SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 1, 91–100 (1989)

Software Enhancement Modelling

C. JONES
Software Productivity Research, Inc. 1972 Massachusetts Avenue, Cambridge, MA 02170, U.S.A.

SUMMARY

Software projects often follow predictable patterns of usage, growth, and decay. Empirical observations of such patterns allow long-range estimates of software enhancements to be carried out with reasonable accuracy. It is also possible to project the impact of deliberate interventions into normal growth and decay processes, such as the intermittent restructuring of ageing software. This report summarizes long-range observations of the growth and decay patterns of software projects.

KEY WORDS Software enhancements Software estimating Software maintenance Long-range software planning Software complexity

The software industry began during World War II, but did not become a recognized profession until the 1960's. The explosive growth in the demand for software that caused programming and software engineering to become among the largest occupations groups in the United States began to accelerate during the 1970's, and has not yet shown any sign of stopping.

This growth in the demand for software led not only to the emergence of the present day computing and software industries, but to a major and unexpected problem: the long-term maintenance and enhancement of ageing software applications.

The problem should not have been a surprise, however. If the history of automobile or aircraft construction is analysed, it can readily be seen that when a major industry goes past 45 years of age, it usually takes more workers to repair and maintain the products of the industry than it takes to build new products.

Today, enterprises in the United States find themselves owners of enormous libraries of ageing software. One major high-technology corporation (Jones, 1988) surveyed their production libraries and discovered that they owned some 28,000 programs and systems which totalled 65,000,000 source code statements of existing software. This is equivalent to about 520,000 function points.

New applications were being added at the rate of about 8,000,000 source code statements or 65,000 function points per year. Old applications were being retired at a rate of about 3,000,000 source code statements or 20,000 function points per year. The net growth in the library was therefore about 5,000,000 source code statements or 45,000 function points per staff year. (Because the oldest applications written in Assembler language were being discarded at the greatest rate, and were being replaced by applications written in more concise high-level languages, the growth of function points was notably higher than the growth of source code.)

1040–550X/89/020091–10$05.00

Received 2 January 1989
Revised 10 May 1989

At the time of the study in the mid 1980's, about 3000 of the total of 7000 software professionals employed by the enterprise were engaged in maintenance and enhancement of the existing products. This is just under 43 per cent of the staff engaged in maintenance.

A corporate software planning model was developed, by the author, that could predict the overall growth of the enterprise production library and the total software staff needed to maintain the library and meet the demand for new applications and functions. The model could project forward in excess of 25 years, although as is typical with such tools, predictions of more than 10 years are normally discounted.

The planning model indicated that if the future resembled the past in terms of demand, and new applications had the same cost and quality aspects of prior applications, then by the end of the century the overall software population of the enterprise might grow to more than 20,000 and that some 14,000 or 70 per cent of them would be engaged in maintenance and enhancement tasks. This projection was dismaying, as might be expected.

However, when the parameters of the model were adjusted to reflect an overall improvement in the quality of new applications and planned intervention to improve the quality and structure of existing applications, then a more sanguine future was predicted. The best case scenario that emerged was a growth from 7000 to almost 10,000 by the end of the century, with about a 60/40 split between development and maintenance.

Large-scale studies and long-range enterprise modelling such as the above are seldom performed in the United States. When they are, the insights and competitive advantages which the studies create make them worthwhile.

More common in the United States are individual software project estimating models, of which there are about 15 that are commercially available and several hundred that are proprietary and used internally by the enterprises which created them.

Most of the software project models deal primarily with the initial development of a project and its first release. Long-range enhancements after the initial release tend to be included less often, although the accumulated costs of enhancement are often much greater than the costs of the initial release. Several commercial estimating models such as SPQR/20 and Checkpoint do include multi-year enhancement and defect removal estimates as normal outputs as well as initial project estimates.

This paper summarizes the factors that need to be included in enhancement models of software maintenance for medium to large applications. It also summarizes a few of the empirical results derived from long-range observations of ageing software projects over many years (Jones, 1985).

THE NATURAL HISTORY OF SOFTWARE APPLICATIONS

Software is often treated as an abstraction, since it is essentially invisible except to specialists who can read the source code and the underlying project specifications. However, software shares many basic similarities with more tangible products and even with natural phenomena.

- Like living organisms and most natural phenomena, software projects follow a life cycle that starts from emptiness, is followed by rapid growth during infancy, enters a long period of maturity, and then begins a cycle of decay that almost resembles senility. The ultimate end of a software project is termination, or the withdrawal from production.

- The gestation period of a software project is proportional to the size of the finished system; i.e. large systems take a very long time before they are delivered, just as large animals tend to have long gestation periods.

- The life expectancy of applications in production is roughly proportional to size. Small applications often disappear from production libraries in a year or two, while large applications tend to exist for 10 to 15 years or even more. This also resembles natural phenomena, where the life expectancy of animals is proportional to their mass.

- Once an application initially enters production, it continues to add new functions and features at a relatively constant rate that ranges between 5 per cent and 15 per cent new functionality per year. This phenomenon appears similar to the gain in body mass of a number of living organisms during youth and adolescence.

- The entropy of software systems continues to increase over time. This is equivalent to saying that many small changes gradually degrade the initial structure of a software project and increase its complexity. This phenomenon is true of all known physical systems and all natural organic entities, so its discovery for software is not surprising.

- Preventive and corrective intervention can delay the onset of entropy for software systems and stretch out the useful period before replacement becomes mandatory. This is more or less equivalent to observations of natural organisms where preventive and corrective intervention can delay the onset or severity of some forms of illness.

The relationship between software and natural phenomena should not be pushed too far, but the basic point is that software life cycles can be modelled as well as the life cycles of any other physical or natural system.

DEFINING THE PARAMETERS OF SOFTWARE ENHANCEMENT MODELS

The first step in long-range modelling of software is to define the scope of the model, and the meanings of the parameters to be used.

Models can range in scope from detailed individual project models to large-scale corporate models. As a rule of thumb, the larger corporate models requiring many more parameters are more sensitive to fluctuation than the individual software project models. This paper deals with individual project software modelling and enhancements only.

Establishing existing conditions for enhancement models

All models must be initialized with a set of existing conditions. For software enhancement modelling, the initial conditions to be established include:

(i) The current size of the application being modelled in terms of both source code size by language and net size in terms of function points. Medium to large

applications will range upward from 500 function points and 50,000 source code statements in aggregate size. This information can be measured directly, either by manual counting or using a variety of commercial tools.

(ii) Knowledge of the structure and stability of the application being modelled, such as the average value of McCabe's cyclomatic complexity number (McCabe, 1976) for the modules within the application. For average to poorly structured software projects, this will probably be a cyclomatic number in excess of 30. This information can be directly measured using a variety of commercial tools.

(iii) A census count of the software staff that built the current version of the application, and empirical data on the probable staffing for the initial year of enhancement work. This information is usually derived by analogy to similar applications already in maintenance status within the enterprise, or by direct count of the staff on current application.

(iv) The calendar period over which time the model is to operate. This means that knowledge of the start date of the project, the date of initial production, and the number of years over which the model is to operate must be specified. Such data are normally available from historical records, or if that is incomplete, from the memory of actual project staff members.

Basic parameters for long-range enhancement modelling

When the initial conditions have been defined, it is then necessary to consider the parameters that will impact on the software over the desired modelling period. Basic parameters include the following:

(i) Historical data on the annual growth of enterprise software projects and the removal rate of dead or unused code. Normally software projects will continue to add new functions at a rate that approximates to 10 per cent per year, and to remove dead or unused code at a rate of perhaps 2.5 per cent per year, leading to a new growth of about 7.5 per cent per year. This is one of the critical factors of enhancement models, and can usually be derived from enterprise project history data.

(ii) The average burdened salary rate of the software staff, from executive to entry levels. There are wide company, industry, and geographic variations, but about $5000 per staff month would be a normal burdened rate.

(iii) Annual inflation rate, since inflation must be factored into long range models. Typical inflation planning figures used in the United States would be about 10 per cent increase per year in terms of overall cost of money.

(v) Annual entropy, or the increase in project complexity over time as the result of numerous small changes. This is a critical parameter that can be derived from

empirical observations of ageing software. Entropy, which can be approximated by the annual increase in the average for the cyclomatic complexity of the application, will range from 1 per cent to more than 10 per cent per year. This information is usually derived from periodic complexity measurements using some of the commercially available tools for that purpose.

Special parameters for long-range enhancement modelling

The basic enhancement estimating parameters are for the most part intuitively obvious and logically necessary for long-range modelling. However, there are a number of special parameters that are not intuitively obvious, but which have been observed empirically on real-life projects that have been studied over extended periods. These special parameters exert such strong influences on project outcomes that they must be dealt with.

(i) Interventions during the modelling period, such as the restructuring of the software using one of the commercial software restructuring tools. The observed effect of restructuring is to lower the entropy or complexity of projects down to a cyclomatic number of 10 or less, but to increase the size of the application in terms of source code volume by about 10 per cent. Functionality and function point totals are usually not affected. Since commercial restructuring tools are available only for COBOL, this special parameter is normally applicable only to COBOL projects.

(ii) Optimal enhancement sizes in terms of productivity rates. Because any update to an existing application will require some recompilation, regression testing, and perhaps updating of existing documents, enhancement productivity is not intuitively related to size. Specifically, very small enhancements often have very low productivity, since the costs of recompiling and regression testing the current application tend to add significant overhead costs (Jones, 1986). Empirical data suggest that enhancement productivity tends to peak when the size of the update is about 3 per cent of the size of the original application. Both small and large updates will typically have lower productivity rates. This factor is true for all languages and both for systems and MIS software projects.

(iii) Complexity and productivity rates. Although it is intuitively obvious that productivity and complexity should have an inverse relationship, the exact amount of productivity reduction can only be derived from empirical observations. For purposes of basic modelling, it can be assumed that productivity will go down at the same rate that complexity or entropy goes up. Two special facts have been observed dealing with software project complexity: projects where the average value of cyclomatic complexity is higher than 50 tend to be untestable; and projects where the average value of cyclomatic complexity is higher than 75 tend to be unmaintainable, since each update can trigger fresh failures.

(iv) Relationship between initial complexity and growth of entropy over time. One of the most subtle parameters for enhancement modelling deals with the relationship between the initial complexity of a project, and the subsequent growth of entropy

over time. Well structured projects with cyclomatic numbers that average 10 or less tend to keep their structures fairly well and do not add complexity at a very steep rate. Poorly structured projects, on the other hand, such as those with cyclomatic complexities over 25 when first released tend to quickly destabilize under the impact of normal maintenance and their entropy rates quickly climb to dangerous levels.

(v) Limitations on the model's outputs, such as a constraint not to go beyond, for example, $500,000 per year in total software expenses or a constraint to limit overall staff size to 10 employees. Limits serve as boundary conditions for models. In real life, such constraints are likely to reflect the demands of enterprise executives to limit staff growth or expenses.

EXAMPLES OF LONG-RANGE SOFTWARE ENHANCEMENT MODELLING

To illustrate the concepts of long-range enhancement modelling, the following are some outputs from a small but representative enhancement model which deals only with the major parameters. The project being modelled will be a hypothetical 250 function point application, written in COBOL and requiring some 25,000 source statements in the initial release.

The enhancement estimating model will be set for 10 years of production, and will be run twice. Once will be to consider the implications of normal growth in entropy or complexity, and once will be to consider the implications of periodic restructuring of the application.

The model matches closely the actual experiences of several projects observed by the author for extended periods. One sequence spanned the years 1974–1979 for three projects and a second spanned 1979–1983 for two projects.

The literature on long-range evolution of software is sparse, and only the pioneering work of Belady and Lehman (1976) cover multi-year histories of actual projects.

Initial conditions of the project being modelled

The project used for these examples is a COBOL-based information system of 250 function points and 25,000 source code statements in aggregate size. The first release of the project was December 1988, and the first full year of maintenance commenced in January 1989.

The average measured complexity of the project modules at the start of production was a cyclomatic number of 30, which reflects mediocre or fairly poor structure. With this starting condition, the growth of entropy or increased project complexity will be fairly high: more than 7 per cent per year.

The project was initially developed over a 16 month period by staff that ranged from 3 to 4 persons, with an average value of 3.75 full time staff. The average burdened salary rate is assumed to be $5000 per staff month. The total development effort totalled 57 person months and the total development costs were $285,000.

The long-range inflation rate is assumed to be 10 per cent, which means that average annual salaries will be increased by that amount.

Ten-year estimates for the project without intervention

Table 1 shows the predicted growth of the project as a result of 10 years of normal new function addition and dead code deletion, assuming 10 per cent new code and 2.5 per cent dead code removal per year. Table 1 assumes only normal maintenance and enhancement work, and does not model the results of interventions such as program restructuring.

At the end of the 10 year modelling period the overall size of the application has grown from 25,000 source code statements to 51,526 source code statements. Empirical observations, by the author, have shown that major projects grow at even faster rates.

The most alarming results from Table 1 are the cautionary statements that the project will become untestable by 1996 and remain so thereafter. This is clearly a condition that will require intervention to head off a potentially serious set of real-life problems should the project actually follow the model's projections.

Table 2 shows the average annual staffing, effort, and costs for enhancement work over the same period. The staffing assumptions include direct technical staff such as programmers, management, and support staff such as technical writers aggregated together. (Note: Aggregating all staff is a deliberate simplification for publication purposes. A full enhancement model can show staff and occupation groups separately.)

Since the initial release of this project cost $285,000 and the cumulative maintenance costs are $808,188 it can be seen that the enhancement model matches the empirical observations that much of the expense of software projects occurs after the initial delivery, assuming of course, reasonable assumptions for the life history of the project and for annual inflation rates.

Table 1. 10-year growth in project size due to enhancements

Year	Code added	Code deleted	Current size	Average complexity	
1988	—	—	25,000	30	
1989	2,500	625	26,875	32	
1990	2,688	672	28,891	35	
1991	2,889	722	31,057	37	
1992	3,106	776	33,387	40	
1993	3,339	835	35,891	43	
1994	3,589	897	38,583	46	
1995	3,858	965	41,476	50	
1996	4,148	1,037	44,587	54	UNTESTABLE
1997	4,459	1,115	47,931	58	UNTESTABLE
1998	4,479	1,198	51,526	62	UNTESTABLE
Total	35,368	8,842			

Table 2. 10-year enhancement staffing, effort and costs

Year	Enhancement staff	Enhancement effort	Enhancement cost	Average complexity	
1988	—	—	—	30	
1989	1.1	5.2	$28,342	32	
1990	1.2	5.7	$34,195	35	
1991	1.3	6.3	$41,673	37	
1992	1.5	6.9	$50,787	40	
1993	1.7	7.7	$61,893	43	
1994	1.8	8.5	$75,429	46	
1995	2.0	9.4	$91,925	50	
1996	2.3	10.6	$113,161	54	UNTESTABLE
1997	2.5	11.8	$139,302	58	UNTESTABLE
1998	2.8	13.2	$171,481	62	UNTESTABLE
Total		85.2	$808,188		

Ten-year estimates for the project with periodic intervention

Because Tables 1 and 2 showed that the years 1996, 1997, and 1998 had such high entropy or complexity that the project would become untestable, it is clear that some form of intervention would be desirable before the project gets out of control.

Since the project is written in COBOL, it would be possible to use one of the commercial restructuring tools to reduce the complexity of the application. In 1989 the technology of restructuring is now more than five years old, and several thousand projects have been restructured.

Empirical observations by the author of the results of COBOL restructuring indicate two common results:

(1) The average cyclomatic complexity of the project is indeed reduced to safe levels.

(2) The restructuring process often tends to increase the size of the application by up to 10 per cent.

Tables 3 and 4 examine the same project shown in Tables 1 and 2, but assume that the project will be restructured at four-year intervals in order to prevent the serious consequences of project complexity getting out of hand.

A comparison of Table 1 with Table 3 illustrates some of the salient features of both enhancement modelling and periodic intervention. Restructuring an application significantly lowers the overall cyclomatic complexity, but the restructuring process tends to increase gross application size significantly: from 51,526 source code statements in Table 1 to 62,346 source code statements as shown in Table 3, i.e. 10,820 more statements.

Table 4 shows the staff, effort, and costs associated with the 10 year enhancement cycle of the intermittently restructured scenario. Note that the increase in application size due to restructuring does not imply either higher effort or greater costs. Indeed, in spite of

Table 3. 10-year growth in project size with periodic restructuring

Year	Code added	Code deleted	Current size	Average complexity	
1988	—	—	25,000	30	
1989	2,500	625	26,875	32	
1990	2,688	672	28,891	35	
1991	2,889	722	31,057	37	
1992	3,416	854	36,725	10	RESTRUCTURED
1993	3,673	918	39,480	11	
1994	3,948	987	42,441	12	
1995	4,244	1,061	45,624	12	
1996	5,019	1,255	53,950	10	RESTRUCTURED
1997	5,395	1,349	57,996	11	
1998	5,800	1,450	62,346	12	
Total	38,804	9,611			

Table 4. 10-year enhancement staffing, effort and costs associated with intervention and periodic restructuring

Year	Enhancement staff	Enhancement effort	Enhancement cost	Average complexity	
1988	—	—	—	30	
1989	1.1	5.2	$28,342	32	
1990	1.2	5.7	$34,195	35	
1991	1.3	6.3	$41,673	37	
1992	1.1	5.0	$36,897	10	RESTRUCTURED
1993	1.3	6.0	$48,473	11	
1994	1.4	6.5	$57,320	12	
1995	1.5	7.0	$67,781	12	
1996	1.6	7.4	$79,357	10	RESTRUCTURED
1997	1.9	8.9	$105,298	11	
1998	2.1	9.6	$124,515	12	
Total		67.5	$623,850		

the increased application size, the enhancement effort is significantly reduced, which matches empirical observations closely.

Assuming reasonably accurate assertions about the impact of periodic intervention and restructuring, it would seem that keeping software project complexity and entropy low pays off in reduced effort and expenses. Empirically in real-life observations, this is indeed what occurs.

Although outside the scope of the current report, enhancement modelling can lead to very interesting questions about the desirability of restructuring new applications before they enter production initially, and about restructuring applications on an annual basis

instead of at four-year intervals. For example, the model illustrated here indicates that pre-deployment restructuring before putting an application into production would be even more effective than delaying the first intervention for four years. Once a model is constructed, many possible scenarios can be explored.

SUMMARY AND CONCLUSIONS

Long-range enhancement modelling at both project and corporate level offers definite benefits to enterprise maintenance planners and managers. Comparatively simple project models with reasonable initial assumptions, a small set of basic estimating parameters, and some knowledge on the part of the planners about special estimating situations can lead to useful results and to the ability to evaluate alternative strategies.

More sophisticated corporate models can lead to early warnings of possible problems, and can alert corporate management to the need for early intervention before hazardous situations escalate. At best, long-range enhancement modelling can help avoid serious maintenance problems. At worst, the models provide interesting insights into the maintenance process and generate questions and topics for future research.

References

Belady, L. and Lehman, M. M. (1976) 'A model of large program development', *IBM Systems Journal,* **15** (3), 225–252.

Jones, C. (1985) *Maintenance and Enhancement of Aging Software*, Software Productivity Research, Inc., Cambridge, Massachusetts, 65 pp.

Jones, C. (1986) *Programming Productivity*, McGraw-Hill, New York, 280 pp.

Jones, C. (1988) *A 10 Year History of Software Engineering in the ITT Corporation*, Software Productivity Research, Inc., Cambridge, Massachusetts, 25 pp.

McCabe, T. (1976) 'A complexity measure', *IEEE Transactions on Software Engineering,* **SE-2**, 308–320.