of software birth but also in the middle of its life may be conceivable just as the demography model that gives one's expected remainder of life.

A mechanical statistics model may not be enough. We had better use the list of replacement factors for forecasting. For example, it is certainly a difficult task to have a reliable prospect of future technology development including computers, networks, peripheral devices, operating systems, and data bases but by means of technology forecasting methodology or the like, we may succeed to a certain degree.

Future user requirements due to business change may also be predicted partly through the use of middle/long range management plans and other materials. Deterioration of maintainability might be projected by the use of software evolution dynamics model like the one proposed by Belady and Lehman, suppose it were customized to fit to the maintenance practice of the organization and its software characteristics.

Let us assume that a lifetime of a given software system can be forecast by a proper precision within some range. Then we can exploit that knowledge in two ways.

maintenance strategy As a maintenance strategy, we can think of two extreme ones [2]:

1. maintain software so that the structure as a whole is kept clean as much as possible (call it iterative enhancement following V. Basili's terminology);
2. taking patch style, execute the maintenance work by the least possible effort (call it quick-fix also following Basili).

The iterative enhancement strategy used to be recommended as a good maintenance practice. Basili [2] insisted that maintenance strategies should be selected based on measurement but did not give definite criteria. Harrison & Cook [6] proposed a method, in which a part of program that frequently undergoes maintenance and a part seldom changed are identified by measurement and the iterative enhancement is applied to the former, the quick-fix to the latter.

We claim that analysis of lifetime gives a good clue to this maintenance strategy decision. The cost comparison between maintenance and replacement is not easy [1, 9]. However, as we have seen in the last section, software replacement can often be taken place before the software gets old by maintenance. Therefore, we have a reason to seriously consider the quick-fix method as a viable alternative. The composite strategy of Harrison & Cook may not be necessary; the quick-fix strategy can be justified as a whole if the given system must be replaced any way in a certain period of time.

A probable strategy is, for example, if the remaining lifetime of a system is forecast less than say 5 years, then the quick-fix maintenance is chosen. If a system has a large size or its targeting business area is relatively stable, then the iterative enhancement style should be considered.

milestone for life cycle management As some companies are already doing, prospected lifetime can be used as a milestone for the software life cycle management. For this purpose, the forecast need not be very precise but the concept of lifetime makes the management more concrete in deciding the timing of replacement and studying total investment plan.

4.2 Monotonicity of Size Growth

Belady & Lehman showed that software size almost monotonically grows by continuous maintenance. Our study shows that the size grows even by replacement in much larger scale than we have expected. It may be natural if the replacement is caused by functional enhancement but it turns out to be true even for the replacement caused by maintainability problems; the average size growth ratio of the former is 2.4 and the latter is 2.1 (but again, we have to note that many samples overlap between these two groups).

We think it implies that there is little serious effort of throwing away unnecessary or redundant functions. Software engineering has been emphasizing the importance of requirements acquisition process but not sufficiently taking care of “unrequirements”. Many maintenance problems may have started from this practice. We suggest that software management should take a measure of consciously reducing unnecessary functions based on the concepts of lifetime, rebirth and evolution over generations. This policy is especially important when we consider the trend of down-sizing and distributed processing.

4.3 Reuse

We have been studying maintenance problem focusing on software replacement process. A major problem of replacement strategy is its cost. Actually, several responders reported cases where replacement was considered but eventually given up and the reason in most cases was cost unjustified.

As pointed out by Basili [2], software reuse can be a key factor for reducing replacement cost. In case of replacement, we need not prepare a general purpose software component library and we can technically focus on the reuse method in a restricted target domain. The consideration of lifetime suggests that when we develop software whose prospected life is short, we should take specific care about the reusability of its components.

Additionally, we would like to note that software replacement can bring other benefits like opportunities of introducing new technologies and educating inexperienced engineers, which are difficult to evaluate in terms of cost/benefit.

Conclusion We conducted a survey of software lifetime and replacement process and found a number of interesting facts. As far as we know, this kind of data has hardly been collected but we believe effective software evolution strategies can be formed upon the knowledge obtained from this kind of data.

References

[1] Barua, A. and Mukhopadhyay, T.: A Cost Analysis of the Software Dilemma: To Maintain or to Replace, *Proc. 22nd Annual Hawaii International Conference on System Sciences Vol. III*, IEEE, 1989, pp. 89–98.

[2] Basili, V. R.: Viewing Maintenance as Reuse-Oriented Software Development, *IEEE Software*, January 1990, pp. 19–25.

[3] Belady, L. A. and Lehman, M. M.: A Model of Large Program Development, *IBM Systems Journal*, Vol. 15, No. 3 (1976), pp. 225–252.

[4] Brown, P. J.: Why does Software Die?, *Proc. 67th Infotech State of the Art Conference*, London, December 1979.

[5] Foster, J.: Program Lifetime: A Vital Statistic for Maintenance, *Proc. Conference on Software Maintenance 1991*, IEEE, 1991, pp. 98–103.

[6] Harrison, W. and Cook, C.: Insights on Improving the Maintenance Process through Software Measurement, *Proc. Conference on Software Maintenance 1990*, IEEE, 1990, pp. 37–45.

[7] Lehman, M. M.; On Understanding Laws, Evolution and Conservation in the Large Program Life-Cycle, *Proc. 67th Infotech State of the Art Conference*, London, December 1979.

[8] Lientz, B. P. and Swanson, E. B.: *Software Maintenance Management*, Addison-Wesley, 1980.

[9] Ruhl, M. K.: Findings and Recommendations from a Software Reengineering Case Study, *Proc. the Second Annual Systems Reengineering Workshop*, March 1991, Silver Spring, Maryland, pp. 101–105.